Applying data integration into reconstruction of gene networks from microarray data

by

Grzegorz Zycinski

Theses Series

DISI, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy

http://www.disi.unige.it/
Applying data integration into reconstruction of gene networks from microarray data

by

Grzegorz Zycinski

July, 2012
Title: Applying data integration into reconstruction of gene networks from microarray data

Advisor: Alessandro Verri
DISI, Univ. di Genova
alessandro.verri@disi.unige.it

Ext. Reviewers: Nicola Ancona
CNR, Bari
ancona@ba.issia.cnr.it

Giuseppe Jurman
FBK, Trento
jurman@fbk.eu
Abstract

The great amount of information produced in the field of molecular biology and genetics opened enormous possibilities for life science researchers to analyze and understand biological phenomenons. This wealth, however, brings new problems, such as how to integrate existing information sources, and how to use already well-developed statistical techniques in the process.

In this thesis, we investigate a possible approach for integration of data produced by high–throughput technologies with available prior biological knowledge. We focus on fusing available data produced by microarray technology, together with established source of biological knowledge, represented by Gene Ontology, into complete statistical methodology for devising significant gene signatures, important for multifactorial diseases (e.g. cancer, neurodegenerative disorders). The idea presented is extensible to different kinds of data and knowledge that display desirable properties.

In this thesis, the statistical methodology is described, the prototype system is detailed, and experimental results are presented. The methodology is applied to gene expression data obtained from patients with Parkinson’s disease, Alzheimer’s disease, as well as prostate cancer, and used to study the pathologies by comparing disease subjects versus controls, as well as to uncover biological activities that change in different stages of the selected pathologies.

Preliminary results indicate that the direction taken is very promising in successful delivery of important biological facts. The implementation was conceived with efficiency in mind, and has already become a foundation for new generation of data exploratory tools that are both data– and knowledge–driven.
To my family
Imagination is the only key to the future. Without it none exists—with it all things are possible.

(Ida M. Tarbell)
Acknowledgements

First of all, I would like to thank my advisor, Alessandro Verri, and all my colleagues at computational biology branch of SlipGURU group at DISI, Genova: Annalisa Barla, Margherita Squillario, Salvatore Masecchia, and Sofia Mosci, for their hospitality and support, as well as for providing challenging and rewarding environment. It is a pleasure working with you all.

I would also thank Barbara Di Camillo and Tiziana Sanavia from Dipartimento Di Ingegneria Dell’Informazione, Università degli Studi di Padova, whose ideas have greatly influenced this work.

Next, I would like to thank the following people that have crossed my path in right moments, in academic, professional and private life: Saverio, Gabriele, Alessandra, Giovanni, Luca, Francesca, Nicoletta, Silvia, Curzio, Veronica, Alessandro, Luca, Nicola, Laura, Paolo, Federica, Francesca, Paola, Rafal, Sebastian, Robert, Paul, Piotr, Witold, Sebastian, Pawel, Fergal, Jakub, Michal, Maciej, Patrycja, Krzysztof, Andrzej, Lukasz, Maciej, among others. Meeting you, discussing with you, and learning from you, was great impulse to go forward and simply to strive to be better in every aspect of my life.

My warmest thoughts go to my parents Anna and Zbigniew, my sister Ewa, and my grandmother Marianna, for making all of this happen, and for their constant love and support.

Finally, I would like to thank especially Julita, Matteo, and Elisa. Without their presence and support, this thesis would quite probably never materialize.

’Tis done.
Contents

Chapter 1 Introduction .............................................. 5
  1.1 Motivation ....................................................... 5
  1.2 Contribution ................................................... 7
  1.3 Content ......................................................... 8

Chapter 2 Data and knowledge ...................................... 9
  2.1 General concepts ............................................... 9
  2.2 Integration in context of life sciences ......................... 10
    2.2.1 The idea .................................................. 10
    2.2.2 Bioinformatics approach .................................. 11
    2.2.3 Machine learning approach .............................. 16
  2.3 Summary ....................................................... 20

Chapter 3 Current state of affairs ................................ 21
  3.1 General process ................................................. 21
  3.2 Variable selection ............................................. 22
  3.3 Variable selection with $\ell_1$ ................................ 24
  3.4 Enrichment with prior knowledge ............................. 26
    3.4.1 Gene Set Enrichment Analysis .......................... 27
    3.4.2 WebGestalt ............................................... 28
  3.5 Summary ....................................................... 29
Chapter 4  Knowledge Driven Variable Selection  

4.1 Prior knowledge as a guide  

4.1.1 General idea  

4.1.2 Features of the approach  

4.1.3 Difficulties in the approach  

4.2 Implementation  

4.2.1 Overview  

4.2.2 Input data  

4.2.3 KDVS core  

4.2.4 Knowledge discovery  

4.2.5 Post-processing and visualization  

4.2.6 Extensions  

4.3 Summary  

Chapter 5  Experimental results  

5.1 Introduction  

5.2 Overall procedure  

5.3 Time line  

5.4 Experiment 1 - proof of concept on PD data  

5.4.1 Goal  

5.4.2 Samples characteristics  

5.4.3 Experiment details  

5.4.4 Results  

5.5 Experiment 2 - Comparison of AD/PD  

5.5.1 Goal  

5.5.2 Sample characteristics  

5.5.3 Experiment details  

5.5.4 Results
Chapter 1

Introduction

1.1 Motivation

The practice of finding relationships between bits of research data has been critical for the development of modern science. Machine learning domain has contributed considerably here, with its rigorous abstract foundations. Such foundations are generally data–driven, where all bits are cast into uniform abstractions, such as numerical vectors, and the algorithms for finding relationships have strong mathematical foundations based on the mathematical properties of such objects. This allows using standardized protocols for analysis of research data. For research fields that rely mostly on quantitative data, such as physics, this approach has been sufficient for many applications [HTF01].

Another great contribution to this practice has come from rule–driven approaches, such as expert systems theory, rule–based reasoning, formal semantics etc. Here, the known facts associated with particular domain of activity are recognized and organized in rules. Such known facts are traditionally referred to as knowledge. In this context, the algorithms for finding relationships are modeled after human–based reasoning and formal logic. This approach has been used, among others, in research fields that relied strongly on qualitative data, such as medical diagnostics [Lug08].

There are cases, however, when besides research data, e.g. obtained with some instrumentation, that are easy to unify within numbers or vectors, one must use also knowledge that was produced by human mode of thinking. One of the most striking examples of this necessity comes from life science domain.

Traditionally, the life science domain was linked to the great amount of knowledge acquired over the years, subsequently refined and codified in textbooks, while gathering it was based on variable, but not overwhelming, amount of experimental data available to
researchers [Str95]. During last several years, however, this view has changed. Two factors are typically attributed to such change. First, the advent of widespread adoption of computer science concepts in life science research methodology; second, the increasing amount of experimental data generated by increasingly sophisticated research instruments [Kel08].

Pioneer works in the early adoption of computers and software in life science can be traced back to the Nineteen Seventies [Wat95, SK99]. They were conducted initially in two parallel tracks: the analysis of biological sequences, such as DNA, RNA and proteins (especially search and retrieval), and analysis of biological molecules with X-ray crystallography. Two among the most known databases in life sciences originated in that time, namely GenBank and Protein Data Bank [Str11, Ber08]. Here, the focus was put on the “bending” of biological facts to be fit into framework of computer science concepts. For example, a DNA sequence has become simply a string of characters, and by using string matching and sequence alignment, life science hypotheses could be stated and later confirmed [DHH+83].

Later, when the amount of accumulated data has grown, bioinformatics was born and primary protocols for researchers were established [Wat95]. Today, biological databases became standard tools for life science [BO01].

The initial historical focus on data that can be fed easily into computer science frameworks, spawned the trend of creating massive data warehouses that integrate many kind of data, the classical example being modern GenBank [BKML+05]. The trend has progressed in parallel with increasing computing power and data storage capabilities, bringing new problems of making good use of such amount of data [GS08].

In addition, new research instruments are being constantly introduced to the field, that can acquire completely new information unavailable before, such as gene expression [LG06], offering great amount of experimental data to the researchers. Since such data is often numeric, an increasing amount of attention is directed to the techniques that can discover hidden relationships within data itself, such as machine learning [HTF01].

In the data–rich perspective, the knowledge accumulated over time by researchers, is codified to be easily available for searching and retrieval, and most frequently became another kind of data. That setting fits into the classical data–driven approach, and well–established methods can be used. Many early problems of life sciences were solved this way [Wat95, TCC+07].

However, with the amount of available research data constantly increasing, the need for explicit recognition of the complex properties of knowledge, in order to better use the present wealth of data, is becoming more clear in both bioinformatics [Phi08, BNT+08] and machine learning [HTF09] communities.
1.2 Contribution

In this thesis, we present an effort to tackle the problem of effective using of knowledge in bioinformatics and life science data processing in general. We aim at combining well-established ideas from data-rich perspective, mastered within bioinformatics, with knowledge-aware approaches, offered by statistical learning, to solve the real life science problems.

In this context, we also present a first step towards a new generation of learning schemes, where the complex nature of prior knowledge is fully recognized. Based on that, the integration of data and knowledge, in order to find meaningful relations between data bits, can be performed in more sophisticated way than in pure data-driven approach.

The purpose is to mine existing data obtained from high-throughput experiments, to discover meaningful relations between data bits, that in turn can be translated to sound life science concepts. More specifically, we strive to discover new knowledge, based on data in various stages of processing, using some additional information available, and utilizing prior knowledge. The schema of the process is shown on Figure 1.1. The emphasized terms are general enough that they can be understood in different ways; this topic is discussed in more detail in chapter 2.

![Diagram of the process](image)

Figure 1.1: The aim of the thesis can be conceptualized as discovering new knowledge from data in various stages of processing, additional information and prior knowledge, using specific discovery method.

To perform such discovery, we use well-established statistical methodology of supervised classification, in the form of “disease vs healthy” experimental setup, a setting popular in modern life science experiments. This step produces meaningful relations that are not “readable” directly by life science researchers. Next, we use annotation process to enrich such relations to make them accessible in descriptive way by life science researchers.

Moreover, we are interested in exploring some specific properties of prior knowledge that can be useful during the discovery process. More specifically, we focus on the existing knowledge that expresses certain level of structural hierarchy, that can be effectively mapped to
individual data bits. In this way, the hierarchy is “transferred” into the data domain, which affects the procedure of discovery of meaningful relations. As for data, we use gene expression measurements obtained from microarrays.

Still in this context, we are not modifying core statistical techniques used in discovery procedure. Instead we adapt existing data to display certain properties, in accordance with prior knowledge.

In this thesis, we devise a methodology for the practical case of such exploration of structural hierarchy, and we present a proof–of–concept software pipeline that implements such concept. We also formulate initial remarks of possible generalization of the investigated proof–of–concept idea into more abstract system.

1.3 Content

This thesis is organized as follows. In chapter 2 we discuss the meaning of data and knowledge, two key concepts that may be understood differently in various research communities. Also there, we present the idea of data and knowledge integration, focusing on two different perspectives: bioinformatics–related and machine–learning–related. Also, we present few practical examples of such integration. In chapter 3 we discuss the current state of affairs in the topical domain of the thesis. We follow the most popular methodology for discovering of new functional relations from microarray data with prior knowledge. In this context, we present in more detail the core concept of variable selection, used in the discovery of new relations in data bits. Also here, we discuss our feature selection technique of choice, namely $\ell_1\ell_2$, used further in this thesis. Next, we present common way of using prior knowledge with variable selection, the enrichment procedure. In chapter 4 we focus on the presentation of methodology of the proof–of–concept approach, as well as the implementation of the approach itself. In chapter 5 we present preliminary results, obtained from the application of proof–of–concept approach to real microarray experiments. Finally, in chapter 6 we present a brief summary of the implementation of proof–of–concept approach. Also here, we present more elaborated comments on the experimental results. We conclude with brief discussion of possible future work that can further clarify and expand the proof–of–concept approach into efficient data and knowledge integration tool.
Chapter 2

Data and knowledge

In this chapter, we discuss the concepts of data and knowledge. Various research communities understand these intuitive terms in their specific way. The main topic of the thesis, as well as the methods of its implementation, very often crosses boundaries between machine learning and bioinformatics. Therefore, it is crucial to present a comparison of most common synonyms, in the context of used techniques, and establish uniform terminology for all the activities described later.

Part 2.1 presents abstract definitions used in knowledge engineering, and places them in the context of life sciences. Part 2.2 focuses on actual integration of data and knowledge in life sciences, presenting some representative examples.

2.1 General concepts

First and foremost, there is no formal definition of terms like data, information and knowledge. However, the consensus based on various contexts suggests the following characterization. Data can be described as set of raw facts that by themselves are without meaning. Information can be described as data that has associated some interpretation, accessible to human mode of thinking. Finally, knowledge can be described as information that can be understood, in terms of forming an abstraction over it, and utilizing it to direct activities and decisions in material world. For further discussion, see for example [KC07].

Following the above reasoning, in the context of life sciences, data can be associated with quantitative measurements performed over instance of some biological phenomenon. Information can be associated with descriptive aspects of the phenomenon in question. Knowledge can be described as results of discovering abstract properties behind phenomenon, that can be formulated with logical rules. In this context, the terms behind the aim of the
thesis, presented on Figure 1.1 can be articulated more precisely.

Both machine learning and bioinformatics communities are realising tasks similar to ours using diverse methodologies, and their theoretical aspects are explored in knowledge engineering community. Each community uses its own specific synonyms for the building blocks of those tasks, as shown in Table 2.1.

<table>
<thead>
<tr>
<th>Term</th>
<th>Machine Learning</th>
<th>Bioinformatics</th>
<th>Knowledge Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw data</td>
<td>data</td>
<td>data</td>
<td>information</td>
</tr>
<tr>
<td>Processed data</td>
<td>data</td>
<td>data</td>
<td>information</td>
</tr>
<tr>
<td>Additional information</td>
<td>N/A</td>
<td>data/information</td>
<td>information</td>
</tr>
<tr>
<td>Prior knowledge</td>
<td>prior knowledge</td>
<td>data</td>
<td>knowledge</td>
</tr>
<tr>
<td>Discovery method</td>
<td>learning phase</td>
<td>data mining</td>
<td>knowledge discovery</td>
</tr>
<tr>
<td>New knowledge</td>
<td>inferred knowledge</td>
<td>entity/relations</td>
<td>knowledge</td>
</tr>
</tbody>
</table>

Table 2.1: General terms used throughout the thesis, as understood in different research communities.

2.2 Integration in context of life sciences

In this section, we briefly present the concept of data and knowledge integration, as studied in the context of life sciences. Regarding specificity of the research questions in those fields, various needs must be fulfilled.

Part 2.2.1 presents the general overview of the problem. Part 2.2.2 focuses on the implementation in bioinformatics, and presents some examples. Part 2.2.3 focuses on the implementation in machine learning, and presents some examples.

2.2.1 The idea

In general sense, data integration refers to combining existing sources of information to either present uniform view over it, or to discover new pieces of information. Across various domains, however, the detailed meaning depends much on the characterization of “data” [Ull97].

Life science field is in particular situation here. Over the years, a huge quantity of biological and chemical knowledge was collected and encoded in human-readable form. Incoming streams of new data provide unprecedented opportunity for investigations, however the following problems arise. Data streams are often isolated and their formats are incompatible. There are no common standards for holding and manipulating information processed in
2.2.2 Bioinformatics approach

In the context of bioinformatics, the term *data* may be encountered in yet different role. Here, there is no clear distinction between data, information and knowledge (as shown in Table 2.1), and every piece of information is referred simply as “data”. The topic of effective and successful aggregation of all available data in order to uncover the underlying mechanisms of biological phenomena is referred to as “data integration”. Considerable amount of research is devoted to this field; see for example [LKI10] for latest reports.

Many public databases offer different data in different contexts. Their comprehensive list, Molecular Biology Database Collection, is updated yearly [GC11]; see also MetaBase [MEB]. Some databases offer dedicated specific content, while some provide specialized aggregated views of the data from singular more specific databases.

Below we present few representative examples of implementation of data integration in bioinformatics.

**Specialized content provider**

BRENDA [BRN] is a specialized relational data base for providing thematic information regarding enzymes. It contains information on properties of all classified enzymes, including data on the occurrence, catalyzed reaction, kinetics, substrates/products, inhibitors, cofactors, activators, structure and stability. All data are connected to literature references. The data and information provide a fundamental tool for research of enzyme mechanisms, metabolic pathways, the evolution of metabolism and, furthermore, for medicinal diagnostics and pharmaceutical research. The database is a resource for data of enzymes, classified according to the Enzyme Commission (EC) system of the International Union of Biochemistry and Molecular Biology (IUBMB) Enzyme Nomenclature Committee, and the entries are cross-referenced to other databases, i.e. organism classification, protein sequence, protein structure and literature references [SCS02]. The content is actively updated [SGC+11]. Partial example output for alcohol dehydrogenase (EC 1.1.1.1) enzyme is presented on Figure 2.1.

**Relational data warehouse**

BioMart [BIM] is an open source data management system that comes with a range of query interfaces that allow users to group and refine data based upon many different
criteria. Data from individual sources (flat files or relational) are converted into “data marts”, in the form of engineered relational tables, which can be accessed manually via standardized web interface and programmatically via API. The software features a built-in query optimizer for fast data retrieval. It provides domain-specific querying of a single data source or can function as a one-stop shop (web portal) to a wide range of BioMarts [HBS⁰⁹]. If any involving data sets share common identifiers, or mappings to a common genome assembly, they can be used to link BioMarts together in integrated queries. The data sets do not have to be located on the same server or at the same geographical location. In addition, each site can utilize their own domain expertise to deploy specific BioMart instance [SHB⁰⁹].

The concept of BioMart is presented on Figure 2.2. Partial structure of mart based on Ensembl¹ as seen in Schema Editor (part of MartConfigurator), is presented on Figure 2.3.

Non relational warehouse

CORE576 [Mai⁰⁶] is a prototype system developed for operating mass spectrometry systems in biological laboratory, designed with non-relational methodology. It utilizes UltraStructure design theory [MLHG⁰⁹], developed for implementation of business processes, that uses notational paradigm of rules to describe complex systems. The system is com-

¹http://www.ensembl.org/
posed of three conceptual “substructures”, presented on Figure 2.2. Here, every piece of knowledge, information, or data is a rule, and interplay between rules is driving force behind all activities of the system.

**Deep structure** is the static core of the system that comprises of ruleforms and animation procedures. Ruleforms are syntactic and semantic forms for expressing rules, relevant to application domain. Each ruleform is implemented as standard relational database table, where rows are the rules that conform to this ruleform. Special network ruleforms express relationships between rules in the form of subject/predicate/object, where all three are also ruleforms. Note that in such model, relationship itself is a rule. Animation procedures is simply any computer code that manipulates the rules, and performs automatic rule inference if needed to “reach” all related rules. The deep structure can represent informational content of hundreds of traditional relational tables by few dozens of essential ruleforms.

**Middle structure** describes “laws” of the system, essentially the content of ruleforms, such as types of data used in system operations and logical relationships between them. The informational content of middle structure consists of expert and user knowledge that is independent of deep structure, and may be modified by non–programmers.

**Surface structure** is actual appearance of real–world system. Its informational content is “generated” automatically by animation procedures, and functional content is provided by developer in the form of user interface. End user interacts with this layer by issuing queries, requesting display of certain elements etc.

In the context of laboratory system, all the entities involved in laboratory environment (like cells, molecules, DNA, membranes, test tubes etc.) can be described as “BioEntities”, and are collected in single ruleform. Various relations between elements of BioEntities may be then expressed, like “Protein ABC is-a Tyrosine Kinase”, “Adenosine is-a Nucleoside”, “Freezer5 contains SampleX”, or “IonX is-derived-from PeptideY and PeptideY observed-
in ExperimentZ” with proper network ruleforms. Thus, the expression of rules involving all entities in the system is effectively decomposed into network of ruleforms, and all inferred relations are calculated in the background by animation procedures.

As a result, the system is very dynamic while retaining its core logical structure. While data sources may evolve very quickly, as well as technology that produces them, the fundamental properties and relationships between biological/molecular/laboratory entities are still the same. Application of non-relational semantically-oriented design methodology, such as Ultra-Structure, captures this aspect clearly, and thus it becomes a very interesting example of still unexplored ways to deal effectively with data integration task.

**Portal oriented provider**

The University of California, Santa Cruz (UCSC) Genome Browser (GB) [UGB] is a source of genome sequence data from variety of vertebrate and invertebrate species and major model organisms, integrated with a large collection of annotations aligned to the biological sequences. The Browser itself is a graphical viewer that supports fast interactive performance. It was built on top of relational database for rapid visualization, examination, and querying of the data at many levels. The project was started in the early days of Hu-
The annotations offered by GB are conceptually divided into data sets called “tracks” that can be presented across the genome [KSF+02]. Initially, tracks were provided only by browser curators. Today, custom tracks can be added by independent researchers. Available tracks include: mRNA alignments, mappings of DNA repeat elements, gene predictions, gene-expression data, disease–association data (representing the relationships of genes to diseases), mappings of commercially available gene chips (e.g., Illumina and Agilent) etc.

The basic paradigm of display is to show the genome sequence horizontally, together with selected annotation tracks. Different colors may be used for different tracks. Such color management enables presenting many data types on a single axis, which provides visually–oriented vertical integration of annotation data.

APOE is well–known gene associated with Alzheimer disease[^1]. The example output from UCSC Genome Browser, presenting some of its features, such as coverage by probes from Affymetrix HG-series microarrays, and current allelic variants connected to diseases, is

[^1]: http://www.omim.org/entry/107741
2.2.3 Machine learning approach

In the context of machine learning, the term data usually refers to numbers, i.e. quantitative information, and the term knowledge usually refers to qualitative non-numerical information, such as plain text, binary streams, graph-like structures, and so on [HTF01].

Since the amount of data produced by existing and emerging life science technologies is rising at fast rate, there are many attempts to mine current data and knowledge stores utilizing existing effective statistical approaches [FSB08].

In the context presented in Part 2.1, the term data and knowledge integration may now be expressed more clearly. Our purpose is to combine existing data sources with prior knowledge sources, using established and replicable approach, to enrich the overall sum of information about biological phenomenon in question, by discovering “hidden” facts or producing new, more structured ones. The task itself may be performed in variety of ways. Here, we focus on approaches that use machine learning methods.

Historically, first attempts to implement this schema have arisen from the earlier works of statistical data integration, that worked purely on data streams. Since many established algorithms worked only on numerical information, there was a need to quantify non-numerical
objects into numbers. Hence, quantified knowledge was included into learning framework in the form of another data stream.

This procedure, while straightforward to implement, brings new problems. First, the knowledge quantification induces possible information loss, when numerical representation is not intuitively clear. It is even more evident when dealing with well-structured information, e.g. in the form of graphs. Second, working only on numerical data sets imposes careful choice of learning technique to use, especially when dealing with large data sets, where one may face ill-posed statistical problems [HTF01].

However in the last time, the recognition of complex nature of knowledge is getting appreciated, and many approaches take into account the additional level of structure present in knowledge [FSB08].

In the context of machine learning, the knowledge can be characterized as “a priori” or “a posteriori”, in terms of its use in the learning process. However, the knowledge itself may also be treated as “prior” or “posterior”, in terms of the outcome. In Table 2.2 we present most common naming issues.

<table>
<thead>
<tr>
<th>Prior knowledge a priori</th>
<th>Existing knowledge is used before or is influencing learning process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior knowledge a posteriori</td>
<td>Existing knowledge is used after learning process has been performed</td>
</tr>
<tr>
<td>Posterior knowledge</td>
<td>Knowledge obtained by any method of knowledge discovery, such as machine learning, data mining, or logical deduction</td>
</tr>
</tbody>
</table>

Table 2.2: Common examples of “a priori” and “a posteriori” terminology.

Here, we present some representative cases of integration. Part 2.2.3.1 presents one of the most effective implementation of integration when prior knowledge is quantified and included, at the relatively low level, into learning framework. Part 2.2.3.2 presents the case where more elaborate structure of prior knowledge is recognized and included effectively into learning technique.

2.2.3.1 Kernel integration

Machine learning (ML) approach solves problems that are defined only by a series of cases or example, rather than by predefined rules. Based on set of data points (patterns, samples) associated with desired outcomes, the ML approach tries to uncover the relationships between patterns and outcomes, based only on the given set. The goal is to predict correct outcome on any previously unseen data point. The data points are usually composed of a
series of variables specific for the problem (features). For more detailed introduction, see for example [HTF01].

This general framework can be made concrete in many ways. One of the most effective idea is to map data points into a high dimensional feature space, where each coordinate corresponds to one feature of the data items, transforming the data into a set of points in an Euclidean space. From there, again, a variety of methods can be used to find relations in the data.

In kernel methods, manipulations on those points can be based on specific functions called kernel functions. Proper kernel functions can capture different aspects of “similarity” between features mapped into Euclidean space [Vap98].

We focus on two examples of gene prediction in yeast, each exploring different aspects of integration. Here, no formal distinction between data and prior knowledge was stated. Hence, those examples correspond, in the terms of vocabulary established earlier in Table 2.1 to previous examples of integration in bioinformatics (presented in part 2.2.2).

**General kernel function.** Pavlidis et al. [PWCG01] performed gene function prediction in yeast by integration of microarray gene expression data with phylogenetic profiles. The goal was to make effective predictions of type: “Does gene X belong to functional class Y?”. Phylogenetic profiles were represented as vectors of features, and expression data were represented as ratio of measurements of gene expression in two different conditions. Machine learning approach utilizing kernel methods was used to perform requested predictions. Here, one specific kernel function was considered.

Three integration approaches were investigated. In early integration, vectors of data of all different types were concatenated to form single set of input vectors and the kernel function was calculated for this single set; the results were fed into machine learning module. In intermediate integration, the kernel function was computed separately for each data type, the results combined and fed into machine learning module. In late integration, the complete machine learning step was performed separately for each data type using specified kernel function, and resulting discriminant values were combined to produce final answer. The concept is presented on Figure 2.6.

**Specialized kernel functions and kernel combining.** Lanckriet et al. [LDBC+04] used similar experimental setup to integrate more data sources and explore more specific kernel functions for them. Here, the focus was put on identification of ribosomal and membrane proteins, via prediction of belonging to proper functional classes. Many additional data sources were included in the integration process, among others: protein homology information, amino acids hydrophobicity information, known protein-protein interaction information.
Here, kernel functions were chosen for specific data sources to better reflect their specificity, and results were combined using matrix representation of kernels \[\text{Vap98}\]. Individual kernels are briefly summarized in Table 2.3.

**Table 2.3: Examples of specialized biological kernels.**

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Data</th>
<th>Similarity Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{SW}$</td>
<td>protein sequences</td>
<td>Smith-Waterman pairwise sequence comparison</td>
</tr>
<tr>
<td>$K_B$</td>
<td>protein sequences</td>
<td>BLAST pairwise sequence comparison</td>
</tr>
<tr>
<td>$K_{Pfam}$</td>
<td>protein sequences</td>
<td>Hidden Markov Model pairwise expectation values from Pfam database</td>
</tr>
<tr>
<td>$K_{FFT}$</td>
<td>hydropathy profiles</td>
<td>Fast Fourier Transform applied pairwise to hydrophobicity patterns of protein sequences</td>
</tr>
<tr>
<td>$K_{LI}$</td>
<td>protein interactions</td>
<td>inner products of rows and columns from binary matrix of known interactions of yeast proteins</td>
</tr>
<tr>
<td>$K_D$</td>
<td>protein interactions</td>
<td>path length between graph nodes in protein interaction graph calculated using random walk</td>
</tr>
<tr>
<td>$K_E$</td>
<td>gene expression</td>
<td>Gaussian kernel function applied pairwise to gene expression profiles</td>
</tr>
<tr>
<td>$K_{RND}$</td>
<td>random numbers</td>
<td>linear kernel function applied to vectors of random numbers; included as control</td>
</tr>
</tbody>
</table>

### 2.2.3.2 Bayesian Network approach

Troyanskaya et al. \[TDO*03\] performed the same task of gene function prediction in yeast, as presented with earlier examples. Here, the prediction component was based on Bayesian probability interpretation; for details see for example \[HTF01\]. Briefly, the system answers the question: “What is the probability, based on evidence presented, that genes X and Y are functionally related”? For each pair of genes, a Bayesian network is created, that produces a posterior belief of this gene pair having a functional relationship. Here, the determination of belief starts from leaf “evidence” nodes, fed with the “similarity” of genes, based on co-expression or other experimental data, available as binary matrices. Later, the belief is “propagated”, based on conditional probabilities, up to the root node, where final belief is determined and scored. The nodes were created based on available data sources, evidence gathering techniques, and prior knowledge based on Gene Ontology.
The focus was put on integration of gene expression (GED) data with protein–protein integration (PPI) data and transcription factor binding sites (TFBS) information. The Bayesian network, presented on Figure 2.7, was modeled partially after Gene Ontology hierarchical structure and BioGRID\(^3\) concepts.

Here, the quantification step was performed only for GED, PPI and TFBS data sources. The graph-like structure of Gene Ontology was explicitly used in the machine learning approach.

![Figure 2.7: Example Bayesian network based on prior knowledge.](http://thebiogrid.org/)

### 2.3 Summary

In this chapter, we discussed the idea of data and knowledge and their integration in the context of life sciences.

Specifically, we traced differences between those general terms across three important research fields (see Table 2.1), and we established the common core of vocabulary that will be used throughout this thesis (see Figure 1.1).

Moreover, we also presented some representative examples of implementation of integration, approached from different angles by bioinformatics and machine learning communities, respectively.
Chapter 3

Current state of affairs

In this chapter, we present the current state of affairs in the topical domain of the thesis. That is, we follow most common approach used today for the problem of identifying gene signatures from microarray data with prior knowledge. First, in part 3.1 we present the general schema. Next, we focus on two important activities performed there. In part 3.2 we discuss the idea of variable selection. We focus especially on one method, $\ell_1 \ell_2$, that is used widely in this thesis, in part 3.3 In part 3.4 we present the idea of enrichment with prior knowledge. In this context, we conclude with two representative examples of the traditional methodology, namely Gene Set Enrichment Analysis (GSEA), presented in part 3.4.1 and WebGestalt, presented in part 3.4.2.

3.1 General process

A general schema of the process is presented on Figure 3.1. The idea arises from the fundamental problem of modern molecular genetics: to identify genes that are differentially expressed between two conditions, that are separable by their phenotype. Having a large number of measurements of gene expression, performed for two conditions, one can identify a set of genes that separates two states, also known as gene signature $[GST^\rightarrow 99]$.

In the terms of data and knowledge integration, the general process can be also formulated in more abstract way, as shown on Figure 3.2.

The first step is typically to produce a candidate gene list, using any technique that allows to identify differently expressed genes between two conditions. During this step, genes are treated independently, without any known relationships. Then, prior knowledge is used to enrich such list, in order to introduce biological meaning. The final enriched list of genes is considered both statistically and biologically meaningful.
Candidate gene list is a list of genes that are differentially expressed between two states, selected with some quantitative approach. It is constructed with some discovery procedure that mines gene expression data. Typically, a variable/feature selection method is employed for this task. Because of its importance, we discuss this topic in more detail in the next section.

In many cases, the final gene list is to be subsequently validated using some biological methodology, that is, the individual expression levels are physically confirmed, e.g. using qPCR approach [VVF08].

In addition, if there are any other significant findings that can characterize the final list of genes, they can be reported as well. For example, insights of biological roles of the final genes, as seen in Gene Ontology, may be reported in the form of the list of significant GO terms that are well–represented across the final gene list. Such information is available through mappings provided by Gene Ontology Consortium.

Figure 3.1: Schema of the general process of identifying gene signatures from microarray data, performed in traditional way. First, a list of candidate genes is devised from the expression data, as the list of the most differently expressed genes between two phenotypic conditions. Typically, it is produced with some ranking schema based on statistical testing, or with variable selection technique. Candidate list, while being statistically meaningful, is not usually biologically meaningful. Therefore, information from prior knowledge is introduced, and candidate list is modified (sometimes heavily) to incorporate biological meaning. A final list contains biologically meaningful genes, according to the current state of prior knowledge.

3.2 Variable selection

To build the candidate gene list, one needs to choose the most differentially expressed genes between two measured conditions. This problem can be cast into the idea of variable/feature selection.

In this setting, expression measurements are considered instances of features or variables, called “data points”, and the goal is to select the most relevant and informative variables, that is, in this context, these genes which expression is most differentiated. Many variable selection methods are proper machine learning (ML) methods, in the sense of performing
Figure 3.2: Schema of the general process of identifying gene signatures from microarray data, performed traditionally, formulated in more abstract way. Here, monolithic data (measurements for two conditions are merged into single data matrix) is mined during knowledge discovery process to obtain its most statistically meaningful features. Then, based on strategy dictated by prior knowledge, the primary results are reformulated to introduce biological meaning, and to produce true new knowledge.

a learning process. Here, the data points are paired with desired “outcome”, in our case the two different expression conditions. Then, a learning procedure explores any potential relationships between samples and outcomes, and constructs a predictive model that produces correct outcome for previously unseen data points. The predictive model, also called “predictor”, is typically modeled as a function of features \[HTF01\]. Nonetheless, many statistical scoring techniques, among them ranking approaches, can be also considered as “variable selection” methods. Variable selection methods are typically divided into the following groups: filters, wrappers and embedded \[GE03\].

**Filter methods** rank features according to their individual relevance. For ML, the ranking is performed independently of the learning process \[GST+99, WMC+01, FGE03\]. Any statistical testing procedure can be also considered as filter method. Here, having appropriate test statistics, one can state null hypothesis, test this hypothesis for each gene (here, having two measurements in different conditions), produce a \emph{p–value}, and decide what \emph{p–value} is considered meaningful. Typically, all genes are ranked according to their \emph{p–values}, and most significant ones are selected. The selection is very often based on constant cutoff value for \emph{p–value}. Being very fast, statistical tests are often employed in routine microarray data analysis to construct candidate gene lists \[STM+05, IWZS09\].

**Wrapper methods** are class of ML–specific techniques. They use separate instances of learning process to browse through subsets of features and score them according to their predictive power \[GWBV02, KJ97, FSMJ03\]. They can be further categorized into forward and backward approaches. In forward approach, the procedure starts with empty set and over time it incorporates variables decided relevant. In backward approach, the procedure starts with all variables and eliminates those decided irrelevant, until only relevant variables remain.

**Embedded methods** are deeply integrated within learning process itself and are specific to the technique being used \[BFSO84, FS97, SDC03, MH08\]. Here, the activity of selecting
is a crucial part of learning process itself.

An important group of machine learning methods are regularisation techniques, where predictive model is constructed by minimizing a well–defined objective function of variables to avoid overfitting that greatly decreases performance of the model [Tib94, ZH05].

In the case of machine learning–based methods, sometimes it is not desirable to generate the predictive model that depends on all features considered. This happens when the overall amount of variables is too great, a good example being microarray data. Instead, only the most significant features can be included into sparse predictive model [GE03].

As mentioned in part 1.2 we are using existing methods to uncover relationships within data, without modifying them. In this thesis, we specifically employ \( \ell_1 \ell_2 \), an embedded variable selection technique, to mine high–throughput data for new biological knowledge. This technique has also been used before for devising gene signatures from microarray data with success [FBM+09, SB11]. Note that \( \ell_1 \ell_2 \) was used already with the general schema presented on Figure 3.1 in this thesis we discuss a new application of this method. \( \ell_1 \ell_2 \) is presented in more detail in the next section.

3.3 Variable selection with \( \ell_1 \ell_2 \)

Foundations

The \( \ell_1 \ell_2 \) [BMRV08, DMDVR09, DMMTV09] is an embedded variable selection method based on supervised learning and regularization principles. In this context, the main task can be described as follows. We consider the collection of \( n \) data points, each represented as \( p \)-dimensional vector \( x \). Each data point is associated to a label \( y \in \{-1, +1\} \), and thus to a class (e.g. “patient” or “control”). The data set is formed by \( n \times p \) matrix \( X \in \mathbb{R}^{n \times p} \), where \( p \gg n \), and is accompanied by the label vector \( Y \in \mathbb{R}^{n} \). We consider a linear model \( f(x) = x \cdot \beta \). The goal is to find a sparse approximation of \( \beta \) such that \( \text{sign}(x \cdot \beta) \sim y \) holds. Here, “sparse” means that many of the coefficients in \( \beta \) are equal to zero.

Regarding the variable selection process, the method follows the principles of elastic net regularized approach [ZH05], where all correlated variables influence the predictive model (which is not always the case [HTF01]), yet the sparsity of the final model is tightly controlled.

The task is solved in two steps. First, the minimization of objective function dubbed naïve elastic net functional [DMDVR09] is performed:

\[
\beta^* = \arg \min_{\beta} \left\{ \frac{1}{n} \|Y - X\beta\|^2_2 + \mu \|\beta\|^2_2 + \tau \|\beta\|^1_1 \right\}
\]
and the following sparse solution is found:

\[ \beta^* = (\ldots, 0, \beta_{m_1}, 0, \ldots, 0, \beta_{m_k}, 0, \ldots), \quad \beta_{m_i} \neq 0, i = 1, \ldots, k; \quad k < p \]

This minimization is performed for small constant value of parameter \( \mu \).

Next, another minimization, of regularized least squares objective function, is performed, and the following solution is found:

\[
\beta = \arg\min_{\beta} \left\{ \frac{1}{n} \| Y - X^* \beta^* \|_2^2 + \lambda \| \beta^* \|_2^2 \right\}
\]

where \( X^* = [X]_{m_i}, i = 1, \ldots, k \), that is, the matrix \( X \) is restricted only to columns that correspond to non-zero coefficients \( \beta_{m_1}, \ldots, \beta_{m_k} \) of the model \( \beta^* \).

Vector \( \beta \) is the final solution of the main task, and \( \text{sign}(x \cdot \beta) \sim y \) holds. To complete variable selection procedure, one can retrieve variables associated with non-zero elements \( \beta_{m_1}, \ldots, \beta_{m_k} \) of \( \beta \) vector.

**Model selection**

The proper execution of the reasoning described before depends, at one point, on the optimal values of two parameters, \( \lambda \) and \( \tau \). Parameter \( \mu \) is always fixed \textit{a priori}; it governs the expected amount of correlation between variables.

The process of finding optimal values of model parameters is called model selection, and it can be performed in many ways \[\text{HTF01}\]. Here, to find optimal values \( \lambda^* \) and \( \tau^* \), a series of cross-validation loops is performed. The details of the whole procedure were explained elsewhere \[\text{BMRV08, DMDVR09, DMMTV09}\]. Here, we would like to mention one of the important principles, technically utilized in the prototype implementation, that is external and internal splits.

During model selection, first, original data set \( X \) is split \( B \) times into training and test part, at random, forming external splits. Next, each training part is split further \( K \) times into training and test part, also at random, forming internal splits. Thus, each internal split can be indexed as \((i_b, j_k), b = 1, \ldots, B, k = 1, \ldots, K\). This principle has been actively used in the design of computational procedures performed by the prototype implementation; this topic is discussed further in parts 4.2.4 and 4.2.5.
3.4 Enrichment with prior knowledge

The most common way to introduce prior knowledge into the general schema presented on Figure 3.1 and Figure 3.2 can be described as follows. After construction of the candidate gene list, member genes are grouped into gene sets, according to specified criteria. The criteria are defined based on prior knowledge. For example, one can create sets of genes common to specific physiological conditions. A very common case here is to group genes according to their association with Gene Ontology (GO) terms. Next, more specialized scores are calculated within gene sets for all member genes, to determine if the gene set is well-represented among candidate gene list.

For example, if members of gene set associated with specific Gene Ontology term are found to be meaningful in candidate gene list, it can be stated that the biological activity, referred by GO term, is particularly affected by differential expression. Therefore, this can signal that such activity is connected with the different phenotypic conditions that are compared. Such connection is a valid biological hypothesis that can be further validated in vivo. This reasoning is valid for any criteria used to construct gene sets.

One simple and popular technique of scoring is over-representation [IWZS09], where for each gene set, the number of member genes is paired with the number of non-member genes. The significance of this ratio can be assessed with suitable distribution, such as hypergeometric or binomial. The case that uses association with Gene Ontology terms as grouping criteria for genes, is one of the most popular tools used in gene expression analysis. It is often referred to as “term enrichment” [1].

This approach depends heavily on the cutoff value for p-value statistics used during construction of candidate list. For example, having a functional group of genes that is confirmed to behave differently between two conditions, if very few member genes “survive” the specified cutoff, then the group may not be pointed as meaningful at all [MLE+03].

Another technique of scoring, dubbed aggregated score approach [IWZS09], relies on more sophisticated treating of gene sets, where the final score for each set is based on individual scores calculated for member genes; such aggregated scores can be calculated in many ways [PLN02] [KV05]. This approach has no cutoff limitation as the previous one, therefore it has received considerable attention of researchers looking for more prior-knowledge-oriented approach to gene expression analysis [IWZS09] [AS09].

We focus here on two representative examples. First, we discuss one of the most popular aggregated score methods implemented, Gene Set Enrichment Analysis (GSEA) [STM+05]. Next, we present an example implementation of over-representation approach, in the form of popular Web portal WebGestalt [ZKS05].

3.4.1 Gene Set Enrichment Analysis

The Gene Set Enrichment Analysis (GSEA) \cite{MLE03, STM05} technique is performed for candidate gene list \( L \) that is sorted according to some ranking procedure that uses specific metric \( r \). A priori defined gene set \( S \) contains genes grouped together according to some criteria that are based on prior knowledge.

The main task is to determine whether the members of \( S \) are randomly distributed throughout \( L \), or can be primarily found at the top or bottom of \( L \). In latter case, we conclude the gene set \( S \) reflects the biological activity that is truly important for phenotypic difference being studied. The idea is presented on Figure 3.3.

![Figure 3.3](image)

Figure 3.3: A general idea of Gene Set Enrichment Analysis (GSEA) approach. Meaningful gene set \( S \) is scored against candidate gene list \( L \), that is sorted according to some ranking procedure. If members of gene set \( S \) are not “randomly” present in \( L \), but they are found either near top or bottom in close proximity to each other, the gene set \( S \) is considered meaningful for the phenotypic difference being studied.

The enrichment score \( ES \) reflects the degree of representation of \( S \) either at top or bottom of \( L \). It is calculated by walking down the list \( L \), increasing a running–sum statistic when a gene in \( S \) is encountered or decreasing it when gene not in \( S \) is encountered. The magnitude of the increment depends on the metric used in ranking. The enrichment score is the maximum deviation from zero encountered in the random walk.

More specifically, the method performs the following calculations. Having candidate gene list \( L = (g_1, \ldots, g_N) \) of length \( N \), ordered according to metric \( r \), an accompanying list of ranking values \( L_r = (r_1, \ldots, r_N) \), and meaningful gene set \( S = (g_1, \ldots, g_S) \) of length \( N_S \), the following calculations are performed, for each position \( i \) in list \( L \):
\[ P_{hit}(S, i) = \sum_{g_i \in S, j \leq i} \frac{\|r_j\|^p}{N_R}, \quad N_R = \sum_{g_i \in S} \|r_j\|^p \]

\[ P_{miss}(S, i) = \sum_{g_i \notin S, j \leq i} \frac{1}{N - N_S} \]

\( ES \) is the maximum deviation from zero of \( P_{hit} - P_{miss} \); high values of \( ES \) are considered meaningful. Next, the significance of \( ES \) obtained is checked. In original proposition \([MLE+03, STM+05]\), the significance is assessed by comparison with \( ES_{NULL} \) score, obtained by randomization of phenotypic states (since the \( ES \) was calculated for given assignment of phenotypic state to each gene), and corrections for multiple hypothesis testing are applied.

The method is one of the most widely used techniques that perform the enrichment of candidate gene list \([HYH+11]\).

### 3.4.2 WebGestalt

WebGestalt (WEB-based GEne SeT AnaLysis Toolkit) \([ZKS05, DPZ10]\) is an integrated data mining system for the management, information retrieval, organization, visualization and statistical analysis of large sets of genes. It has the form of Web portal built on top of specialized gene-centric relational database, GeneKeyDB \([KPB+05]\).

The system provides the following functionalities to the user, among others: the management of the organization of gene set(s) being studied, including a choice of many visualization options; providing Boolean operators on gene sets; retrieving substantial amount of information about each member gene; and assistance in choosing appropriate statistical tests. It recognizes many grouping criteria for genes, such as mapping to Gene Ontology concepts, membership in KEGG\(^2\) pathways, membership in Pfam\(^3\) protein family etc. Each gene set built according to some grouping criterion is dubbed a category.

WebGestalt accepts candidate gene list, and provides statistical methods to check for over-representation of categories, i.e. gene sets associated with individual Gene Ontology terms or KEGG pathways, as discussed in part 3.4.1. Note that to implement statistical testing for over-representation, one needs a reference gene set that provide a “universe of all genes” in particular context \([ELL97]\). For example, over-representation can be checked against

---

\(^2\)Kyoto Encyclopedia of Genes and Genomes

\(^3\)http://pfam.sanger.ac.uk/
the whole species genome known to date, or just against genes presented on particular microarray platform, where the number of genes may be smaller.

In order to check the over-representation, WebGestalt uses the following procedure. Having candidate gene list $L$ of length $n$, and reference gene list $R$ of length $m$, a category set $C$ is mapped to $L$ and $R$. Having $k$ genes from $C$ in $L$, and $j$ genes from $C$ in $R$, the expected value of $k$ is obtained: $k_e = (n/m) \times j$. If $k > k_e$, the category $C$ is declared meaningful or enriched, with the ratio of enrichment $r = k/k_e$.

The significance of enrichment for category $C$ is then assessed with hypergeometric test, if $L$ is drawn directly from $R$ [ELL97], where the following $p$-value is calculated:

$$P_h = \sum_{i=k}^{n} \frac{{m-j \choose n-i} {j \choose i} {m \choose n}}{{m \choose n}}$$

and Fisher’s exact test, respectively, if $L$ and $R$ are two independent gene sets, where the following $p$-value is calculated:

$$P_f = \sum_{i=k}^{n} \frac{{n \choose i} {m \choose j + k - i} {m + n \choose j + k}}$$

Finally, after multiple test adjustment is performed [ELL97], resulting $p$-values are reported. The user can customize many aspects of tests, such as significance level, adjustment procedure, minimum cardinality for significant category etc.

The primary output is the minimal subgraph [Wil10] of Directed Acyclic Graph (DAG) of Gene Ontology, with enriched terms highlighted, as shown on Figure 3.4. Our approach uses very similar graphical representation; this topic is discussed further in part 4.2.5.5.

WebGestalt is a representative of the wide class of tools that perform “term enrichment” [DSH03, MHK05, WFDC+10]. It was used with earlier applications of $\ell_1\ell_2$ to mine microarray data for gene signatures and biological knowledge [FBM+09, FCB+10, BJV+11, SB11].

### 3.5 Summary

In this chapter, we presented the current state of affairs in the topical domain of the thesis, that is, the most common approach for the problem of identifying gene signatures from microarray data with prior knowledge. After presenting general schema in part 3.1, we focused on two important activities performed therein. Variable selection is presented
in part 3.2, here, we focused specially on one method, namely $\ell_1\ell_2$, used widely in this thesis, discussing it in part 3.3. Enrichment with prior knowledge is presented in part 3.4; here, we have shown two representative examples that are widely used, namely Gene Set Enrichment Analysis (GSEA) in part 3.4.1 and WebGestalt in part 3.4.2.
Chapter 4

Knowledge Driven Variable Selection

In this chapter, we focus on the idea and prototype implementation of Knowledge Driven Variable Selection (KDVS) concept.

Part 4.1 presents the ideas behind KDVS. Part 4.2 presents the details of prototype implementation of KDVS concept as pipeline of Python\(^1\) and R\(^2\) applications, published in [ZBV11].

4.1 Prior knowledge as a guide

4.1.1 General idea

Finding alternative ways to combine prior biological knowledge with plain measurement data is still an open research area [Phi08, FSB08]. One of the possible ways, presented in this thesis, is to use the prior knowledge as a guide when applying machine learning procedures, or in general any knowledge discovery technique, to plain numerical data. Referring to the terminology presented in Table 2.2, prior knowledge is used “a priori”.

In this approach, the structure of the prior knowledge, combined with a definition of the problem, devises a strategy to transform original plain data set into more suitable form for applying convenient machine learning procedure. The terms used here are general enough to be adapted to specific problems in question. The concept is illustrated on Figure 4.1.

For example, considering microarray experiment with “disease vs healthy” experimental setup, the following reasoning can be devised. Gene expression measurements are identified

\(^{1}\)http://python.org/
\(^{2}\)http://www.r-project.org/
on physical chips through probesets, that can be associated with gene products, using the information provided by chip manufacturer. Taking into account prior knowledge offered by Gene Ontology, it is possible to transform original gene expression measurements, in the form of big gene expression data matrix, into smaller submatrices associated with Gene Ontology terms, using mapping measurement→gene product→GO term. Then, machine learning procedure may be applied to those smaller submatrices, and partial results may be collected and combined together. Considering smaller size of submatrices, the curse of dimensionality is relaxed, and wider repertoire of methods may be used [HTF01].

In this case, the “strategy” is simply the transforming of primary data set, and “more suitable form” refers to smaller submatrices with less features that are more adapted for machine learning. Finally, “convenient” refers to more flexibility in choosing machine learning procedure, being less bound by curse of dimensionality. The terms presented in the reasoning are summarized in Table 4.1.

<table>
<thead>
<tr>
<th>General Concept</th>
<th>KDVS Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>Transformation of gene expression data matrix into submatrices</td>
</tr>
<tr>
<td>More suitable data</td>
<td>Smaller data sets, curse of dimensionality lifted</td>
</tr>
<tr>
<td>Discovery method</td>
<td>ℓ₁/ℓ₂/other</td>
</tr>
</tbody>
</table>

Table 4.1: Practical implementation of prior knowledge as a guide concept in KDVS.

4.1.2 Features of the approach

4.1.2.1 Usage of prior knowledge

The power of machine learning methods lies in their ability to discover meaningful relationships in the data that are too subtle for human deduction to notice. However, this property can also be the greatest weakness, while being also the greatest strength. While the relation-
ships discovered may be meaningful from statistical viewpoint, they may be meaningless from contextual viewpoint, in this case they may not be biologically valid [VDD11].

One of the possible solutions is to use prior knowledge “a posteriori” (see Table 2.2) to verify and/or enrich the results from machine learning procedure; such method, discussed in more detail in chapter [3] is used for example by GSEA [STM+05]. Our approach explores another possibility, namely to use prior knowledge “a priori”, as presented on Figure 4.2. Taking into account the strategy concept (see Figure 4.1), more detailed comparison, illustrated on Figure 4.3 may be devised.

Figure 4.2: Comparison of usage of prior knowledge in the context of knowledge discovery between traditional approach (“Enrichment”) and KDVS.

Figure 4.3: More detailed comparison of the usage of prior knowledge in the context of knowledge discovery, between traditional approach (up) and KDVS (down).
4.1.2.2 Lifting the curse of dimensionality

Typical “disease vs control” microarray experiment involves millions of single measurements dispersed across biological samples. Here, single chip can measure tens of thousands of different features, and data from all samples are pooled into single data matrix. In addition, the number of samples ($N$) is far smaller than number of features on a chip ($P$), and pooling does not compensate that (see Figure 4.4). As a result, we operate on single atomic data matrix of dimension $N \times P$, where $N \ll P$. Many traditional machine learning approaches are not designed to work with such data sets [HTF09].

![Figure 4.4: Curse of dimensionality in microarray experiments. Gene Expression Data Matrix (GEDM) is formed from gene expression measurements performed on individual biological samples. In this case, typically, the number of biological samples is very low, compared to the number of features measured. This may heavily affect the outcome of classic knowledge discovery approaches.](image)

By breaking the monolithic data sets into smaller ones in smart way, as depicted on Figure 4.5, more elastic approach to the problem becomes possible, since broader repertoire of classical algorithms may be applied.

4.1.2.3 Exploration of the structure of prior knowledge

If prior knowledge displays some form of hierarchy in its structure, such property can be explored on many levels [FPSS96]. Not only it can be used to influence machine learning procedure, as shown in part 4.1.2.2, it can also organize already meaningful output to be more readable to non–statisticians. The idea is presented on Figure 4.6.
4.1.2.4 Harnessing parallel computation

Lifting the curse of dimensionality, described in part 4.1.2.2, is based on generation of particular subsets of original data set. That increases computational requirements, since knowledge discovery techniques must be executed multiple times.

To minimize impact of this inconvenience, high–performance computing (HPC) paradigm may be used [Kum02]; the idea is illustrated on Figure 4.7. In the prototype implementation, we used a micro–environment of few machines, that was organized over local network in peer–to–peer fashion.

4.1.3 Difficulties in the approach

4.1.3.1 Nature of prior knowledge

In the prototype implementation, we explore the hierarchy of Gene Ontology vocabulary, where terms are organized in Directed Acyclic Graph (DAG), depending on how one term is related to another.

It is very tempting to treat this DAG as graph in the mathematical sense [Wil10], and to use directly the wealth of graph algorithms for analysis. In reality, however, one must be aware of the more complicated nature of this knowledge [GOD, CSC07]. For example, Gene Ontology maintains a set of relations between GO terms and gene products (sequence...
that has biological functionality [GOD]). One may expect that, if some ancestor term has associated some child terms, in the sense of semantic meaning, then gene annotated to child node at one point, will be also annotated to each parent node automatically. Unfortunately, this is very often not the case [GOS]. Therefore, from statistical point of view, when one expects some level of correlation within prior knowledge [HTF01], for it to offer meaningful ground truth, this is not desirable property.

One of the solutions is to recover this property manually, and enrich straightforward mappings GO terms ↔ raw measurements by providing complementary features for every GO term, based on its parenthood history. This approach is currently being investigated.

4.1.3.2 Dependency from external information sources

To perform all necessary activities, current KDVS implementation needs some external data, in order: raw microarray data sets for producing uniform gene expression data matrix (GEDM), necessary mappings for producing submatrices of GEDM, and a structure of prior knowledge, in our case Gene Ontology term hierarchy, that can be queried in effective way.

With the abundance of bioinformatic repositories that provide content on–line, it is theoretically possible to create single monolithic application that downloads all data dynamically and performs complete KDVS task, as seen in Figure [DB09]. However, during devising prototype implementation, it was revealed that such approach is unrealistic, for the following reasons.

First, dependency on remote querying introduces unnecessary latency that can contribute significantly to the overall execution time. Moreover, when remote resources become un-
available, it is not possible to conduct experiments at all, which is not acceptable.

Second, knowledge discovery may be performed in parallel to be effective. In that case, it is desirable to provide all data beforehand, where proper data distribution strategy may be devised that optimize both space and time requirements; this is of particular importance for parallel processing [Kum02].

Third, any structural information shall be available in the form that enables fast querying. Here, it is desirable to obtain uniform and compact representation that is stored locally.

**Raw microarray data** Due to sheer size, in prototype implementation manual download of raw microarray data sets is required. After that, they are processed to form GEDM matrix.

In this case, there is a possibility of providing capability of automatic download and processing in the future, since such activity may be performed on single machine, and is effectively isolated from knowledge discovery (see Figure 1.1).

**Mappings GO terms ↔ raw measurements** To produce submatrices of GEDM, one must obtain the mapping between GO terms and microarray probesets. Such mapping is not available with original data set and must be acquired elsewhere. We require manual download of proper annotation file, either directly from manufacturer or any repository that offers such information.
One may notice that such mappings may be obtained only once, for each microarray type. In prototype implementation, however, such mappings are devised every time anew for each experiment. This approach, while not straightforward, has many advantages. First, one does not need to provide any management of pre–computed mappings. Second, one is independent of the changes in annotations themselves, and such changes do occur (e.g. Affymetrix introduced versioning of its annotations). Third, it enables using virtually any microarray platform, as long as proper annotation file is presented. Fourth, in the long run, it enables going beyond microarrays, and incorporating another high–throughput kind of data, again as long as proper annotations are available, either directly from manufacturer or assembled dynamically by researchers. Overall, this approach allowed to reach local balance of time and space requirements.

The mappings in question can be collected in many other ways, for example through combination of the following mappings: probeset → gene, when microarray probesets are matched against current genome sequence [LYWM05, NVE+10], and GO term → gene, available from GO Consortium [ABB+00]. When in both cases reciprocal mappings are also produced, final mapping GO term ↔ probeset may be assembled. In prototype implementation, however, we followed mappings that are already available within microarray annotations. This approach, while convenient, creates a possibility of using outdated coverage of genome with microarray probesets, since genome sequence is being annotated constantly [Bre08]. This issue, however, must be resolved by presentation of up–to–date annotations and is beyond control of KDVS.

**Gene Ontology hierarchy** To obtain structure of prior knowledge, here Gene Ontology hierarchy, we use single RDF-XML file, available from Gene Ontology consortium [ABB+00] 3. This file is redistributed with KDVS prototype. For each experiment, it is parsed anew and information of individual terms, as well as hierarchy of terms, is obtained. All necessary querying is performed on this information.

One may notice that it is possible to obtain such information also from local copy of Gene Ontology relational database 4 or in the form of OBO file. Our approach, while not straightforward, has many advantages. First, informational content of GO database is greatly superficial for our needs. Second, one is independent from additional management of database server to access either remote or local database instance. Third, RDF-XML format can be parsed effectively without any external dependencies, being standard XML dialect. Overall, again we saw reaching local balance of space and time requirements.

---

4 http://www.geneontology.org/GO.database.shtml
4.1.3.3 Hardware and software limitations

The main aim of current implementation was to produce working prototype of KDVS concept, and it was observed that devising fully engineered structure, in the sense of best principles of software engineering \[\text{Som95}\], was unfeasible at such early stage of methodology development. Thus, several decisions were made that heavily influenced the hardware and software requirements. Moreover, the complexity of prototype implementation itself, in the meantime, excluded the possibility of producing straightforward “bag of scripts” artifacts (e.g. in Matlab \[^5\]). Finally, some compromises were established that affect the current implementation in both positive and negative ways.

This part goes into the level of technical details. For full understanding of all the aspects discussed, the reader is referred to part 4.2.

Data pre-processing  To provide all necessary data for early integration (that is, to generate all proper mappings), KDVS prototype uses both pre-distributed and user-provided flat files.

It is possible, however, to construct local data warehouse, to import the data into it, and subsequently to access it with provided API. Such concept, while being feasible from software engineering point of view \[\text{Som95}\], was abandoned due to the unnecessary maintenance of relational database engine during every experiment. Besides, the idea was to provide not another data warehouse with knowledge discovery functionality, exposed as static on-line resource, but to make knowledge discovery a main focus of the implementation, while early integration provides background information in timely and efficient manner, without additional latencies. Such approach can be extended into high-performance computing (HPC) domain with minimal changes.

At the end, a compromise was set to use embedded relational database engine, to manage early integration, that lacks many features, such as client-server architecture. This way, however, the maintenance cost for users with low computer skills is reduced to minimum, while providing enough elasticity to harness the power of relational querying and processing.

Knowledge discovery  In general, data pre-processing phase (see Figure 1.1) is not as demanding for computational resources as knowledge discovery. We noticed, however, that immediate preparatory stage that precedes knowledge discovery phase, can be time and memory consuming, depending on how parallel resources are subsequently harnessed, and how data is to be distributed afterwards.

[^5]: http://www.matlab.com
For example, we decided not to generate explicitly any binary formats, instead we encode their textual equivalents and compress them, to save large portion of transmission capacity in a local network. Such encoding, however, has introduced quite heavy memory requirements for its performance, being an estimated minimum of 4GB of memory. This encoding is to be performed once, before issuing any parallel jobs. Later on, every subordinated worker machine reconstitutes proper binary objects and performs specific calculations over them, during knowledge discovery phase.

It is physically possible to run the current implementation on a single machine. However, for smooth performance, it is highly recommended to use any form of parallel computation available to execute jobs on “worker” machines; all preparatory activities can be performed on single “master” machine. In Table 4.2 we briefly summarized approximate memory requirements for the smooth run of the prototype implementation, for all machine types involved.

However, final memory requirements depend heavily on the specificity of the experiment protocol, such as number of individual measurements performed by high–throughput (HT) platform, number of biological samples processed during the experiment, numerical precision of measurement etc.

<table>
<thead>
<tr>
<th>Machine Type</th>
<th>RAM required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master (main) machine</td>
<td>4+ GB</td>
</tr>
<tr>
<td>Worker machine</td>
<td>2+ GB</td>
</tr>
</tbody>
</table>

Table 4.2: Approximate memory requirements for smooth performance of prototype KDVS implementation. The real memory requirements depend on many more factors, such as the microarray platform type, number of biological samples, knowledge discovery method used, etc.

Non–uniform implementation artifacts The current proof–of–concept implementation was produced as a mixture of Python and R code, that requires some intermediate adaptations between the two environments. We tried to minimize impact of this by extensive usage of flat files and common data sinks, where data is collected together as much as possible, regarding its further usage and storage.

It is possible, however, to implement the KDVS concept in consistent way, unifying technology (e.g. using only Python), storage method (e.g. using industry–standard relational database engine, such as Oracle 6 or MySQL 7), and generalized access to parallel resources (e.g. with Apache Hadoop 8 or Twisted 9). Such advanced implementation goes beyond the scope of this thesis.

---

6http://www.oracle.com
7http://www.mysql.com
8http://hadoop.apache.org
9http://twistedmatrix.com
4.1.3.4 Developing KDVS API

While devising prototype implementation, it has become clear that creating single monolithic application is not desirable, and the concept of Application Programming Interface (API) was introduced. Here, data and computational resources management makes core of KDVS, while any application that uses them is implemented on top of it. Application is not understood here as physical code, but rather as an implementation of particular functionality, realized with any programming language, on any back-end, in any number of software/hardware artifacts. The idea is presented on Figure 4.8.

Figure 4.8: Concept of KDVS API. Data management governs processing of input data, storage and retrieval. Computational resources management governs interactions with parallel resources and execution environment, together with data management. Applications access those functionalities through API.

Such separation allows independent development of core components and applications. It is important for two reasons. First, the landscape of biological data management is very dynamic; new technologies are being developed constantly and new data formats are brought with them. Second, well established approaches, in terms of implementation of knowledge discovery methods, are often devised by researchers that do not work on daily basis with biological data.

Thus, by providing API layer over core components, it is possible to develop new data integration solutions underneath, and to expose core functionality for numerical-oriented research communities.
4.2 Implementation

This section describes prototype implementation of KDVS concept in more detail, as published in [ZBV11]. Although some technical remarks have already been introduced in parts 4.1.3.2 and 4.1.3.3, we provide more rigorous description here.

First, we zoom out the view to present general prototype schema in part 4.2.1. Next, we present more details about input data in part 4.2.2. Next, we describe the core of the system that processes input data, performs local data integration, and prepares knowledge discovery, in part 4.2.3. We focused more on knowledge discovery phase in part 4.2.4. Next, we present currently last stage of the pipeline, namely various post-processing activities, in part 4.2.5. Finally, we comment on possible extension of prototype implementation into full-fledged framework in part 4.2.6.

4.2.1 Overview

General schema of KDVS prototype implementation is presented on Figure 4.9.

According to the concepts of core, API and user application introduced in part 4.1.3.4 (see also Figure 4.8), KDVS core corresponds to the first two terms, while statistical framework is composed of two user applications, interacting with common data storage, namely information ensemble, through API.

KDVS core performs all data management operations and provides skeletal execution environment for user applications. Data management part consists of several processors that work with specific type of data. It also maintains information ensemble, a shared data storage that can be used across user applications. Skeletal execution environment provides convenient functionalities for developers of user applications, such as atomic execution units, variable sharing, interface to parallel computational resources etc.

Within statistical framework, the knowledge discovery concept, based on the reasoning presented in part 4.1, is implemented. Namely, primary monolithic data set is transformed into submatrices, proper variable selection techniques are applied to them, partial results are collected and post-processed to obtain the final output.

4.2.2 Input data

Raw data Raw data is a series of gene expression measurements, performed on microarrays, for specific biological tissue sample. Values of such measurements may be stored in native format (such as CHP for Affymetrix) or as Delimiter Separated Values (DSV) file. Gene expression data are available from many repositories such as Gene Expression Om-
nibus (GEO) [EDL02] or Array Express (AE) [PSK+11]. Raw data needs to be further pre-processed by raw data processor.

Annotation data In all experiments performed so far, we used microarray annotations from Gene Expression Omnibus [10] or directly from Affymetrix [11]. Annotations are available as DSV files, and are fed directly into database processor, since in this case no pre-processing is needed.

Prior knowledge data In KDVS prototype implementation we explore Gene Ontology term hierarchy and basic term data. This information is provided by RDF-XML [12] file,

---

available either as termdb or assocdb builds, from GO Consortium\textsuperscript{13}. The termdb build contains GO terms hierarchy, terms definitions and mappings to other databases. The assocdb build includes termdb and also contains manual and most of the annotations compiled for terms. Currently termdb RDF-XML build is used, being much smaller and informative enough. A copy of “termdb” RDF-XML file is redistributed with KDVS prototype. This file is processed by \textit{GO processor}.

4.2.3 KDVS core

4.2.3.1 Raw data processor

Raw data processor (Figure 4.10) is a series of R\textsuperscript{14} scripts that perform data normalization and quality control, using algorithms implemented in R environment BioConductor \textcite{GCBea04, BWS09}. The choice of specific algorithm depends on the type of microarray platform and raw data quality. This processor accepts raw data as input data. Output data is a gene expression data matrix (GEDM), stored in a Delimited Separated Values (DSV) file.

Figure 4.10: Schema of raw data processor. Raw data are normalized and quality control is performed with R tools from BioConductor package. Final output is gene expression data matrix (GEDM) in DSV file.

4.2.3.2 GO processor

Gene Ontology \textcite{ABB+00} processor (Figure 4.11) accepts RDF-XML file as input data and decomposes encoded DAG structure into set of uniform pairwise relations \textit{ancestor term} \rightarrow \textit{child term}, trading some information loss (namely, relation types and explicit multiple parentage) for simplicity of usage. It also identifies synonymic GO terms, at the level of conformity of GO term identifiers (not semantic similarity), as well as discards obsolete GO terms and their relations. Moreover, it also provides basic data for individual terms, such as symbol and descriptive name. The output data is in the form of a serialized data

\textsuperscript{13}http://www.geneontology.org/GO.downloads.ontology.shtml
\textsuperscript{14}http://www.r-project.org/
structure. It is being used when some information about GO terms is not available in specific microarray annotations, such as term hierarchy or synonymy/obsolescence.

The reason for not using OBO format\textsuperscript{15} was that at the time of implementation, there is still no standard Python parser for OBO files, only Java and Perl solutions are provided\textsuperscript{16}. There exist few Python implementations of OBO parsers, either standalone\textsuperscript{17}, or as part of software libraries\textsuperscript{18} \textsuperscript{19} \textsuperscript{20}; none of them, however, has been so far recognized by Gene Ontology Consortium\textsuperscript{21}.

4.2.3.3 Database processor

The main role of database processor (Figure 4.12) is to build a dynamic information ensemble from all needed information, perform local data integration, and produce input data for knowledge discovery phase (see Figure 4.13).

For each experiment, a local relational database is created, that contains all data from other processors, as well as additional derived data, needed for completion of the experiment. This experiment database is the central component of information ensemble.

To maintain experiment database, we employ SQLite\textsuperscript{22}, the default Python SQL engine. Being an embedded engine with small memory footprint, SQLite implements a subset of standard SQL dialect, sufficient to carry basic data manipulation needed by KDVS database processor. To minimize maintenance efforts and to provide elasticity, SQLite

\textsuperscript{15}http://www.geneontology.org/GO.format.obo-1.2.shtml
\textsuperscript{16}http://www.geneontology.org/GO.tools.shtml
\textsuperscript{17}https://launchpad.net/pawnets
\textsuperscript{18}http://www.paccanarolab.org/software/gfam/index.html
\textsuperscript{19}https://github.com/tanghaibao/goatools/
\textsuperscript{20}http://www.grenoble.prabi.fr/trac/OBITools
\textsuperscript{21}http://www.geneontology.org/GO.tools_by_type.software.shtml
\textsuperscript{22}http://www.sqlite.org/
stores databases in single files. One database file can hold many tables and other structures (views, triggers, etc.) associated with them.

Regarding experiment database activities, first all DSV files are loaded into corresponding raw tables. The processor follows the structure of DSV file header to properly create the table and copies corresponding data into it. Next, local data integration is performed and derived tables, containing combined pieces of information from raw tables, are created.

Currently, the following raw tables are created: GEDM that contains expression measurements from gene expression data matrix, ANNO that contains microarray-specific annotations (obtained from public repository or manufacturer directly as needed), HGNC that contains gene naming information (obtained from HUGO Gene Nomenclature Committee), and LABELS that contains phenotype information for supervised analysis in the knowledge discovery phase (may not be required, for example in unsupervised learning problems). Each table row is identified by its key, unique within the table, namely probe-set name for GEDM and ANNO tables, gene symbol for HGNC, and biological sample name for LABELS. In the prototype implementation, KDVS database processor creates two specific derived tables.

The term2probeset derived table contains all the basic information associated with probeset names regarding GO terms, such as ID, description, available evidence codes etc. Here, we discard control probesets such as AFFX probesets in Affymetrix platforms. Moreover, only GO terms that have any probeset associated according to ANNO data, are considered. In other words, we discard empty submatrices. This table was constructed to obtain the mapping GO term $\rightarrow$ probesets, more useful than the one available by default in ANNO data, namely probeset $\rightarrow$ GO terms.

The probeset2gene derived table contains associations between probeset names and their respective gene products, or sequence references. Here, if the original information from

\footnote{http://www.genenames.org/}
microarray–specific ANNO data is confirmed in HGNC data, gene name is resolved unambiguously; otherwise, the original GenBank accession number, provided by microarray manufacturer, is used to pinpoint any external annotation for the probeset name in question. This table is constructed to control sequence references coming from ANNO data, e.g. to filter out outdated or non–official gene symbols, known pseudo–genes, to resolve obsolete GenBank sequence records etc.

All tables managed by database processor are summarized on Figure 4.13.

4.2.3.4 Information ensemble

Information ensemble is an ad hoc mash–up of experiment database and additional file objects. Experiment database contains all the information, both primal and derived, needed for performing computational experiment. Additional file objects contain information needed for knowledge discovery phase, as well as information that allows the reproducibility of computational experiment. The information ensemble functions as common data sink for all experiment data.
Based on the information gathered in information ensemble, the knowledge discovery phase is prepared and launched.

4.2.3.5 Skeletal execution environment

The idea of skeletal execution environment (Figure 4.14) is to provide simple yet powerful uniform environment for execution of code produced in disparate conditions, by different methodologies, in different communities.

Here, main execution unit is an action, essentially being any Python function that can contain any valid Python code. The action is not intended to run autonomously; it must be registered within execution environment. The environment can then execute all registered actions; execution order is simply the order of registration. Environment also manages error handling, if necessary.

Actions are not operating in void; execution environment offers shared variables storage to use. Here, any action can store any variable at any time, so for example, subsequent action can pick it up and use it. As actions are by assumption stateless, the shared variables storage allows state transfer between actions, and thus implementation of complicated
algorithms and processes is possible.

Execution environment can interface with resources offering parallel computation. In current implementation, for example, there exists an environment that interacts with the in-house middleware, PPlus (presented in part [A.3]), that manages parallel micro-environment of machines over simple local network. Actions registered within this environment can create their own jobs to be run in parallel, and send them through the environment for parallel execution. The difference between actions and parallel jobs (also called tasks) is illustrated on Figure 4.15.

Figure 4.15: Actions are not synonymous with parallel jobs. Individual actions may access parallel resources, share variables related to parallel activity within execution environment, and produce individual parallel jobs (tasks).

Two central Python artifacts, being the two “main applications” of the KDVS prototype implementation, are composed as a set of actions that access KDVS Core through API, executed through specified execution environment.

### 4.2.4 Knowledge discovery

Knowledge discovery can be conceptually divided into two parts: preparatory phase, when data sets are prepared, and execution phase, when knowledge discovery individual methods are executed.
4.2.4.1 Preparatory phase

In this phase, proper data sets are prepared and proper feature selection procedures are assigned to them.

One of the main features of KDVS concept is to lift the curse of dimensionality for statistical experiments that involve feature selection; see part 4.1.2 for more details. This feature is implemented by transforming main monolithic data set according to prior knowledge taken from external source, and instead of running one single execution of feature selection, running many separate feature selection tasks, and combining their results.

To transform primary data set, in prototype implementation we compute GO terms↔raw measurements mapping, that enables to select specific measurements connected with the GO term. Computation of this mapping is performed in database processor; see part 4.2.3 for more details. Selection of relevant measurements is simply a masking of main data set, extracting only relevant rows/columns. That is, for each GO term, a subset of original data set is produced, here called submatrix. The masking procedure is repeated for all considered GO terms, e.g. for the whole domain requested GOD.

The next step is to assign proper machine learning method to be executed for every created submatrix. The natural choice is one of the feature selection methods. $\ell_1\ell_2$ was chosen to be the primary feature selection method for KDVS prototype. Primary experience with this technique [BMRV08], however, has suggested that it may not be necessary to run it on certain submatrices. Specifically, if the number of samples $N$ is much smaller than number of variables (in our case, the total number of measurements) $P (N \ll P)$, it is feasible to run $\ell_1\ell_2$. Otherwise it is not necessary anymore, since contrary to feature selection idea, there is not that many variables to select from [HTF01]. In those cases, a simple estimation of linear regression model is sufficient [HTF01], and all variables in the data set at once may be treated as selected or not, dependent on the estimation outcome. For this purpose, in the KDVS prototype we used implementation (described in part A.2) of Regularized Least Squares (RLS) [Tib94], where the following solution is found:

$$
\beta^* = \arg\min_{\beta} \left\{ \frac{1}{n} ||Y - X\beta||_2^2 + \mu ||\beta||_2^2 \right\}
$$

This solution is obtained always for constant value of $\mu = 0$. This fact frees us from necessity of performing model selection, e.g. through cross validation [HTF01]. In this thesis, we refer to the usage of this implementation as RLS.

The process of assigning machine learning method to the submatrix, is controlled by arbitrary threshold $T$ over the number of variables in the submatrix. That is, if the number of variables in the submatrix is greater than the threshold value ($P > T$), we use $\ell_1\ell_2$; otherwise ($P \leq T$), we use RLS. Table 4.3 summarizes the decision process of assigning
feature selection technique to each submatrix. The preparatory phase is summarized on Figure 4.16.

<table>
<thead>
<tr>
<th>Submatrix size</th>
<th>Conditions</th>
<th>Technique used</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$ samples $\times P$</td>
<td>$P &gt; T$</td>
<td>$\ell_1\ell_2$</td>
</tr>
<tr>
<td>$N$ samples $\times P$</td>
<td>$P \leq T$</td>
<td>RLS</td>
</tr>
</tbody>
</table>

Table 4.3: Assignment of feature selection method in knowledge discovery.

Figure 4.16: Preparatory phase in Knowledge Discovery in KDVS pipeline. GEDM matrix is transformed into submatrices. Each submatrix, depending on its size, receives feature selection method to be executed with it. This phase ends where all submatrices are created and all method assignments are finished.

4.2.4.2 Execution phase

For each technique, we provide data set and label vector to estimate prediction model. In case of RLS, the whole submatrix is the data set. $\ell_1\ell_2$, however, is a double optimization technique with two loops of cross validation [DMDVR09], and it operates on splits of data set. Therefore, the submatrix is split on training and test subset at random, creating different external splits. Each external split is treated as one atomic computational process, that can be performed immediately or can be distributed for parallel execution.

In the prototype implementation, we utilize parallel computational environment extensively, due to computational time requirements of $\ell_1\ell_2$. Each parallel job (also called task), is either one execution of $\ell_1\ell_2$ over external split of submatrix, or execution of RLS over whole submatrix. Parallel jobs are queued, in order of data set size (bigger ones are submitted earlier), into a control server that distributes them, controls their individual execution, and collects their results. This functionality is performed by in–house framework, PPlus[A.3].

The execution phase finishes when all parallel jobs are completed and results from them are fully collected.

The execution phase is summarized on Figure 4.17.
4.2.5 Post–processing and visualization

The main aim of this phase is to collect results from knowledge discovery and systematize them into consistent output from the KDV5 pipeline.

In the prototype implementation, the workflow can be divided into the following conceptual parts: collection and processing of individual results obtained for each submatrix, collection of global results over all submatrices and individual results obtained earlier, and visualization of some aspects obtained from earlier activities. The overall schema is presented on Figure 4.18.

4.2.5.1 Individual results

During knowledge discovery phase (see part 4.2.4), specific feature selection method had been assigned to each submatrix, and subsequently executed within atomic computational processes. In this step we operate on data produced by those processes. The step is composed of the following substeps: data review, error reconstruction, error threshold verification, error plotting, and frequency verification. The overall schema is presented on Figure 4.19.
Figure 4.19: General schema of producing individual results in post-processing. Such results are generated for each individual submatrix produced in preparatory phase of knowledge discovery. Some individual steps may depend on feature selection method used.

**Data review** In this step we work on the results from computational processes in the following way. For $\ell_1\ell_2$, it is checked if all external splits were distributed properly and results from them were produced correctly; we discard the individual results for this submatrix even if a single external split has not been produced correctly. For RLS, we check if single task was distributed properly and if it produced correct results, in sense of machine readability. Note that this step may filter out some submatrices; next step is performed only on those that have passed this one. The concept is illustrated on Figure 4.20.

Figure 4.20: Schema of data review step performed during obtaining individual results in post-processing. Partial results produced by parallel jobs are reviewed if they contain expected data. If not, the submatrix is excluded from further post-processing. For external splits produced by $\ell_1\ell_2$, even single incorrect external split disqualifies the submatrix.

**Error reconstruction** In this step, we need to obtain error estimate for each reviewed submatrix, to use for later decisions. The general schema of this step is presented on Figure 4.21.

For $\ell_1\ell_2$, we collect partial output from external splits. Since we are dealing with results from double cross validation loop, we also need to reach some results from internal splits. In short, the submatrix is divided randomly in two parts, training and test, and this process is repeated a number of times, giving different training and test parts each time. Those divisions form external splits. Further, inside $\ell_1\ell_2$, for each external split, its training part
Figure 4.21: Schema of error reconstruction step, performed during obtaining individual results in post–processing. We analyze partial output for positively reviewed submatrices, and obtain the error value(s) in the way specific for the feature selection method used with individual submatrix. This step does not filter out any submatrices.

is divided further in the same way, into training and test parts at random, and also a number of times, to form internal splits. On each split, internal or external, a prediction model is validated, which produces error values. Our task is to collect error values across all splits, and calculate one final error estimate value, that will be used later to decide if submatrix has “passed” the statistical classification step.

Note that $\ell_1 \ell_2$ technique uses three parameters: $\tau$, $\lambda$ and $\mu$ [DMDVR09, DMMTV09]. While first two parameters are more “technical” ones, i.e. utilized heavily inside the procedure, the $\mu$ parameter is the one “exposed” outside, in the sense that separate list of selected features is reported for each value of $\mu$. Thus, an error estimate is calculated and reported for each value of parameter $\mu$. The reasoning described below is valid for each value of $\mu$.

The idea of nested splits, and all error values obtained (for fixed $\mu$), is illustrated on Figure 4.22.

The most important error calculation components for external split are two values, $\text{err}_{\text{tr}}$ and $\text{err}_{\text{ts}}$, being training and test errors, respectively, for best prediction model, validated with help of internal splits. For completion, we obtain also averaged training and test errors for all internal splits ($kcv_{\text{err}_{\text{tr}}}$, $kcv_{\text{err}_{\text{ts}}}$). Those values are used, if needed so, for manual verification of behaviour of external split.

Having training and test error values for best prediction model ($\text{err}_{\text{tr}}$, $\text{err}_{\text{ts}}$), we average them across all external splits to obtain $\text{avg}_{\text{err}_{\text{tr}}}$ and $\text{avg}_{\text{err}_{\text{ts}}}$. In addition, we calculate the following values across all external splits: standard deviation for $\text{err}_{\text{tr}}$ and $\text{err}_{\text{ts}}$, and median for $\text{err}_{\text{ts}}$.

At the end, average test error across all external splits ($\text{avg}_{\text{err}_{\text{ts}}}$), is used as error estimate.

As mentioned before, we collect error estimates for each value of parameter $\mu$.

For RLS, this step is much simpler, since submatrix is not split in any way. We collect
Figure 4.22: Schema of reconstructing error estimate, specific for $\ell_1/\ell_2$, performed during error reconstruction post-processing step. The reasoning below is valid for each value of $\mu$ parameter. Here, for each external split, we retrieve training and test error for best prediction model ($\text{err}_\text{tr}$, $\text{err}_\text{ts}$), validated with internal splits. For completion, we obtain also average training and test error calculated for internal splits ($\text{kcv}_\text{err}_\text{tr}$, $\text{kcv}_\text{err}_\text{ts}$). Thus, for each external split we have four values. We average $\text{err}_\text{tr}$ and $\text{err}_\text{ts}$ across all external splits to produce $\text{avg}_\text{err}_\text{tr}$ and $\text{avg}_\text{err}_\text{ts}$. The last value, $\text{avg}_\text{err}_\text{ts}$, is our final error estimate, used later to decide if submatrix has “passed” classification step.

Error threshold verification From previous steps, for each eligible submatrix, we obtain error estimate value(s). In this step, error estimate is confronted with external error threshold, to determine if submatrix has “passed” statistical classification step, or in other words, if GO term associated with it is statistically meaningful. All error values for all submatrices are compared against the same error threshold. The general schema of this step is presented on Figure 4.23.

For $\ell_1/\ell_2$, we confront all error estimate values, produced for respected parameter $\mu$ values, with error threshold. The reasoning is as follows. If the error estimate is less than the error threshold, we note an acceptance event for this $\mu$ value, and rejection event otherwise. If at least one acceptance event was noted for some $\mu$ value, we accept the submatrix for further processing, and reject it otherwise.

For RLS, the procedure is simpler. We compare one single error estimate value with the same error threshold. If the error estimate is less than the threshold, we accept the submatrix for further processing, and reject it otherwise.
The procedure is more complicated for $\ell_1\ell_2$, where error estimates were obtained for all values of $\mu$ parameter. Here, we confront each single error estimate with the error threshold. If for some $\mu$ the error is less than the threshold, we accept that $\mu$ value. If at least one $\mu$ value was accepted, the whole submatrix is accepted, i.e. labeled as statistically meaningful. For RLS, we just compare single error estimate and accept submatrix if the error is less than the threshold, and reject otherwise.

We use the same error threshold in all comparisons for all methods.

Note that this step may filter out further some submatrices; next step is performed only on those that have passed this one.

At the end, for $\ell_1\ell_2$ we store all $\mu$ values from acceptance events for submatrices that passed error threshold check. Further, we refer to those values as valid $\mu$ values. Also, for each valid $\mu$ value, $\text{avg.err}_\text{ts}$ is stored to be reported as final error estimate for that valid $\mu$ value. For RLS, we store single error value if submatrix passed passed error threshold check; it will be subsequently reported as final error estimate.

**Error plotting** This step is performed only for $\ell_1\ell_2$ for all submatrices that passed error threshold verification. Here we use all error values obtained in error reconstruction step.

For averaged training and test errors for all internal splits ($kcv.err_{\text{tr}}, kcv.err_{\text{ts}}$), in addition, we retrieve the full grid of such values for respective values of $\tau$ and $\lambda$ parameters used in $\ell_1\ell_2$ model selection procedure, and produce the logarithmic plot of those two error surfaces over the same grid ($\text{error vs log}_{10}\tau$ and $\text{log}_{10}\lambda$, respectively). The example is presented on Figure 4.24.

For training and test errors obtained for best prediction model ($err_{\text{tr}}, err_{\text{ts}}$) for each external split, we produce logarithmic box plots of these values against all $\mu$ values ($\text{error vs log}_{10}\mu$). The example is presented on Figure 4.25.
Figure 4.24: Example error surfaces plotted for $\ell_1 \ell_2$ output. Here, we visualize surfaces of average training (red) and test (blue) errors, obtained from internal splits, over the grid of $\tau$ and $\lambda$ values (in logarithmic scale).

**Frequency verification** In this step, we operate on submatrices that have passed error threshold verification. While previous steps focused on submatrices, here the focus is put on variables selected from submatrices. Since for $\ell_1 \ell_2$ we are dealing among others with splits of the original data set, we need to establish a notion of variable being *properly selected*. The general schema of this step is illustrated on Figure 4.26.

$\ell_1 \ell_2$ produces sparse list of selected variables (features) for each value of parameter $\mu$. From the previous step, we obtained a subset of valid $\mu$ values, collected from acceptance events. Thus, for individual submatrix, we have obtained in total $M_v \times K$ selected variables, where $M_v$ is the number of valid $\mu$ values, and $K$ is the number of external splits. We check, for each individual variable, how frequent it appears across all $M_v \times K$ selected variables; the frequency is cast as percentage ($freq_{perc}$). We compare it with given frequency threshold. If the frequency for individual variable is greater than frequency threshold, we note the variable as *properly selected*, and note it as *not properly selected* otherwise.

For RLS, the procedure is simpler. If only the submatrix has passed error threshold verification, we treat all its variables as *properly selected*, otherwise as *not properly selected*. Thus, this decisive action conforms with the introduction of computationally simpler technique for variable selection, described in part 4.2.4.
4.2.5.2 Individual output

At the end, for each individual submatrix that passed error threshold verification, we produce the following individual output, depending on what computational technique was used.

For $\ell_1/\ell_2$, we store error surface plot and box plots described in Error plotting, and for each valid $\mu$ value we store a list of annotated properly selected variables, along with their appearance frequencies, chosen according to the reasoning described in Frequency verification.

For RLS, we store simply a list of annotated properly selected variables.

In the individual output, all properly selected variables, being essentially raw measurements (specifically, probesets for microarrays), are annotated with biological references. Currently, we attach the following biological annotations: gene symbols\(^{24}\), ENTREZ Gene identifiers\(^{25}\), and Genbank sequence identifiers\(^{26}\). The annotations allow to resolve the function of measurement pointed by machine learning procedure as properly selected. We chose somewhat redundant annotation schema to mitigate a common unresolved problem of gene naming, that affects not only bioinformatics, but also the processing of life science data in general \cite{CLF05}. In this context, annotations are ordered regarding their strength in resolving functionality of requested sequence, from gene symbol being the “strongest”, down to GenBank identifier being the “last resort”. Also, in some cases gene symbol and/or

\(^{24}\)http://www.genenames.org
Figure 4.26: Schema of frequency verification step, performed during obtaining individual results in post-processing. Here we determine properly selected variables for each submatrix. For $\ell_1 \ell_2$, we evaluate each individual variable reported selected by the procedure. If it was selected frequently enough across all external splits, according to some threshold, we accept it as properly selected variable. For RLS, the submatrix has already passed error threshold verification, so we treat all its variables as properly selected ones.

ENTREZ Gene identifier may not be present. In such cases, since all biological sequences have been traditionally submitted to Genbank [BKML+05], the GenBank ID may be used to track back complete annotation history of the sequence.

Regarding this, each “variable” can be followed back from Gene Expression Data Matrix, where being raw measurement identifier, down to individual output, where being biologically meaningful through annotating process. This concept is illustrated on Figure 4.27.

Figure 4.27: A schematic way back from measurement in Gene Expression Data Matrix, through variable in knowledge discovery, down to fully annotated entity in individual output.

The example list of variables produced for a submatrix processed with $\ell_1 \ell_2$ is presented on Figure 4.28. The example list of variables produced for a submatrix processed with RLS is presented on Figure 4.29.

4.2.5.3 Global results

With processing of all individual results, we also gather some additional data to obtain “global picture” of the knowledge discovery phase.

We recall that submatrices are essentially grouped, depending on what computational tech-
Figure 4.28: The sample list of variables produced for a submatrix processed with $\ell_1\ell_2$, generated in individual output.

Figure 4.29: The sample list of variables produced for a submatrix processed with RLS, generated in individual output.

We are interested in the group level information, since more detailed information can be obtained there. We may be interested, however, also in joining some results across groups, for example to produce a list of submatrices that pass error threshold verification in all groups at once. To do this, we introduce an idea of degree of freedom, connected with each group. For example, for $\ell_1\ell_2$, we are interested in analysing results for different values of parameter $\mu$ separately [DMDVR09, DMMTV09]. In this context, each distinct $\mu$ value can be treated as one degree of freedom in producing global results. In case of RLS, there is no parameter range to depend on, but still we can conceptually assign an abstract degree of freedom to it. Having degrees of freedom assigned for all groups, we perform merge of results by pairing specific degrees of freedom from specific groups. This way, we can present some global reports across all entities, at the level we find appropriate or meaningful.

For example, we may assign indexes (0, 1, 2) to range $(\mu_1, \mu_2, \mu_3)$ of parameter $\mu$ for $\ell_1\ell_2$--
related group, and “virtual” index $-1$ for RLS–related group. We collect some results connected to particular $\mu$ value within $\ell_1\ell_2$–related group, as well as results for RLS–related group that will be associated with “virtual” index. Based on those assignments, we can now combine parts of the results corresponding to specified indexes, i.e. $0 \leftrightarrow -1$, $1 \leftrightarrow -1$, $2 \leftrightarrow -1$, $0 \leftrightarrow 1 \leftrightarrow 2$ etc.

This concept is illustrated on Figure 4.30.

![Figure 4.30: The concept of group level information vs global level information in global results collected during post-processing. On the left, results for $\ell_1\ell_2$ depend on range of 3 values of parameter $\mu$, and we can report output for each value separately, as well as output for the whole group. Results for RLS do not depend on any parameter, but we still assign “virtual” parameter 0 there. Therefore, we can produce reports across $\ell_1\ell_2$ and RLS groups in automated way. On the right, the generalization of this concept to more groups is presented.](image)

On **group level**, we collect: all submatrices participating in the group, submatrices from this group that passes error threshold verification procedure, submatrices from this group that do not pass error threshold verification procedure. For $\ell_1\ell_2$, in addition, for valid $\mu$ values, we collect submatrices associated with them. This concept is illustrated on Figure 4.31.

On **global level**, we collect: variables properly selected at least once in any submatrix, variables that were never properly selected anywhere, variables selected at least once in submatrices that passed error threshold verification, variables never selected in submatrices that passed error threshold verification, histograms built for above sets of variables (number of appearances of selected/non–selected variable across submatrices is counted). All sets and histograms are later split and reported for each group and each degree of freedom (here, $\ell_1\ell_2/\mu$ and RLS/0 respectively). Note that for RLS the number of properly selected variables that are not selected for submatrices that do not pass error threshold verification
step, is always zero (since all variables for passing submatrix are automatically selected), therefore this specific list/histogram pair is not built.

### 4.2.5.4 Global output

**Specific part** For group of submatrices associated with $\ell_1 \ell_2$, for each $\mu$ value, we produce a listing of submatrices that pass error threshold verification step, more specifically a list of GO terms associated with such submatrices, with some basic data like: mean of test error, standard deviation of test error, mean of prediction error, standard deviation of prediction error, median of test error, total number of variables associated with the submatrix, total number of properly selected variables associated with the submatrix. The example output is presented on Figure 4.33.

Note that for each submatrix (associated with GO term), during error threshold verification step, we have noted acceptance and rejection events, that excluded some $\mu$ values for particular submatrices. Therefore, lists mentioned here may have different lengths. Basically, a GO term appears on the list associated with $\mu$ value, if for a submatrix an acceptance event was noted for that $\mu$ value.

Such detailed report is not necessary for RLS group, where we are not operating on splits.
Figure 4.32: Different variable lists generated in group level within global results during post-processing. In general, we count appearance or absence of each single variable, across specific submatrices, selected or not. We collect: variables that are properly selected at least once in any submatrix, variables that were never properly selected anywhere, variables selected at least once in submatrices that passed error threshold verification, variables never selected in submatrices that passed error threshold verification. For each list, an additional histogram is generated.

**Merge part** Based on reasoning, presented in previous part (4.2.5.3), about merging certain results across groups, we request specific merges between $\ell_1\ell_2$–related and RLS–related results. In the prototype implementation, we used $(0, \ldots, m-1)$ indexes for $(\mu_1, \mu_2, \ldots, \mu_m)$ range of $\mu$ values for $\ell_1\ell_2$–related results, and 0 index for RLS–related results. Currently, we produce merged results according to the following pairings: $0 \leftrightarrow 0, 1 \leftrightarrow 0, \ldots, m-1 \leftrightarrow 0$. For pair $(A, B)$ we produce reports of the following structure:

```
[output part related to A]
...
[output part related to B]
...
```

Specifically, for each requested merge between degrees of freedom, as shown on Figure 4.30, we currently produce a listing of submatrices that pass error threshold verification step, with the following additional information: identifier of GO term associated with submatrix, GO term name, an *error estimate* specific for the submatrix, total number of variables associated with the submatrix, total number of properly selected variables associated with the submatrix, number of true positives, true negatives, false positives and false negatives,
<table>
<thead>
<tr>
<th>GO term ID</th>
<th>GO term name</th>
<th>Tot vars</th>
<th>Sel vars</th>
<th>Error estimate</th>
<th>#TP</th>
<th>#TN</th>
<th>#FP</th>
<th>#FN</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO:0003677</td>
<td>DNA binding</td>
<td>2699</td>
<td>1295</td>
<td>0.262235</td>
<td>262</td>
<td>618</td>
<td>432</td>
<td>100</td>
<td>0.2727</td>
</tr>
<tr>
<td>GO:0000166</td>
<td>nucleotide binding</td>
<td>2641</td>
<td>1570</td>
<td>0.257119</td>
<td>26</td>
<td>1</td>
<td>635</td>
<td>415</td>
<td>101</td>
</tr>
<tr>
<td>GO:0008270</td>
<td>zinc ion binding</td>
<td>2532</td>
<td>1489</td>
<td>0.252428</td>
<td>262</td>
<td>64</td>
<td>5</td>
<td>405</td>
<td>100</td>
</tr>
<tr>
<td>GO:0004872</td>
<td>receptor activity</td>
<td>2025</td>
<td>1535</td>
<td>0.271277</td>
<td>250</td>
<td>628</td>
<td>422</td>
<td>112</td>
<td>0.2524</td>
</tr>
</tbody>
</table>

Histograms

We produce histogram reports separately for each degree of freedom. Note that we produce more histograms than we report here. Currently, for $\ell_1\ell_2$–related data, for each $\mu$ value, we report: a histogram of variables that were properly selected at least once in submatrices that passed error threshold verification step, together with histogram of variables that were not properly selected at least once in submatrices that passed error threshold verification step. The respective samples are presented on Figures 4.35 and 4.36.

<table>
<thead>
<tr>
<th>Selected at least once: 4334</th>
<th>Nodes passing TS error: 154</th>
<th>Nodes in BP:18029</th>
</tr>
</thead>
<tbody>
<tr>
<td>1767_s_at</td>
<td>TGB3</td>
<td>14</td>
</tr>
<tr>
<td>1852_s_at</td>
<td>TNF</td>
<td>12</td>
</tr>
<tr>
<td>2067_f_at</td>
<td>BAX</td>
<td>10</td>
</tr>
<tr>
<td>1998_i_at</td>
<td>BAX</td>
<td>10</td>
</tr>
<tr>
<td>448_s_at</td>
<td>MEH1</td>
<td>9</td>
</tr>
<tr>
<td>33460_s_at</td>
<td>ZEB1</td>
<td>9</td>
</tr>
</tbody>
</table>
Not selected at least once: 5926 Nodes passing TS error: 154 Nodes in BP:18029
1735_g_at TGFβ3 14
41445_at TGFβ1 14
1634_s_at TGFβ1 14
259_s_at TNF 14
2060_at BCL2 14
380_at TBX5 12

Figure 4.36: The sample histogram output, for histogram of variables not properly selected at least once, produced for specific μ value for ℓ₁ℓ₂–related data. The columns, from left to right, are: probeset identifier, gene symbol, and variable count. The header refers to total number of variables in this histogram, number of GO terms that passed error threshold verification step associated with this μ value, and total number of GO terms in considered GO domain (hence we reconstruct functionalities limited to specific GO domain).

Considering RLS–related data, and the fact that all variables are counted as properly selected when the submatrix passed the error threshold verification step, we report only the histogram for variables selected at least once. The sample is presented on Figure 4.37.

Selected at least once: 4303 Nodes passing TS error: 571 Nodes in BP:18029
2038_g_at BCL2 43
1847_s_at BCL2 43
1909_at BCL2 43
1910_s_at BCL2 43
2060_at BCL2 43
31724_at SHH 24

Figure 4.37: The sample histogram output, for histogram of variables properly selected at least once, produced for RLS–related data. The columns, from left to right, are: probeset identifier, gene symbol, and variable count. The header refers to total number of variables in this histogram, number of GO terms that passed error threshold verification step for RLS, and total number of GO terms in considered GO domain (hence we reconstruct functionalities limited to specific GO domain).

4.2.5.5 Visualization

We recall that in the prototype implementation we are working with Gene Ontology, a source of prior knowledge that displays certain structure. We would like to emphasize some properties of the output produced by the pipeline, in the context of that structural properties.

Since the main structural property of GO is Directed Acyclic Graph (DAG) internal organization, it is desirable to explore direct connections between selected GO terms and their placement in DAG. To do this, in prototype implementation we incorporate two procedures, realized in the form of R scripts.

Simple minimal local neighborhood Knowledge discovery produces a list of meaningful GO terms (each submatrix was associated with one GO term), without assuming any direct connection between the terms.
non–statistical connections between them. In order to perform any visualizations based on these results, there is a need to reconstruct local neighborhood of GO terms in question. Essentially, there is a need to construct a minimal subgraph from whole GO DAG, that contains all requested nodes and conforms with the topology of parent DAG. Then, the resulting graph is plotted, with the requested nodes highlighted. The construction of minimal subgraph is currently performed with R package \texttt{GOStats}, and plotting is done with GraphViz graph visualization software, interfaced directly from R.

For example, APBB2 is a gene strongly associated with progress of Alzheimer’s disease. There are three GO terms from Molecular Function domain associated to it: \textit{protein binding} (GO:0005515), \textit{transcription factor binding} (GO:0008134), \textit{beta-amyloid binding} (GO:0001540). A minimal subgraph for those terms, built with \texttt{GOstats} package, contains additional inferred terms: \textit{binding} (GO:0005488), \textit{molecular function} (GO:0003674) (root term of MF domain), and \textit{all} (abstract term added by \texttt{GOstats} that associates all roots of three GO domains). The plot of subgraph is presented on Figure 4.38.

**Semantic clustering** To explore some non–trivial associations between statistically meaningful GO terms, it is necessary to use the GO structural properties in more refined way. In the prototype implementation, we apply a hierarchical agglomerative clustering (with average linkage) to the list GO terms obtained from knowledge discovery. The clustering makes use of Resnik semantic similarity \cite{Res99}, normalized to the maximum observed value, to assess the degree of relatedness between two GO terms. More precisely, for terms \(C_1\) and \(C_2\):

\[
Sim_{Resnik}(C_1, C_2) = \max_{C \in MICA(C_1, C_2)} IC(C)
\]

where \(IC\) is the Information Content \cite{LSBG03} and \(MICA\) indicates the most informative common ancestors of terms \(C_1\) and \(C_2\) in the GO DAG. The \(IC\) for the GO term \(C\) is defined as:

\[
IC(C) = -\log \left( \frac{freq(C)}{freq(root)} \right)
\]

that is, the negative logarithm of the ratio between a frequency of the term \(C\) in a corpus of annotations and a frequency of the root term. The numerator here corresponds to the number of times the term \(C\) and each of its descendants occur in the pre–specified set of GO annotations. The denominator here corresponds to the sum of the frequencies of all

---


\[29\]http://www.graphviz.org


\[31\]http://www.geneontology.org/GO.downloads.annotations.shtml
Figure 4.38: Example plot of minimal subgraph obtained for three GO terms (green) of Molecular Function domain, identified for APBB2 gene, strongly associated with progress of Alzheimer’s disease. The subgraph contains also inferred terms (grey), added during building of subgraph, that lead up to the abstract root of the whole Gene Ontology (here drawn as “all”).

GO terms.

The IC decreases monotonically when moving from the leaves toward the root node (IC = 0). The intuition behind the use of the IC is that the more probable a concept is, the less information it conveys. The definition of MICA is not straightforward in GO DAG, since GO terms can have several disjoint common ancestors. In evaluating Sim\textsubscript{Resnik} we take into account the lowest common ancestor with maximum IC \textsubscript{[LSBG03]}. The more informative the common ancestor is, the greater the information shared by the concepts, and consequently their similarity. The direct output from the clustering is a set of semantically homogeneous clusters of requested GO terms.

The clusters are visualized in the similar way as before. That is, the construction of minimal subgraph is performed, and resulting graph is plotted, this time with the clusters of nodes highlighted using different colours; same tools are used to produce visual output.
An example plot of four semantic clusters of terms, located near the root of Molecular Function domain, is presented on Figure 4.39.

**Interactive visualization** We are experimenting with the introduction of interactivity into visual results. In our case there is a need to display large DAG graphs in an effective manner, that also allows certain level of individual exploration of the structure presented \cite{HMM00, VD04}. Currently, we evaluate various graph visualization software packages \cite{SMO+03, BHJ09}, and explore new visualization paradigms that come both from bioinformatics \cite{SOR+11} and network analysis communities \cite{CSWE05}.

### 4.2.6 Extensions

In the light of implemented proof–of–concept, the following rationale may be formulated, that describes the possibility for expanding the KDVS concept into full scalable framework.
4.2.6.1 General ideas

More data and knowledge sources may be introduced. In this thesis, we focused on the proof-of-concept implementation that has been demonstrated on series of experiments that utilize one “data type”, being microarray data. New data types may be tried, however, even with the existing implementation, requiring only small adapting changes. Such try may serve as a starting point for further re-evaluation of data type for knowledge discovery experiment in general [KC07, HTF01].

More possibilities in utilizing available relations between data and knowledge sources may be explored. For example, it may be worth to generalize mappings obtained from prior knowledge in the current implementation. Since they are essentially similar to JOIN clauses in SQL query language [GMUW08], similar principles may be devised for them, and subsequently implemented, even using existing relational database systems as back-end.

The more flexible output may be produced, taking into account all possible target use groups of the framework, including non-computer scientists. For it, more interactivity shall be provided in the output, exploring more intuitive approaches to presentation of non-numerical information, such as clusters or graphs. In parallel, the core statistical output may be more annotated with non-numerical information, suitable for life science researchers, taking off the burden of manual re-annotation practice [Bre99, DV00].

4.2.6.2 Usage of prior knowledge

The general idea of prior knowledge as a guide in integration process may be further conceptualized into abstract procedure, that can reflect various strategies. More elaborate strategies may be devised as well. We have already provided some basic foundation of terminology in part 4.1.1.

Data processor concept may be refined to incorporate wide range of data sets produced by high-throughput technologies. In the prototype implementation we focused on the physical interchangeability of data in the form of Delimiter Separated Values (DSV) files. More refinement of this approach may facilitate data parsing, and thus facilitate development of new knowledge discovery strategies.

The concepts implemented in database processor, especially local data integration, may be refined to explore the ideas from data mining and best practices from relational database theory [GMUW08].
4.2.6.3 Knowledge discovery

The knowledge discovery concept shall be refined to introduce wider choice of implementations of machine learning algorithms. In particular, more emphasis shall be put on modularity and interchangeability of code, since many existing implementations have not been produced with software engineering principles in mind. The concept of execution environment in particular may be expanded to encompass validation of presence and format of shared variables between actions.

Also, since machine learning algorithms may be very space and time consuming (a good example being \( \ell_1 \ell_2 \)), the more general design and implementation of parallelization shall be introduced. The possibility here is to use a series of interchangeable adapters for various parallel and high–performance computing (HPC) environments and platforms. The idea may be jointly refined together with execution environment principle.

4.3 Summary

In this chapter, we presented the Knowledge Driven Variable Selection (KDVS) pipeline, starting with abstract methodology down to concrete implementation. In part 4.1 we discuss the core idea behind the pipeline methodology. More specifically, starting with the presentation in part 4.1.1 we discuss the potential positive features in part 4.1.2 and potential difficulties in part 4.1.3. Subsequently, in part 4.2 we present the practical implementation of the ideas discussed. In part 4.2.1 we focus on devising abstract schema for the pipeline, which elements are filled in subsequent parts. First, we discuss input data in more detail in part 4.2.2. Next, we present KDVS Core, that contains relatively stable functionality as of this writing, in part 4.2.3. Here, we cover the functionality mainly connected to input data processing and data integration, as well as the basic ideas of execution environment devised for the pipeline. Next, we present a prototype implementation of knowledge discovery part (according to schema presented on Figure 1.1) in part 4.2.4. Next, we focus on post–processing and visualization in part 4.2.5, currently the most volatile part of implementation in terms of closed functionalities. Finally, we focus briefly on possible extensions in part 4.2.6, where we discuss the changes that may transform the abstract pipeline methodology into full–fledged software framework.
Chapter 5

Experimental results

In this chapter, we present the experimental outcome from the data and knowledge integration, performed by prototype implementation of Knowledge Driven Variable Selection concept.

First, we specify the concept of experiment and its outcome in part 5.1. Next, we present general procedure followed by all presented experiments in part 5.2. The approximate time line of experiments performance in parallel with the development of methodology is presented in part 5.3. We present specific details of each experiment in parts 5.4, 5.5, and 5.6. Finally, we focus on a series of artificial experiments for assessing statistical stability of results in part 5.7.

5.1 Introduction

In this thesis, by an experiment we understand the analysis of microarray expression data obtained from patients in various stages of particular diseases. In all cases, the statistical classification is performed, involving comparison of measured gene expression for disease state and control state. Here, we consider the outcome for such experiments on three levels: functional landscape level, gene signature level, and individual gene level.

On functional landscape level, the aim is to reconstruct general landscape of gene expression in the disease state of question, and compare it to known molecular interactions specific for that disease. The outcome on this level comes in the form of descriptive remarks, and is produced with extensive literature verification. Positive outcome enables to confirm the validity of basic statistical properties of the approach, in the sense of performing classification on data sets modified by application of prior knowledge.
On *gene signature level*, the aim is to obtain, if possible, new complete signatures of genes that can allow performing classification of type “disease vs control”, as well as, or even more effectively than, well-established statistical approaches that do not utilize prior knowledge at such early stage. The outcome on this level comes in the form of list of genes, and is produced with discriminative power of machine learning methods. Positive outcome enables the possibility of routine usage of presented methodology in daily life science research.

On *individual gene level*, the aim is to identify and pinpoint possibly less known genes, which expression was statistically meaningful to the disease in question, and verify them as possible new members of already established networks of molecular interactions, involved in diseases in question. The outcome of this level comes in the form of descriptive remarks, and is produced with verification of significance hypothesis through literature mining. Positive outcome enables the possibility of using presented methodology to discover new life science knowledge.

During development of the proof of concept pipeline, we focused on producing the positive outcome on the *functional landscape level*, to verify the basic validity of the approach.

In this thesis, however, only preliminary results have been presented for some experiments, since functional verification takes considerable amount of time, and is performed typically by researchers with life science background.

We also signal here the possibility of going beyond functional verification, to pinpoint the outcome on the remaining levels. However, the realization of this point is beyond the scope of this thesis.

### 5.2 Overall procedure

General schema of KDVS experiment, as performed by the prototype implementation, is presented on Figure 5.1. All presented experiments were performed according to this schema. Implementation of this experimental schema comprises the current state of physical KDVS software pipeline. We also present on Figure 5.2 a visual comparison between the pipeline devised in part 4.2.1 (see also Figure 1.1 and Figure 4.9) and experimental schema. Some noticeable differences can be attributed to the high variability of proof–of–concept implementation.

All data sets used here were placed in public domain after collection and deposited in Gene Expression Omnibus (GEO), one of the main public databases that collect gene expression data [GEO].

In order, binary data containing proper expression measurements was collected and pre–
processed with proper BioConductor\(^1\) algorithms, in order to correct optical artifacts presented in original measurements, and convert them to statistically valid form. Specifically, normalization and quality control was always performed. If statistical classification is to be performed in knowledge retrieval phase, two (or more) raw data sets, each connected with different phenotypic characteristics, are used. In such case, data from multiple data sets, after pre–processing, is joined together into single Gene Expression Data Matrix, in the form of DSV file.

Annotations related to expression platform (in our case, specific microarray model) were collected as well, in the form of Delimiter Separated Values (DSV) file. In case of performing statistical classification in knowledge retrieval, label information was presented as DSV file.

All DSV files were loaded into SQLite experiment relational database, in the form of single binary file. Within such database, local data integration was performed.

Platform annotations may already provide a direct relationship between DNA probesets and Gene Ontology (GO) terms. Since all presented biological experiments were performed on Affymetrix microarrays, such information was present. Based on that, the reciprocal mapping \( GO \text{ term} \rightarrow \text{probesets} \) is generated. In addition, gene naming information was obtained, based on data provided by HUGO Gene Nomenclature Committee\(^2\) (HGNC), that are included with KDVS prototype.

Following \( GO \text{ term} \rightarrow \text{probesets} \) mapping, non–empty submatrices of Gene Expression Data Matrix were generated, in the form of binary files. Each submatrix, in turn, is associated with some GO term.

In knowledge discovery, for each submatrix, a statistical classification was performed with \( \ell_1\ell_2/RLS \), and results were collected, in the form of single prediction model for each submatrix.

In initial post–processing phase, all partial results were gathered, classification error estimate was calculated, and selected entities were extracted. At this point two levels of selection were considered. On first level, for each submatrix that passes classification error,

---

1http://bioconductor.org/
2http://www.genenames.org/
The idea of the output produced during knowledge retrieval and post-processing is pre-
sent on Figure 5.3.

Figure 5.3: Idea of output from knowledge discovery and post-processing phases, as seen in experimental setup. After submatrices have been produced, knowledge discovery is performed according to procedure described in part 4.2.4; the direct output of which are prediction models. During initial post-processing, we select submatrices that pass error threshold verification procedure. The GO terms associated with those submatrices comprise of first selection level. Also here, we determine properly selected variables, according to frequency verification procedure. Such variables comprise of second selection level. During final post-processing, we produce individual and global output, as well as all visual artifacts. The post-processing was described in detail in part 4.2.5.1.

Table 5.1 presents the summary of data sets used in the KDVS experiment, along with corresponding information distinction introduced in Table 2.1, and file type used by KDVS.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Information kind</th>
<th>KDVS File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw expression data</td>
<td>Raw data</td>
<td>BIN</td>
</tr>
<tr>
<td>Gene Expression Data Matrix</td>
<td>Processed data</td>
<td>DSV</td>
</tr>
<tr>
<td>Label information</td>
<td>Additional information</td>
<td>DSV</td>
</tr>
<tr>
<td>Platform metadata</td>
<td>Additional information</td>
<td>DSV</td>
</tr>
<tr>
<td>Gene Ontology data</td>
<td>Prior knowledge</td>
<td>RDF-XML</td>
</tr>
</tbody>
</table>

Table 5.1: Summary of file types used during KDVS experimental setup, as compared to semantic concepts presented in Table 2.1.

5.3 Time line

The relative time line of all experiments reported in this thesis is presented on Figure 5.4.
5.4 Experiment 1 - proof of concept on PD data

5.4.1 Goal

In this experiment, we analyzed microarray data derived from patients with late stage of Parkinson’s disease (PD). We stated simple research question here. The goal was to reconstruct general landscape of molecular characterization of Parkinson’s disease. Having Gene Ontology as prior knowledge source, for the sake of simplicity, we focus on the functionalities spanning single GO domain, Molecular Function (MF) [GOD].

5.4.2 Samples characteristics

Biological samples, from patients and controls, were collected from three different brain regions, and put together in the “superseries”. 40 disease and 53 control samples were obtained, giving 93 samples in total.

5.4.3 Experiment details

Raw data set (GSE20295 [ZJMD05 ZLL+10]) was obtained from GEO [GEO] repository. After pre–processing and normalization, a 93 × 22215 gene expression data matrix (GEDM) was produced, and stored in the form of DSV file. Label information was generated as well and stored as DSV file. Platform annotation data set for Affymetrix HG U133A microarray (GPL96) was obtained from GEO repository, with Download full table... option 3, and stored in the form of DSV file. The DSV files were loaded into local experiment database Experiment.db, and local data integration was performed therein, as described in part 4.2.3.

The preparatory step of knowledge discovery phase resulted in generation of 2574 non–empty submatrices, each corresponding to specific GO term from MF domain. For each submatrix, a binary classification problem, of discriminating disease from healthy status, 

was considered. During execution phase, the distributed calculations were performed in parallel environment. At the end of knowledge discovery phase, a classification error was estimated and a list of relevant variables for each submatrix, as described in part 4.2.4.

We focused further on the results obtained from $\ell_1\ell_2$ algorithm. We considered as predictive only those submatrices that scored a classification accuracy higher than 70%, obtaining a final list of meaningful 188 GO terms spanning MF domain.

### 5.4.4 Results

Some of the most relevant selected GO terms are presented in Table 5.2. The complete list of selected GO terms for this experiment is presented in Table 5.3. A very basic graph showing nearest local neighborhood of selected GO terms is presented on Figure 5.5.

<table>
<thead>
<tr>
<th>GO term ID</th>
<th>GO term name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO:0020037</td>
<td>heme binding</td>
</tr>
<tr>
<td>GO:0003774</td>
<td>motor activity</td>
</tr>
<tr>
<td>GO:0005507</td>
<td>copper ion binding</td>
</tr>
<tr>
<td>GO:0031402</td>
<td>sodium ion binding</td>
</tr>
<tr>
<td>GO:0005262</td>
<td>calcium channel activity</td>
</tr>
<tr>
<td>GO:0003777</td>
<td>microtubule motor activity</td>
</tr>
<tr>
<td>GO:0004896</td>
<td>cytokine receptor activity</td>
</tr>
<tr>
<td>GO:0004402</td>
<td>histone acetyltransferase activity</td>
</tr>
<tr>
<td>GO:0051059</td>
<td>NF-kappaB binding</td>
</tr>
<tr>
<td>GO:0032395</td>
<td>MHC class II receptor activity</td>
</tr>
<tr>
<td>GO:0004129</td>
<td>cytochrome-c oxidase activity</td>
</tr>
<tr>
<td>GO:0004181</td>
<td>metallo- and carboxypeptidase activity</td>
</tr>
<tr>
<td>GO:0005388</td>
<td>calcium-transporting ATPase activity</td>
</tr>
<tr>
<td>GO:0015085</td>
<td>calcium ion transmembrane transporter activity</td>
</tr>
<tr>
<td>GO:0005539</td>
<td>glycosaminoglycan binding</td>
</tr>
</tbody>
</table>

Table 5.2: Some of the most relevant GO terms selected in Experiment 1.

Taking into account biological relevance of GO terms selected, we found that the pipeline has reconstructed the molecular landscape of Parkinson’s disease fairly well, although we limited our focus to conceiving rather general viewpoint. Selected GO terms belong to several GO classes, including: general (intracellular, cytoplasm, negative regulation of biological process), nervous system (neurotransmitter transport, transmission of nerve impulse, learning, or memory), response to stimuli (behavior, temperature, organic substances, drugs, or endogenous stimuli), that are routinely colligated with Parkinson’s disease processes and activities [Fea04, LHR09, PDO].
Table 5.3: All selected GO terms obtained in Experiment 1.

5.4.4.1 Final remarks

Results for this experiment were reported in [ZBV11]. Summary of basic experiment properties is presented in Table 5.4.

5.5 Experiment 2 - Comparison of AD/PD

5.5.1 Goal

Alzheimer’s disease (AD) and Parkinson’s disease (PD) are common neurodegenerative diseases with similar, albeit not identical, molecular background [Mar99]. Although considerable amount of research has been devoted to them, the successful cure is yet to be developed [NE03].

In this experiment, we put more elaborated research question. That is, given the gene expression data from two closely related diseases, in terms of symptoms and pathology, could we identify any statistically sound common regulatory mechanisms between them? And in parallel, could we identify any mechanisms specific for them?
### Disease Characteristics

<table>
<thead>
<tr>
<th>Disease Data Set</th>
<th>Late stage of diagnosed Parkinson’s disease GSE20295</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Data Set</td>
<td>GSE20295</td>
</tr>
<tr>
<td>Platform Annotation Data Set</td>
<td>GPL96</td>
</tr>
<tr>
<td>Microarray Type</td>
<td>Affymetrix HG U133A</td>
</tr>
<tr>
<td>Number of disease samples</td>
<td>40</td>
</tr>
<tr>
<td>Number of control samples</td>
<td>53</td>
</tr>
<tr>
<td>Raw Data Normalization Used</td>
<td>GCRMA</td>
</tr>
<tr>
<td>Classification accuracy threshold</td>
<td>70%</td>
</tr>
<tr>
<td>Number of GO terms selected</td>
<td>188</td>
</tr>
</tbody>
</table>

Table 5.4: Summary of basic information for Experiment 1.

To give more complete answer, we introduced additional factor into consideration, namely development of the disease over time. This requires gathering data from patients that express a certain level of disease development at given time point. For neurodegenerative diseases, however, it is often difficult to specify precisely a state of development at given time point (with precision of days, for example). Therefore, we considered two loosely defined time points, “early” and “late”, for both AD and PD.

We used four public data sets from GEO repository that best matched the time points stated, namely: GSE9770 for AD early stage [LDB+07, LRV+08], GSE5281 for AD late stage [LDB+10], GSE6613 for PD early stage [SEM+07], and GSE20295 for PD late stage [ZJMD05, ZLL+10].

### Sample characteristics

**GSE9770 (early AD)** Disease samples were collected from patients that already present typical Alzheimer’s features, yet before dementia was developed. Although we refer to the stage as “early”, it is in fact an intermediate stage, since it is difficult to diagnose Alzheimer’s disease in early stage in general [LDB+07, LRV+08]. Samples were harvested from 6 different brain regions, giving 34 disease samples in total.

To perform statistical classification in “disease vs healthy” setup, we used control data coming with AD “late” stage (included within GSE5281), collected from 74 samples.

Regarding control samples, taking into account more detailed deliberations, we excluded the samples related to visual cortex, being not directly colligated with Alzheimer’s disease development [LDB+07, LRV+08]. In this case, we excluded 17 specific control samples.

Overall, we used 34 disease and 57 control samples, giving 91 samples in total.

79
GSE5281 (late AD)  Disease samples were collected from patients that express full spectrum of Alzheimer’s features. Samples were harvested from 6 different brain regions, giving 87 disease samples in total. Control samples were included in the same data set, giving 74 control samples in total. Overall, 161 biological samples were obtained.

Taking into account more detailed deliberations, we excluded the samples related to visual cortex, being not directly colligated with Alzheimer’s disease development [LDB+07, LRV+08]. In this case, we excluded 14 disease and 17 control samples. Overall, we used 73 disease and 57 control samples, giving 130 samples in total.

GSE6613 (early PD)  Biological samples were collected from affected and healthy people, where patients were affected both by early–stage Parkinson’s disease and other neurogenerative diseases. As a result, three groups were formed: PD group, other NDD group, and control group. For our purposes, we excluded second group, focusing on PD group and control group entirely. The blood samples were used across all groups as biological samples. For disease group, total of 50 samples were collected. For control group, 22 samples were collected. In total, we recognized 72 samples.

Overall, we used 50 disease and 22 samples, giving 72 samples in total.

GSE20295 (late PD)  Biological samples were obtained from affected and healthy people in post mortem way. Total of 15 patients with advanced Parkinson’s disease and 15 healthy controls were used as donors. For each person, specific tissue samples from three different brain areas were harvested. Respectively, 40 disease–related and 53 control–related samples were collected. In total, we recognized 93 biological samples.

Overall, we used 40 disease and 53 samples, giving 93 samples in total.

5.5.3 Experiment details

Having Gene Ontology as prior knowledge source, we focused on the analysis of functionalities spanning Molecular Function GO domain (MF) [GOD].

We conducted four separated pipeline runs, that is for early AD, late AD, early PD and late PD. Each run was performed in the same way, for their respective data sets. For the sake of description, we present here a single run, namely early PD, in more detail.

**Early PD run**  Raw data set (GSE6613 [SEM+07]) was obtained from GEO [GEO] repository, and we performed its pre–processing and normalization with GCRMA [WIG+04],
after which, a $72 \times 22215$ gene expression data matrix (GEDM) was produced, and stored in the form of DSV file. Label information was generated as well and stored as DSV file. Platform annotation data set for Affymetrix HG U133A microarray (GPL96) was obtained from GEO repository, with Download full table... option, and stored in the form of DSV file. The DSV files were loaded into local experiment database Experiment.db, and local data integration was performed therein, as described in part 4.2.3.

The preparatory step of knowledge discovery phase resulted in generation of 2574 non-empty submatrices, each corresponding to specific GO term from MF domain. One may notice that the number of submatrices is the same as in the previous experiment (described in part 5.4). This is because the gene expression data were collected on the same microarray platform type, and the current implementation re-creates the needed mapping dynamically each time, as explained in part 4.1.3.2.

For each submatrix, a binary classification problem, of discriminating disease from healthy status, was considered. During execution phase, the distributed calculations were performed in parallel environment. At the end of knowledge discovery phase, a classification error was estimated and a list of relevant variables for each submatrix, as described in part 4.2.4.

We focused further on the results obtained from $\ell_1 \ell_2$ algorithm. We considered as predictive only those submatrices that scored a classification accuracy higher than specified threshold, namely 70%. Regarding this, we collected a list of 85 meaningful GO terms that span MF domain.

**Other runs** For other runs, the basic run description stays the same. The differences are, besides data sets and platforms, when applicable, also between final GEDM dimensions. We collected all basic differences in Table 5.5. The results are presented separately in Table 5.6.

<table>
<thead>
<tr>
<th>NDD stage</th>
<th>Data set</th>
<th>Platform set</th>
<th>MA type</th>
<th>GEDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD early</td>
<td>GSE9770</td>
<td>GPL570</td>
<td>HG U133+</td>
<td>91 × 54613</td>
</tr>
<tr>
<td>AD late</td>
<td>GSE5281</td>
<td>GPL570</td>
<td>HG U133+</td>
<td>130 × 54613</td>
</tr>
<tr>
<td>PD early</td>
<td>GSE6613</td>
<td>GPL96</td>
<td>HG U133A</td>
<td>72 × 22215</td>
</tr>
<tr>
<td>PD late</td>
<td>GSE20295</td>
<td>GPL96</td>
<td>HG U133A</td>
<td>93 × 22215</td>
</tr>
</tbody>
</table>

Table 5.5: Basic differences between different pipeline runs in Experiment 2. The columns are: disease at particular time point, GEO ID of corresponding gene expression data set, GEO ID of platform annotation data set, type of microarray platform (here, all are Affymetrix chips), Gene Expression Data Matrix dimensions: #samples×#variables (here, variables are probesets).

5.5.4 Results

The basic results for all pipeline runs are summarized in Table 5.6. Full lists of selected GO terms were omitted for clarity.

<table>
<thead>
<tr>
<th>NDD stage</th>
<th>Class. accuracy thr.</th>
<th>Selected GO terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD early</td>
<td>70%</td>
<td>85</td>
</tr>
<tr>
<td>AD late</td>
<td>70%</td>
<td>324</td>
</tr>
<tr>
<td>PD early</td>
<td>70%</td>
<td>85</td>
</tr>
<tr>
<td>PD late</td>
<td>70%</td>
<td>179</td>
</tr>
</tbody>
</table>

Table 5.6: Summary of basic results obtained from Experiment 2. The columns are: disease at particular time point, classification accuracy threshold, and number of selected GO terms.

We also present here a series of comparisons performed for lists of selected GO terms obtained from four pipeline runs. The comparisons are presented as nearest local neighborhoods and plotted accordingly.

**AD early vs AD late**  Here we focus on Alzheimer’s disease specific output, comparing resulting GO terms generated for AD early and AD late stages. The functional landscape, plotted as nearest local neighborhood over MF domain, is presented on Figure 5.6.

**PD early vs PD late**  Here we focus on Parkinson’s disease specific output, comparing resulting GO terms generated for PD early and PD late stages. The functional landscape, plotted as nearest local neighborhood over MF domain, is presented on Figure 5.7.

**AD early vs PD early**  Here we focus on early stage of diseases, comparing resulting GO terms generated for AD early and PD early stages. The functional landscape, plotted as nearest local neighborhood over MF domain, is presented on Figure 5.8.

**AD late vs PD late**  Here we focus on late stage of diseases, comparing resulting GO terms generated for AD late and PD late stages. The functional landscape, plotted as nearest local neighborhood over MF domain, is presented on Figure 5.9.

5.5.4.1 Final remarks

Preliminary results for this experiment were partially introduced in [SMZB11]; more detailed analysis is to be published. For initial comments see part 6.2. Summary of basic experiment properties is presented in Table 5.7.
Table 5.7: Summary of basic information for Experiment 2. The columns are: disease at particular time point, GEO ID of corresponding gene expression data set ("disease" in "disease vs control" experiment setup), literature reference to "disease" data set, GEO ID of corresponding gene expression data set ("control" in "disease vs control" experiment setup), GEO ID of platform annotation data set, type of microarray platform (here, all are Affymetrix chips), number of "disease" samples, number of "control" samples, BioConductor algorithm used for raw gene expression data normalization.

5.6 Experiment 3 – Increasing interpretability of PC data

5.6.1 Goal

In this experiment, we analyzed microarray data derived from patients with prostate cancer (PC). Based on the encouraging initial results from experiments with increasing complexity, we stated here the following research question. Given tumor–related biological samples with different physiological profile, could we compare functional landscapes of gene expression between them?

We focused on two physiological conditions: state of primary prostate tumor tissue and state of metastatic prostate tumor tissue [CM10]. Based on that, we formulated two condition setups: normal vs tumor and tumor vs metas, where "normal", "tumor", and "metas" refer to gene expression in normal tissue, primary tumor tissue, and metastatic tumor tissue, respectively. In “normal vs tumor”, we confront gene expression between normal and primary tumor samples. In “tumor vs metas”, we confront gene expression between primary tumor and metastatic tumor samples.

Aside the normal pipeline proceeding, a new method of interpretability was used in this experiment, namely semantic clustering of Gene Ontology terms (described in detail in part 4.2.3).

Having Gene Ontology as prior knowledge source, we focused on the functionalities spanning two separate GO domains, namely Biological Process (BP) and Molecular Function (MF) [GOD]. This allowed us to to formulate four experimental setups:

- normal vs tumor, focused on MF domain
- normal vs tumor, focused on BP domain
- tumor vs metas, focused on MF domain
- tumor vs metas, focused on BP domain

83
In this thesis, we present preliminary results for all four setups. For BP domain, we present only brief comments. For MF domain, we present more detailed analysis that has been published in [ZSB\(^+\)12].

### 5.6.2 Sample characteristics

We used a GSE6919 [CMD\(^+\)07, YLJ\(^+\)04] “superseries” data set from GEO [GEO] repository. Biological samples were collected from both patients with prostate cancer and controls. Four separated groups of samples (“subseries”) were considered: from normal prostate tissue (free of any pathological alteration), from primary prostate tumor tissue, from metastatic prostate tumor tissue, and from normal prostate tissue adjacent to tumor. Gene expression was measured on three microarray platforms, namely: Affymetrix Human Genome U95B Array, Affymetrix Human Genome U95C Array, and Affymetrix Human Genome U95 Version 2 Array. For our purposes, we focused only on samples associated with Affymetrix Human Genome U95 Version 2 Array (identified in GEO repository as GPL8300\(^5\)).

For normal vs tumor experimental condition, we recognized 63 primary tumor samples and 80 control samples, giving 143 samples in total. For tumor vs metas experimental condition, we recognized 25 metastatic tumor samples and 63 primary tumor samples, giving 88 samples in total.

### 5.6.3 Experiment details

We conducted four separated pipeline runs, that is for normal vs tumor BP, normal vs tumor MF, tumor vs metas BP, and tumor vs metas MF. Each run was performed in the same way, for their respective data sets. For the sake of description, we present here a single run, namely normal vs tumor MF, in more detail.

**normal vs tumor MF run** After obtaining raw data set (GSE6919 [CMD\(^+\)07, YLJ\(^+\)04]) from GEO [GEO] repository, we extracted proper data for specific sample types considered in the run (here, primary tumor samples and control samples), and performed its pre-processing and normalization with GCRMA [WIG\(^+\)04]. After that, a 143 x 12625 gene expression data matrix (GEDM) was produced, and stored in the form of DSV file. Label information was generated as well and stored as DSV file. Platform annotation data set for Affymetrix HG U95A v2 microarray (GPL8300) was obtained from GEO repository, with Download full table... option\(^6\), and stored in the form of DSV file. The DSV files

were loaded into local experiment database Experiment.db, and local data integration was performed therein, as described in part 4.2.3.

The preparatory step of knowledge discovery phase resulted in generation of 2340 non-empty submatrices, each corresponding to specific GO term from MF domain.

For each submatrix, a binary classification problem, of discriminating disease from healthy status, was considered. During execution phase, the distributed calculations were performed in parallel environment. At the end of knowledge discovery phase, a classification error was estimated and a list of relevant variables for each submatrix, as described in part 4.2.4.

We focused further on the results obtained from \( \ell_1 \ell_2 \) algorithm. We considered as predictive only those submatrices that scored a classification accuracy higher than specified threshold, namely 70%. Regarding this, we collected a list of 67 meaningful GO terms that span MF domain.

Other runs For other runs, the basic run description stays the same. In this case, the differences are in experimental setup (that is, types of samples considered and GO domains considered), total number of non-empty submatrices generated, and between final GEDM dimensions. All basic differences are collected in Table 5.8. The results are presented separately in Table 5.9.

<table>
<thead>
<tr>
<th>Exp setup</th>
<th>GO domain</th>
<th>#Subm</th>
<th>GEDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal vs tumor</td>
<td>MF</td>
<td>2340</td>
<td>143 × 12625</td>
</tr>
<tr>
<td>normal vs tumor</td>
<td>BP</td>
<td>4088</td>
<td>143 × 12625</td>
</tr>
<tr>
<td>tumor vs metas</td>
<td>MF</td>
<td>2340</td>
<td>88 × 12625</td>
</tr>
<tr>
<td>tumor vs metas</td>
<td>BP</td>
<td>4088</td>
<td>88 × 12625</td>
</tr>
</tbody>
</table>

Table 5.8: Basic differences between different pipeline runs in Experiment 3. The columns are: experimental setup, considered GO domain, number of non-empty submatrices produced during run, Gene Expression Data Matrix dimensions: #samples×#variables (here, variables are probesets).

5.6.4 Results

The basic results for all pipeline runs are summarized in Table 5.9. Full lists of selected GO terms were omitted for clarity.

<table>
<thead>
<tr>
<th>Exp setup</th>
<th>GO domain</th>
<th>Class. acc. thr.</th>
<th>Sel. GO terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal vs tumor</td>
<td>MF</td>
<td>70%</td>
<td>67</td>
</tr>
<tr>
<td>normal vs tumor</td>
<td>BP</td>
<td>70%</td>
<td>117</td>
</tr>
<tr>
<td>tumor vs metas</td>
<td>MF</td>
<td>70%</td>
<td>171</td>
</tr>
<tr>
<td>tumor vs metas</td>
<td>BP</td>
<td>70%</td>
<td>279</td>
</tr>
</tbody>
</table>

Table 5.9: Summary of basic results obtained from Experiment 3. The columns are: experimental setup, considered GO domain, classification accuracy threshold, and number of selected GO terms.
Conventional plots  We have obtained several basic plots of nearest local neighborhood of selected GO terms from respective pipeline runs. The plot for normal vs tumor MF run is presented on Figure 5.10. The plot for tumor vs metas MF run is presented on Figure 5.11.

Since we have not reconstructed biological functionalities spanning Biological Function (BP) domain before, we obtained basic plot of nearest local neighborhood of selected GO terms over BP domain, presented on Figure 5.12. Selected terms came from normal vs tumor BP run. It can be observed that, because the sheer size of the plot, in line with the greater size of BP domain DAG subgraph in Gene Ontology [GOD], the plot itself does not provide any visual clues. This shows clearly the necessity to employ more sophisticated methods of functional interpretation.

Semantic clustering – example and interpretation  We present here the interpretation of functional landscape reconstructed by the pipeline, obtained with semantic clustering procedure. Basically, a set of GO terms is clustered according to metric developed specifically for GO DAG and its properties; the method is described in more detail in part 4.2.5.

We performed the clustering procedure for lists of selected GO terms obtained from all pipeline runs. We focus here on the specific result that easily displays the utility of the method.

The clusters of selected GO terms obtained from tumor vs metas MF run are presented on Figure 5.13.

Clusters 1, 2 and 3 (see legend on Figure 5.13 for reference) include GO terms involved in Binding of molecules to cell receptors (e.g. calmodulin, growth factor, insulin-like growth factor binding) or to other molecules inside the cell (e.g. nucleic acid, SH3 domain, NF-kappaB, and transcription factor binding). This conforms with the fact that the functions occurring in tumor cells are related to the binding of key molecules, such as calmodulin (that mediates several processes as metabolism, inflammation, intracellular movements, immune response), several growth factors (as the insulin-like growth factor binding that has been correlated with the risk of prostate cancer in a large longitudinal study [Sch07]), and other key molecules (involved in signaling pathways regulating the cytoskeleton, the Ras protein and the Src kinase). It may be noticed that these three clusters (1, 2, 3) are physically close to each other because of their semantic meaning.

The biggest cluster (7) is Enzymatic Activities, including GO terms such as, among others, methyltransferase activity, ATPase activity, oxidoreductase Activity and metalloendopeptidase Activity. The majority of these enzymes are related to the metabolism. Moreover, metalloproteinases are known to be fundamental for tumor invasion [Sch07].
On the right side of the Figure 5.13, one may observe the following clusters: Transporter Activities (9), that contains GO terms like calcium channel activity and lipid transporter activity, Structural Constituent (12), with terms like structural constituent of cytoskeleton, extracellular matrix structural constituent, and GTPase Activator Activity (10), containing enzymatic mechanisms like Rho guanyl-nucleotide exchange factor activity and small GTPase regulator activity. The involvement of several enzymatic functional classes suggests that all of them are fundamental to meet the needs of the aggressive tumor cells.

Summary  Semantic clustering, a method developed specifically according to Gene Ontology DAG properties, seems promising in enabling more detailed analysis of the output produced by the pipeline presented in this thesis. The current experience with it suggests few directions of possible future enhancements. First, to focus on visual aspects of the clustering, to involve more interactivity with the visual output. For example, static graph plot may be complemented by dynamic graph that can be interacted with, through panning, zooming, rotations etc. Second, to connect results from semantic clustering directly with literature extraction activities, that may result in more automatic obtaining of arguments needed for functionality verification (while still under human supervision). Overall, the technique seems promising and will be used in further experiments performed with the pipeline.

5.6.5 Final remarks

Results for this experiment were submitted for publication [ZSB+12]. Summary of basic experiment properties is presented in Table 5.10.

<table>
<thead>
<tr>
<th>Disease Characteristics</th>
<th>Prostate cancer (PC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disease Data Set</td>
<td>GSE6919</td>
</tr>
<tr>
<td>Control Data Set</td>
<td>CMD+07 YLJ+04</td>
</tr>
<tr>
<td>Platform Annotation Data Set</td>
<td>GSE6919</td>
</tr>
<tr>
<td>Microarray Type</td>
<td>GPL8300</td>
</tr>
<tr>
<td>Raw Data Normalization Used</td>
<td>Affymetrix HG U95A v2</td>
</tr>
<tr>
<td>RMA/aQM</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experimental Setup</th>
<th>normal vs tumor Primary tumor</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Disease” Tissue type</td>
<td>Healthy</td>
</tr>
<tr>
<td>“Control” Tissue type</td>
<td>63</td>
</tr>
<tr>
<td>Number of “Disease” samples</td>
<td>80</td>
</tr>
<tr>
<td>Number of “Control” samples</td>
<td>63</td>
</tr>
<tr>
<td>GO domain used</td>
<td>BP 70% MF 70%</td>
</tr>
<tr>
<td>Classification accuracy threshold</td>
<td>117 67</td>
</tr>
<tr>
<td>Number of GO terms selected</td>
<td>BP 70% MF 70%</td>
</tr>
</tbody>
</table>

Table 5.10: Summary of basic information for Experiment 3.
5.7 Assessing algorithmic stability with subsampling

5.7.1 Goal

One of the properties of computational methods that involve machine learning is their stability, that describes the behavior of the method when its input changes. In general, one should strive for using a stable method, where the results do not change dramatically when the input is to some extent modified [Vap98].

In order to explore the stability of the proposed methodology, we have performed a series of artificial experiments that involve subsampling [Vap98]. Here, some of the samples are excluded from analysis, and all the data associated with them are skipped. That is, having \( N \) samples \( S = s_1, s_2, \ldots, s_N \) and \( N \times P \) data matrix \( X \in \mathbb{R}^{N \times P} \) with learning data, we exclude \( i \) samples \( S'_i = s'_1, s'_2, \ldots, s'_i \), and we use \( N - i \) samples \( S \setminus S'_i \), as well as new \( (N - i) \times P \) data matrix \( X' \in \mathbb{R}^{(N-i) \times P} \), where columns associated with samples \( S'_i \) are removed.

We used data set GSE6919 [CMD+07, YLJ+04], already utilized in part 5.6, in tumor vs metas MF–focused setting, as base for subsampling. We considered two series of cases. In the first one (\( S\)-runs), we removed single sample three times (with replacement), thus obtaining 3 data subsets: S1,S2,S3. In second one (\( F\)-runs), we removed 1/4 of samples four times (with replacement), thus obtaining 4 data subsets: F1,F2,F3,F4. The cases are summarized in Table 5.11.

<table>
<thead>
<tr>
<th>Data subset/Run ID</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Removed samples</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Final number of samples</td>
<td>87</td>
<td>87</td>
<td>87</td>
<td>66</td>
<td>66</td>
<td>66</td>
<td>66</td>
</tr>
</tbody>
</table>

Table 5.11: Summary of data subsets used in artificial experiments for stability assessment with subsampling.

5.7.2 Experiments

Having Gene Ontology as prior knowledge source, we focused on the functionalities spanning single GO domain, Molecular Function (MF) [GOD]. For each data subset, a separated KDVS pipeline run was conducted, to obtain a list of meaningful GO terms. In total seven runs were conducted (S1–S3,F1–F4). In addition, for each data subset, we used \( \ell_1 \ell_2 \) on the whole data subset to obtain the list of meaningful GO terms. In total, seven such \( \ell_1 \ell_2 \)–runs were performed (LS1–LS3,LF1–LF4).

Below we present briefly single \( S\)-run and \( F\)-run, as well as the description of \( \ell_1 \ell_2 \)-run.
**S–run** After obtaining raw data set, we removed single random sample from consideration and performed the pre–processing and normalization as described before in part 5.6. After that, a gene expression data matrix (GEDM) was produced, and stored in the form of DSV file. Label information was generated as well and stored as DSV file. Platform annotation data set was obtained as before and stored in the form of DSV file. The DSV files were loaded into local experiment database Experiment.db, and local data integration was performed therein, as described in part 4.2.3. The preparatory step of knowledge discovery phase resulted in generation of submatrices that correspond to specific GO term from MF domain. For each submatrix, a binary classification problem, of discriminating disease from healthy status, was considered. During execution phase, the distributed calculations were performed in parallel environment. At the end of knowledge discovery phase, a classification error was estimated and a list of relevant variables for each submatrix, as described in part 4.2.4. We considered as predictive only those submatrices that scored a classification accuracy higher than specified threshold, namely 70%.

**F–run** After obtaining raw data set, we removed 1/4 random samples from consideration and performed the pre–processing and normalization as described before in part 5.6. The run proceeded as previously described for **S–run**.

**ℓ₁ℓ₂–run** For specific data subset, we performed binary classification on the whole data matrix with ℓ₁ℓ₂. There, we obtained the list of selected variables as described in [DMMTV09] [BMRV08], and mapped them into probesets specific for the microarray platform. Next, for such list, we performed GO enrichment analysis of member probesets with WebGestalt [ZKS05, DPZ10]. The enrichment was performed against the whole human genome, with significance threshold 0.05. As a result, we obtained a list of meaningful (significantly enriched) GO terms. We refer to this workflow (from binary classification to obtaining enriched GO terms) as **ℓ₁ℓ₂–run**.

### 5.7.3 Stability assessment

Each KDVS pipeline run, namely \{S1,S2,S3,F1,F2,F3,F4\}, as well as the corresponding ℓ₁ℓ₂–run, namely \{LS1,LS2,LS3,LF1,LF2,LF3,LF4\}, produced a list of meaningful GO terms. We transformed them into sets by removing repetitions. To investigate the algorithmic stability, we calculated the overlap percentages for chosen sets of terms.

Having \(k\) sets of GO terms \(T_1,T_2,\ldots,T_k\) (of possibly different length), we calculated the **overlap percentage** for them as:

\[
OVL(T_1,T_2,\ldots,T_k) = \frac{|\cap_{i=1}^{k} T_i|}{\sum_{i=1}^{k} |T_i|} \times 100\%
\]
Since the lists of meaningful GO terms produced with KDVS runs and \( \ell_1 \ell_2 \)-runs were obtained with different methodologies, they cannot be directly compared. To solve this issue, we calculated the overlap percentages for the subseries of runs performed by the same methodology. The subseries, along with corresponding overlap percentages, are presented in Table 5.12.

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Subseries</th>
<th>Overlap percentage</th>
<th>OVL value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDVS</td>
<td>S1,S2,S3</td>
<td>( OVL(S1, S2, S3) )</td>
<td>96.34%</td>
</tr>
<tr>
<td>( \ell_1 \ell_2 )</td>
<td>LS1,LS2,LS3</td>
<td>( OVL(\ell_1, \ell_2) )</td>
<td>65.00%</td>
</tr>
<tr>
<td>KDVS</td>
<td>F1,F2,F3,F4</td>
<td>( OVL(F1, F2, F3, F4) )</td>
<td>88.91%</td>
</tr>
<tr>
<td>( \ell_1 \ell_2 )</td>
<td>LF1,LF2,LF3,LF4</td>
<td>( OVL(\ell_1, \ell_2) )</td>
<td>38.98%</td>
</tr>
</tbody>
</table>

Table 5.12: Summary of subseries and corresponding overlap percentages used for stability assessment with subsampling.

We chose the overlap percentage as a simple method for investigating stability assessment of KDVS methodology. Through it, we performed indirect comparisons with classical methodology, represented by \( \ell_1 \ell_2 \)-based workflow that involves term enrichment. The values presented in Table 5.12 suggest that the new methodology may be indeed considered stable, but more thorough investigation is required to confirm such statement with confidence.

### 5.8 Summary

In this chapter we presented results obtained from computational experiments performed with the knowledge discovery pipeline devised in this thesis. First in part 5.2 we described the experimental procedure itself, and we compared its current implementation state to the design principles outlined earlier in chapter 4. Next we presented experimental data in chronological order of experiments performed. This order roughly follows the amount of functionality implemented and tested, starting from prototypical state and finishing in current state. As the first experiment, we presented in part 5.4 the early results that verified soundness of statistical concepts and basic correctness of overall experimental protocol. In second experiment, presented in part 5.5 the goal range was expanded to realize more sound research question. In third experiment, presented in part 5.6 the interpretability of the results was increased. Although the complete analysis of all the experimental outcome has not yet been finished, we demonstrated the potential of methodology devised in this thesis in solving real research problems, that appear at the edge of two distinct domains: life science and computer science. Our claims gain even more support from initial investigations of the algorithmic stability of new method, presented in part 5.7.
Figure 5.5: Nearest local neighborhood of selected GO terms, obtained in Experiment 1, plotted over MF domain. Selected terms are plotted in green.
Figure 5.6: Functional landscape of Alzheimer’s disease observed over time in Experiment 2, presented as nearest local neighborhood of involved GO terms from MF domain. Common terms are plotted in red. Specific terms for AD early stage are plotted in yellow. Specific terms for AD late stage are plotted in green.
Figure 5.7: Functional landscape of Parkinson’s disease observed over time in Experiment 2, presented as nearest local neighborhood of involved GO terms from MF domain. Common terms are plotted in red. Specific terms for PD early stage are plotted in yellow. Specific terms for PD late stage are plotted in green.
Figure 5.8: Functional landscape of early stage observed for both Alzheimer’s and Parkinson’s disease in Experiment 2, presented as nearest local neighborhood of involved GO terms from MF domain. Common terms are plotted in red. Specific terms for AD early stage are plotted in yellow. Specific terms for PD early stage are plotted in green.
Figure 5.9: Functional landscape of late stage observed for both Alzheimer’s and Parkinson’s disease in Experiment 2, presented as nearest local neighborhood of involved GO terms from MF domain. Common terms are plotted in red. Specific terms for AD late stage are plotted in yellow. Specific terms for PD late stage are plotted in green.
Figure 5.10: Functional landscape of gene expression data derived from primary tumor tissue confronted with normal prostate tissue (normal vs tumor MF run) in Experiment 3, plotted over MF domain. Selected GO terms are plotted in green.
Figure 5.11: Functional landscape of gene expression data derived from primary tumor tissue confronted with metastatic tumor tissue (tumor vs metas MF run) in Experiment 3, plotted over MF domain. Selected GO terms are plotted in green.
Figure 5.12: Functional landscape of gene expression data derived from primary tumor tissue confronted with normal prostate tissue (normal vs tumor BP run) in Experiment 3, plotted over BP domain. Selected GO terms are plotted in green.
Figure 5.13: Semantic clusters produced for selected GO terms from tumor vs metas MF run in Experiment 3, plotted as nearest local neighborhood over MF domain.
Chapter 6

Conclusions

In this concluding chapter we summarize the work presented in this thesis, comment on the experimental results, and discuss the future work.

6.1 Discussion of the proposed solution

In this thesis, we presented a concept of Knowledge Driven Variable Selection (KDVS) pipeline, as well as its prototype implementation, as a novel way to integrate data and knowledge. Here, data is numerical information, where knowledge is an information that express certain level of structural properties. By relating those kinds of information, we obtain the effect of integration that can produce more desired results, in terms of interpretability and usage during solving real research problems.

We focused on gene expression data, produced by microarrays, as the data source, and Gene Ontology graph structure, as the knowledge source. Gene expression data are used today in routine way in the “disease vs control” experiments, where gene expression is measured in two distinct biological samples, that express desired difference on phenotypic level. Gene Ontology (GO) can be defined as a hierarchical collection of biological and biochemical concepts, known as terms. Terms describe characterized molecular processes and interactions in the most precise way up to date, and their hierarchy reflects current state of understanding of those processes and interactions.

The way to relate data and knowledge, in this context, goes through genes. Since data contain gene expression measurements, each measurement is identified by probeset, and each probeset is associated with one or more known genes. The gene sequence, in turn, is associated to one or more GO terms by its functionality, i.e. the information what function in the cell is performed by the protein associated with gene in question. Such information
can be compiled from many sources, such as microarray annotations, offered by microarray manufacturer, or through GO Annotations collection.

Having forward relation probeset→term(s), it is possible to generate reciprocal relation term→probeset(s). This mapping is generated in local data integration, and becomes the foundation of data and knowledge integration.

By applying masking technique, and isolating only measurements specific for requested GO term, the original set of measurements is transformed into subsets, and the hierarchy taken from knowledge is introduced into the data.

At this stage, we can utilize machine learning/statistical learning approach of choice on each of the data subsets, to discover any hidden relationships within it. The setting used here is classification within supervised learning, where data elements are associated with two classes, in our case, “disease vs control” two phenotypic conditions of biological samples.

Within this setting, one can use one of the feature selection techniques, that, along with performing classification, also pinpoint some crucial data elements that can draw clear statistical difference between two classes present within data subset.

The output of the pipeline created here is twofold. At first level, having performed classification within supervised learning setup, we recognize data subsets that express statistically meaningful difference between two classes associated with their data elements. Since each data subset is associated with GO term, the result is the list of meaningful GO terms. At second level, we report the features selected during performing classification, in our case the features being genes.

Outcome on both levels can help in the interpretation of “disease vs control” experiments. First level output, that is, list of meaningful GO terms, can be used to deciphering and showing broader context of the experiment, by highlighting the meaningful biological functionality recorded in the gene expression data. Second level output, that is, selected genes obtained for each meaningful GO term, can be used in two ways. First, to assemble global gene signature, that allows clear separation of two distinct biological states, viewed through their gene expression. Second, to further follow biological role of significant genes, in order to confirm existing functional mechanisms and to check the possibilities of involving in new ones, if possible. In this context, the outcome of the pipeline can potentially be used in knowledge discovery.

Having output from both levels, it is possible to apply additional techniques on it, designed with the knowledge specificity in mind. One example is semantic clustering, devised for specificity of graph structure of Gene Ontology, that can be applied to list of meaningful GO terms and ease the interpretation burden. Those additional techniques may further increase the analysis of the outcome, especially for life science researchers.
6.2 Comments on the experimental results

In this thesis, we focused both on devising the concept of data and knowledge integration pipeline, and on producing working proof–of–concept implementation of said pipeline, that can be used with data coming from real experiments. Thus, we presented some experimental data and outcome obtained at the time of this writing.

6.2.1 Experiment 1

The experiment, described in part 5.4, was performed at the very early stage of pipeline development, as a primary verification test for the soundness of statistical concepts.

The goal was to reconstruct functional landscape of gene expression in Parkinson’s disease (PD), obtained through standard “disease vs control” experimental setup. We focused on functionalities spanning single Gene Ontology domain, Molecular Function (MF). At the end, it was shown that the most meaningful GO terms pointed by the pipeline are conforming to the standard activities associated with development of PD.

At this stage only preliminary functional analysis of output data was performed. To ease the interpretability of the results, we introduced here the plotting of nearest local neighborhood over GO DAG, as a convenient visual component to assess the relations between meaningful GO terms. Even at such early stage it could be observed that some level of topological correlation may exist between meaningful GO terms, as shown by the graph. That gave the impulse for using more sophisticated techniques later.

Overall, results obtained were convincing enough to continue the development.

6.2.2 Experiment 2

The experiment, described in part 5.5, was intended as a first full pipeline solution to the complex problem of comparing gene expression for closely related diseases, Alzheimer’s disease (AD) and Parkinson’s disease (PD).

The goal was to compare functional landscape between the diseases, as well as finding specific functionality for each of them. To do this, we introduced “early” and “late” time points of disease development, and we formulated four individual “disease vs control” experimental setups, in the form of runs. As before, we focused on Molecular Function (MF) related functionality. During each run, we reconstructed functional landscape for specific disease at specific development time point. The individual comparison was performed on the pair of selected outcomes of specific runs.
For reconstructing functional landscape of Alzheimer’s disease (AD early vs AD late), we noticed considerable overlap between the two stages. This can be attributed to the fact that, for this experiment, we treated intermediate stage as early one, that cannot be distinguished on the gene expression level from the late one. In the case of AD, there do exist data sets that contain gene expression data for even earlier phase of disease development, e.g. GSE1297 [BGC+04], GSE28146 [BBP+11]. However, they were not used here, because either number of disease samples was insufficient, or microarray platform type was incompatible. Regarding this, it is possible for this part of the experiment to be repeated with different data set representing early development of AD.

For reconstructing functional landscape of Parkinson’s disease (PD early vs PD late), a clear distinction between two stages can be observed. Also, one can observe the gain in the molecular functionality that has been “activated” in later stage. The consequence of this is being currently verified elsewhere. The distinction suggests rational choice of data sets used here.

For comparing early development stages of diseases (AD early vs PD early), AD–related functionality spans considerable amount of the overall results. This may be attributed to the same fact of treating intermediate stage as early one. Regarding this, it is possible also for this part of the experiment to be repeated with updated data set.

For comparing late development stages of diseases (AD late vs PD late), the molecular functionality for both diseases seems “balanced”, in the sense that no particular term abundance can be observed for any side. This may confirm the fact that both diseases are affecting the same tissues and alter similar molecular processes.

The complete outcome is currently being produced, therefore the experiment has not yet been fully concluded. However, even at this stage it may be noticed that greater interpretability of the results may be achieved by considering more visual–oriented data interpretation procedures. That confirmed the rationale gained from Experiment 1, and this direction was explored in the next experiment.

### 6.2.3 Experiment 3

The experiment, described in part [5.6], was intended as a next logical step in the progress of pipeline development. It involved stating the complex research problem of comparing gene expression for different prostate cancer–related samples, as well as providing extended interpretation of the results.

The goal was to compare functional landscape between the specified prostate cancer (PC) cases, namely primary tumor cases and metastatic tumor cases.

We used rich data set to formulate two primary comparisons within “disease vs control”
experimental setup, namely normal vs tumor and tumor vs metas (see experiment description for more details). Also, we explored for the first time, two domains of Gene Ontology, namely Biological Process (BP) and Molecular Function (MF), thus creating four distinct runs. During each run, we reconstructed functional landscape for specific conditions focusing on specific GO domain.

We focused further on tumor vs metas MF run, applying for the first time the semantic clustering technique, in a proof–of–concept manner, to explore the potential gain in the interpretability. Results obtained so far are promising, and more detailed analysis is currently being produced.

We also obtained some conventional visualization artifacts to be presented with this thesis. As the previous one, the experiment has not yet been fully concluded. At this point, however, the current proof–of–concept implementation of the pipeline seems to be stable enough to be used with further problems.

6.3 Future work

Even though the proof–of–concept implementation of the pipeline has reached stable state in terms of usability, there is still plenty of open possibilities for improvement, both in terms of implementation and methodology.

6.3.1 Short–term goals

To enhance feasibility of the statistical output, work is ongoing to make more balanced data subsets in terms of statistical soundness. The problem is described in more detail in part 4.1.3.1. In short, very often gene product associated to child GO term, is not associated with parent GO term, contrary to one would expect from hierarchical structure. This results in unbalanced data subsets being produced with masking, which may heavily affect statistical classification results. We would like to restore this property by hand and implement it in the existing proof–of–concept as soon as possible.

To increase application integrity and management, work is ongoing in revising components implemented in R, in sense of both for migrating to Python (like nearest local neighborhood (NLN) plotting, and semantic clustering), or for more tight coupling (like raw data processor). This activity in general depends on the factors that remain beyond the direct scope of the pipeline itself. For example, to plot NLN graph, the minimal subgraph must be constructed first, based on the Gene Ontology structure, and this procedure implemented from scratch. Likewise, to perform semantic clustering, various immediate metrics must be
gathered over Gene Ontology structure, and core procedure re-implemented as well. The situation is even more complicated here, since Python is not considered a common language for bioinformatics, where C/C++, Java and Perl dominate the space in general [FG08], and R is considered an informal standard in microarray-related bioinformatics, thanks to rich capabilities of BioConductor package repository [Dra11]. On the other hand, Python is well-known in scientific computing environment [Oli07], thanks to the numpy and scipy libraries of numerical and scientific-oriented procedures This includes, in particular, machine learning community. Therefore, it seems reasonable to develop as much tight integration of various pipeline software artifacts around Python as possible.

To enhance the general usability of the prototype implementation, work is ongoing to lower memory requirements by exploring new storage management methods. The problem is explained in more detail in part 4.1.3.3. The issue is related to maintaining the balance between flexibility of textual information and volume of binary data structures, heavily utilized by dependent Python numerical packages. Currently various approaches are being evaluated.

6.3.2 Long-term goals

The long-term goals conform to the issues presented in part 4.2.6 in which the overview of the possible extensions of the pipeline was discussed. We would like to follow them afterwards when short-term goals are completed.

6.3.3 Additional remarks

Potential usage in network inference approaches

Taking into account the parallel between the workflow of KDVS and the enrichment analysis (see Figure 4.3 for reference), it is theoretically possible to replace the enrichment analysis procedure with KDVS, in any context it may occurs. For example, when identifying discriminant pathways from microarray gene expression data with machine learning [BJV+11], a feature selection, followed by pathway enrichment, is performed before subnetwork inference and subnetwork comparison steps. In theory, it is possible to replace pipeline components of feature selection and pathway enrichment with KDVS, and to present adapted KDVS output, among others, as input to specific network inference algorithm [BBA07]. The full discussion of such case goes beyond the scope of this thesis.

1http://www.scipy.org/
Potential usage for next generation sequencing (NGS) data

There exist other sources of gene expression measurements, such as direct mRNA sequencing or sequence tag-based approaches, that can produce massive amounts of data with next generation sequencing (NGS) high–throughput technologies [AC97, MWM^+08, HLSK^+11]. In theory, it could be possible to use such data as an input to KDVS, as long as the proper mappings between expression measurements and prior knowledge concepts can be devised (see 4.2.4.1 for more details). The full discussion of such case goes beyond the scope of this thesis.

Similar approach devised in parallel

The idea of transforming initial monolithic gene expression data set into smaller ones, according to prior knowledge, was used independently by Maglietta et al. [MPC^+07], to devise a statistical methodology for identifying functional categories of genes deregulated in pathological conditions.

Here, the initial gene expression data matrix (GEDM) is transformed into smaller submatrices, according to mapping microarray probesets into genes associated with Gene Ontology terms spanning Biological Process (BP) domain. Next, statistical classification is performed on smaller submatrices with Regularized Least Squares (RLS). In addition, multiple cross validation is performed, error rate of predictor is devised, and two permutation tests are used to assess the statistical significance of obtained error rate, along with subsequent false discovery rate (FDR) estimations for multiple hypothesis testing. The p–values from permutation tests, as well as FDR values, are used to select significant processes that appear deregulated. Similarities and differences between this approach and KDVS are listed briefly in Table 6.1.

<table>
<thead>
<tr>
<th>Maglietta et al.</th>
<th>KDVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation of GEDM into smaller submatrices</td>
<td></td>
</tr>
<tr>
<td>Using Gene Ontology as transformation guide</td>
<td></td>
</tr>
<tr>
<td>Focus on BP domain</td>
<td>Ability to utilize BP,MF,CC domains</td>
</tr>
<tr>
<td>Mapping of probesets to BP genes not characterized in detail</td>
<td>Mapping of probesets to domain genes devised as reproducible and extensible procedure</td>
</tr>
<tr>
<td>Statistical classification performed for each submatrix</td>
<td></td>
</tr>
<tr>
<td>RLS as classification method</td>
<td>ℓ₁ℓ₂/RLS as classification method</td>
</tr>
<tr>
<td>N/A</td>
<td>Feature selection performed</td>
</tr>
<tr>
<td>Predictor error rate derived</td>
<td></td>
</tr>
<tr>
<td>Variable error rate threshold</td>
<td>Constant error rate threshold</td>
</tr>
<tr>
<td>Permutation tests for statistical assessment</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 6.1: Brief summary of similarities and differences between the methodology of KDVS and methodology presented by Maglietta et al.
Since both approaches were based on different statistical methodologies, it is not desirable to compare results directly using numerical values [GB07]. For results involving prior knowledge, however, a comparison may be performed against known reference knowledge. For example, when identifying significantly deregulated functional categories of genes that are connected to particular disease, one can compile a reference list of functional categories known to be involved with high confidence in the development of the disease. Such reference knowledge can be compiled from the sources of manually verified evidences, such as KEGG PATHWAY/DISEASE databases [KGS+12].

Currently, a work is ongoing to devise automated and extensible methods to compile such reference knowledge from reliable sources, and to use them in comparisons between results produced by KDVS and signature enrichment methods. With such procedures, it is possible to perform detailed comparison with methodology of Maglietta et al. At this moment, such comparison goes beyond the scope of this thesis.
Appendix A

Software libraries

A.1 KDVS software pipeline

The software pipeline in the current state comprises of core Python package, that provides basic functionality and controls local data integration, and two monolithic Python scripts that perform separate parts of actual pipeline flow. Both scripts access core package through public API. This design was dictated by the conceptual development of the pipeline, where data integration part has been established relatively quickly, and statistical and parallel computational concepts were evolving too dynamically to maintain cleanly separated functionalities.

The software pipeline is currently under the review process to be published as open source.

A.1.1 Core Python package

We present here the core Python package (kdvs.core) that provides the basic functionality for KDVS software pipeline. The main package contains a series of subpackages, that in turn contain modules; each module encapsulates a portion of functionality. We enumerate all modules, and we list their public classes and objects (with italics), together with public class methods and functions (with monospace), along with short descriptions.

a) kdvs.core.GO.GEDM

Provides functionality for manipulating gene expression data matrix (GEDM).

get_GEDM_probesets

Get all probesets associated with GEDM.
get_GEDM_rows
Get rows of expression values associated with given probesets, filter columns according to samples if needed.

get_GEDM_samples
Get all samples associated with GEDM.

b) kdvs.core.GO.GOTermTree
Provides functionality for parsing RDF-XML release of Gene Ontology data into GO term tree (effectively decomposing DAG graph into set of \{ancestor \rightarrow child\} relations).

parse_go_rdf_xml
Parse RDF-XML file provided by GO that contains descriptions of GO terms, including terms hierarchy.

c) kdvs.core.GO.GOTermTreeManip
Provides functionality for manipulating Gene Ontology term tree.

collect_immediate_children
Given GO term, collect all of its immediate children. Immediate children of term X are GO terms related directly with “is_a” or “part_of” relations with X.

collect_subtree_terms
Given root term, collect all terms from rooted term subtree into a set.

collect_subtree_terms_with_depth
Given root term, collect all terms from rooted term subtree into a set, as well as mapping of numerical depth level to list of terms at that level (\{depth level \rightarrow terms\}).

subtree
Given tree of GO terms, execute given callback function on the requested term on the given level of the GO term subtree. Proceed with recursive calls for every child of the requested term. Callback function accepts the requested term, list of children of the requested term, and current level depth. Any additional arguments are passed to callback function through variable argument list.

d) kdvs.core.GO.HGNC
Provides functionality for manipulating gene naming-related data.

create_probeset2gene
Create derived table probeset2gene based on imported DSV data.

get_geneid
Get gene naming-related data for requested GEDM probeset.

get_probeset2geneid
Get mapping of GEDM probesets to gene-naming related data.

\(^1\)http://www.genenames.org/
e) kdvs.core.GO.annotation
Provides functionality for manipulating specific annotation data for microarrays.

- create_term2probeset
  Create derived table term2probeset based on imported DSV data.
- get_ns_terms
  Get all GO terms available for requested GO domain (also referred to as namespace).
- get_probesets
  Get all probesets available through term2probeset mapping.
- get_term2probeset
  Get mapping of GO terms to GEDM probesets for requested GO domain.
- get_term2size
  Get counts of associated probesets for all GO terms available for requested GO domain \(\{\text{term} \to \#\text{probesets}\}\).

f) kdvs.core.GO.subm
Provides functionality for extraction of GEDM submatrices and handling labels.

- get_labels
  Get mapping of samples to labels.
- get_subm_for_term
  Get submatrix of GEDM values for requested GO term. Submatrix is returned as list of rows of expression values.

g) kdvs.core.config
Provides configuration handling.

- evaluate_cfg_file
  Evaluate requested KDVS configuration file, and return all configuration variables. The variables will be later put among shared variables of execution environment. If requested, also parse default configuration file with the rule that user variables overwrite default ones.
- get_default_GO_termdb_release
  Return full path for default GO release RDF–XML file.
- get_default_cfg_file_path
  Return full path for default KDVS configuration file.
- get_default_internal_data_root_path
  Return full path for directory with KDVS internal data.
- get_default_R_data_root_path
  Return full path for directory containing default R local environment.

h) kdvs.core.db
Provides layer for all database operations performed by KDVS.
**KDVSDb**

General manager of all DB operations performed by KDVS. Currently configured specifically for SQLite3, it provides: handling of meta-database that contains information of all used single databases (also referred to as *tablespaces*), handling of multiple tablespaces, including opening, closing, and copying/moving tables between them, as well as automatic data import from DSV files.

- **close**
  Close opened tablespaces handled by this manager.

- **copy_table**
  Copy/move database table between two existing tablespaces.

- **get_tmpdb**
  Return handle for requested tablespace; if tablespace does not exist, create it.

- **get_tmpdb_loc**
  Return path for file containing requested tablespace.

- **load_csv**
  Import data from DSV file to a table in requested tablespace.

**probeset2gene_schema**

Default DDL schema for *probeset2gene* table.

**term2probeset_schema**

Default DDL schema for *term2probeset* table.

---

1) **kdvs.core.error**

Provides generic error handling.

**KDVSError**

General KDVS exception.

**KDVSWarning**

General KDVS warning.

---

2) **kdvs.core.execenv.execenv**

Provides functionality of encapsulated execution environment for small dependent computational tasks.

**ExecEnv**

Implements basic execution environment, where set of actions (separated computational tasks) is executed in order, and common set of environment variables (environment state) is available for any action to store and retrieve data.

- **add_action**
  Add new action to the environment and schedule for execution. Actions are executed in the add order. Action is given as a function with its arguments.

---

2http://www.sqlite.org/
add_var
Add new variable to execution environment, with possible replacement.
Variables are managed in a standard Python dictionary, therefore all rules
for adding items to normal dictionaries apply here as well.

clear_actions
Clear all added actions.
del_var
Remove requested variable from execution environment. If variable does not
exist, do nothing.
execute
Execute all added actions so far in add order (FIFO). When any action
throws an exception during its run, the whole execution is stopped and
diagnostic information is returned.
format_action_spec
Return textual representation of action.
update_vars
Add all variables from requested dictionary to execution environment. See
add_var for details. Variables are added without replacement.
var
Retrieve value of variable present in execution environment.
varkeys
Retrieve keys of all existing variables present in execution environment.

LoggedExecEnv
Implements logged execution environment, where global logger is provided for
any action to use, according to specified configuration. Logger must be initialized
outside and passed in configuration variables. The logger is available through
shared environment variable “logger”.

execute_all
Execute all actions; if any action fails, log the diagnostic information.

k) kdvs.core.execenv.pplus_env
Provides functionality of parallel execution environment, based on PPlus.

PPlusExecEnv
Implements logged PPlus execution environment, according to specified config-
uration. The logger is available through shared environment variable “logger”.
PPlus connection is available through shared environment variable “pplus_connection”.

deserialize_pzp_from_filekey
Deserialize object from requested PPlus remote file.
pprint_to_filekey
Serialize textual representation of given input object to PPlus remote file.
serialize_pzp_to_filekey
Serialize given input object as PPlus remote file.

serialize_txt_to_filekey

Serialize given sequence of strings to PPlus remote file.

l) kdvs.core.metadata.annotation_metadata
   Provides various “metadata” (omitted for clarity) for handling DSV files with non-numerical, heavily parseable content, such as microarray platform annotations, HGNC data etc. Metadata is used when DSV files are processed, e.g. during import their content into database, or parsing individual lines.

m) kdvs.core.metadata.experiment_metadata
   Provides various “metadata” (omitted for clarity) for handling DSV files with numerical-oriented content, such as GEDM and labels information. Metadata is used when DSV files are processed, e.g. during import their content into database, or parsing individual lines.

n) kdvs.core.metadata.go_metadata
   Provides various “metadata” (omitted for clarity) for handling GO data. Metadata is used when RDF–XML release is being parsed, and information parsed from it is being consulted later.

o) kdvs.core.provider
   Contains set of useful providers for various generic classes of objects handled by KDVS.

   KDVSMetadata
   Abstract class for KDVS “metadata” instances. It is used to transmit particular metadata to specified method or function.

   db_provider
   Abstract provider of Python DB API 2.0. This class is used to specify some driver-specific DB functionalities in KDVSDB. In particular, this class provides proper “connect” functionality, as well as driver-specific SQL data types and error handling.

   file_provider
   Return proper file opener for given file path, suitable for use with context manager, regardless of file type. Provides transparent handling of compressed files.

   fpBzip2File
   Wrapper class to allow opening files compressed with “bzip2” from within context manager.

   fpGzipFile
   Wrapper class to allow opening files compressed with “gzip” from within context manager.

   sqlite3_db_provider
   Concrete subclass of db_provider for SQLite3 database back-end.

   sqlite3_provider_cfg
This class holds configuration parameters for SQLite3 DB provider.

**p) kdvs.core.rint**

Provides simple layer for integration with R environment.

- **R**
  
  Return global R interpreter instance.

- **Rcall**
  
  Execute given R statement(s) in the context of current R environment.

- **Rimport**
  
  Import given R package “as is”, i.e. without additional package parameters, in the context of current R environment.

- **Robj**
  
  Return global R environment, including all R objects available through it. Used to access R constants and R built-in functions.

**q) kdvs.core.util**

Provides various utility classes and functions (omitted for clarity).

### A.1.2 l1l2_experiment.py

The `l1l2_experiment.py` script performs the pipeline flow down to (including) knowledge discovery phase, as shown on Figure A.1. It performs all the functionality described in part 4.2.4, using all needed core functionalities through API to accomplish it.

![Figure A.1: The amount of functionality implemented in KDVS pipeline that is performed by l1l2_experiment.py script.](image)

### A.1.3 l1l2_postprocess.py

The `l1l2_postprocess.py` script performs the finishing part of pipeline flow, that is post-processing phase, as shown on Figure A.2. It performs all the functionality described in part 4.2.5, using all needed core functionalities through API to accomplish it.

---

3 [http://www.r-project.org/](http://www.r-project.org/)

114
A.2 L1L2Py

$L1L2Py$\footnote{http://slipguru.disi.unige.it/Software/L1L2Py/index.html} is an implementation of $\ell_1\ell_2$ feature selection technique, presented in [BMRV08, DMDVR09, DNIMTV09]. The method is also discussed briefly in part 3.3. The technique consists of two stages.

During Stage I, the minimal set of relevant variables is identified, in terms of prediction error. More specifically, optimal values of parameters $\tau_{opt}$ and $\lambda_{out}$ are selected within a k–fold cross validation loop for a small, fixed value of parameter $\mu$.

During Stage II, for fixed optimal values $\tau_{opt}$ and $\lambda_{out}$, related sets of relevant variables are identified, for increasing values of the correlation parameter $\mu$.

KDVS uses the wrapper procedure l1l2py.model_selection that executes both stages sequentially. It is used with each external split of sufficiently large data submatrix, as described in parts 4.2.4.1 and 4.2.4.2.

$L1L2Py$ also contains an implementation of Regularized Least Squares algorithm [Tib94]:

$$\beta^* = \arg\min_{\beta} \left\{ \frac{1}{n} \| Y - X\beta \|_2^2 + \mu \| \beta \|_2^2 \right\}$$

KDVS uses l1l2py.algorithms.ridge_regression for $\mu = 0$, with each sufficiently small submatrix, as described again in parts 4.2.4.1 and 4.2.4.2.

The L1L2Py implementations of $\ell_1\ell_2$ and $RLS$ are used as “feature selection” methods in knowledge discovery phase. Note that $RLS$, unlike $\ell_1\ell_2$, is not a feature selection technique in the sense presented in chapter 8, yet it still takes part in obtaining properly selected variables during frequency verification step of post–processing, as discussed in part 4.2.5.1.

The L1L2Py library has been published as open source.
A.3 PPlus

PPlus is an ad hoc implementation of simple environment to execute Python code in parallel on many computers. It was designed to facilitate data transport over distributed environment, by exposing a simple interface while handling details in the background. Also, it allows to isolate code execution, and all data used and/or produced during it, within experiments, so that one machine can participate in many such experiments.

PPlus is based on Parallel Python (PP), a simple yet effective framework for parallel execution of Python code. Parallel Python has provided core functionality of extracting Python code, maintaining communication over set of machines, transporting data over the network, and maintaining any parallel jobs running.

A.3.1 Environment

The environment controlled by PPlus, inherited directly from Parallel Python, comprise a set of machines (also called nodes) that offer their resources to execute assigned parallel jobs (also called tasks). Each node runs the server process in the background, which provides visibility over the local network to other nodes. That process also maintains all data transfer activities.

Each server process offers certain number of slots that will handle incoming parallel jobs. For example, server process that runs on machine with 4 CPU cores may offer 3 slots to handle parallel jobs. In this case, up to 3 parallel jobs may run simultaneously on that machine. The remaining one runs server process itself, as well as may perform other activities.

The Python code to be executed by PPlus can be categorized in the following manner. A worker code is distributed over the network to the nodes to be executed there. It produces partial results that are to be collected later. A master code actually distributes the worker code, then collects all partial results and produce final results.

Both worker code and master code can do any computations, import modules (possibly with some restrictions), and produce files.

The code to be executed in parallel, composed of master code and worker code, needs to be placed on one of the nodes. The server process that runs master code is designated as master process. For this code execution, other server processes are seen by master process as worker processes. The master process distributes all parallel jobs (i.e. worker code) to worker processes, collect partial results and produces final results.

5http://slipguru.disi.unige.it/Software/PPlus/index.html
6http://www.parallelpython.com/
In this setting, we can imagine some possible configurations. One possible option is to have a dedicated machine that performs a constant role of master node for subsequent experiments, and controls local group of “subordinated” machines that perform only “worker” role. Another possibility is to have all local group of machines offering slots for parallel jobs, effectively becoming fully interconnected, and each machine can perform both “master” and “worker” role. The first setting, being more centralized, may be preferred for computationally intensive experiments. The second setting, being more distributed, almost in peer–to–peer fashion, may be preferred for less computationally demanding, but more frequent experiments that can run in parallel.

Note that server process can control many independent code execution events. For example, we can start master code more than once on the same machine; in that case, the two runs are considered different code execution events. We can expand this concept to the point that any machine that runs a server process and is visible over local network, can effectively perform both roles of “master” and “worker” machine at the same time. Each code execution event is treated as separate experiment.

Finally, note that the concept of master and worker processes can be also achieved for single machine with more than one CPU core. In that case, master process is controlled by one CPU core, while worker processes are controlled by the specified amount of remaining CPU cores. This setting allows us effectively to debug applications written for using PPlus.

### A.3.2 Experiments

Internally, PPlus uses the concept of experiments to organize the code and data. The “experiment” consists of the code that performs a specific task, including pieces to be executed in parallel (i.e. master code and all worker code), as well as all regular files produced by that code. A single instance of the worker code, submitted for remote execution, is also called worker task or worker job. We will refer to all the Python code within an experiment as experiment code.

Master and/or worker code may produce an external file that must be accessed by any other code, for example file produced by one job must be accessed by another one. To ease maintenance related to this need, and in the same time to ease running of multiple subsequent experiments over the network, the experiment code can use a shared file system resource to store files produced during execution, and to access them back if needed. This functionality is controlled by PPlus in a transparent way, and it is exposed through an API.

The experiment code can access and store any remote files in a dedicated experiment directory, created specifically for that purpose on the shared disk resource. Both master code and all worker code have an access to the experiment directory. The files produced
by different experiments are physically separated, and as a result, many experiments can run simultaneously without data corruption.

### A.3.3 Storage management

*Shared storage resource* is any physical storage (hard disk, RAID device etc.) that is visible over the network through single mount point. Each participating machine mounts the same location on the same device. Experiment code creates *experiment directory* there, in dynamic way, when the experiment is started with execution of master code. The information of experiment directory is then passed down to each parallel job that is issued within this experiment, so a single job can access files in the storage, and store all the files produced by itself.

Also, when the new experiment is started, each participating machine is maintaining its own *local cache*. For example, a parallel job may access shared storage, but it needs to produce some *intermediate files* that do not need to be transferred over the network; in that case, a job can maintain such files in local cache. Local caches are created in dynamic way, and are isolated between experiments. More precisely, if any machine starts participating in any experiment in any way (that is, by becoming master or worker), the local cache specific for the experiment is created there.

Files are accessed, in shared storage and in local cache, by *keys*, not by physical file system paths. Each key is a string that uniquely identifies the file. Within the experiment, all *file keys* should be unique to avoid unwanted data corruption.

Such separation, both on shared storage level and local cache level, allows machines to participate in more than one experiment at a time.

A general schema of all the relations between machines participating in PPlus activities is presented on Figure A.3.

### A.3.4 Implementation

PPlus uses instances of *PPlusConnection* Python class to handle all the operations; this class provides the public PPlus API.

First, master code creates *master instance* of *PPlusConnection* class. During creation of master instance, *experiment ID* is granted, *experiment directory* is created, and *local cache* associated with the experiment is created on that machine. The master code can use public API to upload/download files to/from shared storage, create temporary files in local cache, and issue parallel jobs to worker processes, if any of them is visible over the network. The master instance also handles, in background, the communication between
server processes, the monitoring of jobs execution, and the collection of any partial results that may be returned from jobs. There is no specific need to “close” the PPlusConnection; the experiment ends simply when master code collects all partial results and finishes its execution.

When any parallel job is sent to be executed on particular machine, the worker code creates worker instance of PPlusConnection class. This instance is informed of the current experiment ID, experiment directory, and also maintains the local cache that is dynamically created. Note that the local cache is created when the first parallel job is issued to a participating machine; every subsequent parallel jobs from the same experiment are reusing the same local cache. The worker code, then, can use public API in the same way as master code. It can also return partial result to master code, i.e. to indicate any failures during job execution.

When a parallel job signalizes any failure, and there is a need for the job to be repeated, it may be re-submitted up to certain number of times, if requested.

### A.3.5 Summary

The PPlus library was used for performing all of the experiments described in this thesis, and now it is being routinely used for other experiments as well. The PPlus library has been published as open source.
Bibliography


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Journal</th>
<th>Volume, Issue, Pages, Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CSWE05]</td>
<td>PJ Carrington, J Scott, and S Wasserman (Eds.)</td>
<td>Models and Methods in Social Network Analysis (Structural Analysis in the Social Sciences)</td>
<td>Cambridge University Press</td>
<td>2005</td>
</tr>
<tr>
<td>[DHH+83]</td>
<td>RF Doolittle, MW Hunkapiller, LE Hood, SG Devare, KC Robbins, SA Aaronson, and HN Antoniades</td>
<td>Simian sarcoma virus onc gene, v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor</td>
<td>Science</td>
<td>221(4607):275–277, 1983</td>
</tr>
<tr>
<td>[DSH+03]</td>
<td>G Dennis, B Sherman, D Hosack, J Yang, W Gao, H Lane, and R Lempicki</td>
<td>DAVID: Database for Annotation, Visualization, and Integrated Discovery</td>
<td>Genome Biology</td>
<td>4(9):R60, 2003</td>
</tr>
</tbody>
</table>


From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 17:37–54, 1996.


[PDO] Parkinson Disease, Late-Onset at OMIM. URL: [http://www.omim.org/entry/168600][cited Jan 19, 2011].


127


[UGB] UCSC Genome Browser Home. URL: [http://genome.ucsc.edu](http://genome.ucsc.edu) [cited Oct 20, 2011].


