FEATURE SELECTION TOOLBOX

– A BRIEF MANUAL –

(for all versions, including 0.5x.x)

Petr Somol

e-mail: somol@utia.cas.cz,
ps326@cam.ac.uk

The work on the software originates at:
The Department of Pattern Recognition,
Institute of Information Theory and Automation,
Pod vodárenskou věží 4, 182 08 Prague 8, Czech Republic

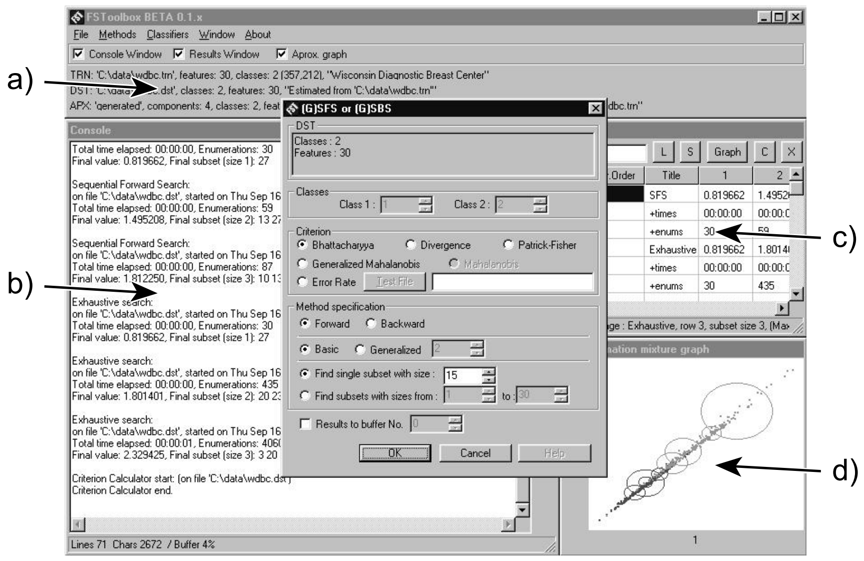
Contents

Chapter 1

Feature Selection Toolbox Overview

This brief manual describes the Feature Selection Toolbox (FST) software package. The FST scope is basically feature selection in statistical pattern recognition. That means the software incorporates a number of different algorithms doing in different ways often the same thing – selecting features based on input data and certain probabilistic functions.

The feature selection problem is more or less a special case of a much broader problem of subset selection. Suppose a large set of (D) items is given from which we need to find a small (d) subset being optimal in a certain sense. In statistical subset selection being optimal usually means being most suitable for classification or data approximation. Although it may not be apparent, the subset selection problem may become prohibitive because of its computational complexity. It is not possible to rank the items in our set simply according to their individual properties and select only the best. The item properties may depend strongly on the other items and a subset of individually "bad" features may prove to be rather "good". Because of this uncertainty the only apparent way of searching for optimal subsets is – simply to evaluate all the possible item combinations. But this is exactly the prohibiting point. Testing all subsets is a combinatorial problem that requires an exponential amount of computational time. In other words, even with relatively small sets our lifes are too short to wait for results. Luckily a lot of more or less advanced alternative algorithms exist to solve the problem in a reasonable way. As the feature selection knowledge-base evolves and broadens, new algorithms and approaches appear and the potential user may quickly become lost when choosing an appropriate way to solve his subset selection problem. The FST software package aims to offering a wide range of different feature selection algorithms, both to allow their comparison for teaching/learning purposes as well as finding solutions of practical problems. As the main FST domain remains the feature selection, not many other tools have been implemented in it so far. However the system contains some additional tools like a simple gaussian classifier or a data-conversion utility.



```
TRN: 'C:\data\wdbc.trn', features: 30, classes: 2 (357,212), "Wisconsin Diagnostic Breast Center"
DST: 'C:\data\wdbc.dst', classes: 2, features: 30, "Estimated from 'C:\data\wdbc.trn'"
APX: 'generated', components: 4, classes: 2, feat                              dbc.trn"
```

```
Total time elapsed: 00:00:00, Enumerations: 30
Final value: 0.819662, Final subset (size 1): 27

Sequential Forward Search:
on file 'C:\data\wdbc.dst', started on Thu Sep 16
Total time elapsed: 00:00:00, Enumerations: 59
Final value: 1.495208, Final subset (size 2): 13 27

Sequential Forward Search:
on file 'C:\data\wdbc.dst', started on Thu Sep 16
Total time elapsed: 00:00:00, Enumerations: 87
Final value: 1.812250, Final subset (size 3): 10 13

Exhaustive search:
on file 'C:\data\wdbc.dst', started on Thu Sep 16
Total time elapsed: 00:00:00, Enumerations: 30
Final value: 0.819662, Final subset (size 1): 27

Exhaustive search:
on file 'C:\data\wdbc.dst', started on Thu Sep 16
Total time elapsed: 00:00:00, Enumerations: 435
Final value: 1.801401, Final subset (size 2): 20 23

Exhaustive search:
on file 'C:\data\wdbc.dst', started on Thu Sep 16
Total time elapsed: 00:00:01, Enumerations: 4060
Final value: 2.329425, Final subset (size 3): 3 20

Criterion Calculator start: (on file 'C:\data\wdbc.dst')
Criterion Calculator end.

Lines 71  Chars 2672  / Buffer 4%
```

DST
Classes : 2
Features : 30

Classes
Class 1 : 1   Class 2 : 2

Criterion
[✓ Bhattacharyya] Divergence Patrick-Fisher
Generalized Mahalanobis Mahalanobis
Error Rate

Method specification
[✓ Forward] Backward
[✓ Basic] Generalized 2
[✓ Find single subset with size : 15]
Find subsets with sizes from : 1 to : 30

Results to buffer No. 0

.Order	Title	1	2
	SFS	0.819662	1.4952
	+times	00:00:00	00:00:0
	+enums	30	59
	Exhaustive	0.819662	1.8014
	+times	00:00:00	00:00:0
	+enums	30	435

ge : Exhaustive, row 3, subset size 3, (Ma

ation mixture graph

Figure 1.1: Feature Selection Toolbox: *a) Main Window, b) Console Window, c) Result Storage Table, d) Approximation Mixture Display.*

When using the FST do not expect much leadership from the system. A user who does not know much about feature selection methods may become lost in the jungle of parameters and/or misleading results. We therefore recommend to follow references mentioned throughout this text to find details about particular method properties or even the basic theoretical foundations. A good way to start is reading either Devijver–Kittler's book [1], Fukunaga [2] or e.g. Bishop [3] covering both the essential theory and most of the basic feature selection methods. For information on recent, more advanced methods see especially: Pudil et al. [16] and Somol et al. [22] on floating search methods, Novovičová et al. [13] and Pudil et al. [15] on mixture approximation approaches, Somol and Pudil [24] on oscillating algorithms, Yu and Yuan [27], Somol et al. [23] and Somol et al. [25] on Branch & Bound optimal techniques. The other references listed in the Bibliography section of this manual relate more specifically to either the FST itself, additional aspects of its use or additional information on particular implemented methods and their comparison.

This manual may not cover all the tiny details of FST usability and applicability, however it should be sufficient to enable reasonable work. To give an impression how the FST is designed: the important part of the code is encapsulated in a rather independent C kernel, while the user interface depends on Powersoft Optima++ C++ design environment and component libraries. Some of the kernel algorithms could therefore be ported to other systems, while the FST as a whole is a native Win32 application. The kernel size is about as few as 10000 lines of code, however it implements tens, if not hundreds of methods (if we distinguish between different modifications of equally founded algorithms).

Chapter 2

Using The Software – Basics

First thing to do with a new software is to properly install it. In case of the FST no installation is necessary, as the software files are self-contained. Simply create a new directory and copy all the FST files there. To start the FST application launch the 'fst50e.exe' executable.

Remark: Depending on a particular FST version this file may be named in a sligtly different way, e.g. 'fst49f.exe' or 'fst50b.exe' etc.

To start using the FST quickly follow the examples given here. Most of the important FST properties will then become apparent.

2.1 Parametric Feature Selection Methods Basics

Suppose our task is to select features using a given training data-set, one of available criterion functions and a particular feature selection method. For example suppose we want to select $d = 10$ features out of a total number of $D = 30$ on the sample "mammogram" dataset using the simple SFS method and the Bhattacharyya distance as a criterion function. Theoretically this approach requires an assumption of the normality of the dataset because our Bhattacharyya implementation is based on normal distributions. For sake of simplicity do not consider this an important question now. The sample dataset should be included as a part of the FST package and named 'wdbc.trn'. For details about the dataset open the 'wdbc.trn' file in any standard text editor and read the commentary in the file header.

Having launched the application, open the *Console* window (see figure 1.1–b) by clicking the *Console Window* checkbox on the main panel. The *Console* window will gather all the textual protocols on performed operations and obtained results. Open the data file using menu *File – Open*. See section 5 for more information about the data format. Open file 'wdbc.trn'. Having opened the file notice the change on the main panel – now it contains a line of text describing the opened TRN file (picture 1.1 a). To use the SFS method with the Bhattacharyya distance we need to estimate the normal distribution parameters using menu *File – Estimation(TRN->DST)*. Distribution parameters obtained in this way may be saved to a .dst file for future use using menu *File – Save DST as ...*.

Remark: Files of the .dst type may be later opened using menu *File – Open*. To access these files from the File Open dialog select *Files of type: – Data structure files (*.DST)*.

Now we can continue by invoking the feature selection method. From menu *Methods* select *(G)SFS or (G)SBS*. A method dialog appears. Ensure *Bhattacharyya* is selected in the *Measure* listbox inside the Criterion frame (see picture 4.3). Ensure *Forward* and *Basic* choices are selected in the *Method specification* frame (see picture 4.5). Write 10 into the *Find single subset with size:* field (picture 4.2 a). Press the *OK* button to start the computation.

After the computation end the *Console* window will contain a protocol about the results found. Most of the methods are accessible in a similar way from the *Methods* menu, with the only exception of the *Approximation methods*.

2.2 Approximation Methods Basics

The approximation methods (see Novovičová et al. [13]) should be used especially if we know nothing about the structure of the training dataset and the number of samples is as big as possible. These methods build a special normal distribution model that can be used both for feature selection and classification.

To illustrate these methods we will use a simple "speech" dataset containing 15-dimensional "yes" and "no" pronounced word patterns. Suppose we want to select $d = 5$ features out of a total number of $D = 15$ using the approximation method to build a 6-component model initialised by the so-called 'Dogs & Rabbits' initialisation strategy. Using menu *File–Open* open file 'speetr.trn'. From menu *Methods* choose *Approximation methods*. A detailed method dialog appears (see picture 4.10). Ensure *Full Data* is selected in the *Data source* frame (see 4.10 a). Select *Dogs & Rabbits* in the *Initialization* frame (see 4.10 b). Ensure *Approximation* method type is choosen in the *Procedure parameters* frame (see 4.10 c). Set the percentual EM-algorithm convergence stop limit to 0.5 (d). Write 5 into the *Single Run – Find reduced subset size:* field (e) and 6 into the *Components* field. Press the *OK* button to start the computation.

The results will be written to the *Console* as usual. Unlike the parametric methods described above the approximation methods retain the resulting mixture model in memory (as indicated on the main panel in the APX: line, see 1.1 a). The mixture model can be saved to a file using menu *File – Save APX as...*. Files of this type can be opened later using *File – Open*, where *Files of type: – Approximation mixture files (*.APX)* must be selected in the File Open dialog. The obtained mixture model can be now used for classification with the Pseudo–Bayes classifier.

Select menu *Classifiers – Pseudo-Bayes*. Try to use the classifier to check the correct classification rate. Ensure *Test for error rate* is selected in the *Classification Purpose* frame (see picture 4.13 a). Select *Other File* in the *Test for error rate* frame (b) and then using the *Open* button select the testing file 'speezk.trn'. Press the *OK* button to start the computation. Classification results will be output again to the *Console* window.

Remark: To assess the classification performance reliably, different datasets (or dataset parts) should be used for training and testing.

Chapter 3

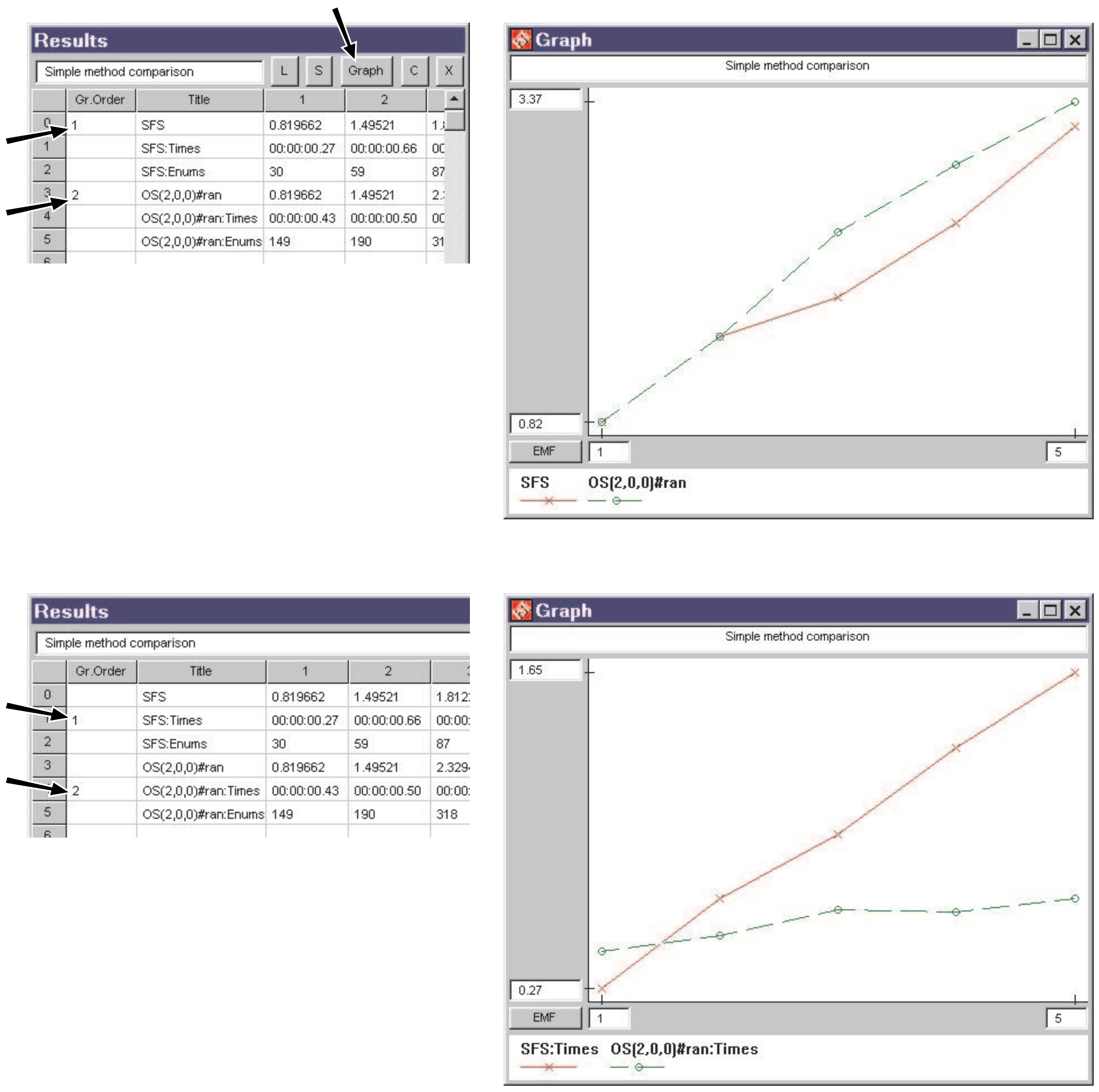
Result Storage and Processing

All the feature selection or classification method results are written in form of a textual report to the *Console* window. No information is stored internally concerning the selected features for other use. The only exception are the approximation methods that generate a mixture model to be used later with the Pseudo-Bayes classifier. The textual report contains a detailed information about the course of the finished algorithm, elapsed time, number of criterion evaluations, final criterion value, the selected subset described by a list of feature indexes starting from 0, processed source data file etc.

As the gathered protocols usually contain important information, the *Console* allows to store its contents by pressing a small *S* button on its right side to a text file. The *C* button serves for cleaning (erasing) all of the *Console* contents. A textual file can be opened using the *L* button, e.g. to enable continuing previously interrupted work session. The *R* button just puts a time-mark to the text. The *X* button closes the *Console* window, however all its contents remain preserved in memory and will appear next time the window is opened (within the current application run). The same holds for the *Results* window (see below).

Single-value numerical results, i.e. resulting criterion values, computational time and final number of criterion evaluations can be additionally stored in the *Results* table (see picture 1.1 c). Such records can be used later e.g. for generating graphs. The table window can be opened similarly as the *Console* – using the *Results Window* switch on the main panel. However for most of cases we recommend to set-up the whole workplace simply using menu *Window – Results Desktop*.

Collecting the results in the table may become interesting if we needed to call feature selection algorithms in series, e.g. for different target subset sizes d.



	Gr.Order	Title	1	2	
0	1	SFS	0.819662	1.49521	1.
1		SFS:Times	00:00:00.27	00:00:00.66	00
2		SFS:Enums	30	59	87
3	2	OS(2,0,0)#ran	0.819662	1.49521	2.
4		OS(2,0,0)#ran:Times	00:00:00.43	00:00:00.50	00
5		OS(2,0,0)#ran:Enums	149	190	31
6					

	Gr.Order	Title	1	2	3
0		SFS	0.819662	1.49521	1.812
1	1	SFS:Times	00:00:00.27	00:00:00.66	00:00:
2		SFS:Enums	30	59	87
3		OS(2,0,0)#ran	0.819662	1.49521	2.329
4	2	OS(2,0,0)#ran:Times	00:00:00.43	00:00:00.50	00:00:
5		OS(2,0,0)#ran:Enums	149	190	318
6					

Figure 3.1: Invoking the *Graph* window from the *Results* window. These two *Results* – *Graph* window pairs illustrate how to select different table rows for drawing a different graph.

Whenever you need to compute a series of results, set the required subset size interval in the appropriate method dialog, as seen on picture 4.2 (b). Suppose we want to compute results for $d \in < 2, 5 >$. Resulting criterion values should be stored to a specified row of the *Results* table. Select the row number (use 0 for instance) using switch (c) in a method dialog (as seen on picture 4.2). The results will be stored as follows: the cell in row 0, column *Title* will contain a short method textual identifier, columns 2 to 5 will contain criterion values of selected subsets of 2 to 5 features. The next row will contain records about the spent computational time and the next row will contain numbers of criterion evaluations. Note that each series of results of a single method occupies three subsequent rows in the *Results* table. Keep it in mind when selecting a row to store results of next methods.

The *Results* table can be saved to a file (with extension ".res") using button *S* that can be found in the top right corner of the table window (as seen e.g. on picture 3.1). Opening an existing file is possible using the *L* button. Button *C* clears the table. Button *X* closes the window. The window can be temporarily closed without loosing the results.

3.1 Graphs

Series of results stored in the *Results* table may be used for drawing simple graphs. Before invoking the *Graph* window the user must specify which rows of the *Results* table should be used as well as their order. To select rows, mark them in the *Gr.Order* column using single unique digits. First row to be represented in the graph should be marked by 1, the second by 2 etc. The graph can be displayed not only for rows containing numerical results (criterion values, numbers of evaluations), but also for the time records. Time records are first converted to a single value representing a total number of seconds. Having marked the rows, click the *Graph* button to open the *Graph* window.

Remark: The *Gr.Order* column may be used not only to specify graph row order, but also to specify graph appearance. Use expressions of type a:b:c:d instead of single digits, where each character (a-d) must be replaced by a single digit. The 'a' represents row order (like if the expression was just a single digit a), 'b' specifies graphical line type, 'c' specifies line color and 'd' specifies cross-point symbol type. A legal expression is e.g. 1:1:1:1 or 2:3:4:5.

Once the *Graph* window has been opened, the graph still can be modified. Rewrite numerical values assigned to axis beginnings and ends to change displayed intervals. The graph may be exported in form of an Extended Windows Metafile (EMF) file for further processing.

3.2 Options

The textual and/or graphical form of results can be modified by setting some of the application-wide options accessible from menu *Options*:

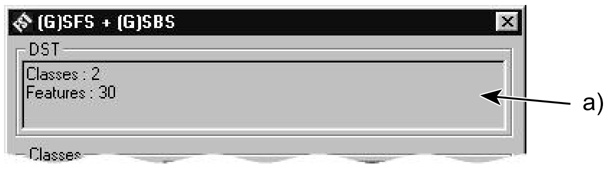
- *compact console protocol* – Textual result output is reduced to short single-line reports.
- *binary indexes in console output* – Feature subsets are described as 01000101.. instead of 1,5,7,..
- *randomize random seed* – Relevant mostly for the oscillating search if set to use random initialisation (see picture 4.8 f). If unchecked, the results of the randomly initialised algorithm would be the same when repeating equally set-up computations.
- *stop on criterion evaluation errors* – Normally the algorithms ignore errorous criterion values. An errorous value may result from e.g. numerical problems caused by weakly conditioned covariance matrices. If this happens in one of many candidate computational branches, the algorithm simply ignores the branch as it does not improve the current candidate subset. Set this switch e.g. if you need to assess the behavior of the criterion function.
- *SFS and SFFS output single result only* – Normally both SFS and SFFS output a series of results for all dimensionalities 0 to d after a single computation, because due to SFS and SFFS principle these results are obtained anyway.
- *Approx.Methods: hide ellipses in drawing* – Normally the *Approximation Graph* window (see picture 1.1 d) displays both the projection of training data samples and the current mixture in form of equipotential plane ellipses. Set this switch to display raw data only. Read 4.7 for more details.
- *Approx.Methods: draw different shapes for diff. classes* – Normally the samples and mixture ellipses of different classes are distinguished by color only. If you need to capture the image and use it for b/w printing, set this switch. Read 4.7 for more details.

Chapter 4

Invoking the Methods

The following sections describe the behavior and parameters of feature selection methods implemented in the FST. All of them can be invoked from the *Methods* menu. With exception of the Approximation methods all the others require a normal model (.DST) residing in memory. Typically, before invoking a feature selection method the user would open a .TRN file and convert it to the .DST using menu *File – Estimation(TRN->DST)* or directly open on exisitng .DST file.

The set of implemented feature selection methods consists both of classical and newer or even experimental ones. To find out more about the topic and about the informational resources read section 1.

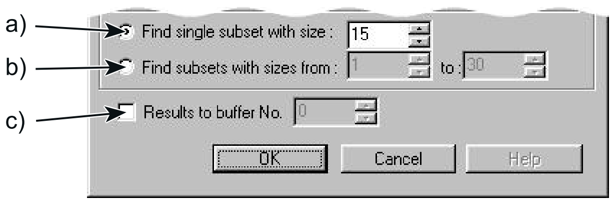


Classes : 2
Features : 30

Figure 4.1: Each method dialog shows a brief information about the currently processed dataset.

Selecting a method from the *Methods* menu opens its method dialog allowing detailed parameter settings. Any method dialog usually allows to choose from a set of related methods as well as to set different method-dependent parameters. All the method dialogs are designed as similarly as possible and differ from each other in method-specific parameter controls only (the Approximation method dialog differs more significantly from the others because the method itself is based on a different concept).

Each dialog has an informational section in its upper part as seen on figure 4.1. A short information about the data is displayed there to prevent any confusion about what is to be processed. The next dialog section common among the method dialogs is the Criterion frame described in the following sub-section (see figure 4.3). The last common part is in the bottom of the dialog (see figure 4.2).



Find single subset with size : 15
Find subsets with sizes from : 1 to : 30
Results to buffer No. 0

Figure 4.2: Each method dialog requires setting the target subset size or an interval of sizes for repeated method calls.

The bottom dialog part contains (besides the ordinary buttons: *OK* to start the computation, *Cancel* to return to the main application) a mandatory parameter field for target subset size specification. Either a single subset size d (see figure 4.2a) or an interval of sizes (see figure 4.2b) must be selected to enable the method to run. If the interval is selected, the method will be invoked repeatedly for all relevant target subset sizes.

Final results are stored to the *Console* window by default. However, they can be stored additionally to the *Results* table as described in section 3. To enable the additional value storage check the checkbox shown on figure 4.2c and specify the destination table row number.

Most of method dialog parameter fields offer pre-specified choices, e.g. the subset size field is pre-set to $D/2$ by default. Despite that we recommend to revise carefully all settings. The final results may be affected not only by incorrect, but also by legal but unsuitable settings.

4.1 Criterion Functions

An essential part of method preparation is choosing a particular criterion function. The criterion function is then used during the course of the algorithm to evaluate candidate feature subsets. In fact, the subset selection methods aim to find a subset yielding the highest possible criterion function value. FST offers a choice of basic criterion functions being known as probabilistic distance measures. All these functions generally yield higher values for those feature subsets (sub-spaces), for which the class normal distributions overlap the least and therefore class separability should be higher. However, different criteria may lead to different feature selection results. For details and discussion see e.g. [1, 26].

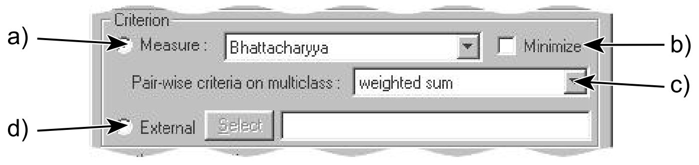


Figure 4.3: Criterion function selection frame (available on most method dialogs).

To select one of available criterion functions for use in a search method use the listbox as on figure 4.3a. All the available criteria are defined for two-class problems only. To overcome this limitation we allow combination of pair-wise results if the number of classes is higher than 2. For such cases select one of pair-value combination strategies using a listbox shown on figure 4.3c. All the feature selection methods are built to maximise criterion values. However, if minimalisation is wanted (e.g. to explore the "worst" subsets) and the particular method allows it, check the *Minimize* checkbox as on figure 4.3b.

If the built-in criterion functions are not sufficient, the FST offers a choice to use external functions (see figure 4.3d). Such an external function may be e.g. a user-constructed classifier – the result of the feature selection process would then be probably more useful than those obtained using the standard criterion functions. However, using external functions is not flawless and requires additional programming effort. Following description is therefore meant mostly for enthusiasts.

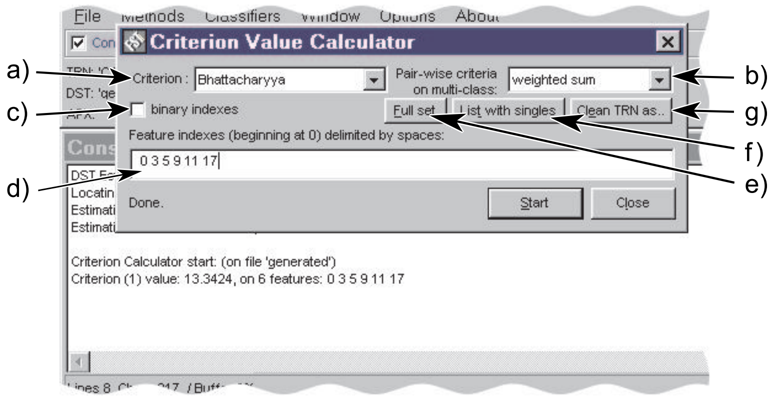
–

To use external criteria, FST requires the user to supply an external executable that can be invoked with special command-line parameters (see later) and returns a simple textual file containing a single number as a result. Such an executable should then be selected using the *Select* button as shown on figure 4.3d. Let us demonstrate the parameter passing on a hypotetical executable 'crit.exe'. The FST would invoke it as follows: 'crit -p001101...01001'. The '-p' option is followed by a binary list of D values. Ones depict the currently selected candidate features, zeros depict the unselected ones. The number of ones (feature candidates) may differ from the target subset size d during the computation. The FST then expects to find a textual file named 'class.res' in the same working directory from which the 'crit.exe' had been invoked. The 'class.res' should contain nothing else than the C-style floating result value. If the FST fails to open and read this file, the computation may not stop if the *stop on criterion errors* option had not been set (see section 3.2).

This way of external criterion function calling may be considered rather unpleasant, however it has been programmatically the simplest way to extend the FST usability. Such an external function support should be considered experimental also because of another limitation – due to certain Windows memory leaks the repeated process invocation may cause the whole system become unstable after several thousands or tens of thousands criterion evaluations.

–

Apart from particular method dialogs the criterion functions are concerned also in the Criterion Value Calculator (see figure 4.4). The tool can be invoked using *File – Criterion Value* menu to enable user-specified subset evaluation.



Criterion : Bhattacharyya
Pair-wise criteria on multi-class: weighted sum
binary indexes
Feature indexes (beginning at 0) delimited by spaces:
0 3 5 9 11 17
Done.

```
Criterion Calculator start: (on file 'generated')
Criterion (1) value: 13.3424, on 6 features: 0 3 5 9 11 17
```

Figure 4.4: The Criterion Value Calculator enables computation of a criterion value (a,b) for a specified feature subset (d). To access the calculator use menu *File – Criterion Value*. The (c) checkbox enables entering binary form of subsets (001101011) instead of numerical indexes (2 3 5 7 8). (The (b) option is relevant for 3- and more-class data only.)

Moreover the calculator can be used to evaluate criterion values for full sets of features (see figure 4.4e), or for a list of criterion values computed on single features (i.e. on subsets of size 1, see figure 4.4f). This type of information can help in understanding underlying data properties – occassionally certain features yield apparently high values and therefore may be immediately considered as important. On the other hand the list of singles can reveal numerical problems following from badly conditioned covariance matrices – look for features yielding value $-5e + 300$ which marks computational failure. Remark: watch what happens if some subset had been specified in (d) before generating the list of singles (f).

The (g) choice is meant for experimental use only. It requires both a TRN and a appropriate DST data to be opened in the FST. Then it generates a reduced copy of the TRN data file, containing only a "promising" subset of original features, i.e. those features yielding reasonable individual criterion values (not equal to $-5e{+}300$). This may help when working with very large sparse matrices that would normally prevent the feature selection methods to find any result in a reasonable time.

4.2 Sub-Optimal Methods – Sequential Search

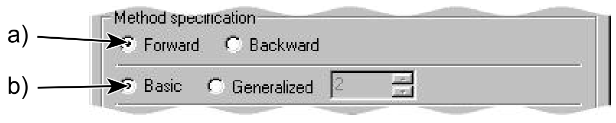


Figure 4.5: Sequential Search Parameters.

The sequential search based methods are the simplest methods implemented in the FST. The well known Sequential Forward Selection (SFS) or Sequential Backward Selection (SBS) can be invoked by setting the *Basic* option as on figure 4.5b and either the *Forward* or *Backward* direction of search (see figure 4.5a). In general the *forward* algorithm versions are faster than the *backward* ones. However, it is not usually possible to predict which version would yield better results.

The option *Generalised* (see figure 4.5b) would set the methods to run in the generalised mode where groups of features are tested in single steps instead of single features as in SFS and SBS. Generalised sequential methods are known as GSFS(r) and GSBS(r) where r is to be user-specified and means the number of features tested in single steps. Increasing the r value for generalised methods may improve the results sligtly at a cost of significantly increased computational time. For further algorithm descriptions see e.g. [1].

Although the simple sequential methods are rather popular because of their speed and simplicity, the FST offers a better choice – the Oscillating search (see section 4.5) can be set-up to run even faster than the SFS while yielding better results.

4.3 Sub-Optimal Methods – Plus-l–Minus-r

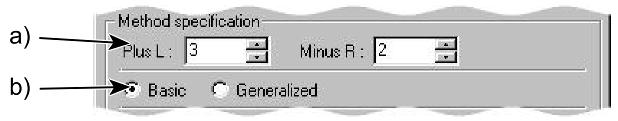

Method specification
Plus L : 3   Minus R : 2
[✓ Basic] Generalized

Figure 4.6: Plus-l–Minus-r Search Parameters.

The Plus-l–Minus-r methods constitute a first attempt to overperform the simple sequential methods. They require the user to specify two parameters l and r as shown on figure 4.6a. Parameter values determine the prevailing search direction – for $l > r$ the algorithm runs in a *forward* manner, for $l < r$ in a *backward* manner, $r = l$ is illegal. The algorithm uses a very simple backtracking to repeatedly improve solutions found in previous steps.

As with sequential methods a *Generalised* version is available (see figure 4.6b). Both standard and generalised algorithm versions are usually outperformed especially by the Floating search (see section 4.4) and are included here for comparison purposes only. For more information on Plus-l–Minus-r see e.g. [1].

4.4 Sub-Optimal Methods – Floating Search

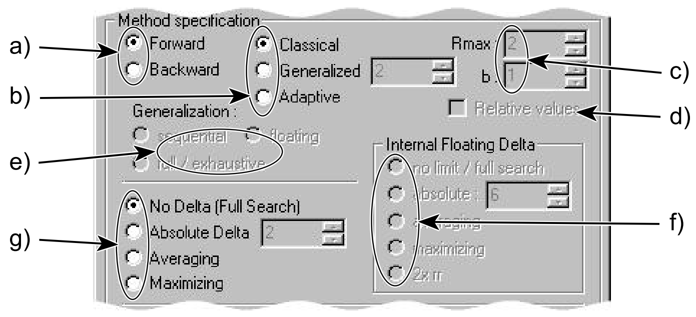

Method specification
[✓ Forward] Backward
[✓ Classical] Generalized 2 Adaptive
Rmax : 2
b : 1
Relative values
Generalization : sequential floating full / exhaustive
Internal Floating Delta: no limit / full search, absolute : 6, averaging, maximizing, 2x rr
[✓ No Delta (Full Search)] Absolute Delta 2 Averaging Maximizing

Figure 4.7: Volba parametr plovoucch metod.

The Floating Search constitutes a powerful and advantageous approach to subset selection. As with previous methods, the Floating Search is defined either in a *forward* or *backward* version. All the floating search algorithms keep track of the so-far best solutions in all dimensions. They utilise a context-based backtracking that attempts to improve any solution obtained so-far in any dimension and therefore to prepare better starting points for further search targeted to the required dimensionality. The Floating Search algorithms are fast and more effective than any of the algorithms mentioned in previous sections. After a single run they can yield solutions for all dimensions.

To select the prevailing direction of search use switch (a) as shown on figure 4.7. The *forward* algorithm version may be considered as a good choice for most of problems.

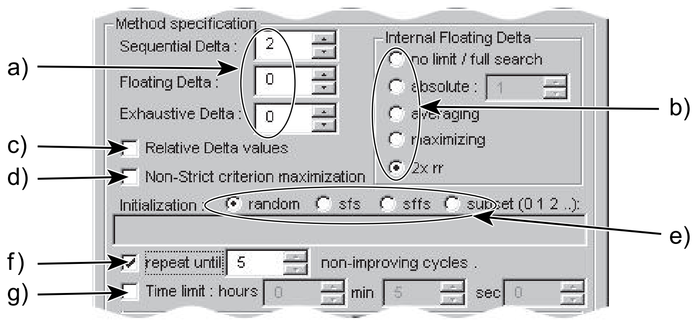
The Floating Search principle has been further developed. Switch (b) offers a selection of: the *Classical Floating Search* (recommended for most problems) as described in Pudil et al. [16], the experimental *Generalised* Floating Search (for a brief description of the "generalisation" idea read section 4.2) or the *Adaptive Floating Search* as described in Somol et al. [22].

The *adaptive* algorithm version focuses more thoroughly to the target dimensionality, therefore it may find a better solution than the *classical* version. However this is usually achieved at a cost of significantly increased computational time. Moreover the algorithm does not yield all the results for all dimensionalities in one run any more. Currently the Oscillating search (see the following section) is a better tool if possibly the very best solutions are required and the nature of the problem prohibits a use of optimal methods. The *classical* floating search then remains a good universal choice yielding very good results at a reasonable computational time cost and a reasonable implementation complexity.

If the *Adaptive* floating search is selected, additional options become enabled (c,d,e). The meaning of the *Rmax* and b parameters is described in detail in [22]. As usual, higher values may improve the results at a cost of slower computation. Switch (d) enables setting the parameter values (figure 4.8c) in percentual form (values 1 to 100) relatively to the total number of features (D). Switch (e) affects the internal computation of single adaptive algorithm steps – when a step involves a group of features, one of methods selected using (e) is used as a sub-procedure. If the *floating* method is selected as a sub-procedure in (e), an additional set of options becomes available (f). These settings are experimental and the default *2xrr* choice should work the best.

An important setting common to all floating search algorithm versions is the Δ limit. The limit can be set to prevent the algorithm from potentially senseless computations once the required result of a given dimensionality has been achieved. For detailed description of the Δ meaning see [16, 22]. To get the best results set the (g) switch to *No Delta/Full Search*. This ensures the algorithm to finish the computation and to fully utilise its potential. The other choices can be set to save computational time. The *Averaging* and *Maximizing* choices cause a dynamic Δ prediction throughout the course of algorithm based on the longest backtracking sequence followed so-far. Choosing the *Absolute Delta* option lets the user specify a constant Δ limit – the SFFS algorithm then will stop at dimension $d + \Delta$, the SBFS at $d - \Delta$.

4.5 Sub-Optimal Methods – Oscillating Search



Method specification
Sequential Delta : 2
Floating Delta : 0
Exhaustive Delta : 0
☐ Relative Delta values
☐ Non-Strict criterion maximization

Internal Floating Delta
○ no limit / full search
○ absolute : 1
○ averaging
○ maximizing
[✓ 2x rr]

Initialization : [✓ random] ○ sfs ○ sffs ○ subset (0 1 2 ..):

✓ repeat until 5 non-improving cycles .
☐ Time limit : hours 0 min 5 sec 0

Figure 4.8: Oscillating Search Parameters.

Oscillating Search is a highly effective and flexible search concept (see Somol et al. [24]) and has been implemented in three concrete versions in the FST. The most important parameter is always the oscillation cycle depth, denoted Δ. Simply said, the algorithm updates cyclically a working set of a requested number of d features by removing/adding one or more features at a time. The oscillation cycle depth determines the maximum number of features allowed to be exchanged in the working subset during one cycle. Higher Δ values may give better results but require more computational time. For more details on the method behavior see the cited paper.

The Δ parameter may be specified using fields (a) as shown on figure 4.8. Legal Δ values are positive integers. If some of the (a) fields contains 0, the particular algorithm version will not be called, otherwise the different algorithm versions would be called in sequence. If the fields contained e.g. 10, 5, 3, the algorithm would run as follows: first the fastest but least powerful *sequential oscillating search* would be called with $\Delta = 10$, then the obtained result would be taken as an entry point for the slower *floating oscillating search* with $\Delta = 5$, then its result would be used again as a starting point for the most promising, but slowest *full oscillating search* with $\Delta = 3$. However, calling a single method at a time is reasonable enough, as mentioned in the following.

One of the biggest advantages of the oscillating search is its ability to tune (improve) solutions obtained previously in any other way. Its computation always starts from a pre-specified point, i.e. an initial subset of d features. Therefore an integral part of method invokation is its initialisation. Switch (e) as shown on figure 4.8 enables setting the initialisation type. Besides s single *SFS* or *SFFS* (full search not restricted by a floating limit constant) it is possible to specify a concrete initial subset by choosing the *subset* option and writing the feature indexes (indexed from 0 and separated by spaces, e.g. 0 1 5 10 14 etc..) to the now available empty textual field. The number of indexes must correspond with the specified target subset size (see figure 4.2a).

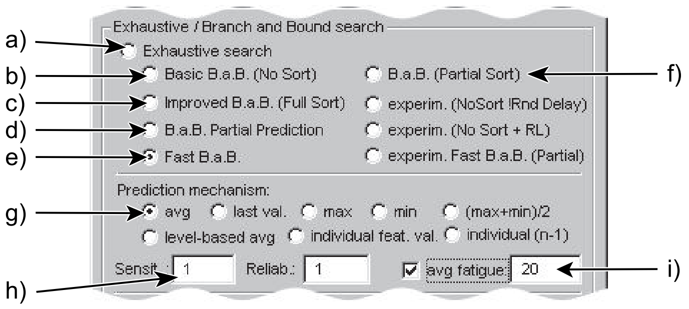
The last available initialisation strategy is a *random* initialisation. The "randomness" may seem as a disadvantage, however we usually achieve the best results using this approach. Moreover, this initialisation is advantageous also because its speed – as opposed to the SFS, SFFS or other algorithmic initialisations it practically does not increase the overall amount of computational time. Experiments show, that when being repeated the random initialisation may result both in an "average" oscillating search result (comparable with other methods) or in a significantly better one. This in fact illustrates the limitations (the sub-optimality) of the sequential search principle adopted by most of known sub-optimal methods – the sequential methods simply follow a defined search path and may easily end up in a local extreme. If the goal is really to find the best possible result on a high-dimensional problem we recommend to select the *random* initialisation and request its automatic repeating to increase the chances of finding something better. Do it using the *repeat until* option as shown on figure 4.8f. Specify then a number of algorithm runs (a good choice is e.g. 5). In fact the internal run counter is then reset any time the new algorithm run resulted in overall improvement – that means the total number of runs may be higher and the user-specified number will actually mean the number of non-improving runs before the cycle ends.

Option (g) enables to specify a maximum allowed computational time. The nature of the oscillating search allows this, as some working subset of d features is always available. Stopping the algorithm after a given amount of time simply returns the best result achieved so-far.

Options (b) can affect the internal computation behavior in case of the *floating oscillating search*. These choices should be considered experimental and the default one (*2xrr*) should be used practically always. Option (c) enables setting the Δ (figure 4.8a) values in percentual form (values 1 to 100) relatively to the total number of features (D). Option (d) broadens the extent of the search a little bit. It forces the algorithm to follow even less prospective search paths. For more details on all the parameters see the cited paper.

Remark: According to our experience two algorithm configurations are particularly powerful: 1) The *sequential oscillating search* with small Δ values and (optionally repeated) random initialisation is one of the fastest methods of all and is well suitable e.g. for problems of very high dimensionalities where practically all the other methods (including SFS) fail to find solutions in a reasonable time. Moreover, starting from different initial points with repeated random initialisation gives good chance to find solutions, that could not be found using standard sequential methods due to their principal definitions. The randomised *sequential oscillating search* also may succeed in cases where due to numerical problems the others fail to find any solution. 2) The *full oscillating search* gives usually very good results even with very small Δ. Although its speed decreases significantly with increasing Δ, our experiments show that even with $\Delta = 2$ or $\Delta = 3$ the algorithm usually outperforms all the other algorithms including the SFFS. Again, random (optionally repeated) initialisation results in better algorithm speed and better chance to find a real optimum.

4.6 Optimal Methods



Exhaustive / Branch and Bound search

Exhaustive search

Basic B.a.B. (No Sort) | B.a.B. (Partial Sort)

Improved B.a.B. (Full Sort) | experim. (NoSort !Rnd Delay)

B.a.B. Partial Prediction | experim. (No Sort + RL)

[✓ Fast B.a.B.] | experim. Fast B.a.B. (Partial)

Prediction mechanism:

[✓ avg] last val. max min (max+min)/2

level-based avg individual feat. val. individual (n-1)

Sensit.: 1 Reliab.: 1 [✓ avg fatigue:] 20

Figure 4.9: Optimal Search Parameters.

All the implemented optimal search methods are grouped in this dialog. The simplest choice would be the *Exhaustive* search (see figure 4.9a). This algorithm simply tests all the possible feature subset configurations. Therefore it becomes extremely slow with increasing dimensionality. It should be used in cases where: the overall dimensionality (D) is very small, or target subset size d is close to 1 or D, or an external non-monotonic criterion function is used. One of the Branch & Bound algorithms should be used whenever possible to obtain the result faster.

In general all the optimal methods are significantly slower than the sub-optimal ones (that is the main reason to define sub-optimal methods). The speed difference between the slowest and fastest Branch & Bound algorithm may be described by factor 10 to 100, but even the fastest Branch & Bound methods become unusable with increasing dimensionality of the problem, especially if d is close to $D/2$. According to our experience the practical limitation lies somewhere around $d = 25, D = 50$ on current PC machines. This may differ a little bit depending on dataset and criterion properties.

Choices b) to f) on figure 4.9 then offer different Branch & Bound algorithms. These algorithms can be used with monotonic criterion functions only, however this property is fulfilled by all the standard functions available in the FST:

- Choice b) – *Basic B.a.B.(No Sort)* is the simplest Branch & Bound algorithm defined for feature selection purposes in 1978 by Narendra and Fukunaga [2]. It requires no additional parameters. It should be used for comparison and demonstration purposes only. *Remark: This algoithm is denoted as 'B.a.B. (No Sort)' in older FST versions.*

- Choice c) – *Improved B.a.B.(Full Sort)* is the most widely accepted Branch & Bound version featuring an additional internal heuristics for accelerating search-tree construction. It requires no additional parameters. This algorithm can be considered a reference algorithm when assessing the speed of more advanced prediction-based algorithms (described below). *Remark: This algoithm is denoted as 'B.a.B. (Full sort)' in older FST versions.*

- Choice d) – *B.a.B. Partial Prediction* invokes the *Branch and Bound with Partial Prediction* as described in Somol et al.[25]. We consider this algorithm the best choice for general use, as its performance is notably higher than that of the previous algorithms. As this is a prediction-mechanism based algorithm, additional parameters may be set for it as described in the following. However, the standard setting is supposed to be the best and the others serve mostly for experimental pusposes only. *Remark: This algoithm is denoted as 'B.a.B. (Partial 2)' in older FST versions.*

- Choice e) – *Fast B.a.B.* invokes the *Fast Branch & Bound* as described in Somol et al. [23]. This is the most powerful algorithm known to us and implemented so-far. It uses a prediction mechanism to predict the search tree topology and to omit many non-prospective computations. Additional parameters must be set as described in the following. *Remark: This algoithm is denoted as 'Fast B.a.B.' or 'Restrictive B.a.B.' in older FST versions.*

- Choice f) and the 3 next ones serve for experimental purposes only.

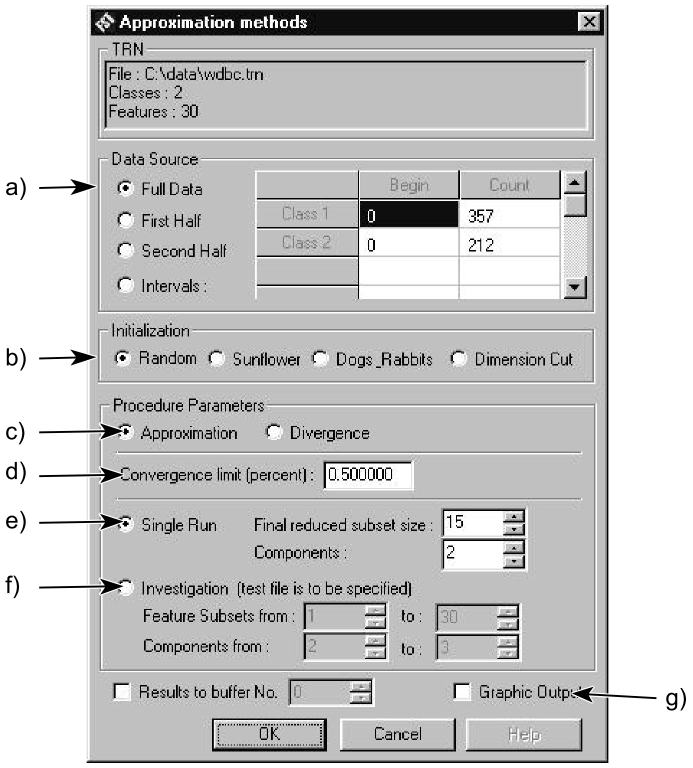
Remark: all the Branch & Bound algorithms are implemented to construct a *minimum solution tree* as described in Yu and Yuan [27].

The predicive algorithms (choices d) and e)) may be affected by setting different additional parameters. Different prediction mechanism types are implemented here, mostly for experimental purposes. The best prediction mechanism for vast majority of problems is the *avg – averaging prediction mechanism* as shown on figure 4.9g. It collects the information about average criterion value changes caused by feature removals during the search tree construction process.

Option i) enables to affect the speed of the *averaging prediction mechanism* learning process. The *fatigue* can be set to values greater or equal to 1. Close-to-1 values cause the learning process to prefer recent values. That means the prediction information would evolve quickly dependend on particular search tree context. High values or even deselecting the i) option causes the prediction mechanism to stabilise with time. This is supposed to be more advantageous in general, however it may not be the case everytime.

The *Fast B.a.B.* algorithm requires two additional parameters that may affect both its speed and its robustness. The *Sensitivity* (denoted *optimism constant* γ in the paper) value affect the way the algorithm utilises the prediction mechanism to restrict search tree construction. It can be set to real values close to 1, while 1 is the default and most 'independent' value. Reasonable values should lie within approximately $(0.2, 2.0)$. The *Reliability* (denoted δ in the paper) value can be set to slow-down the prediction mechanism start. This parameter sets the number of learning steps required before the prediction mechanism is allowed to start. Legal values are therefore integers greater or equal to 1. We recommend to set this value approximately to 5. See the paper [23] for more information on the parameters meaning.

4.7 Approximation Methods



Approximation methods

TRN
File : C:\data\wdbc.trn
Classes : 2
Features : 30

Data Source
[✓ Full Data] First Half Second Half Intervals :

	Begin	Count
Class 1	0	357
Class 2	0	212

Initialization
[✓ Random] Sunflower Dogs_Rabbits Dimension Cut

Procedure Parameters
[✓ Approximation] Divergence

Convergence limit (percent) : 0.500000

[✓ Single Run] Final reduced subset size : 15
Components : 2

Investigation (test file is to be specified)
Feature Subsets from : 1 to : 30
Components from : 2 to : 3

Results to buffer No. 0 Graphic Output

OK Cancel Help

Figure 4.10: Approximation Method Parameters.

The Approximation methods (for details see [15, 13]) are principally different from the others and therefore need a special setting dialog (figure 4.10). Frame (a) can be used to restrict the approximation training process to certain data parts (intervals) only. Using this restriction can be advantageous if only a single data set is available – its first half may then be used for training and later the second half for testing.

Switches (b) gathered inside the *Initialization* frame allow selection of the mixture parameters initialisation strategy. Switch (c) allows to select either the so-called *Approximation method* (better for data-representation, usable with more than 2 classes, see [15]), or the so-called *Divergence* method (better for class–discrimination, restricted for use with 2 classes only, see [13]).

Both of the methods repeat computational cycles consisting of mixture parameters recalculation and estimation of the new mixture Likelihood value (data fitting "quality"). The algorithms are known to converge, i.e. the Likelihood differences (current - last value) tend to be smaller with time. To prevent the computation from infinite run we restrict it by setting a lowest allowed Likelihood percentual difference (parameter d), that actually characterizes a moment when no further improvement of the mixture model can be reasonably expected.

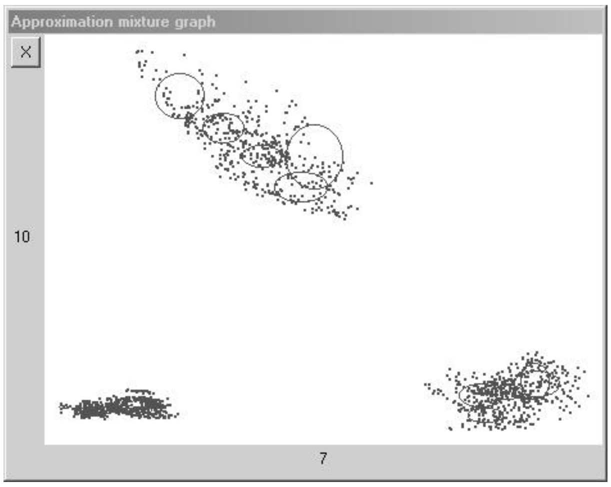
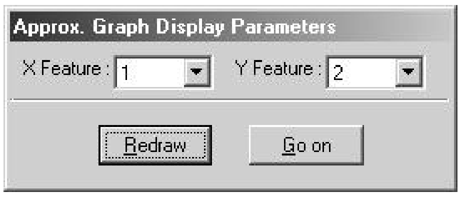


Figure 4.11: 2D Data projection with mixture component equipotential planes drawn as ellipses.

As usual in FST the target subset size is to be specified. In addition to that, a number of mixture model components is to be specified as well (see figure 4.10). The (f) choice serves for experimental purposes – it allows repeating the algorithm for different feature subset sizes and numbers of components followed by a classification process to find the best subset/components configuration yielding highest classification rates.



Approx. Graph Display Parameters

X Feature : 1 Y Feature : 2

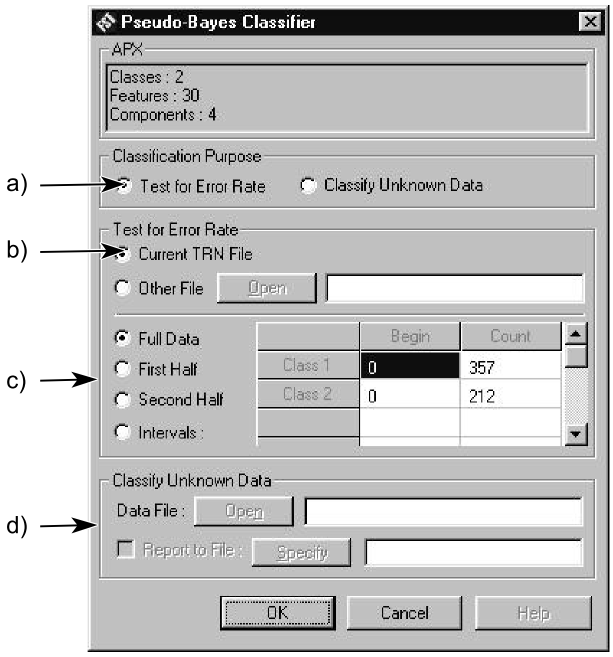
Figure 4.12: Selecting the dimensions for 2D approximation data display.

Mixture model evolution can be observed visually during the course of the algorithm in form of a 2D data-space projection with components displayed using correllation ellipses. The projection is displayed in the *Approximation Graph* window (see figures 1.1c and 4.11) that must be opened before the approximation start. In addition, the *Grpahic Output* option must be checked (see figure 4.10g).

The *Approximation Graph* window contents are then redrawn after each approximation computation step. The computation remains suspended until the user clicks '*Go on*' on the small dialog popping up after each computational step (see figure 4.12) and allowing different 2D subspace projections to be selected and drawn.

–

Both approximation mixture based methods generate a mixture model that can be later used for classification. As a part of the process a feature subset is selected. The mixture model is actually build on the selected subspace, while the remaining dimensions are used in a form of "background" information.



Pseudo-Bayes Classifier

APX

Classes : 2
Features : 30
Components : 4

Classification Purpose

[✓ Test for Error Rate] Classify Unknown Data

Test for Error Rate

[✓ Current TRN File]
Other File

[✓ Full Data]
First Half
Second Half
Intervals :

	Begin	Count
Class 1	0	357
Class 2	0	212

Classify Unknown Data

Data File :

Report to File :

Figure 4.13: Pseudo-Bayes Classifier Settings.

The obtained mixture can be used with the so-called pseudo-Bayes classifier (accessible from menu *Classifiers – Pseudo-Bayes*) either for classification of unclassified/unknown data stored as a single-class .TRN file (see figure 4.13a) or for classification rate estimation (figure 4.13b). In case of classification rate estimation either the current .TRN file (switch b) on figure 4.13) or a different one can be used. Set of choices c) allows to restrict the testing process to a specified part of the test data.

If classifying unknown data, select the data file using the *Open* button in the *Classify unknown data* frame as shown on figure 4.13d. Use the *Report to file* checkbox to specify a log-file, that would contain the classification information.

Chapter 5

Accepted Data Formats

Because of our primary goal – to make a software usable for different purposes in different scientifical applications – we had to choose a universal and simple way of data input. The FST therefore processes data in a form of standard text files with numerical values stored using the C language convention. To let the application recognize the meaning of the data, all files must contain a simple textual header, that can be usually created without bigger effort. The header stores all the essential information about the data, like the number of classes, features, samples, etc. An example and description of required header contents follows.

Suitably formated data files may then be modified in different ways using the File Manipulation Dialog tool (see sect.5.1). This tool enables data file cutting or joining according to different user-specified settings (e.g. which features to preserve, which data intervals to delete, etc.).

The FST currently supports three basic file types:

1. Basic data files containing data samples, i.e. vectors of feature values describing a certain number of samples from a set of classes. They are depicted by a **.TRN** file extension. The ordering of numerical values is assumed to be according to (most important first): class (set of pattern vectors), sample (pattern vector), feature (pattern vector member).

   This file type is the most universal one, because it may be used (in its original or processed form) with all FST tools. It is possible to build an approximation model upon it (see .APX description in the following) or to estimate gaussian mixture parameters from it (see .DST description in the following). In this sense the basic data file can be used for feature selection, data approximation, classification rate estimation or even for classification of unclassified data.

It should be noted, that opening the file does not cause any memory allocation as such. During the opening phase the FST only reads the data header and records the position of the first data record within the file. The actual data are read later, possibly repeatedly during the feature selection process. This enables to use large training sets on systems with moderate physical memory.

The appropriate header for basic data files should look like in the sample below. All keywords must be placed at the beginning of lines. The first line must contain the `#datafile` keyword. The `#title` line is optional. The `#features` and `#classes` lines are mandatory. The `#features` keyword must be followed by a value depicting the number of features, separated by any whitespace character(s). Remark: whitespace characters are used as standard delimiters to separate numbers and/or keywords. The `#classes` keyword must be followed by a value depicting the number of classes, whitespace, and a series of class sizes separated by commas. The ";" character at the beginning of a line depicts a remark – following text will be ignored. Empty lines are simply ignored. No keywords, comments or special characters may occure after the `#data` keyword, which depicts the actual start of the dataset. Remark: the FST simply remembers the position of the first character after the `#data` keyword and reads the file from that position whenever needed without any additional format checking. It is your responsibility to ensure that the data part of the file contains nothing else than the proper number of numerical values separated by white space(s).

```
#datafile
#title Wisconsin Diagnostic Breast Center
; Features are computed from a digitized image of a fine needle;
; description ...

#features       30
#classes        2       357,212
#data
13.54
14.36
87.46
566.3
.
```

.

2. Gaussian distribution parameter files – depicted by a **.DST** file extension. These files are usually estimated from the .TRN file. This can be done using menu *File – Estimation(TRN->DST)*. These files contain gaussian distribution mean vectors and covariance matrices. This form of input data is required for use with all the feature selection methods (both sub-optimal and optimal) accessible from the *Methods* menu, where standard probabilistic distance measures (like Bhattacharyya etc.) are used to evaluate feature subsets. The file header is structured similarly to the header described above. Also the general restrictions and rules are similar. In this case the first line must contain the keyword `#datastructure` only, the `#classes` keyword is followed by a single value only and the data section of the file is divided to three sub-sections, beginning with `#apriori`, `#mean` and `#cov`. The meaning of this division is apparent.

```
#datastructure
#title Estimated from 'C:\data\wdbc.trn'
; Wed Jul  7 18:54:45 1999

#classes 2
#features 30
#apriori
 0.6274165202108963
 0.3725834797891037

#mean
 12.14652380952381
 17.91476190476189
 78.07540616246497
.
.

#cov
 3.170221722043872
 -0.2642603945960405
 20.95534960406635
 237.7691602059926
.
.
```

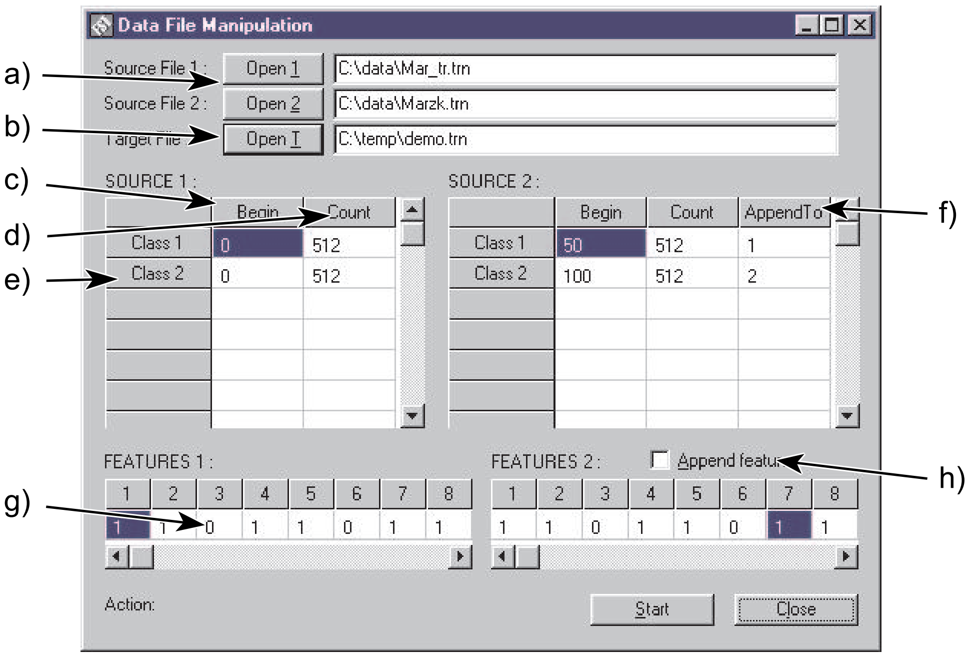
3. Approximation mixture model files – depicted by a **.APX** file extension. These files are usually generated from the .TRN file. This can be done using one of the approximation methods accessible through menu *Methods – Approximation Methods*. For description of these methods read sect. 4.7. These files can be used for classification purposes with the Pseudo-Bayes classifier accessible through menu *Classifiers – Pseudo-Bayes*.

```
#approximation data
#title Generated from C:\data\wdbc.trn
; selected features:  1 3 4 7 8 14 21 23 24 28
; by 'approximation' method with convergence limit 0.5 %
; initialization: Dogs & Rabbits
; performed on: Tue Sep 25 16:18:51 2001

#classes 2
#components 8
#features 30 1,0,1,1,0,0,1,1,0,0,0,0,0,1,0,0...

#apriori
 0.6274165202108963
 0.3725834797891037

#weights
 0.1757116985087308
 0.002801120448179272
 0.06112943177559165
.
.

#parameters
 9.638537006707566      0.7655738368590722
```

```
17.78022997031067      3.939819206179633
61.36174928636417      4.973341312296493
284.5855396949107      45.23822608619243
0.09591644237278267    0.0145545454326604
 .
 .
```

5.1 Data File Conversion

The FST offers a conversion filter to enable certain .TRN data file transformations. To use the filter open the Data File Manipulation dialog from menu *File* – *Manipulation(TRN)* (see picture 5.1). The filter allows among others: deleting sample intervals, deleting certain feature values from all the samples, reducing number of classes, but also joining two files to increase number of samples per class, or to add classes, or to add features etc.



Source File 1: C:\data\Mar_tr.trn
Source File 2: C:\data\Marzk.trn
Target File: C:\temp\demo.trn

SOURCE 1:

	Begin	Count
Class 1	0	512
Class 2	0	512

SOURCE 2:

	Begin	Count	AppendTo
Class 1	50	512	1
Class 2	100	512	2

FEATURES 1:

1	2	3	4	5	6	7	8
1	1	0	1	1	0	1	1

FEATURES 2: ☐ Append featu

1	2	3	4	5	6	7	8
1	1	0	1	1	0	1	1

Action:

Figure 5.1: Data File Manipulation dialog.

Let us demonstrate the filter functionality on two examples. First suppose we need to modify a 15-dimensional, 2-class speech data file 'speetr.trn' so that the resulting file contained samples 50 to 120 from the second class only and where features 4 and 7 were deleted from all the samples. Press the *Open 1* button (a) to open file 'speetr.trn'. Press the *Open T* button (b) to select a name for the target file to be created.

Table *Source 1* now contains a list of classes. Each row contains two values where the first one depicts the initial relative index of the first class sample to be included in the target file (initially 0) and the second depicts the number of samples to be copied (initially all the available ones). Rewrite the number of *Class 1* samples in the *Count* column (d) to 0. For the second class change the *Begin* (c) value to 50 and the number of *Class 2* samples in the *Count* column (d) to 70 (we want to copy samples between the 50th and 120th one). Table *Features 1* (g) can be used to mark features for deletion; rewrite values in columns 4 and 7 to 0. The actual file conversion can be now started by pressing the *Start* button. In case of successfull conversion you can try to open the new .trn file in some external text editor to see the commentary in the new file header.

The second example will be on joining two .TRN files. Press the *Open 1* button (a) to open the 2-class file 'speetr.trn', and *Open 2* button to open the second 2-class file 'speezk.trn'. The files contain compatible information, therefore their join makes sense. Press the *Open T* button (b) to select a name for the target file to be created.

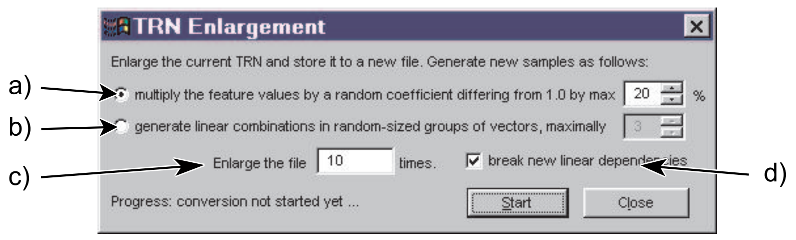
If the two files have equal number of features, the filter can be started immediately using the *Start* button. In that case the target file will contain 4 classes copied simply from the source files. However, suppose we would like to join the pairs of classes to get two new bigger ones. Mark the inter-file class relations using the *AppendTo* (f) column as follows: write 1 to the *Class 1, AppendTo* cell of the Source 2 table and 2 to the *Class 2, AppendTo* cell. These values serve as pointers to classes in the first datafile, to which the second datafile class data should be appended. Pressing the *Start* button now would result in generating a new 2-class file as specified.

Note that two-file joins can be done in a number of different ways, e.g. some of the *AppendTo* (f) indexes can be omitted. Another possibility is to append second file features to the features in the first file. If the same number of non-empty classes is specified in both source tables and these classes are linked using the *AppendTo* (f) indexes as well so that each pair of linked classes has equal number of samples, then it is possible to check the *Append features* (h) checkbox to request a target dataset in which each sample would contain all the feature values from both of the source files.

As could be seen, the Data File Manipulation dialog provides a rather powerful set of conversion filters that can be combined to produce a new file in a single conversion run.

–

The FST offers also a file filter allowing data file enlargement. Enlarging the training set may prove beneficial for the approximation mixture based methods, especially with training sets of unsufficient sizes.



TRN Enlargement

Enlarge the current TRN and store it to a new file. Generate new samples as follows:

- multiply the feature values by a random coefficient differing from 1.0 by max 20 %
- generate linear combinations in random-sized groups of vectors, maximally 3

Enlarge the file 10 times. [✓] break new linear dependencies

Progress: conversion not started yet ...

Start Close

Figure 5.2: Data File Enlargement Filter Dialog.

To invoke the filter use menu *File – Enlarge and Save TRN as...*. This will open the File Enlargement dialog as shown on figure 5.2. The filter works with the current TRN file, therefore the user must have opened the desired source TRN file before invoking the filter dialog. Two enlargement methods are available. Both of them enlarge the file by creating n copies, where n is the user-parameter specified in field 5.2c. Therefore for $n = 3$ the resulting file size would be 4x the original one, because the original data samples are copied as well.

The *constant multiplication* based method (see figure 5.2a) uses the user-specified parameter (let us denote it α) as follows: it multiplies each sample vector by a random scalar generated randomly on interval $< 1 - 0.01 * \alpha, 1 + 0.01 * \alpha >$. To prevent grouping of similar samples the new file will contain sequences of samples of the following type: $o_1, o_2, o_3, \ldots, c_{1,1}, c_{2,1}, c_{3,1}, \ldots, c_{1,2}, c_{2,2}, c_{3,2}, \ldots, c_{1,n}, c_{2,n}, c_{3,n}, \ldots$, i.e. the original sample o_1 will be used for creating $c_{1,i}, i = 1, \ldots n$ etc.

The *linear combination* based method (see figure 5.2b) uses its user-specified parameter (let us denote it β) to generate $n * class\_size$ new samples for each class by means of linear combination. Each new sample is generated as follows: first a number (let us denote it Γ) of source samples (let us denote them $\mathbf{y}_i, i = 1, \ldots, \Gamma$) is determined randomly on interval $< 2, \beta >$. Γ original samples are then selected randomly from the current class. Following that, Γ random non-zero weights (let us denote them $\Delta_i, i = 1, \ldots, \Gamma$) are generated to fulfill $\sum_{i=1}^{\Gamma} \Delta_i = 1$. One new data sample $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ is then generated so that $x_j = \sum_{i=1}^{\Gamma} y_i * \Delta_i, j = 1, \ldots, D$. The whole procedure is repeated separately for each new sample with newly regenerated Γ, $\mathbf{y}$'s, Δ's etc.

Both of the methods as described yield new samples linearly dependent on the original ones. To prevent linear dependencies and to add a slight noise to the data check the (d) checkbox. Method (a) will then use the α parameter to generate new random multipliers for each single value separately. Method (b) will make use of the same α parameter as well, to additionally multiply each single value obtained in a way described above.

Of course none of these enlarging procedures can add any relevant information that had not been already contained in the original dataset. However it may affect the mixture model training process, which is usually strongly unreliable if the number of samples in each class is less than 10x–100x the dimensionality D. Exact recommendation regarding the sufficient training data size can be hardly stated, as it depends on the particular problem properties. In general the more training samples are available the better and more reliable results can be expected.

Chapter 6

Implementation Remarks

The biggest effort has been put on quality of the kernel functions. It is not possible to describe all significant code properties, however some of them could be noted. Generally all the feature selection algorithms are computationally demanding and spend usually a lot of computational time. FST code is therefore optimised to avoid unnecessary slow-downs whenever possible, e.g. while accessing n-dimensional memory structures etc. A particularly important part of the code is the implementation of criterion functions, because the algorithms usually spend most of their time evaluating criterion values on otherwise quickly generated candidate subsets. Despite the intention to achieve highest code speed possible we did not omit basic error-recovery mechanisms.

Many subset search algorithms are theoretically defined in two versions: "bottom-up" and "top-down". The first version starts with an empty feature subset; features are then added to that subset to achieve the required cardinality. In the second case the algorithm starts with a set of all features that is reduced in subsequent algorithm stages. Many algorithms then allow combining of both "bottom-up" and "top-down" steps to achieve better search efficiency. Moreover, these algorithm steps often involve testing groups of features instead of single ones only. To enable implementation of all kinds of often complicated algorithms (as e.g. the oscillating search, combining effectively its basic approach with different other methods) we needed to find a flexible and fast way of processing the features. Our implementation of each method is therefore built around a working vector of D feature state *identifiers*. Each *identifier* value depicts, whether the given feature is currently selected or not. The *identifier* values can depict even other feature states depending on currently processed algorithm sub-steps, e.g. whether the feature is to be fixed and included in the final result or whether it is selected temporarily only. We use different positive values to depict selected features, while other values depict features outside the current candidate subset. Mentioning the existence of such a technological "detail" as the *identifier* vector is has a good reason: a simple redefinition of *identifier* value meanings allowes us to change instantly the direction of search as well as other algorithm properties. In this way we have been able to define a virtually unique code implementing both the "bottom-up" and "top-down" search for every algorithm known to exist in two search direction versions. It should be noted that this approach does not cause any measurable algorithm slow-down. Moreover, the code becomes significantly simpler and lucid in this way, what results in shorter debugging times as well.

As an experimental platform the FST has been serving for testing and development of new methods. A lot of work can be done in the future. We are considering to implement a set of parallelised procedures, porting the whole system to Visual C++ to enable creating a better user interface, or extending the applicability of the whole package. We plan to use the FST as a base for building a complex, user-oriented system with decision support based on a simple expert system.

Chapter 7

Known Problems

The software is presented "as-is". Most of the kernel code is supposed reliable. However, the same cannot be said about the user interface. Here you can find a list of known problems. Some of them will be corrected in next software versions. Unfortunately a lot of problems follow from weak Powersoft Optima++ library implementation, what was not apparent at the time we started writing the software in 1996. These problems cannot be compensated on our side now, mostly because Sybase decided to stop any further Optima development some time ago. It should be also noted, that the Optima++ compiler produces a native Windows 95 code, which should run without problems on every current Windows version, however we cannot guarantee that e.g. the latest Windows XP or so would not cause unexpected problems. We are considering to port the whole software to Visual C++ environment, however, this is a long-term goal.

Kernel problems:

- From implemented criterion functions, only the Bhattacharyya should be considered fully reliable because it has been checked thoroughly and its results have been verified using different implementations on different systems. Also the Divergence caused no problems so far. The other criterion functions occasionally suffered numerical problems causing isolated non-monotonical behavior. Because of speed requirements no internal numerical overflow checking is implemented, therefore be prepared to observe problematic results with certain problematic datasets.

- File import filters assume line lenghts not exceeding ca. 1000 characters (including whitespace). If your source data are organised e.g. in tables where each row is stored as one textual line exceeding the limit, the file can be opened seemingly correctly but final results may not be correct. Older FST versions also occasionally generated (using the File Manipulation dialog) files causing Import Error messages – in that case the problem followed from too long comment lines in file headers. Such lines can be simply removed manually to make the file usable.

- The system is meant to be used for solving problems of reasonable dimensionalities. That means we have used it successfully with datasets having several hundreds of dimensions. If you attempt to process data of tens of thousands of dimensions, expect problems. Memory allocation errors may appear, some methods will refuse to work, or the gaussian mixture parameter estimation will fail. Generally the approximation methods should be usable even with these extreme dimensionalities. Moreover, the lists of features are indexed internally by int variables, thus the system cannot work with more than about 32000 features. With extreme dimensionalities the generated DST and APX files may contain lines too long to be readable by the FST.

- Some seemingly legal algorithm settings may result in error messages – this does not affect the usability of the system, because these errors usually indicate that a particular algorithm should not be used in that case – i.e. using advanced methods for selecting single feature only etc.

- Approximation methods may cause the application crash during the final stage of computation. The problem has been corrected and should not appear any more, however we cannot guarantee it without further testing.

- Be aware of weakly-conditioned covariance matrices resulting from e.g. loose datasets containing lots of 0 values. Although the algorithms yield reasonably looking results, it may be a nonsense. However, this is more or less a theoretical problem.

- Feature indexes should start from 0 everywhere. However a forgotten case of indexes starting from 1 may be discovered. We are currently checking this. This should not happen with any of the standard methods.

- Minor problems regarding more or less the programming culture – without straight influence on the FST functionality.

User interface problems:

- Serious application crashes should be rare. However, it is a good idea to restart the application occasionally, especially before long-lasting computations. The user interface components taken from Optima++ may refuse to process messages after some longer application run, what results in a behavior where everything looks OK, algorithms compute, but no new results appear in neither the *Results* or *Console* windows.
- The *Console* window may occasionally stop to accept new input having filled about 33% of its memory capacity. This means algorithms would continue running but no new information would appear in the *Console*. In that case restart the application.
- The *Results* window may occasionally stop to accept new input as well. Restart the application.
- The *Approximation Graph* window is not properly implemented, therefore it looses its contents when overlapped by other windows.
- The *File – Manipulation* dialog may occasionally refuse to complete the conversion process properly. Ensure no table field (inside the tables nor in the feature lists) remained selected for editation (marked by a bevelled field border). If it does not help, try to restart the application.
- Some method dialog setting combinations may still be found to be inconsistent with actual method requirements, what would result in error messages. In such cases, the method algorithm is supposed to be "the one to make the decision" – simply do not use such setting combination any more.
- Graph to EMF export filter produces Extended Windows Metafiles that may not be openable correctly with certain graphical applications. This is a Powersoft library problem that cannot be resolved on our side.
- Keyboard-based control may not be correctly functional everywhere, therefore the software may be hard to use without a mouse (or other pointing device) connected.
- Certain text messages and inscriptions contain traces of "czenglish", i.e. improperly translated czech words or sentences. That holds for this manual as well.

Chapter 8

Comments And Acknowledgements

The work on the FST started as a part of my PhD thesis. About 60% of the kernel code had been programmed at that time to verify new methods proposed in the thesis and to compare them with traditional ones. Later, the code had been found useful also for other purposes what resulted in further work. The kernel has been extended and improved throughout the years. The windows user interface has been added later at the time when Powersoft Optima++ constituted the only Delphi-like tool for fast C++ user interface programming. Consequences of this decision are discussed in the previous chapter.

All the code including all its ups and downs has been programmed by me so far. Such a single-programmer work may be considered advantageous because of consistency of all the method implementations – the system as a whole thus may be considered well suitable for method comparison purposes. On the other hand there was not enough time to build the user interface to as good state as we would like to. Also the system as a whole could not be tested thoroughly and different problems may become apparent any time.

Although the programming work has been done by a single person, the ever evolving concept, choice of algorithms, overall architecture and user interface properties have been thoroughly consulted within a group of consultants. The main consultant and project co-leader is prof. Pavel Pudil, Head of the Department of Pattern Recognition at the Institute of Information Theory and Automation, Prague. Any enquires regarding the FST should therefore be directed either to the author's address (see the front page) or even better to prof. Pudil, `pudil@utia.cas.cz` or `pudil@fm.vse.cz`.

We appretiate all the comments and help provided by our colleagues, especially from: Dr. Jana Novovičová, Dr. Jiří Grim, Karel Fuka, Pavel Žid, Jan Plešingr and Dr. Michal Haindl.

The work has been supported by grants of the Czech Ministry of Education CEZ:J18/98:31600001, LNOOB107 (EuroMISE - Kardio centrum), Aktion 29p7 and the Grant Agency of the Czech Republic No 402/01/0981.

The FST software has been used for verifying and evaluating new methods, demonstration and teaching purposes as well as for solving real-world problems. A list of publications discussing either the software itself or mentioning its results can be found on the following pages.

Petr Somol
October 2001

Bibliography

[1] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall, 1982.

[2] K. Fukunaga. *Introduction to Statistical Pattern Recognition: 2nd edition.* Academic Press, Inc., 1990.

[3] Ch.M. Bishop, G. Hinton *Neural Networks for Pattern Recognition.* Clarendon Press, November 1995.

[4] A. K. Jain and D. Zongker. Feature selection: Evaluation, application and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:153 – 158, 1997.

[5] J. Kittler, P. Pudil and P. Somol. Advances in Statistical Feature Selection. Tutorial for ICAPR 2001, Rio de Janeiro – *Lecture Notes in Computer Science, Springer Verlag*, Volume 2013, pp. 425, 2001.

[6] M. Kudo and J. Sklansky. A comparative evaluation of medium- and large-scale feature selectors for pattern classifiers. In *Proceedings of the 1st IAPR TC1 International Workshop on Statistical Techniques in Pattern Recognition*, pp. 91–96, Prague, 1997.

[7] M. Kudo and J. Sklansky. On numerical evaluation of maximum-likelihood estimates for finite mixtures of distributions. *Kybernetika*, 34(4):429–434, 1998.

[8] M. Kudo, P. Somol, P. Pudil, M. Shimbo and J. Sklansky. Comparison of classifier-specific feature selection algorithms. In Proceedings of 3rd IAPR International Workshop STIPR 2000, Alicante, Spain – *Lecture Notes in Computer Science, Springer Verlag*, Volume 1876, pp. 677, 2000.

[9] H. A. Mayer, P. Somol, R. Huber, and P. Pudil. Improving statistical measures of feature subsets by conventional and evolutionary approaches. In Proceedings

of 3rd IAPR International Workshop STIPR 2000, Alicante, Spain – *Lecture Notes in Computer Science, Springer Verlag*, Volume 1876, pp. 77, 2000

[10] H. A. Mayer and P. Somol. Conventional and Evolutionary Feature Selection of SAR Data Using a Filter Approach. In *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida*, 2000.

[11] P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C-26:917–922, September 1977.

[12] J. Novovičová and P. Pudil. *Dealing with Complexity - A Neural Networks Approach*, chapter Feature Selection in Statistical Pattern Recognition by Modified Model with Latent Structure. Springer Verlag, Berlin, 1997.

[13] J. Novovičová, P. Pudil, and J. Kittler. Divergence based feature selection for multimodal class densities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:218–223, 1995.

[14] P. Pudil and J. Novovičová. *Feature Transformation and Subset Selection*, chapter Novel Methods for Feature Subset Selection with Respect to Problem Knowledge. Kluwer, 1998.

[15] P. Pudil, J. Novovičová, N. Choakjarernwanit, and J. Kittler. Feature selection based on the approximation of class densities by finite mixtures of special type. *Pattern Recognition*, 28(9):1389–1398, 1995.

[16] P. Pudil, J. Novovičová, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119–1125, November 1994.

[17] P. Pudil, J. Novovičová, and P. Somol. Feature selection in statistical pattern recognition via model with latent structure. In *Proceedings of the IEEE European Workshop on Computer-Intensive Methods in Control and Signal Processing*, pp. 217–222, Prague, 1996.

[18] P. Pudil, J. Novovičová, P. Somol, and R. Vrňata. Conceptual base of feature selection consulting system. In *Proceedings of the 1st IAPR TC1 International Workshop on Statistical Techniques in Pattern Recognition*, pp. 125–134, Prague, 1997.

[19] P. Pudil, J. Novovičová, P. Somol, and R. Vrňata. *Statistical Techniques in Pattern Recognition*, chapter Feature Selection Expert – A User Oriented Approach. Springer Verlag, Berlin, 1998.

[20] P. Pudil and J. Novovičová. Novel methods for subset selection with respect to problem knowledge. *IEEE Transactions on Intelligent Systems (Special Issue on Feature Transformation and Subset Selection*, pp. 66–74, March/April 1998.

[21] P. Pudil, J. Novovičová and P. Somol. Prior Knowledge Guided Approach to Statistical Feature Selection, To appear in *Pattern Recognition Letters, Elsevier, Netherlands*, 2001

[22] P. Somol, P. Pudil, J. Novovičová, and P. Paclík. Adaptive floating search methods in feature selection. *Pattern Recognition Letters*, 20(11-13):1157–1163, December 1999.

[23] P. Somol, P. Pudil, F. J. Ferri, and J. Kittler. Fast branch & bound algorithm for feature selection. *Invited paper for the 4th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida, 2000.

[24] P. Somol and P. Pudil. Oscillating Search Algorithms For Feature Selection. In: *the 15th IAPR International Conference on Pattern Recognition, Proceedings, Barcelona, Spain*, Vol. 2, pp. 406–409, 2000.

[25] P. Somol, P. Pudil and J. Grim. Branch & Bound Algorithm with Partial Prediction For Use with Recursive and Non-Recursive Criterion Forms. Proceedings Int. Conf. on Advances in Pattern Recognition, Rio de Janeiro – *Lecture Notes in Computer Science, Springer Verlag*, Volume 2013, pp. 230, 2001.

[26] P. Somol and P. Pudil. Feature Selection Toolbox as a Multi-Purpose Tool in Statistical Pattern Recognition. In: *Proceedings of WPRIS 2001*, Setubal, Portugal, 2001.

[27] B. Yu and B. Yuan. A more efficient branch and bound algorithm for feature selection. *Pattern Recognition*, 26:883–889, 1993.