A tutorial-based guide to the ROSETTA system:

A Rough Set Toolkit for Analysis of Data

by Torgeir R. Hvidsten

torgeir.r.hvidsten@umb.no

Edition 1: May, 2006

Edition 2: April, 2010

Edition 2.1 October 2013

Contents

Chapter 1 Introduction

1.1 Machine learning in molecular biology

Biology has traditionally focused on classifying living systems (hierarchically) into increasingly smaller parts, and on studying these parts separately. This reductionistic research approach has culminated in molecular biology, where single molecules in terms of genes and gene products have been studied independently. With the publishing of the first complete genome sequence in 1995 (the bacteria *Haemophilus influenzae Rd* (Fleischmann, Adams et al. 1995)), the premises for this research have changed. A number of genome sequencing projects are now providing researchers with the basic instructions for the operation of entire organisms at an increasing speed (Bernal, Ear et al. 2001). Thus, DNA sequence data to some degree has facilitated a transition from molecular *genetics* (i.e. the study of single genes) to *genomics* (i.e. the study of all genes in a genome). Genomics has undergone a subsequent change from the mapping and sequencing of genomes to the more complex task of understanding and determining, at a genome-wide scale, gene and protein function, protein-protein interaction, protein-ligand interactions, gene regulation, etc. This part of genomics has been coined *functional genomics*.

The development of functional genomics and high-throughput experimental technologies created the need for computers to store and analyze large amounts of data. As was the case for genomics, *bioinformatics* developed from being a discipline mainly associated with sequence databases and sequence analysis to a computational science using biological data to do e.g. functional genomics. Although different definitions and views of bioinformatics exist, most researchers now use bioinformatics as a generic term for both the storage and maintenance of biological data and the use of computational data analysis methods and algorithms in functional genomics-related studies (Kanehisa and Bork 2003). Bioinformatics thus involves a number of scientific fields including mathematics, statistics, informatics, physics, chemistry, biology and medicine.

One commonly used methodology in bioinformatics and functional genomics is that of *machine learning*. Machine learning addresses the problem of using computers to *learn* general concepts from observations and knowledge, and has traditionally been developed in two different schools. Statisticians develop learning methods based on the mathematical frameworks of probability theory and statistics(Hastie, Tibshirani et al. 2001). Computer scientists often develop methods based on models of intelligent systems (e.g. methods inspired by biology such as genetic algorithms and neural networks, or methods based on logic such as rule learning, see the section on machine learning below) (Mitchell 1997). The differences are primarily due to the fact that statisticians have mostly been interested in pure data analysis, while computer scientist have also been interested in building intelligent systems (e.g. robots with *artificial intelligence* (Russell and Norvig 1995)). However, these different views are somewhat converging, forming hybrids using elements from both statistics and computer science (e.g. *pattern recognition* (Theodoridis and Koutroumbas 2003)).

Induction refers to generalizing from observations to broad concepts and differs from *deduction* that refers to using general concepts (or theories) to infer specific hypotheses. In molecular biology, induction is particularly relevant since the general theories have not yet been worked out. For example, we know that a relationship exists between sequence and structure, but this relationship is not well understood in terms of theories that may be used to deduce good structural models for a particular protein sequence. However, we do have examples of this relationship in terms of protein structures that are experimentally solved. And machine learning methods are designed to induce models based on *examples*, partially describing the assumed underlying functional relationship between, in this case, sequence and structure. The most common application of such models is that of prediction. However, given a model that can reliably predict protein structure from sequence (in particular for unseen proteins, i.e. proteins that were not available when the model was induced), this model obviously includes general concepts that may also be used to understand the relationship. And this understanding may in time lead to general theories. Consequently, machine learning may be used both for *predictive* and for *descriptive* purposes. In molecular biology, and in particular in functional genomics, a number of problems may be addressed using the concepts of examples and machine learning. And successful application of such methods could lead to situations where biological experiments are used to obtain information on a (representative) set of cases, models are automatically induced from these examples and finally used to fill in the missing knowledge for the remaining cases. This is the philosophy of structural genomics: to solve the structure of at least one protein from each protein family experimentally and to predict the structure of the remaining proteins using sequence similarity to proteins with solved structures (Chandonia and Brenner 2006).

One of the major obstacles for effective use of machine learning in functional genomics has been the lack of structure in the existing biological knowledge in terms of computer readable databases and annotations. Text mining and automatic inference from free text has therefore been one major part of bioinformatics and will continue to be so (Shatkay and Feldman 2003). Thus, controlled vocabularies such as Gene Ontology (Ashburner, Ball et al. 2000) for protein function has been important for machine learning approaches to biology.

1.2 Rough set-based rule learning and the ROSETTA system

Rough set-based rule learning has proven to be a particularly successful approach in bioinformatics. The approach condenses tabular data into IF-THEN rules. The IF-part of each rule specifies a minimal pattern needed to discern observations with different labels, e.g.

- **IF** Gene A is up-regulated **AND** Gene D is down-regulated

  THEN Tissue is healthy
- **IF** Transcription factor F bind **AND** Transcription factor V bind

  THEN Gene is co-regulated with Gene H
- **IF** Protein structure include motif D **AND** water-octanol coefficient of ligand > c

  THEN Binding affinity is high

Unlike most machine learning method, rule-based models are easily legible and may thus be used to understand the underlying pattern in data in addition to be used for prediction. The rough set framework is moreover particularly suited for handling noise and ambiguous data by inducing approximate models in terms of models and rules that have multiple outcomes.

The ROSETTA system is a software package that implements rough set-based rule induction and include a number of additional features such as model validation. This system is implemented with a user friendly graphical interface and is used by a large community of scientists. Examples of applications in bioinformatics include:

- Cancer classification (Nørsett, Lægreid et al. 2004; Dennis, Hvidsten et al. 2005)
- Gene function prediction (Lægreid, Hvidsten et al. 2003)
- Gene regulation (Hvidsten, Wilczynski et al. 2005)
- Protein-ligand interaction modeling (Strömbergsson, Kryshtafovych et al. 2006; Strömbergsson, Prusis et al. 2006)

See Chapter 7 for a complete list of applications.

1.3 Readers guide

Chapter 2 gives a general introduction to different machine learning method, putting the current approach in context.

Chapter 3 gives an introduction to rough set-based rule induction using the example of cancer classification.

Chapter 4 introduces the ROSETTA system and guides the user trough a step-by-step tutorial using data from a published cancer classification study.

Chapter 5 continuous the tutorial of Chapter 4 and introduce more advanced features such as cross validation, indiscernibility graphs and ROC analysis.

Chapter 6 shortly describes how the ROSETTA system can be run in command-line.

Chapter 7 gives a list of bioinformatics challenges were the ROSETTA system has been applied.

Chapter 8 gives guidelines and further references for the advanced user.

Chapter 2 Machine learning – an introduction

Machine learning deals with the problem of using computers to learn general concepts from *training sets*. A training set consists of a finite number of observations labeled or annotated with class knowledge and is assumed to constitute a partial description of an underlying functional relationship between the observations and the classes. In general, the labels may be continuous values or even more complex structures. However, here we will deal with the so called *classification* problem in which the training observations are assumed to belong to a finite set of classes and we want to learn a model or *classifier* capable of assigning an observation to one of these classes. Moreover, we will in general assume two classes, since problems with more than two classes easily may be reduced to a set of two-class problems.

Most machine learning methods represent the observations in terms of *features*. Each observation is a set of measurements, one for each such feature, collectively constituting a *feature vector*. Each observation may alternatively be viewed as a point in the multidimensional space spanned by the features (i.e. the *feature space*). Of course, not all classification problems are easily represented in this way, and choosing the right features is a very important issue specific to each classification problem.

The machine learning methods mainly differ in how they represent the induced model. A number of different designs exist with different advantages and disadvantages. A short overview will be given in the next paragraphs (Mitchell 1997; Theodoridis and Koutroumbas 2003).

2.1 Clustering methods

Methods for discovering natural, underlying classes from a set of observations are called *clustering* or *unsupervised learning*. These methods are used when no class knowledge is available. Consequently, methods utilizing labeled training sets are called *supervised learning* reflecting the conceptual idea that a supervisor provides the labels to the learning system.

Clustering methods are divided into *iterative* methods and *hierarchical* methods. The *k-means* algorithm is the most used iterative approach. It starts with a set of *k* randomly chosen clusters of observations and iteratively (a) calculates the center of each cluster (i.e. the *centroid*), (b) assigns each observation to the cluster defined by the closest centroid and (c) returns to (a) until no more observations change clusters. The centroid of a cluster and the closeness of two observations may easily be calculated in the feature space by using e.g. the notion of distance. The *k*-means algorithm is fast and uses little memory, but depends on the initial number and configuration of clusters. A well known related method is that of *self-organizing maps*.

The most popular hierarchical clustering method is *agglomerative* hierarchical clustering. It starts with the observations as single clusters and subsequently merges the two most similar clusters until all observations reside within one big cluster. The distance between two clusters may easily be calculated as the average distance between all pairs of observations in the two clusters (*average linkage*) or the longest/shortest distance between two observations in the two clusters (*complete/single linkage*). The result of the algorithm is a tree of clusters (*dendrogram*) illuminating the similarity structures in the data set. Since the method needs to compute and store the distance between all clusters, it is much slower and uses much more memory than for example the *k*-means algorithm.

2.2 Bayes classification rule

The *Bayes classification rule* states that an observation should be assigned to the class with the highest probability given the probability distribution of feature vectors in each class. It may be proven that this rule results in an optimal *error rate* for classification (i.e. fraction of training observations classified to the wrong class). However, the true probability distribution is normally not known and hence needs to be estimated. The difficulty of estimating the distributions from the training data is why other methods exist and often perform better on real world problems.

There are two basic concepts for estimating probability distributions from data; *parametric* and *non-parametric* methods. A parametric method assumes a distribution structure (e.g. the normal distribution) and calculates its parameters from the data (e.g. average and variance for the one-dimensional normal distribution). A non-parametric method is based on constructing *histograms* from the data using for example Parzen windows or k nearest neighbor density estimation, or simulation methods such as Monte Carlo simulation or bootstrapping. In the one dimensional case, a histogram is constructed by dividing the observations into bins and using the fraction of observations from each bin as probability estimates. In the multidimensional case, however, bins are replaced by hypercubes (e.g. *Parzen windows*). If N observations are needed from each bin to get good probability estimates in the one dimensional case, N^n observations are needed in the n-dimensional case. The dramatic increase in the number of observations needed to get good estimates is often referred to as the "*curse of dimensionality*".

2.3 Linear classifiers

Linear classifiers use a line (in two dimensions) or a hyperplane (in multiple dimensions) to separate two classes of observations in feature space. These methods generally consist of a cost function (e.g. error rate) and an optimization algorithm which iteratively changes the parameters defining the hyperplane so that the cost function is minimized over the training set.

2.4 Non-linear classifiers

If linear classifiers do not yield good results, the problem might be that the classes are not linearly separable. *Artificial neural networks* (ANNs) are one popular method for nonlinear problems and are based on networks of so-called *perceptrons*. A perceptron is a simple computational unit that multiplies each input value with a weight and sums up the products. In principle, the output from the perceptron is 0 if the sum is less than a particular threshold and 1 otherwise. ANNs consist of layers of perceptrons, where the output of each perceptron in one layer is connected to the input of each perceptron in the next layer. The first layer (i.e. the *input layer*) consists of the same number of perceptrons as the number of features and the last layer (i.e. the *output layer*) consists, in the case of two classes, of one perceptron. The network is trained by iteratively inputting the feature vectors to the first layer, calculating the output of each perceptron until the last perceptron, comparing the output value with the true class label and updating the weights for each perceptron by propagating the error backwards in the network (the *backpropagation algorithm*). The training stops when the network is no longer improving its classification.

Another popular method for nonlinear problems is (nonlinear) *support vector machines* (SVMs). The SVMs first map the observations in the feature space into another space using a *kernel function*. A maximally separating hyperplane is then constructed based on the observations closest to the region that separates the two classes (the *support vectors*). The performance of SVMs greatly relies on the choice of kernel function and to what degree the kernel function is able to map the original classification problem into a linearly separable one.

2.5 Context-dependent classifiers

A classifier is *context dependent* if the classification does not only depend on the feature vector of one observation, but also on the feature vectors of the other observations and on the dependencies between the classes. The task then becomes to simultaneously assign a class sequence to a sequence of observations. This corresponds to the problem of optimally aligning two sequences and therefore often occurs in DNA and amino acid sequence analysis. One of the most common approaches to this problem is to assume that the class of one observation only depends on the class of the previous observation. This model is called a (first-order) *Markov model* and may be utilized to find the optimal class sequence with a reasonable amount of computation (using e.g. dynamic programming).

2.6 *k*-nearest neighbor classifiers

k-nearest neighbor approaches are based on classifying observations according to the class labels of the *k* closest training observations in the feature space. This is probably the simplest and most intuitive approach among all supervised methods, and is therefore commonly used.

2.7 Decision trees

Decision trees and *rule-based* classifiers work on discrete (i.e. categorical) values or by dividing the feature space into boxes (two dimensions) or hypercubes (multiple dimensions), and by combining these into complex decision surfaces (i.e. surfaces in the feature space separating the classes).

Decision trees classify observations by sorting them down a tree from the root node to the leaf nodes, where the leaf nodes actually provide the classification. Each node corresponds to a feature and redirects the observations to different child nodes depending on their values for that feature. The tree is constructed top-down by iteratively selecting the most class-separating feature as a node.

2.8 Rule-based classifiers

A related approach is that of learning a set of IF-THEN rules. Note that a decision tree may be represented as a set of rules by translating each path in the tree (from root to leaf) into a rule. A number of rule learners exist, and a more detailed description of rough set-based rule learning will be given in section Chapter 3.

2.9 Feature selection

Feature selection refers to the problem of selecting the most important features so as to reduce their number and at the same time retaining class separability allowing classification. There are a number of reasons for doing feature selection. The obvious reason relates to reducing the computational cost of inducing classifiers. However, more important is the fact that the number of features translates directly into the number of *classifier parameters* (e.g. the number of perceptron/weights in an artificial neural network). And there is a fundamental principle in machine learning stating that the higher the ratio between the numbers of training examples and the numbers of classifier parameters, the better the induced classifier will perform on unseen observations (e.g. more observations per dimension/feature gives better estimates of the probability distribution and hence better performance using Bayes classification rule).

There are two broad approaches to feature selection. *Filter* methods select features according to some evaluation criterion (e.g. correlation between the feature and the class knowledge) and then induce a classifier based on these features. *Wrapper* methods use the classifier itself as the evaluation criterion, and select the features that result in the best classification performance.

Feature generation/extraction refers to constructing new features based on different combinations of the old features. One example is rotating the feature space to possibly obtain better class separation (e.g. using principle component analysis).

2.10 Bootstrapping, bagging and boosting

Bootstrapping (Efron and Tibshirani 1993) is a general re-sampling method that allows statistical inference about a summary statistic (e.g. sample mean) from a data set without knowing the sample distribution. The idea is to randomly draw with replacement a large number of new data sets from the original data set and to calculate the summary statistic from each such bootstrap sample. This provides several values for the summary statistic which may be used to infer for example its variance or confidence interval.

Bagging (Breiman 1996) and *boosting* (Schapire 1990) are general methods for improving the classification performance of any supervised method. Bagging (bootstrap aggregation) uses bootstrapping to sample a large number of training sets from the original set of examples. A model is induced from each such bootstrap sample and combined (aggregated) during classification to obtain what is often a better classification performance. Boosting is a similar method in which a weight is associated with each training example. Models are iteratively induced from the training set according to these weights and used to re-classify the examples. The weights are subsequently updated to put more emphasis on incorrectly classified examples. If the applied learning method cannot utilize the weights directly, bootstrap training sets may be constructed according to the weights (i.e. each example is drawn with a probability corresponding to the weight).

2.11 Genetic algorithms

Genetic algorithms are used to solve search problems where solutions can be coded as strings of 0's and 1's. An initial population of solutions is generated randomly and the best solutions, according to some fitness function, are iteratively chosen to breed new generations of solutions using genetic operators such as mutation and crossover. Supervised learning involves a number of search problems that may easily be approached with genetic algorithms. One example is feature selection, where each solution may be interpreted as a mask for including or excluding features.

2.12 Time complexity

The *big O* notation is used to describe the worst case running time of an algorithm as a function of its input size *n*. For example, the agglomerative hierarchical clustering algorithm using single linkage has a time complexity of $O(n^2)$ (i.e. it computes the "all-against-all" distance between observations in feature space). Hence, if 100 observations take 10 seconds to cluster, then 10000 observations (which is a typical number of genes in a microarray experiment) take 27.8 hours.

Algorithms that have a worst case running time of $O(n^k)$, where k is a constant, are so-called polynomial-time algorithms. Problems for which no polynomial-time algorithm has yet been discovered are said to belong to the class of *NP-complete* problems (NP stands for non-polynomial). Such problems need to be approached with approximation algorithms that find "good enough" solutions. For example, finding the optimal subset of features (which is the goal of features selection discussed earlier) is NP-complete (i.e. it requires searching trough all 2^n-1 subsets and hence has a time complexity of $O(2^n)$). Feature selection may for example be approached with the wrapper method using a genetic algorithm, or with the filter method using the correlation coefficient between each feature and the class labels. The latter approach of reducing a multi-dimensional problem into considering one dimension at a time (starting with the "best" dimension) is often referred to as a *greedy* approach.

2.13 Classifier evaluation

A classifier is best evaluated by applying it to a set of unseen observations (i.e. a *test set*). To obtain good estimates of the true classification performance it is important to use a test set that is representative for the observations that the classifier is likely to encounter in the future. In practice, it is common to divide the available labeled observations (i.e. examples) randomly into a training set and a test set. The training set is used to induce a classifier and the test set is used for estimating the classification performance. If few observations are available, which is commonly the case, *cross validation* may get the most out of the data in terms of performance estimation. *k*-fold cross validation refers to dividing the examples into *k* equally sized subsets and using one subset for testing and the rest for training. This is done repeatedly so that each subset acts as a test set once and is part of the training set k-1 times. If *k* equals the number of examples, this method is referred to as *leave-one-out* cross validation. To get good estimates of the classifier performance it is important that information contained in the test set is not used in the training. For example, feature selection should be done after splitting the available examples into training and test sets. Doing feature selection on all available examples implies using the class knowledge contained in future test sets to induce the classifier and hence may lead to optimistic estimates of the true classification performance.

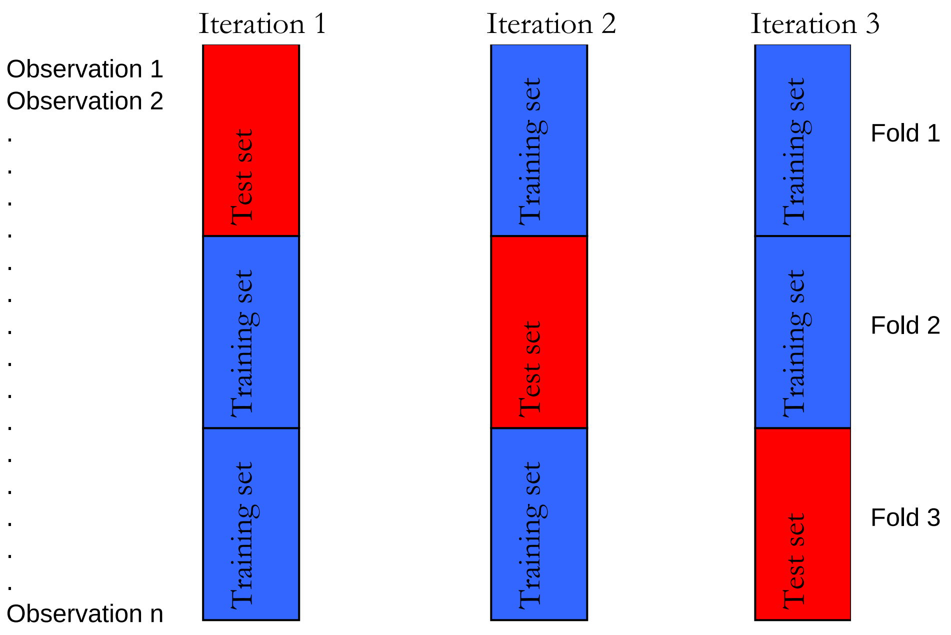


Figure 1 3-fold cross validation.

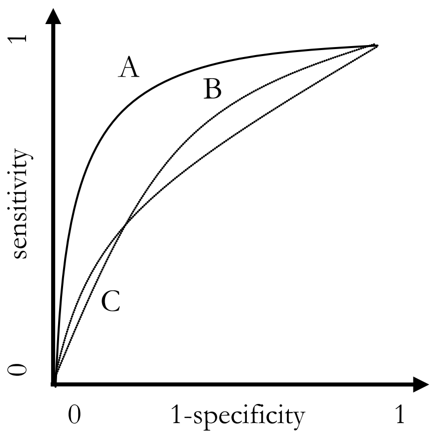


Figure 2 Example ROC curves. Clearly, classifier A performs better than both B and C. However, classifier B only performs better than C on low threshold values, while C performs better than B on high threshold values. Nonetheless, the AUC value of B is larger than that of C.

2.14 Performance measures and ROC analysis

A number of statistics exist for measuring the performance of a classifier on a test set. *Accuracy* is simply the fraction of test observations classified to the correct class (error rate = 1-accuracy). However, accuracy may provide insufficient information when the classes contain different numbers of examples or when making one type of error is more severe than making another.

Given two classes of positive and negative observations,

- *false positives* (FP) are negative observations classified to the positive class,
- *false negatives* (FN) are positive observations classified to the negative class,
- *true positives* (TP) are correctly classified positive observations and
- *true negatives* (TN) are correctly classified negative observations.

This information may be organized in a confusion matrix (Table 1).

Table 1 The confusion matrix

		Predicted	
		Negatives	**Positives**
Actual	**Negatives**	TN	FP
	Positives	FN	TP

Furthermore, sensitivity and specificity are the fractions of correctly classified positive and negative observations, respectively (i.e. TP/(TP+FN) and TN/(TN+FP)). Many classification methods do not perform classification directly, but rather output a value representing the certainty that a test observation belongs to the positive class. Hence, we are left with the problem of choosing a certainty threshold for selecting the positive class as the classification. The receiver operating characteristic (ROC) curve may be constructed by plotting sensitivity against specificity for the full range of possible threshold values (see Figure 2). A number of classification applications are associated with different costs for making a false positive classification compared to making a false negative classification. The ROC curve graphically displays the threshold-independent classification performance and provides a vehicle for controlling the number of false positives and false negatives. Increasing the threshold value reduces the number of false positives, but at the same time increases the number of false negatives. The *area under the ROC curve* (AUC, (Hanley and McNeil 1982)) is often used to measure the threshold independent classification performance using one single number (i.e. AUC equal to 1 signifies a perfect discrimination of the positive and negative examples, while AUC equal to 0.5 signifies no discriminatory capability at all). The standard error of this measure is calculated using the Hanley-McNeil formula (Hanley and McNeil 1982). However, one should be aware that two ROC curves obtained using two competing classifiers may intersect and hence indicate that one classifier performs better for one range of threshold values, while the other performs better for another range of threshold values (see Figure 2). This information is of course lost when computing the AUC value.

2.15 Overfitting and classifier selection

A classifier is said to *overfit* the training set if there exists another classifier that performs worse on the training set, but better on the test set. A general principle for handling overfitting is related to the principle of *Occam's razor* which states that the simplest model fitting the data should be used. Hence, according to this principle we should for example use the artificial neural network with the fewest perceptrons classifying the training set satisfactorily. This principle also applies to choosing a classification method. One should for example avoid using a nonlinear method on a linearly separable classification problem. This is of course related to the principle that the ratio between the number of training observations and the number of classifier parameters should be as large as possible (see the discussion in the feature selection paragraph above).

Chapter 3 Rough set-based rule learning

Pawlak's *rough set theory* (Pawlak 1982; Pawlak 1991; Komorowski, Pawlak et al. 1999) and *Boolean reasoning* (Brown 1990) constitute a mathematical framework for inducing rules from examples. It is this framework that is implemented in the ROSETTA system (Øhrn 1999; Komorowski, Øhrn et al. 2002).

This chapter will go through the core elements of this framework using examples. Chapter 3 will then describe how this framework is implemented in the ROSETTA system, and lead you through a step-by-step tutorial based on an extended version of the example used here.

Table 2 An example decision table.

		Conditional attributes				**Decision attribute**
	Patients	**Gene1**	**Gene2**	**Gene3**	**Smoking**	**Site of origin**
Objects (i.e. observations)	**P1**	↓	↓	0	Yes	Lung
	P2	0	0	0	Yes	Lung
	P3	0	↓	↑	No	Colon
	P4	0	0	0	Yes	Lung
	P5	0	↓	0	Yes	Lung
	P6	↓	↓	0	Yes	Lung
	P7	↓	↑	0	No	Colon
	P8	↓	↑	0	No	Colon
	P9	0	↑	0	Yes	Colon
	P10	↓	↓	↑	No	Lung
	P11	0	↓	0	Yes	Lung
	P12	0	↓	0	Yes	Lung
	P13	0	↓	↑	No	Colon
	P14	0	↑	↑	No	Colon
	P15	↓	↑	0	No	Colon
	P16	↓	↓	↑	No	Colon
	P17	0	↓	0	Yes	Lung
	P18	0	↓	↑	No	Lung

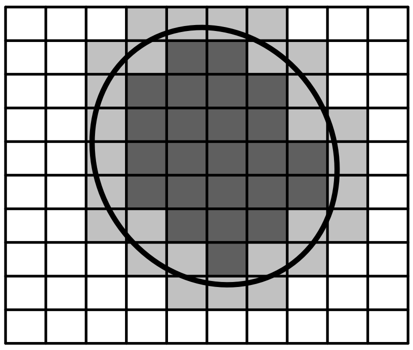


Figure 3. The rough set (the ellipse) cannot be uniquely defined by the equivalence classes (the squares), and is defined by the lower approximation (dark grey) and the upper approximation (dark plus light grey).

3.1 The rough set theory

The *rough set theory* is a mathematical framework for analyzing tabular data. An *information system* is a table with observations (called *objects*) as rows, features (called *attributes*) as columns and discrete values as entries. The theory sees the data in terms of *equivalence classes*, i.e. sets of objects that are indiscernible (indistinguishable) with respect to the attributes. A rough set is a set of objects that cannot be uniquely represented by these equivalence classes since the set only partly overlaps with at least one of them. It may hence only be approximately described either by the equivalence classes completely contained in the set (the *lower approximation*) or the equivalence classes with at least one object in the set (the *upper approximation*) (see Figure 3).

The *decision attribute* is a unique attribute dividing the objects into *decision classes* and is provided by domain experts or a separate source of information. The information system with the decision attribute constitutes the training set or the so-called *decision system* (e.g. Table 2). In particular, decision classes may be rough in which case the class knowledge itself cannot be uniquely represented using the data in the information table.

Example 1

Table 2 is an example of a decision system. 18 cancer patients (i.e. *objects*) are divided into two groups (i.e. *decision classes*) according to the location of the original tumor: lung (L) or colon (C). Four properties (i.e. *attributes*) are recorded about the patients: the expression level of three genes in the metastatic tumor compared to healthy tissue (0: unchanged, ↓ down-regulated, ↑ up-regulated) and whether the patient smokes or not.

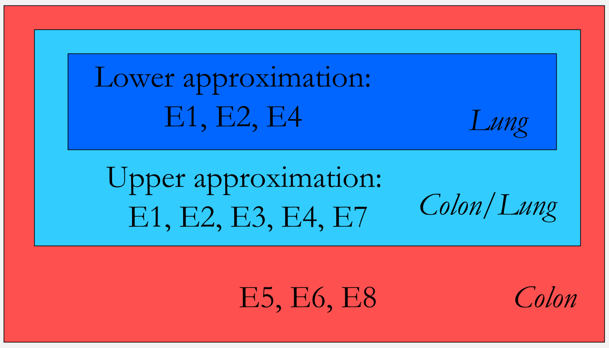
Note for example that:

- Patients P5, P11, P12 and P17 are indiscernible with respect to the recorded attributes and therefore constitute an *equivalence class*.
- Patients P3, P13 and P18 also constitute an equivalence class. However, these patients belong to different decision classes. The set of all decision classes related to an equivalence class (i.e. {C, L} in this case) is called the *generalized decision* of this equivalence class.

The decision system may be summaries in terms of equivalence classes as follows:

Equivalence classes	Gene1	Gene2	Gene3	Sm.	Site of origin (Generalized decision)
E1 = {P1,P6}	↓	↓	0	Yes	{L}
E2 = {P2, P4}	0	0	0	Yes	{L}
E3 = {P3,P13,P18}	0	↓	↑	No	{C, L}
E4 = {P5, P11, P12, P17}	0	↓	0	Yes	{L}
E5 = {P7, P8, P15}	↓	↑	0	No	{C}
E6 = {P9}	0	↑	0	Yes	{C}
E7 = {P10, P16}	↓	↓	↑	No	{C, L}
E8 = {P14}	0	↑	↑	No	{C}

The decision classes of lung and colon patients are *rough sets* because they cannot be defined uniquely in term of the equivalence classes. They can, however, be defined by an upper and lower approximation. Decision class L, for example, may be defined by those equivalence classes where all patents have decision class L, that is, the *lower approximation* (i.e. equivalence classes E1, E2 and E4), or by those equivalence classes that have at least one patient with decision class L, that is, the *upper approximation* (i.e. E1, E2, E3, E4 and E7):



Note that we in this example know which patients belong to which decision class. What we really are interested in here is whether these classes can be recognized given the data (i.e. the attributes).

3.2 Reducts

We will now see how *Boolean reasoning* can be used to reduce the data in a decision system. A *Boolean function* (i.e. a function that evaluates to true or false), called the *discernibility function*, is constructed for each object. This function is true for all attribute combinations that discern this object from objects with a different decision. The function is simplified and its minimal solutions interpreted as so-called *reducts* (Skowron and Rauszer 1992). A reduct is a minimal set of attributes discerning one object from all objects with a different decision. The reducts may be approximate (*approximate reducts or α-reduct*) in which case a sufficiently large fraction α of objects are discerned (Skowron and Nguyen 1999).

Finding *all* reducts is an NP-complete problem (Skowron and Rauszer 1992). However, there is a number of approximation algorithms, including greedy algorithms (Johnson 1974) and genetic algorithms (Vinterbo and Øhrn 2000), that may be used to search for reducts. They are all based on constructing the discernibility function which has a time complexity of $O(n^2)$.

Example 2

Again let us consider the equivalence class version of the decision system in Table 2:

Equivalence classes	Gene1	Gene2	Gene3	Sm.	Site of origin (generalized decision)
E1 = {P1,P6}	↓	↓	0	Yes	{L}
E2 = {P2, P4}	0	0	0	Yes	{L}
E3 = {P3,P13,P18}	0	↓	↑	No	{C, L}
E4 = {P5, P11, P12, P17}	0	↓	0	Yes	{L}
E5 = {P7, P8, P15}	↓	↑	0	No	{C}
E6 = {P9}	0	↑	0	Yes	{C}
E7 = {P10, P16}	↓	↓	↑	No	{C, L}
E8 = {P14}	0	↑	↑	No	{C}

We may now construct the discernibility function by first building a discernibility matrix specifying which attributes that discerns the different equivalence classes:

	E1	**E2**	**E3**	**E4**	**E5**	**E6**	**E7**	**E8**
E1	∅							
E2	∅	∅						
E3	G1, G3, S	G2, G3, S	∅					
E4	∅	∅	G3, S	∅				
E5	G2, S	G1, G2, S	G1, G2, G3	G1, G2, S	∅			
E6	G1, G2	G2	G2, G3, S	G2	∅	∅		
E7	G3, S	G1, G2, G3, S	∅	G1, G3, S	G2, G3	G1, G2, G3, S	∅	
E8	G1, G2, G3, S	G2, G3, S	G2	G2, G3, S	∅	∅	G1, G2	∅

Note for example, that:

- The entry E1-E1 is empty (∅) because, obviously, the equivalence classes E1 cannot be discerned from itself.
- The entry E1-E2 is also empty because we do not bother to discern equivalence classes with the same generalized decision.
- The entry E1-E3 has a different generalized decision and includes attributes Gene1 (G1), Gene3 (G3) and Smoking (S) for which equivalence classes E1 and E3 have different values, e. g. G1 is down-regulated for E1 while it is unchanged for E4.
- The discernibility matrix is symmetric, for example, entries E1-E4 and E4-E1 are identical and thus we only need to consider one half of the matrix.

The minimal information needed to discern E1 from *all* other objects with different decision may now be represented in the form of a *discernibility function*:

f_{E1} (G1, G2, G3, S) =

(G1 **OR** G3 **OR** S) **AND** (G2 **OR** S) **AND** (G1 **OR** G2) **AND** (G3 **OR** S) **AND** (G1 **OR** G2 **OR** G3 **OR** S)

In order for this function to be true, at least one of the attributes from each E1-related entry (blue) in the discernibility matrix need to be included. Thus, the function can be simplified to:

f_{E1} (G1, G2, G3, S) = (G1 AND S) OR (G1 AND S) OR (G2 AND S)

which reflects the three *reducts*: {G2, G3}, {G1, S}and {G2, S}. Note that each reduct contains the minimal set of attributes that are represented in all the ANDs in the discernibility function. Thus, they constitute the minimal information needed to discern equivalence class E1 from other patients not in decision class L.

Note that {S} is a possible *approximate reduct* or *α-reduct*. It would discern E1 from all equivalence classes except E6, i.e. it would discern 16 of 17 objects (α = 0.94). Approximate reducts are often applied in order to make the reducts more robust and less susceptible to noise or errors in the data.

Note also that:

- We could build a discernibility function from all entries in the discernibility matrix. The corresponding reduct would discern all objects from all other object with different generalized decision. Such reducts are called *full-reducts*, while the reducts found in the example above are called *object-related reducts*. The latter is most commonly used for inducing rule models.
- We could also choose to fill in entries in the discernibility matrix even if the generalized decisions are the same. This would result in reducts that discern objects even from the same decision classes. The matrix we constructed above is built *modulo decision*, which is the common choice when inducing rule models.
- Finally, we filled in all attributes in the matrix that had different attribute values for the two equivalence classes in question. Of course, more complex definitions of discernibility could be used, including definition that would make the discernibility matrix non-symmetric. For example, we could decide that we only want to base our decisions on whether genes are up-regulated or down-regulated: unchanged expression cannot discern two objects. In this case patient P1, for example, would be discernible from P2, but P2 would not be discernible from P1 (i.e. the discernibility matrix would be non-symmetric). Such definitions of discernibility are often used when the decision system includes *missing values* (e. g. the expression level of a gene could not be determined).

3.3 Decision rules

IF-THEN rules are constructed by reading of the values for each attribute in the reduct (IF-part called *antecedent* or *premise*, e.g. $a_1=v_1$ AND $a_2=v_2$, where a_1 and a_2 are attributes in a reduct and v_1 and v_2 are attribute values) and associating them with one or more decision classes (THEN-part called *consequent*, e.g. $d=d_1$ OR $d=d_2$, where d is the decision attribute and d_1 and d_2 are decision classes). The THEN-part will only include one decision class unless the decision class is rough with respect to the attributes in the reduct. Rules are evaluated according to how general they are (i.e. *coverage*: the fraction of objects from the decision class in the THEN-part that also matches the IF-part) and how specific they are (i.e. *accuracy*: the fraction of objects matching the IF-part that are from the decision class of the THEN-part) (both coverage and accuracy are computed for each decision class in the THEN-part).

Example 3

Lets consider the reducts {G2, G3}, {G1, S} and {G2, S} discerning equivalence class E1 in **Example 2**. These reducts would generate the following rules by reading of the attribute values for equivalence class E1:

		Support	Accuracy	Coverage
R1	**IF** Gene2(↓) AND Gene3(0) **THEN** Site (Lung)	6	1.0 (6/6)	0.60 (6/10)
R2	**IF** Gene1(↓) AND Smoking(Yes) **THEN** Site (Lung)	2	1.0 (2/2)	0.20 (2/10)
R3	**IF** Gene2(↓) AND Smoking(Yes) **THEN** Site (Lung)	6	1.0 (6/6)	0.60 (6/10)

Note, for example, that:

- The first rule, R1, matches 6 objects in Table 2 and thus its *support* is 6 (i.e. patients P1, P5, P6, P11, P12 and P17).
- Of all the 6 objects that match the IF-part of rule R1, all 6 are also members of the decision class in the THEN-part of rule R1 (i.e. decision class Lung). Thus the *accuracy* of the rule is 1.0.
- Of all the 10 objects matching the THEN-part of rule R1 (i.e. all objects from decision class Lung in Table 2), 6 matches the IF part of rule R1. Thus the coverage is 0.60, i.e. the rule describes 60% of the objects in the decision class Lung.
- We want accurate *and* general rules (high accuracy and high coverage), thus R1 may be considered a better rule than R2.

Moreover, the approximate reduct {S} from **Example 2** would generate the following rule:

IF Smoking(Yes) **THEN** Site (Lung) OR Site (Colon)

This rule have two values for support/accuracy/coverage: one for each decision class in the THEN-part:

	Lung	Colon
Support	8	1
Accuracy	0.89 (8/9)	0.11 (1/9)
Coverage	0.80 (8/10)	0.13 (1/8)

By repeating the procedure in Example 2 for all equivalence classes, the following additional rules would result (support, accuracy and coverage is given for lung first and colon second when applicable):

		Support	Accuracy	Coverage
R4	**IF** Gene2(0) **THEN** Site (Lung)	2	1.0	0.20
R5	**IF** Gene2(↓) AND Gene3(↑) **THEN** Site (Lung) **OR** Site (Colon)	2, 3	0.4, 0.6	0.20, 0.38
R6	**IF** Gene2(↓) AND Smoking(No) **THEN** Site (Lung) **OR** Site (Colon)	2, 3	0.4, 0.6	0.20, 0.38
R7	**IF** Gene3(0) AND Smoking(No) **THEN** Site (Colon)	3	1.0	0.38
R8	**IF** Gene2(↑) **THEN** Site (Colon)	5	1.0	0.60
R9	**IF** Gene1(↓) AND Gene3(↑) **THEN** Site (Lung) OR Site (Colon)	1, 1	0.5, 0.5	0.10, 0.13

Rules R1-9 thus constitutes our rule model and is a general description of the decision system in Table 2. This model have two main proposes:

- Predictive purpose: It can be used to predict the site of origin for new patients (see section 3.4).
- Descriptive purpose: It can be used to better understand what separates patients with different sites of origin. For example, lung cancer is strongly linked to smoking (especially apparent from the approximate reduct), but there is also strong patterns in the expression levels (especially that defined by rule R1). The strongest pattern for colon cancer is the up-regulation of Gene 2 (i.e. rule R8).

3.4 Classification

Objects are classified by first identifying the rules with a matching IF-part and then by letting these rules cast votes to the decision classes in the corresponding THEN-parts. The number of votes cast by each rule corresponded to the *support* of the rules (i.e. the number of objects matching both the IF- and THEN-part of the rule), giving preference to rules that are general.

When performing ROC analysis, the decision class(es) obtaining a fraction of votes higher than the voting thresholds from the ROC analysis are considered predictions (see section 2.14). If no particular costs are associated with making false positive classifications over false negative classifications, the threshold corresponding to the point on the ROC curve balancing sensitivity and specificity equally is often chosen (i.e. the point closest to (0,1) or, equivalently, the "northwestern-most" point on the ROC curve).

Example 4

Lets assume that we have two new patients with the following measurement values:

Patient	Gene1	Gene2	Gene3	Smoking	Site of origin
P19	↓	↓	↓	Yes	Unknown
P20	↓	↑	↑	Yes	Unknown

Patient P19 would match rules R2 and R3 in **Example 3,** casting 2 and 6 votes to decision class Lung, respectively. Thus 8 out of 8 votes would say that lung is the site of origin.

Patient P20 would match rules R2, R8 and R9. R2 casts 2 votes for Lung, R8 casts 5 votes for Colon, while R9 casts one vote for Lung and one for Colon. Thus Lung receives 3 out of 9 votes (0.33), while Colon receives 6 out of 9 votes (0.66).

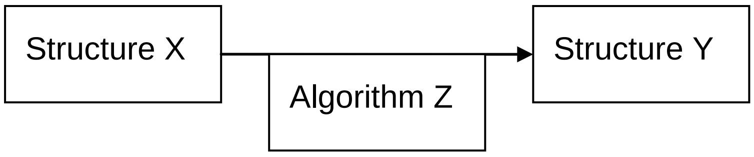
If we consider making a wrong classification to decision class L equally costly as making a wrong classification to decision class C, we would use a decision threshold equal to 0.5 for both classes (this corresponds to choosing the class with the highest fraction of cotes). Thus we would predict lung as the site of origin for patient P19 and colon for patient P20.

However, we might consider a wrong classification to decision class C much more costly than a wrong classification to L. Let us assume that the rule system is used to decide whether to look for the original tumor in the lung first or in the colon first. Lets further assume that the latter is considerable more expensive. Thus, we decide that the decision threshold for Colon is 0.7 and for Lung 0.3. In this case, both patients P18 and P19 would be predicted to class L.

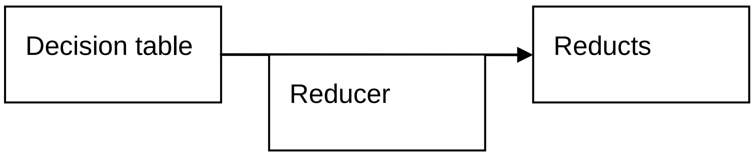
Chapter 4 The ROSETTA system

The ROSETTA system is a software package for inducing rough-set based rule models as described in Chapter 2. In addition to the core features described there, the system includes a large number of algorithms for discretization, reduct computation, rule pruning and classifier evaluation. All these features are facilitated by a graphical user interface available for Windows. However, the system also includes a command line version that has been compiled on many different platforms, including Unix and Linux.

The system has two main components: *structures* and *algorithms*. Structures are different data sets such as decision systems, reducts, rules, etc. Algorithms are applied to structures to produce new structures. For example, algorithms for reduct computation are applied to decision tables to produce reducts:



E.g.



In addition, the ROSETTA system includes a number of meta-algorithms that consists of an ordered assembly of several algorithms to accomplish commonly used tasks. This includes an algorithm for doing cross validation (described in detail in Chapter 5) and a number of algorithms specifically written for common bioinformatics tasks (described in a number of chapters starting with Chapter 6).

In this chapter, tutorial 1 gives a step-by-step instruction to how the ROSETTA system can be used to induce rule models and how rule models can be used for prediction. The tutorial is based on a published study (Dennis, Hvidsten et al. 2005), and the data is the real world version of the toy example used to illustrate the rough set-based rule learning framework in Chapter 3.

Tutorial I: Markers for the site of origin of metastatic Adenocarcinoma

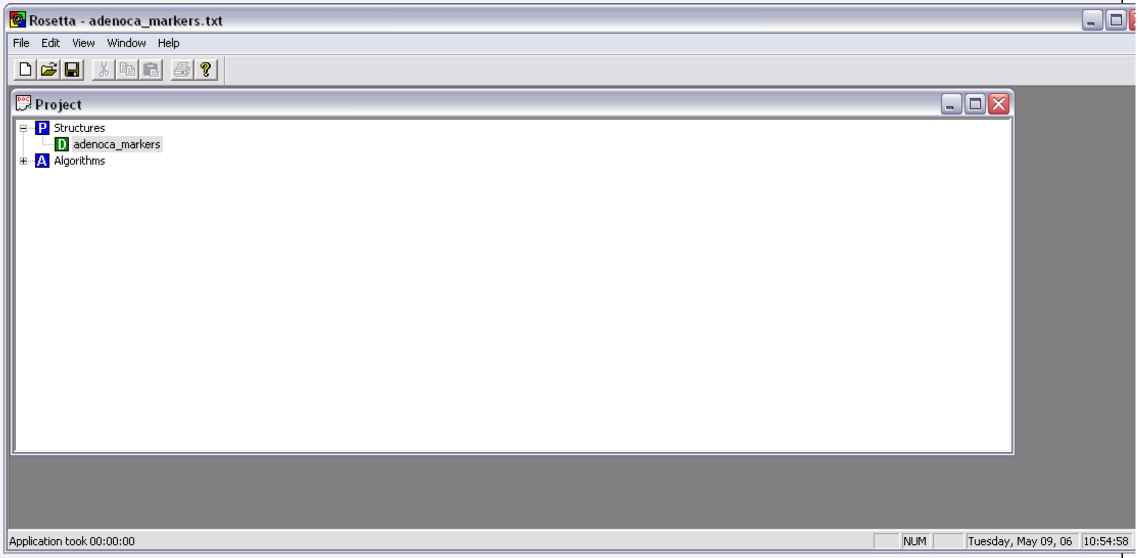
Background: Most cancers present at their site of origin – that is, it is the primary tumor which causes symptoms in the patient who then attends their doctor. Some 10-15% of cancers, however, present as metastases in solid organs, body cavities or lymph nodes. Most of these secondary tumors are adenocarcinomas, for which the seven commonest primary sites are breast, colon, lung, ovary, pancreas, prostate and stomach. The prognosis and therapy of patients with metastatic adenocarcinoma are linked to the site of origin, so these sites, and others, are investigated by clinical examination, radiology and serum tumor markers. If no primary cancer is found, then the metastatic deposit is usually biopsied, to confirm the diagnosis of malignancy and to subtype the tumor. Unfortunately, adenocarcinomas from different locations have similar microscopic appearances, which confound identification of the primary site. Patients with metastatic adenocarcinoma of unknown origin make up around 3% of all cancer patients and this category is among the ten most common malignancies.

Data: The expression patterns of 27 markers were assessed in a series of 261 adenocarcinomas. 12 markers were scored as either present or absent (+ or -). The remaining markers showed variation in intensity between tumors and were scored as weak, intermediate or strong (*0*, *1*, *2* or *3*). Furthermore, *Undefined* indicates cores which are missing and therefore cannot be scored.

Aim: To predict the site of origin using the expression profile of the 27 candidate markers taken from the secondary tumor.

How?

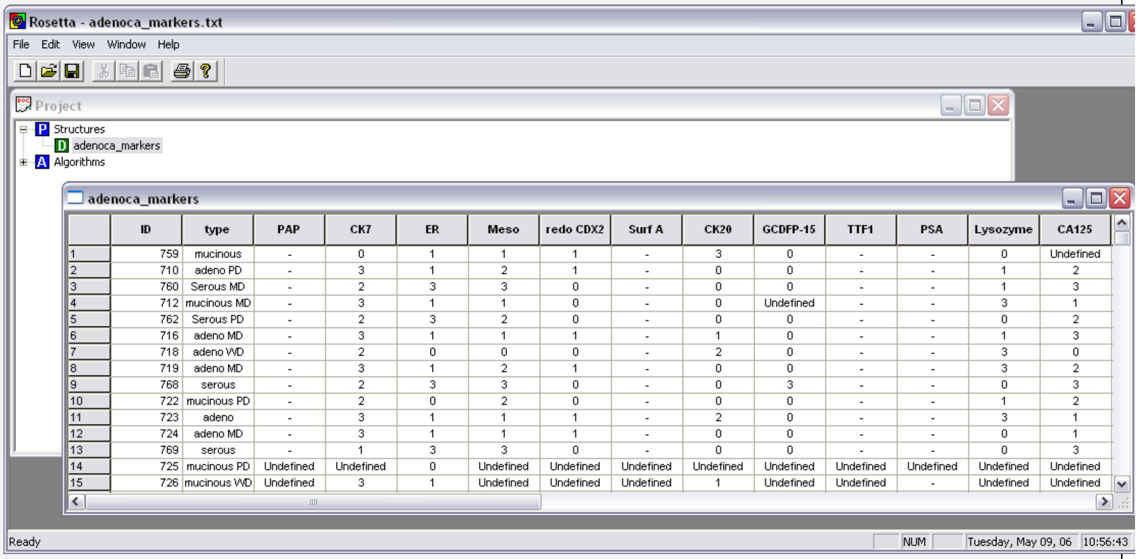
To import the dataset (i.e. the decision table), select **Open...** from the main **File** menu. Locate the file **adenoca_markers.txt** on your disk drive, select **Plain format** and press **OK** twice. The decision table will now be placed immediately below the root of the **Structures** node in the project tree:


Rosetta - adenoca_markers.txt
File Edit View Window Help
Project
Structures
adenoca_markers
Algorithms
Application took 00:00:00
NUM
Tuesday, May 09, 06
10:54:58

Note:

- **Plain format** refers to a decision table that is stored in a plain text file.
- Other formats, such as Excel sheets can be imported using the **Decision table importer (ODBC)** option rather than the **Plain format** option.
- You can change the name of structures in the project tree (as done in the screenshot above) by selecting the structure, then clicking the structure once and typing the new name. **Note: Be sure to update the structure names as shown in the screenshots throughout this tutorial.** The default names given by ROSETTA are not the names shown in the screen shots.

By double-clicking the decision table in the project tree, or, equivalently, right-clicking the decision table and selecting **View…,** you can browse the data:

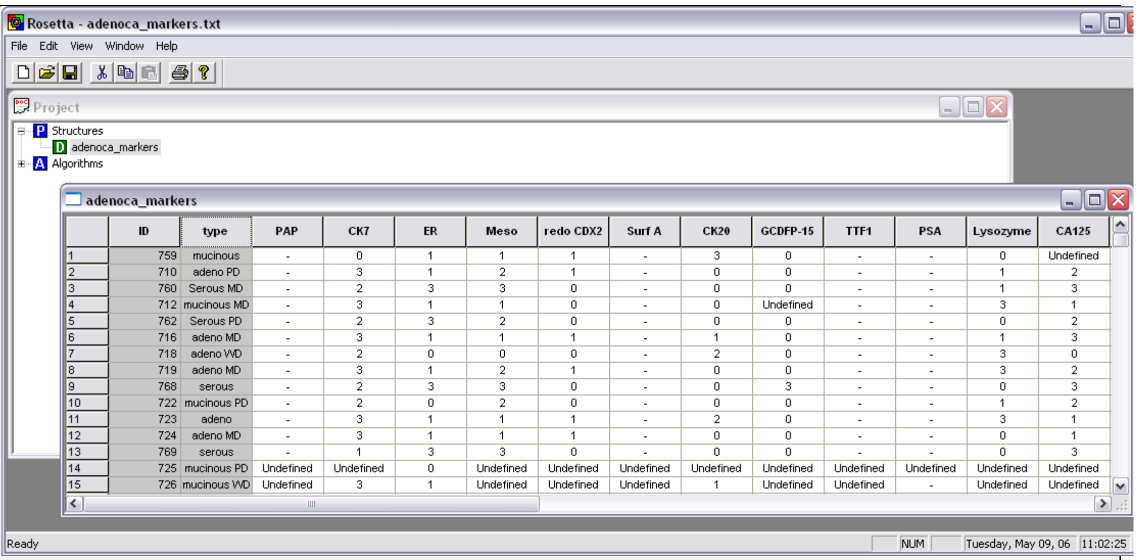


	ID	type	PAP	CK7	ER	Meso	redo CDX2	Surf A	CK20	GCDFP-15	TTF1	PSA	Lysozyme	CA125
1	759	mucinous	-	0	1	1	1	-	3	0	-	-	0	Undefined
2	710	adeno PD	-	3	1	2	1	-	0	0	-	-	1	2
3	760	Serous MD	-	2	3	3	0	-	0	0	-	-	1	3
4	712	mucinous MD	-	3	1	1	0	-	0	Undefined	-	-	3	1
5	762	Serous PD	-	2	3	2	0	-	0	0	-	-	0	2
6	716	adeno MD	-	3	1	1	1	-	1	0	-	-	1	3
7	718	adeno WD	-	2	0	0	0	-	2	0	-	-	3	0
8	719	adeno MD	-	3	1	2	1	-	0	0	-	-	3	2
9	768	serous	-	2	3	3	0	-	0	3	-	-	0	3
10	722	mucinous PD	-	2	0	2	0	-	0	0	-	-	1	2
11	723	adeno	-	3	1	1	1	-	2	0	-	-	3	1
12	724	adeno MD	-	3	1	1	1	-	0	0	-	-	0	1
13	769	serous	-	1	3	3	0	-	0	0	-	-	0	3
14	725	mucinous PD	Undefined	Undefined	0	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
15	726	mucinous WD	Undefined	3	1	Undefined	Undefined	Undefined	1	Undefined	Undefined	-	Undefined	Undefined

Ready NUM Tuesday, May 09, 06 10:56:43

By using the scroll-bar at the bottom of the window you can view all the attributes. Note that the last attribute is automatically interpreted as the decision attributes and is in bold.

Obiouslty, the two first attribute (ID and type) cannot be used to induce a general rule model. These attribute can be masked by right-clicking the attribute names and selecting **Masking…**, **Disable** and then **OK**. These attributes will now be grey to indicate that they will not take part in any further analysis:

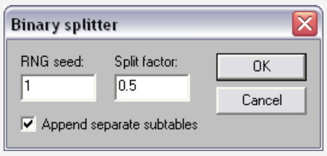


	ID	type	PAP	CK7	ER	Meso	redo CDX2	Surf A	CK20	GCDFP-15	TTF1	PSA	Lysozyme	CA125
1	759	mucinous	-	0	1	1	1	-	3	0	-	-	0	Undefined
2	710	adeno PD	-	3	1	2	1	-	0	0	-	-	1	2
3	760	Serous MD	-	2	3	3	0	-	0	0	-	-	1	3
4	712	mucinous MD	-	3	1	1	0	-	0	Undefined	-	-	3	1
5	762	Serous PD	-	2	3	2	0	-	0	0	-	-	0	2
6	716	adeno MD	-	3	1	1	1	-	1	0	-	-	1	3
7	718	adeno WD	-	2	0	0	0	-	2	0	-	-	3	0
8	719	adeno MD	-	3	1	2	1	-	0	0	-	-	3	2
9	768	serous	-	2	3	3	0	-	0	3	-	-	0	3
10	722	mucinous PD	-	2	0	2	0	-	0	0	-	-	1	2
11	723	adeno	-	3	1	1	1	-	2	0	-	-	3	1
12	724	adeno MD	-	3	1	1	1	-	0	0	-	-	0	1
13	769	serous	-	1	3	3	0	-	0	0	-	-	0	3
14	725	mucinous PD	Undefined	Undefined	0	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
15	726	mucinous WD	Undefined	3	1	Undefined	Undefined	Undefined	1	Undefined	Undefined	-	Undefined	Undefined

You can close the window by clicking the red cross in the upper right corner.

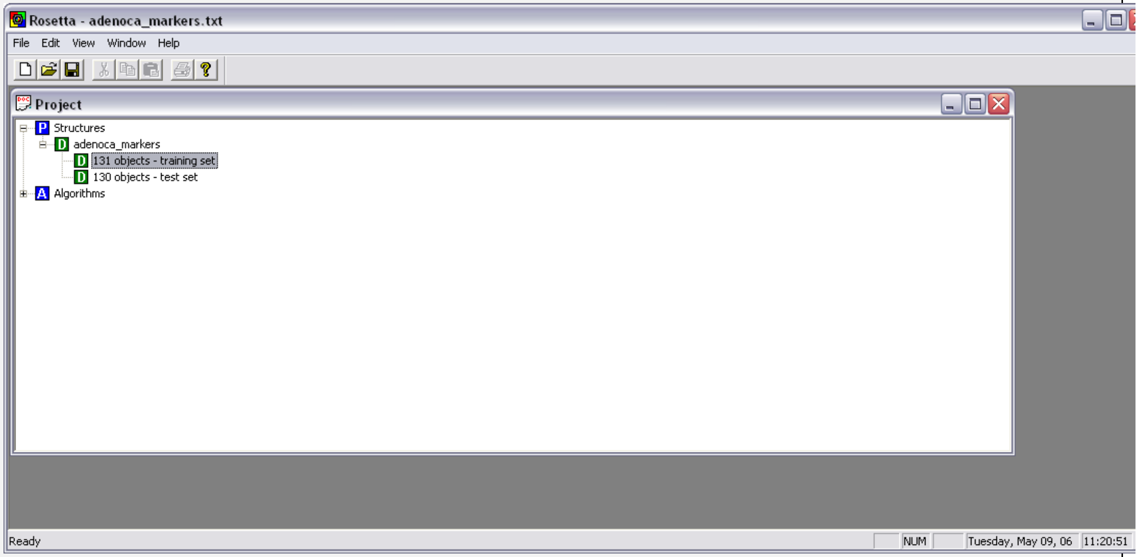
We now want to do a simple analysis in which we induce a rule model on one part of the data (*training set*) and use this model to classify the objects in the remaining data (*test set*).

Right-click the decision table, choose **Other** and select **Split in two…**:



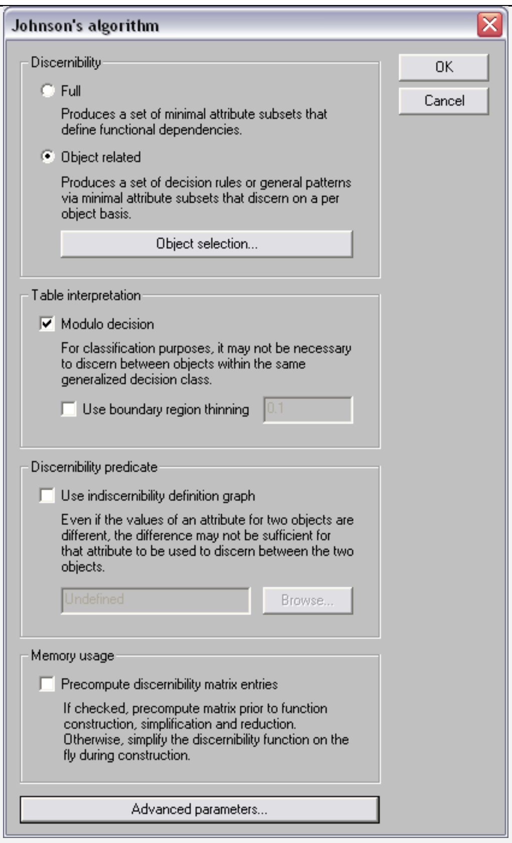
RNG seed: 1
Split factor: 0.5
☑ Append separate subtables

The **Split fraction** is the fraction of data that should be part of the training set. The **RNG seed** is the input to the random number generator that produces the random slit of the data: the same seed will produce the same split, and thus this number is used for reproducibility. Leave the values unchanged and press **OK**:



Structures
adenoca_markers
131 objects - training set
130 objects - test set
Algorithms

To induce rules, right-click the training set, choose **Reduce** and select **Johnson's algorithm…** This algorithm is very fast because it uses a greedy search to find one reduct (or one reduct per object in the case of object-related reducts):



Johnson's algorithm

Discernibility

- ( ) Full — Produces a set of minimal attribute subsets that define functional dependencies.
- (●) Object related — Produces a set of decision rules or general patterns via minimal attribute subsets that discern on a per object basis.

Table interpretation

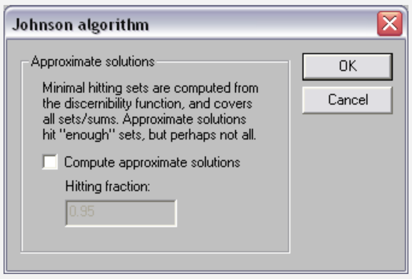
- [✓] Modulo decision — For classification purposes, it may not be necessary to discern between objects within the same generalized decision class.
- [ ] Use boundary region thinning: 0.1

Discernibility predicate

- [ ] Use indiscernibility definition graph — Even if the values of an attribute for two objects are different, the difference may not be sufficient for that attribute to be used to discern between the two objects.
- Undefined

Memory usage

- [ ] Precompute discernibility matrix entries — If checked, precompute matrix prior to function construction, simplification and reduction. Otherwise, simplify the discernibility function on the fly during construction.



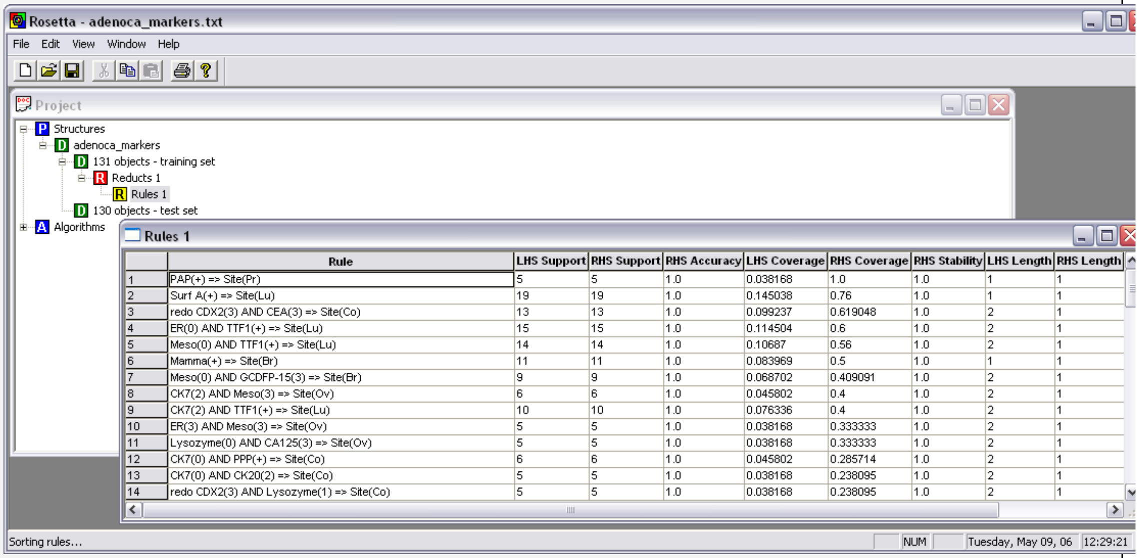
Johnson algorithm

Approximate solutions

Minimal hitting sets are computed from the discernibility function, and covers all sets/sums. Approximate solutions hit "enough" sets, but perhaps not all.

- [ ] Compute approximate solutions
- Hitting fraction: 0.95

Select Object related as above and make sure that **Approximate solutions** is turn off under **Advanced parameters …** Press **OK** twice. Reduct and Rule structures will now appear in the project tree under the training set (Click **+** to expand the tree if necessary). Double-click the rules:



Rosetta - adenoca_markers.txt

Project:
- Structures
  - adenoca_markers
    - 131 objects - training set
      - Reducts 1
        - Rules 1
    - 130 objects - test set
- Algorithms

Rules 1

	Rule	LHS Support	RHS Support	RHS Accuracy	LHS Coverage	RHS Coverage	RHS Stability	LHS Length	RHS Length
1	PAP(+) => Site(Pr)	5	5	1.0	0.038168	1.0	1.0	1	1
2	Surf A(+) => Site(Lu)	19	19	1.0	0.145038	0.76	1.0	1	1
3	redo CDX2(3) AND CEA(3) => Site(Co)	13	13	1.0	0.099237	0.619048	1.0	2	1
4	ER(0) AND TTF1(+) => Site(Lu)	15	15	1.0	0.114504	0.6	1.0	2	1
5	Meso(0) AND TTF1(+) => Site(Lu)	14	14	1.0	0.10687	0.56	1.0	2	1
6	Mamma(+) => Site(Br)	11	11	1.0	0.083969	0.5	1.0	1	1
7	Meso(0) AND GCDFP-15(3) => Site(Br)	9	9	1.0	0.068702	0.409091	1.0	2	1
8	CK7(2) AND Meso(3) => Site(Ov)	6	6	1.0	0.045802	0.4	1.0	2	1
9	CK7(2) AND TTF1(+) => Site(Lu)	10	10	1.0	0.076336	0.4	1.0	2	1
10	ER(3) AND Meso(3) => Site(Ov)	5	5	1.0	0.038168	0.333333	1.0	2	1
11	Lysozyme(0) AND CA125(3) => Site(Ov)	5	5	1.0	0.038168	0.333333	1.0	2	1
12	CK7(0) AND PPP(+) => Site(Co)	6	6	1.0	0.045802	0.285714	1.0	2	1
13	CK7(0) AND CK20(2) => Site(Co)	5	5	1.0	0.038168	0.238095	1.0	2	1
14	redo CDX2(3) AND Lysozyme(1) => Site(Co)	5	5	1.0	0.038168	0.238095	1.0	2	1

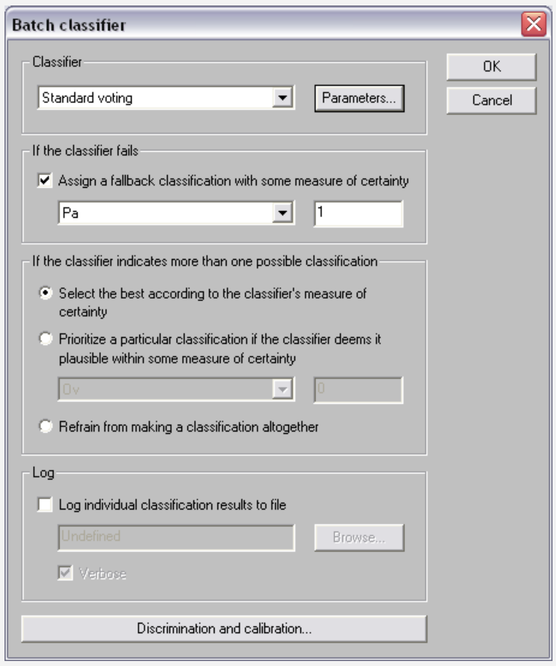
Sorting rules... NUM Tuesday, May 09, 06 12:29:21

In the screen shot above, the rules are sorted by **RHS Coverage** (i.e. RHS is Right Hand Side or THEN part of the rule, LHS is Left Hand Side or IF-part of the rule). To sort the rule, right-click the LHS Coverage tag and Select **Sort**. For each rule, the following statistics are given:

- **LHS support**: Number of objects in the training set matching the IF-part.

- **RHS support**: Number of objects in the training set matching the IF-part *and* the THEN-part (LHS and RHS support is the same unless the THEN-part contains several decisions).
- **RHS Accuracy**: RHS support divided by LHS support (Accuracy is 1.0 unless the THEN-part contains several decisions).
- **LHS Coverage**: LHS support divided by the number of objects in the training set.
- **RHS Coverage**: RHS Support divided by the number of objects in the decision class listed in the THEN part of the rule.
- **RHS Stability**: Not applicable for the Johnson algorithm (always 1.0).
- **LHS Length**: Number of attributes in the IF-part of the rule.
- **RHS Length**: Number of decisions in the THEN-part of the rule.

The rules can now be used to classify the objects in the test set. Right-click the test set and select **Classify…:**

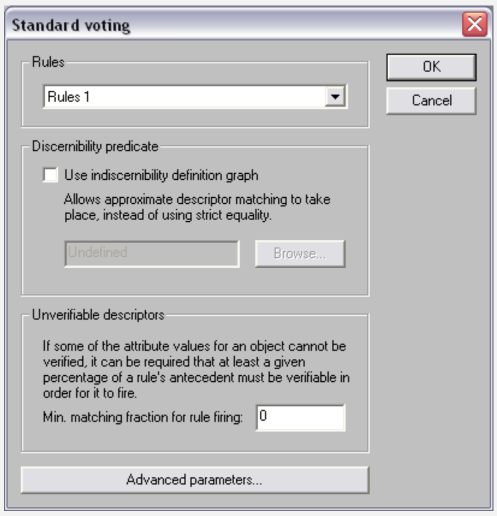

Batch classifier

Classifier: Standard voting

If the classifier fails
[✓] Assign a fallback classification with some measure of certainty
Pa 1

If the classifier indicates more than one possible classification
(●) Select the best according to the classifier's measure of certainty
( ) Prioritize a particular classification if the classifier deems it plausible within some measure of certainty
Ov 0
( ) Refrain from making a classification altogether

Log
[ ] Log individual classification results to file
Undefined
[✓] Verbose


Standard voting

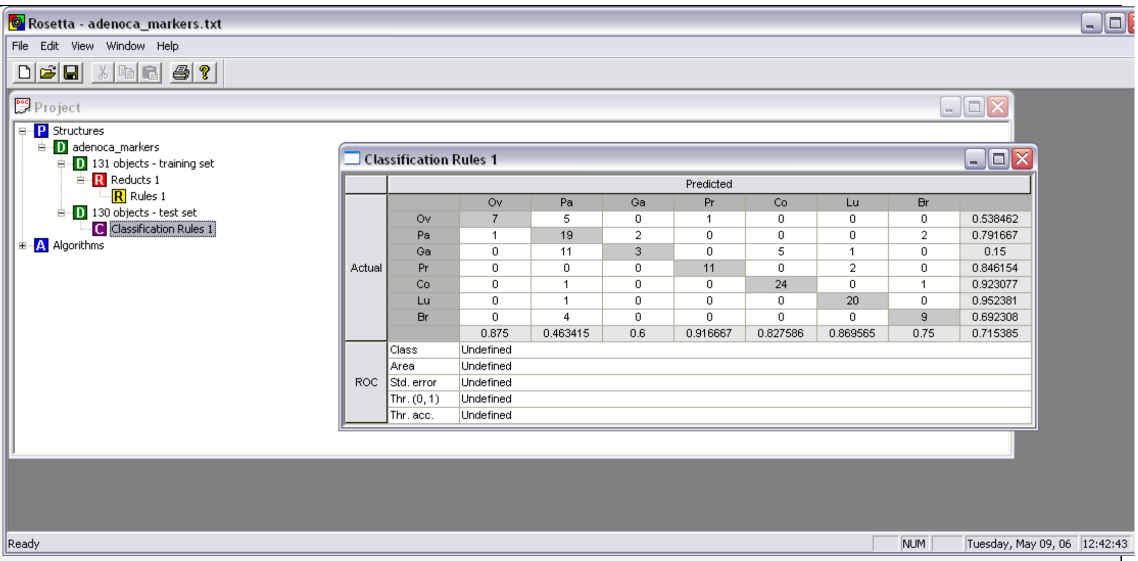
Rules: Rules 1

Discernibility predicate
[ ] Use indiscernibility definition graph
Allows approximate descriptor matching to take place, instead of using strict equality.
Undefined

Unverifiable descriptors
If some of the attribute values for an object cannot be verified, it can be required that at least a given percentage of a rule's antecedent must be verifiable in order for it to fire.
Min. matching fraction for rule firing: 0

Select **Standard voting** as the classification method and make sure that **Rules 1** is selected under **Parameters…** Also, choose **Pa** as your fallback classification. If no rules match an object, this class will be used. Press **OK**.

A new structure (a confusion matrix) will appear below the test set in the project tree. Double-click this structure to see the classification results:



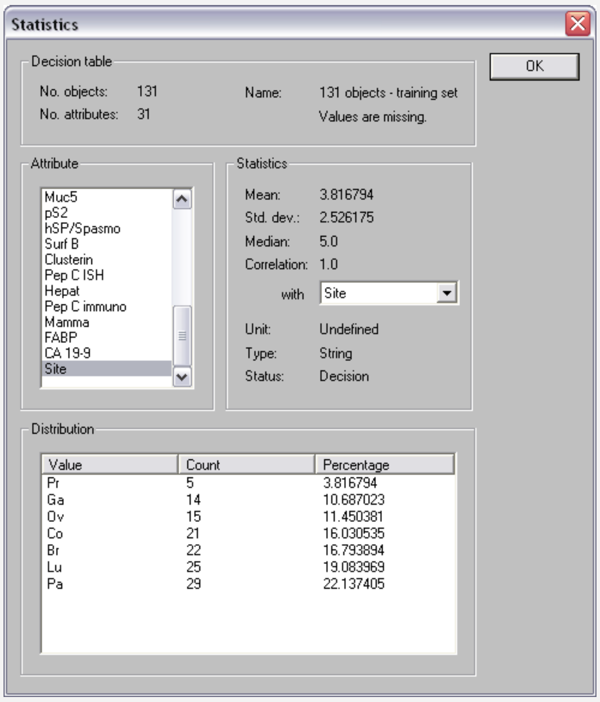
		Predicted							
		Ov	Pa	Ga	Pr	Co	Lu	Br	
Actual	Ov	7	5	0	1	0	0	0	0.538462
	Pa	1	19	2	0	0	0	2	0.791667
	Ga	0	11	3	0	5	1	0	0.15
	Pr	0	0	0	11	0	2	0	0.846154
	Co	0	1	0	0	24	0	1	0.923077
	Lu	0	1	0	0	0	20	0	0.952381
	Br	0	4	0	0	0	0	9	0.692308
		0.875	0.463415	0.6	0.916667	0.827586	0.869565	0.75	0.715385
ROC	Class	Undefined							
	Area	Undefined							
	Std. error	Undefined							
	Thr. (0, 1)	Undefined							
	Thr. acc.	Undefined							

The confusion matrix shows the overall accuracy (i.e. 0.715385), as well as the sensitivity and accuracy for each class. For example, the Ov decision class has a sensitivity of 0.54 (i.e. of 7+5+1 = 13 objects actually belonging to Ov, 7 was correctly classified as Ov: 7/13 = 0.54) and an accuracy of 0.88 (i.e. of 7+1 = 8 objects predicted to Ov, 7 were actually belonging to this class: 7/8 = 0.88).

You can save the whole project tree by selecting **Save as…** in the main **File** menu, and specifying the project name, e.g. **adenoca_markers.ros**.

The ROSETTA system includes a large number of other useful features not discussed here. For example:

- By right-clicking a structure and selecting **Statistics…** you can find useful information such as the class distribution in your training set:



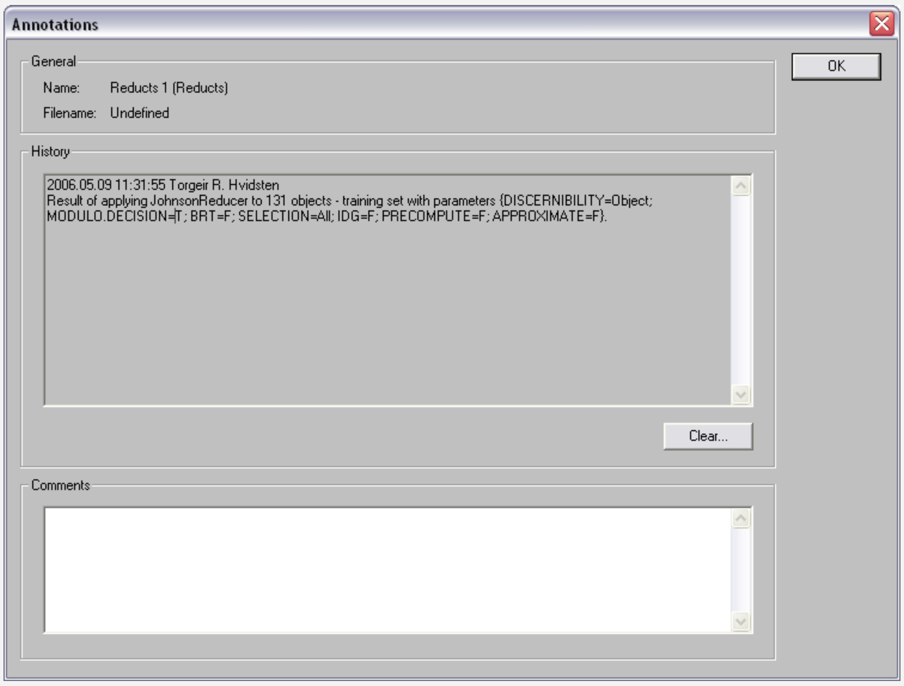
No. objects: 131
No. attributes: 31
Name: 131 objects - training set
Values are missing.

Attribute: Muc5, pS2, hSP/Spasmo, Surf B, Clusterin, Pep C ISH, Hepat, Pep C immuno, Mamma, FABP, CA 19-9, Site

Mean:	3.816794
Std. dev.:	2.526175
Median:	5.0
Correlation:	1.0
with	Site
Unit:	Undefined
Type:	String
Status:	Decision

Value	Count	Percentage
Pr	5	3.816794
Ga	14	10.687023
Ov	15	11.450381
Co	21	16.030535
Br	22	16.793894
Lu	25	19.083969
Pa	29	22.137405

- By right-clicking a structure and selecting **Annotations…** you can see which algorithm and parameters you used to compute reducts:



General
Name: Reducts 1 (Reducts)
Filename: Undefined

History

```
2006.05.09 11:31:55 Torgeir R. Hvidsten
Result of applying JohnsonReducer to 131 objects - training set with parameters {DISCERNIBILITY=Object;
MODULO.DECISION=T; BRT=F; SELECTION=All; IDG=F; PRECOMPUTE=F; APPROXIMATE=F}.
```

- By right-clicking a structure you can also perform a number of other operations such as exporting the structure in various file formats, removing the structure from the project tree, duplicating the structure in the project tree (e.g. you might want to remove rules to see the effect on classification, while still holding on to the original rule set), etc.

Chapter 5 Model evaluation in the ROSETTA system

Model evaluation is essential in supervised learning. In Tutorial 1 in Chapter 4 we simply divided our data set into a training set and a test set and used the model induced from the training set to predict the outcome (decision) of the objects in the test set. However, dividing the data differently would most likely result in different performance as measured in e.g. accuracy. Thus we want to test our model on several test sets in order to estimate the variance of our performance measure. Also, there exists several different performance measures. In addition to the common accuracy measure, ROSETTA also supports Receiver Operating Characteristic (ROC) analysis (see section 2.14).

5.1 Cross validation

Cross validation is a technique that provides several test sets while fully utilizing all the available data for training and testing (see section 2.13 and Figure 1). The data is divided into *k* approximately equally sized subsets (i.e. folds). Each fold is then consecutively used as a test set while the remaining *k-1* folds are used as a training set. Thus each object appears in the test set once and in the training set *k-1* times. The prediction performance is recorded for each test set and variance is computed.

The Rosetta system provides a meta-algorithm for performing cross validation. It takes as input the specific algorithms that the user want to execute in each cross validation iteration, and then performs the cross validation automatically. Tutorial 2 gives a step-by-step introduction using the data set from Tutorial 1.

TUTORIAL 2: CROSS VALIDATION

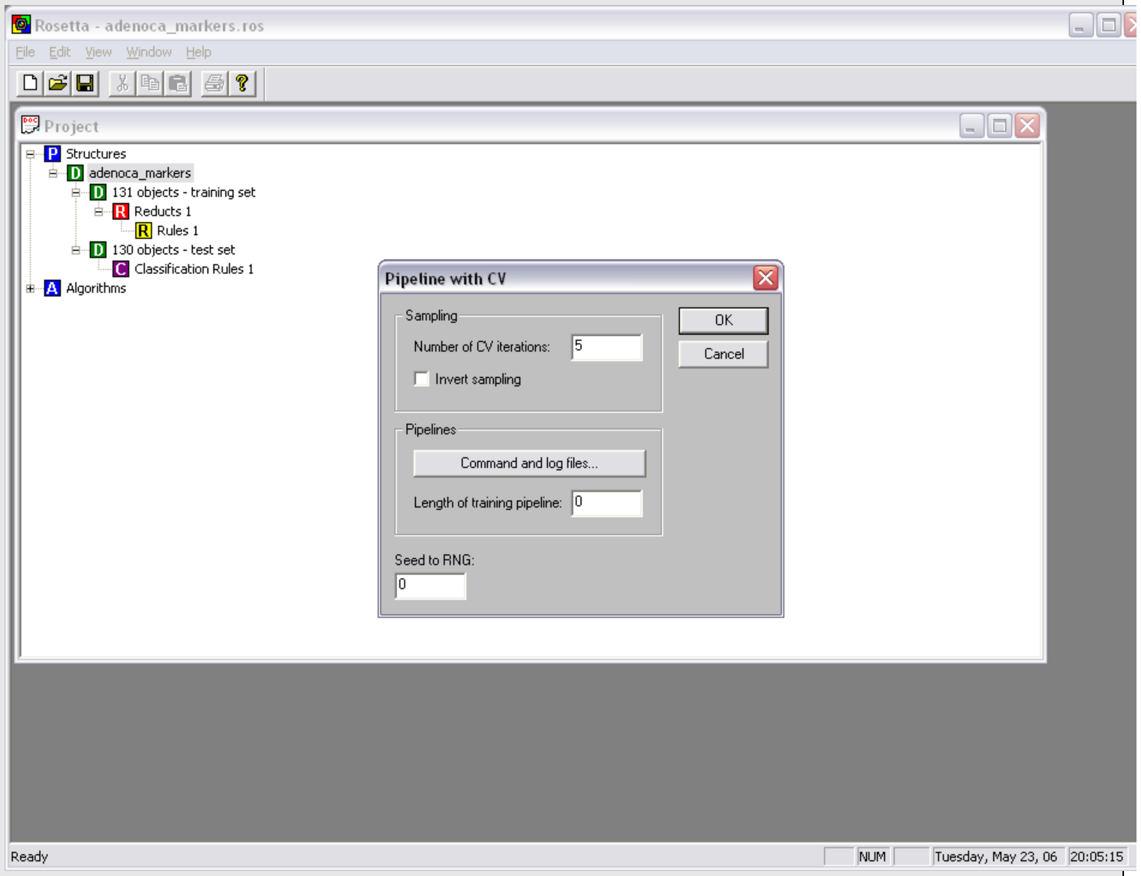
Background: See Tutorial 1.

Aim: To obtained reliable estimates of the prediction performance of the ROSETTA system applied to the metastatic adenocarcinoma data in Tutorial 1.

How?

Load the saved project from Tutorial 1 (or if this is not available, simply load the **adenoca_markers.txt** file).

Right-click the data set, choose **Execute** and select **Pipeline script with CV…**:



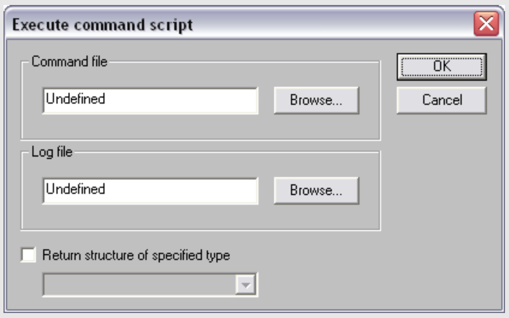
Number of CV iterations: 5

Length of training pipeline: 0

Seed to RNG: 0

The **Number of CV iterations** specifies the number of folds in the cross validation. The **Seed to RNG** option ensures reproducibility (i.e. the same RNG seed produce the same split into e.g. five folds).

Select **Command and log files …**:



Command file: Undefined

Log file: Undefined

The **Command file** is a text file that is expected to include a pipeline of algorithms to be executed in each iteration of the cross validation. The Log file is simply the file in which ROSETTA will output the results of the cross validation.

For example, the following command file will perform the exact same training and classification as was done in Tutorial 1:

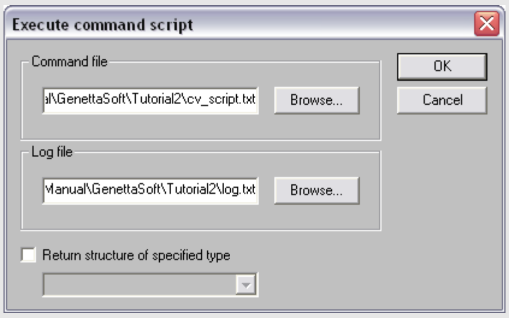
```
JohnsonReducer

{DISCERNIBILITY=Object; MODULO.DECISION=T; BRT=F;
SELECTION=All; IDG=F; PRECOMPUTE=F; APPROXIMATE=F}

BatchClassifier

{CLASSIFIER=StandardVoter; FRACTION=0.0; IDG=F; SPECIFIC=F;
VOTING=Support; NORMALIZATION=Firing; FALLBACK=T;
FALLBACK.CLASS=Pa; FALLBACK.CERTAINTY=1.0; MULTIPLE=Best;
LOG=F; CALIBRATION=F; ROC=F}
```

The first algorithm, the **JohnsonReducer**, searches for reducts and produces rules. It will be applied to each training set in each cross validation iteration. The second algorithm, the **BatchClassifier**, will use the rules from the **JohnsonReducer** algorithm to classify the test set in that iteration.

Note:

1. The script could potentially include many algorithms for the training set, the test set or both. These would be executed in sequence. The number of algorithms to be used on the trainings set is specified by the **Length of training pipeline** in the main cross validation window. The remaining algorithms will be used on the test set. In this example, the **Length of training pipeline** is 1**.**
2. The name and the parameters of the different algorithms may be obtained by first running the algorithm in the ROSETTA system, then right-clicking the resulting structure and selecting **Annotations…** For example, the annotations from the reduct structure in Tutorial 1 include the parameters for the **JohnsonReducer** shown above (see the last screenshot of Tutorial 1) and the confusion matrix include the parameters for the BatchClassifier.

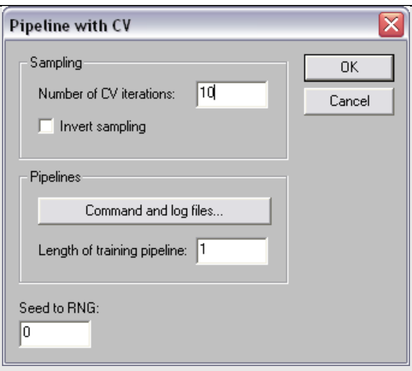
Save the commands in a text file using a text editor and specify a log file:



Command file: al\GenettaSoft\Tutorial2\cv_script.txt

Log file: Manual\GenettaSoft\Tutorial2\log.txt

Press **OK**, set the **Length of training pipeline** to 1 and the **Number of CV iterations** to 10:

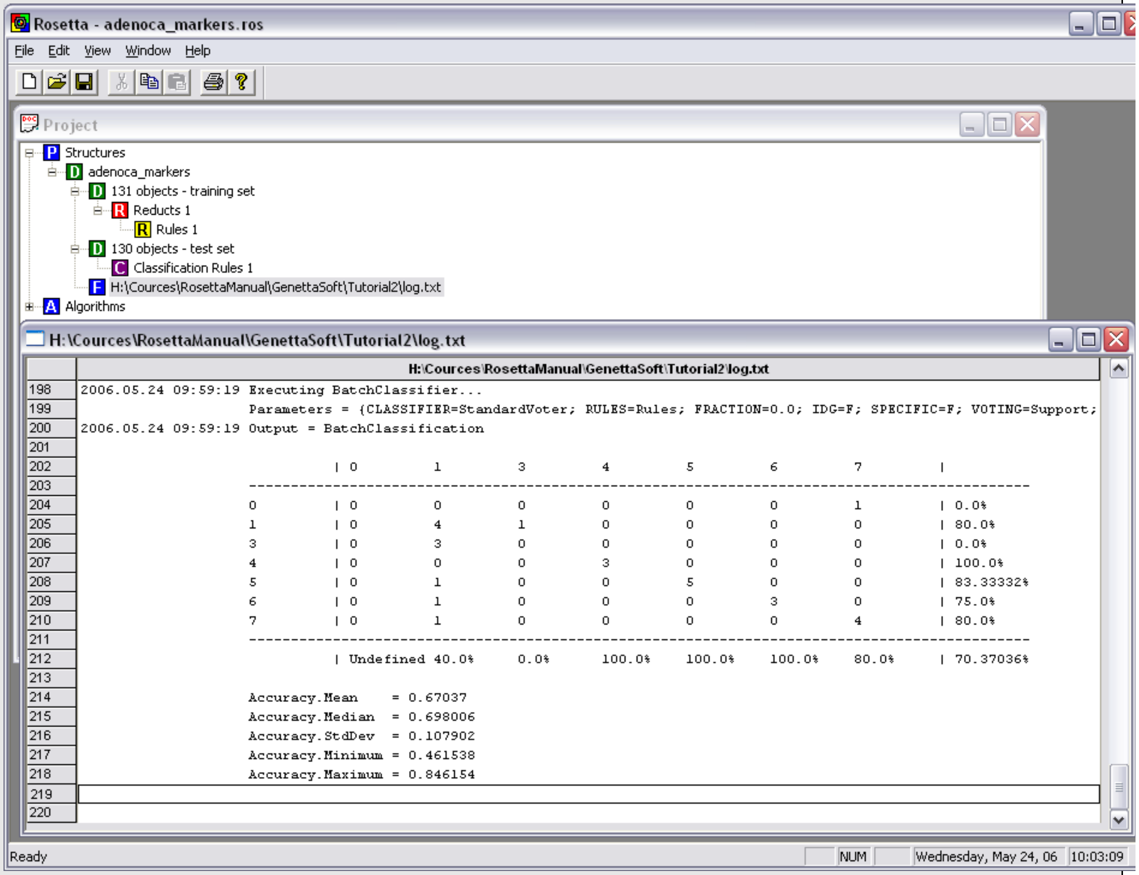


Sampling
Number of CV iterations: 10
Invert sampling [ ]

Pipelines
Length of training pipeline: 1

Seed to RNG: 0

Press **OK**. The cross validation log-file will now appear in your project file. Double-click to open it:


Rosetta - adenoca_markers.ros
File
Edit
View
Window
Help
Project
Structures
adenoca_markers
131 objects - training set
Reducts 1
Rules 1
130 objects - test set
Classification Rules 1
H:\Cources\RosettaManual\GenettaSoft\Tutorial2\log.txt
Algorithms
H:\Cources\RosettaManual\GenettaSoft\Tutorial2\log.txt
Ready
NUM
Wednesday, May 24, 06
10:03:09

```
2006.05.24 09:59:19 Executing BatchClassifier...
                    Parameters = {CLASSIFIER=StandardVoter; RULES=Rules; FRACTION=0.0; IDG=F; SPECIFIC=F; VOTING=Support;
2006.05.24 09:59:19 Output = BatchClassification

                              | 0         1         3         4         5         6         7         |
                    --------------------------------------------------------------------------------------------
                    0         | 0         0         0         0         0         0         1         | 0.0%
                    1         | 0         4         1         0         0         0         0         | 80.0%
                    3         | 0         3         0         0         0         0         0         | 0.0%
                    4         | 0         0         0         3         0         0         0         | 100.0%
                    5         | 0         1         0         0         5         0         0         | 83.33332%
                    6         | 0         1         0         0         0         3         0         | 75.0%
                    7         | 0         1         0         0         0         0         4         | 80.0%
                    --------------------------------------------------------------------------------------------
                              | Undefined 40.0%     0.0%      100.0%    100.0%    100.0%    80.0%     | 70.37036%

                    Accuracy.Mean    = 0.67037
                    Accuracy.Median  = 0.698006
                    Accuracy.StdDev  = 0.107902
                    Accuracy.Minimum = 0.461538
                    Accuracy.Maximum = 0.846154
```

The log-file includes a confusion matrix for each test set in each iteration of the cross validation. At the end of the file, the average accuracy and its standard deviation is given.

Chapter 6 Running the ROSETTA system command line

The ROSETTA system can be run command line with the syntax:

```
clrosetta algorithm parameters [filename]
```

TUTORIAL 3: COMMAND LINE CROSS VALIDATION

Tutorial 2 can be run command line with the following command:

Clrosetta CVSerialExecutor “INVERT=F; NUMBER=5; SEED=0; LENGTH=1; FILENAME.COMMANDS= cv_script.txt; FILENAME.LOG=log.txt” adenoca_markers.ros

where cv_script.txt is the command line from Tutorial 2:

```
JohnsonReducer

{DISCERNIBILITY=Object; MODULO.DECISION=T; BRT=F;
SELECTION=All; IDG=F; PRECOMPUTE=F; APPROXIMATE=F}

BatchClassifier
```

{CLASSIFIER=StandardVoter; FRACTION=0.0; IDG=F; SPECIFIC=F; VOTING=Support; NORMALIZATION=Firing; FALLBACK=T; FALLBACK.CLASS=Pa; FALLBACK.CERTAINTY=1.0; MULTIPLE=Best; LOG=F; CALIBRATION=F; ROC=F}

Note that adenoca_markers.ros is assumed to be in the internal ROSETTA format (i.e. saved from within ROSETTA).

TIPS: The name and the parameters of the different algorithms may be obtained by first running the algorithms in the ROSETTA system GUI, then right-clicking the resulting structure and selecting **Annotations…**

Chapter 7 Bioinformatics applications

Applications of the ROSETTA system spans a number of different bioinformatics areas:

- Medical diagnosis (Nørsett, Lægreid et al. 2004; Dennis, Hvidsten et al. 2005; Beisvag, Lehre et al. 2006; Zhou, Zhou et al. 2006)
- Gene function prediction from gene expression profiles (Hvidsten, Komorowski et al. 2001; Hvidsten, Laegreid et al. 2003; Lægreid, Hvidsten et al. 2003; Wabnik, Hvidsten et al. 2009)
- Protein function prediction from structure (Hvidsten, Laegreid et al. 2009)
- Gene regulation (Hvidsten, Wilczynski et al. 2005; Wilczynski, Hvidsten et al. 2006)
- Molecular interaction prediction (Strömbergsson, Kryshtafovych et al. 2006; Strömbergsson, Prusis et al. 2006; Kontijevskis, Wikberg et al. 2007; Kierczak, Rudnicki et al. 2008)
- Structure classification (Cao, Liu et al. 2006)

For a continuously updated list of bioinformatics papers that have used the ROSETTA system see: http://www.citeulike.org/user/TRHvidsten/tag/rosetta.

Chapter 8 Advanced use of the ROSETTA system

The ROSETTA system is an object library implemented in C++ and can be employed directly in software development (code available here: http://rosetta.sourceforge.net/). An overview of the classes can be found in *The ROSETTA C++ Library: Overview of Files and Classes* (Øhrn 2000).

A more comprehensive description of algorithms available in the ROSETTA system are found in the *ROSETTA Technical Reference Manual (Øhrn 1999).*

Also see the *GENOMIC ROSETTA - Application Mode User Manual* (Andersson and Vesterlund 2005) for details about meta-algorithms developed for bioinformatics-specific application of the ROSETTA system.

More information is found at the ROSETTA website:
http://www.lcb.uu.se/tools/rosetta/.

.

References

Andersson, R. and J. Vesterlund (2005). GENOMIC ROSETTA - Application Mode User Manual. Uppsala, The Linnaeus Centre for Bioinformatics.

Ashburner, M., C. A. Ball, et al. (2000). "Gene ontology: tool for the unification of biology. The Gene Ontology Consortium." Nat Genet **25**(1): 25-9.

Beisvag, V., P. K. Lehre, et al. (2006). "Aetiology-specific patterns in end-stage heart failure patients identified by functional annotation and classification of microarray data." Eur J Heart Fail **8**(4): 381-9.

Bernal, A., U. Ear, et al. (2001). "Genomes OnLine Database (GOLD): a monitor of genome projects world-wide." Nucleic Acids Res **29**(1): 126-7.

Breiman, L. (1996). "Bagging predictors." Machine learning **24**: 123-140.

Brown, F. M. (1990). Boolean reasoning : the logic of Boolean equations. Boston, Kluwer Academic Publishers.

Cao, Y., S. Liu, et al. (2006). "Prediction of protein structural class with Rough Sets." BMC Bioinformatics **7**: 20.

Chandonia, J. M. and S. E. Brenner (2006). "The impact of structural genomics: expectations and outcomes." Science **311**(5759): 347-51.

Dennis, J. L., T. R. Hvidsten, et al. (2005). "Markers of adenocarcinoma characteristic of the site of origin: development of a diagnostic algorithm." Clin Cancer Res **11**(10): 3766-72.

Efron, B. and R. J. Tibshirani (1993). An introduction to the Bootstrap. London, Chapman & Hall.

Fleischmann, R. D., M. D. Adams, et al. (1995). "Whole-genome random sequencing and assembly of Haemophilus influenzae Rd." Science **269**(5223): 496-512.

Hanley, J. A. and B. J. McNeil (1982). "The meaning and use of the area under a receiver operating characteristic (ROC) curve." Radiology **143**: 29-36.

Hastie, T., R. J. Tibshirani, et al. (2001). The Elements of Statistical Learning. New York, Springer.

Hvidsten, T. R., J. Komorowski, et al. (2001). "Predicting gene function from gene expressions and ontologies." Pac Symp Biocomput: 299-310.

Hvidsten, T. R., A. Laegreid, et al. (2003). "Learning rule-based models of biological process from gene expression time profiles using gene ontology." Bioinformatics **19**(9): 1116-23.

Hvidsten, T. R., A. Laegreid, et al. (2009). "A comprehensive analysis of the structure-function relationship in proteins based on local structure similarity." PLoS One **4**(7): e6266.

Hvidsten, T. R., B. Wilczynski, et al. (2005). "Discovering regulatory binding-site modules using rule-based learning." Genome Res **15**(6): 856-66.

Johnson, D. S. (1974). "Approximation algorithms for combinatorial problems." Journal of Computer and System Sciences **9**: 256-278.

Kanehisa, M. and P. Bork (2003). "Bioinformatics in the post-sequence era." Nat Genet **33 Suppl**: 305-10.

Kierczak, M., W. R. Rudnicki, et al. (2008). Construction of Rough Set-Based Classifiers for Predicting HIV Resistance to Nucleoside Reverse Transcriptase Inhibitors. Studies in Fuzziness and Soft Computing, Granular Computing: At the Junction of Rough Sets and Fuzzy Sets. Berlin / Heidelberg, Springer. **224:** 249-258.

Komorowski, J., Z. Pawlak, et al. (1999). Rough sets: A tutorial. Rough Fuzzy Hybridization: A New Trend in Decicion-Making. S. K. Pal and A. Skowron, Springer**:** 3-98.

Komorowski, J., A. Øhrn, et al. (2002). The ROSETTA Rough Set Software System. Handbook of Data Mining and Knowledge Discovery. W. Klösgen and J. Zytkow, Oxford University Press: 554-559.

Kontijevskis, A., J. E. Wikberg, et al. (2007). "Computational proteomics analysis of HIV-1 protease interactome." Proteins **68**(1): 305-12.

Lægreid, A., T. R. Hvidsten, et al. (2003). "Predicting gene ontology biological process from temporal gene expression patterns." Genome Res **13**(5): 965-79.

Mitchell, T. M. (1997). Machine Learning. New York, McGraw-Hill.

Nørsett, K. G., A. Lægreid, et al. (2004). "Gene expression based classification of gastric carcinoma." Cancer Lett **210**(2): 227-37.

Pawlak, Z. (1982). "Rough Sets." International Journal of Information and Computer Science **11**(5): 341-356.

Pawlak, Z. (1991). Rough sets: theoretical aspects of reasoning about data. Theory and decision library. Series D, System theory, knowledge engineering, and problem solving. Dordrecht ; Boston, Kluwer Academic Publishers: 229.

Russell, S. and P. Norvig (1995). Artificial Intelligence. New Jersey, Prentice-Hall.

Schapire, R. E. (1990). "The strength of weak learnability." Machine learning **5**: 197-227.

Shatkay, H. and R. Feldman (2003). "Mining the biomedical literature in the genomic era: an overview." J Comput Biol **10**(6): 821-55.

Skowron, A. and H. S. Nguyen (1999). Boolean reasoning scheme with some applications in data mining. Third European Symposium on Principles and Practice of Knowledge Discovery in Databases, Springer-Verlag.

Skowron, A. and C. Rauszer (1992). The discernibility matrices and functions in information systems. Intelligent Decision Support: Handbook of Applications and Advances in Rough Sets Theory. R. Slowinski, Kluwer Academic Publishers: 331-362.

Strömbergsson, H., A. Kryshtafovych, et al. (2006). "Generalized modeling of enzyme-ligand interactions using proteochemometrics and local protein substructures." Proteins **65**(3): 568-79.

Strömbergsson, H., P. Prusis, et al. (2006). "Rough set-based proteochemometrics modeling of G-protein-coupled receptor-ligand interactions." Proteins **63**(1): 24-34.

Theodoridis, S. and K. Koutroumbas (2003). Pattern recognition. Amsterdam ; Boston, Academic Press.

Wabnik, K., T. R. Hvidsten, et al. (2009). "Gene expression trends and protein features effectively complement each other in gene function prediction." Bioinformatics **25**(3): 322-30.

Wilczynski, B., T. R. Hvidsten, et al. (2006). "Using local gene expression similarities to discover regulatory binding site modules." BMC Bioinformatics **7**: 505.

Vinterbo, S. and A. Øhrn (2000). "Minimal approximate hitting sets and rule templates." International Journal of Approximate Reasoning **25**: 123-143.

Zhou, W., C. Zhou, et al. (2006). Feature Selection for Microarray Data Analysis Using Mutual Information and Rough Set Theory. IFIP International Federation for Information Processing, Artificial Intelligence Applications and Innovations. Boston, Springer. **204:** 492-499.

Øhrn, A. (1999). Discernibility and Rough Sets in Medicine: Tools and Applications. Trondheim, Norwegian University of Science and Technology.

Øhrn, A. (1999). ROSETTA Technical Reference Manual. Trondheim, Norwegian University of Science and Technology.

Øhrn, A. (2000). The ROSETTA C++ Library: Overview of Files and Classes. Trondheim, Norwegian University of Science and Technology.