# Tutorial for **InfoSel++**: **Info**rmation Based Feature **Sel**ection C++ Library

*Authors:*
Jacek BIESIADA, Włodzisław DUCH, Adam KACHEL, Marcin BLACHNIK

Jacek Biesiada - **Jacek.Biesiada@polsl.pl**
Marcin Blachnik - **Marcin.Blachnik@polsl.pl**
Włodzisław Duch - **wduch@is.umk.pl or Google: Duch**
Adam Kachel - **Adam.Kachel@polsl.pl**

June 9, 2010

# Contents

# Chapter 1

# Introduction

InfoSel++ is a library of the C++ language based on the information theory developed in the Division of Computer Science at Silesian University of Technology and Department of Informatics at Nicolaus Copernicus University in Toruń.

Our main goal is to provide capability to do the following tasks easily and quickly:

1. Implement and test new ideas and variants of feature ranking algorithms and selection algorithms.
2. Generate simple statistics about attributes.
3. Perform experiments with hybrid algorithms such as Markov Blanket, Fast Correlation Based Filter(FCBF), and others.
4. Test algorithms on various different datasets (*e.g.*, UCI Repository or GDataset Repository).
5. Integrate Infosel++ into other C++ libraries, as an option.

Feature selection is a fundamental problem in tasks such as classification, data mining, image processing, and conceptual learning. Feature selection problems refer to the task of identifying and selecting a useful subset of features. This subset is then used to represent the target from a larger set which may have included redundant or even irrelevant information.

Algorithms of feature subset selection can be classified into three categories based on whether algorithms of feature selections are done independently of the learning algorithm or not. The first technique is called the filter approach, where feature selection is performed independently of the learning algorithm. When the goal is the maximization of the accuracy of a given feature subset, we often use, for the good of the subset, relevance functions and learning algorithms. This defines the wrapper approach. The last type of selection approach is the embedded approach, where only classification algorithms with different types of heuristic organization are used to search the important subset of features.

During the process of designing the selection algorithm, we must take into consideration four basic issues that determine the nature of the search process:

1. a starting point in the features space,
2. organization of the search,

3. an evaluation strategy of the selected subset,

4. a criterion for ending the search.

This library is based on information theory and especially on probability, which allow the development of different types of feature selection algorithms.

The software described in this document comes free of charge for non-commercial use, and with no warranty of any kind. A license is required for commercial users. Full details of the conditions of use are available via:

http://kzi.polsl.pl/~jbiesiada/Infosel/index.html

where updates for the software will also be made available. Please send bug reports, requests for new features, and feedback to Jacek Biesiada or Adam Kachel at the e-mail address on the title page.

# Chapter 2

# Getting Started: Installing and Running the Software

Infosel++ library has been tested in Linux (under gcc compilator) and Windows.

The library comes in a gzipped tar file. Files can be extracted from the appropriate .tar.gz file by typing (for example):

gunzip infosel++.xx.tar.gz
tar -xvf infosel++.xx.tar or once
tar -xvzf infosel++.xx.tar.gz

at the command line, where xx is the version number. This will create a new directory called infosel++.xx. Change to this directory before attempting to compile a library and executable program.

To compile executable files, change the main directory to the directory called "build/GNUC++." There will be a makefile called a "_build.sh"; type in the a command line ". _build.sh" and this should compile the library and create a main executable infosel++.g.exe in the directory "dist/bin." (All paths refer to the main directory of the project.)

You have the possibility of running infosel++.g.exe in three ways. Either choose to type infosel++.g.exe, and you will run a program interactively. Or you may choose to use either infosel++.g.exe as a command line application, or use the application in script mode ($-s$ parameter).

To analyze your own data, you must prepare an input file in the appropriate format. All values should be numerical and separated by commas (csv format). The program does not handle string values. **Note: The program does not handle missing values. You have to inpute them before you begin the analysis. Most of the problems that arise stem from incorrectly formatted input data**. Examples of input files are in the directory "prj/libdocs/examples/data."

## 2.1 Infosel++ as a command line application

The most important and useful parameter is a "$-h$" [help], which provides short descriptions of available options. This parameter can also be used as follows:

```
prj/libdocs/examples/data>infosel++.g.exe -h infosel.help
```

In this case, all infosel++ help will be written in the "infosel.help" file.

The standard structure of command line usage of infosel++ is:

**Example 1.**

```
prj/libdocs/examples/data>infosel++.g.exe -in iris.dat -out irisdata.out -execall
```

In this case, the infosel++.g.exe application will process all implemented algorithms (*−execall* parameter) in the standard algorithm package. Results will be written in "*irisdata.out*" (*−out* parameter), and the calculations will be performed with the well-known dataset Iris (*−in* parameter)(where names of flowers were converted to integer values). The user of the infosel++ application can find short and concise descriptions of parameters in appendix C. The longest version of this appendix is found below:

```
-s|scr|script file_name
  Performs processing of a given script file.
```

This parameter is to allow use of an application in script mode, where all parameters can be stored in one file. A simple example of an input file will be shown later.

```
-in|input|data file_name
  Sets up input data file.

-out|output|report file_name
  Sets up output report file.
```

These two parameters allow a user of the infosel++ application to set up input and output files with data and with results. Example 2.1 shows how we can use both these parameters.

```
-p|prec|precision real_val
  Sets up real data precision.
```

"-p" parameter represents precision of calculation and number of printed significant digits. If we choose "-p 0.001," it means that the output values will have three significant digits.

```
-pm|pmeth|partition meth_name|meth_name(val_list)
  Sets up data partition method.
```

This parameter sets up the method of discretization input data. Below are five possible choices:

1. **FineGrain** – This discretization method is used for data processed before using the infosel++ application (also called external processing or external discretization). All data points are treated as a unique value (usually used with external preprocessing wrapper).
2. **EquiWidth** – This discretization method divides data into a k-specified number of separate bins of equal width. The default number is 24. For selecting the number of bins, we can use several rules of thumb; for example, the value of k is the $\sqrt{n}$ rounded to the nearest whole number, where $n$ is the number of instances in the dataset.
3. **EquiUniqs** – This method of discretization performs a partitioning of the domain into n parts of equal number of unique values.
4. **EquiFreqs** – This discretization method divides data into a k-specified number of separate bins of equal frequency.

5. **MaxUniDif** – This method of discretization performs a partitioning of the domain into n parts separated at maximal unique value differences.

6. **MaxFrqDif** – This method of discretization performs a partitioning of the domain into n parts separated at maximal data frequency differences.

```
-cp|cpos|classpos int_val
  Sets up classes column position.
```

The classes column generally can be in any column in the dataset, and parameter "-cp" gives us the possibility of informing the infosel++ application where the column of the classes is.

```
-lc|lastc|classislast bool_val
  Sets up a flag if classes column is last.
```

Most datasets in the UCI repository have their class column in the last column and the default value of the "-lc" parameter is true.

```
-e|excl|excluded str_list
  Sets up labels for excluded classes.
```

This parameter makes possible the exclusion of some classes from calculation. For example, if we have four classes $0, 1, 2, 3$, we can limit calculation to 0 and 2, and we can exclude 1 and 3 (-e 1,3).

```
-ne|note|notexcluded str_list
  Sets up labels for non-excluded classes.
```

This parameter is a negation of the previous parameter (not -e).

```
-m|merg|merged str_list
  Sets up labels for merged classes.
```

Some times in our calculation we want to perform a calculation of one class vs. all others. This parameter gives us this possibility. For example, if we have four classes and we want to perform a calculation of 1 vs. joint(2,3,4), we can merge class 2,3,4 (-m 2,3,4).

```
-nm|notm|notmerged str_list
  Sets up labels for non-merged classes.
```

This parameter is a negation of the previous parameter (not -m).

```
-exe|exec|exeselalg alg_tag|alg_tag(val_list)
  Executes a given selection algorithm.
```

This parameter gives us the possibility of selecting the selection algorithm and specifying input parameters. For example, for mifs algorithm with $\beta = 0.7$ we can specify: -exe mifs(beta=0.7).

```
-execall
  Executes all selection algorithms.
```

This parameter allows us to execute all selection algorithms with default parameters.

```
-silent
  Performs processing without displaying output info.
```

The application will be processed silently.

```
-verbose
  Performs processing displaying more output info.
```

The application will be processed in verbose mode. In this mode, we will have more information about progress of the calculation.

```
-break
  Breaks processing commands.
```

## 2.2 Example of Infosel++ application input script

Infosel++ can also work in script mode. Several examples of the input script are stored in the directory **prj/libdocs/examples/scripts**. The script below performs the calculation for a datafile *sample4a.dat*. The results will be stored in *sample4a.rep*. Precision of calculation was set up for 0.001 and the class vector is in the last column. Input data will be discretized with equal-width discretization in 24 bins. Battiti's [**?**] algorithm will be performed twice, once with $\beta = 0.7$ and a second time with $\beta = 1.0$.

```
prj/libdocs/examples/scripts>more sample0.inp

output = "sample4a.rep";      # name of the output file: sample4a.rep, default: "default.rep"
input = "sample4a.dat";       # name of the input file (with data): sample4.dat,
                                                          default: "default.dat"

precision = 0.001;            # setup of precision input/output data
classislast = true;           # setup of classislat flaq, default: true
partition = equiwidth(24);    # selection of discretization method, here: equal width with 24 bins

exec( MIFS(0.7, 1.0) );       # selection and execution of MIFS algorithm
                                                     with $\beta=0.7$ and $\beta=1.0$
```

The calculation is performed by using the following command:

```
prj/libdocs/examples/scripts>infosel.g.exe -s sample0.inp
```

## 2.3 Example of Infosel++ application session

The simplest way to use an infosel++ application is in the interactive mode. Below we have an example of a computational session. Calculations are performed with *corral.data* and are usually done in nine steps. The first two steps are related to input and output files. After this we have four decisions to make as to how data should be processed: where is the class vector, exclusion of class, mergence of class, and precision of calculation. The next two questions regard the selection of the discretization algorithm and the selection algorithm. The last questions refer to the parameters of the selection algorithm. After calculating, the application poses formal questions about the repetition of calculation with other algorithms.

```
prj/libdocs/examples/data> ./infosel++.g.exe

  |
  | InfoSel++
  | Information Theoretic Feature Selection Program
  | by Adam Kachel & Jacek Biesiada
  |

Output report file name: corral.out
opening the output report file....
Input data file name: corral.data

Is the classes column last (y/n) ? y
Exclusion classes set (space-separated):
Mergence classes set (space-separated):
Real data precision (e.g. 0.001): 0.001

Partition methods:
[0] - FineGrain
[1] - EquiWidth
[2] - EquiUniqs
[3] - EquiFreqs
[4] - MaxUniDif
[5] - MaxFrqDif
>> choose item: 0
opening the input data file....

Selection algorithms:

[mbr    ] MarkovBlanket:  [bdr    ] Measure:BD-Ran  [mifsu  ] Mifs:MIFSU
[chq    ] Chisquare:Chin  [kdr    ] Measure:KD-Ran  [fid    ] Mrmr:FID
[pcc    ] Correl:Pearson  [klr    ] Measure:KLD-Ra  [fiq    ] Mrmr:FIQ
[fcbf   ] Fcbf:FastCBF    [mdr    ] Measure:MD-Ran  [mid    ] Mrmr:MID
[fsc    ] Fscore:Fscore   [sdr    ] Measure:SD-Ran  [miq    ] Mrmr:MIQ
[gdd    ] Gddist:GDdist   [mi     ] Mi:MInfo        [suc    ] Suc:SUCcoeff
[ks_cbf ] Kscbf:Kolmogor  [amifs  ] Mifs:AMIFS      [tsc    ] Tscore:Tscore
[ksc_cbf] Kscbf:Kolmogor  [mifs   ] Mifs:MIFS

>> choose space-separated items (none = all): mi
Do you want to set up all/main/no algorithm parameters (a/m/n) ? n

MInfo algorithm:

running....
Executed successfully 1 algorithm(s)

Do you want to choose other algorithms (y/n) ? n
Do you want to process another data file (y/n) ? n
   <<<< END OF PROGRAM >>>>

prj/libdocs/examples/data>
```

# Chapter 3

# Implemented Algorithms

## 3.1 Filter algorithm

.

The structure of a generalized filter, wrapper, and hybrid algorithm was proposed in [**?**]. Algorithms within the filter model are illustrated through a generalized filter algorithm (shown in figure 3.1). For a given data set D, the algorithm starts the search from a given subset S (an empty set, a full set, or any randomly selected subset) and searches through the features space by means of a particular search strategy. Each generated subset S is evaluated by an independent measure M and compared with the previous best one. If it is found to be better, it is regarded as the current best subset. The search iterates until a predefined stopping criterion such as an information measure or correlation measure is reached. The algorithm outputs the last best subset $S_{best}$ as the final result. By varying the search strategies and evaluation measures used in steps 5 and 6 in the algorithm, we can design different individual algorithms within the filter model. Since the filter model applies independent evaluation criteria (distance measure, information measure, consistency measure, and dependency measure) without involving any mining algorithm, it does not inherit any bias of a mining algorithm, and it is also computationally efficient in comparison with wrapper or hybrid algorithms.

## 3.2 Implemented Algorithms List

The following feature selection algorithms were implemented in Infosel++, and have been split into four distinctive groups: ranking, redundancy shifting ranking, redundancy removal ranking, and others.

**1. Ranking method**

1. CC (pcc) - Pearson Correlation Coefficient [**?**, **?**]
2. t-score (tsc) - t-score statistics (for two-class problem) [**?**, **?**]
3. F-score (fsc) - F-score statistics (for multi-class problem) [**?**]
4. $\chi^2$ (chq) - $\chi^2$-score statistics [**?**, **?**]
5. MI (mi) - Mutual Information [**?**, **?**]
6. SUC (suc) - Symmetrical Uncertainty Coefficient [**?**, **?**]

**Filter Algorithm**
**input:** $D(F_0, F_1, ..., F_{N-1})$ // a training data set with N features
$S$ // a subset from which to Start the search (Starting subset)
$\delta$ // a stopping criterion
**output:** $F_{best}$ // an optimal Final subset

```
01    begin
02          initialize: F_best = S;
03          γ_best = eval(S, D, M); // evaluate S by an independent measure M
04          do begin
05                S = generate(D);// generate a subset for evaluation
06                γ = eval(S, D, M); // evaluate the current subset S by M
07                   if (γ is better than γ_best)
08                      γ_best = γ;
09                      Sbest = S;
10          end until (δ is reached);
11          return F_best = S;
12    end;
```

Figure 3.1: Generalized filter algorithm.

7. MDr (mdr) - Matusita Distance ranking[**?**, **?**]
8. KDr (kdr) - Kolmogorov Distance ranking[**?**, **?**]
9. KLDr (kldr) - Kullback Distance ranking [**?**, **?**]
10. BDr (bdr) - Bhattacharatyya Distance ranking [**?**, **?**]
11. SDr (sdr) - Samon Distance ranking [**?**, **?**]

**2. Redundancy shifting ranking**

1. MIFS (mifs) - Mutual Information Feature Selection [**?**]
2. MIFS-U (mifsu) - Mutual Information Feature Selection under Uniform Information Distribution [**?**].
3. AMIFS (amifsu) - Adaptive Mutual Information Feature Selection [**?**]
4. MID (mid) - Mutual Information Difference [**?**]
5. MIQ (miq) - Mutual Information Quotient [**?**]
6. FCD (fcd) - F-test Correlation Difference [**?**]
7. FCQ (fcq) - F-test Correlation Quotient [**?**]

**3. Redundancy removal ranking**

1. FCBF (fcbf) - Fast Correlation Based Filter [**?**]

2. K-S CBF (*ks_cbf*) - A Kolmogorov-Smirnov Correlation-Based Filter [**?**, **?**]
3. K-SC CBF (*ksc_cbf*) - A Kolmogorov-Smirnov Class Correlation-Based Filter [**?**]
4. PRBF (prb) - Pearson Redundancy Based Filter [**?**]

**4. Others**

1. Markov Blanket approximation (mbr) [**?**, **?**]
2. GD-distance ranking (gdd) [**?**]

Acronyms in parentheses are used in the Infosel++ menu system. Using a ranking filter is the simplest method for feature ordering based on criteria such as dependency measure, information, distance, and consistency measure. Usually ranking filters are used as an initial step in analysis where we want to select a subset of informative (relevant) features, especially in case of highly multidimensional datasets. The main disadvantage of a ranking filter is an inability to reject redundant features.

## 3.3 Ranking method

Below we can find a list of implemented feature relevancy indices used in ranking algorithms. First is the commonly used Pearson correlation coefficient (pcc):

$$\rho(f_i, f_s) = \sum_{lk} \frac{cov(f_{ik}, f_{sl})}{\sigma_i \sigma_s} = \sum_{lk} \frac{(f_{ik} - \bar{f}_i)(f_{sl} - \bar{f}_s)}{\sigma_i \sigma_s} \tag{3.1}$$

where $\bar{f}_s(\bar{f}_i)$ and $\sigma_i(\sigma_s)$ represent averages and standard deviations, respectively. These two measurements are defined in Eq. 3.2.

$$\begin{aligned} \bar{f}_i &= \frac{1}{n} \sum_{j=1}^{n} f_{ij} \\ \sigma_i &= \sqrt{\frac{1}{n} \sum_{j=1}^{n} (f_{ij} - \bar{f}_i)^2} \end{aligned} \tag{3.2}$$

The next two measures are also widely used in statistics, namely t-score (tsc, Eq.3.3) and F-score (fsc,Eq. 3.4). t-score is a separability measure used for a two-class problem, and F-score is used in a multi-class problem. F-score will reduce to t-score in two-class problems, with the relation $F = t^2$.

The t-score measure is a separability between two-classes calculated as the difference between the average value in class $1\left(\bar{f}_{i1}\right)$ and class $2\left(\bar{f}_{i2}\right)$, and also takes into account dispersions described as average standard deviations for class 1 $\left(\frac{s_{i1}^2}{n_1}\right)$ and class 2 $\left(\frac{s_{i2}^2}{n_2}\right)$. To calculate the separability for multi-class data, sometimes we can use the average value of the t-score for all pairs of classes.

$$t(f_i, C) = \frac{\bar{f}_{i1} - \bar{f}_{i2}}{\sqrt{\frac{s_{i1}^2}{n_1} + \frac{s_{i2}^2}{n_2}}} \tag{3.3}$$

F-score, the more general measure for describing separability for multi-class problems, was mentioned above, and is defined as a factor of interclass variance (numerator in Eq. 3.4) and intraclass variance (denominator in Eq. 3.4):

$$F(C, f_i) \;=\; \left[\sum_k n_k \left(\bar{f}_{ik} - \bar{f}_i\right)^2 / \left(K-1\right)\right] \Big/ \sigma^2 \tag{3.4}$$

The standard deviation for feature $f_i$ is defined:

$$\sigma^2 \;=\; \sigma^2(f_i) \;=\; \left[\sum_k \left(n_k - 1\right) \sigma_{ik}^2\right] \Big/ \left(n - K\right); \tag{3.5}$$

where $\sigma_{ki}^2$ is a standard deviation within a class, $n_k$ is the number of elements in class k, $k$ is the index of class, $n$ is the number of all data, and $K$ is the number of classes.

$$\sigma_{ik}^2 \;=\; \sigma^2(f_{ik}) \;=\; \frac{1}{n_k - 1} \sum_j^n \left(f_{ikj} - \bar{f}_{ik}\right)^2 \tag{3.6}$$

The average value in a particular class $\bar{f}_{ik}$ is defined as:

$$\bar{f}_{ik} \;=\; \frac{1}{n_k} \sum_j^{n_k} f_{ikj} \tag{3.7}$$

The first relevance index implemented is based on probability and is called $\chi^2$ (Eq. 3.8). Many such measures based on probability are equal to 0 when two features are statistically independent. This measure and Mutual Information (MI) are not normalized and vary in range $[0, \infty)$.

$$\chi^2(f_i, C) = \sum_{jk} \frac{\left(p(f_i = x_j, C_k) - p(f_i = x_j) * p(C_k)\right)^2}{p(f_i = x_j) * p(C_k)} \tag{3.8}$$

Mutual Information measure and Symmetrical Uncertainly Coefficient [**?**, **?**] can be defined as a function of Shannon's entropy. The next four equations describe a Shannon's entropy for class (Eq. 3.9), features (Eq. 3.10), joint entropy (Eq. 3.11), and the last conditional entropy (Eq. 3.12) as a function of joint entropy (between feature and class), and feature entropy.

$$H(C) = -\sum_{i=1}^{K} p(C_i) \log_2 p(C_i) \tag{3.9}$$

$$H(f_i) = -\sum_x p(f_i = x) \log_2 p(f_i = x) \tag{3.10}$$

$$H(C, f_i) = -\sum_{x,i=1}^{K} p(f_i = x, C_i) \log_2 p(f_i = x, C_i) \tag{3.11}$$

$$H(C|f_i) = H(C, f_i) - H(f_i) \tag{3.12}$$

MI (mi) - Mutual Information [**?**, **?**]

$$MI(C, f_i) = H(C) + H(f_i) - H(C, f_i) \tag{3.13}$$

SUC (suc) - Symmetrical Uncertainly Coefficient [**?**, **?**]

$$SUC(C, f_i) \;=\; 2.0 \frac{MI(C, f_i)}{(H(f_i) + H(C))} \tag{3.14}$$

## 3.4 Measure of relevance based on comparison of probability distribution.

All measures below are determined by the difference between the joint probability (class and feature) and the product of marginal probability for class and feature. And feature ($f_i$) is statistically independent from class C, if $P(C = c_l, f_i = f_{iy}) - P(C = c_l)P(f_i = f_{iy}) \;=\; 0$.

1. MDr (mdr) - Matusita Distance [**?**, **?**]

$$MD(C, f_i) \;=\; \left\{ \sum_{l=1}^{L} \sum_{y \in Y} \left[ \sqrt{P(C = c_l, f_i = f_{iy})} - \sqrt{P(C = c_l)P(f_i = f_{iy})} \right]^2 \right\}^{\frac{1}{2}} \tag{3.15}$$

2. KDr (kdr) - Kolmogorov Distance [**?**, **?**]

$$KD(C, f_i) \;=\; \sum_{l=1}^{L} \sum_{y \in Y} \left| P(C = c_l, f_i = f_{iy}) - P(C = c_l)P(f_i = f_{iy}) \right| \tag{3.16}$$

3. KLDr (kldr) - Kullback Distance [**?**, **?**]

$$\begin{aligned} KLD(C, f_i) \;&=\; \sum_{l=1}^{L} \sum_{y \in Y} \left[ P(C = c_l, f_i = f_{iy}) - P(C = c_l)P(f_i = f_{iy}) \right] \\ &\times\; \log_2 \frac{P(C = c_l, f_i = f_{iy})}{P(C = c_l)P(f_i = f_{iy})} \end{aligned} \tag{3.17}$$

4. BDr (bdr) - Bhattacharatyya Distance [**?**, **?**]

$$\begin{aligned} \Psi(C, f_i) \;&=\; \\ &-\; \log_2 \sum_{l=1}^{L} \sum_{y \in Y} \left[ P(C = c_l, f_i = f_{iy}) P(C = c_l) P(f_i = f_{iy}) \right]^{\frac{1}{2}} \end{aligned} \tag{3.18}$$

5. SDr (sdr) - Samon Distance [**?**, **?**]

$$SD(C, f_i) = 1 - \sum_{l=1}^{L} \sum_{y \in Y} \left\{ \min \left[ P(C = c_l, f_i = f_{iy}), P(C = c_l) P(f_i = f_{iy}) \right] \right\} \quad (3.19)$$

Only in a few examples can we find some analytical relation between the measures mentioned above (and only for a two-dimensional problem); for that reason, the usability of these measures in feature selection should be evaluated empirically. For example, in the paper by Sacco [**?**], we can find a relation between the Kollmogorov measure and the Sammon measure expressed by the following equation:

$$KD(C, f_i) = 2\left(1 - SD(C, f_i)\right) \quad (3.20)$$

## 3.5 Redundancy-shifting ranking

The main disadvantage of ranking filters is their inability to discover important interactions between features, as well as their inability to reject redundant features. The first attempt to employ this group of algorithms (called redundancy-shifting algorithms) in solving this problem uses a heuristic based on the minimum-redundancy-maximum-relevancy approach, and was described by Battiti [**?**]. The theoretical analysis of this heuristic was proposed by Peng [**?**].

| Type | Acronym | Full Name | Formula |
|---|---|---|---|
| Discrete | MIFS | Mut. inf. feat. sel. [**?**] | $MI(f_i, C) - \beta \sum_{j \in S} MI(f_i, f_j)$ |
| | MIFS-U | Mut. inf. feat. sel. (uni.) [**?**] | $MI(f_i, C) - \beta \sum_{j \in S} \frac{MI(f_s, C)}{H(f_s)} MI(f_i, f_j)$ |
| | AMIFS | Adap. Mut. inf. feat. sel. [**?**] | $MI(f_i, C) - \frac{1}{\|S\|} \sum_{j \in S} \frac{MI(f_i, f_j)}{H(f_j)}$ |
| | MID | Mut. inf. difference [**?**, **?**] | $MI(f_i, C) - \frac{1}{\|S\|} \sum_{j \in S} MI(f_i, f_j)$ |
| | MIQ | Mut. inf. quotient [**?**, **?**] | $MI(f_i, C) / \frac{1}{\|S\|} \sum_{j \in S} MI(f_i, f_j)$ |
| Continuous | FCD | F-test corr. difference [**?**, **?**] | $F(f_i, C) - \frac{1}{\|S\|} \sum_{j \in S} \|c(f_i, f_j)\|$ |
| | FCQ | F-test corr. quotient [**?**, **?**] | $F(f_i, C) / \frac{1}{\|S\|} \sum_{j \in S} \|c(f_i, f_j)\|$ |

Table 3.1: Various schemes to search for the next feature in a Minimum-Redundancy-Maximum-Relevancy (mRMR) optimization condition.

Following feature selection processing, the number of features does not change in such algorithms; and redundant features should be *shifted* to the end of the set of the ordered features. Proposed heuristics and their improvements are presented in the table below (Tab. 3.1).

A typical selection algorithm can be represented in 5 steps and can be realized as follows:

**Feature selection algorithm:**
1. **Initialization:** set $\mathcal{F} \longleftarrow$ initial set of n features, $\mathcal{S} \longleftarrow$ empty set
2. Calculate $MI(f_i, C)$ relevance indices and create an ordered list $\mathcal{F}$ of features according to the decreasing value of their relevance.
3. Take as $f_i$ the first feature from the $\mathcal{F}$ list, $\mathcal{F} \longleftarrow \mathcal{F}/\{f_i\}$, $\mathcal{S} \longleftarrow \{f_i\}$
4. **Selection:** repeat until desired number of features is selected.
a) Compute all entropies and mutual information between features $f_i$, $f_s$ and heuristic $H_r$ based on mRMR condition.
b) **Selection of the next feature:** choose feature $f_i \in F$ that maximizes heuristic $H_r$, $\mathcal{F} \longleftarrow \mathcal{F}/\{f_i\}$, $\mathcal{S} \longleftarrow f_i$
5. Output set $\mathcal{S}$ containing the selected features.

Figure 3.2: Feature selection – redundancy *shifting* filter algorithm.

The first improvement of Battiti's algorithms was proposed by Kwak *et al.* [**?**]. This approach assumes that information is distributed uniformly to fulfill the following condition (Eq. 3.21):

$$\frac{H(f_s|C)}{H(f_s)} = \frac{I(f_s; f_i|C)}{I(f_s; f_i)} => I(f_s; f_i|C) = I(f_s; f_i) \frac{H(f_s|C)}{H(f_s)} \tag{3.21}$$

This feature selection algorithm tries to maximize mutual information for a pair of features $I(C; f_i, f_s)$, (area 2, 3, and 4 in Fig. 3.3). In this case, mutual information can be rewritten using the formula for multivariate mutual information as follows:

$$I(C; f_i, f_s) = I(C; f_s) + I(C; f_i \| f_s) \tag{3.22}$$

Mutual information between output C and feature $f_i$ for given $f_s$ is represented here by $I(C; f_i \| f_s)$ (area 3 in Fig. 3.3). The sum of areas 2 and 4 represents $I(C; f_s)$. To maximize Eq. 3.21 we need to find the maximal value of the second part $[I(C; f_i \| f_s)]$ because calculating this part requires as much work as the calculation of $I(C; f_i, f_s)$. We will try to approximate this value by $I(f_i; f_s)$ and $I(C; f_i)$, which requires two dimensional probability distribution. The conditional mutual information $I(C; f_i \| f_s)$ can be represented as:

$$I(C; f_i \| f_s) = I(C; f_i) - \{I(f_s; f_i) - I(f_s; f_i \| C)\} \tag{3.23}$$

Mutual information between feature $f_i$ and $f_s$ $[I(f_i, f_s)]$ corresponds to area 1 and 4, and $I(f_s; f_i \| C)$ matches area 1. The difference between $I(f_s; f_i) - I(f_s; f_i \| C)$ corresponds to area 4 in Fig. 3.3.

Using equation Eq. 3.21, we can represent the second term in Eq. 3.22 as follows:

$$\begin{aligned} I(C; f_i \| f_s) &= I(C; f_i) - \left(1 - \frac{H(f_c \| C)}{H(f_s)}\right) I(f_s; f_i) \\ &= I(C; f_i) - \frac{I(C \| f_s)}{H(f_s)} I(f_s; f_i) \end{aligned} \tag{3.24}$$

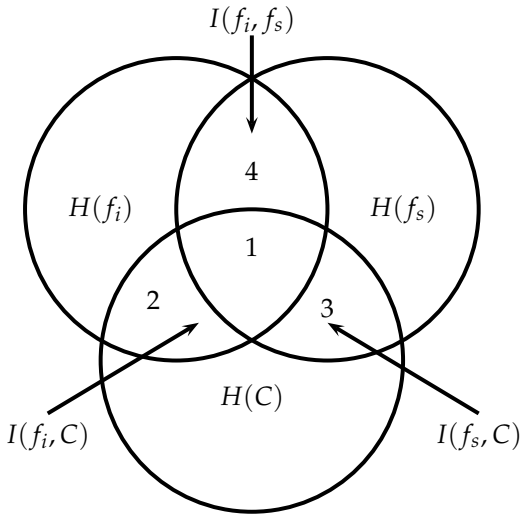


Figure 3.3: The relation between entropies and mutual information for features and output classes.

With this formula we revise heuristics in Step 4 in the selection algorithm as follows:

4. **Selection:** repeat until desired number of features is selected.
a) Compute all entropies and mutual information between features $f_i$, $f_s$ and heuristic $H_r$ as follows:

$$I(C; f_i) - \beta \sum_{s \in \mathcal{S}} \frac{I(C \| f_s)}{H(f_s)} I(f_s; f_i) \tag{3.25}$$

b) **Selection of the next feature:** choose feature $f_i \in F$ that maximizes heuristic $H_r$, $\mathcal{F} \longleftarrow \mathcal{F}/\{f_i\}$, $\mathcal{S} \longleftarrow f_i$

The second attempt to improve a Minimum-Redundancy-Maximum-Relevancy condition is proposed by Tesmer **[?]** (called AMIFS - Adaptive Mutual Information Feature Selection), where the authors analyze the relation between redundancy and mutual information for two features. Cover *et al.* **[?]** observe the following rule for two features:

$$0 < I(f_s; f_i) < Min\{H(f_i), H(f_s)\} \tag{3.26}$$

where $H(f_i)$ corresponds to the entropy of feature $f_i$. This expression is illustrated in Fig. 3.4; in particular, if one feature is completely redundant ($f_i \subset f_s$), then the mutual information is maximal [$I(f_i, f_s) = H(f_i)$].

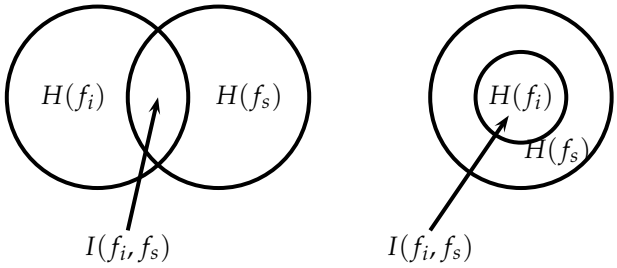


Figure 3.4: The relation between entropies and mutual information for two features.

In this case, the AMIFS algorithm modifies step 4 as follows:

---

4. **Selection:** repeat until desired number of features is selected.
a) Compute all entropies and mutual information between features $f_i$, $f_s$ and heuristic $H_r$ as follows:

$$I(C; f_i) - \sum_{s \in \mathcal{S}} \frac{I(f_i \| f_s)}{N_s \tilde{H}(f)} \tag{3.27}$$

b) **Selection of the next feature:** choose feature $f_i \in F$ that maximizes heuristic $H_r$, $\mathcal{F} \longleftarrow \mathcal{F}/\{f_i\}$, $\mathcal{S} \longleftarrow f_i$

---

where

$$H(f) \;=\; Min\{H(f_i), H(f_s)\} \tag{3.28}$$

and $N_s$ is the number of already selected features.

The right-hand term in 3.27 is an adaptive redundancy penalization term which corresponds to the average of the mutual information between each pair of selected features, divided by the minimum entropy between them. Comparing 3.27 with 3.25, we can see that the parameter $\beta$ has been replaced by an adaptive term represented by the denominator of the summation argument in Eq. 3.27. In *MIFSU* (see 3.25), the redundancy correction term always divides the $I(f_i; f_s)$ by the entropy of the selected features $H(f_s)$. If $H(f_s)$ is greater than the entropy of the candidate feature $H(f_i)$, the *MIFSU* algorithm would give more importance to the relevance, instead of penalizing the redundancy more strongly.

The last most important attempt to establish a relation between the conditions of Max-Dependency, Max-Relevancy and Min-Redundancy was made by Peng *et al.* [**?**, **?**]. In the second paper, the authors prove that combining the Max-Relevancy and the Min-Redundancy criteria , *i.e.*, the **mRMR** criterion, is equivalent to the Max-Dependency criterion if one feature is selected (added) one at a time.

To formalize this approach, we need to define these four criteria. The first criterion (Max-Relevancy) helps to find the most informative subset of features and can be written as follows:

$$\max D(S,C), \quad D = MI(\{f_i, i = 1,...,m\}; C) \tag{3.29}$$

where **S** represents the set with m features $\{f_i\}$ which jointly have the largest dependency on the target class C. For continuous attributes, this can be defined as:

$$\begin{aligned} MI(S_m, C) &= p(S_m, C) \log \frac{p(S_m, C)}{p(S_m)\, p(C)} dS_m dC \\ &= p(S_{m-1} f_m, C) \log \frac{p(S_{m-1}, f_m, C)}{p(S_{m-1} f_m)\, p(C)} dS_{m-1} df_m dC \\ &= p(f_1, .., f_m, C) \log \frac{p(f_1, ..., f_m, C)}{p(f_1, .., f_m)\, p(C)} df_1 ... df_m dC \end{aligned} \tag{3.30}$$

The Max-Dependency criterion uses multi-dimensional probabilities, and it is hard to implement. The alternative approach uses a *maximum relevancy* (Max-Relevancy) criterion to select a suboptimal (almost "optimal") subset of features. This criterion can be formulated as the average value of the mutual information between features ($f_i$) and class ($C$):

$$\max Re(S,C), \quad Re = \frac{1}{\|S\|} \sum_{f_i \in S} MI(f_i; C) \tag{3.31}$$

It is probable that the features selected according to Max-Relevancy could have rich redundancy. Therefore, to improve the utility of the subset of selected features, the following *minimum redundancy* (Min-Redundancy) condition can be added in order to select mutually exclusive features:

$$\min Rd(S,C), \quad Rd = \frac{1}{\|S\|^2} \sum_{f_i \in S} MI(f_i; f_s) \tag{3.32}$$

The criterion which combines these two constraints is called "minimum-redundancy-maximum-relevance" (mRMR). For this purpose, we can define function $\Phi = Re - Rd$, which will be used as a new criterion:

$$\max \Phi(Re, Rd) \quad \Phi = Re - Rd; \tag{3.33}$$

In this case, the **mRMR** condition modifies step 4 in the feature selection algorithm as follows:

Comparing the condition proposed by Battiti [**?**] with Eq. 3.34, it can be seen that the parameter $\beta$ has been replaced by $\frac{1}{s-1}$. Peng *et al.* [**?**] in their paper proposed several variations of the **mRMR** condition called Mutual Information Quotient (MIQ), Mutual Information Difference (MID), F-test correlation difference (FID), and F-test correlation quotient (FIQ). Respective equations are presented in table 3.5.

4. **Selection:** repeat until desired number of features is selected.
a) Compute all entropies and mutual information between features $f_i$, $f_s$ and heuristic $H_r$ as follows:

$$I(C; f_i) - \frac{1}{s-1} \sum_{s \in \mathcal{S}} I(f_i, f_s) \tag{3.34}$$

b) **Selection of the next feature:** choose feature $f_i \in F$ that maximizes heuristic $H_r$, $\mathcal{F} \longleftarrow \mathcal{F}/\{f_i\}$,$\mathcal{S} \longleftarrow f_i$

## 3.6 Redundancy-removing ranking

The next group develops algorithms which attempt to solve a redundancy problem by exploiting a concept of predominant features (relevant and non-redundant features). The feature selection algorithm involves two steps: (1) selecting a subset of relevant features and (2) removing non-predominant features. This simple strategy is presented in figure 3.5 , where we can see two separate steps. The first step is a relevancy analysis where a cut-off value of an evaluation function (mutual information, symmetrical uncertainty coefficient, *etc.*) can be used for the selection of a subset of relevant features or a number of expected features.

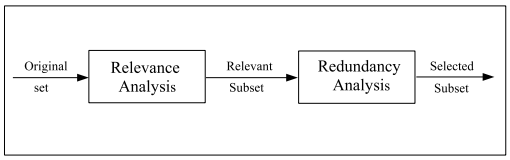


Figure 3.5: Two stage framework of feature selection.

The second step involves a redundancy analysis. The general idea of this step is presented in Figure 3.6. After ordering features (relevancy analysis), we initialize the next step by taking the feature most strongly correlated (C-correlation) with class. If we can find features redundant with this feature, we remove these features from the subset. This procedure is repeated until we have only predominant features. In Figure 3.6, six features are selected as important to the target and ordered according to their relevancy (C-correlation). $F_1$ is most relevant and $F_6$ is least relevant. In the first round, $F_1$ is selected as a predominant feature, and the next two features ($F_2$, $F_3$) are removed as features which carry the same information. In the next step we pick up feature $F_4$, and features $F_5$ and $F_6$ are removed based on $F_4$. The final predominant subset contains two features $F_1$ and $F_2$.

### 3.6.1 FCBF - Fast Correlation-Based Filter

The first algorithm which exploited this idea was proposed and named FCBF (Fast Correlation-Based Filter) by Lei Yu [**?**]. The crucial point in considering all this is a definition of redundancy and how the redundancy is evaluated (*i.e.*, approximated by using heuristics in determining an approximate Markov blanket). In the FCBF algorithm, feature $F_j$ is redundant to feature $F_i$ if the F-correlation with feature $F_i$ is stronger then the C-correlation with class $\mathcal{C}$. In their paper [**?**],

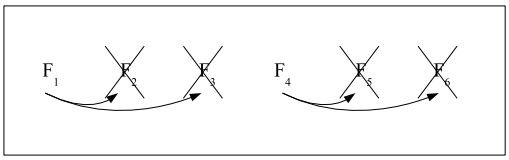


Figure 3.6: Selection of predominant features – removing redundant features.

the authors used a Pearson correlation coefficient and a Symmetrical Uncertainty Coefficient [**?**, **?**] to express F-(C-)correlation. The complexity of the algorithm can by considered in two terms: calculation of the relevancy function and determination of predominant features. The calculation of the SUC or of the Pearson correlation coefficient has linear complexity $O(n)$ in terms of the number of instances. In the case of predominant feature determination, we have one part related to a relevancy (sorting) which has complexity $O(NlogN)$. The second part is related to a redundancy analysis which has a best-case complexity $O(N)$ when only one feature is selected and all the rest of the features are removed, and a worst-case complexity $O(N^2)$ when all features are selected. This makes the FCBF method substantially faster than algorithms of subset evaluation based on a greedy sequential search.

As shown in Figure 3.7, for a data set S with N features and class C, the algorithm finds a set of predominant features $S_{best}$ . In the first level (lines 1-7), it calculates the $J(F_i, C)$ (C-correlation – originally SUC ) value for each feature, selects relevant features into a $S'_{list}$ list based on a predefined threshold $\delta$ value of relevance indices or of number of features, and orders them in descending order according to their $J(F_i, C)$ values. In the second step (lines 8-22), it further processes the ordered $S'_{list}$ list to select predominant features. A feature $F_p$ that has already been determined to be a predominant feature can always be used to filter out other features for which $F_p$ forms an approximate Markov blanket. Since the feature with the highest C-correlation does not have any approximate Markov blanket, it must be one of the predominant features. So the iteration starts from the first element in the $S'_{list}$ list (line 8) and continues to the end of the list. For all the remaining features (from the one right next to $F_p$ to the last one in $S'_{list}$), if $F_p$ happens to form an approximate Markov blanket for $F_q$ (line 13), $F_q$ will be removed from $S'_{list}$ list . After one round of filtering features based on $F_p$, the algorithm will take the remaining feature right next to $F_p$ as the new reference (line 13) to repeat the filtering process. The algorithm stops until no more predominant features can be selected.

### 3.6.2 Kolmogorov-Smirnov Correlation-Based Filter Approach.

The next two redundancy removal algorithms also employ a two-stage strategy (relevancy and redundancy analysis stages) for selecting predominant features. These two algorithms determine another way to approximate the Markov Blanket. This very simple heuristic assumes that two features are redundant if their probability distribution is equal. This property is symmetrical. Equality of probability distributions can be determined by several statistical tests such as the Kolmogorov-Smirnov and Pearson goodness-of-fit tests. Limitation of these test is related to type of data, number of examples (instances), *etc.*; and algorithms proposed by Biesiada *et al.* [**?**, **?**, **?**, **?**] should be used carefully. The level of approximation of the Markov Blanket is expressed by the significance level used in statistical tests. The complexity of the algorithm is the same as in the FCBF algorithm.

**input:** $S(F_1, F_2, \ldots, F_N, C)$ // training data
**threshold:** $\delta$ // a predefined threshold
**output:** $S_{best}$ // an optimal subset
**Level 1**
1. **begin**
2. for i=1 to N do begin
3. calculate $J(F_i, C)$;
4. if ( $J(F_i, C) \geq \delta$ )
5. append $F_i$ to $S'_{list}$;
6. end;
7. order $S'_{list}$ in descending $J(F_i, C)$ value;

**Level 2**
8. $F_p$=*getFirstElement*($S'_{list}$)
9. do begin
10. $F_q$=*getNextElement*($S'_{list}$, $F_p$)
11. if( $F_q$ <>NULL )
12. do begin
13. $F'_q = F_q$
14. if ( J($F_p$,$F_q$) $\geq$ J($F_q$,C) )
15. remove $F_q$ from $S'_{list}$;
16. $F_q$ = *getNextElement*($S'_{list}$,$F'_q$);
17. else $F_q$ = *getNextElement*($S'_{list}$,$F_q$);
18. end until ( $F_q$ == NULL );
19. $F_p$ = *getNextElement*($S'_{list}$,$F_p$);
20. end until ( $F_p$ == NULL );
21. $S_{best}$=$S'_{list}$;
22. **end**;

Figure 3.7: Fast Correlation-Based Filter algorithm.

**Kolmogorov-Smirnov test for two distributions.**
The Kolmogorov-Smirnov (K-S) test [**?**] is used for evaluating two distributions if they are equal, and it may also be used as a test for feature redundancy. The K-S test consists of the following steps:

- The discretization process creates $k$ clusters (vectors from roughly the same class), each typically covering a similar range of values. measuring frequencies of different classes.
- The construction of the empirical cumulative distribution function $F1_i$, and $F2_i$ for two sample populations is based on the frequency table.
- $\lambda$(K-S statistics) is proportional to the largest absolute difference of $|F1_i - F2_i|$:

$$\lambda = \sqrt{\frac{n_1 n_2}{n_1 + n_2}} \sup |F1_i - F2_i| \quad for \;\; i = 1, 2, ..., k. \tag{3.35}$$

When $\lambda < \lambda_\alpha$ then the two distributions are equal, where $\alpha$ is the significance level and $\lambda_\alpha$ is the K-S statistics for $\alpha$ [**?**]. One of the features with approximately equal distribution is redundant. In the experiments described, equal numbers of examples in the training set $n_1 = n_2 = n$ were used, and the number of examples should be at least 100.

The Kolmogorov-Smirnov test is a good basis for the Correlation-Based Selection algorithm (K-S CBS) for feature selection. This algorithm is presented in Fig. 3.8. The first feature ranking performed requires selection of the ranking index; below, the F-score index Eq. 3.4 is used, but, in general, we can also use any other indices. The threshold for the number of features left for further analysis may be determined by using the frapper approach, *i.e.*, evaluating the quality of results as a function of the number of features. In the second step, redundant features are removed using the K-S test. The optimal $\alpha$ significance level for feature removal may also be determined by cross-validation.

**Algorithm K-S CBS:**
**Relevance analysis**
1. Create an $\mathcal{S}$ list by ordering features according to the decreasing value of their relevance indices.
**Redundancy analysis**
2. Initialize $F_i$ the first feature in the $\mathcal{S}$ list.
3. Use K-S test to find and remove from $\mathcal{S}$ all features for which $F_i$ forms an approximate redundant cover $\mathcal{C}(F_i)$.
4. Move $F_i$ to the set of selected features; take as $F_i$ the next remaining feature in the list.
5. Repeat step 3 and 4 until the end of the $\mathcal{S}$ list.

Figure 3.8: A two-step Kolmogorov-Smirnov Correlation Based Selection (K-S CBS) algorithm

This is, of course, a rather generic algorithm, and other ranking indices and tests for equality of distributions may be used instead. Two parameters – the threshold for relevancy and the threshold for redundancy – are successively determined using cross-validation, but in some cases there may be a clear change in the value of these parameters, which change helps find the optimal values of the thresholds.

### 3.6.3 Pearson's Redundancy-Based Filters.

The Pearson $\chi^2$ test measures the difference between the probability distribution of two binned random variables. If a feature is redundant, then the hypothesis that its distribution is equal to an already selected feature should have high probability. $n$ independent observations of two random variables $X, X'$ are given in the training data, where for the Pearson $\chi^2$ test to be valid $n$ should be more than 100. The test for $X, X'$ feature redundancy proceeds as follows:

- Frequencies $f_i, f'_i$ of occurrences of feature values in each bin are recorded (counting unique feature values).

- Frequency counts are the basis on which the empirical probability distributions $F_i$ and $F_i'$ are constructed, and the $\chi^2(X, X')$ matrix is constructed as follows:

$$\chi^2(X, X') = \sum_{i=1}^{k} \frac{\left(F_i - F_i'\right)^2}{F_i'} \tag{3.36}$$

A large value for $\chi^2$ or a different number of unique feature values indicate that features are not redundant. When $p(\chi^2) > \alpha$, then the two distributions are equivalent to $\alpha$ significance level, and thus one of the features is redundant.

Pearson's Redundancy-Based Filter (PRBF) algorithm is presented in Fig. 3.9. First, the relevance is determined using the symmetrical uncertainty. Other relevance criteria may also be used. After that the $\chi^2$ test is applied to remove redundancy.

---

**Algorithm PRBF:**
Relevance analysis
1. Calculate $SU(X, C)$ relevance indices and create an ordered list $\mathcal{S}$ of features according to the decreasing value of their relevance.
**Redundancy analysis**
2. Take as $X$ the first feature from the $\mathcal{S}$ list
3. Find and remove all features for which $X$ is approximately equivalent according to the Pearson $\chi^2$ test
4. Set the next remaining feature in the list as $X$ and repeat step 3 for all remaining features in the $\mathcal{S}$ list.

---

Figure 3.9: A two-step Pearson's Redundancy-Based Filter (PRBF) algorithm.

## 3.7 Other implemented algorithms.

The *Markov Blanket filter* is a more computationally intensive procedure which supports redundancy reduction using the concept of the Markov Blanket and should be able to find the optimal subset of pedominant features. This algorithm was proposed by Koller *et al.* [**?**] and also was used for a biomedical problem (micro-array analysis) by Xing *et al.* [**?**]. The approximate Markov Blanket used in this filter can be expressed as follows:

$$\begin{aligned} \Delta\left(F_i|\mathbf{M}\right) \quad = \quad & \sum_{f_M, F_i = f_i} P\left(\mathbf{M} = f_M, F_i = f_i\right) \\ & D\left(P\left(C|\mathbf{M} = f_M, F_i = f_i\right) \| P\left(C|\mathbf{M} = f_M\right)\right) \end{aligned} \tag{3.37}$$

where $D\left(P\|Q\right) = \sum_x P\left(x\right) \log\left(P\left(x\right) \| Q\left(x\right)\right)$ is a Kullback-Leibler divergence. If $\Delta\left(F_i|\mathbf{M}\right) = 0$, then $\mathbf{M}$ is a Markov Blanket for $F_i$, which is the equivalent to the equality of the conditional probabilities $[P\left(C|\mathbf{M} = f_M, F_i = f_i\right) = P\left(C|\mathbf{M} = f_M\right)]$.

Using equation number 3.37 after Koller *et al.* [**?**] and Xing *et al.* [**?**] we can formulate the algorithm with a fixed number of features in the Markov Blanket as parameter $(k)$ as follows:

**Algorithm** *Markov Blanket Filter* **:**
**Initialization**
1. Set F $\leftarrow$ initial set of n features, S $\leftarrow$ empty set
**Iterate - until F is empty**
2. For each feature $f_i \in \mathbf{F}$, let $\mathbf{M}_i$ be the set k features
(parameter in algorithm) $f_j \in \mathbf{F} - \{f_i\}$
3. Compute $\Delta\left(f_i|\mathbf{M_i}\right)$ for each i.
4. Choose the i that minimizes $\Delta\left(f_i|\mathbf{M_i}\right)$.
5. $\mathbf{F} = \mathbf{F} - \{f_i\}$, $\{f_i\} \rightarrow S$.

Figure 3.10: Markov Blanket Based Filter (mbr – Markov Blanket ranking) algorithm.

The last implemented algorithm in the standard part of the Infosel++ library is called GD-distance ranking [**?**]. The main disadvantage of this algorithm is its inability to perform selection when feature sets include redundant (strongly correlated) features. This main feature comes from the structure of the Mahalanobis distance [**?**], where the determinant will be equal to zero when the subset includes redundant features. The GD-distance is defined using the Mahalanobis (Eq. 3.38) distance as follows:

$$D_{GD}(f,C) \;=\; D(f,C)^T\; T^{-1}\; D(f,C) \tag{3.38}$$

where $T$ is the information matrix, and where we have pairwise mutual information (between all pairs of features).

$$T(f_i,f_j) \;=\; \begin{pmatrix} MI(f_1,f_1) & MI(f_1,f_2) & \ldots & MI(f_1,f_n) \\ MI(f_2,f_1) & MI(f_2,f_2) & \ldots & MI(f_2,f_n) \\ \ldots & \ldots & \ldots & \ldots \\ MI(f_k,f_1) & MI(f_f,f_2) & \ldots & MI(f_k,f_n) \end{pmatrix} \tag{3.39}$$

and $D(f,C)$ is defined by the Mantaras distance $d_{LM}(C,f_i) \;=\; H(C|f_i) + H(f_i|C)$ [**?**] as follows:

$$D(f,C) \;=\; \begin{pmatrix} d_{LM}(f_1,C) \\ d_{LM}(f_2,C) \\ d_{LM}(f_3,C) \\ \ldots \\ d_{LM}(f_k,C) \end{pmatrix} \tag{3.40}$$

The Mantaras distance [**?**] fulfills all distance axioms (positive definiteness, symmetry and triangle inequality):

1. $d_{LM}\left(P_A,P_B\right) \geq 0$ and equal iff $P_A = P_B$.
2. $d_{LM}\left(P_A,P_B\right) = d_{LM}\left(P_B,P_A\right)$

3. $d_{LM}(P_A, P_B) + d_{LM}(P_B, P_C) \geq d_{LM}(P_A, P_C)$

The redundant features can be removed from the initial subset using the simple property of the information matrix ($T$), which is as follows: two features are redundant if $t_{ij} = t_{ii}$, where $t_{ij}, t_{ii} \in T$. After removing all redundant features, we can perform a sequential forward search. In each step of the algorithm, a new feature is added that gives the lesser increase of the GD-measure value. This can be represented by the following scheme:

---

**Algorithm** *GD-distance Filter* **:**
**Initialization**
1. Set F $\leftarrow$ initial set of n features, S $\leftarrow$ empty set
**Iterate - until F is empty**
2. For each feature $f_i \in \mathbf{F}$ compute $D_{GD}(S, f_i, C)$.
3. Choose the $f_i$ that minimizes $D_{GD}(S, f_i, C)$.
4. $\mathbf{F} = \mathbf{F} - \{f_i\}$, $\{f_i\} \rightarrow S$.

---

Figure 3.11: GD-Distance Based Filter (gdr – GD-distance ranking) algorithm.

# Chapter 4

# Infosel++ Library for Algorithm Developer

Infosel++ is a class library for development of feature selection algorithms based on probability distributions. Infosel++ is a cross-platform library. Version 1.00 has been released for Windows (Microsoft Visual C++) and Linux (gcc 4.2) systems.

Infosel++ includes a collection of general purpose classes, independent of machine learning, declared in the namespace InfoSelCore::Common. The UML graph of InfoSelCore::Common namespace is presented in figure 4.2. InfoSelCore::Common classes include arrays, hash tables, math-elements, and output/input streams, as well as some built-in mechanisms for dealing with C++ initialization order, options, temporary generation, and cleanup, as well as interrupt handling.

The important concepts for feature selection provided by the library have been defined in the namespace InfoSelCore::Data and include point (element of column: *datum*), column with data (feature: *datcolumn*), vector with data (*datvector*), dataset (*datset*), subset of features (*subdatset*), data file (*datfile*), *etc.*The Infosel++ class names are in parentheses and italicized. The namespace InfoSelAPI includes several classes and methods important for the implementation of algorithms, such as table (sparse table based on hash table technique), tensor0D, tensor1D, and tensor2D.

In this namespace we also have a temporary structure for storing a calculated value during feature selection processing, as well as a method related to the calculation of probability distribution (one- or multidimensional) called *prob* (*distrib*).

All implemented algorithms are stored and ordained automatically in special separate modules called algorithm packages, which allow the creation of hybrid feature selection algorithms automatically.

The architecture of the InfoSel++ library is layered. Individual layers are grouped into modules responsible for the sequential steps of the processing starting with the input, which is a collection of data, ending with the targeted results.

In the source code, the library modules correspond to the namespaces of the C++ language; in the UML diagrams they are represented by a symbol of the package. It should be noted that at the present stage of the development of the library, the layers are not specifically indicated in the source code. Hence, in UML diagrams they are represented only by the "straight edges of the system." A diagram of the generalized static structure of the InfoSel++ library is illustrated in Figure 4.1. Description of all library modules is presented below.

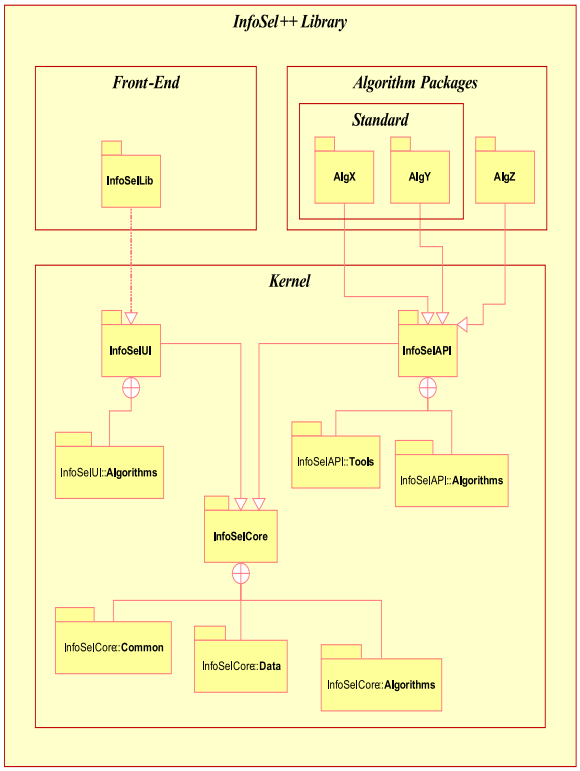


Figure 4.1: Main architecture of library.

- InfoSelLib – a module of front, high-level user interface libraries, the purpose of which is encapsulation of the internal structure of the library, with simultaneous access to most of the services offered by the low-level interface;
- InfoSelUI – a low-level user interface library module enabling direct access to the kernel library and execution of a feature selection process for selected algorithms;
- InfoSelUI::Algorithms – a submodule enabling the use of all feature selection algorithms (the ALGs collection), the classes and modules of which have been included and registered in the library;
- InfoSelCore – a central module of the kernel library responsible for the execution of major operations related to feature selection processes, such as reading data and input parameters, conversion of the input data to the appropriate structure in accordance with the adopted model, carrying out pre-processing of the input data, and execution of selected algorithms;
- InfoSelCore::Common – a submodule of useful elements for general purposes;
- InfoSelCore::Data – a submodule of classes representing main concepts associated with the input data structure, presented in the UML diagram in figure 4.3;

- InfoSelCore::Algorithms – a submodule of basic elements used in both the implementation and the utilization of the algorithms;
- InfoSelAPI – an API module enabling implementation of feature selection algorithms and their inclusion in the kernel library;
- InfoSelAPI::Tools – a submodule of tools which simplify writing the definition of an algorithm, presented in the UML diagrams in figure 4.4 and figure 4.5;
- InfoSelAPI::Algorithms – a submodule of elements enabling programmatic addition (registration) of classes and modules to the library;
- AlgX, Algy, AlgZ – modules X, Y, Z with feature selection algorithms added to the library.

The main library layer contains the kernel specific modules. The main role of the kernel is to read the data file and input parameters, to perform all processing on the initial data set, and to execute the requested feature selection algorithms. The most important data structures and other elements of the program are stored in a central module of the library, which is called InfoSelCore. This module consists of 3 submodules: InfoSelCore::Common, InfoSelCore::Data, and InfoSelCore::Algorithms. Another layer of the library is a collection of modules with selection algorithms defined (Algorithm Packages). Thanks to the open architecture of the library, each algorithm can be integrated as an independent, separately compiled module. Elective expansion of the library requires neither interference in the already existing code, nor its recompilation. This guarantees free and easy extensibility. For convenience, similar algorithms can be grouped into a single module. The name of the module may be the same as the name of a single algorithm or of the algorithm family. Module creation and implementation of its internal algorithms in the library environment require full access to the public elements of the library kernel. This is achieved using the InfoSelAPI module (API - Algorithm Programming Interface). Moreover, this module offers a number of additional elements grouped in submodules InfoSelAPI::Algorithms and InfoSelAPI::Tools, and intended primarily for registering algorithms and writing their definitions. The API interface also introduces specific requirements for the implementation of algorithms in the source code of the module. For instance, each feature selection algorithm should be represented in the form of a separate C++ class derived from the base class InfoSelCore::Algorithms::Algorithm. The content of the definition of the algorithm is stored in the form of virtual methods, also derived from the base class. Adoption of the object oriented model makes the overall library code clearer and more concise, comparable to mathematical notation. In particular, the extended object model has been adopted in the structure of the input data, where classes representing the main concepts of feature selection, such as input file, features and feature sets, space of subsets of features, and probability distribution, have been specified.

The execution of a complete feature selection process for the given input data and for the selected algorithms is done by selecting the appropriate user interface library. The low-level interface, intended exclusively for use in the C++ code, offers a module InfoSelUI (UI - User Interface). This module allows any given code direct access to the core elements of public libraries and collections registered in the algorithms library. This collection is located in the submodule InfoSelUI::Algorithms. The use of the library at a higher level, *i.e.*, without direct reference to the kernel, is provided by the front layer of the library. This layer includes a series of high-level user interfaces, which make the majority of the low-level interface services available, while obscuring the InfoSelUI code library. The InfoSelLib module of the front layer contains the basic interfaces in the form of two classes: Script and Entry, both intended for use in the C++ code. The former class is a convenient-to-use interface library that provides easy connection between

library and other system components, such as a command-line program with an interactive interface in the text mode, or a GUI type application. In addition, the Entry class allows the library to be used with other data processing software. The class Script is a scripting interface, which allows the library to be used in batch mode.

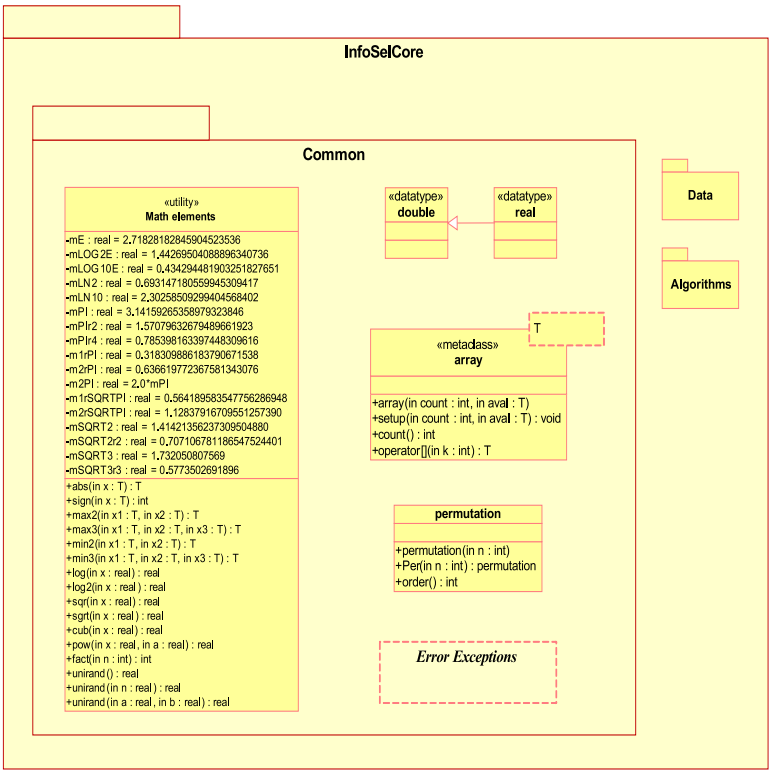


Figure 4.2: General elements of library.

The general elements of the library listed below are a part of InfoSelCore namespace:

- Math elements – a collection of mathematical constants and functions which are commonly used in the library code;
- array – a class template representing any type of data arrays;
- permutation – a simple class representing permutation of integers of the n-th order;
- double – an equivalent of the C++ double type;
- real – an alias name for the double type;
- xError – a base exception covering all types of errors in the library;
- xUnknownExn – an exception representing unknown errors;
- xToolError – an exception for all errors related to the tools for algorithm implementation;

- xInOutError – an exception for all errors associated with the input/output operations;

Another part of the library consists of elements of the InfoSelCore::Data, which describes the algebraic elements of dataset. All these elements and their definitions are as follows:

- datum – a class representing a single (atomic) value of the input data;
- datcolumn – a class representing a column of data in the input file. Each column is a total aggregate of objects of class datum.
- datvector – a class representing a vector of data in the input file, each vector is a partial aggregate of objects of class datum;
- datset – a class representing a collection of all data read from the input file. The set is organized as a complete aggregate of datcolumn objects and may also be formally treated as an aggregate of objects of class datvector;
- subdatset – a class representing a subset of the data set of partial aggregation;
- datspace – a class representing a space (set) of all data subsets. The space can be formally treated as an aggregate of subdatset objects;
- datfile – a class representing the input data file. The file is a total aggregate of datset objects and is also the root of the object graph modeling the structure of the input data in the library.
- repfile – a class representing the output file containing the report of the feature selection process and the calculation results;

- Algorithm - a base class of all registered classes of feature selection algorithms;
- xAlgsError - exception errors associated with algorithms library;
- xUnknownAlgorithm - an exception for unknown algorithm error;
- Register - a class template used in the registration directive of an algorithm ALG class.

The last group lists the main elements included in InfoSelAPI:Tools namespace, which is responsible for implementing the probability tables, distribution tables, and classes used for algorithm creation (implementation).

- table – a multi-dimensional array of real-type numbers, used for storing the interim and final feature selection calculation results. For each dimension the table has a separate size identifying the size of a subarray;
- tensor – a multidimensional tensor created as an array of class table with fixed size for all dimensions;
- tensor0D – 0-dimensional tensor (single number);
- tensor1D – 1-dimensional tensor (vector of numbers);
- tensor2D – 2-dimensional (square matrix of numbers);

- BigTable – a variant of a class table array, whose size/dimensionality can take large values. To avoid overloading the RAM memory, this structure was implemented using hashing techniques;
- field – a multidimensional continuous field whose values are arrays of class table.
- Probabilities – procedures that return an array of class table. Return arrays contain probability distribution of the input data, differentiated according to the type of data structures, imputed (passed) as arguments of procedure calls;
- Big-Probabilities – same procedures, as described above, except that return arrays are of class BigTable;
- Gaussians – same procedures as desribed above, returning an object of field type with the density probability distribution function of the input data smoothed with a Gaussian function;
- ALGs – a collection of objects of all included and registered classess of feature selection algorithms;
- Entry – a class representing the type of mandatory front interface, allowing easy linking with the other components of the system;

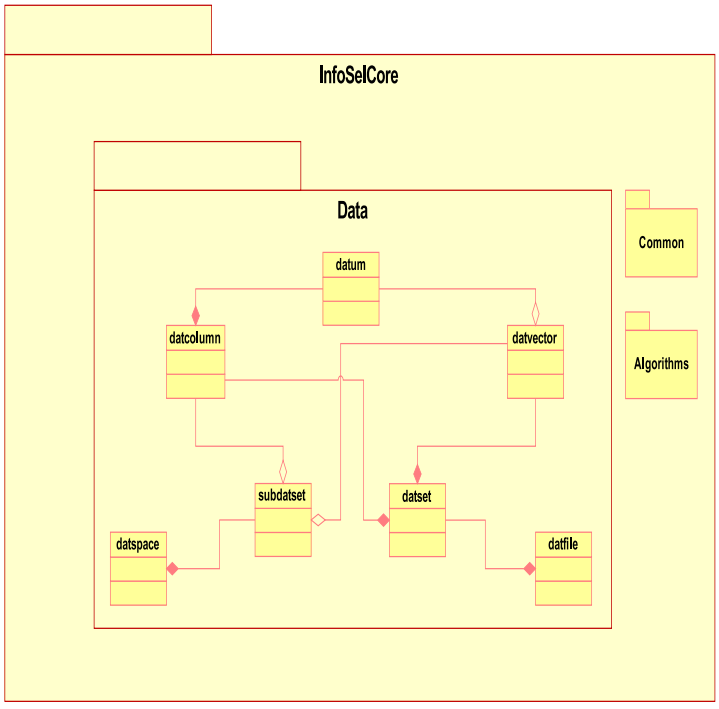


Figure 4.3: Aggregate elements of input data.

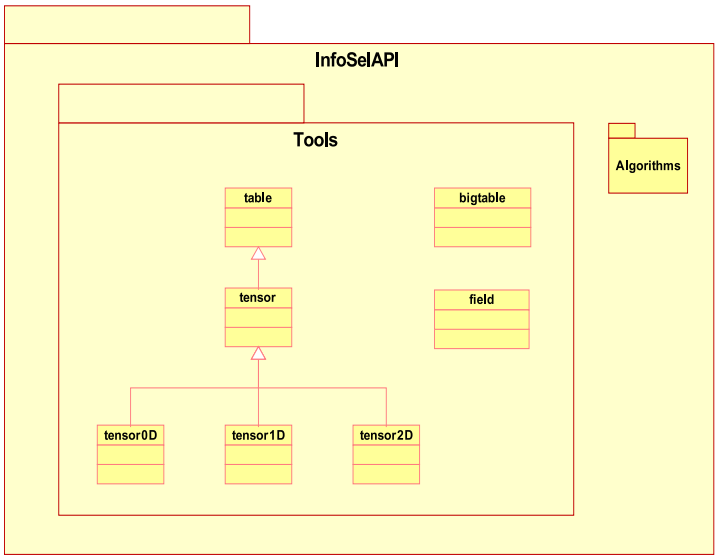


Figure 4.4: Basic classes used for algorithm implementation.

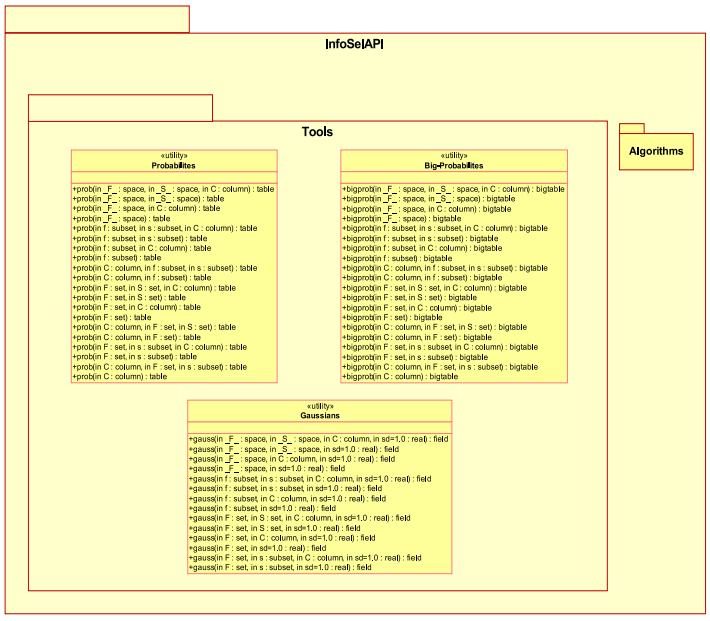


Figure 4.5: Details of basic classes used for algorithm implementation.

- Algorithm – a helper class representing a record with the description of a feature selection algorithm used in the interface Entry;
- Script – class representing the type of scripting front interface which allows the use of the library by a program running in batch mode.

## 4.1 Code Example

A commonly used ranking in feature selection uses Mutual Information as a critical function, defined as a special case of Kullback-Leiber divergence:

$$D_{KL}(p(f,C)\|p(f)p(C)) \;=\; MI(f,C) \;=\; \sum_{i,k\in C} p(f_i,c_k)\log\frac{p(f_i,c_k)}{p(c_k)p(f_i)} \tag{4.1}$$

or uses entropy for the feature and the vector class, and joint entropy between feature and class:

$$\begin{aligned} MI(f,C) &= H(f)+H(C)-H(f,C) \\ &= -\sum_i p(f_i)\log p(f_i) - \sum_{k\in C} p(c_k)\log p(c_k) + \sum_{i,k\in C} p(f_i,c_k)\log p(f_i,c_k) \end{aligned} \tag{4.2}$$

Implementation of ranking based on Mutual Information with implemented algebraic structure in the Infosel++ library:

```
1.  const int vnt=variant();
2.  const real delta=delta_;

3.  datspace _F_(F,1);
4.  datspace _S_(S,1);

5.  table P_FC = prob(_F_,C);
6.  table P_F = prob(_F_);
7.  table P_C = prob(C);

8.  table H_FC = -sum(sum(P_FC*log2(P_FC)));
9.  table H_F = -sum(P_F*log2(P_F));
10. table H_C = -sum(P_C*log2(P_C));

11. vector MI_FC = H_C + H_F - H_FC;

12. rep.log() << "Entropy H(F,C) = " << H_FC << endl;
13. rep.log() << "Entropy H(F) = " << H_F << endl;
14. rep.log() << "Entropy H(C) = " << H_C << endl;
15. rep.log() << "MI-coefficient MI(F,C) = " << sorted << MI_FC << endl;

16. // vector MI_FC = sum(sum( P_FC*log2(P_FC/(P_F^P_C)) ));
17. // rep.log() << "Mutual information MI(F,C) = " << sorted << MI_FC << endl;

18. S.reversion(true);

19. if (vnt==vRank){
20.  subdatset f = argmax(_F_, MI_FC );
21.   while (!f.isnil()) {
22.   F-=f;
23.   S+=f;
24.    if (completed(file)) break;
25.    f = argmax(_F_, MI_FC);
26.    if (MI_FC(f) <= delta ) break;
27.   }
28. }
```

The first two lines define two parameters. The first is "vnt", which represents a search method, in our case, a ranking, the simplest kind of search method; the second parameter,

"delta", is a cut-off value for finishing a ranking. Below this value all features will be rejected (default value 0.05). The next two lines (3. and 4.) represent two subsets, one initial subset with all features and a second one where features are stored after fulfilling the condition of our algorithm. The stored feature is deleted from the initial subset of features. The next three lines (5-7) define tables of joint probability between features and class for all features in subset F as well as probabilities of features in subset F and class C, respectively. Lines 8-10 represent entropies for the subset of features, class, and joint entropy between class and features (defined in eq. 4.2). In line 11 we have defined Mutual Information as a function of entropies (eq. 4.2). This line is equivalent to line 16 where we show the possibility of using Mutual Information defined by equation 4.1. The sole function of line 17 is to report values of Mutual Information.

Lines 12-15 represent reports about calculated entropies and mutual information. Line 18 describes the direction of sorting of features based on the ranking function. The main algorithm starts at line 20, where we initiate an algorithm by selecting the feature with the maximum value of MI. Lines 21-27 represent main while loop, which is finished when subspace F is empty or a maximal value of MI in subset F is below the delta value (line 26).

In case of the Battiti [**?**] algorithm, where the evaluated (maximized) formula is presented in the first row of table 3.1. We need to add (modify) several lines but not the reporting line (line 12-16 in previous list of code). Firstly, add the line with mutual information (1a). Then modify line 25 and delete line 26. The important changes to the previous code are listed below. This is an implementation of the Battiti algorithm:

```
1a. matrix MI_FF = sum(sum( P_FF*log2(P_FF/((P_F^P_F)*Per(4)(0,2,1,3))) ));

2a. rep.log() << "Mutual information MI(F,C) = " << sorted << MI_FC << endl;
3a. rep.log() << "Mutual information MI(F,F) = " << MI_FF << endl;

4a.  S.reversion(true);

5a.  subdatset f = argmax(_F_, MI_FC );
6a.  while (!f.isnil()) {
7a.   F-=f;
8a.   S+=f;
9a.   if (completed(file)) break;
10a.    f = argmax(_F_, MI_FC-beta*sum(_S_,*MI_FF) );
11a.  }
12a. }
```

These two simple examples show that the Infosel++ interface can be a useful framework for developers for creating and testing algorithms based on probability distribution.

# Appendix A

# Project directory structure

```
build                :  project distribution building
   GNUC++            :  building with any GNU C++ compiler
     out             :  intermediate object files
   VisualC++         :  building with the Microsoft Visual C++ ver.8
dist                 :  project distribution directory
   bin               :  executable files
   lib               :  library/archive files
   include           :  C++ include files
   doc               :  documentation files
   src               :  sample source files
   make              :  sample make files
     out             :  intermediate object files
   test              :  command files for testing programs
   work              :  command files for running programs
prj                  :  main project directory
   library           :  source files of main library
     kernel          :  kernel source files
     frontend        :  front-end source files
   libpacks          :  standard package files with algorithm implementation
   libapps           :  source files of library applications
     stdcon          :  standard console source files
     samples         :  samples source files
   libtests          :  source files of library tests for integral
                                (all) and units: API, UI and front-end
   libdocs           :  source files of library documentation
     manual          :  manual source files
     design          :  library design documentation
     examples        :  example files for data, scripts and algorithm packs
     manual          :  manual source files
     reference       :  DOXYGEN reference files
```

# Appendix B

# Concise dictionary

Below we can find several definitions and concepts important to the understanding of the tutorial (*cf.* **[?]**).

F is a symbolical representation of a full set of features , where $F_i$ is a one feature, and $S_i = F/\{F_i\}$.

**Definition 1 (Strong relevance)** *A feature $F_i$ is strongly relevant if*

$$\mathbf{P}(C|F_i, S_i) \neq \mathbf{P}(C|S_i)$$

**Definition 2 (Weak relevance)** *A feature $F_i$ is weakly relevant if*

$$\mathbf{P}(C|F_i, S_i) = \mathbf{P}(C|S_i) \ \ and$$
$$\exists S_i^{'} \subset S, \ such \ \ as \ \mathbf{P}(C|F_i, S_i^{'}) \neq \mathbf{P}(C|S_i^{'})$$

**Corollary 1 (Irrelevance)** *A feature $F_i$ is irrelevant if*

$$\forall S_i^{'} \subseteq S, \mathbf{P}(C|F_i, S_i^{'}) = \mathbf{P}(C|S_i^{'}) \tag{B.1}$$

**Definition 3 (Markov Blanket)** *Given a feature $F_i$, let $M_i \subseteq F(F_i not \in M_i)$, $M_i$ is said to be a Markov Blanket for $F_i$ if*

$$\mathbf{P}(F - M_i - \{F_i\}, C|F_i, M_i) \ = \ \mathbf{P}(F - M_i - \{F_i\}, C|M_i) \tag{B.2}$$

**Definition 4 (Redundant feature)** *Let G be the current set of features. A feature is redundant and hence should be removed from G if it is weakly relevant and has a Markov Blanket $M_i$ within G .*

**Definition 5 (C-correlation)** *The correlation between any feature $F_i$ and class C is called C-correlation, denoted by $SUC_{i,c}$*

**Definition 6 (F-correlation)** *The correlation between any pair of features $F_i$ and $F_j$ ($i \neq j$) is called F-correlation, denoted by $SUC_{i,j}$.*

**Definition 7 (Approximate Markov Blanket)** *For two relevant features $F_i$ and $F_j$ ($i \neq j$), $F_j$ forms an approximate Markov Blanket for $F_i$ if $SUC_{i,c} \geq SUC_{i,c}$ and $SUC_{i,j} \geq SUC_{i,c}$ .*

**Definition 8 (Predominant feature)** *A relevant feature is predominant if it does not have any approximate Markov Blanket in the current set.*

# Appendix C

# Command line parameters

Synatx of parameters recognized by infosel++ aplication:

```
-h|help [file_name]
  Displays this help info or writes it to a given file.

-s|scr|script file_name
  Performs processing of a given script file.

-in|input|data file_name
  Sets up input data file.

-out|output|report file_name
  Sets up output report file.

-p|prec|precision real_val
  Sets up real data precision.

-pm|pmeth|partition meth_name|meth_name(val_list)
  Sets up data partition method.

-cp|cpos|classpos int_val
  Sets up classes column position.

-lc|lastc|classislast bool_val
  Sets up a flag whether classes column is last.

-e|excl|excluded str_list
  Sets up labels for excluded classes.

-ne|note|notexcluded str_list
  Sets up labels for non excluded classes.

-m|merg|merged str_list
  Sets up labels for merged classes.

-nm|notm|notmerged str_list
  Sets up labels for non merged classes.

-exe|exec|exeselalg alg_tag|alg_tag(val_list)
  Executes a given selection algorithm.

-execall
  Executes all selection algorithms.

-silent
  Perform processing without displaying output info.

-verbose
  Performs processing displaying more output info.

-break
```

Breaks processing commands.

# Bibliography


[1] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Trans. Neural Networks*, vol. 5, July 1994.

[2] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Trans. on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 491–502, 2005.

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. F. Brian, *Numerical recipes in C, The art of scientific computing*. Cambridge, University Press, Cambridge, UK, 1988.

[4] W. Duch, "Filter methods," in *Feature extraction, foundations and applications* (I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, eds.), pp. 89–118, Berlin, Heidelberg, New York: Physica Verlag, Springer, 2006.

[5] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, M. Caligiuri, C. Bloomfield, and E. Lander, "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring," *Science*, vol. 286, pp. 531–537, 1999.

[6] C. Ding and H. Peng, "Minimum redundancy feature selection from microarray gene expression data," *Journal of Bioinformatics and Computational Biology*, vol. 3, pp. 185–205, June 2005.

[7] W. Duch, T. Wieczorek, J. Biesiada, and M. Blachnik, "Comparison of feature ranking methods based on information entropy," in *Proc. of Int. Joint Conf. on Neural Networks*, pp. 1415–1420, IEEE Press, 2004.

[8] T. Vilmansen, "Feature evaluation with measures of probabilistic dependence," *IEEE Transaction on Computers*, vol. 22, pp. 381–388, April 1973.

[9] N. Kwak and C.-H. Choi, "Input feature selection for classification problems," *Neural Networks, IEEE Transactions on*, vol. 13, pp. 143–159, Jan 2002.

[10] M. Tesmer and P. Este'vez, "AMIFS: Adaptive Feature Selection by Using Mutual Information," in *Proc. of Int. Joint Conf. on Neural Networks, Budapeszt*, pp. 1415–1420, IEEE Press, 2004.

[11] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of Machine Learning Research, JMLR 5*, pp. 1205–1224, Oct 2004.

[12] W. Duch and J. Biesiada, "Feature selection for high-dimensional data: A kolmogorov-smirnov correlation-based filter solution," in *Advances in Soft Computing*, pp. 95–104, Springer, 2005.

[13] J. Biesiada and W. Duch, "A Kolmogorov-Smirnov correlation-based filter solution for microarray gene expressions data," in *14th Int. Conference on Neural Information Processing (ICONIP07), LNCS*, vol. 4985, pp. 285–294, Springer, 2008.

[14] M. Blachnik, W. Duch, A. Kachel, and J. Biesiada, "Feature Selection for Supervised Classification: A Kolmogorov-Smirnov Class Correlation-Based Filter," in *AIMeth, Symposium On Methods Of Artificial Intelligence*, Gliwice, Poland, 10-19 November 2009.

[15] J. Biesiada and W. Duch, "Feature Selection for High-Dimensional Data: A Pearson Redundancy Based Filter," in *Advances in Soft Computing*, vol. 45, pp. 242–249, Springer, 2008.

[16] D. Koller and M. Sahami, "Toward optimal feature selection," in *ICML-96: Proc. of the 13th Int. Conf. on Machine Learning*, pp. 284–292, San Francisco, Morgan Kaufmann, 1996.

[17] E. Xing, M. Jordan, and R. Karp, "Feature selection for high-dimensional genomic microarray data," in *Proc. of the 8th Int. Conf. on Machine Learning (ICML2001)*, 2001.

[18] J. Lorenzo, M. Hermandez, and J. Mendez, "GD: A Measure based on Information Theory for Attribute Selection," in *Proc. Int. Conf. IBERAMIA, LNAI*, vol. 1484, Springer, 1998.

[19] W. J. Sacco and W. S. Copes, "Reduction of the class of feature evaluation techniques in pattern analysis," *Pattern Recognition*, vol. 4, pp. 331–332, October 1972.

[20] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[21] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

[22] M. Evans, N. Hastings, and B. Peacock, *Statistical Distributions*. John Wiley & Sons, New York, 2000.

[23] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973.

[24] R. L. de Mantaras, "A distance-based attribute selection measure for decision tree induction," *Machine Learning*, vol. 6, pp. 81–92, 1991.