miniDVMS v1.8 : A User Manual

(The Data Visualisation & Modeling System)

Dharmesh M. Maniyar

Aston University

June 2006

Aston University

miniDVMS v1.8 : A User Manual
(The Data Visualisation & Modeling System)

Dharmesh M. Maniyar

Summary

Today, the data available to tackle many scientific challenges is vast in quantity and diverse in nature. The exploration of heterogeneous information spaces requires suitable mining algorithms as well as effective visual interfaces. miniDVMS v1.8 provides a flexible visual data mining framework which combines advanced projection algorithms developed in the machine learning domain and visual techniques developed in the information visualisation domain. The advantage of this interface is that the user is directly involved in the data mining process. Principled projection methods, such as generative topographic mapping (GTM) and hierarchical GTM (HGTM), are integrated with powerful visual techniques, such as magnification factors, directional curvatures, parallel coordinates, and user interaction facilities, to provide this integrated visual data mining framework. The software also supports conventional visualisation techniques such as principal component analysis (PCA), Neuroscale, and PhiVis.

This user manual gives an overview of the purpose of the software tool, highlights some of the issues to be taken care while creating a new model, and provides information about how to install and use the tool. The user manual does not require the readers to have familiarity with the algorithms it implements. Basic computing skills are enough to operate the software.

Keywords: Data visualisation, machine learning, information visualisation, graphical user interface.

Contents

Chapter 1

Introduction

1.1 Motivation

The wide availability of ever-growing datasets from different domains has created a need for effective knowledge discovery and data mining. For data mining to be effective, it is important to include the domain expert in the data exploration process and combine the flexibility, creativity, and general knowledge of the domain expert with automated machine learning algorithms for better results. The principal purpose of visual data exploration is to present the data in a visual form with interactive exploration facilities, allowing the domain expert to get insight into the data, draw conclusions, and understand the structure of the data.

The exploration of heterogeneous information spaces requires suitable mining algorithms as well as effective visual interfaces. Visual techniques on their own cannot entirely replace analytic nonvisual mining algorithms to represent a large high-dimensional dataset in a meaningful way. Rather, it is useful to combine multiple methods from different domains for effective data exploration. We integrate both mining algorithms (principled projection algorithms) and visual methods (from information visualisation domain) in such a way that the visualisation results can be explored in detail and intermediate steps of the mining algorithms can be visualised and further guided by the domain expert. This allows users to control and steer the mining process directly based on the given visual feedback.

Projection of high-dimensional data on a lower-dimension space is an important step to obtain an effective grouping and clustering of a complex high-dimensional dataset. Here, we use the term *projection* to mean any method of mapping data into a lower-dimensional space in such a way that the projected data keeps most of the topographic properties (i.e. 'structure') and makes it easier for the users to interpret the data to gain useful information from it. Traditional projection methods such as principal component analysis (PCA), Neuroscale [1] and self-organizing maps (SOM) [2] are widely used in the knowledge discovery and data mining domain. For many real-life large high-dimensional datasets, the generative topographic mapping (GTM) [3], a principled projection algorithm, provides better projections than those obtained from the traditional projection methods [4]. Moreover, since the GTM provides a probabilistic representation of the projection manifold, it is possible to analytically describe (local) geometric properties anywhere on the manifold. The details of how these geometric properties of manifold can be used during visual data mining are presented in Section 3.1.1.

It has been also argued that a single two-dimensional projection, even if it is non-linear, is not usually sufficient to capture all of the interesting aspects of a large high-dimensional datasets. Hierarchical extensions of visualisation methods allow the user to "drill down" into the data; each plot covers a smaller region and it is therefore easier to discern the structure of the data. Further information on creating and using a hierarchical visualisation model is presented in Chapter 3.

1.2 The integrated visual data mining framework

The integrated visual data mining framework combines principled projection algorithms and visual techniques to achieve a better understanding of the data space. It conforms to Shneiderman's

mantra [5],"Overview first, zoom and filter, details on demand", to provide an effective interface (tool).

To support the 'overview first' stage of Shneiderman's mantra, output of the projection algorithms and basic visualisation aids such as highlight, rotate, etc., are provided for exploring a large high-dimensional dataset. For the second stage, 'zoom and filter', visualisation aids such as zooming, filtering interesting regions on the projection manifold with the use of magnification factor and directional curvatures plots, etc., are provided. This allows the user to identify and concentrate on interesting subsets of the projection we obtained in the first stage. The third stage, 'details-on-demand', is supported using local parallel coordinates and the ability to save subsets of the data. Integration with other visualisation tool is also possible at various stages.

Interactive visual methods support the construction of HGTM models and allow the user to explore interesting regions in more detail. Visual aids are provided at each stage of the HGTM model development. First, a base (*Root*) GTM is trained and used to visualise the data. Then the user identifies interesting regions on the visualisation plot that they would like to explore in greater detail. After training the child GTMs and seeing the lower level visualisation plots, the user may decide to proceed further and model in greater detail some portions of the lower level plots. Thus, HGTM allows domain experts to segment the input space interactively using data visualisation.

When the dataset is very large, the higher-level projection plots may be cluttered and confused (with densely clustered and overlapping projections). This makes it difficult for the user to select locations for submodels at the next level. In such cases, an alternative semi-automatic submodel initialization algorithm [6], based on minimum message length (MML) criteria, which decides both the number of submodels and their location can be used for higher-level projections of the visualisation hierarchy and then the domain expert can take control to guide the lower-level projections.

miniDVMS v1.8 is an interactive software tool that supports this framework. The details of the framework are reported in [7].

1.3 Data Visualisation & Modelling System (DVMS)

We have developed a Data Visualisation and Modelling System (DVMS) to facilitate domain experts with visualisation and modelling algorithms. The miniDVMS v1.8 is a smaller version of DVMS which supports visualisation using PCA, NeuroScale, PhiVis [8], GTM and HGTM algorithms. miniDVMS is designed as an easy-to-use, interactive, graphical tool to help the users to visualise and understand data. The software can be used to visualise any data, as long as it is in the required format (which is discussed in 2.2).

The software is developed in MATLAB[1] using the NETLAB toolbox [9]. It can work as a stand-alone application on Microsoft Windows and GNU/LINUX platforms.

1.3.1 Installing miniDVMS

The miniDVMS software can be used without a MATLAB installation. miniDVMS is provided on a CD. Complete stand-alone version of miniDVMS v1.8 with all the required libraries is around 250MB. User should carry out following steps to run miniDVMS on a machine:

- Copy the directory with `dvms.bat`, the `bin` directory, and the `MCR.zip` file from the CD on to the hard disk.
- Uncompress the `MCR.zip` file `dvms.bat` file is. This should create a `MCR` directory containing the MATLAB run-time library.
- Double click the batch file (`dvms.bat`) to run the software.

If above process gives any error, the problem might be in the initialisation of the Matlab Component Runtime (MCR). Please use the MCRInstall.exe file, provided in the 'extras' folder on the CD, to manually install MCR on the machine and set the `path` environment variable appropriately.

Alternatively, miniDVMS v1.8 can directly be run from the CD, which is slower. This document is available in the `docs` directory in the software CD.

[1] The MathWorks Inc., `http://www.mathworks.com/`

Chapter 2

Using miniDVMS

This chapter provides information about how to use the miniDVMS software. The entire process of developing new models using miniDVMS can be divided into 4 steps as below:

1. creating the *configuration* file;
2. creating the *data* file;
3. creating *models*;
4. using the *models*.

Section 2.1 and Section 2.2 describe the 1st and the 2nd steps respectively. Chapter 3 highlights important issues to be taken care of while creating a new visualisation model and using it.

2.1 The *configuration* file

The *configuration* file contains information about preprocessing required on the data and the properties of the data. It also hold information about options for how the output is generated. The *configuration* file can be created using a text editor. An example of a sample configuration file is given below:

```
Begin Header
$LABELING
1
$NORMALISATION
0
$NO_VARIABLES
10
$DIM_LATENT
2
$PROPERTYHEADER
0
$NUM_OF_LABELS
4
$LABEL_NAMES
CLASS 1
CLASS 2
CLASS 3
CLASS 4
End Header
```

The *configuration* file must start with the `Begin Header` row and it must have the `End Header` row as the last row. For the current version of miniDVMS, there should be six different type of options stored in the *configuration* file. Each option starts with the `$` sign on a separate row.

- `$LABELING :` This option is used to specify if the records in the data file have labeling information. The class information is only used to color the data points on the projection plots and is not included in the variables used by the visualisation projection. As described in Section 2.2, the last field of the *data* file can be a label field. If the *data* file has the label information, then this option should be set to `1`. Otherwise it should be set to `0`. The example of the *data* file shown in Section 2.2 has label information, so the *configuration* file should have `$LABELING` option set to `1`.

- `$NORMALISATION :` This option is used to specify whether to normalise the data or not. If it is set to 0, the data will not be normalised. If it is set to 1, the data is normalised by treating each variable as independent and normalising it to have a mean of zero and standard deviation of one, as

$$\tilde{x}_i^n = \frac{{x_i}^n - \bar{x}_i}{\sigma_i}, \tag{2.1}$$

  where $n = 1, ..., N$ labels the patterns, $\bar{x}_i$ and ${\sigma_i}^2$ represent the mean and variance of the ith variable respectively.

  If this flag is set to 2, the data is normalised using the whitening technique [10].

  Data having diverse scales across the variables should be normalised for useful results.

- `$NO_VARIABLES :` This represents dimension of the data space. It should be a number specifying number of variables the *data* file contains. As explained in the section 2.2, the number of variables in the *data* file is total number of fields in the *data* file except the ID field (the first field) and the labeling field (if any, it should be the last field).

- `$DIM_LATENT :` This variable decides the dimension of latent space to be used. Generally, this variable is set to `2`; if the user wants a 3D projection, they can set it to 3. Currently this variable should be set only to 2.

- `$PROPERTYHEADER :` If the first row of the data file contains variable ('property') names, this option is set to `1`, else it should be `0`.

- `$NUM_OF_LABELS :` This variable is to inform the software about the number of labels (classes) in the dataset. The following row should contain a value indicating the number of classes in the dataset.

- `$LABEL_NAMES :` This variable should be followed by the class labels. Each label name should be specified on a separate row. Thus, there should as many rows as the number of classes in your dataset.

2.2 The *data* file

The *data* file should contain the raw data inputed to the system in the Comma Separated Value (CSV) file format. If the `$PROPERTYHEADER` option in the *configuration* file is set to `1`, the first row of the

data file should consist of a list of variable names. Otherwise, the data file should not contain any header row. As the format suggests, the columns in the *data* file should be separated by "comma" and each record is on separate row. The first column is the ID (in the case of drug discovery data, it might be the compound ID; in bioinformatics applications it might be a gene identifier). Subsequent columns, until the last column, should be the values (miniDVMS v1.8 supports only numeric values) of different variables (for example, screening results, physicochemical data, etc.).

It is often useful to include a 'label' information that classifies data points. This classification is used to colour the data points, which helps to understand the relationships in the dataset. It also helps the user to see if the different classes are clearly separated in the visualisation plots, which is a useful criterion for determining whether the training process is complete during the creation of hierarchical visualisation models. The last column of the labelled data is for the labeled data (data with known classes). If the flag `$LABELING` is set to 1 in the *configuration* file, then the user has to provide last column of data file as label information. These label values should be consecutive integers from 1 (i.e. the first 'class' is labelled with 1, the second with 2 etc.). Otherwise the last column should be the last variable of the actual data.

An example of a *data* file ...

```
1,2.6,-2.2,  ...  0.45,6.68,1
5,37.6,0.7,  ...  2.82,8.47,1
8,-49,0.6,   ...  4.44,7.36,2
10,-9.2,4.4, ...  3.81,0.67,2
11,15.8,4.4, ...  3.46,55.38,2
13,50.6,0.4, ...  4.831,42.56,3
14,13,-3.2,  ...  2.319,8.17,3
15,-2,16.4,  ...  2.958,1.58,3
...
...
```

Data files in this format can be directly generated using common tools such as Microsoft Excel, SPSS, etc.

The following issues should be considered when creating a data file.

2.2.1 Data selection

Data selection is a vital part of the process because if the variables that are chosen do not contain useful information, it is impossible to get any insight from a visualisation tool. However, this does not mean that every possible variable should be included. The reason for this is that if too many variables are used, then the interesting underlying relationships in the data can be obscured by unimportant variations (or 'noise') on other variables. Luckily, visualisation by its very nature helps the user explore a dataset, and so the results can be used to guide variable selection.

2.2.2 Data preprocessing

miniDVMS v1.8 requires all variables to be expressed as numbers (i.e. it does not explicitly cater for discrete variables). It is also helpful if all the variables are measured on a similar scale. For example,

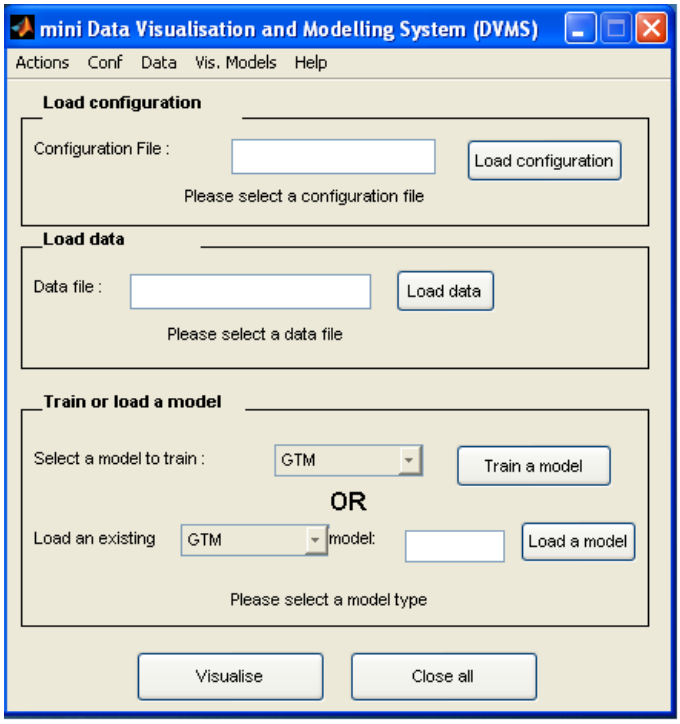

Figure 2.1: Main interface of the DVMS

if the range of one variable is -1000 to 1000, and the range of the others is -1 to 1, then the first variable will dominate the results. A common technique is to *normalise* each variable to have a mean of zero and standard deviation of one. miniDVMS v1.8 provides this facility as described in the section 2.1.

Normalisation works well in most circumstances, but problems can still arise if there are significant *outliers*: data values which are *very* different from the norm. This may prevent the model from being trained successfully, but more usually, the visualisation plot shows the bulk of the data in one large indistinguishable cluster and just a few data points well separated from it. One of the advantages of using visualisation is that it enables the user to see the presence of these unusual points and then exclude them from the main analysis. Alternatively, if HGTM is used, then sub-models can be placed to split the outliers from the rest of the data.

The miniDVMS v1.8 requires every entry in the data matrix to have a value. It is possible to train GTM (and HGTM) on datasets where some values are missing, this is a future extension for the tool. If some values are missing, then either the data point should be excluded or they should be replaced by the mean value for the corresponding variable.

It is required to load the *configuration* and *data* files before training a model. It can be done using the main interface of the miniDVMS as shown in the Figure 2.1. The interface has 'Conf' and 'Data' menus to do the task or the user can also use the 'Load Configuration' and 'Load Data' sections of the main interface. Once a model is trained, the user can save it and test it on the testing set.

Chapter 3

Creating and using visualisation models

This chapter discusses different issues one should consider during the development of visualisation models. There is more to developing a good visualisation model than simply running a training algorithm. Model development is a process, and each stage must be carefully considered if the end result is to be useful. Two important issues in creating a good model data selection and data pre-processing, are discussed in Section 2.2.1 and Section 2.2.2 respectively. Two other important steps; model training and model evaluation, are discussed here.

3.1 Visualising trained models

Models, trained as explained in Section 3.2, can be loaded to visualise the data. Loading an existing model is simple. It can be done using the 'Model' menu or the 'Load Model' button interface on the main screen of the miniDVMS (Figure 2.1). The status of the model loading is displayed just below the 'Load Model' textbox. The 'Visualise' button on the main interface of the miniDVMS (Figure 2.1), allows the data to be visualised in the latent space according to the loaded data and model.

3.1.1 Visualisation of a PCA, Neuroscale, or GTM model

Figure 3.1 demonstrates the interface for PCA, Neuroscale, and GTM model visualisation. Using the interface, the user can explore nearest points in data space and can even save the latent space points as a comma delimited text file.

It is very useful to relate the visualisation of latent space to the data space. This facility is provided by the 'Local parallel coordinates (LPC)' button available on the 'Latent Space Visualisation' interface (Figure 3.1). The user can adjust the number of nearest neighbors to be displayed for the data space property visualisation. Once the LPC button is pressed, the user can left click any latent space point to visualise the nearest points to that point in data space. The LPC generated, displaying properties

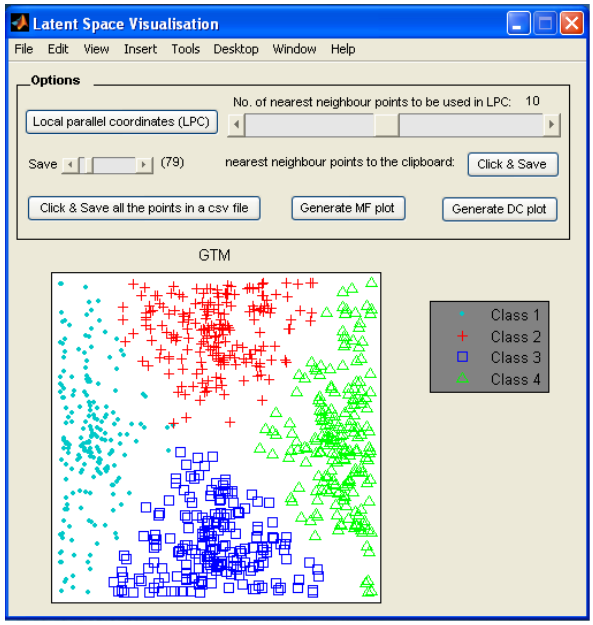

Figure 3.1: Data visualisation in latent space using a GTM model

in the data space (as shown in Figure 3.9), has some interactive facilities too. The user can select particular IDs (the width of the line associated with the ID will increase while it is selected) by clicking on the corresponding line or ID. If the properties header was set in the *data* file, right clicking the ID will give a list of property names with actual data space values for that particular ID. If the properties header was not specified, only the actual data space value is displayed in a list. An example can be seen in Figure 3.9. The LPC facility is deactivated by right clicking on the latent plot.

One of the main advantages of using GTM–based models is that it is possible to analytically calculate the Magnification Factors (MF) [11] and the Directional Curvature (DC) [12] of the GTM projection manifold. MFs of a GTM projection manifold, Ω, are calculated as the determinant of the Jacobian of the GTM map f [11]. Magnification factor plots are used to observe the amount of stretching in a GTM manifold at different parts of the latent space, which helps in understanding the data space, outlier detection, and cluster separation. Tiňo *et. al.* [12] derived a closed-form formula for directional curvatures of the GTM projection manifold, Ω, for a latent space point $\mathbf{x} \in \mathcal{H}$ and a directional vector $\mathbf{h} \in \mathcal{H}$. Directional curvature plots allow the user to observe the direction and amount of folding in the GTM manifold. This can help the user detect regions where the GTM manifold does not fit the data well. It is possible that groups of data points far apart when projected onto the projection manifold are close together in the data space due to high folding in the manifold. This neighborhood preservation in the data space can be spotted with a strong curvature band on the corresponding directional curvature plot.

The magnification factor is represented by color shading in the projection manifold (e.g., see Figure 3.3). The lighter the color, the more stretch in the projection manifold. The direction of folding in the projection manifold plot is presented using a small line for each part of the projection manifold

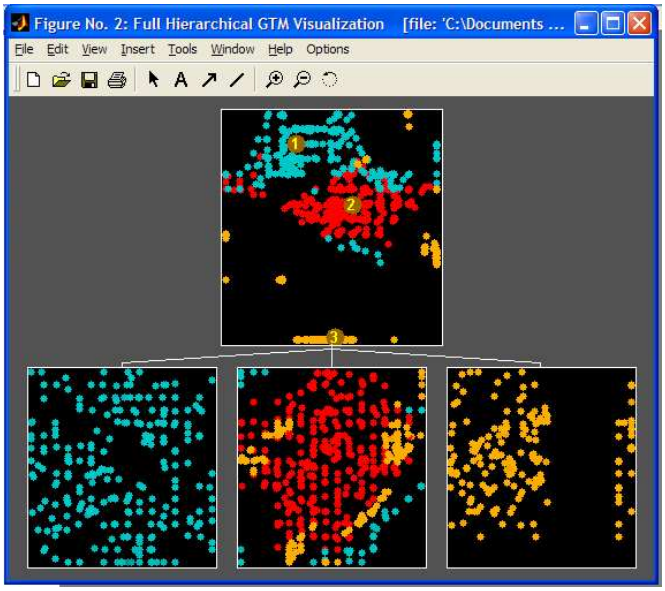

Figure 3.2: An example of hierarchy generated using the HGTM algorithm

in the directional curvature plots (e.g., see Figure 3.4). The length and the shade of the background color represents the magnitude of folding. The longer the line and the lighter the background color, higher the folding (curvature).

3.1.2 Visualisation of a HGTM model

HGTM model on data is visualised as a hierarchy of GTMs as shown in Figure 3.2. The interface provides an '`Options`' menu which can be used to display magnification factors (as demonstrated in Figure 3.3) and compute and show directional curvatures (as shown in Figure 3.4).

3.2 Training a model

The purpose of training a model is to adjust the model parameters (sometimes known as weights) so that the model fits well to the data. The quality of the fit is measured using an *error function*: the smaller the value of the error function (which may be negative) the better the fit. Note that the error function for GTM and HGTM is quite different from that for Neuroscale, and hence the values cannot be compared between these models.

The key question is how well the model fits the underlying generator of the data; we say that a good model *generalises* well to new data. This can be measured by *testing* the model (i.e. evaluating the error function) on a separate dataset. It is this property of generalisation that enables the user to train the model on a smaller sub-sample of the data (usually a relatively slow process) and then visualise the main dataset (usually a fast process).

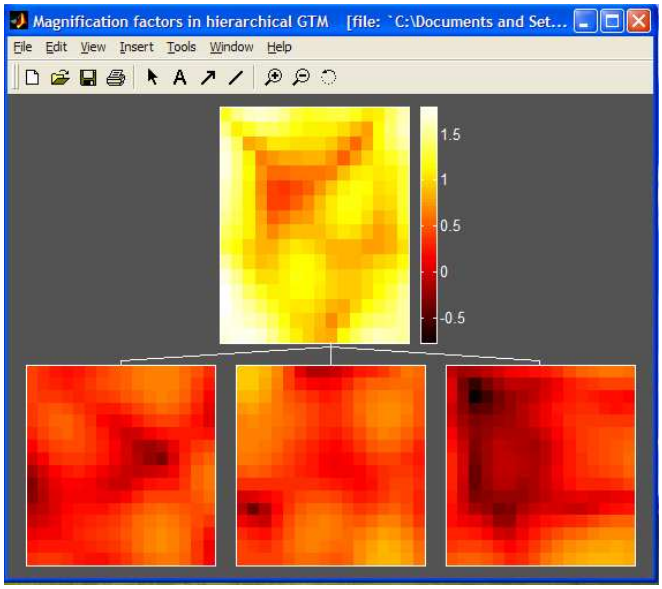

Figure 3.3: Corresponding Magnification Factor for the HGTM hierarchy

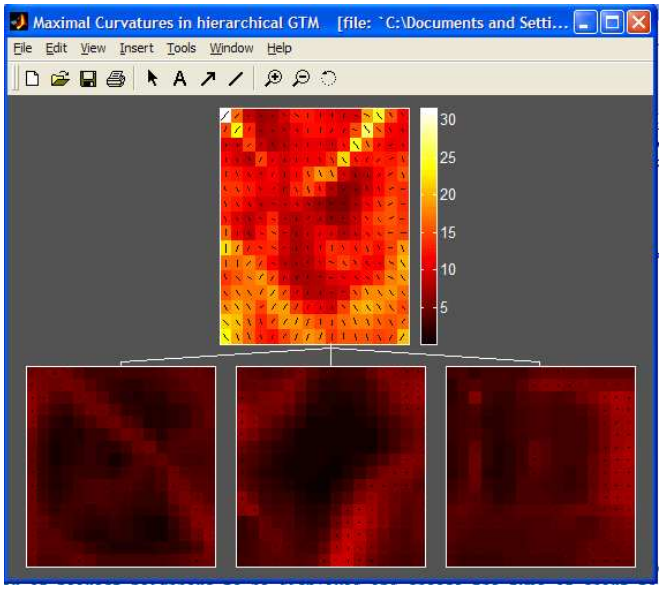

Figure 3.4: Corresponding Directional Curvature for the HGTM hierarchy

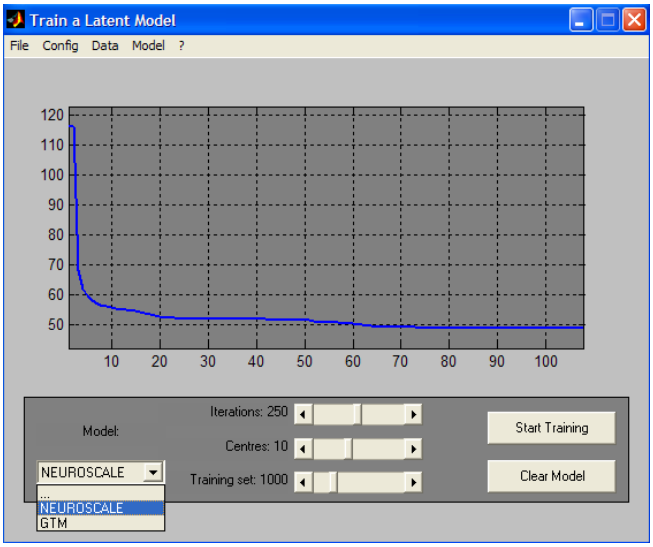

Figure 3.5: Interface for training a Neuroscale or GTM model

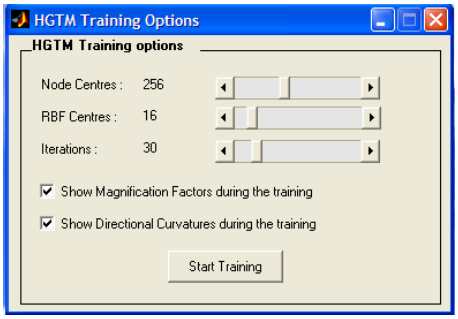

Figure 3.6: HGTM adjustable parameters during the training

3.2.1 Deciding parameters

When training a model, there are certain macro-level parameters that the user needs to determine. Adjustable parameters settings for training a Neuroscale or GTM model can be seen in Figure 3.5 and for the HGTM model can be seen in Figure 3.6. The main architectural parameter for NeuroScale is the number of RBF centres. For GTM and HGTM they are the number of node centres (Gaussians) and number of RBF centres.

Because PCA does not have any user-adjustable parameters, a PCA model is created simply by using the 'load model' facility.

Model complexity

Model complexity is a function of the size and structure of the model. Typically, large numbers (number of RBF centres or node centres) allow the model to be more complicated. If the number is too small, then the model will be too simple and will have a large error on the training data. If the number is too large, then the model will have a low error on the training data, but a larger error

on new data because the model is too specific to the details of the training data (i.e. the model is *overtrained* or *overfitted* to the data).

One way to determine a good value for architectural parameters is to train several models with a range of values and compare the generalisation performance. We should look for the simplest model that generalises well.

NeuroScale: number of hidden units (RBF centres). The larger the number, the more complex the projection function can be.

GTM: The GTM can be interpreted as a two-dimensional rubber sheet in data space: spherical blobs placed on the sheet capture the fact that the data lies near to, but not exactly on, the sheet.

1. Number of node centres. The Gaussians are the spherical blobs: the more that there are, the better the data can be modelled. However, the number of training iterations is proportional to the number of Gaussians, so using too many can make training very slow. It is harder to overfit, although this is possible.
2. Number of RBF centres. This governs the complexity of the map from the computer screen to data space: effectively the amount of stretch and curvature of the rubber sheet. The larger the number, the more complex the map.

HGTM: as this consists of a tree of GTM models, the architectural parameters for the GTM need to be set as each individual model is trained. In addition, the user will need to decide the number of levels and the number of child nodes at each level. To a large degree, this is a matter of how well the current set of visualisation plots explains the data. The issue is discussed further in Section 3.2.2.

Training iterations

The user has to decide how many iterations the algorithms should run for. The principle of determining when to stop training the single models (GTM and NeuroScale) is straightforward: each model should be trained until the error value has converged. During training, graphs are plot the logarithm of error values. Once the error plot has reached a plateau (as shown in Figure 3.5), no more training is required. If the error curve has not reached a plateau when the training algorithm terminates, then the model should be trained further by clicking again on the 'Start training' button. Training a hierarchical model is recursive: once the top level GTM has been trained, every leaf node in the tree can be extended with child models. The next section provides more information on issues concerning training an HGTM model.

3.2.2 Interactive training for HGTM model

The additional aspects of training a hierarchical model are: how to add child plots; when and why to add child plots and when to stop.

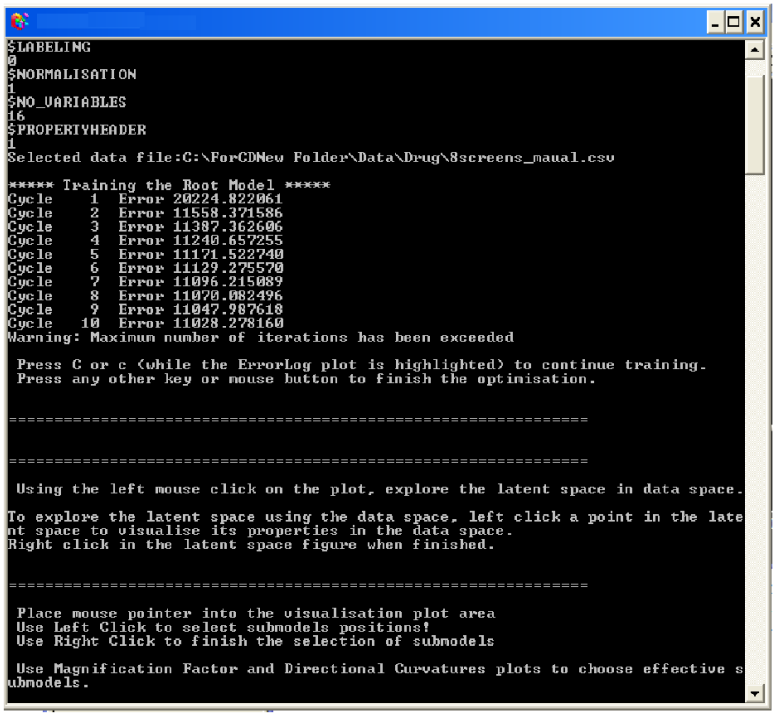

Figure 3.7: Typical interaction during the HGTM training

How to add child GTMs. Child models are added to a leaf node in the current tree. The user selects points, $\mathbf{c}_i \in \mathcal{H}, i = 1, 2, ..., A$, in the latent space that correspond to centres of the subregions they are interested in. The points $\mathbf{c}_i$ are then transformed via the map f to the data space. Then the subregions are formed using Voronoi compartments [13].

Adding child GTMs using miniDVMS is easy. The user can left click on the parent GTM plot to select centres for the submodels and right click when the selection of submodels is finished. Relevant instructions are provided on the plot and in the miniDVMS interaction window (as shown in Figure 3.7) during the training process.

When to add child GTMs. GTM models the data as a curved and stretched two-dimensional sheet. However, if the data points that a leaf model in the tree is responsible for do not lie close to such a surface, then the visualisation plot will be misleading. So, the basic principle of adding new child GTMs is to partition the data so that *locally* it lies close to a two-dimensional sheet. We can use the *parallel coordinate* facility provided by miniDVMS to explore patterns of nearest points (measured using Euclidean distance) from the point selected in the latent space as shown in Figure 3.9. This can be very useful in understanding different regions of the latent space as the user can see the corresponding data space value.

Thus the user should add a child GTM to a leaf model if:

1. The plot is cluttered with too many points and we can not see separate clusters.

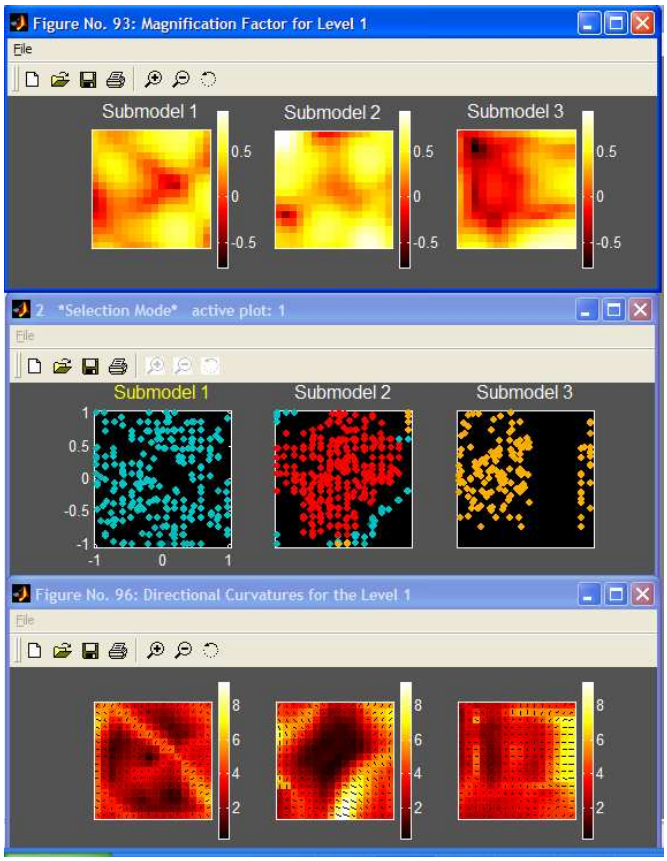

Figure 3.8: An example of plots during the HGTM training

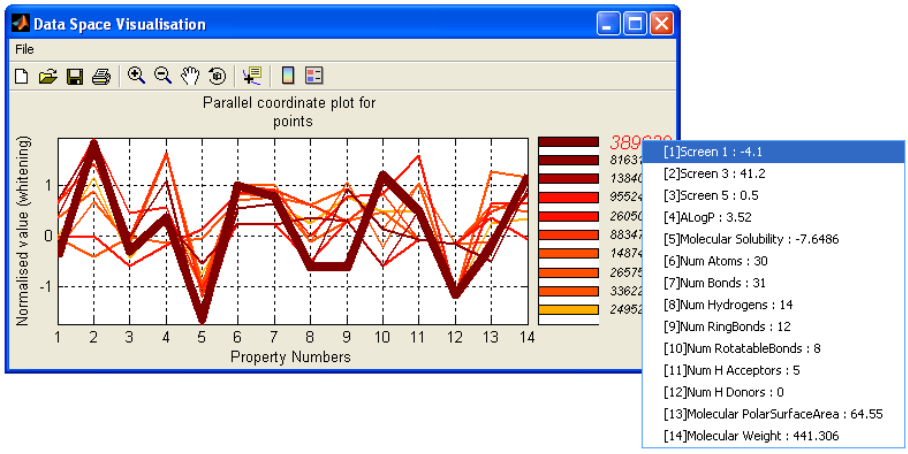

Figure 3.9: Exploring the data space using *parallel coordinate* technique

2. With the help of the curvature plots, the user decides if the model is not flat. It is particularly helpful to put child models on either side of bands of large curvature, as this 'slices' the data into two simpler segments. For example, notice in Figure 3.8 there is a strong curvature band in the bottom-right corner of submodel 2. Having two submodels either side of this curvature could be useful. From the labels (color code) of the data points, it can be confirmed that it was a good decision.
3. The magnification factor plot shows that some areas of the map are being stretched a long way. Putting child models in regions of high data density, creates child plots that are less stretched.

When to stop. One should stop adding models when the visualisation plots are telling everything that one needs to know. One way of deciding this is when the leaf node plots look similar to their parents.

If we are visualising the data, and not trying to build predictive models, then it is not necessary to create a single GTM plot for each significant data cluster; it is enough if the leaf nodes show well separated clusters of data.

Training effectiveness is shown using a similar error graph as shown in Figure 3.5. We should look for the training error to end with a plateau, which means that the learning algorithm is approximating to a minimum of the learning cost function. As this stage, we can change parameters and start training the model again or can decide to train the same model further.

3.2.3 Model evaluation

There are two main aspects of model evaluation: how well the model fits the underlying data generator and how informative the visualisation plot is.

The first of these is best measured by generalisation performance: computing the error measure on a testing dataset. A good model should have a similar value of error per data point on the test set and the training set.

Assessing the quality of the visualisation plots themselves is something that is subjective. The magnification factor and curvature plots (for GTM) can help with this, as can a more detailed exploration of local regions with the visualisation of nearest points in data space (using the parallel coordinate technique as shown in Figure 3.9). Some experimentation with the model architecture and the variables that are included is an inevitable part of exploring the data and improving the model. Once a good visualisation model is created the visual results are relatively easy to understand the data for the domain experts.

Bibliography

[1] D. Lowe, M. E. Tipping, Neuroscale: Novel topographic feature extraction with radial basis function networks, Advances in Neural Information Processing Systems 9 (1997) 543–549.

[2] T. Kohonen, Self-Organizing Maps, Springer-Verlag, Berlin, 1995.

[3] C. M. Bishop, M. Svensén, C. K. I. Williams, GTM: The generative topographic mapping, Neural Computation 10 (1998) 215–234.

[4] D. M. Maniyar, I. T. Nabney, B. S. Williams, A. Sewing, Data visualization during the early stages of drug discovery, Journal of Chemical Information and ModellingASAP Web Release: http://pubs3.acs.org/acs/journals/doilookup?in_doi=10.1021/ci050471a.

[5] B. Shneiderman, The eyes have it: A task by data type taxonomy for information. visualizations, Proceedings of the 1996 IEEE Symposium on Visual Languages 3 (6) (1996) 336–343.

[6] I. T. Nabney, Y. Sun, P. Tiňo, A. Kabán, Semisupervised learning of hierarchical latent trait models for data visualization, IEEE Trans. on Knowledge and Data Engineering 17 (3) (2005) 384–400.

[7] D. M. Maniyar, I. T. Nabney, Visual data mining using principled projection algorithms and information visualization techniques, in: Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2006.

[8] M. E. Tipping, C. M. Bishop, Mixtures of probabilistic principal component analysers, Neural Computation 11 (2) (1999) 443–482.
URL `citeseer.ist.psu.edu/tipping98mixtures.html`

[9] I. T. Nabney, Netlab: Algorithms for Pattern Recognition, 1st Edition, Springer, 2001.

[10] C. M. Bishop, Neural Networks for Pattern Recognition, 1st Edition, Oxford University Press, 1995.

[11] C. M. Bishop, M. Svensén, C. K. I. Williams, Magnification factors for the GTM algorithm, Proceedings IEE Fifth International Conference on Artificial Neural Networks (1997) 64–69.

[12] P. Tiňo, I. T. Nabney, Y. Sun, Using directional curvatures to visualize folding patterns of the GTM projection manifolds, Artificial Neural Networks - ICANN (eds) G. Dorffner, H. Bischof and K. Hornik (2001) 421–428.

[13] F. Aurenhammer, Vornoi diagrams - survey of a fundamental geometric data structure", ACM Computing Surveys 3 (1991) 345–405.