First publ. in: E-systems and e-man for cybernetics in cyberspace (Vol. 1) : 2001 IEEE International Conference on Systems, Man, and Cybernetics, Tucson, Arizona, October 7 - 10, 2001. Piscataway, NJ : IEEE Service Center, 2001, pp. 123-128

FINDING CLUSTERS IN PARALLEL UNIVERSES

DAVID PATTERSON and MICHAEL R. BERTHOLD

Tripos Inc., Data Analysis Institute
601 Gateway Blvd., Suite 720, South San Francisco, CA 94080, USA
eMail: {pat,berthold}@tripos.com

Abstract

Many clustering algorithms have been proposed in recent years. Most methods operate in an iterative manner and aim to optimize a specific energy function. We present an algorithm that directly finds a set of cluster centers based on an analysis of the distribution of patterns in the local neighborhood of each potential cluster center through the use of so-called *Neighborgrams*. In addition this analysis can be carried out in several feature spaces in parallel, effectively finding the optimal set of features for each cluster independently. We demonstrate how the algorithm works on an artificial data set and show its usefulness using a well-known benchmark data set.

Keywords

Clustering, multiple feature spaces, neighborgrams.

1 Introduction

Clustering has been one of the most used methods for the analysis of large data sets over the past decade [1]. Most algorithms attempt to find an optimal set of cluster centers by adjusting their position and size iteratively, through subsequent small adjustments of the parameters. Many variants of such algorithms exist, the most prominent example is probably Kohonen's Linear Vector Quantization [2]. Many variants using imprecise notions of cluster membership have also been proposed [3].

Adjusting the cluster parameters iteratively (usually by means of a gradient descent procedure) makes much sense in the case of vast amounts of data for positive and negative examples. Approaches that try to find cluster centers more directly have also been proposed; many examples can be found in algorithms that train local basis function networks, such as a constructive training algorithm for Probabilistic Neural Networks [4]. This algorithm iterates over the training instances and whenever it needs a new cluster to model a newly encountered pattern, a new cluster center is introduced at its position. Such an approach depends on the order of training examples and is therefore not guaranteed to find an optimal set of cluster centers either.

However, if the data set is highly skewed and the focus of the analysis aims to extract a model that explains the minority class with respect to all other examples, a more direct approach can be taken. Such data is available in many applications, a promient example is drug discovery research where the focus lies on the identification of few promising active compounds in a vast collection of available chemical structures that mostly show no activity or are otherwise regarded as useless.

In this paper we describe an algorithm that finds an optimal set of cluster centers directly by computing so-called *Neighborgrams* for each positive example. These neighborgrams summarize the distribution of positive and negative examples in the vicinity of each positive example. We can then extract the set of best cluster centers from the set of neighborgrams, according to some optimality criterion. The algorithm relies on the fact that the class of interest (the positive examples) has a reasonably small size (usually up to several thousand examples), whereas the opposing examples can be as numerous as desired.

In addition we describe an extension of this algorithm to find clusters in parallel universes. This becomes enormously useful in cases where different ways to describe entities exist. In drug discovery, for example, various ways to describe molecules are used and it is often unclear, which of these descriptors are optimal for any given task. It is therefore desirable if the clustering algorithm does not require that a certain descriptor is chosen a-priori. Clustering in parallel universes solves this problem by finding cluster centers in these different descriptors spaces, in effect parallelizing feature (space) selection with clustering.

2 Neighborgrams

The algorithm to find clusters directly is based on a measure that ranks so-called *Neighborgrams* according to a certain quality measure. Each Neighborgram summarizes the neighborhood of one positive example through a histogram of positive vs. negative examples in the neighborhood of that pattern.

A neighborgram summarizes the distribution of two[1] classes in a certain neighborhood of an arbitrary point in the feature space. In order to do this we need a (in this case normalized[2]) distance function $d : D \times D \longrightarrow [0, 1]$ as well as a pre-defined binning: $[b_i^-, b_i^+]$, $1 \leq i \leq B$ where B indicates the number of bins and $0 \leq b_i^- < b_i^+ \leq 1$ and usually (although not necessarily) $b_{i-1}^+ = b_i^-$. Note that the binning does not need to cover the entire range of the distance function, in most cases it is even desirable to concentrate only on a small local neighborhood (i.e. $b_B^+ < 1$). In addition we assume that a set of positive and negative examples T^p resp. T^n exist.

A Neighborgram for a certain pattern $\vec{x}$ is then defined as a pair of counts for each bin:

$$\mathrm{NG}_i^p(\vec{x}) = \left|\left\{\vec{x^p} \in T^p \,:\, b_i^- \leq d(\vec{x}, \vec{x^p}) < b_i^+\right\}\right|$$

for the positive examples $\vec{x^p}$, and

$$\mathrm{NG}_i^n(\vec{x}) = \left|\left\{\vec{x^n} \in T^n \,:\, b_i^- \leq d(\vec{x}, \vec{x^p}) < b_i^+\right\}\right|$$

for the negative examples $\vec{x^n}$.

Figure 1 shows an example of a neighborgram. The bottom part shows a two-dimensional feature space with positive (light circles) and negative examples (dark circles). The dashed circles indicate the binning of the distance function surrounding a certain pattern, chosen as centroid of a neighborgram. The top part of the figure shows the resulting neighborgram for this pattern in the center of its cluster. Note how the number of positive examples declines drastically with increasing distance while the negative examples behave complementary. Visually such a behavior could be used to indicate a "good" cluster center.

Figure 2 on the other hand shows a neighborgram for a positive pattern at the outside of the cluster. Again, the dashed circles indicate the binning of the corresponding distance function. The distribution of positive and negative example in the corresponding neighborgram indicates that the true center lies probably more in the region of the third bin in this particular (artificial!) case.

[1] This concept can easily be extended to more than two classes, in this paper we constrain ourselves to the case of two classes for sake of simplicity.

[2] If the distance function is non-normalized the binning needs to be adjusted accordingly.

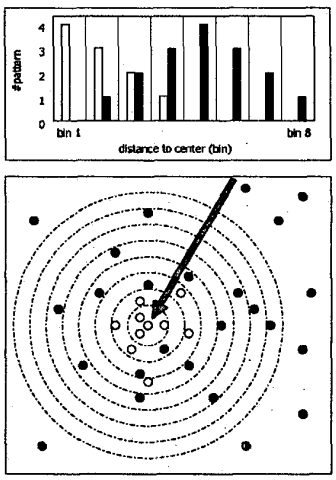

Figure 1: Example *Neighborgram* for a "good" centroid (see text).

Neighborgrams depend on a series of parameters. Not only can we adjust the number of bins and the range of the distance function that we want to cover but in addition the bins can be distributed non-linearly. In our current implementation we offer linear, quadratic, and logarithmic distribution of an arbitrary number of bins. The underlying distance function also has an impact on the resulting distributions; for binary features we use tanimoto or hamming distances, for numeric features the Euclidean distance.

3 Finding Cluster Centers

As mentioned in the previous section, just looking at some neighborgrams suggests "good" and "bad" cluster centers. If were able to formalize such a measure and use that to rank neighborgrams we could use the clustering algorithm shown in table 1.

3.1 Ranking Neighborgrams

In order to choose the "best" neighborgram, a ranking procedure is needed. The most obvious choice would be to simply count the number of positive examples that can be found in a circle around the center without encountering any negative example. The radius of this circle is:

$$\mathrm{radius}_{\mathrm{crisp}}^{\mathrm{best}}(\mathrm{NG}(\vec{x})) =$$

$$\max\left\{j \,:\, 1 \leq j \leq B \wedge \sum_{i=1}^{j} \mathrm{NG}_i^n(\vec{x}) = 0\right\}$$

Table 1: The algorithm to generate a set of cluster centers based on a ranking of Neighborgrams for each positive example.

```
1) Generate one Neighborgram for each positive example
2) WHILE (enough positive examples left) DO
  3) find "best" Neighborgram and determine optimal cluster size
  4) remove all positive examples that are covered by the resulting cluster
  5) re-compute Neighborgrams for remaining positive examples
6) END WHILE
```

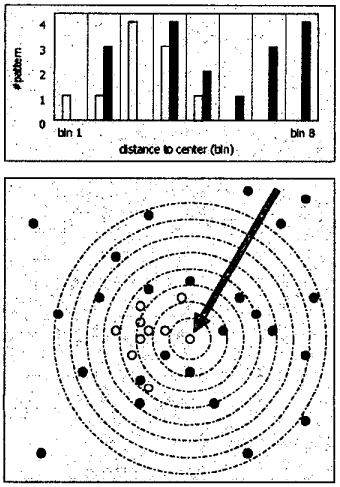

Figure 2: Example *Neighborgram* for a "bad" centroid (see text).

and the ranking coefficient would then simply be the number of positive example within this radius:

$$\text{score}^{\text{best}}_{\text{crisp}}(\text{NG}(\vec{x})) = \sum_{i=1}^{\text{radius}^{\text{best}}_{\text{crisp}}} \text{NG}^p_i(\vec{x})$$

Obviously this ranking mechanism is very strict and already fails to assign reasonable scores to the two example neighborgrams shown in figures 1 and 2. We therefore relaxed the condition for the negative examples and use a measure that tries to find a minimal ratio θ of positive to negative examples instead.

$$\begin{aligned}&\text{radius}^{\text{best}}_{\text{soft}}(\text{NG}(\vec{x}),\theta) = \\ &\max\Big\{j : \quad 1 \le j \le B \\ &\qquad \wedge\, \forall k : 1 \le k \le j : \\ &\qquad\quad \text{A}(\text{NG}^p_i(\vec{x}),k) \ge \theta\Big\}\end{aligned}$$

where

$$\text{A}(\text{NG}^p_i(\vec{x}),j) = \frac{\sum_{i=1}^{j} \text{NG}^p_i(\vec{x})}{\sum_{i=1}^{j} (NG^p_i(\vec{x}) + \text{NG}^n_i(\vec{x}))}$$

computes the normalized accumulated positive patterns until bin j. The score for this measure computes as:

$$\text{score}^{\text{best}}_{\text{soft}}(\text{NG}(\vec{x}),\theta) = \sum_{i=1}^{\text{radius}^{\text{best}}_{\text{soft}}} \text{NG}^p_i(\vec{x},\theta)$$

Obviously more sophisticated measures could be used - a weighting mechanism for the distance to the centroid might make sense for some applications, or one could estimate the 95% confidence interval and find a corresponding upper (or lower) bound. In some applications it might also be more desirable to focus on large clusters instead, which would put more weight on the radius of the cluster rather than the amount of positive examples it covers.

However, the clustering algorithm in table 1 using the above way to find the best neighborgram at each step is already quite successful in finding a good set of representative cluster centers for a given data set. In the following section we will show how this method can be used to find clusters in parallel universes, an extension that is of growing interest in life science applications.

4 Finding Cluster Centers in Parallel Universes

In many applications samples can have different representations. These can arise because in one feature space different similarity or distance metrics are used or because different ways to describe the same element exist, actually resulting in different feature spaces altogether.

In effect we now assume that we have a set of universes $u_0, \cdots, u_m$ (m being the number of different

ways to describe patterns) and for each training example, several descriptions exist. That is, each example vector $\vec{x}$ in T^p resp. T^n can be seen as a concatention of vectors from several universes:

$$\vec{x}^{\{p|n\}} = \left(\vec{x}^{\{p|n\},1}, \cdots, \vec{x}^{\{p|n\},m}\right)$$

and

$$\vec{x}^{\{p|n\},l} \in u_l \; : \; 1 \leq l \leq m$$

The algorithm shown in table 1 can easily be extended to handle different representations of the same training instance. In fact, it is more straightforward to do so using the neighborgram-ranking based algorithm than for an iterative version. We need to only compute the neighborgrams for each positive instance in each universe:

$$\mathrm{NG}_i^{p,l}(\vec{x}) = \left|\left\{\vec{x^{p,l}} \in T^p \; : \; b_i^- \leq d(\vec{x}, \vec{x^{p,l}}) < b_i^+\right\}\right|$$

Afterwards the neighborgrams are ranked, irrespective of which universe they belong to, and the best one is chosen. We then remove all patterns covered by the resulting cluster in the corresponding universe and in addition remove those patterns also from all other universes. Effectively this changes only lines (3) and (4) of the algorithm shown in table 1:

```
...
 3') find "best" Neighborgram in any
     universe and determine optimal
     cluster size in this universe
 4') remove all positive examples in
     all universes that are covered by
     the resulting cluster
...
```

Note that it is sufficient for each pattern to be covered by one cluster in one universe. In effect, the algorithm finds the best set of clusters spread out over several universes.

5 Results

5.1 Artificial Data

In order to demonstrate how the algorithm finds clusters in the parallel universes, we have generated an artificial data set. About one thousand six-dimensional data points were generated, one third of which was labeled positive. The remainder was used as negative examples. The six dimensions were divided into three universes, each consisting of two dimensions (universe $u_0 = (x_0, x_1)$, $u_1 = (x_2, x_3)$, $u_2 = (x_4, x_5)$). The positive examples exhibit three clusters:

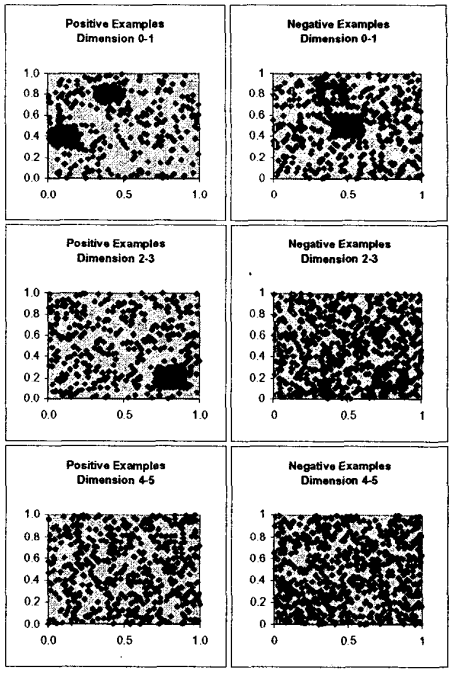

Figure 3: The artificial data set - split into 3 universes. Each universe contains two of the six dimensions and the above plots show the distribution of the positive resp. negative examples.

- a radial distribution with mean $x_0 = 0.1$, $x_1 = 0.4$, and $x_2, \cdots, x_5$ exhibiting random noise. Essentially this forms a cluster that is only expressed in universe u_0.
- a radial distribution with mean $x_0 = 0.4$, $x_1 = 0.8$, and $x_2, \cdots, x_5$ exhibiting random noise. Again, this cluster is only expressed in universe u_0.
- a radial distribution with mean $x_2 = 0.8$, $x_3 = 0.2$, and x_0, x_1, x_4, x_5 exhibiting random noise. This cluster is only expressed in universe u_1.

The negative examples have only one cluster at $x_0 = 0.5$, $x_1 = 0.5$, expressed in universe u_0. Figure 3 shows the distributions of positive and negative examples in the three universes. All clusters have a radius of 0.1 and both classes are also uniformly spread out over the entire feature spaces, although with lower density, as can be seen in the plots.

Running the cluster algorithm will generate neighborgrams for all positive examples in all three universes. The algorithm will then pick the neighborgram

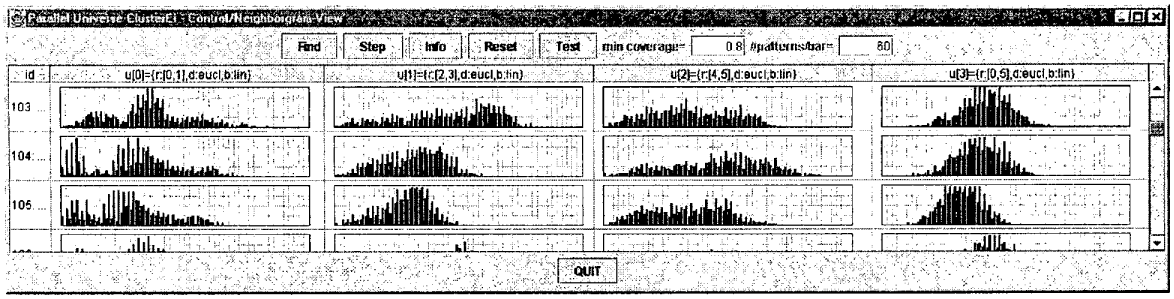

Figure 4: The first pick of the clustering algorithm is the Neighborgram for pattern $\vec{x}_{104}$ in universe 0. The cluster of positive examples is clearly visible. Note how the neighborgram for pattern $\vec{x}_{103}$ in universe 0 nicely displays the two positive and one negative cluster.

for $\vec{x}_{i=104,u=0}$ first, i.e. pattern 104 in the first universe. Figure 4 shows the neighborgrams for all 3 universes as well as the neighborgram for the entire 6-dimensional space. We used $\theta = 80\%$. This particular pattern has coordinates

1. $(x_0 = 0.105, x_1 = 0.393)$

and lies close to the center of the first cluster. The best radius is computed as $r = 0.08$, again close to the real value (considering that 50 bins are used to model the distance $[0, 1]$, this estimate is off by only one bin).

After removing the examples covered by this pattern (a total of 170), the next cluster centers are:

2. $(x_2 = 0.814, x_3 = 0.194)$, which removed 104 patterns, and
3. $(x_0 = 0.357, x_1 = 0.803)$, removing another 43 patterns.

These first 3 clusters describe the underlying cluster structure nicely and subsequent clusters found by the algorithm only model artifacts of the data, as can also be seen by the drastically smaller number of patterns that are removed (12, 8, 6, ...).

This example shows how the algorithm finds good candidates for cluster centers close to the optimal location at each step. Clusters that are only expressed in a part of the feature space (a universe) are extracted and then used to filter out the corresponding positive examples.

5.2 DNA Data

In order to demonstrate the usefulness of this algorithm, we have used the DNA dataset from the StatLog project [5]. This data set contains 3186 examples and the task is to predict boundaries between exons and introns. A window of 60 nucleotides is presented as a 180-dimensional binary vector. Groups of three bits represent an amino acid and three classes are used to describe the middle point of the window as intron-extron boundary, extron-intron boundary, or non of both. 2000 randomly chosen examples are used for training and the remaining 1186 for testing.

From [5] is known that the middle portion of the window carries substantially more information than the borders. It would therefore be interesting to see if our parallel universe clustering algorithm can pick out more clusters from that portion of the feature space. We have divided the 180 binary features into three universes, each consisting of 60 bits and ran the algorithm described above on the resulting training patterns.

Due to space constraints we can not list the results with much detail but a couple of observations are worth being noted:

- the influence of the threshold θ was less critical than expected. Obviously with higher θ more and smaller clusters will be generated but the effect on generalization performance is small.
- Since our current version only builds a model for one class, we build three one-class classifiers. The algorithm created clusters mostly in the second universe, representing the features $60 - 120$. Specifically, for class 0, 21 clusters were build in universe 2, only one in universe 1 and none in universe 3 (class 1: $(0, 18, 0)$, class 2: $(3, 38, 2)$).
- Performance of those three independent classifiers is not straightforward comparable to results of a three-class classifier. For class 0 we achieved an error rate of 22.68%, class 1: 9.78%, and class 2: 9.61%, which - when averaged - is roughly comparable to the results reported in [5].

6 Related Work

The algorithm initially was derived from work on constructive training algorithms for probabilistic neural networks [4] and has strong similarities to Neighborhood Plots [6], although neighborgrams model local behavior with respect to one positive example. Neighborhood Plots visualize the averaged neighborhood behavior of the entire data set. The neighborgram clustering algorithm shares properties with Mountain Clustering [7] in that both algorithms remove patterns after a covering cluster was found. However, the local estimation of a density through an individual neighborgram reduces the computational complexity tremendously.

7 Conclusions

We have presented a new approach to clustering, which finds good cluster centers for local neighborhoods based on a model of the neighborhood of each positive example. In addition the algorithm can easily be extended to handling several feature spaces in parallel, which offers very interesting potential in life science applications such as drug discovery. Future work will focus on better ranking metrics for neighborgrams and integration of the neighborgram visualization technique in visual data exploration environments.

References

[1] Michael Berthold and David J. Hand, Eds., *Intelligent Data Analysis: An Introduction*, Springer Verlag, 1999.

[2] T. Kohonen, "The self-organizing map", *Proceedings of the IEEE*, vol. 78, pp. 1464, 1990.

[3] Rajesh Davé and Raghu Krishnapuram, "Robust clustering methods: A unified view", *IEEE Transactions on Fuzzy Systems*, vol. 5, no. 2, pp. 270–293, May 1997.

[4] Michael R. Berthold and Jay Diamond, "Constructive training of probabilistic neural networks", *Neurocomputing*, vol. 19, pp. 167–183, 1998.

[5] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, Eds., *Machine Learning, Neural and Statistical Classification*, Ellis Horwood Limited, 1994.

[6] D.E. Patterson, R.D. Cramer, A.M. Ferguson, R.D. Clark, and L.E. Weinberger, "Neighborhood behavior: A useful concept for validation of molecular diversity descriptors", *Journal of Medicinal Chemistry*, vol. 39, pp. 3049–3059, 1996.

[7] Robert P. Velthuizen, Lawrence O. Hall, Laurence P. Clarke, and Martin L. Silbinger, "An investigation of mountain method clustering for large data sets", *Pattern Recognition*, vol. 30, no. 7, pp. 1121–1135, 1997.