# DVMS v3.0 User Guide

Dharmesh M. Maniyar (v1.8, 2007)
John R. Owen (v2.4, 2009)
Shahzad Mumtaz (v3.0, 2011)

NEURAL COMPUTING
NCRG
RESEARCH GROUP

ASTON UNIVERSITY

June 2011

ASTON UNIVERSITY

# DVMS v3.0 User Guide

Dharmesh M. Maniyar (v1.8, 2007)
John R. Owen (v2.4, 2009)
Shahzad Mumtaz (v3.0, 2011)

**Preface**

The data available to tackle many scientific challenges is vast in quantity and diverse in structure. With the continuous advance of science and computers, datasets will not in the future become smaller or less common. The exploration of heterogeneous datasets requires suitable mining algorithms as well as effective visual interfaces. DVMS is a software system which combines advanced projection algorithms (developed in the machine learning domain) and visualisation techniques (developed in the information visualisation domain). DVMS enables to the user to become directly involved in the data mining process. Principled projection methods, such as Generative Topographic Mapping (GTM), and Hierarchical GTM (HGTM), are integrated with powerful visualisation techniques, such as magnification factors, directional curvatures, and parallel coordinates, to provide a visualisation environment. DVMS also includes Principal Component Analysis (PCA) the most commonly-encountered visualisation technique, the variants of GTM (such as GTM (log-version), GTM with simultaneous feature saliency (GTM-FS), GTM-FS (log-version)), Gaussian Process Latent Variable Model (GPLVM), and also NeuroScale (NSC) the one lesser-known visualisation technique.

This guide, the DVMS User Guide, provides an overview of DVMS, gives installation details, and describes how to create a statistical model of a dataset. DVMS is easy to use – the user will need only a basic knowledge of the statistical techniques it implements.

**Keywords:** dvms, user, guide, ncrg, aston, university, uk, data, visualisation, modelling, system

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This introduction gives: some background on DVMS's development; a discussion on DVMS's overall design ethos; instructions for the installation of DVMS.

## 1.1 About DVMS

A program, the *Data Visualisation and Modelling System* (DVMS), has been developed in the Neural Computing Research Group (NCRG), Aston University. DVMS provides an easy-to-use interface to useful visualisation and modelling algorithms (some of which were invented by the NCRG). The following algorithms have been included in DVMS: PCA, GTM, GTM (log-space version), GTM-FS, GTM-FS (log-space version), GPLVM, HGTM, , LTM, and NSC. DVMS has been designed to help the user understand and investigate datasets. The software can be used to probe more or less any dataset (providing the dataset can be submitted in the required format, as given in Chapter 2).

DVMS is written in Matlab and is highly dependent on the Matlab-based toolbox, Netlab. (Netlab was written by Professor Ian T. Nabney of the NCRG.) The original DVMS, version 1.8, was written by Dharmesh M. Maniyar, a PhD student in the NCRG during 2005-08. An upgraded version, version 2.3, was released in June 2009 by John R. Owen, also a PhD student in the NCRG. The main new additions in version 2.3 were: algorithms to produce NSC plots of MDL166 molecular-fingerprint data; an improved NSC-training GUI; the column-type row. Molecular-fingerprint plotting was requested by Pfizer Ltd. (who co-funded, along with the BBSRC, the PhD project supporting DVMS). In August 2009, version 2.4 of DVMS was released, the main enhancement being the inclusion of the Latent Trait Model (LTM). The current updated version, version 3.0, was released in December 2011 by Shahzad Mumtaz, also a PhD student in the NCRG during 2009-2012. The whole structure of the DVMS is revised in the current version to make it more simpler and easy to use by using object oriented features of Matlab. This revised version includes use of object oriented features provided by Matlab for managing the classes of configuration file, data files and training algorithms. Moreover, some of the major changes have also been done in GUI as well. Furthermore log-space versions of GTM and GTM-FS have also been incorporated in this version. Log-space version of GTM is based on the code that computes log probabilities of Gaussian mixture model. The original code to compute log probabilities was written by Alexander G. Dimitrov as contribution to Netlab and is available on NCRG website[1]. We made some changes in the code to enable it to compute log probabilities of Gaussian mixture model in high-dimensional data space to avoid numerical problems. Furthermore, an internee (i.e. Yigit Yildrim) working with Prof. Ian Nabney have included GPLVM in the current version of DVMS.

## 1.2 Why DVMS?

The wide availability of datasets from different domains has created a need for effective knowledge discovery and data mining. Data mining is most effective when the user can use machine learning and visualisation algorithms interactively. No algorithm to date can reproduce the insight or flexibility of

[1]http://www1.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/contributions/

a human expert. The principal purpose of visual data-exploration is to gain insight into a dataset, to draw conclusions, and to understand the structure of a dataset.

The exploration of heterogeneous information spaces requires suitable mining algorithms, as well as effective visual interfaces. Visual techniques alone cannot entirely replace analytic non-visual mining algorithms to represent large high-dimensional datasets in an easy-to-interpret way. Instead, it is useful to combine multiple methods from different domains for effective data exploration. DVMS integrates both mining algorithms, and visualisation methods (from information visualisation theory), in such a way that the results of a visualisation can be explored in detail, and intermediate steps of the mining algorithms can be visualised and guided by the expert. So within DVMS the user can interact with the data mining process, guided by visualisations.

The projection of high-dimensional data to a low-dimensional space (usually 2D) is an important step in obtaining effective clustering of large high-dimensional datasets. Here the term *projection* is used to mean any method of mapping data into a low-dimensional space in such a way that the projected data retains most of its structure. Projection methods include Principal Component Analysis (PCA), Gaussian Process Latent Variable Model (GPLVM) [5] and NeuroScale (NSC) [7]. (PCA is a well-known method and has been common in data mining since the 1980s.) For many real-life large-scale datasets, Generative Topographic Mapping (GTM) [3], developed in Aston University's NCRG, provides better projections than those of the traditional methods [10]. Moreover, since GTM provides a probabilistic representation of the projection manifold, it is possible to analytically describe (local) geometric properties at any point on the manifold. Details of how these geometric properties can be used in visual data mining are given in Section 3.2.1.

It has been argued that a single 2D projection, even if it is a non-linear projection, is not usually sufficient to capture all of the interesting structure in a large high-dimensional dataset. Hierarchical extensions of visualisation methods allow the user to "drill down" into the data; each plot contains a smaller region making the plot more interpretable. Instructions for creating hierarchical visualisation models with DVMS are given in Chapter 3.

## 1.3 The Visual Data Mining Framework

The visual data mining framework combines principled projection algorithms with visual techniques to allow the user to gain insight into a dataset. The framework conforms to Shneiderman's mantra [12] ("Overview first, zoom and filter, details on demand") to provide a useful piece of software.

To support the "overview first" stage of Shneiderman's mantra, output of the projection algorithms and basic visualisation aids, such as highlight and rotate, are provided for exploring a large high-dimensional dataset. For the second stage, "zoom and filter", visualisation aids such as zooming, and filtering with the aid of magnification factors, are provided. This enables the user to identify and concentrate on interesting subsets of the projections from the first stage. The third stage, "details on demand", is provided by plots of local parallel coordinates, and the ability to save subsets of the data. Integration with other visualisation tools is also possible at some stages.

Interactive visual methods support the construction of HGTM models and allow the user to explore interesting regions in magnified detail. Visual aids are provided at each stage of HGTM model development. First a base/root GTM is trained and used to visualise the data. Then the user can identify interesting regions on the visualisation plot for further exploration. After training a child GTM and seeing the lower-level visualisation plot, the user is given the ability to explore the lower-level plot in greater detail. Thus HGTM allows the user to interactively segment the input space.

With particularly large datasets ($> 3,000$ points), the higher-level plots produced may be rather cluttered. This will make it difficult for the user to select subregions of the plot to build submodels. In such cases a semi-automatic submodel-initialisation algorithm [8], based on minimum message length (MML) criteria, can be used. This algorithm will enable the user to guide the lower-level projections. Further details of the overall framework provided by DVMS are given in [9].

## 1.4 Steps to Install DVMS

DVMS has been compiled with the MATLAB compiler to run as a standalone application under Windows. It can be installed by following the steps below.

1. Create a directory on your hard disk to hold DVMS – this section assumes the directory is named `dvms_hold`. Copy all files and directories from the DVMS CD-ROM to `dvms_hold` (after this copy the CD-ROM is not required again). Enter `dvms_hold` – you should be able to find DVMS's two main distribution directories `distrib` and `src`. Enter the `distrib` directory and click on `dvms_pkg.exe`. This will unzip the main DVMS executable and some associated files.
2. The program VCREDIST_X86 should install automatically during the unzip. This installation will take around 10 minutes depending on the speed of your computer.
3. The MATLAB Compiler Runtime (MCR) should also install automatically during the unzip. (You should be able to click "Next" to all/most of the installation options.) DVMS is written in MATLAB and the MCR is the underlying MATLAB interpreter.
4. The files in Table 1.1 should now be present.
5. Once VCREDIST_X86 and the MCR have been installed on your computer, DVMS can be started by clicking on the executable `dvms.exe`. DVMS can now always be started by clicking on this executable; it won't be necessary to repeat the installation of VCREDIST_X86 or the MCR.

| **File** | **Description** |
|---|---|
| `dvms_pkg.exe` | The three files below zipped-up. |
| `MCRInstaller.exe` | The program that installs the MCR. |
| `dvms.exe` | The MATLAB-compiled DVMS executable. |
| `readme.txt` | A minor file created by MATLAB. |

Table 1.1: DVMS's installation files.

# Chapter 2

# Using DVMS

This chapter describes how to use DVMS. The process of building a statistical model with DVMS can be divided into four main steps:

1. Creation of the configuration file (mostly metadata on the data file).
2. Creation of the data file (the dataset to be analysed).
3. Model training (e.g. NSC model training).
4. Model visualisation and interpretation.

The initial interface provided with DVMS to support above stated process is shown in Figure **??**.

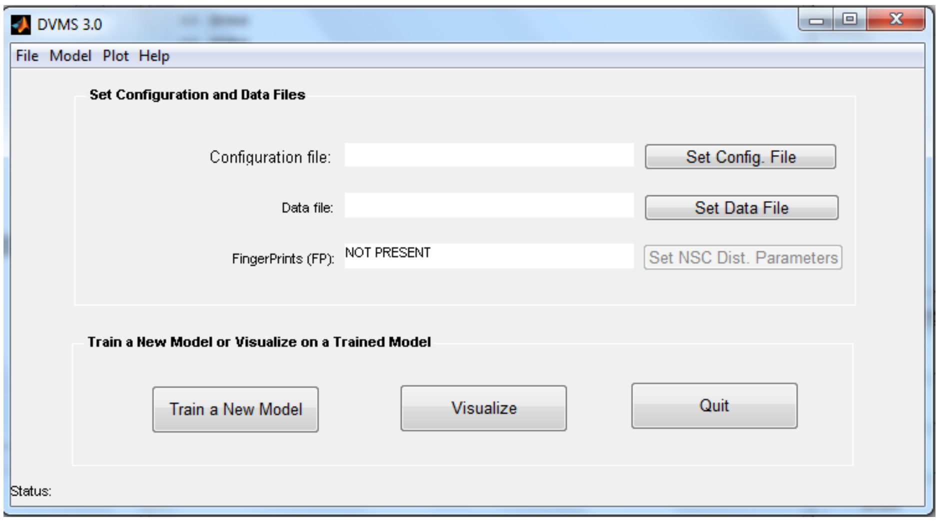


Figure 2.1: DVMS graphical interface.

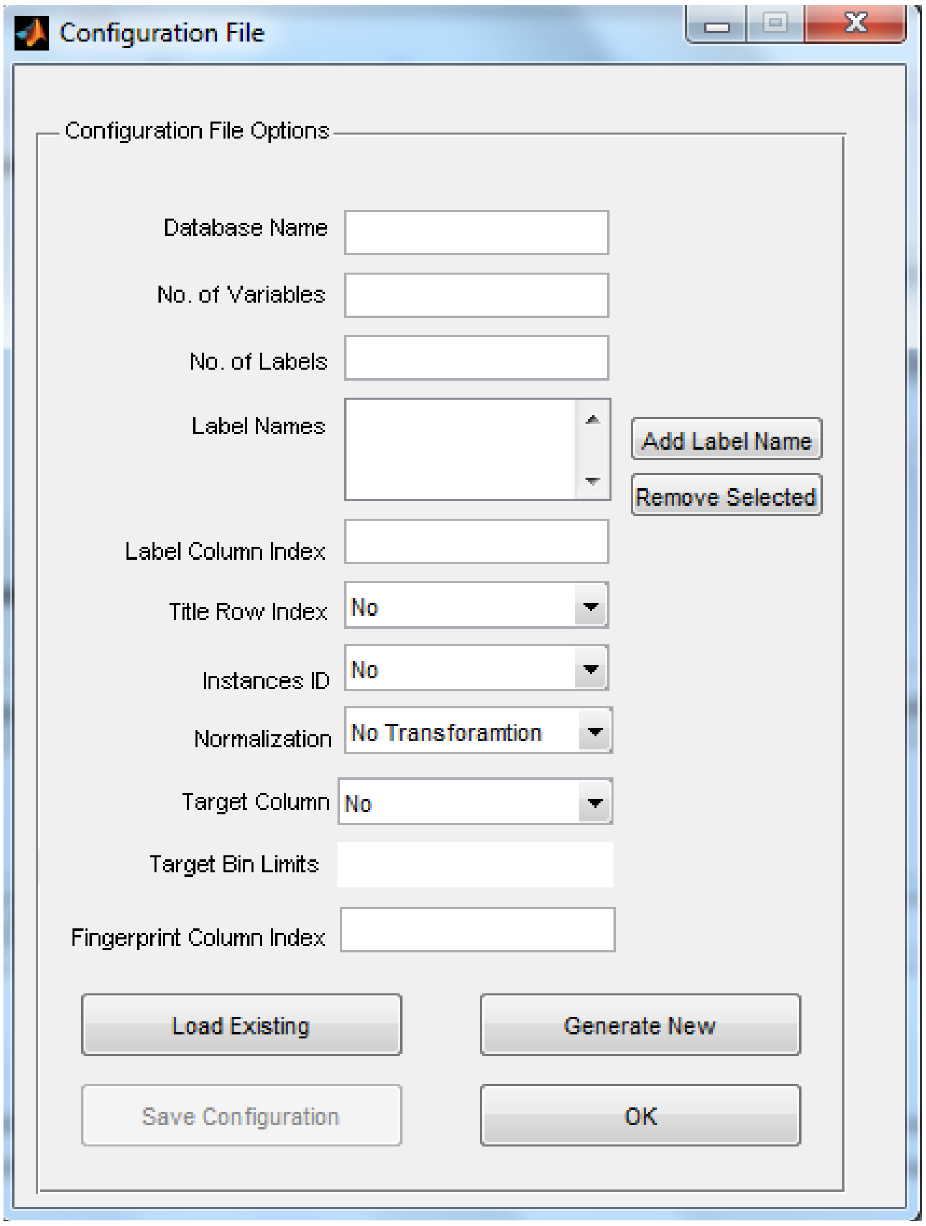
Sections A.1 and 2.1.2 describe steps (1) and (2) respectively. Steps (3) and (4) are covered in Chapter 3.

## 2.1 DVMS's Configuration and Data Files

Two files are central to building a model with DVMS – the configuration file and the data file. The configuration file contains mostly metadata on the data file; the data file contains the dataset itself. To understand the format of these two files is to inspect the example files supplied with DVMS – the reader is advised to look at these files whilst reading this chapter. Configuration files (extension .cfg) can be read with any text editor. Data files (extension .csv) are best read with a spreadsheet. Loading a data file in the current version of DVMS brings up the progress bar (Progress bar code was originally written by Ohad Gal in 2003 and available on website i.e. http://www.mathworks.com/matlabcentral/fileexchange/3607-progressbar).

### 2.1.1 Format of the Configuration File

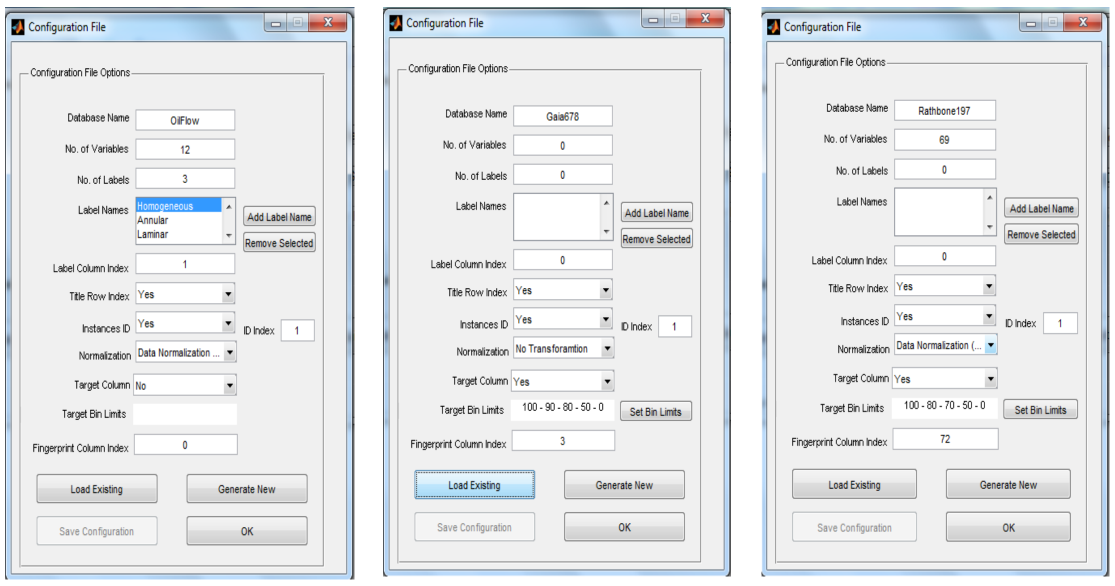
The first thing to do when building a new model is to write a DVMS configuration file. This file, commonly called the *config. file*, is mostly metadata about the dataset. The easiest way to generate the configuration file is to use the graphical interface (see Figure 2.2) provided with the current version of DVMS. However, a configuration file can also be generated using any of the text editor as supported by the previous versions of the DVMS and details are provided in Appnendix 2.2. Some examples for using the graphical interface to generate configuration files are shown in Figure 2.3.


Configuration File

Configuration File Options

Database Name:
No. of Variables:
No. of Labels:
Label Names:
Label Column Index:
Title Row Index: No
Instances ID: No
Normalization: No Transforamtion
Target Column: No
Target Bin Limits:
Fingerprint Column Index:

Figure 2.2: Configuration file graphical interface.



(a)

Database Name: OilFlow
No. of Variables: 12
No. of Labels: 3
Label Names: Homogeneous, Annular, Laminar
Label Column Index: 1
Title Row Index: Yes
Instances ID: Yes
ID Index: 1
Normalization: Data Normalization ...
Target Column: No
Target Bin Limits:
Fingerprint Column Index: 0

(b)

Database Name: Gaia678
No. of Variables: 0
No. of Labels: 0
Label Names:
Label Column Index: 0
Title Row Index: Yes
Instances ID: Yes
ID Index: 1
Normalization: No Transforamtion
Target Column: Yes
Target Bin Limits: 100 - 90 - 80 - 50 - 0
Fingerprint Column Index: 3

(c)

Database Name: Rathbone197
No. of Variables: 69
No. of Labels: 0
Label Names:
Label Column Index: 0
Title Row Index: Yes
Instances ID: Yes
ID Index: 1
Normalization: Data Normalization (...
Target Column: Yes
Target Bin Limits: 100 - 80 - 70 - 50 - 0
Fingerprint Column Index: 72

Figure 2.3: (a) Configuration file for OilFlow dataset. (b) Configuration file for Gaia678 dataset (FingerPrint data only). (c) Configuration file for Rathbone197 dataset (continuous data as well as FingerPrint data).

### 2.1.2 Format of the Data File

The data file contains the dataset – the data to be processed by DVMS. This file should contain one record per row, and each field in a row should be separated by a comma. The file should be saved as comma separated values with the extension (.csv). There are exactly three possible row-types in a data file (Table 2.1): the title row, the column-type row, and a data row.

| **Row-Type** | **Description** |
|---|---|
| Title | A title (label) for each column.<br>If present must be row 1 (i.e. the first row).<br>This row is optional but recommended. |
| Column-Type | Entries specify type of each column.<br>Must always be present as either row 1 or 2<br>(depending on absence/presence of title row). |
| Data | A data row holds the data values for each column. |

Table 2.1: Row-types of the data file.

The title row contains name labels for each column. Its inclusion is recommended but it can be omitted by setting the configuration file. The column-type row must always be present as either row 1 or 2 depending on the absence/presence of the title row. A data row contains the data entries for each column.

The column-type identifiers (Table 2.2) are used in the fields of the column-type row. Columns can be present in any order. The ID column must occur exactly once. V column can occur any number of times. The LABEL, TARGET, and FINGERPRINT columns can occur either once or not at all.

It is often possible to classify a dataset by attaching a label to each point. Classification is used to colour data points – this helps to highlight any clustering present in a dataset. It also helps the user to see if the different classes are clearly separated. (This is a useful criterion for determining whether the training process is complete during the creation of hierarchical visualisation models.) Labels in the label column should be consecutive integers, starting from 1; these integers correspond to the class

| **Column-Type Identifier** | **Description** | **No. Occurrences** |
|---|---|---|
| ID | ID column.<br>Column of IDs (strings or integers). | 1 (Must occur once.) |
| V | Variable column.<br>Column of data values for a variable (reals). | Any no. |
| LABEL | Label column.<br>Column of label values (must be integers).<br>Values correspond to labels in config. file. | 0 or 1 |
| TARGET | Target column.<br>Column of target values (reals).<br>Used to produce NSC plots. | 0 or 1 |
| FINGERPRINT | Fingerprint column.<br>Column of fingerprints (FP-strings).<br>Used to produce NSC plots. | 0 or 1 |

Table 2.2: Column-type identifiers of the data file.

labels specified in the config. file.

Not all models can use a target and/or label column, and not all models can process descriptor and/or fingerprint data. (Note that it makes no sense to plot against a target column and a label column simultaneously – so a data file can contain either a target column or a label column but not both.) Table 2.3 summarises the input combinations available to each model. Attempting to input data not allowed by this table will generate a runtime error in DVMS.

| **Model** | **Target Col.?** | **Label Col.?** | **Descriptors?** | **Fingerprints?** |
|---|---|---|---|---|
| HGTM | No | Yes | Yes | No |
| GTM | Yes | Yes | Yes | No |
| GTM (log-version) | Yes | Yes | Yes | No |
| GTM-FS | Yes | Yes | Yes | No |
| GTM-FS (log-version) | Yes | Yes | Yes | No |
| GPLVM | Yes | Yes | Yes | No |
| LTM | Yes | Yes | No | Yes |
| NSC | Yes | Yes | Yes | Yes |

Table 2.3: Column and data combinations for each model.

## 2.2 Data Selection and Preprocessing

Data selection is an important process because if the variables that are chosen do not contain useful information, it will be impossible to get any insight from a visualisation model. This does not mean, however, that every variable should be included in a dataset. If too many variables are included, the important relationships/structures in the dataset may become obscured. Visualisation, by its very nature, will help the user explore a dataset, which in turn will help with variable selection.

DVMS, in its current form, requires all variables to be continuous variables; DVMS cannot currently process discrete variables.

Scaling-related weightings can occur if variables are measured on widely-different scales. Say, for example, one variable is in the range $-10,000.....10,000$, and another is in the range $-1.0.....1.0$, then the first variable will dominate results. A common way around this is to normalise the dataset so that each variable has a mean of 0 and a standard deviation of 1. Setting the Normalization property in the config. file allows the user to normalise a dataset. *Dataset normalisation should always be used, unless there is a good reason not to do so.*

Normalisation works well for most datasets, but problems can still arise if there are significant outliers (data values which are very different to the norm). This may prevent the model from being properly trained, but more often will lead to a plot with most points in one large cluster, and a small number of outliers a long way from the cluster. Visualisation will help the user identify outliers and exclude them from the analysis. *(The user must be very careful here: it is unexplained outlying data that can lead to new scientific theories!)* If HGTM is being used, the user will be able to create submodels to split the outliers from the main data.

DVMS needs every entry in the data matrix to have a value (i.e. no data fields should be empty/null). (It is actually possible to train GTM and HGTM models on datasets where some values are missing, and this is a possible future modification to DVMS.) If a value is missing, then the data point should be removed altogether, or the value should be replaced by the variable's mean value.

Before training a model the user will need to load both the config. file and the data file – this is done from the main GUI of DVMS. Once a model has been successfully trained the next steps are: save it (with a carefully chosen filename); load it back in; visualise it.

# Chapter 3

# Creating and Visualising Models

This chapter discusses different issues one should consider during the development of a visualisation model. There is more to developing a good-quality visualisation model than simply running a training code. 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-quality model are data selection, and data preprocessing. These are discussed in Section 2.2. Two other important steps, model training and model evaluation, are now discussed.

## 3.1 Training a Model

The purpose of training a model is to adjust the model's parameters (sometimes known as "weights") so that the model fits the data as well as possible. The quality of fit is measured using an error function: the smaller the value of the error function (which in some cases can be negative), the better the fit. Note that the error function for GTM and HGTM is quite different for that of NSC, and so error values cannot be compared between models.

A key consideration is how well the model fits the underlying generator of the data. A good-quality model will generalise well to new data. Generalisation can be measured by running the model with a test dataset i.e. evaluating the error function on a dataset different to the dataset used for training. So when training a model, the user will need at least two datasets: one for training (parameter adjustment) and one for testing (model evaluation). To train a model, a graphical interface (as shown in Figure 3.1) is provided that assists the user by selecting a training algorithm from the drop down list provided.

### 3.1.1 High-Level Network-Architecture Parameters

When training a model, there are certain high-level network-architecture parameters that the user needs to set for certain algorithms. If the training algorithm selected is the PCA, it does not have any high-level parameters, and is created by simply using the "Train Model" button (see Figure 3.2). Adjustable parameter settings for training a model can be seen in Figure 3.4 (GTM, GTM (log-space version), GTM-FS, GTM-FS (log-space version), LTM and HGTM) and Figure 3.3 (NSC). GTM, GTM-FS, GTM (with log based Gaussian), LTM and HGTM all have two main high-level parameters: the number of node centres (Gaussians), and number of RBF centres. The main high-level parameter for NSC is the number of RBF centres.

Model complexity is a function of the size and structure of the model. The greater the number of RBF centres and node centres, the more flexible/complex the model. If the numbers are too small, then the model will be too simple and will have a large error on the training data. If the numbers are too large, then the model will have a low error on the training data, but a large error on new data because the model is too specific to the training data – a phenomenon known as *overfitting* or, less commonly, *overtraining*. If overfitting is present, it usually becomes apparent when the number of points in the test dataset is much less than (say 25%) the number of points in the training dataset.

One way to determine a good value for the high-level parameters is to train several models with a range of values and compare the generalised performance. A good model will be as simple as possible, and will fit the data well. Here are some notes on the high-level parameters for the various models:

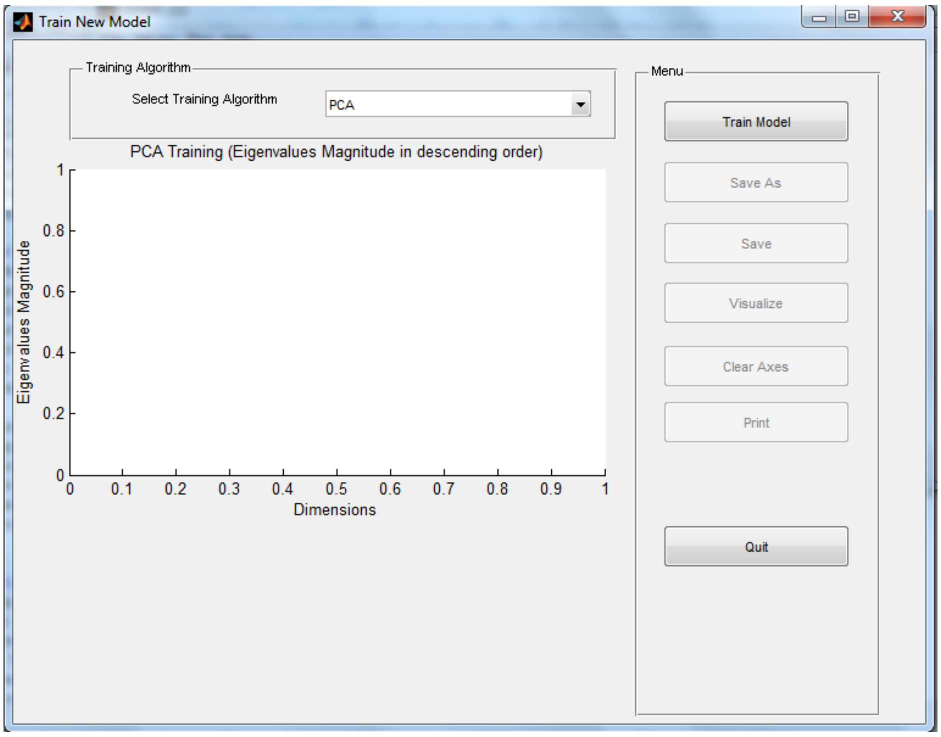


Figure 3.1: Train new model graphical interface.

**GTM**: GTM, as a model, can be interpreted as being a 2D rubber sheet in data-space: spheres placed on the sheet capture that fact that the data lies near to, but not exactly on, the sheet. The two main high-level parameters are: (1) number of node centres; (2) number of RBF centres. In (1) the Gaussians are the spheres: the more spheres, the better the data can be modelled. However, the number of training iterations is proportional to the number of Gaussians, so using too many Gaussians can make training very slow. Overfitting, whilst possible with GTM, is less likely to happen than with the other models used by DVMS. In (2) the number of RBF centres governs the complexity of the map from the computer screen to the data-space – effectively the amount of stretch and curvature of the rubber sheet. The larger the RBF value, the more complex the map. Different variations of the GTM such as GTM with log based Gaussian model is implemented to support the high-dimensional data space and use same high-level parameters as for GTM. GTM-FS also uses same high-level parameters as for GTM.

**LTM**: Use high-level parameters as for GTM.

**HGTM**: HGTM consists of a tree of GTM models, the high-level parameters for each GTM model need to be set as each is trained. In addition, the user will need to choose 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 plots fit the data. This is further discussed in Section 3.1.2.

**NSC**: Number of hidden units (RBF centres): the larger the number, the more complex the projection function can be.

The user will need to choose the maximum number of training iterations the training algorithm should run for. The method of determining when to stop training the single models (GTM, GTM-FS, GTM (with log based Gaussian), LTM and NSC) is straightforward: each model should be trained until the error value has converged. During training, DVMS will plot a logarithmic graph of the error values. Once the error curve has reached a flat level (Figure 3.4), no more training is required. *If the error curve has not reached a flat level when the maximum number of iterations has been reached, then the maximum number of iterations should be increased and the model retrained.* The training of a HGTM model is recursive: once the top-level GTM has been trained, every leaf node in the tree

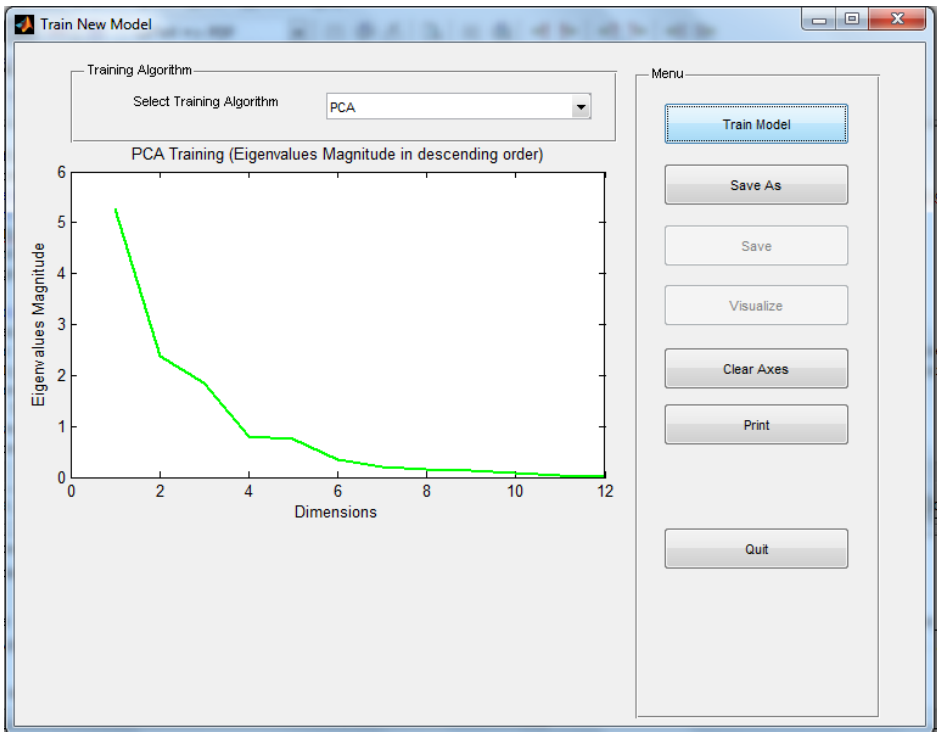


Figure 3.2: Train PCA window.

can be extended with child models. The next section provides more information on training HGTM models.

## 3.1.2 Interactive HGTM Training

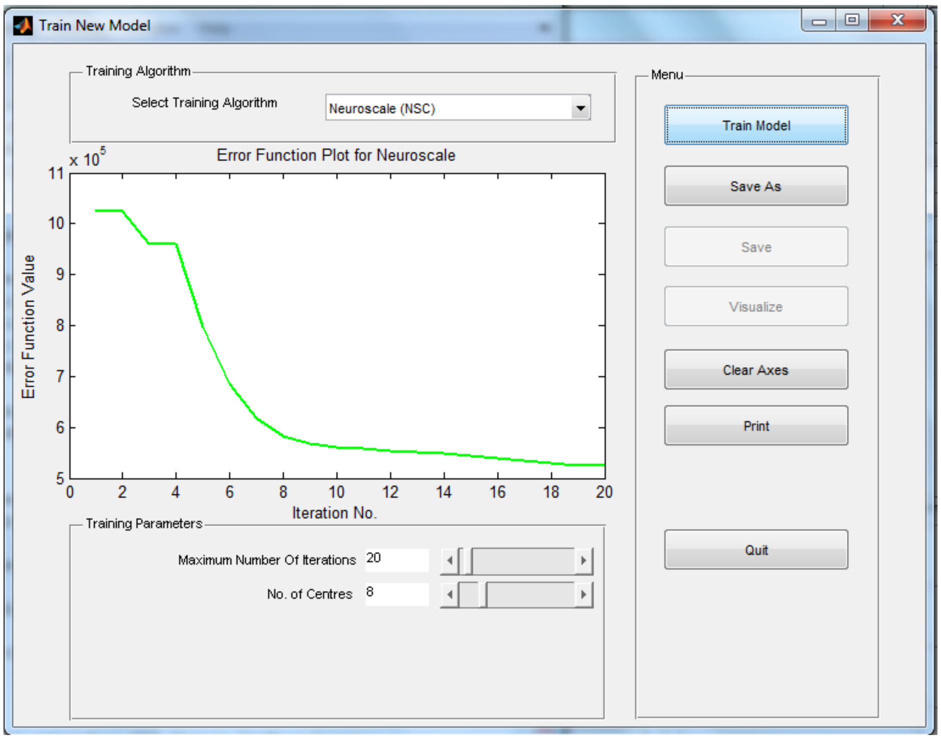
How are child GTM plots added? When should child GTM plots be added? When should their addition be stopped? These are the three questions that should be asked when training a HGTM model, and each is now answered.

**How are child GTM plots added?** To add the child plots, "HGTM interactive mode" button must be pressed first on the latent space window. Thereafter, the user selects points in the latent-space that correspond to centres of the subregions of interest. The points are then transformed via a map $f$ to the data-space. Then subregions are formed using Voroni compartments [1]. Adding child GTMs in DVMS is very simple. Click on the parent GTM plot to select centres for the submodels, and right click to end the selection of submodels. Relevant instructions are provided on screen during training.

**When should child GTM plots be added?** GTM models the data as a curved and stretched 2D sheet. However, if the data points at a leaf in the tree do not lie close to such a surface, then the plot will be misleading. The successful addition of child GTMs requires the data to be partitioned so that it lies locally-close to a 2D sheet. The user should add a child GTM to a leaf model if:

1. The plot is cluttered with too many points and separate clusters cannot be seen.
2. With the help of the curvature plots, the user decides 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 simpler segments.
3. The MF plot shows that some areas of the map are being stretched a long way. Putting child models in regions of high data-density will create child plots that are less stretched.

**When should the addition of child GTM plots be stopped?** No futher models should be added when the plots give enough information. One way of deciding this is by comparing leaf nodes to their parents; if they look very similar, no further information is available. If the data is simply



Select Training Algorithm: Neuroscale (NSC)

Maximum Number Of Iterations: 20

No. of Centres: 8

Figure 3.3: Training Neuroscale (NSC) window.

being visualised, and no predictive modelling is being carried out, 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 by a graph similar to the error graph of Figure 3.4. The training error should end with a flat level, which means that the learning algorithm is nearing the minimum of the learning cost function. At this stage, if required, parameters can be changed and the model retrained.

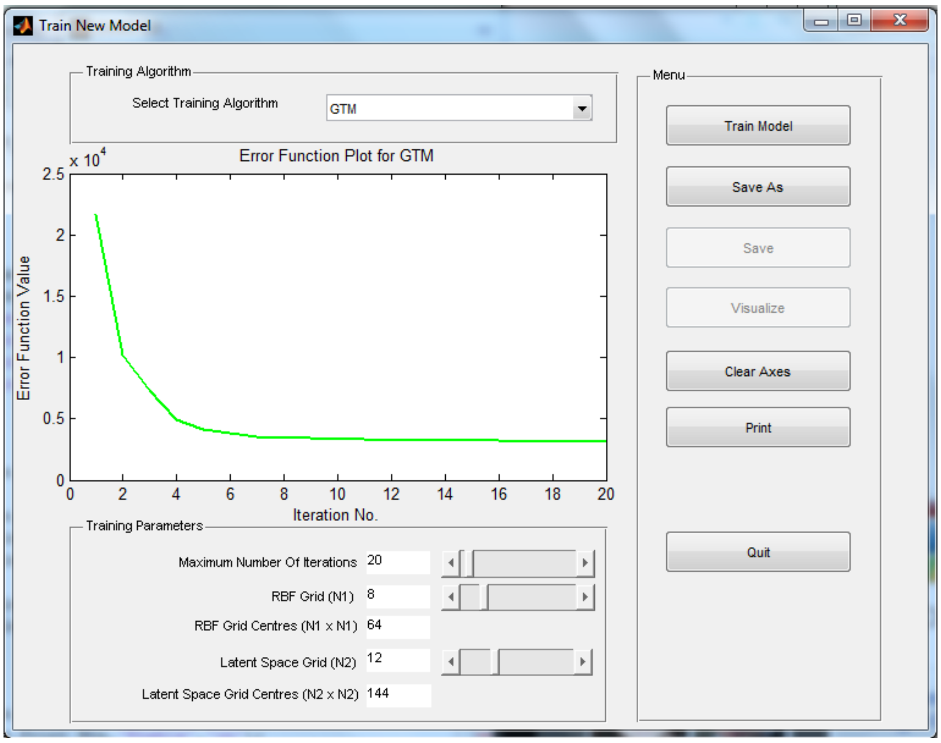
## 3.2 Visualising Trained Models

Models, trained as explained in Section 3.1, can be loaded to visualise the data. Existing models can be loaded by simply pressing the "Visualize" button on DVMS's training window or visualizing the test dataset by pressing "Visualize" button using primary GUI that brings the "Test Visualize" window appeared. Previously trained model using different algorithms can be loaded by selecting the algorithm type from the drop down list and clicking on the "Load Trained Model" and then selecting and pressing the "Visualize" button . Pressing the "Visualise" button either on the "training window" or "test visualize window" causes the model to be plotted in latent-space.

### 3.2.1 Model Visualisation (Non-HGTM)

Figure 3.5 gives the interface for PCA, GTM, GTM-FS, GTM (with log based Gaussian) and LTM visualisations. By using the interface the user can explore nearest points in data-space, and can even save the latent-space points as a (.csv) file.

It is very useful to be able to relate the visualisation of latent-space to that of data-space. A facility to do this is provided by the "Local Parallel Coordinates" (LPC) frame available on the latent-space visualisation interface (Figure 3.5). The LPC container frame has slider to select the number of neighbouring points, option button for single or multiple LPC windows appeared for the selected point/points on the latent space visualization. After the "Click to Activate LPC Mode" button has been pressed, the user can click on any point/points in the latent-space to produce a chart of the



Select Training Algorithm: GTM

Maximum Number Of Iterations: 20
RBF Grid (N1): 8
RBF Grid Centres (N1 x N1): 64
Latent Space Grid (N2): 12
Latent Space Grid Centres (N2 x N2): 144

Figure 3.4: Training GTM window.

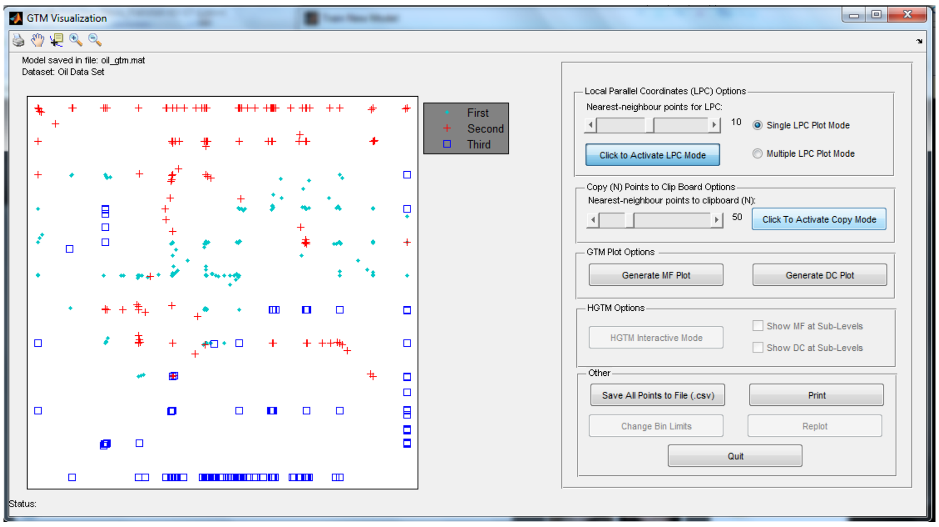
point's data-space variables. The LPC plots generated for the two selected points on latent space plot (see Figure 3.6(a)) with 10 neighbours, as shown in Figure 4.2(b), are interactive plots. The user can select particular IDs by clicking on the corresponding line or ID. If the column-title row was given in the data file, right clicking the ID will give a list of property names with data-space values for the clicked ID. If the column-title row was not given, the data-space values will be displayed but without any property names. Figure 4.2(b) shows an example of the LPC window. The window can be removed by right-clicking in the latent-space plot.

One of the main advantages of using GTM-based models is that it is possible to analytically calculate the magnification factors (MFs) [2] and the directional curvatures (DCs) [11] of the GTM projection manifold. MFs of a GTM projection manifold, $\Omega$, are calculated as the determinant of the Jacobian of the GTM map $f$ [2]. MF plots are used to observe the amount of stretching in a GTM manifold at different parts of the latent-space, which helps with data-space interpretation, outlier detection, and cluster separation. Nabney et al. [11] derived a closed-form formula for the DCs of the GTM projection manifold ($\Omega$) for a latent-space point $\mathbf{x} \in H$ and a directional vector $\mathbf{h} \in H$. DC plots illustrate 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. If folding in the manifold is particularly high, it is possible for clustered points in the data-space to appear disparate on the projection manifold. This clustering in the data-space can be spotted as a strong curvature band on the corresponding DC plot.

The MF is represented by colour shading in the projection manifold (Figure 3.7(a)). The lighter the colour, the more the stretch in the projection manifold. The direction of folding in the projection manifold plot is given by a fine line for each part of the projection manifold in the DC plots (Figure 3.7(b)). The line length and the shade of the background colour represents the magnitude of folding. The longer the line, and the lighter the background colour, the greater the folding.

### 3.2.2 Model Visualisation (HGTM)

A HGTM data model is visualised as a hierarchy of GTMs as shown in Figure 3.8(a) and Figure 3.8(b). The interface provides an "Options" menu which can be used to display MFs (Figure 3.8(c)),


Model saved in file: oil_gtm.mat
Dataset: Oil Data Set

Nearest-neighbour points for LPC: 10
[✓ Single LPC Plot Mode] Multiple LPC Plot Mode

Nearest-neighbour points to clipboard (N): 50

Status:

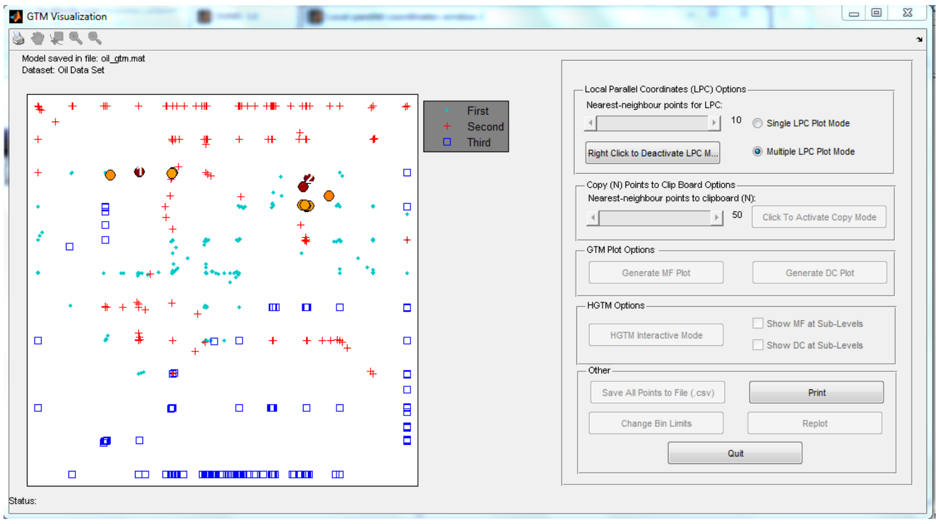
Figure 3.5: GTM Visualization space.

and compute and display DCs (Figure 3.8(d)).

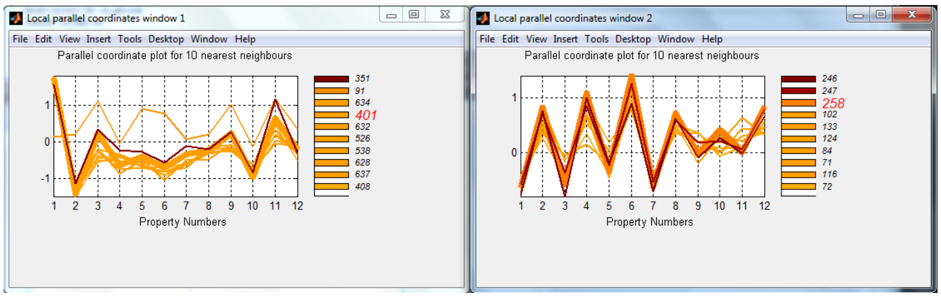
### 3.2.3 Model Evaluation

There are two main ways of evaluating a model. One is to assign a parameter to measure how well the model fits the data ("goodness of fit"). The other is to make a subjective judgement of the quality of a plot.

Choosing a parameter to measure goodness of fit is not a problem: the error can be calculated for a test dataset. In a good-quality model, the error for the test dataset will be similar to the error for the training dataset. Assessing the quality of a plot is largely subjective, but there are some objective measures that can help. In a GTM plot, MF and DC subplots can help measure plot quality. Some experimentation with the high-level network-architecture parameters, *particuarly the maximum number of training iterations*, may help refine a model. A good-quality plot will be easy to interpret. The ideal plot will reveal some hitherto unknown structures in a dataset – providing new insights, perhaps, for a piece of research.

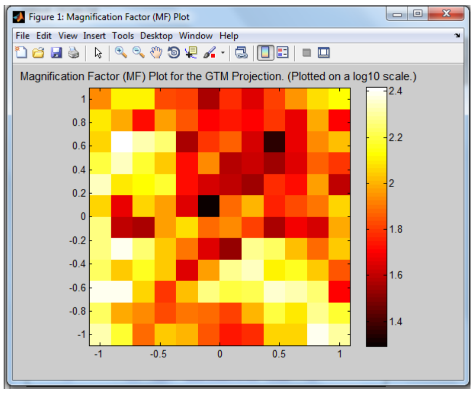

GTM Visualization
Model saved in file: oil_gtm.mat
Dataset: Oil Data Set
First
Second
Third
Local Parallel Coordinates (LPC) Options
Nearest-neighbour points for LPC:
10
Single LPC Plot Mode
Multiple LPC Plot Mode
Right Click to Deactivate LPC M...
Copy (N) Points to Clip Board Options
Nearest-neighbour points to clipboard (N):
50
Click To Activate Copy Mode
GTM Plot Options
Generate MF Plot
Generate DC Plot
HGTM Options
HGTM Interactive Mode
Show MF at Sub-Levels
Show DC at Sub-Levels
Other
Save All Points to File (.csv)
Print
Change Bin Limits
Replot
Quit
Status:

(a)

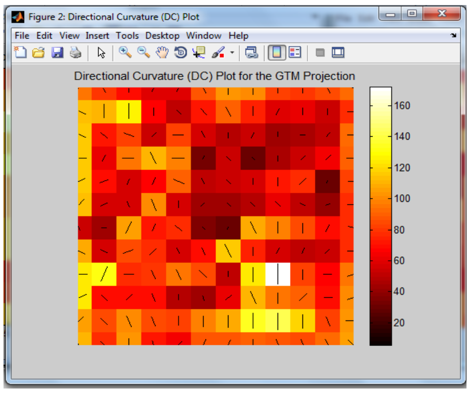

Local parallel coordinates window 1
File Edit View Insert Tools Desktop Window Help
Parallel coordinate plot for 10 nearest neighbours
Property Numbers
351
91
634
401
632
526
538
628
637
408
Local parallel coordinates window 2
Parallel coordinate plot for 10 nearest neighbours
Property Numbers
246
247
258
102
133
124
84
71
116
72

(b)

Figure 3.6: GTM Visualization (a) 2D Space Visualization. (b) LPC windows.

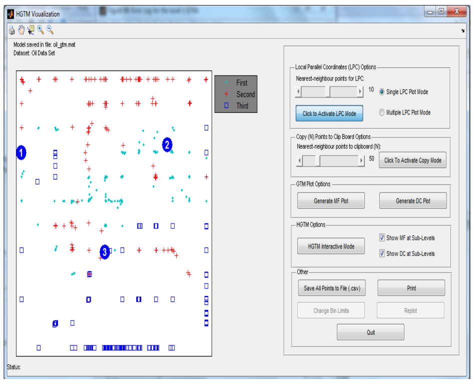


(a)

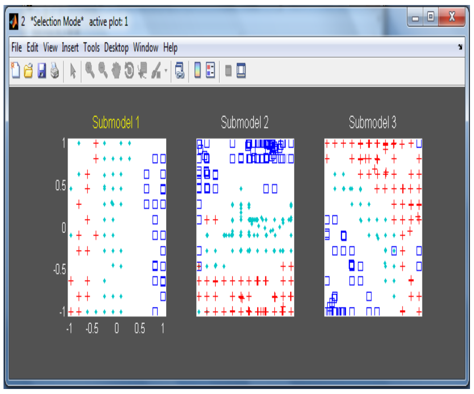


(b)

Figure 3.7: (a) Magnification Factors for GTM. (b) Directional Curvature for GTM.



(a)


2 *Selection Mode* active plot: 1
File Edit View Insert Tools Desktop Window Help
Submodel 1
Submodel 2
Submodel 3
1
0.5
0
-0.5
-1
-1 -0.5 0 0.5 1

(b)

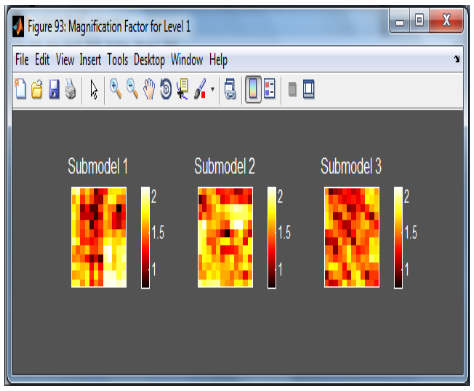


Figure 93: Magnification Factor for Level 1
File Edit View Insert Tools Desktop Window Help
Submodel 1
Submodel 2
Submodel 3
2
1.5
1

(c)

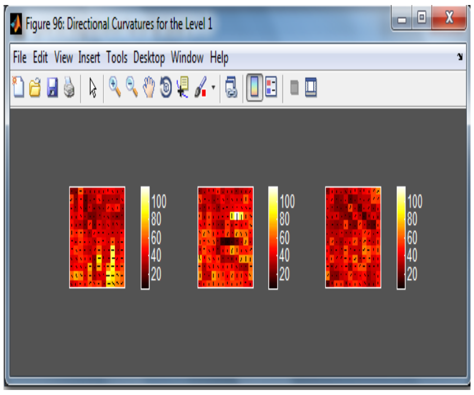


Figure 96: Directional Curvatures for the Level 1
File Edit View Insert Tools Desktop Window Help
100
80
60
40
20

(d)

Figure 3.8: (a) HGTM Mode Latent Space Visualization (b) HGTM childs. (c) Magnification Factors for HGTM childs. (d) Directional Curvature for HGTM childs.

Chapter: Plotting MDL166 fingerprints

# Chapter 4

# Plotting MDL166 Fingerprints with NeuroScale

A molecular fingerprint is a one-zero bit-string describing the presence/absence of molecular features within a single molecule. This chapter does not describe molecular fingerprint theory – for some theory the reader should consult a textbook on chemoinformatics such as [6]. Given a dataset of molecules, a fingerprint can be generated for each molecule, and these fingerprints can be plotted using NeuroScale (NSC). Several types of fingerprint are in common use; DVMS can process MDL166 fingerprints, which as their name indicates, are 166 bits in length. This chapter describes how to produce NSC-based plots of MDL166 fingerprints in DVMS.

## 4.1 Fingerprint Generation with Pipeline Pilot

The first step in producing an NSC-based fingerprint plot is to generate some fingerprints. This is done by feeding a dataset of molecules into *Pipeline Pilot*, a high-quality program from Accelrys Software Inc. (The *Molecular Fingerprint* component within Pipeline Pilot produces MDL166 fingerprints.) Instructions on the use of Pipeline Pilot are not included here (Pipeline Pilot has very good help pages). The column of fingerprints output by Pipeline Pilot will need to be merged with a DVMS data file using a simple mini-tool called *FPMerge*. FPMerge is available by e-mail request from the NCRG, Aston University.

## 4.2 Configuration and Data Files for Fingerprint Plotting

No special config. or data files are required for fingerprint plotting; standard DVMS config. and data files are used. An example config. file for generating a fingerprint plot is shown in Figure 2.3(b) and Figure 2.3(c).

## 4.3 The Distance Matrix Used for Fingerprint Plotting

An NSC-based fingerprint-model works by generating a matrix of the distances between all pairs of molecules in the dataset. The distance matrix is computed as follows. Let $x$ and $y$ be any two molecules in the dataset. Define $d$ the distance between these two molecules to be:

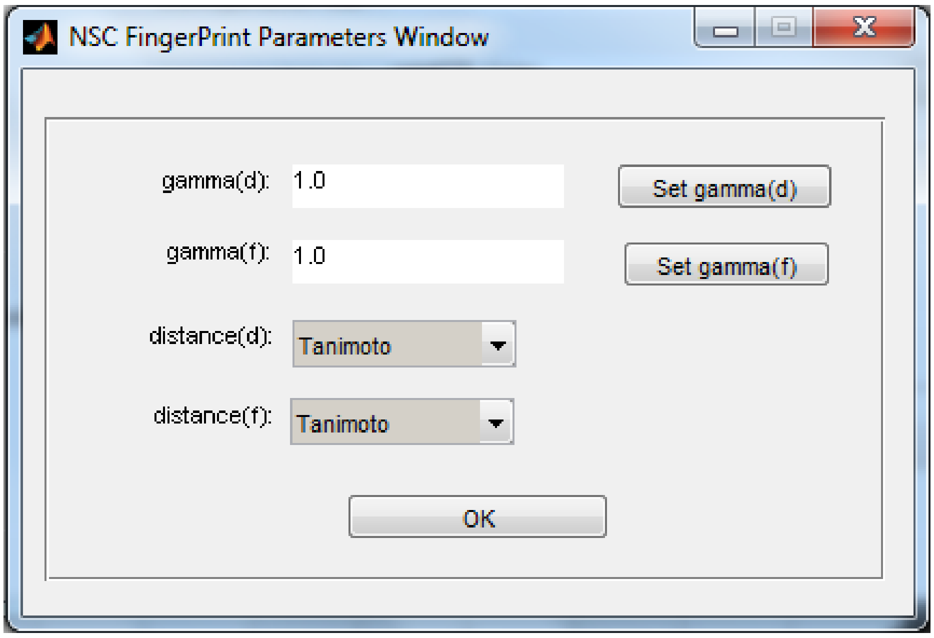
$$d(x, y) = \gamma_d d_d(x_1, y_1) + \gamma_f d_f(x_2, y_2).$$

Where:

- $x_1$ is the vector of descriptor data for molecule $x$
- $y_1$ is the vector of descriptor data for molecule $y$
- $x_2$ is the MDL166 bit-string for molecule $x$

- $y_2$ is the MDL166 bit-string for molecule $y$
- $d_d$ is the descriptor data distance-function
- $d_f$ is the fingerprint data distance-function
- $\gamma_d$ is an arbitrary multiplying-factor
- $\gamma_f$ is an arbitrary multiplying-factor

The vectors of descriptor data and fingerprint data are derived from the dataset. If no descriptor data is present, $d_d = 0$, and the NSC plot will be of the fingerprint data only. The distance functions $d_d$ and $d_f$ are chosen by the user from menu of possible functions (possible functions include Tanimoto, Euclidean, and Dixon-Koehler). The multiplying factors $\gamma_d$ and $\gamma_f$ are arbitrarily set by the user to emphasize either $d_d$ or $d_f$. (If no emphasis is required then both of these factors should be set to 1.) The distance matrix is assembled by computing $d(x, y)$ for all molecules. Once all values in the distance matrix have been computed, the matrix is fed into an NSC model and a plot generated. The user can set the distance functions, $\gamma_d$, and $\gamma_f$ on DVMS's primary GUI by clicking on the "Set NSC FP parameters". NSC Fingerprint parameters window is shown in Figure 4.1.

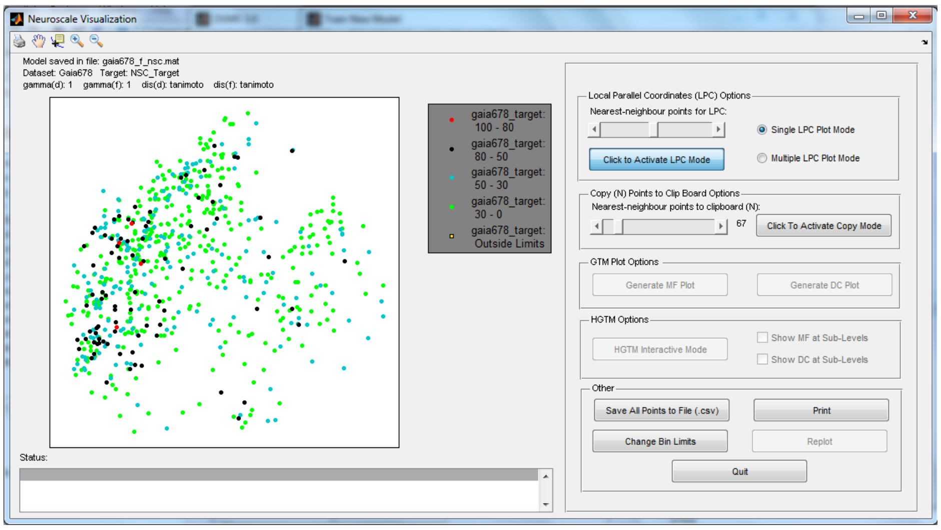


gamma(d): 1.0
gamma(f): 1.0
distance(d): Tanimoto
distance(f): Tanimoto

Figure 4.1: Train new model graphical interface.

## 4.4 An Example NSC-Based Fingerprint Plot

An example NSC-based plot of a fingerprint-containing dataset (i.e. Gaia678 dataset) is in Figure 4.2(a). Another example NSC-based plot of combined fingerprint and descriptors dataset (i.e. rathbone197) is in Figure ??. (These plots are best viewed/printed in colour.) Clustering in these plots is inconclusive – clustering may have been observed had more points been available. As can be seen on the GUI, several options exist for the user to choose from including: Change Bin-Limits, Replot, and Local Parallel Coordinates (LPC) (active if there are descriptors in the data).



Model saved in file: gaia678_f_nsc.mat
Dataset: Gaia678 Target: NSC_Target
gamma(d): 1 gamma(f): 1 dis(d): tanimoto dis(f): tanimoto

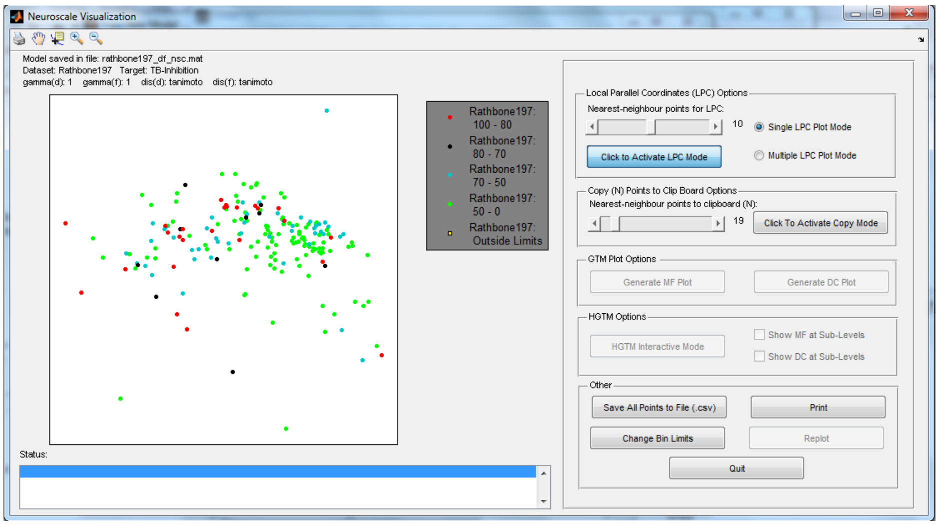
| | |
|---|---|
| ● | gaia678_target: 100 - 80 |
| ● | gaia678_target: 80 - 50 |
| ● | gaia678_target: 50 - 30 |
| ● | gaia678_target: 30 - 0 |
| □ | gaia678_target: Outside Limits |

Nearest-neighbour points to clipboard (N): 67

[✓ Single LPC Plot Mode] Multiple LPC Plot Mode

Neuroscale Visualization
Local Parallel Coordinates (LPC) Options
Nearest-neighbour points for LPC:
Click to Activate LPC Mode
Copy (N) Points to Clip Board Options
Click To Activate Copy Mode
GTM Plot Options
Generate MF Plot
Generate DC Plot
HGTM Options
HGTM Interactive Mode
Show MF at Sub-Levels
Show DC at Sub-Levels
Other
Save All Points to File (.csv)
Print
Change Bin Limits
Replot
Quit
Status:

(a)



Model saved in file: rathbone197_df_nsc.mat
Dataset: Rathbone197 Target: TB-Inhibition
gamma(d): 1 gamma(f): 1 dis(d): tanimoto dis(f): tanimoto

| | |
|---|---|
| ● | Rathbone197: 100 - 80 |
| ● | Rathbone197: 80 - 70 |
| ● | Rathbone197: 70 - 50 |
| ● | Rathbone197: 50 - 0 |
| □ | Rathbone197: Outside Limits |

Nearest-neighbour points for LPC: 10

Nearest-neighbour points to clipboard (N): 19

[✓ Single LPC Plot Mode] Multiple LPC Plot Mode

(b)

Figure 4.2: NSC visualization. (a) FingerPrints data only i.e. Gaia678. (b) FingerPrints and descriptor dataset combined i.e. Rathbone197.

# Chapter 5

# Plotting MDL166 Fingerprints with the Latent Trait Model

This chapter is similar to Chapter 4 on plotting MDL166 fingerprints with NeuroScale. The Latent Trait Model enables the user to plot datasets of fingerprints (one-zero bit strings). Only fingerprints can be plotted – the dataset cannot contain any non-fingerprint (i.e. descriptor) data. This chapter gives a brief introduction to the Latent Trait Model, an example config. file, and an example plot.

## 5.1 Introduction to the Latent Trait Model

The main reference on the Latent Trait Model (LTM) is a 2001 paper by Kabán and Girolami [4]. DVMS contains NETLAB code which implements the theory given in this paper. This chapter does not give any theory on LTM – for theory the reader is referred to the paper.

LTM enables the user to plot a dataset of MDL166 fingerprints (against activity value or class). It is well suited to plotting fingerprints as it was originally designed to model discrete data, and can be regarded, informally, as a "sort of discrete GTM". LTM works by using a Bernoulli noise model rather than a Gaussian noise model as is used in GTM. To use LTM, the user will first need to generate a dataset of MDL166 fingerprints as described in Section 4.1.

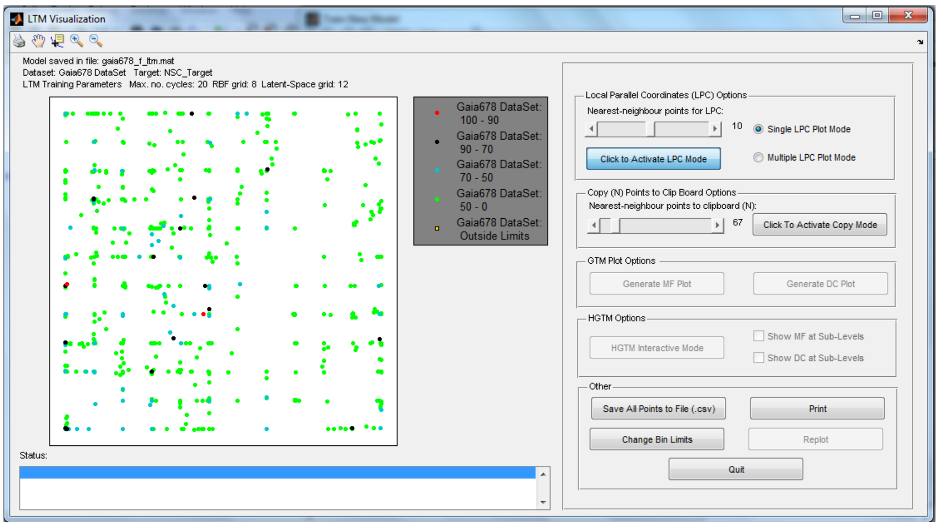
## 5.2 Configuration and Data Files for LTM

*Note: normalisation must be turned off when using LTM.* The LTM algorithm has been designed to process discrete data directly – in the case of fingerprints this data is one-zero columns. An example config. file for LTM is given in Figure 2.3(b):

The data file will typically consist of only three columns: an ID column, a target (or label) column, and a fingerprint column. The respective column-type identifiers for these columns are: ID, TARGET (or LABEL), FINGERPRINT. The best way to understand the format of the config. and data files is to inspect the example files supplied with DVMS.

## 5.3 An Example LTM-Based Fingerprint Plot

An example of a dataset of fingerprints plotted with LTM is in Figure 5.1. (The figure is best viewed/printed in colour.) Clustering in the plot is inconclusive – some clustering of the most active molecules is present, but this could be purely coincidental. The plot contains 678 molecules – not enough to identify much in the way of clustering.



Model saved in file: gaia678_f_ltm.mat
Dataset: Gaia678 DataSet Target: NSC_Target
LTM Training Parameters Max. no. cycles: 20 RBF grid: 8 Latent-Space grid: 12

Local Parallel Coordinates (LPC) Options
Nearest-neighbour points for LPC: 10
[✓ Single LPC Plot Mode] Multiple LPC Plot Mode

Copy (N) Points to Clip Board Options
Nearest-neighbour points to clipboard (N): 67

Show MF at Sub-Levels
Show DC at Sub-Levels

Figure 5.1: An LTM-based fingerprint plot visualization (A dataset of Gaia678).

# Appendix A

# Configuration File (Using Text Editor)

2.2 To understand the format of the configuration files is to inspect the example files supplied with DVMS – the reader is advised to look at these files. Configuration files (extension .cfg) can be read and writing with any text editor.

## A.1 Format of the Configuration File

The first thing to do when building a new model is to write a DVMS configuration file. This file, commonly called the *config. file*, is mostly metadata about the dataset. A config. file can be created with more or less any text editor, and must be saved with the file extension (.cfg). An example config. file is given below. Blank lines are ignored by DVMS. Comments begin with a hash (#) and must start on a new line.

```
# Example No. 1 DVMS Configuration File
# Shahzad Mumtaz, 1 June 2011
# Neural Computing Research Group (NCRG), Aston University, UK

# Indicate start of config. file
$CONFIG_START

# Second row of data file contains column-name strings
$DATASET_NAME
OilFlow
$NO_VARIABLES
12
$LABEL_NAMES
Homogeneous
Annular
Laminar
$LABELS_COLUMN_INDEX
1
$TITLE_ROW_INDEX
1
$ID_COLUMN_INDEXSTATUS
1
$ID_COLUMN_INDEX
1
$NORMALIZATION
1
CONFIG_END
```

The value for a tag should immediately follow the tag itself. Tags can take 0, 1, or more values. In this example some of the tags are redundant as they duplicate tag default values (the user may wish, however, to include them to emphasise values as is done here). The complete set of tags available in DVMS, together with their default values, are in Table A.1 below. (You are likely to find this table quite handy when writing config. files.) Another example config. file is given below. This file is for a dataset that contains a target column and no label column.

```
# Example No. 2 DVMS Configuration File
# Shahzad Mumtaz, 1 June 2011
# Neural Computing Research Group (NCRG), Aston University, UK

# Indicate start of config. file
$CONFIG_START
$DATASET_NAME
Rathbone197
$NO_VARIABLES
69
$TITLE_ROW_INDEX
1
$ID_COLUMN_INDEXSTATUS
1
$ID_COLUMN_INDEX
1
$NORMALIZATION
1
$TARGET_COULUMN_INDEX
1
$BIN_LIMITS
100
90
80
70
50
0
$FINGERPRINT_COLUMN_INDEX
72
CONFIG_END
```

In this example the tags `$NORMALISATION` and `$TITLE_ROW` have been omitted as they do not need to be changed from their default values. Note that the `$BIN_LIMITS` tag must followed by exactly 5 values, each on a consecutive line.

| **Tag** | **Default Value** | **Possible Values and Description** |
|---|---|---|
| `$CONFIG_START` | N/A | Must be first tag in the config. file. |
| `$CONFIG_END` | N/A | Must be last tag in the config file. |
| `$DATASET_NAME` | DATASET_NAME | A string holding the dataset's name. |
| `$NO_LABELS` | 0 | Numeric value to represent the number of variables. |
| `$LABEL_NAMES` | N/A | Label name strings for each data class. Used in conjunction with `$NO_LABELS`. |
| `$LABELS_COLUMN_INDEX` | 0 | Index of the label's column. |
| `$TITLE_ROW` | 1 | 0: First row of data file does not contain column name strings. |
| | | 1: First row of data file does contain column name strings. |
| `$ID_COLUMN_INDEXSTATUS` | 1 | 0: First row of data file does not contain ID column. |
| | | 1: First row of data file does contain ID column. |
| `$ID_COLUMN_INDEX` | 0 | Index of the ID's column. |
| `$NORMALISATION` | 1 | 0: Apply no transformation to dataset. |
| | | 1: Apply normalisation (mean = 1, var = 0). |
| | | 2: Apply data whitening. |
| `$TARGET_COLUMN_INDEX` | 1 | 0: Data does not contain target column. |
| | | 1: Data does contain target column. |
| `$BIN_LIMITS` | [100; 90; 80; 50; 0] | Bin-limits used for NSC plotting when fingerprints present in dataset. Must have exactly 5 values with each value on a separate line. |
| `$FINGERPRINT_COLUMN _INDEX` | 1 | 0: Data does not contain finger print column. |
| | | N: "N" represent a number as index of finger print column. |

Table A.1: The complete set of config. file tags.

# Appendix B

# Glossary of Acronyms

This appendix gives brief and informal definitions of the main acronyms used in this user guide.

**DC:** Directional Curvature
A parameter that measures curvature in a GTM manifold.

**DVMS:** Data Visualisation and Modelling System
DVMS is the subject of this guide. A program developed in the NCRG for visualising and modelling datasets.

**GTM:** Generative Topographic Mapping
A non-linear mapping of data from n-space to a space of lower dimensions, usually 2-space (for plotting). GTM is mathematically more complex than PCA and NSC. It was first published in 1998 and is one of the most advanced dimension-reduction techniques available.

**GTM-FS:** Generative Topographic Mapping with Feature Saliencies
Similar to GTM, except that feature saliencies are generated in addition to a GTM model.

**GUI:** Graphical User Interface
A graphics-based user interface – essential for visualisation.

**HGTM:** Hierarchical Generative Topographic Mapping
Similar to GTM, except that data is explored using hierarchies of GTM plots.

**LTM:** Latent Trait Model
LTM was designed to model datasets of discrete data – in the case of DVMS, molecular fingerprints (one-zero bit strings) are modelled. LTM is similar to GTM except that a different noise model is used. GTM uses a Gaussian noise model; the LTM implemented in DVMS uses a Bernoulli noise model. Other possible LTM noise models (not implemented in DVMS) include Poisson and multinomial.

**LPC:** Local Parallel Coordinates
Let $\mathbf{p}$ be a point in latent-space. Then the LPC of $\mathbf{p}$ is the set of its data-space variables.

**MF:** Magnification Factor
A parameter that measures stretching in a GTM manifold.

**NCRG:** Neural Computing Research Group
A research group based at Aston University, Birmingham, UK.

**NN:** Neural Network
NN is sometimes used as an abbreviation for Neural Network.

**NSC:** NeuroScale
A non-linear mapping of data from n-space to a space of lower dimensions, usually 2-space (for plotting). First published in 1997.

|  |  |
|---|---|
| **PCA:** | Principal Component Analysis<br>A linear mapping of data from n-space to a space of lower dimensions, usually 2-space (for plotting). PCA is one of the simplest dimension-reduction techniques available. It was invented in 1901 and is still widely used today. |
| **PhiVis:** | Probabalistic Hierarchical Interactive Visualisation<br>A Matlab-based software package for visualisation. PhiVis was developed in the NCRG in the late-1990s. |
| **RBF:** | Radial Basis Function<br>Radial basis functions are used in RBF-based neural networks. An RBF $\phi$ is a real-valued function whose value at a point $\mathbf{x}$ depends only on the magnitude of $\mathbf{x}$:<br>$$\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|).$$ |

# Bibliography

[1] F. Aurenhammer. Voronoi diagrams – survey of a fundamental geometric data structure. *ACM Computing Surveys*, 3:345–405, 1991.

[2] C. M. Bishop, M. Svensén, et al. Magnification factors for the GTM algorithm. *Proceedings of the 5th IEE International Conference on Artificial Neural Networks*, pages 64–69, 1997.

[3] C. M. Bishop, M. Svensén, et al. GTM: The generative topographic mapping. *Neural Computation*, 10:215–234, 1998.

[4] A. Kabán and M. Girolami. A combined latent class and trait model for the analysis and visualization of discrete data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):859–872, 2001.

[5] Neil D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *In NIPS*, page 2004, 2004.

[6] A. R. Leach and V. J. Gillet. *An Introduction to Chemoinformatics.* Springer, The Netherlands, 2007.

[7] D. Lowe and M. E. Tipping. NeuroScale: Novel topographic feature extraction with radial basis function networks. *Advances in Neural Information Processing Systems*, 9:543–549, 1997.

[8] I. T. Nabney and A. Kabán. Semisupervised learning of heirarchical latent trait models for data visualization. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):384–400, 2005.

[9] I. T. Nabney and D. M. Maniyar. Visual data mining using principled projection algorithms and information visualization techniques. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 643–648, 2006.

[10] I. T. Nabney, D. M. Maniyar, et al. Data visualization during the early stages of drug discovery. *Journal of Chemical Information and Modeling*, 46(4):1806–1818, 2006.

[11] I. T. Nabney, Y. Sun, et al. Using directional curvatures to visualize folding patterns of the GTM projection manifolds. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 421–428, 2001.

[12] 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):336–343, 1996.