UPTEC X 07 019
Examensarbete 20 p
Mars 2007

Analysis of biomarker data for multiple sclerosis in a workflow environment

Sara Grey

UPPSALA UNIVERSITET

Bioinformatics Engineering Programme

Uppsala University School of Engineering

UPTEC X 07 019	**Date of issue 2007-03**

Author

Sara Grey

Title (English)

Analys of biomarker data for multiple sclerosis in a workflow environment

Title (Swedish)

Abstract

In this project data from patients with multiple sclerosis was analysed for biomarkers using multivariate analysis methods. The analysis was conducted in a workflow environment, InforSense Knowledge Discovery Environment (KDE), to evaluate its performance in comparison to the computational programming language MATLAB.

Keywords

Multiple sclerosis, multivariate analysis, proteomics, biomarker, workflow environment

Supervisors

Hugh Salter and Kerstin Nilsson
AstraZeneca R&D

Scientific reviewer

Erik Bongcam-Rudloff
Linneaus Centre for Bioinformatics

Project name	Sponsors
Language **English**	Security
ISSN 1401-2138	Classification
Supplementary bibliographical information	Pages **47**

Biology Education Centre Biomedical Center Husargatan 3 Uppsala
Box 592 S-75124 Uppsala Tel +46 (0)18 4710000 Fax +46 (0)18 555217

Analysis of biomarker data for multiple sclerosis in a workflow environment

Sara Grey

Sammanfattning

Multipel skleros är en autoimmun nervsjukdom som det idag inte finns något botemedel för. I ett pågående projekt på AstraZeneca letar man efter sätt att diagnosticera sjukdomen, främst genom att identifiera biologiska kännetecken, så kallade *biomarkörer*, i proteinuttrycksdata. Med hjälp av sådana biomarkörer hoppas man kunna förenkla diagnosticeringen av sjukdomen och lättare följa effekten av behandlingar. Stora mängder data gör det nödvändigt att tillämpa multivariata analysmetoder för att finna samband mellan diagnos och proteinuttryck. Den här typen av analys är vanligt förekommande på AstraZeneca och det skulle därför vara önskvärt att kunna automatisera analyser. Arbetsflödesmiljön InforSense KDE erbjuder den möjligheten. Potentiella fördelar med KDE är att många olika analysverktyg integreras på samma plattform och att arbetsflöden kan återanvändas och modifieras för att passa aktuella önskemål.

Syftet med projektet är att utvärdera hur bra KDE är för att skapa en multivariat analys för att hitta prediktiva biomarkörer i multipel skleros-data i jämförelse med beräkningsspråket MATLAB.

Slutsatsen är att KDE trots det stort utbudet av funktioner och möjligheten att återanvända arbetsflöden är en alltför instabil och långsam plattform för den här typen av analyser. KDE kan med fördel användas för att integrera olika analysverktyg men bör inte användas för avancerade analysflöden som bygger på KDEs egna funktioner.

Civilingenjörsprogrammet Bioinformatik
Uppsala universitet mars 2007

Contents

1 Introduction

This degree thesis has been conducted at AstraZeneca R&D Södertälje as part of a larger multiple sclerosis project. Multiple sclerosis is a complex neuro-degenerative disease with many on-going simultaneous processes which makes it difficult to predict symptoms and to diagnose the disease. The multiple sclerosis project aims at finding biological characteristics, or *biomarkers*, that can function as indicators of the disease. Biological features are in this case protein expression levels obtained from two-dimensional electrophoresis gels. The vast amount of data produced by this technique requires advanced and powerful data analyses for identifying features of interest. Such data analyses are the multivariate analysis methods. With multivariate methods all variables are analysed simultaneously and underlying patterns can be identified. If data is labelled according to class, in this case diseased or healthy, prediction models can be created. Prediction models use known samples and their respective label to create mathematical models that classify unknown samples and identify important features. Such features can be used for diagnosing the disease. The objective is to find as few biological features as possible that can predict the outcome of the disease in order to save costs, time and minimising the discomfort for the patients.

This type of multivariate analysis can be used for building prediction models for many datasets of different character. AstraZeneca has several similar on-going projects where it would be useful to have some kind of general guidelines for conducting such analyses. A workflow environment, InforSense KDE, has been introduced in order to create analysis flows that can be reused infinitely or modified to suit the current analysis. KDE is a platform where data is handled primarily in the form of tables. Table operations are represented by nodes that are connected into a network. The nodes provide a structured graphical overview of the analysis flow. Many different tools can be integrated on the platform, such as the computing language MATLAB and the visualisation tool Spotfire. Once a workflow has been created it can be distributed as a general component that can be executed by many users repeatedly. These are advantages that make KDE a potentially useful environment.

Previously this multivariate analysis has been done using MATLAB. The advantages of MATLAB are the fast executions and the enormous possibilities to design scripts. Disadvantages are difficulties in visualising and comparing results and getting an overview of how scripts are related. It is also difficult to distribute MATLAB scripts to users that are not familiar with computing languages. The workflow environment KDE could provide solutions to these disadvantages, but the question is if the performance and results of a multivariate analysis in KDE would be equivalent to MATLAB.

1.1 Project objectives

The objectives of this thesis were to

- Perform a multivariate analysis for identifying predictive biomarkers in proteomic data from patients with multiple sclerosis
- Evaluate the performance of the workflow environment InforSense KDE on the above mentioned analysis and compare it to MATLAB
- Integrate MATLAB with the workflow environment InforSense KDE and evaluate the performance on the above mentioned analysis

2 Biological background

2.1 Multiple sclerosis

Multiple sclerosis (MS) is a chronic, autoimmune neuro-degenerative disease with a life expectancy of more than twenty years after discovery. It affects more women than men and is most common in the Northern hemisphere. It typically outbreaks in the early adulthood. Symptoms and prognosis are fairly unpredictable and depend on age, gender and environmental factors. Early symptoms are loss in sensation and vision, clumsiness and fatigue. This is caused by demyelination and degeneration of axons throughout the nervous system (Calabresi 2004, Bielekova & Martin 2004, Ibrahim & Gold 2005).

The disease can be categorised into three subgroups: (1) relapse remitting MS (RR) where the patient has outbreaks of symptoms that last from hours to weeks with no progression between relapses, (2) secondary progressive MS (SP) which differs from RR in that symptoms worsen between relapses until body functions gradually deteriorate and (3) primary progressive MS (PP) where symptoms increase continually (Noseworthy *et al.* 2000).

The heterogeneity of the disease makes it difficult to diagnose and to predict symptoms. There are many different stages of the disease and many parallel on-going processes affecting the total state (Bielekova & Martin 2004).

2.2 Biomarkers

Biomarkers are measurements of biological characteristics that can distinguish between different biological states, for example healthy or diseased. A biomarker can be a quantitative measurement from an analysis or a combination of measurements. The MS project looks for surrogate biomarkers, which are biomarkers that give information about clinical prognosis and the effects of treatment. To be useful, surrogate biomarkers should indicate the prognosis of a treatment faster than the time needed for following the progress in patients. They are desirable features in drug development as they may decrease time and cost (LaBaer 2005, Bielekova & Martin 2004).

For some complex diseases, such as MS, it is impossible to find a single biomarker. Many processes work simultaneously in MS making it difficult to screen all mechanisms at once. 95% of all biomarkers for MS target the inflammatory stages of the disease, while few target processes such as remyelination and demyelination that could correspond better to long-term disability (Bielekova & Martin 2004).

2.3 Proteomics

Proteomics is the study of the proteome, which is the total complement of proteins present in a biological system. The proteome is not constant and changes after cell type and time. A method for finding expressed proteins is two-dimensional polyacrylamide gel electrophoresis (2D PAGE). Large amounts of data can be generated by this high-throughput method because of the possibility to run multiple gels and the large number of protein spots on each gel (Chang *et al.* 2004). Proteins are separated according to isoelectric point (pI) and molecular weight (MW), see figure 1. These two measures are independent of each other so that protein spots become evenly distributed over the gel. 2D PAGE can detect differences in proteins down to one charge and can estimate the number of proteins present in a biological system (Kenrick & Margolis 1970, O'Farrell, 1975).

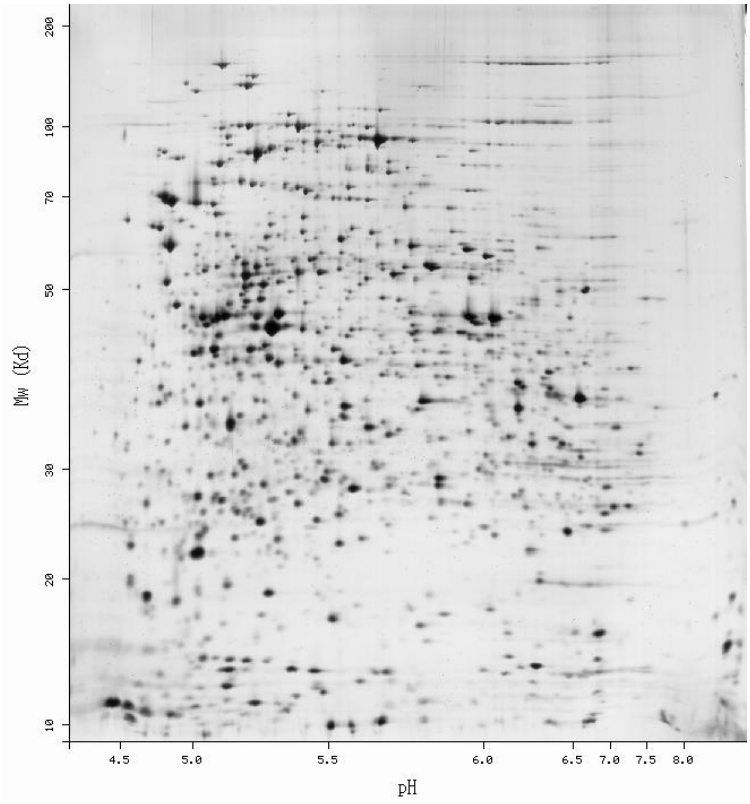


Figure 1: A two-dimensional gel where proteins are separated according to isoelectric point (pI) and molecular weight (MW). Picture from SWISS-2DPAGE.

Data is extracted from gels using spot identifying software such as PDQuest and Progenesis. Transferring spots on a stained gel to computable data includes the following three steps: (i) protein spot detection, (ii) spot quantitation, (iii) gel-to-gel matching of spot patterns (Marengo *et al.* 2005, Rosengren *et al.* 2003).

The stained gel is scanned to assign each pixel with coordinates for x and y and a signal intensity value. Limits for smallest, largest and faintest acceptable spot are set to identify the pixels that will be considered. The result is an image consisting of marked spots with densities modelled as ideal Gaussian distributions. Spots can be matched across gels, by comparing with a reference gel, a so called *master image*. Statistical analysis is performed to establish if there are differentially expressed proteins on the matched gels (Marengo *et al.* 2005, Rosengren *et al.* 2003). When processing of the gels is completed spot information is obtained as data. Spots are represented as data matrices where protein spot intensities form rows and samples form columns (Chang *et al.* 2004).

Missing values in the data can be caused by biological or experimental factors. It is important to distinguish between these two and examine if there are systematic errors and which missing values they correspond to. Missing values can be excluded from further analysis, but may hold important information. For example a decrease in protein level may appear as a missing value (Chang *et al.* 2004).

3 Material and Methods

3.1 Multivariate modelling

In multivariate modelling all variables are evaluated simultaneously. There are several methods available, for example random forests, neural networks, support vector machines and projection methods such as principal component analysis and partial least squares (Eriksson *et al.* 2001, Hastie *et al.* 2001).

A model is a simplified approximation of reality and therefore computable. A model can never fully explain reality so that a part of the data will not be fitted into the model. This is described by the relationship:

Data = Model (factors, parameters) + Noise

A well-tuned empirical model will approach the correct model so that the noise will be negligible (Eriksson *et al.* 2001).

3.1.1 Predictive modelling

With large amounts of data, such as those generated from microarrays or two-dimensional electrophoresis gels, it is often of interest to classify data into groups. Samples can be labelled according to a certain outcome, for example diseased or healthy. Groups can also be organised according to similarity (Bø & Jonassen 2002). The aim with predictive modelling is to build a mathematical model that can correctly predict the outcome of new samples. A subset of the data is used for training the model and the remaining samples are applied to the model to get an estimate of its predictive performance (Hastie *et al.* 2001).

3.1.2 Projection methods

One group of multivariate modelling methods are the projection methods or dimension reducing methods. Projection methods assume that there are underlying linear relationships that can describe the data, so called *latent variables*. The measured variables are modelled as linear combinations of a set of latent variables that follow two basic ideas: (1) The measurements are by definition the sum of the underlying latent variables. (2) The data are assumed to be measurements of a set of similar observations.

The idea is to let every observation of the data be represented by a point in multidimensional space. The points can then be projected onto a surface of fewer dimension, see figure 2. This way the data points are described by fewer variables. The surface is rotated in space to fit the data points in the best possible way, in principal component analysis (PCA) to maximise

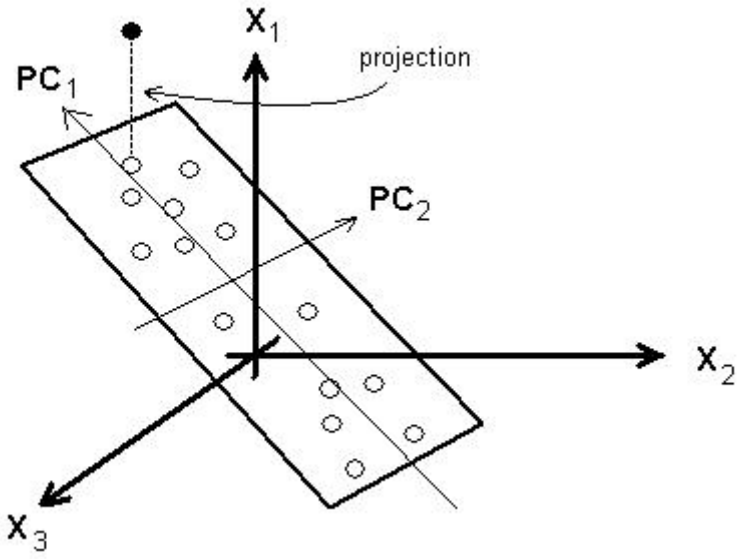


Figure 2: Data points described by the axes X_1, X_2 and X_3 are projected onto a plane of lesser dimension. The plane is rotated to fit the points in the best possible way. The axes of the new plane are the principal components PC1 and PC2. Figure modified after Eriksson *et al.* (2001).

the explained variance and in partial least squares (PLS) to maximise the covariance between two matrices X and Y (Eriksson *et al.* 2001).

3.1.2.1 Principal Component Analysis (PCA) is a multivariate projection method for finding systematic variations in a matrix with data that is assumed to be correlated. In PCA the projections of the data points onto a surface are called the *scores*. The *loadings* are the weights combining the original variables to form the scores. The loadings represent the orientation of the surface in space where variance is maximised and hold information on how important every variable is to the model. The axes that span the surface are called the *principal components*, see figure 2.

PCA can be used for classifying objects by estimating the distance of new samples to the model. Samples that are very distant to the model are referred to as outliers. Data is on the form N observations and K variables. The relation is described by

$$X = TP' + E$$

where T is the score matrix, P are the loadings and E is the residual. The scores and loadings are orthogonal linear combinations of the variables and

observations. A standard procedure for finding the principal components is the NIPALS (Non-linear Iterative PLS) algorithm. It works on data that is symmetrically distributed in the following way:

1. Get a starting vector t
2. Calculate the loadings, p

$$p = \frac{X't}{t't}$$

(the projection of X onto t)

Norm p to $\|p\| = 1$

3. Calculate the scores, t

$$t = \frac{X'p}{p'p}$$

(the projection of X onto p)

4. Test if the change in t has reached convergence, i.e. if

$$\frac{\|t_{old} - t_{new}\|}{\|t_{new}\|} < \varepsilon$$

($\varepsilon = 10^{-6}$ or smaller)

If not, return to 2

5. Deflate X by removing the present component and return to 1 to compute the next component

Explanation from Wold, Esbensen & Geladi (1987).

3.1.2.2 Partial Least Squares (PLS) Data with strongly collinear, noisy and numerous X-variables can be analysed with PLS-regression. PLS is based on the same idea as PCA. The difference is that PLS relates *two* data matrices X and Y by a linear multivariate model so that the dimension is reduced for both X and Y, the covariance is maximised and the structure of X and Y is modelled. Y are response variables to the predictor variables X (Wold, Sjöström & Eriksson 2001). The Y-matrix could contain the outcome of a disease and the X-matrix protein expression data. Proteomics data is well analysed with PLS because of the structure of the data, which is few samples and many correlated variables.

The X and Y-variables are assumed to be modelled by the same latent variables and are assumed to be dependent. The precision of the model parameters increase with the amount of relevant variables and observations.

Data is on the form N samples with K X-variables and M Y-variables. The X-scores, T, are orthogonal, estimated as linear combinations of the original variables, X, with the coefficients W.

$$T = XW$$

$$X = TP' + E$$

where P' are the loadings and E the X-residuals, E is small.

$$Y = TC' + F$$

where C' are the Y-weights and F the Y-residuals. A multiple regression model can be created with the following relation

$$Y = XWC' + F = XB + F$$

where the PLS-regression coefficients are

$$B = WC$$

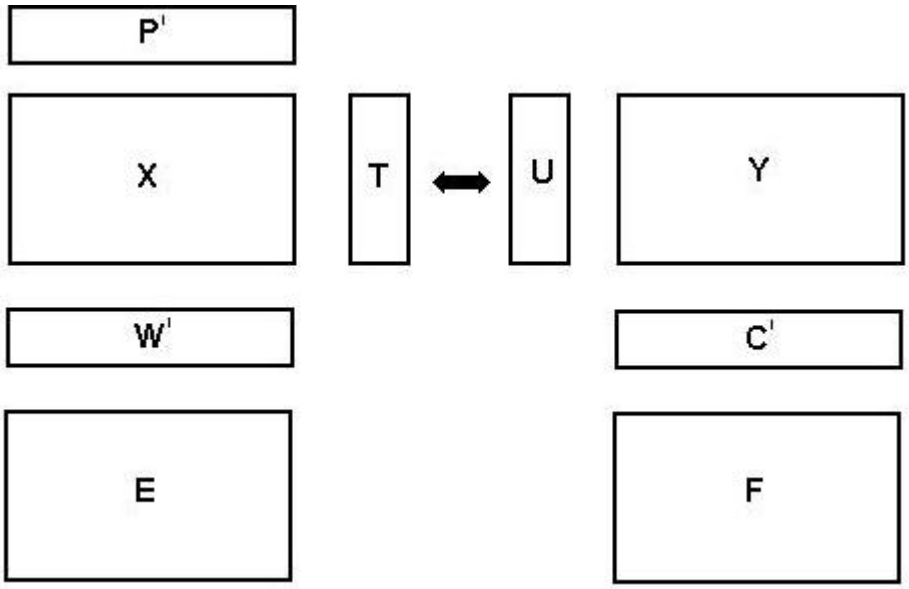


Figure 3: Matrix representation of data tables in PLS. Modified after Eriksson *et al.* (2001).

Depending on the type of data there are different methods of scaling and transformation that apply. Appropriate scaling for the X-matrix is usually

auto-scaling, i.e. mean-centring the data and scaling to unit-variance by dividing each variable by its standard deviation. Auto-scaling gives the variables the same weight since variable importance is related to variance in PLS. Variables of particular interest can be given other weights in order to stand out. For numerical stability it is good to mean-centre both the X and Y-block by subtracting the averages from all variables (Wold, Sjöström & Eriksson 2001).

The two most used algorithms for PLS are NIPALS and SIMPLS (Straight-forward implementation of a Statistically Inspired Modification of the PLS method) (deJong 1993). The NIPALS algorithm works on transformed, scaled and centred X and Y-data. The NIPALS algorithm for PLS is equivalent to 1,5 iterations of the NIPALS algorithm for PCA:

1. Get a start vector u, usually one of the Y-columns. If Y is univariate,

$$u = y$$

2. Calculate the X-weights, w

$$w = \frac{X'u}{u'u}$$

(the projection of X onto u)

Norm w to $\|w\| = 1$

3. Calculate the X-scores, t

$$t = Xw$$

4. Calculate the Y-weights, c

$$c = \frac{Y't}{t't}$$

(the projection of Y onto t)

5. Update the Y-scores, u

$$u = \frac{Yc}{c'c}$$

6. Test if the change in t has reached convergence, i.e. if

$$\frac{\|t_{old} - t_{new}\|}{\|t_{new}\|} < \varepsilon$$

($\varepsilon = 10^{-6}$ or smaller)

7. Deflate X and Y by removing the present component (not necessary for Y)

$$p = \frac{X't}{t't}$$

$$X = Xtp'$$

$$Y = Ytc'$$

8. Return to 1 to compute next component until all significant components have been found

Explanation from Wold, Sjöström & Eriksson (2001).

3.1.3 Cross validation

The basic idea with cross validation is to divide data into K equally sized sets. Each set is excluded from model construction once and used as new data for estimating the predictive ability of the model (Ambroise & McLachland 2002, Hastie *et al.* 2001).

There are two reasons to perform cross validation on PLS models. The first is to validate the predictive power of a model by testing it with data that have not been included in the model. The ideal case is to test the final model with external data, but such data is not always available. Samples are usually few and are needed for building the model. Cross validation can provide an option to unknown data, as data that are excluded from building the model can serve as new data (Eriksson *et al.* 2001, Wold, Sjöström & Eriksson 2001). However, having a small sample size can cause bias in performance estimation, because cross validated data follow three dependencies: (1) Training and test sets are dependent, since the selection of samples for training set consequently puts all the unselected samples in the test set. (2) Training sets are inter-dependent, since information gained from the samples of a model holds information on samples that could potentially be selected for the next model. (3) Test sets are inter-dependent, since samples that have contributed to previous models give information about samples that could be selected for next test set (Wickenberg-Bolin *et al.* 2006).

Further, there is a risk of getting too optimistic error rates when selecting the best parameter settings from a large number of alternative settings using cross validation. To avoid bias an external set of samples is needed for validating the prediction estimate of the best model (Soeria-Atmadja *et al.* 2005).

The second is to determine the best number of PLS-components for the model. It is important not to over-fit a model by adding PLS-components

until the data is described perfectly. The predictive ability may then decrease since it is a measure of how well the model can predict the outcome of new data. The contribution of each PLS-component can be estimated by cross validation. The excluded group serves as validation set, to estimate differences between the actual and predicted Y-values (Wold, Sjöström & Eriksson 2001, Eriksson *et al.* 2001).

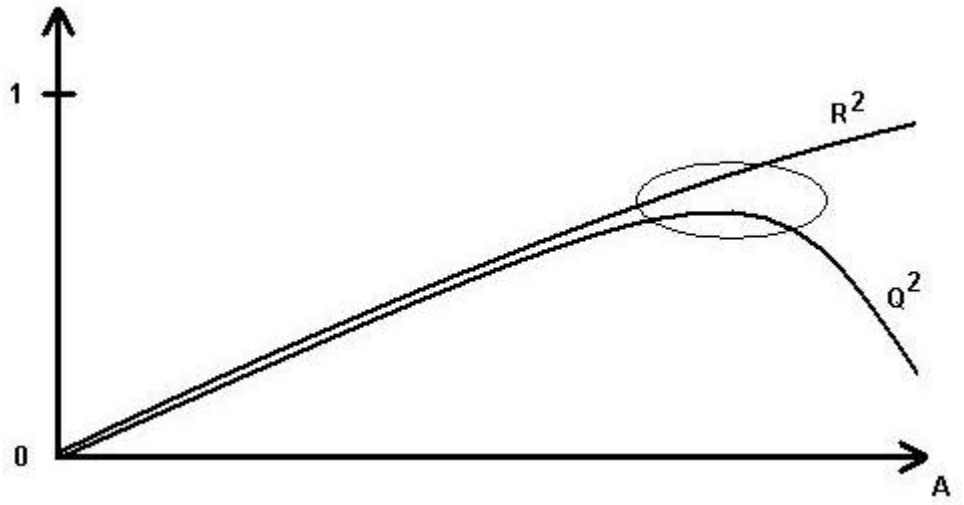


Figure 4: The relationship between explained variance, R^2, and the predicted variance, Q^2, where A is the number of PLS-components. There is a trade-off between the model's fit, described by R^2, and the model's predictive ability, described by Q^2. R^2 is proportional to the number of PLS-components in the model, while Q^2 reaches a maximum. Modified after Eriksson *et al.* (2001).

There is a trade-off between a model's fit and predictive ability. Figure 4 shows the relationship between R^2, the explained variation of a model, and Q^2, the predicted variation (from cross validation). R^2 increases in proportion to the number of PLS-components and the complexity of the model while Q^2 reaches a plateau. The point where Q^2 has its maximum corresponds to the number of components that gives the model the best predictive power (Eriksson *et al.* 2001, Hastie *et al.* 2001).

3.1.3.1 "Three-fold bootstrap cross validation" The dataset is randomly divided into three equally sized groups. Each group is excluded once from the model and used as test set. This results in a good balance in size between training and test set. Then the dataset is randomly divided into three new groups and the same procedure followed. The process is repeated N times.

3.1.4 Variable selection

All variables in a large dataset may not be relevant for a study. To identify the best subset of variables is referred to as the feature subset selection problem. Finding subsets of features is relevant because costs can be reduced if fewer features need to be measured, time can be saved and perhaps less complicated technology needs to be used (Bø & Jonassen 2002). Moreover, if a dataset has many variables with low weights variable selection may improve the predictive ability (Freyhult *et al.* 2005).

One approach is to consider each variable's individual ability to assign the correct outcome to a sample. The K features with highest score are then selected as the best feature subset. Bø & Jonassen (2002) state that it can be inappropriate to rank features individually because a pair-wise ranking can find informative variables that would not be selected in an individual feature search. Another approach is to find correlated variables and exclude those that show high similarity with others. This could be advantageous for proteomics data which has a lot of correlated variables. A feature subset can also be obtained by reducing the dimensionality of the model by selecting the N first principal components.

Test set samples should not be part of the variable selection procedure. Ambroise *et al.* (2002) has shown that there will be selection bias if samples from a test set that are not part of the model are included while selecting the best features. To avoid such bias the same feature selection method must be applied to all training sets.

3.1.4.1 Variable importance for the projection (VIP) The parameters of a PLS model hold information on the contribution of the individual variables to the model. Variables that are important for modelling Y have large PLS-regression coefficients while variables that are important for modelling X have large loadings. The variable importance for the projection score estimates how well a variable contributes to both X and Y. The VIP score is a sum of squares of the PLS-weights, weighted with the Y-variance for each PLS-component (Wold, Sjöström & Eriksson 2001).

$$VIP_k = \sqrt{\sum_{a=1}^{A} \left(W_{ak}^2 SSY_a \frac{K}{SSY_{tot}} \right)}$$

where k is the variable number from 1 to K, a is the component number from 1 to A, W is the PLS-weights, SSY is the explained variance (%) and SSY_{tot} is the cumulative explained variance (%) (Eriksson *et al.* 2001).

3.1.5 Prediction performance measures

To evaluate the predictive ability of a model it is necessary to have some parameters to measure this by. A predictive model tries to distinguish between groups. The simplest case handles two classes, they can be called true and false. Parameters for assessing if a prediction is correct or not are true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), see table 1.

Correct class	Predicted class	
	True	False
True	TP	FN
False	FP	TN

Table 1: Prediction performance parameters.

A standard measure of predictive power is the total success rate, i.e.

$$\text{Predictive ability} = \frac{TP + TN}{TP + TN + FP + FN}$$

This can be rewritten as

$$\text{Predictive ability} = \frac{\text{Number of correctly predicted samples}}{\text{Total number of samples}}$$

Measures of how well each class is predicted are sensitivity and specificity. They can also be referred to as *recall*:

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Sensitivity and specificity are parameters that remain the same irrespective of the population tested, given that they have been estimated thoroughly.

Measures for the accuracy that samples that have been predicted as one class actually belong to that class are positive predictive value (PPV) and negative predictive value (NPV). They can also be referred to as *precision*:

$$\text{PPV} = \frac{TP}{TP + FP}$$

$$\text{NPV} = \frac{TN}{TN + FN}$$

PPV and NPV are of interest when a certain population is to be examined. They give the probability that a test result is correct in relation to the test population size (LaBaer 2005). For a test to be accepted as a diagnostic test the precision parameters are examined, since they should correspond to an accurately distributed population.

3.1.6 Cutoff

A PLS model gives predicted Y-values for the samples. These are continuous values that need to be categorised into one of the classes of the model. To decide whether a predicted Y-value belongs to for example class A or class B a cutoff value is needed. The cutoff will label the samples according to:

Predicted value $<$ cutoff $\Rightarrow$ Class A
Predicted value $\geq$ cutoff $\Rightarrow$ Class B

Depending on the aim with the prediction model, different cutoffs may be applicable. It is possible to influence how well one particular class is predicted by adjusting the cutoff value. If the aim is to predict all classes equally well, both the sensitivity and specificity parameters should be maximised. Then the maximum number of true positives and true negatives will be obtained at the same time. If one class is more important the cutoff can be set to optimise the number of correct predictions for this class.

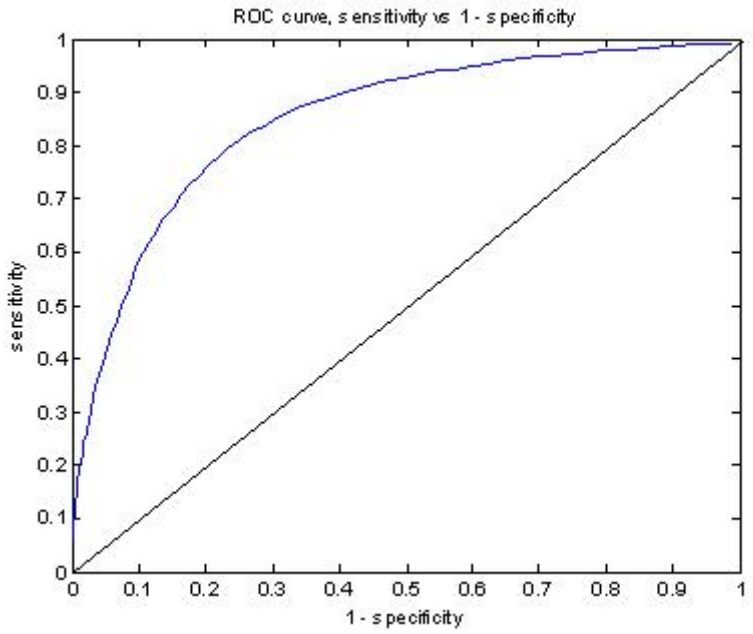


Figure 5: A receiver operating characteristic (ROC) curve. The straight line corresponds to sensitivity and specificity measures obtained with pure chance. The curved line approaches the top left corner, which is where sensitivity and specificity are the greatest possible at the same time.

There are several ways to identify the desired cutoff value. The sensitivity and specificity values for the model need to be computed for a range of cutoff

values. They can be visualised in a receiver operating characteristic (ROC) curve where sensitivity is plotted as a function of (1 - specificity), see figure 5. The point in the top left corner corresponds to the highest combined sensitivity and specificity values, in this case the cutoff where both classes are predicted equally well (Vining and Gladish 1992). An ROC curve can also be plotted for other parameters than the cutoff.

Another way to visualise the relationship between sensitivity, specificity and the predictive power is to plot all three parameters as functions of the cutoff values. The point where the curves intersect is where sensitivity and specificity both are the largest possible. See figure 6.

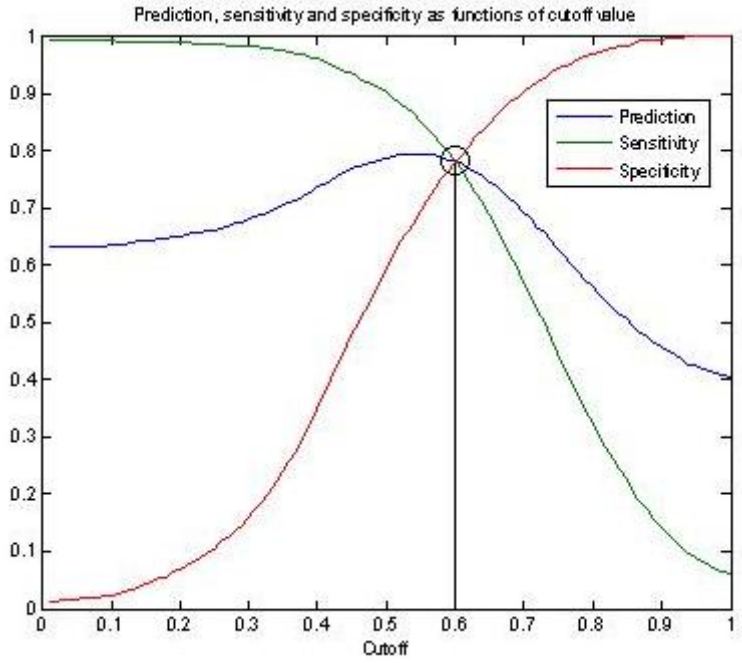


Figure 6: Alternative visualisation option to identify cutoff value. The point where the curves intersect is where sensitivity and specificity both are the largest possible.

3.2 Inforsense KDE

The KDE system is based on J2EE (Java 2 Platform, Enterprise Edition) technology, which is a free software from Sun Microsystems. JBoss is used for the application server and Apache Tomcat is the web container (Saied 2006, personal communication). Both are open source. The standard installation is *client-server*, with a single server to which many clients can connect. The minimum requirements for a client machine are 1 GB RAM and 250

MB free disk space. The minimum requirements for a server machine with up to five clients are 2 GB RAM and 10 GB free disk space per user. The server software alone requires 250 MB. KDE can be launched using a webstart Discovery Portal, or via the InforSense Client (InforSense KDE 3.1 Installation and Setup Guide). See Appendix B for a screenshot of the InforSense KDE platform.

In KDE workflows are built from components, or nodes, that carry out operations on data. See figure 7. Data are primarily handled in the form of tables, passed on in a network of connected executable nodes. Any node in the network can be executed and its result obtained. All preceding nodes will be executed in prior.

Figure 7: An example of a workflow in KDE. The node with caption "iris" is a table containing the columns sepalLength and sepalWidth. The node with caption "Derive" is a Derive node which creates new columns from KDE expressions. In this case a new column named sepalArea is derived from the product of the two columns. When the Derive node is executed a new table is obtained - the node with caption "sepal area", which contains the columns from "iris" and the derived column sepalArea.

A node consists of three parts: input, action and output. Input can be either a file pointer or the output of another node. The input is modified by the action and passed on through the output port. The output can be viewed using visualisation tools in KDE such as Table Editor, Spotfire and Text Viewer. A table is the most common data format but there are other formats such as *model* (for regressions and statistics) and *database*.

There are nodes for all kinds of operations. There are nodes for pre-processing data such as scaling, deleting and joining tables. There are nodes for statistical analysis, database operations, text mining and bioinformatics tools. There are nodes for iterating over workflows using for and while loops. There are nodes for scripts using Perl, Groovy or MATLAB.

The nodes are Java classes and can be freely developed through a software development kit (SDK). The nodes belong to the InforSense package *com.kensingtonspace.sdk.node*. A number of structures, classes and methods are available for node development, mainly based on java.sql.ResultSet. The node class has three parts: NodeDescriptor, Prepare and Process. The

NodeDescriptor provides the characteristics of the node such as the number of input ports, parameters, output ports and their formats. The Prepare section handles the metadata of the node. This section is executed in real-time, which gives the user information on whether formats are correct and what the output of the node will be. The Process part is where the actual computation is done. Anything that can be performed in Java can be inserted here. The metadata from Prepare is used as a template for the ResultSet of Process (InforSense KDE 3.1 Development Guide).

An entire workflow can be turned into a complete analysis package. This means that a developer can create analysis flows that can be used by other users through the web interface. The developer can decide which parameters the user can change and the user needs no advanced data experience to run the analysis.

3.3 Dataset

Proteomic data comes from multiple sclerosis patients diagnosed at the Karolinska Hospital. Protein expression profiling with two-dimensional gel electrophoresis was conducted at AstraZeneca by Bo Franzén and Jan Ottervald. The data was adjusted for batch variation and log-transformed by Kerstin Nilsson.

The dataset has 1499 variables and 98 samples. 36 samples are from a control group diagnosed with "other neurological disease" and 62 samples are from patients diagnosed with MS.

3.4 Software/Hardware

Software

- InforSense KDE 3.1, InforSense Limited
- Spotfire DecisionSite 8.1, Spotfire AB
- MATLAB 7.0, MathWorks, Inc.
- PLS Toolbox 3.5, Eigenvector Research, Inc.
- Eclipse 3.1, Eclipse Foundation, Inc.
- Java Runtime Environment 5.0, Sun Microsystems, Inc.

Hardware

- PC with 2 GB RAM. InforSense KDE local server and client
- PC with 1 GB RAM. InforSense KDE client
- Linux Red Hat with 4 GB RAM. InforSense KDE global server

3.5 Experimental work

3.5.1 Data analysis

3.5.1.1 Pre-processing The data was labelled according to diagnosis with class 0 for MS and class 1 for the control group. The class labels formed the Y-variable. Together with the X-block of 1499 variables the total data block had 1500 variables and a key for identifying the samples. PCA was conducted to identify potential outliers.

The data in each trainingset was pre-processed before analysis, by auto-scaling X and mean-centring Y.

3.5.1.2 Prediction model and cross validation The data was modelled using PLS-regression with the aim to find an optimal model with regard to number of PLS-components, number of variables and cutoff value. The best number of PLS-components would be the number corresponding to the model with best predictive ability. The best number of variables would be the fewest possible without losing predictive power, as it is advantageous to reduce cost and time by analysing few features during experiments. The optimal cutoff value would be the one that equally well predicts diseased and control samples.

Models for 15 rounds of variable selection using the VIP score were computed. In each round the 100 variables with lowest VIP score were excluded from further analysis. This procedure was repeated for 1 to 10 PLS-components. The predictive power of each model was estimated for cutoff values between 0 and 1, with 0.01 steps.

The prediction model was computed for a range of random datasets, through a three-fold bootstrap cross validation procedure (see section 3.1.3.1). The three-fold partition was made so that each of the three groups received one third of the samples from the diseased class and one third from the control class.

3.5.1.3 Parameters For each model prediction parameters described in section 3.1.5 were calculated.

3.5.2 KDE

Workflows were built in KDE using already existing KDE nodes. Some functions were not supported so a few nodes were developed in Java, see table 2 for a complete list. See figure 17 in Appendix C for an overview of the main workflow.

3.5.2.1 Pre-processing In KDE scaling can be done in many ways. There are script nodes where practically any expression can be stated, but

Node	Input	Action	Output
ScaleAll	Table with X and Y-blocks	Auto-scales or mean-centres selected columns	Scaled table
ExtractModel	PLS model	Extracts variance from model	Variance as a table
VIP	Weights from a PLS node and variance from ExtractModel	Computes VIP score for all variables	VIP score for all variables
DeleteColumns	Column with column names to be deleted and a table	Deletes columns from a table	Table
PLSPrediction	Table with Y-predictions and table with correct Y-values	Calculates prediction parameters	Table with prediction parameters
AddColumns	Parameters and Table	Appends parameters as constant columns to table	Table with constant columns

Table 2: Nodes developed in Java to complete the workflow analysis.

as they operate on one column at a time they are not suitable for data with as many variables as the dataset in this project. There are also normalisation nodes with a selection of pre-defined types of scaling. None of these nodes were applicable for this analysis, so a node named ScaleAll was written in Java.

The ScaleAll node takes a table with X-block and Y-block as input. It selects which columns that belong to X and Y and which type of scaling to apply for each set - either auto-scaling or mean-centring. As described in section PLS auto-scaling was selected for X and mean-centring for Y.

3.5.2.2 Prediction model and cross validation The data was randomly split into three sets using a Partition node. This node uses a random seed represented by an integer. The seed must be changed in order to

create a new random sampling. By using the loop value of a For node which changes during iteration, different partitions could be obtained. Before the partition the samples from class 0 and class 1 were separated. Then each class was partitioned into three groups. Each third was joined to a third from the opposite class. To identify the sets each group was given a tag, like "Test1", etc. Then a Filter node was used to select the appropriate sets for training set and test set as input for the PLS modelling.

A PLS model was built from a training set using the PLS node. The output of the PLS node is a table containing the weights and a model with some parameters, such as the variance and cumulative variance. As there was no possibility to use the parameters from the model, a node was developed to extract the parameters from the node. This node was named ExtractModel. It takes a PLS model as input and outputs the variance of the model.

The weights and the variance were used for calculating the VIP score of the variables in the model, see section 3.1.4.1. There was no node for performing this, so a VIP node was developed. The VIP node takes the weights and variance of a PLS model as input and outputs a column with the VIP score for every variable together with the key of the variable. The keys of the N variables with lowest VIP score were identified in order to delete them from the X-block.

The Delete node in KDE only supports deleting columns manually, or by using a KDE expression. Therefore it could not be used for deleting the variables with lowest VIP score, since these variables were defined in a column. A Join node could not be used for this either as it works on rows instead of columns. The most timesaving alternative was to create a node, DeleteColumns, which deletes columns from a table using a column containing the variable names to be deleted. The new table was written to a file to be used as input data for the next round of variable selection.

The PLS model was validated with a test set using the PLSApply node. This node takes a PLS model and a table with test data as input. The output is the Y-predictions for the test samples. The Y-predictions were written to a database to be used for calculating parameters for prediction performance, together with the correct Y-values for the test set and some parameter settings identifying the model (the number of components and variables).

In order to perform the 15 variable selections and the models with 1 to 10 PLS-components, iterations were done on the workflow using several For nodes.

The For node takes two tables as input: one table with the data to be manipulated and one table with a loop variable where each row corresponds to one iteration. The node has a script parameter called "Do this before each loop". Here, parameters may be set and changed according to loop values or other parameters. The node has a parameter with a pointer to an "inner"

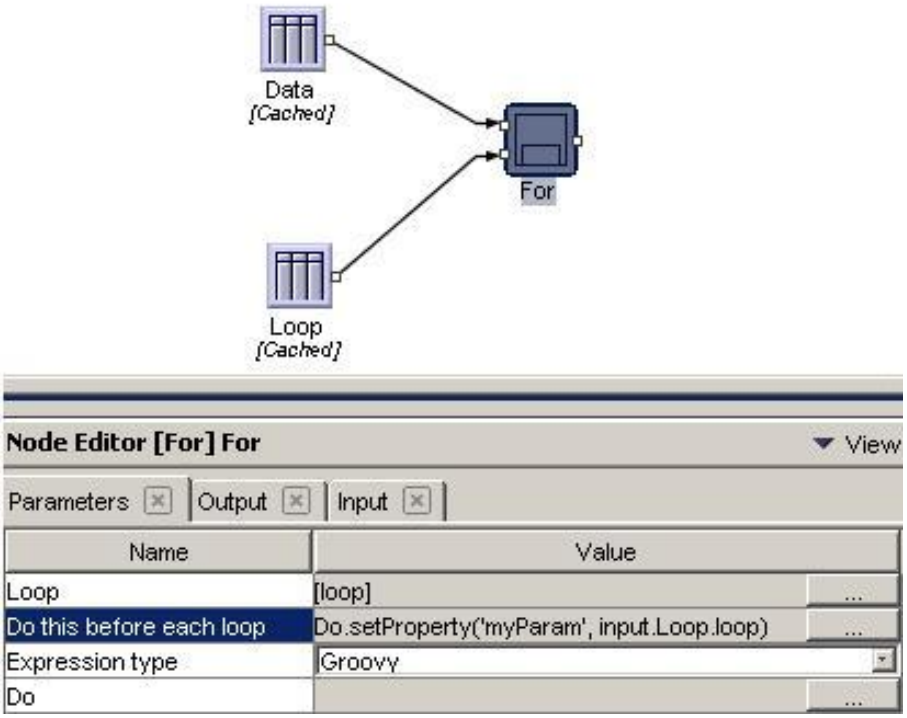


Figure 8: Workflow in KDE with a For node. The For node takes two tables as input, a data table and a loop table. Parameters can be defined or updated during iteration in the "Do this before each loop"-section, either using Groovy or the KDE language. The parameter "Do" points to a new workflow, which is where the data processing takes place.

workflow where the actual data processing takes place. The table that is input to the For node will be input to the inner workflow. The outputs from each iteration are joined through union with the previous results.

The For node always inputs the same data table to the inner workflow. This was a problem for variable selection since columns are removed during iteration. The workaround was to delete the input table and import the correct data from Userspace. The table resulting from each variable selection was subsequently written to Userspace to be imported during next iteration.

There is a While node in KDE which takes the output from a previous iteration as input for the next, which is what the variable selection procedure requires. Unfortunately the While node has not as extensive support for setting parameters by scripting as the For node and could therefore not be used.

In total, four For nodes were used. The first node handles the variable

selection. It contains the workflow where most of the data processing is handled. This node is iterated 15 times. It is incorporated into another For node, which handles the number of PLS-components to be used for the model. This node iterates 10 times and is incorporated into another For node. The third For node handles the selection of the training and test set to be used for model building and validation. It iterates three times, since data is partitioned into three sets and each set is used as test set once. In each round it selects one set for test set and the remaining two sets for training set. The third For node is incorporated into the last For node, which creates random partitions of data. This node should be iterated at least 100 times. However one iteration of this node takes approximately 100 minutes.

3.5.2.3 Parameters The saved Y predictions were used for estimating the average predictive ability of each model parameter setting. There was no node available for calculating such parameters so the node PLSPrediction was made. It takes the predicted Y and the correct Y as input and has a parameter for cutoff value. It computes the total predictive performance, sensitivity, specificity, PPV and NPV.

A workflow was made that iterates over all model parameter settings (number of components and variables). It estimates the average predictive ability for each parameter setting for one cutoff value. A workflow was also made that calculates prediction parameters for a range of cutoffs in a selected interval with a selected step size.

3.5.3 MATLAB

MATLAB is a programming language for creating mathematical functions and performing numeric computations.

Some functions from PLS Toolbox were used for the multivariate analysis. Additional functions that follow the procedure described under section 3.5.1 were written in MATLAB.

3.5.4 KDE using MATLAB

There is support in KDE for integrating external software such as Spotfire and MATLAB. There are several nodes available for MATLAB functions. There is a node for importing data from a .mat-file (MatViewer), a node that allows for direct input of MATLAB scripts (Input Script) and a node for using external MATLAB functions in .m-file format (Generic Matlab). The Generic Matlab node was used for executing the same scripts that were made for the MATLAB analysis.

4 Results and Discussion

4.1 Data analysis

4.1.1 Pre-processing

Pre-processing in MATLAB and KDE with mean-centred Y and auto-scaled X had the exact same result. This was verified with test data. PCA did not identify any outliers.

4.1.2 Prediction model and cross validation

The PLS models were not the same in KDE and MATLAB because of a bug in the PLS node: the weights from the PLS node differ from the weights calculated in MATLAB. The weights from MATLAB were verified with SIMCA, which is a software product for performing multivariate analyses. KDE uses the NIPALS algorithm, which InforSense claims to have verified with SIMCA.

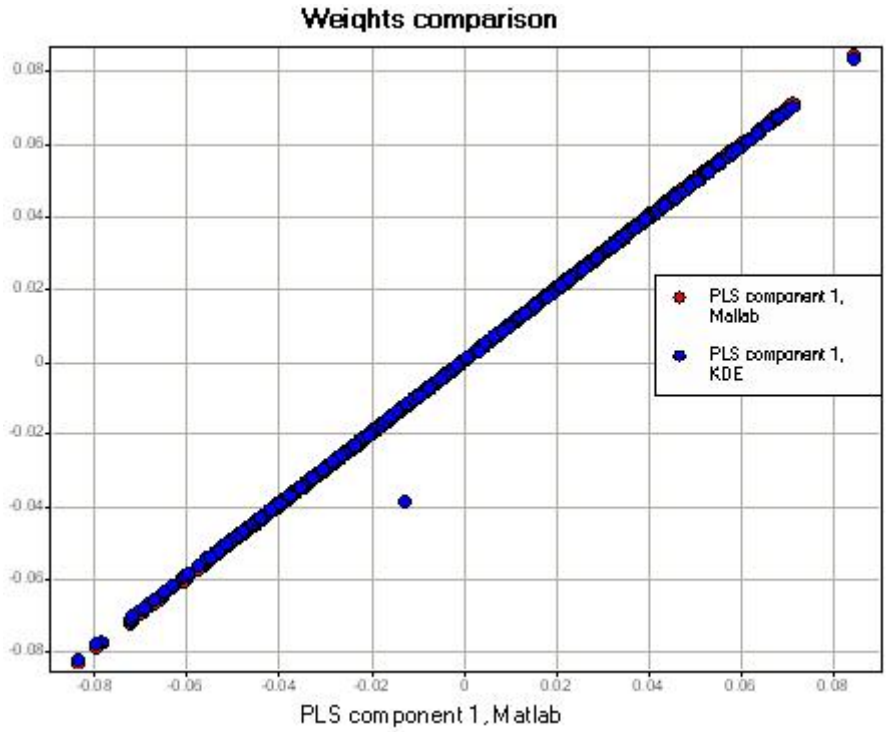


Figure 9: The PLS-weights for the first PLS-component, computed in KDE and MATLAB respectively, plotted against the first PLS-component in MATLAB. The weights are almost identical except for one sample.

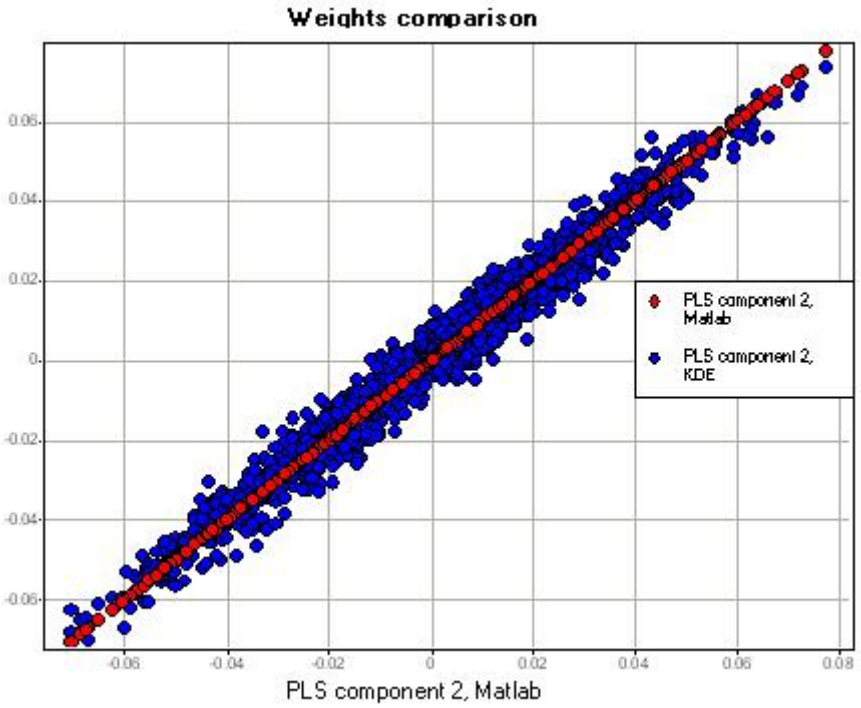


Figure 10: The PLS-weights for the second PLS-component, computed in KDE and MATLAB respectively, plotted against the second PLS-component in MATLAB. The weights differ, but are correlated.

The weights are almost equivalent for the first PLS-component, see figure 9. For all other components the weights are correlated but not equal, see figure 10 for the second PLS-component.

The VIP score was exactly the same using the VIP node in KDE and a VIP function in MATLAB. However, since the PLS-weights differed the VIP scores consequently were a bit different. Despite this, the same variables were selected during the variable selection procedure. The variables have the same correlation between themselves, even though the weights have greater values in MATLAB. This was verified with test data.

Both MATLAB and KDE identified 0,60 as the best cutoff value, see figures 11 and 12. The reason that the cutoff is higher than 0,5, which could be expected in a well-balanced model, is that there are more samples from class 0 than class 1.

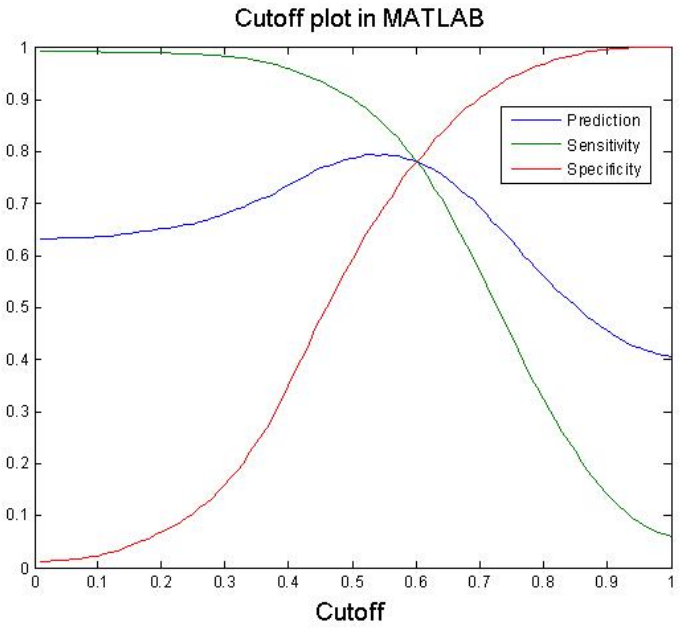


Figure 11: The optimal cutoff value is 0,60 according to MATLAB.

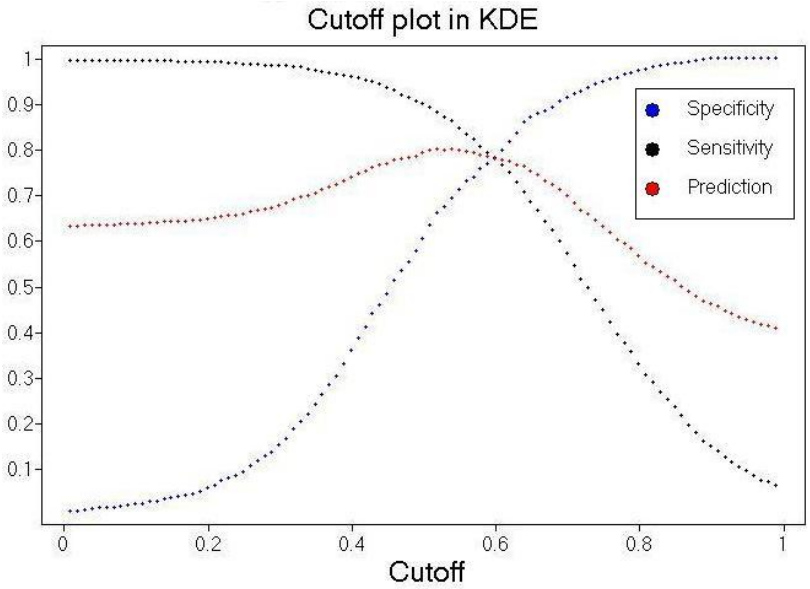


Figure 12: The optimal cutoff value is 0,60 according to the analysis in KDE.

PLS component	Variance (%) KDE	MATLAB	Cumulative variance (%) KDE	MATLAB
1	82,87	82,82	82,87	82,82
2	14,0	14,01	96,87	96,83
3	2,46	2,5	99,33	99,33
4	0,53	0,53	99,86	99,86
5	0,12	0,12	99,98	99,98

Table 3: Y-variance computed for models with one to five PLS-components. The variance is almost identical in KDE and MATLAB. The last two components add almost no further information and since the total variance approaches 100% it is not necessary to add more components.

Random partition of cross validation sets was done in MATLAB using the function *randperm(n)*, which performs a random permutation of the integers from 1 to n. In KDE a random seed was used in the Partition node. The random seed is generated by the *java.util.Random* class. The seed depends on the size of the table, so using the same seed for two tables of different size does not mean that the same row index will be selected for both.

The number of cross validation rounds differed in MATLAB and KDE. A run in MATLAB could without effort consist of 200 random partitions. In KDE the maximum number reached was 95. The execution could not be repeated several times in KDE as the analysis of one random partition takes approximately 100 minutes, compared to 34 seconds in MATLAB. The consequence was that the total number of PLS-components was decreased to five, instead of ten, to reduce execution time. First it was validated that PLS-components six to ten did not contribute any further to the model, see table 3.

4.1.3 Parameters

The PLS prediction node in KDE returns the same parameter values as the PLS functions in MATLAB. Test data was used to verify this.

Models with two PLS-components have the highest overall predictive performance, see figures 13 and 14. KDE gives in general slightly more optimistic results for the predictive performance. The reason is probably because fewer rounds have been run in KDE than in MATLAB, as more rounds tend to smooth out the curves. This can be seen in figures 13 and 14 where the curves in MATLAB are smoother than the curves in KDE - more than twice as many runs have been committed in MATLAB as in KDE.

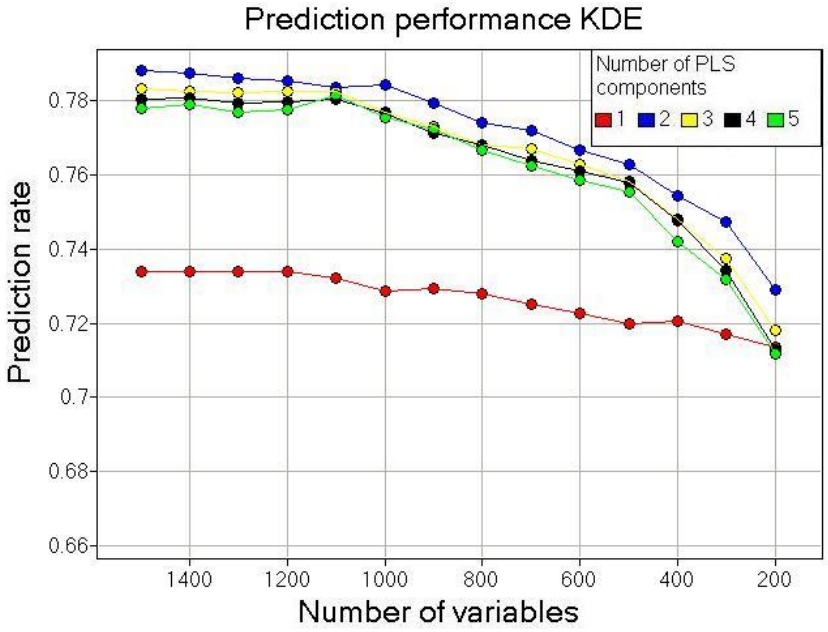


Figure 13: The predictive performance, cutoff 0,60 for 95 rounds in KDE.

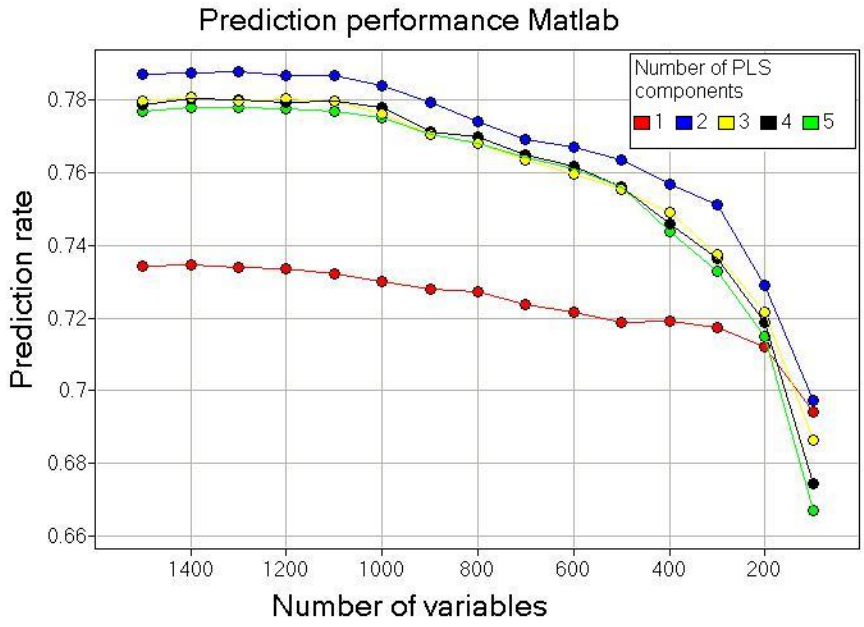


Figure 14: The predictive performance, cutoff 0,60 for 200 rounds in MATLAB.

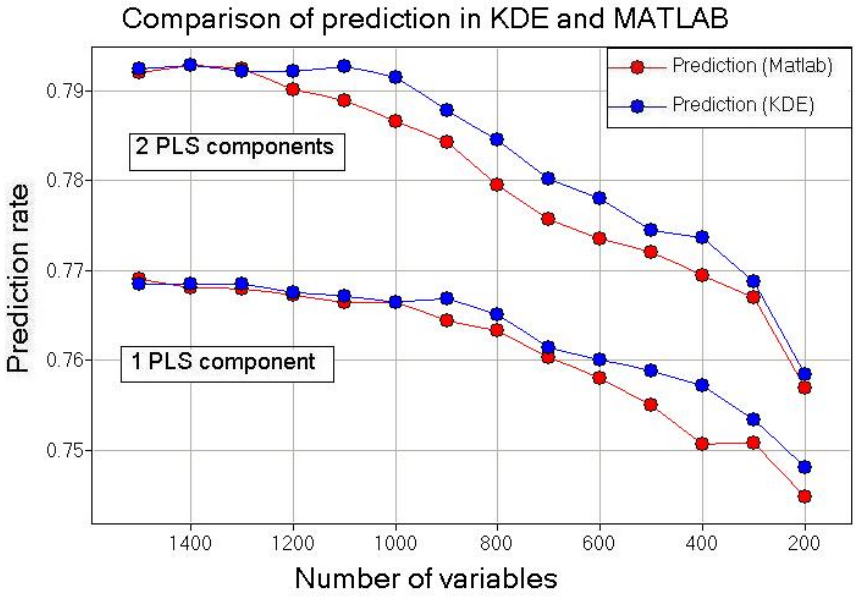


Figure 15: A comparison of the two first PLS-components in KDE and MATLAB. The prediction scores are very similar.

The dissimilarity could perhaps also be a consequence of the difference in PLS-weights between MATLAB and KDE. The overall impression is however that the results are fairly similar. In figure 15 the predictive performance for models with one and two PLS-components are compared. From the figure it is clear that the difference between prediction performance in KDE and MATLAB is minimal.

The highest prediction scores are 78,8% in KDE for two PLS-components and 1400 variables and 78,8% in MATLAB for two PLS-components and 1400 variables, which is very good predictive ability. See figures 13 and 14. If models with one PLS-component had the best scores it is possible that the data could be described with one single relation. The fact that models with two PLS-components have the highest prediction score gives a hint that there must be several underlying relations. The plots also show that the predictive performance decreases with variable selection.

There is a trade-off between the predictive ability of a model and number of variables. What is considered the best model must be a decision balanced on those two properties.

4.2 Evaluation of KDE

4.2.1 User-friendliness

KDE seems fairly easy to use at first glance. The concept of nodes and flow of information is straightforward and the basic idea is intuitive. InforSense provides some examples of workflows that demonstrate how to use some of the simpler nodes. However, as soon as workflows and nodes become more complicated, documentation is sparse and function not always obvious. There is little description of underlying methods in use, making it difficult to compare results with external analyses. For example, how is scaling done and what is the exact algorithm used for the PLS node?

There is a KDE scripting language available for performing column operations in script nodes and for setting parameters. The KDE language has some documentation, but there are no examples on how to use it. There is also very little documentation on how to integrate the external Java script language Groovy. It is supposed to be equivalent to the KDE language in some sense but there is no clear definition on when, where and how to use each language.

4.2.2 Functionality

Simple nodes, such as the numerous pre-processing nodes, are useful and fulfil their purpose, but more advanced nodes lack functions that are necessary for in-depth analyses. The workaround is to create nodes yourself. It is possible to develop nodes locally using a script node, but debugging is not available and can be complicated. A better way is to create nodes in Java. The set structure for nodes in KDE is fairly easy to understand and all Java functions are supported. Debugging is possible through Eclipse, a developing environment for programming languages, which makes it easy to find errors. Eclipse simplifies node development since the structure of each variable can be explored.

It seems as if InforSense has put effort into developing a wide range of analysis tools without extending the functionality of each tool outside basic needs. Anyone interested in a particular tool will find that there are features that are not supported. One example is the PLS node. The only results that can be obtained from the node explicitly are the weights. The node also outputs a model that serves as input for the PLSApply node. The model can be viewed with the visualisation tool Text Viewer where some parameters can be found such as the variance and mean of X and Y. The only way to extract this information is to create nodes yourself. It is reasonable to request other parameters besides the PLS-weights when performing even the simplest form of PLS.

4.2.3 Dysfunctionality

There is a global server for KDE at AstraZeneca. It could not be used for workflow development. One reason was that Matlab could not be run globally because of issues with license and Linux. Another reason was that it is necessary to be the administrator of a server in order to develop nodes. A third reason was that execution was faster on a local server with only one user than on the global server where many users share the resources.

The PLS/PCAApply nodes have an option to output the input data if there are 1000 columns at most. It is not possible to choose which columns to include in the output, so this option cannot be used on larger datasets. The workaround is to join the output to the original table to retain important columns such as keys and other parameters. This is a quite demanding operation that costs unnecessary time.

The results for the weights from the PLS node and MATLAB and SIMCA differ. The PLS node ranks the PLS-components in a different order than MATLAB and SIMCA. It identifies the first principal component as the component with the best correlation between X and X, instead of X to Y. This complicates comparing and validating results.

The output of some MATLAB nodes used to be of the type *object* instead of numeric format, which means that the output could not be applied to any numeric functions. This bug has been solved by InforSense, but it addresses another issue concerning type conversions. There is no efficient way to convert the format of an entire dataset. The node Column Converter works on one column at a time, which results in a tremendous amount of work when columns are numerous.

With nested For nodes that send parameters across levels there is a problem with parameter names: parameters on different levels cannot have the same name. A parameter will not be recognised by a workflow if more than one For node separate them. With four nested For nodes this means that some parameters must go through name changes four times.

Nodes with several output ports cannot be connected directly to nodes with several input ports. If a node outputs two tables, the same two tables cannot be input directly to a node with two input ports. An intermediate node is needed, even though it does not change the data in any sense. A Delete node can be used without deleting anything to separate two outputs, which seems unnecessary and time-consuming.

4.2.4 Development

The possibility to develop nodes is a strong feature of KDE. The node structure is fairly easy to understand and anything that can be done in Java is allowed. The documentation leaves a lot of unanswered questions, but there are a lot of example nodes that come along with the installation

that serve as a valuable source of information.

The source code for the existing KDE nodes is not available, so there is no possibility to extend KDE nodes.

There is no support for adding documentation to your own nodes in the way KDE nodes have documentation. The Development Guide claims that it is possible, but from personal communication with InforSense it is clear that this will not be available until next release of KDE.

It is necessary to have access to and be administrator of a KDE server in order to develop nodes. Node development and testing is performed on the server machine, which means that ordinary client users cannot develop nodes. However, anyone with a license can install a KDE server locally.

4.2.5 Stability

Due to cacheing of memory, the nodes sometimes do not update their metadata. When iterating over a workflow this can be a problem, especially when input columns change for each iteration. If the nodes do not update their metadata it is likely that some columns that should be included will not be selected. There are commands for selecting particular columns by using the KDE language instead of specifying each column name manually. These commands, that for example select all columns of a certain type, are very time-consuming.

The server occasionally shuts down during execution. It usually happens when there is an overflow of cached memory. The results from the current execution will be lost, but usually the server can be rebooted and the client reconnected. To avoid losing information from long executions it is possible to continually write results to a file but this takes time and is not an optimal solution.

The node "Export to Userspace" sometimes deletes the table it is exporting and also the file it was exporting it to.

4.2.6 Reusability

An advantage of KDE is that once a workflow has been created it can be reused infinitely, provided that the same dataset or a dataset of the exact same form is applied. Having a workflow that performs a standard analysis makes it possible to easily create or extend workflows for testing alternative parameter settings, etc. A workflow can be deployed as a generic component to be used as a node in other workflows.

Applying new data to an existing project involves going through the workflow by hand and correcting all inputs to the nodes to make sure that the correct columns are chosen, unless the data is on the exact same form with the same number of variables and the same column names. The caching of memory can cause problems. Each node needs to update its metadata

when new data is applied to a workflow and as this not always happens automatically nodes may have to be forced to update.

Nodes developed in Java become part of the selection of nodes in KDE and can be used in any workflow by any user on the same server.

Analyses made in KDE can be traced-back and recorded for documentation purposes.

4.2.7 Time aspect

Execution time is one of the biggest drawbacks of InforSense KDE. Compared to performing identical analyses in MATLAB, the difference in execution time is extreme. One run, i.e. one partition of data into three sets followed by a complete variable selection and PLS optimisation, takes 6200 seconds in KDE in average. In MATLAB the equivalent is 34 seconds, see table 4. That makes KDE 180 times slower than MATLAB. The result is that 200 runs were executed in MATLAB while in KDE the equivalent was 95 runs.

Number of runs	KDE (time in seconds)	MATLAB (time in seconds)	Complete MATLAB (time in seconds)
1	6200 (620)	34	40
100	∞ (1354)	3400	7300

Table 4: Execution time of one run and 100 runs in KDE and MATLAB respectively. One run means one three-fold bootstrap crossvalidation with 15 variable selections and five PLS-components. The MATLAB analysis includes computation of prediction parameters for one cutoff, while in KDE an additional workflow is needed for this. The execution time for that workflow is inside brackets. The "complete MATLAB" column shows the execution time for the originally intended analysis with ten PLS-components and 100 cutoff values.

Two of the reasons that could cause the slowness of KDE are (1) the Java platform, as Java is not known for being a fast executive language, and (2) that every single node must receive data, operate on data and transmit data in a very structural manner. A lot of simultaneous processes take place beside the computation. For example submitted tasks can be overviewed during execution, both as a thread of actions and directly in the workflow.

As there is a risk of losing data during long executions, an option is to save results to userspace continuously. But exporting data is time-consuming and only the most important results should be saved to cut down execution time. Saving data to a database is less time-consuming than writing to a file. The time of adding data to a table in a database is independent of the size

of the table in the database. The time it takes to read a file from Userspace, union it to new data and write the result to the same file, depends on the size of the file in Userspace. When the file is empty the execution time is identical to the execution time of adding data to a database. However, when the file is large the time increases considerably, see table 5.

Number of rows	File (time in seconds)	Database (time in seconds)
1	6201	6161
700575	18421	6678

Table 5: KDE execution time for saving a table to a file and a database during one run of the workflow described in section 3.5.2 respectively. Number of rows corresponds to the table size of the data already present in the file and the database before addition of new results. Execution time is practically independent of size when saving results to a database but increased almost three-fold when writing results to a large file.

Another time aspect that should be mentioned is the time it takes to build a workflow in KDE. Simple workflows consisting of a few nodes are easy to create and do not require much time. Time spent on developing workflows increases in proportion to the complexity of the workflow. This is mainly due to lack of documentation and bugs in the KDE nodes. Developing scripts in MATLAB that perform equivalently advanced analyses is considerably faster. The main analysis in this project took a few days to develop in MATLAB, compared to several weeks in KDE.

4.2.8 Bugs

A number of bugs were solved by InforSense during this project. The remaining bugs that have not already been mentioned are the following.

The metadata of the PLSApply node is not updated which makes it difficult to use the node in an iterative environment, for example where the number of PLS-components changes. The workaround is to set a parameter for the number of components and to force it to update using the script-part of the For node.

When working on the inner workflow of a For node the system may hang if a node is added from the right-click popup menu. The workaround is to drag and drop nodes from the component tree.

The output from the PLSApply node has the same column names as the output from the PCAApply node, i.e. PCAComp1, etc. It would be less confusing if the column names were something like PLSComp1, similar to the output of the PLS node.

4.3 Evaluation of MATLAB

The advantage of MATLAB is the computational power, i.e. the unlimited range of options to create functions that do exactly what is needed and the efficient execution.

The disadvantage is difficulties to visualise results and keep track of how and where functions are used.

4.4 Evaluation of KDE using MATLAB

Integrating MATLAB with KDE only has advantages. The execution time of MATLAB scripts does not differ significantly from executions in MATLAB alone. The advantage is the possibility to easily visualise results, both with Table Editor and with Spotfire. MATLAB results are integrated into KDE perfectly.

An interesting alternative would be to let all MATLAB functions be separate and represented by one Generic MATLAB node each. The nodes could be connected in a similar fashion as functions are called in MATLAB. However, iterations should preferably be handled by MATLAB scripts rather than with KDE nodes to reduce execution time. This combination would be like selecting the positive features from both KDE and MATLAB. The aspect of well-arranged functions in KDE combined with the fast execution of MATLAB should be a winning concept.

4.4.1 Dysfunctionality

A disadvantage is that MATLAB currently does not work on a global server, which means that anyone interested in using it will need a local server. The disadvantage of working on a local server is that sharing work with other users becomes complicated.

Not all MATLAB-scripts can be run in KDE. If the output has structs containing big amounts of data KDE cannot handle it. Structs are viewed with the MATLAB node MatViewer which has a limit of 1000 columns and 100 rows. They can be converted to the KDE table format with the MatViewer node but the size limit makes it useless for large datasets as the one studied here. The solution is to output MATLAB results as matrices, which are interpreted as tables in KDE.

The XML-wrapper used for uploading and maintaining MATLAB-scripts has an unnecessarily complicated structure. The scripts must be organised into a four-levelled file system. When changes are made in the XML-wrapper the Generic Matlab nodes that use the affected scripts do not update the changes. They must first be re-created or the entire workflow re-opened.

5 Conclusions and Recommendations

The idea of integrating many tools on one platform is brilliant, but the KDE platform feels somewhat immature and unstable. The range of functions available is exhaustive and outnumbers the workflow environment Pipeline Pilot from SciTegic. Pipeline Pilot is a workflow environment that previously has been used mainly in chemometrics. Another strong feature of KDE is the possibility to develop your own nodes that complete and personalise your analysis, which is not possible with Pipeline Pilot. This can compensate for the lack of function that some nodes suffer. While a wide range of analysis tools is covered, only basic functions are supported for more complex nodes. On the other hand, simpler nodes such as the pre-processing nodes are very useful and functional.

The structured overview of the analysis flow is a very good feature of KDE. The many options available to visualise data at any point in the workflow makes it easy to trace and modify every step of the analysis. Another asset is the possibility to create workflows that can be reused by many users over and over again. A workflow can be used as guidelines for other analyses and be modified to suit current demands.

The greatest drawback of InforSense KDE is instability. There are many bugs in version 3.1, even though many of them have been corrected for during this project. Cacheing of memory creates problems with selecting the correct data source during an analysis. This makes it difficult to trust that a workflow performs the correct analysis unless all nodes have been manually controlled. A workflow that already has been executed and its results validated can during a subsequent execution give different results or error messages.

Another weakness is the exceptionally long execution time of workflows. It is not reasonable to have a 180 times slower execution in KDE compared to MATLAB. It seems as if InforSense did not expect as extensive and computation-demanding workflows as the ones that were made for this project. It is however reasonable to request the possibility to create workflows of this complexity. The small and simple examples they provide all have relatively fast executions, but that is because they only involve up to ten nodes. The most complex workflow in this project has 70 nodes that were executed 225 times in one round.

The overall impression of InforSense KDE is that it has great ambitions, but does not manage to fulfil essential features such as stability, reasonable execution time and more than basic functionality of complex nodes. If you are looking for a reliable platform where you can develop advanced analysis flows in a straight-forward manner, KDE is not it. The advantages of reusability, wide range of functions and ability to develop your own

nodes cannot outweigh the disadvantages of instability and extreme time consumption. However, to use KDE as a platform for integrating different software tools is excellent. Tools such as MATLAB and Spotfire can easily be accessed within KDE.

My recommendation is to use InforSense KDE for integrating tools such as MATLAB and Spotfire but not to build complex workflows, especially not with iteration.

The next release of InforSense KDE, version 4.0, is coming up this spring. It would be very interesting to see what is new and if the stability has been improved. InforSense has promised an entirely new platform with more functions and many new features. But new functions and features cannot compensate for immense execution time and an unstable platform.

6 Acknowledgements

I would like to thank my supervisors Kerstin Nilsson and Hugh Salter at AstraZeneca R&D for support during this project. I would also like to thank Lena Milchert for helping me to get started with KDE and for useful discussions on issues with KDE. I wish to thank Alan Saied at InforSense support for answering my never-ending questions on KDE and for letting me visit InforSense in London. I want to thank my opponents Rezin Dilshad Mohammad and Christine Andersson for useful comments on the report. I also wish to thank my scientific reviewer Erik Bongcam-Rudloff at the Linneaus Centre for Bioinformatics.

Finally I would like to thank all the nice people at AstraZeneca and in particular my student colleague Beata Werne.

7 References

Ambroise C, McLachlan GJ. *Selection bias in gene extraction on the basis of microarray gene-expression data.* PNAS 2002, 99 6562-6566

Bielekova B, Martin R. *Development of biomarkers in multiple sclerosis.* Brain 2004, 127 1463-1478

Bø TH, Jonassen I. *New feature subset selection procedures for classification of expression profiles.* Genome Biology 2002, 3(4):research00017.1-00017.11

Calabresi, PA. *Diagnosis and Management of Multiple Sclerosis.* American Family Physician 2004, 70 1935-1944

Chang J, Van Remmen H, Ward WF, Regnier FE, Richardson A, Cornell J. *Processing of Data Generated by 2-Dimensional Gel Electrophoresis for Staitstical Analysis: Missing Data, Normalization, and Statistics.* Journal of Proteome Research 2004, 3 1210-1218

DeJong, S. *SIMPLS: an alternative approach to partial least squares regression.* Chemometrics and Intelligent Laboratory Systems 1993, 18 251-263

Eriksson L, Johansson E, Kettaneh-Wold N, Wold S. *Multi- and Megavariate Data Analysis.* Umetrics Academy, Umeå 2001. ISBN 91 973739 1 X

Freyhult E, Prusis P, Lapinsh M, Wikberg JES, Moulton V, Gustafson MG. *Unbiased descriptor and parameter selection confirms the potential of proteochemometric modelling.* BMC Bioinformatics 2005 6 50

Hastie T, Tibshirani R, Friedman J. *The Elements of Statistical Learning.* Springer Verlag, New York 2001. ISBN 0 387 95284 5

Ibrahim SM, Gold R. *Genomics, proteomics, metabolomics: what is in a word for multiple sclerosis.* Current Opinion in Neurology 2005, 18 231-235

InforSense KDE 3.1 Development Guide

InforSense KDE 3.1 Installation and Setup Guide

Kenrick KG, Margolis J. *Isoelectric focusing and Gradient Gel Electrophoresis: A Two-Dimensional Technique.* Journal of Analytical Biochemistry 1970, 33 204-207

LaBaer, J. *So, You Want to Look for Biomarkers (Introduction to the Special Biomarker Issue).* Journal of Proteome Research 2005, 4 1053-1059

Marengo E, Robotti E, Antonucci F, Cecconi D, Campostrini N, Righetti PG. *Numerical approaches for quantitative analysis of two-dimensional maps: A review of commercial software and home-made systems.* Proteomics 2005, 5 654-666

Noseworthy JH, Lucchinetti C, Rodriguez M, Weinshenker BG. *Multiple Sclerosis.* The New England Journal of Medicine 2000, 343 938-952

O'Farrell PH. *High resolution two-dimensional electrophoresis of proteins.* The Journal of Biological Chemistry 1975, 250 4007-4021

Rosengren AT, Salmi JM, Aittokallio T, Westerholm J, Lahesmaa R, Nyman TA, Nevalainen OS. *Comparison of PDQuest and Progenesis software packages in the analysis of two-dimensional electrophoresis gels.* Proteomics 2003, 3 1936-1946

Soeria-Atmadja D, Wallman M, Björklund ÅK, Isaksson A, Hammerling U, Gustafson MG. *External Cross-Validation for Unbiased Evaluation of Protein Family Detectors: Application to Allergens.* PROTEINS: Structure, Function, and Bioinformatics 2005, 61 918-925

Vining DJ, Gladish GW. *Receiver Operating Characteristic Curves: A Basic Understanding.* Radiographics 1992, 12 1147-1154

Wickenberg-Bolin U, Göransson H, Fryknäs M, Gustafson MG, Isaksson A. *Improved variance estimation of classification performances via reduction of bias caused by small sample size.* BMC Bioinformatics 2006, 7 127

Wold S, Sjöström M, Eriksson L. *PLS-regression: a basic tool of chemometrics.* Chemometrics and Intelligent Laboratory Systems 2001, 58 109-130

Wold S, Esbensen K, Geladi P. *Principal Component Analysis.* Chemometrics and Intelligent Laboratory Systems 1987, 2 37-52

SWISS 2DPAGE, http://www.expasy.org/swiss-2dpage/viewer&ac=all&map=ECOLI, 13 february 2007.

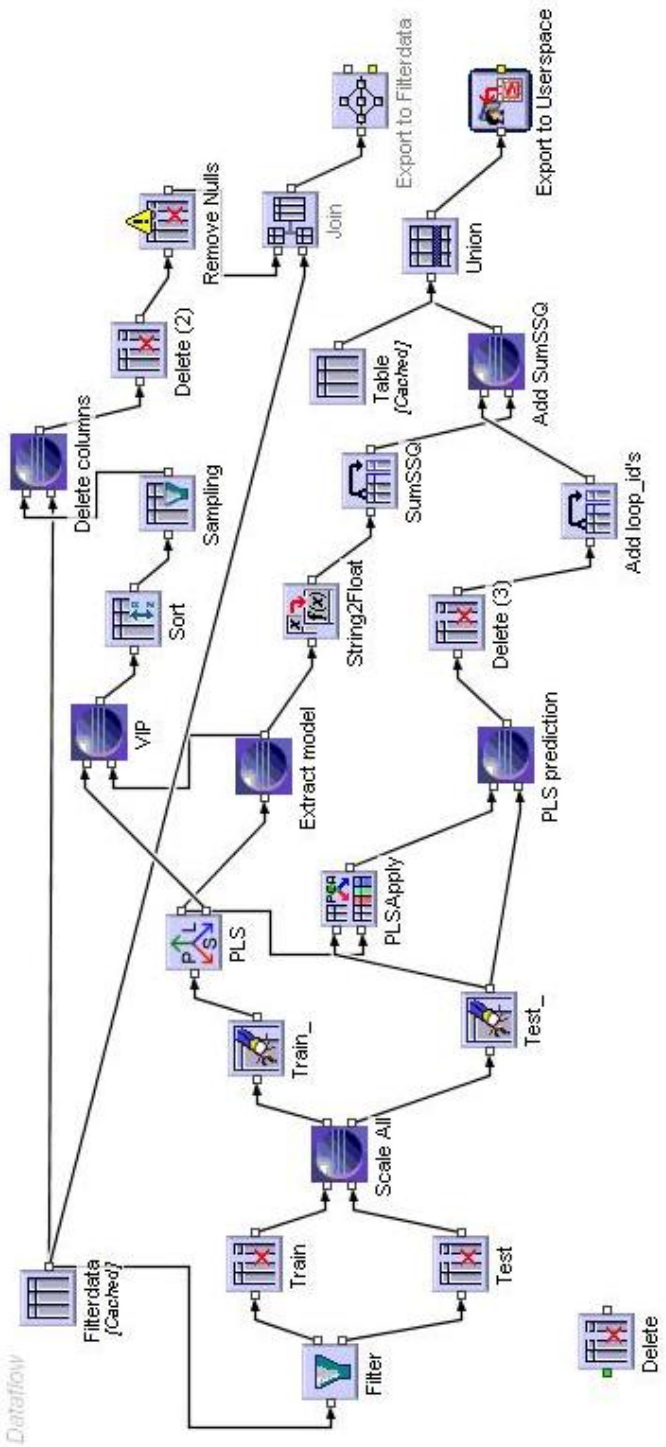
APPENDIX

Appendix A

Abbreviations

2D PAGE	Two-dimensional polyacrylamide gel electrophoresis
FN	False negative
FP	False positive
J2EE	Java 2 platform, enterprise edition
MS	Multiple sclerosis
MW	Molecular weight
NIPALS	Non-linear iterative partial least squares
NPV	Negative predictive value
PC	Principal component
PCA	Principal component analysis
pI	Isoelectric point
PLS	Partial least squares
PP	Primary progressive multiple sclerosis
PPV	Positive predictive value
ROC	Receiver operating characteristic
RR	Relapse remitting multiple sclerosis
SDK	Software development kit
SIMPLS	Straight-forward implementation of the PLS method
SP	Secondary progressive multiple sclerosis
TN	True negative
TP	True positive
VIP	Variable importance to the projection

Appendix B. Figure 16: Screenshot of InforSense KDE



Appendix C. Figure 17: Main workflow

Node	Input	Action	Output
ScaleAll	Table with X and Y-blocks	Auto-scales or mean-centres selected columns	Scaled table
ExtractModel	PLS model	Extracts variance from model	Variance as a table
VIP	Weights from a PLS node and variance from ExtractModel	Computes VIP score for all variables	VIP score for all variables
DeleteColumns	Column with column names to be deleted and a table	Deletes columns from a table	Table
PLSPrediction	Table with Y-predictions and table with correct Y-values	Calculates prediction parameters	Table with prediction parameters
AddColumns	Parameters and Table	Appends parameters as constant columns to table	Table with constant columns

Appendix D. Table 6: Nodes developed in Java