ModuleInducer: automating the extraction of knowledge from biological sequences

by

Oksana Korol

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.Sc. degree in
Computer Science, Bioinformatics Option

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

In the past decade, fast advancements have been made in the sequencing, digitalization and collection of the biological data. However the bottleneck remains at the point of analysis and extraction of patterns from the data. We have developed a method that is aimed at widening this bottleneck by automating the knowledge extraction from the biological data. Our approach is aimed at discovering patterns in a set of DNA sequences based on the location of transcription factor binding sites or any other biological markers with the emphasis of discovering relationships. A variety of statistical and computational methods exists to analyze such data. However, they either require an initial hypothesis, which is later tested, or classify the data based on its attributes. Our approach does not require an initial hypothesis and the classification it produces is based on the relationships between attributes. The value of such approach is that is is able to uncover new knowledge about the data by inducing a general theory based on basic known rules.

The core of our approach lies in an inductive logic programming engine, which, based on positive and negative examples as well as background knowledge, is able to induce a descriptive, human-readable theory, describing the data. An application provides an end-to-end analysis of DNA sequences. A simple to use Web interface accepts a set of related sequences to be analyzed, set of negative example sequences to contrast the main set (optional), and a set of possible genetic markers as position-specific scoring matrices. A Java-based backend formats the sequences, determines the location of the genetic markers inside them and passes the information to the ILP engine, which induces the theory.

The model, assumed in our background knowledge, is a set of basic interactions between biological markers in any DNA sequence. This makes our approach applicable to analyze a wide variety of biological problems, including detection of *cis*-regulatory modules and analysis of ChIP-Sequencing experiments. We have evaluated our method in the context of such applications on two real world datasets as well as a number of specially designed synthetic datasets. The approach has shown to have merit even in situations when no significant classification could be determined.

Acknowledgements

This work would not have been possible without the help and support of many wonderful people. I would like to thank my supervisor, Marcel Turcotte for his deep and versatile knowledge in bioinformatics and his eager willingness to share it with me. Your patience and guidance has helped me navigate the sea of exciting research directions and find true inspiration, which I hope to follow in the future. Thank you for making these two years not only informative and interesting, but truly fun!

To my darling husband Sergiy - your encouragement and support in this endeavor has helped me find who I am, build my confidence and become deeply happy and content. Thank you for this! (Your love I take for granted :). Also thank you to my parents for raising me to have no second thoughts about education and hard work.

I am grateful to all the inspiring mentors I had during my studies. Stéphane Aris-Brosou for his meticulous attention to detail and for being my "social network hub" and introducing me to my future supervisor, my friends and most of the bioinformatics faculty. Thank you to Stan Matwin for introducing me to all the inspiring machine learning topics, for interesting ideas for this work, and for organizing all those informative Text Analysis and Machine Learning Group (TAMALE) meetings. Thank you to Lucia Moura for helping me get my footing back with combinatorics, recursion and all the other significant topics of combinatorial algorithms. Also thank you to Nathalie Japkowicz and Mohak Shah for offering their great ideas on the evaluation of our method.

I would also like to thank Marjorie Brand and Jeffrey Dilworth for stimulating discussions and thank you, Marjorie, for sharing your data with us.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Since the first mapping of the human genome in 2001 (Human Genome Project [13]), great advancements in genetic science have been predicted, including predictions of genetic predisposition to diseases and personalized genetic medicine [9]. However, 10 years past and with wider availability of individual genome screening, these predictions are yet to be fulfilled in entirety. One of the reasons for this is that the step from obtaining the data to fully understanding it is much more complex and takes much longer than anticipated. The scientific development in this area, however, did not stand still, and the new mechanisms of gene regulation are being discovered and our understanding of it becomes more and more complex. Coupled with a huge amount of experimental data generated every day, the task of discovering new laws of interactions in gene transcription becomes a real challenge. Analyzing it by hand, and even using exhaustive automated methods are often unfeasible, due to the volume of data. Instead, machine learning techniques, such as Support Vector Machines (SVM), Bayesian Classifiers, and others have been shown to produce good results. However these methods assume that each data object is presented as a vector of attributes and all data is stored in a single relation. This is often not the case when gene transcription is concerned, since biological data often contains complex structural elements. Discovering these structural elements remains a formidable challenge, which often falls to the human expert. Our main motivation for this work is to develop a method that can discover new complex relationships in the biological data.

Another challenge, facing the experts in molecular biology, is understanding and be-

ing able to experimentally verify the conclusions of the complex data mining analysis, provided by a data mining technique. Very often the rules underlying the resulting classification are not reported or, as is the case with the SVM, are very difficult to understand. This makes it hard to design an *in vivo* or *in vitro* experiment to confirm the findings and therefore becomes an obstacle in data acquisition. Our next motivation is to create a method, results of which could be easily interpreted by a non-expert in machine learning.

Our last motivation is to make our method easily accessible and easy to use. A large number of algorithms, which can be useful in solving biological problems, never make it to the experts in biology, because they require complex installation and have a steep learning curve. With this work we intend to provide a full analysis circle, complete with data pre-processing and a simple-to-use user interface.

1.2 Biological Question

Studying gene expression lies at the heart of modern biological sciences. Understanding it may lead to cures of deadly diseases, increased crop yield, or answering the fundamental questions of evolution of life. However, much is still undiscovered in the mechanisms behind this process. In this section we present the high-level overview of gene expression, relevant to this work.

Every cell in an organism has the same DNA. However, complex organisms are comprised of a variety of different cell types, like muscle cells, hair cells, etc. Part of the reason for this diversity lies in the way DNA is translated into functional proteins. The mechanism of this translation is described in central dogma on Figure 1.1.

Figure 1.1: Central Dogma of Molecular Biology: transfer of sequential information in biological systems.

In general case, shown with solid arrows on Figure 1.1, DNA is first transcribed to an RNA, which is then translated into a protein that carries function. In this work we focus on the transcription stage of this process.

A key step in transcription happens when specialized proteins, called transcription factors, bind to specific areas on the DNA sequence and initiate the copy of a section of DNA into a single-stranded RNA. In simpler prokaryotic organisms, one transcription factor is usually responsible for the transcription of one gene, however in eukaryotes transcription is more complicated [46]. The expression of genes in eukaryotes is achieved by relatively few transcription factors (about 10 times less than the number of expressed genes)[46]. This suggests that in eukaryotes regulatory proteins work in complexes, contributing to a combinatorial gene expression regulation. By combinatorial we mean that a number of transcription factors bind at specific areas on the DNA and interact with each other either directly or through other proteins that bind to them. These complexes could be spread out over large areas of the DNA, from thousands of nucleotides away from the promoter, to the intragenic regions. Figure 1.2 from [2] illustrates the complexity of a transcription complex of a typical eukaryotic gene.

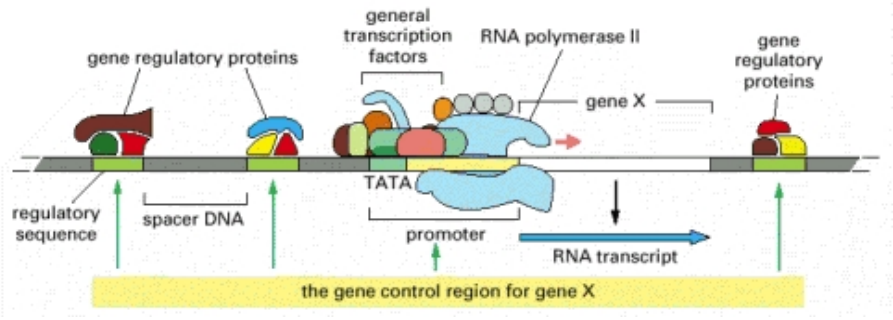

Figure 1.2: The gene control region of a typical eucaryotic gene. Figure from [2]. Regulatory sequences are binding sites on the DNA. Promoter - area containing binding sites of part of the transcription complex; is located upstream and near the gene. Gene control region spans all areas on the DNA containing binding sites for all transcription factors that affect gene transcription.

The sequence on the DNA to which the transcription factor binds is usually short, 5-20 nucleotides (nt) long and is called transcription factor binding site (TFBS). The order in which regulatory proteins bind, DNA methylation and flexibility, the location of the TFBSs, as well as other factors, such as presence of expressed transcription factors and whether they bind on a direct or reverse strand - all play a role in the kind of a transcription complex that forms and subsequently in the expression or repression of a

gene. For example, as described in [46], the complex that contains MADS TFBS and Ets binding site downstream on the reverse strand is important for cell proliferation and differentiation in humans. While the same MADS TFBS but with MATα2 binding site upstream on the reverse strand is part of the complex that regulates mating type determination in yeast. Understanding the principles of transcription factor assembly on specific DNA sites is important to understanding how eukaryotic genes are differentially expressed [46]. According to Lindlöf et al.[32], identifying motif combinations is especially important when studying complex processes such as cold acclimation of an organism.

1.2.1 *Cis*-Regulatory Modules

Regulatory elements of a DNA sequence are usually located in the vicinity of the coding sequence, either upstream (promoter region), downstream or in intragenic regions. In the case of many higher eukareotic organisms (e.g. humans), TFBS tend to occur in clusters in the upstream regions and are called *cis*-regulatory modules (CRM). Finding these CRMs experimentally is laborious and time consuming[17], therefore a lot of computational methods have been developed for this purpose. Our method can assist in solving the problem of finding and describing CRMs and it stands out from the rest of the methods by its ability to detect CRMs based on the over-represented relationships of TFBS within a cluster, while the rest of the methods mostly examine the over-representation of the TSBS or their clusters. For a discussion on the related methods see Section 2.1 in the next chapter.

1.2.2 ChIP-Sequencing

High-throughput experiments have been developed to detect protein binding sites on a DNA sequence, such as ChIP-Sequencing. ChIP-Sequencing (ChIP-Seq) is a current experimental method, used to study how proteins interact with DNA to regulate gene expression. It is the next-generation sequencing technology, capable of producing tens of millions of sequence reads during each run. The outcome of this experiment is a large set of relatively short sequences (about 300 nt), with the binding site of the transcription factor of interest located approximately in the middle of each peak. The peaks are determined by aligning ChIP-Seq sequences to the original genome and finding the areas where the binding sites of interest are over-represented. For instance, in Palii et al.[39], the authors studied the binding of TAL1 transcription factor in two cell lines. Analysis of this outcome is often tedious and time consuming. Due to the large amount of data,

human experts can not directly analyze the data. The automatic processing, on the other hand, either involves a testing of a prior hypothesis, set forth by the experts, or produces a classification, the results of which are not easily understood and therefore distrusted by biologists. Our method aims to address these issues and introduce automatic extraction of knowledge from ChIP-Seq data.

1.3 Inductive Logic Programming

Inductive Logic Programming (ILP) is a branch of machine learning, which follows a logical approach to learning. It was first introduced by Stephen Muggleton in 1991 [38]. ILP works by inducing a hypothesis based on a set of positive and negative examples and background knowledge. All data in ILP, including examples, background knowledge, as well as induced hypothesis are represented as logic programs. The short list of main logic programming concepts, used in this text, is presented below:

- Predicate is a boolean-valued function, which consists of a predicate symbol and a list of parameters. For example, $father(john, ann)$ - John is a father of Ann.
- Term is either a constant, variable (in this document, variables start with a capital letter, while constants start with a small letter), or a function.
- Clause is a finite disjunction of predicates (eg. $fruit(X), color(X, red), has\_seeds(X)$).
- Substitution is an assignment of terms to variables.
- θ-subsumption. A clause P_1 θ-subsumes clause P_2, if there exists a substitution θ, for which $P_1\theta \subseteq P_2$.

While logic programming is usually concerned with deductive inference (reasoning from general to specific), ILP works by reversing this process and using induction to derive general explanation from the specific instances. The input to the ILP system is background knowledge B, a set of positive examples E^+ and a set of negative examples E^-. The output is a theory T (also called a hypothesis), which describes all positive and none of the negative examples in terms of the background knowledge. The background knowledge and theory are described as a set of rules, which are a set of definite clauses of the form:

$$g \leftarrow b_1, b_2, ...b_n$$

where $g, b_1, b_2, ...b_n$ are predicates. For example, a definite clause describing a rule for a grandparent will be: $grandparent(X, Y) \leftarrow parent(X, A), parent(A, Y)$.

In general, when inducing a theory ILP orders clauses from the background knowledge by θ-subsumption and searches this lattice for a suitable theory by repeating the following steps:

1. Randomly select a positive example to be generalized.
2. Construct a bottom clause. Bottom clause is a most specific clause that entails the selected example. This is called a saturation step.
3. Construct a more general clause by searching a subset of literals in the bottom clause that have the best score. This is called a reduction or search step.
4. Add a clause with the best score to the current theory and remove the examples that cover it from future search. This is called a cover removal step.

The calculation of the score for the best clause depends on the ILP system. Compression is an example of a score function, which is widely used (e.g. default Aleph scoring function). It is defined as follows:

$$C = \alpha P - N - L$$

where P is the number of positive examples, covered by the clause; α is an inflation constant; N is the number of negative examples, covered by the clause; and L is the length of the clause.

The theory, induced by the ILP system, must meet the following conditions:

- Prior satisfiability: $B \wedge E^- \not\models \square$. This condition requires that none of the negative examples can be proved true from the background information alone, which provides us with certainty that the data available to solve a problem is consistent.
- Prior necessity: $B \not\models E^+$. This condition requires at least one positive example not to be entailed directly by the background knowledge presented, which provides us with certainty that the learning problem is not already solved by the background rules in B.

- Posterior satisfiability: $B \wedge T \wedge E^- \not\models \square$. This condition requires that none of the negative examples should be entailed from the theory.
- Posterior sufficiency: $B \wedge T \models E^+$. This condition requires all of the positive examples to be entailed from the theory.

In most systems it is possible to relax posterior sufficiency and posterior satisfiability to allow for a certain amount of noise.

ILP was selected as the core of our approach because of the following reasons:

- As discussed in Section 1.2, relationships between TFBS and other factors, such as DNA composition, play an important role in gene expression of eukaryotes. ILP allows for a relational representation of data, as well as induces relational theory describing it, which is highly desirable when dealing with real biological data.
- The result of ILP classification is an easy to understand theory that can be proved or disproved experimentally.
- New data can be added back to the background knowledge and thus improve the predictive power of ILP.
- ILP is able to discover a theory even when data is scarce or noisy.

1.4 Contribution

Our main contributions are listed below:

- We identify the limitations of existing CRM finding methods.
- We pose the problem of CRM discovery and sequence analysis based on motifs as a relational learning problem.
- We provide a novel application of ILP to find relationships between regulatory elements in a set of co-regulated DNA sequences. This application can assist in finding *cis*-regulatory modules and analyzing ChIP-Seq data.
- We analyze the impact of different kinds of synthetic data on positive-only learning and provide a new method of negative example generation, which, when used on

regulatory sequences data with ILP, provide for a more relational theory than widely used null model based on Markov chain of order 1.

- Recognizing the difficulty for a non-programmer to deal with logic programs, we provide an end-to-end analysis solution, with a Web interface and a data pre-processing module.

- We apply our method on two real world datasets, *C. elegans* [63] and human [39] and provide the analysis of the results.

A poster with preliminary results was presented at the 15th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2011), Mon-BUG Bioinformatics Symposium 2011, and the manuscript, based on this work, is being written.

Chapter 2

Literature Review

The advancements in the sequencing technologies and the analysis of the resulting sequences have revealed that the number of genes in a genome does not directly correspond to the complexity of the carrier organism. It has been suggested that the reasons for the complexity could instead lie in the more elaborate gene transcription mechanism, which, among other things, include *cis*-regulatory DNA sequences [31]. The method we have developed, ModuleInducer, aims at analyzing *cis*-regulatory regions of related genes and discovering relationships between biological markers, such as TFBS, that describe these *cis*-regulatory modules. However, our method is not limited to this application. Any type of DNA sequences with any type of biological markers (for instance ChIP-Seq experiment results) can be analyzed. Our main contributions are extracting a relational theory describing such biological data, as well as discovering *cis*-regulatory modules (CRMs) *in silico*.

In this chapter, we will review the current developments in the research to justify the selection of our problem as well as means of solving it.

2.1 CRM Finding Methods

Even though a human is morphologically and behaviorally much more complex then a worm, its genome contains only twice as many genes [13]. Currently there are several explanations as to how this complexity can stem from a relatively small amount of genes, among which are alternative splicing, DNA rearrangement and non-protein coding genes [35]. To add to this list is a theory that complexity correlates with the number of gene expression patterns, namely with the elaboration of the *cis*-regulatory DNA se-

quences [31]. Since the annotation of such sequences is lagging far behind the annotation of the genes that they regulate, a number of computational methods have been developed to fill this void.

At first, the methods focused primarily on discovering the recurring sequence patterns, like TFBS, in the *cis*-regulatory sequences. The methods range from deterministic algorithms to machine learning approaches. Early deterministic approaches would exhaustively enumerate all possible motifs (sequence pattern of nucleotides that is widespread and/or has biological significance) of a given length [56, 53, 41] and therefore are limited in the length of input sequences and motifs. The machine learning approaches [61, 10] allow for larger input but require a large set of training data to avoid over-fitting. However, when the TFBS found by these methods were verified *in vivo*, it became apparent that majority of them are non-functional, the observation that was termed the Futility Theorem [60]. The way to overcome the Futility Theorem is to use contextual information, like sequence conservation and clustering of TFBS within *cis*-regulatory modules.

The next generation of methods analyzed *cis*-regulatory sequences by looking for CRMs. As per recent classification of such methods by Van Loo et al. [58], these methods could be divided into 3 groups: CRM scanners, CRM builders and CRM genome screeners. CRM scanners require a prior knowledge of the possible CRM composition, which they use to scan the input sequence for possible CRM occurrences [15, 42]. CRM genome screeners look for overrepresented motif clusters without assumptions of which TFBS could constitute a CRM [20]. Our method falls into the CRM builder classification. CRM builders either find their own TFBS or use existing position specific scoring matrices (PSSMs) and look for combinations of TFBS that might constitute a module in the set of co-regulated or co-expressed sequences. Out of the variety of methods that also fall into this category, our method has the most similarities with ModuleSearcher [1], ModuleMiner [57], and CREME [48], since these methods also use existing PSSM to model TFBS in the input sequences.They first identify the location of PSSMs in the input sequences, which identify TFBS and then look for clusters of these TFBS that could be classified as CRMs. ModuleSearcher uses a maximum sum of scores for PSSM matches and a branch-and-bound algorithm to identify CRMs with the highest score. ModuleMiner is building upon ModuleSearcher by identifying the most discriminative CRMs. CREME selects only single overrepresented PSSMs matches and then calculates the statistical significance score for combinations of matches. All these 3 methods have

shown to work well on large datasets (human and mouse). However, they all work on genes of species for which the pre-computed alignments exist, making them restricted to only a few well-studied genomes (currently human, mouse and rat) [16]. These tools also make assumptions on where to search for CRMs (like 10000 nt upstream of coding sequence for [1] and [57]), which makes them inapplicable to analyze modern ChIP-Seq experiment data.

However, the largest shortcoming of most of the current CRM finding methods is that they are unable to detect CRMs that are based not on over representation of motifs or clusters of motifs, but on an association of motifs in the cluster. In [47] the authors call this situation a spatial motif combination. In other words, the first generation of CRM finding methods are able to find CRMs based only on single motif over-representation. The second generation can detect CRMs when single motifs are not over-represented, but their combination is. The third generation should be able to detect CRMs even when the combinations of motifs is not over-represented, but there is an underlying structure to the CRM. When Segal et al. suggested this classification, they also presented a method that falls in the last category. Their method identifies novel TFBS then looks for over-represented clusters of TFBS in small windows that are moved along the regulatory sequences. This allows the method to identify CRMs with motifs that are clustered close together at some location in regulatory sequence (like close to transcription start site), but are not over-represented in the whole sequence. However, this method is unable to find CRMs with underlying structure (e.g. when a motif follows another motif at some large distance). The method we are proposing is able to overcome this limitation. With the use of inductive logic programming, we are able to not only classify CRMs based on a spatial motif combinations, but also discover complex logical rules describing a CRM structure.

2.2 Relationship Finding Methods

Currently there are two main computational approaches to mining biological data that can handle relational information: inductive logic programming and propositionalization [29]. Propositionalization is a technique to transform relational (first order logic) representation to propositional (fixed sized feature vectors). However, the transformation often results in the loss of information [36], since the language of propositions is less expressive than that of first order logic. In addition, some relational rules can not be

propositionalized at all, which is the case with recursive rule definitions [36]. After application of propositionalization to a relational data, any kind of propositional classifiers, like support vector machines, neural networks, etc. can be used. These propositional learners have already earned their respect in the field of machine learning and are being successfully applied in the field of bioinformatics [5, 37]. However, even though propositionalization takes care of the relational context in the input data, application of any propositional classifier afterwards will not yield the discovery of the relational context in the data. In addition, the rules underlying the resulting classification are usually not reported or very hard to understand, especially in the case of highly successful methods, like SVM. This poses a problem for the application of these methods in bioinformatics, since usually the researcher would like to understand the hypothesis behind the classification and confirm it by testing it *in vivo*.

For these reasons, ILP is a better choice for the discovery of CRM. By operating directly on relations, it is able to not only handle relational input data, but produce a classification described in relational form. ILP provides an additional exploratory value to the research for the two following reasons. Firstly, it is easier to visualize the input data, compared to the array of values of propositional learners. Secondly, introducing a new analysis concept to the system is usually quick and straightforward and does not require a change to the input data (i.e. a matter of adding an extra rule to the background knowledge definition). ILP is also, to the best of our knowledge, the only method that is able to discover new rules about the classification data, by combining simpler background rules into a disjunction of rules. Because of these advantages, ILP has already been successfully applied to a number of bioinformatics problems, like studying tumor evolution in breast cancer [7]; automated protein fold signature discovery [55, 54]; analyzing structure activity relationships in chemical compounds for drug discovery [27, 28, 51] and many others. ILP is also widely used in combination with other computational methods. For instance, in combination with SVM to classify toxic compounds [37, 33] or using ontologies to mine complex biological relationships from existing data [34, 40]. In the last case, the authors combined the use of object oriented features of Java programming language to outsource computationally intensive functionalities and therefore speed up the execution of ILP. The expressive power of ILP has also been used in [6] for the task of finding not highly conserved motifs in protein sequences and combining this information with decision trees and SVM in the task of finding protein homology.

In the view of this recent research, we feel the proposed combination of Java preprocessing and ILP rules engine implemented in ModuleInducer is a best and not previously explored choice for the problem of discovering complex relational CRMs.

Chapter 3

Method Description

3.1 Problem Description

From a data mining perspective, we are faced with a problem of classifying a set of sequences over alphabet A, C, G, T. These sequences could be either upstream regions of the co-regulated genes, ChIP-Seq peak sequences, or any DNA fragments of interest, and they contain matches to some motifs of interest (regulatory elements or TFBSs). Our task is to discover the complex relationships between these motifs, which are over-represented in the positive sequence set, as presented on Figure 3.1.

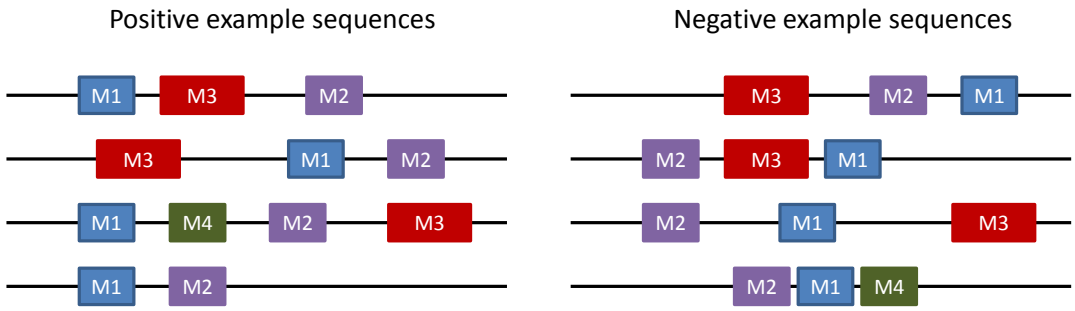

Figure 3.1: Example of the classification problem. M1, ... M4 are names of TFBS, matched to the sequence.

Figure 3.1 illustrates the example dataset, from which the traditional propositional learners will not be able to extract classification. Each motif is present exactly the same number of times in the positive and negative sequence set and the approximate location

of motifs (beginning, middle, end of the sequence) also could not be distinguished for positive and negative examples. The defining difference is that in positive sequences motif M1 is always before motif M2. Our method should be able to discover a rule like this.

The input to our system is a set of positive and negative sequences (negative sequences are optional) and a set of motifs, represented as PSSMs. The system should be able to match each motif to each sequence and discover a theory, describing the data in terms of relationships between the matched motifs.

3.2 Implementation

Module Inducer is a combination of several different technologies and computer science paradigms brought together using Java. Java was selected for several reasons. First, its object oriented nature makes it intuitive to build complex applications by separating different functionalities into blocks and then combining them. Second, since Java's platform, Java virtual machine, runs on most operating systems and hardware types, the code is easily portable. Also, Java native interface allows for an easy integration with other technologies, which can be beneficial for future integration, since a lot of bioinformatics software is written in different scripting languages. Last, it is a preferred language used in web technologies, making it easy to create a web interface for the finished tool.

The application can be divided into 3 main modules: user interface, data management and ILP engine. The high level diagram of the module interaction and data flow through the system is presented in Figure 3.2

Details of implementation of each module, shown in Figure 3.2 are described in the following sections.

3.2.1 User Interface

User Interface (UI) is a Web Interface, written using Java Server Facets (JSF) 2.0 technology[8]. JSF is a standard framework for building web interfaces for Java applications, which is based on a model-view-controller (MVC) software architecture. This makes for an easy separation of functional logic (model) and interface (view) with its rendering logic (controller). In the case of Module Inducer, model is represented by managed beans, which are linked to Data Management module. View is represented by xhtml

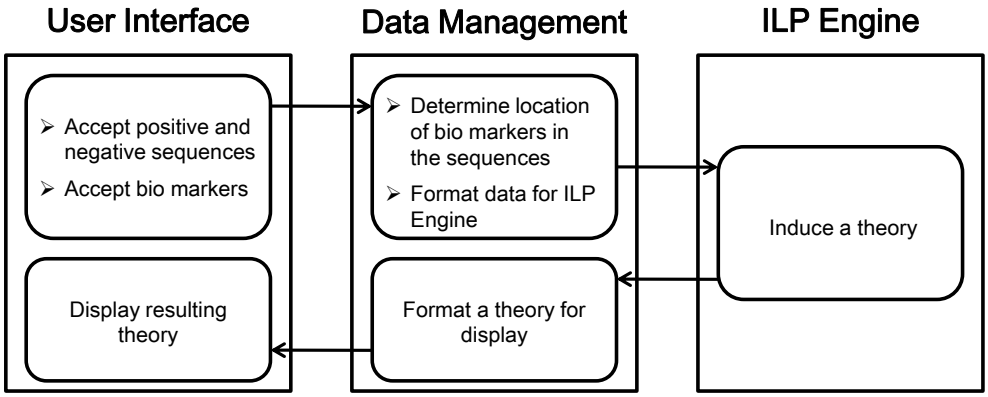

Figure 3.2: High Level Project Diagram. The data flow is as follows. Positive and negative sequences and possible biomarkers (i.e. PSSMs for TFBSs) are entered through User Interface. For simplification purposes, other system parameters are omitted in this diagram. The data are sent to Data Management, where position of each biomarker in each sequence is determined and resulting data are formatted for ILP. ILP Engine is then invoked, which induces a theory based on the positive and negative examples and system defined background knowledge. The resulting theory is then formatted in the Data Management and sent to User Interface for display.

documents with JSF enabled Facelets tag libraries. Controller is represented by Faces servlet, which is part of the JSF framework and will not be discussed here. Diagram of the user interface module is presented on Figure 3.3.

From the web interface, the user can work with two types of data: example *C.elegans* data [63] and custom data. *C.elegans* data are pre-filled from the article[63], however the theory is induced every time and not saved from the previous run. Custom data allow for the user to input his/her own data for analysis. Due to the current hosting limitations, we limit the number and length of the input sequences to 150 and 3000 nt respectively and the number of regulatory elements to 15. This is only a UI limitation, core Module Inducer is able to handle much larger data size.

Screen captures of the user interface are available in the Additional Information section B.5. The application is currently running at `http://induce.eecs.uottawa.ca/`.

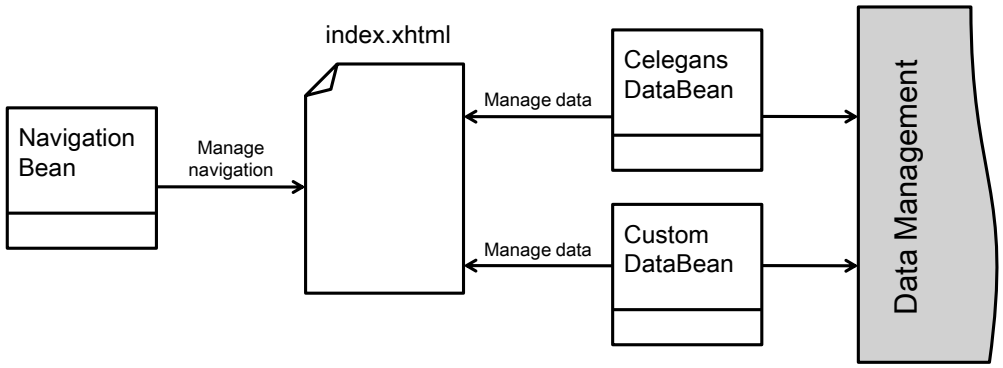

Figure 3.3: User Interface Diagram. User interacts with index.xhtml page, which has 3 managed beans associated with it. NavigationBean keeps track of what kind of data user works with (i.e. *C.elegans* or custom), while CelegansDataBean and CustomDataBean manage the data that user puts in and interact with Data Management module to produce an output theory.

3.2.2 Data Management

The Data Management module is in essence the core of Module Inducer tool. It is a Java program that accepts the user input, pre-processes and formats it for the ILP Engine and invokes the ILP Engine. The class diagram of Data Management module is shown in Figure 3.4.

The design of the Data Management module was inspired by Weka Data Mining Software [19]. Namely, the idea of having everything related to one experiment encapsulated in an Explorer class and then having an Experimenter class, which can perform various machine learning tests (such as leave-one-out, 5-by-2, etc.) to benchmark and compare the performance of different experiments.

Explorer

Explorer is a container for one experiment. The *first step* to create an experiment would be to provide positive and negative sequences. These are encapsulated, with all the relevant logic, in RegulatoryRegionService interface. The interface is implemented by four classes:

1. CelegansRegRegionService. The data in this class are hardcoded from the Zhao et al. [63]

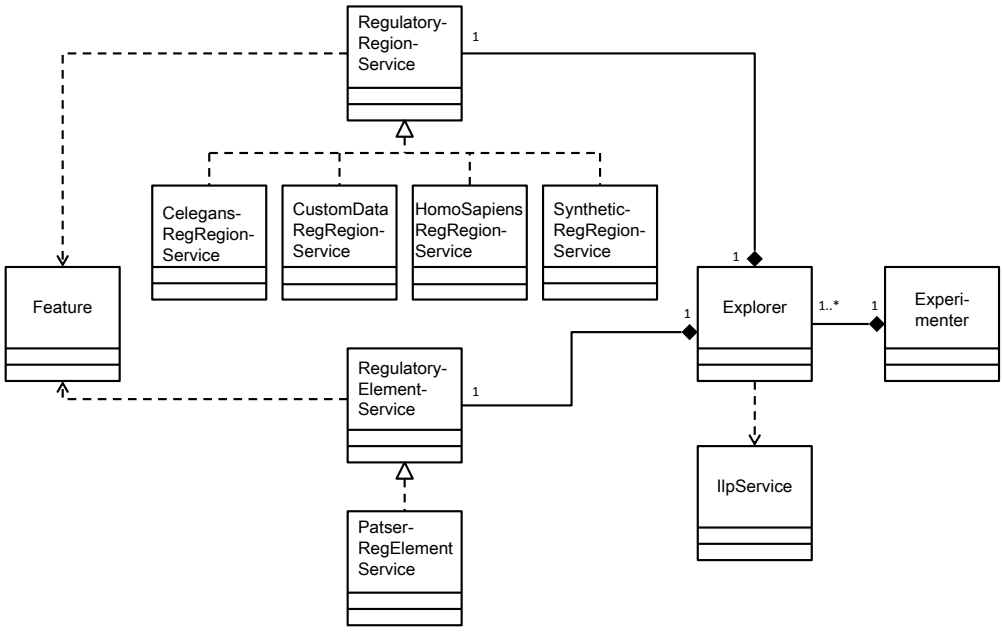

Figure 3.4: Class Diagram of Data Management Module. RegulatoryRegionService and RegulatoryElementService are interfaces that are responsible for handling sequences and biological markers respectively. Each of these 2 services produce a list of Feature data structures, but in case of RegulatoryRegionService its sequences, and in case of RegulatoryElementService its TFBS. The two services are necessary to create an Explorer object, which instantiates IlpService, which in turn runs Aleph engine and induces the theory based on the provided data. Experimenter is an automated test class, which operates with Explorer objects.

2. CustomDataRegRegionService. This class accepts user input either as a file or as a string.

3. HomoSapiensRegRegionService. Data from Palii et al. [39]. This class reads ChIP-Seq data from the provided .map files and extracts the sequences from *Homo Sapiens* hg18 genome.

4. SyntheticRegRegionService. Created for the validation of the approach, this class generates synthetic sequences either based on the example sequences (with the same composition and biomarker frequency) or with the specified sequence composition and type of synthetic sequence generation. The details on the different kind of synthetic data generation methods are presented later, in Section 4.1.2.

Second step in creating an experiment, is to supply biological markers, or TFBS. These are encapsulated, with relevant logic, in RegulatoryElementService interface. Currently there is one class that implements this interface - PatserRegElementService. This class wraps Patser[21] tool to scan and locate PSSMs in a DNA sequence. It allows PSSM data to be submitted in several formats, such as as:

- fasta-like file, with the name of PSSM followed by the matrix;
- sequence in the same format, as a fasta-like file;
- directory with individual PSSM files.

Both RegulatoryRegionService and RegulatoryElementService encapsulate their data as a list of Feature data structures. The Feature data structure was inspired by Generic Feature Format (GFF) version 3, developed by the Sequence Ontology Project [44]. We have not followed the GFF format in entirety in the interest of efficiency, since most of the attributes would not be used in our system. However, those attributes that were used in the system follow the GFF format and the feature type values were taken from the GFF controlled vocabulary. This gave us the flexibility of re-using the same data structure for both sequence and TFBS data, as well as gave us an added benefit of hierarchically grouping all the data in the system.

Once both experimental steps are completed, Experimenter object can be created. It in turn invokes IlpService, which is responsible for:

- creating background knowledge and positive and negative examples files;
- invoking ILP Engine.

The theory, produced by ILP Engine is formatted and retained in the Explorer.

Experimenter

Experimenter is an automated test class for experiments. It requires an Explorer object (i.e. experiment) to be initialized and offers a set of tests to validate the experiment. Currently available tests are:

1. Five-by-two test with no stratification. Randomly divide the dataset (positive and negative examples together) in half and repeat this 5 times.

2. Ten fold stratified cross-validation test. Divide positive and negative examples into 10 equal parts. Run the experiment on 9 out of 10 parts. Repeat 10 times, until all parts were excluded from the run.

3. Leave-one-out test. For each run, leave one positive example out and compare the results between all runs.

4. Imbalance test. Run experiment with many more negative, then positive examples. Gradually decrease the number of negative examples and compare the theories.

5. Set of tests to exercise separate background rules.

More detailed description of the tests, as well as the test setup and measures of performance are presented in the Chapter 4.

3.2.3 ILP Engine

ILP engine is the "thinking" part of our system. It consists of background knowledge and positive and negative examples, which are executed on Aleph[49] system to produce a theory, describing the data in terms of the background knowledge rules. The background knowledge, examples and the resulting theory are all represented as logic programs. The first two are generated, together with the script to execute them, by Data Management module. The theory is captured from the Aleph output. We also provide Aleph search restrictions, which fine-tune the learning to the level of noise, accuracy and speed of execution, which we find acceptable. The Aleph engine with background knowledge and search restrictions can be thought of as a trained expert with knowledge about the motif interactions in the DNA sequence and the ability to apply this knowledge to make logical conclusions with reasonable accuracy at reasonable time, when presented with experiment data. The experiment data is represented, in this case, by positive and negative examples. This section presents the details of each part of the learning module.

Background Knowledge

The background knowledge is a set of prior assumptions about the world, which are used in learning. In our case, the "world" is a set of interactions that are possible between motifs in a DNA sequence. If we think of the DNA sequence as time and motifs as temporal intervals, then we can represent these interactions as relationships between temporal intervals, described in [3]. Since the search space of the ILP algorithm

grows exponentially with the addition of every background knowledge rule, specifying all possible interactions would not be feasible. Therefore we have selected a subset of interactions, based on our knowledge and discussions with experts. The rules, used in induction, together with their descriptions are presented below:

- `has_feature(Seq, Motif)` - regulatory sequence `Seq` contains a match to a regulatory element `Motif` (represented by a PSSM). This rule is reported in the resulting theory if the significant over-representation of a regulatory element is found in sequences.

- `has_feature(Seq, Motif, Strand)`. Since a match to a regulatory element can be found on a direct strand ($5'$ to $3'$ direction) as well as reverse ($3'$ to $5'$ direction), this rule is a more restricted version of the previous, where over-representation is reported only if the direction of the motif match is the same.

- `before(Motif1, Motif2, Seq)` - regulatory element `Motif1` is located before `Motif2` in sequence `Seq`. This rule is reported in the resulting theory if a particular motif is found before another in significant number of sequences. Note that over-representation of any of the two motifs is not necessary.

- `chromosome(Seq, Chr)` - sequence is found on a chromosome `Chr`. When experiment sequences are located in different chromosomes, this rule can be useful, especially as one of the conjuncted terms in a rule (i.e. `has_feature(Seq, m1), chromosome(Seq, II)` - sequences on the chromosome II always have a motif `m1`)

- `distance_interval(Motif1, Motif2, Seq, Dist, Offset)`. This predicate represents a relationship between two motifs using a distance and an offset. During the saturation step, the identity of the two motifs and their distance are determined from the currently selected example. The value of the offset is selected from a fixed list of constants, namely 0, 5, 10, 15, 30, 100. Literals are added to the bottom clause for all the identified combinations of motifs, distances and offsets. During the reduction step, the boolean value of the predicate is true if and only if there are occurrences of `Motif1` and `Motif2` in `Seq` at distance `Dist` $+/-$ `Offset`. Offset is important because biological data are often noisy, but also because the underlying processes that we are modelling might be plastic, i.e. tolerate distance variations.

- `pos_lteq(Seq, Motif, Position)`. We number the positions inside the input sequences in a symmetric way, with 0 in the middle, increasing indexes to the right

and decreasing to the left (i.e. [-3, -2, -1, 0, 1, 2, 3]). This numbering inherently provides us with extra information on the relative location of the motif inside the sequence. The rule `pos_lteq` means that a motif `Motif` is located at or less than the index `Position` inside a sequence. This information is especially useful when analyzing ChIP-Seq data, because it shows how close the motif is located from the peak.

- `pos_gteq(Seq, Motif, Position)` - similar to the rule above, only the location of the motif is greater than or equal to the index `Position`.

Positive and Negative Examples

Positive and negative examples represent the experiment data that need be analyzed. Both are described as logic programs and the predicate that describes them is presented below.

```
tfbs(<regEl name>, <regSeq name>, <start position>, <end position>,
     <strand>, <score>)
```

where:

- `<regEl name>` - name of the regulatory element (usually, the name of the PSSM that represents it);
- `<regSeq name>` - name of the regulatory sequence this regulatory element belongs to;
- `<start position>` - position of the first nucleotide (inclusive) of the regulatory element in regulatory sequence;
- `<end position>` - position of the last nucleotide (exclusive) of the regulatory element in regulatory sequence;
- `<strand>` - a flag that can take two values: `"D"`, if the match of the regulatory element was found on a direct strand ($5'$ to $3'$), or `"R"` if the match was found on a reverse strand ($3'$ to $5'$);
- `<score>` - score of the match, as reported by PATSER.

This predicate is automatically generated by the Data Management module and all the required information is extracted from the input sequences and provided PSSMs. The Data Management module also generates two lists that indicate which of the regulatory sequences are to be considered as positive or negative examples.

Search Restrictions

The success and the speed of learning with the ILP depends on the restrictions set on the Aleph engine. The restriction we have implemented is as follows:

- Maximum number of nodes to be explored when searching for an acceptable clause. We have increased it to 100000 from the default 5000. This increases the learning time, however also increases the accuracy of the accepted clauses.
- Noise. In Aleph the noise is represented by the maximum number of negative examples that can be covered by a rule. We set it to 0.5% of the number of negative examples.
- Minimum accuracy of the accepted clause. In Aleph, the accuracy of the clause is the same as precision (see Table **??** for definition). We set it to 90%.
- Minimum number of positive examples to be covered by the rule. This parameter is set to either 3 or 1% of the number of positive examples,whatever is larger. Setting it to a higher number, then the default 1, reduces the search space and provides the rules with higher coverage in the resulting theory.

The above parameters were determined from observing the results of multiple runs of our method on a number of different datasets, which will be presented in Chapter 5. Current user interface does not allow to change these parameters, however the back-end allows to vary any of them for each individual run.

Chapter 4

Experimental Setup

The success of the evaluation of any knowledge discovery algorithm depends on three factors. First is the selection of the test data, which exercises all functionalities of the method. Second is the selection of the evaluation metric, which captures the wide array of performance measures. And third is the selection of the test method. Second and third factors go hand-in-hand, since the selection of the performance metric depends on the kind of tests that can be performed on the system. In this chapter we first describe different types of data that were available for our testing. Then we describe the types of tests, selected for performance evaluation, together with their respective evaluation metric.

4.1 Experiment Data

The success of any type of learning, whether it is a human or machine learning, depends on the quality of positive and negative examples, supplied to the system. ModuleInducer, being a machine learning method for extracting a relational theory from biological data, is no exception. To verify and test our method we have used synthetic data as well as two types of real data from biological experiments: *C.elegans* muscle specific data from Zhao et al.[63] and human haematopoietic lineages from Palii et al.[39]. The real data was contrasted with automatically generated synthetic data for positive-only learning and for testing the robustness of the system. The details of each type of data used in our experiments are presented below.

4.1.1 Finding Regulatory Elements

The "thinking" part of our method operates with biological markers, found inside the analyzed sequences. Therefore locating the markers is an important step on the path to learning an interesting theory. If the method that locates markers is too stringent and finds few of them, then the learning algorithm will not have enough information for the theory extraction. On the other hand, if the method produces too many markers with low match score, this introduces noise to the learning system, which has a detrimental effect on the quality of the resulting theory as well as the running time.

Locating biological markers is an interesting and complex topic in its own right. Since it was not the primary topic of our research, we have integrated the existing and well-established Patser algorithm from Stormo et al.[21] to locate regulatory elements, represented as PSSMs. Patser converts PSSMs to Position Score Matrix using the sequence background composition. We calculate this composition from all the input sequences combined and supply it to Patser as background composition. This provides for a better discrimination between background noise and a true PSSM match. The matches, found by Patser, are provided with a score. The score is calculated as a P-value of the information content of an aligned PSSM (more details in [21]). Since the number of accepted matches is an important factor that will influence resulting ILP theory, we determined Patser cut-off score from the real data. We examined human jurkat and erythroid sequences from Palii et al.[39]. The authors reported the regulatory elements discovered in the experiment in the form of IUPAC motifs (see Glossary A for details). We have searched for these motifs as regular expressions in the reported regulatory sequences and noted the number of matches per motif. Then, using TOMTOM algorithm [18], we have located PSSMs, corresponding to each motif, and used the Patser algorithm with different cut-off scores to locate these PSSMs. We found that a cut-off score of 5.0 produced the closest number of matches per motif to that of regular expression matching.

4.1.2 Synthetic Data

The purpose of synthetic data in our experiments is two-fold. It is used as negative example sequences in positive-only learning. It is also used as both positive and negative example sequences to design specific experiments for the purpose of verification of our method and assessment of its performance in various controlled environments. When used in positive-only learning, the synthetic sequences have to be generated as similar to

the positive sequences as possible, while avoiding over-fitting the data, which will hinder the discovery of rules, describing it. We have explored this variance-bias trade-off, as it is known in machine learning field, to determine the most suitable way to generate synthetic sequences. Four different types of sequence generation methods were developed and are described below.

Synthetic sequences based on Markov chain of order 0

With Markov chain model, we consider a DNA sequence as a sequence of random letters over the alphabet $\{A, C, G, T\}$, where each letter depends on the m letters before it. In the case of Markov chain order 0, $m = 0$, therefore the letters in the sequence are independent of each other. We generate this synthetic sequence by using the same nucleotide composition as the model sequence. Typically, model sequences are positive example sequences.

Synthetic sequences based on Markov chain of order 1

With Markov chain order 1, $m = 1$ (as described above), therefore each letter in the sequence depends on one letter before it. Here we build a transition matrix, using nucleotide composition of the model sequence, to determine the probability of occurrence of a letter, given a previous letter. The synthetic sequence is generated using the probabilities from the transition matrix. This model is most widely used in the statistical analysis of biological sequences.

Synthetic sequences with randomly planted PSSMs

Generating synthetic sequences with planted PSSMs is a two step process. First, we generate background sequences using one of the 3 different methods:

1. Same composition as model sequence. This is identical to above simple synthetic sequences.

2. Equal composition. We use A : C : G : T ratios of 0.25 : 0.25 : 0.25 : 0.25.

3. Reverse composition. We switch frequencies of A with C and G with T of the model sequences to reverse A:T and C:G compositions.

Second, the background sequences are planted with PSSMs, found in the model sequences, in the following way. We calculate the number of matches for each PSSM in all

the model sequences. Then we plant each PSSM the same number of times at random locations in random synthetic sequences. The hypothesis behind creating this model is that by tailoring the synthetic data to produce the same frequency of PSSM matches as the model data will force the ILP engine to discover more relational rules about the data. Thus, we increase the system variance to hinder the discovery of the simpler rules based only on belonging to a set of sequences - the task that attribute-based learning methods, such as SVM, have already shown to perform successfully.

Synthetic sequences with positionally planted PSSMs

We developed this method to fit the synthetic data even closer to the real one, then with randomly planted PSSMs. With this method, we plant PSSMs with the probability of their occurrence at a specific section of the sequence. The background sequences are generated in the following way. All model sequences are aligned at the start. A window of certain size (default is 50) is moved without overlap along the alignment. For each PSSM, we estimate the probability of its occurrence in each window. We then construct a synthetic sequence, window by window, by randomly selecting a PSSM, then generating a random event with this PSSM probability and then either placing this PSSM at a current sequence position, or filling it with a sequence of equal composition.

4.1.3 *C.elegans* Data

C.elegans data was extracted from the study by Zhao et al.[63]. As positive sequences we have selected the data that the original study used for the evaluation of the accuracy. The data consisted of 16 upstream regions of length 2000 nt of *C.elegans* genes, which were experimentally shown to contain muscle-regulating CRMs. The authors have identified 18 muscle-regulating motifs and ranked them according to an Over Representation Index (ORI). We have used the PSSMs of these motifs, provided in the supplemental information, as the input to our system.

To select the best fitting synthetic sequences as the negative examples, we designed a series of experiments looking at the number of PSSM matches in real vs. different types of synthetic sequences. Our goal was to find a set of synthetic sequences that produce on average the same number of matches for each PSSM, as the original sequences. This way we will fit the negative example data closely to the positive, which will ensure that our ILP engine will find more relational rules (like `before()`) and fewer rules that

are based on the belonging to a set (like `has_feature()`).

Our first experiment was to plot 4 types of sequence generation methods against real data. The results can be seen on Figure 4.1. Visual inspection of the results shows that only Markov chain order 1 synthetic data fits closer to our expectations. It produces approximately the same number of matches, as model data, while reflecting most of the match frequencies. Both PSSM planting methods reflect the frequency of each PSSM match well, but on average produce twice more matches. Markov chain order 0 synthetic sequences produce a similar number of hits overall, but do not reflect the frequency of the matches. Therefore we exclude this method from further analysis.

Our next step was to tune the synthetic data with planted PSSMs to produce less matches. We have run an experiment, planting the PSSMs in different types of background data (same, equal, and reverse compositions, as described above). The resulting matches, averaged over 50 runs, are presented on Figure 4.2. Based on the variance of the matches, the results for the 3 different background composition did not statistically differ from each other and did not bring the number of matches down. Therefore we have selected a simpler equal composition as a background to continue with our tuning.

From the experimental results we have noticed that synthetic sequences with planted PSSMs consistently produce approximately twice the matches expected. This is probably due to the fact that these PSSMs were found in background sequences in approximately same numbers as in *C.elegans* sequences. Guided by this observation, we have reduced the probability of planting each PSSM in half, and the resulting number of matches was found to be close to *C.elegans*, while still reflecting the frequency of the matches. See the results on Figure 4.3

The result of this experiment leaves us with three types of synthetic sequences that match original PSSM match frequency: sequences generated by MC1 model, sequences with planted PSSMs, and sequences with positionally planted PSSMs. Since based on the frequency of the PSSM matches alone, it is impossible to say which dataset will perform better with our method, all three types were analyzed based on the hypothesis they produced and the results are presented later in the chapter.

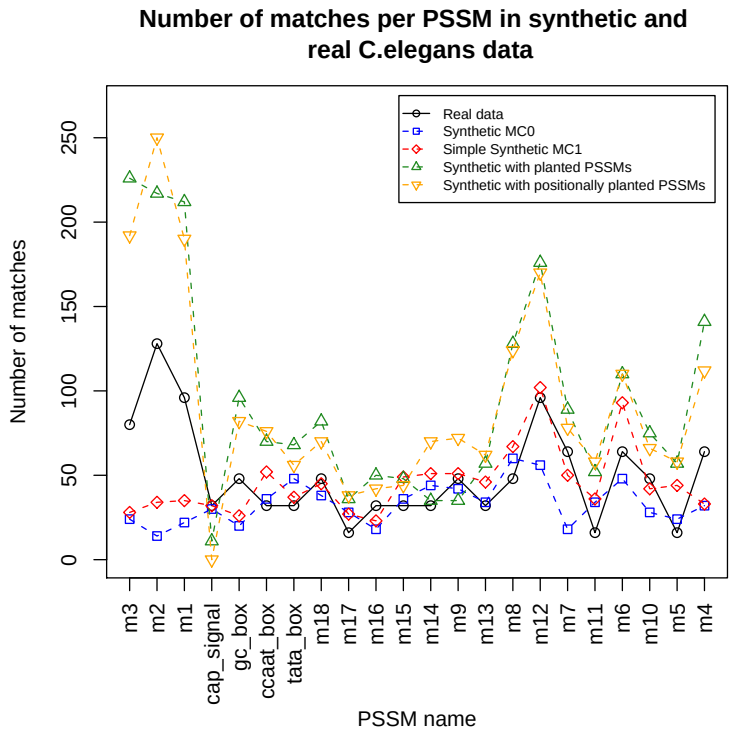

Figure 4.1: Comparison of the number of PSSM matches in *C.elegans* and three types of synthetic data. Simple synthetic sequences were generated using the nucleotide composition of the *C.elegans* sequences, while two kinds of synthetic data with planted PSSMs used the frequency of PSSM occurrence in *C.elegans* data to plant PSSMs in background sequences. The number of matches for synthetic sequences was averaged over 50 runs.

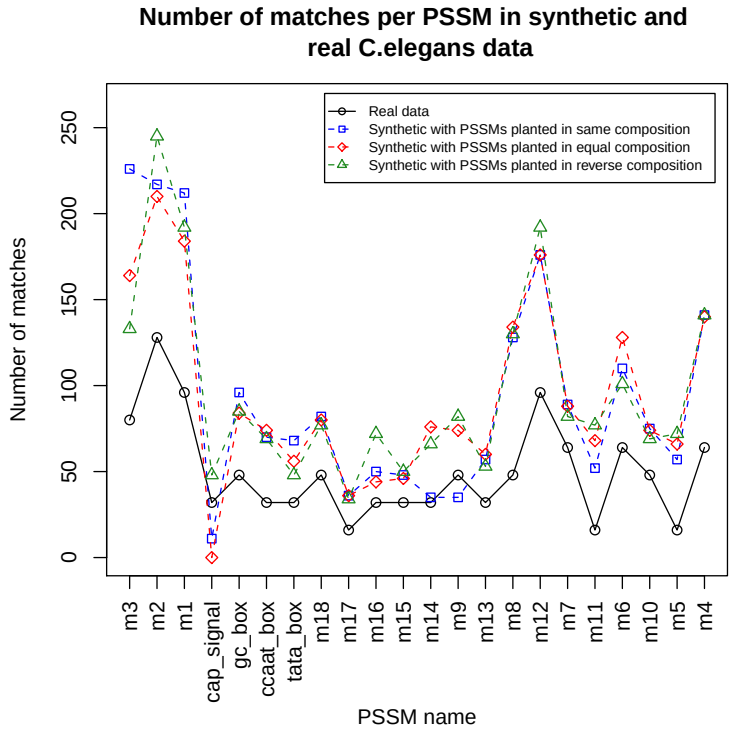

Figure 4.2: Comparison of the number of PSSM matches in *C.elegans* and synthetic data with PSSMs planted in three different background sequences. Same composition indicated the same nucleotide frequency as in *C.elegans* sequences. Equal composition is 0.25 : 0.25 : 0.25 : 0.25 ration of A : C : G : T. Reverse composition swaps AT and CG content of the *C.elegans* sequences. The number of matches for synthetic sequences was averaged over 50 runs.

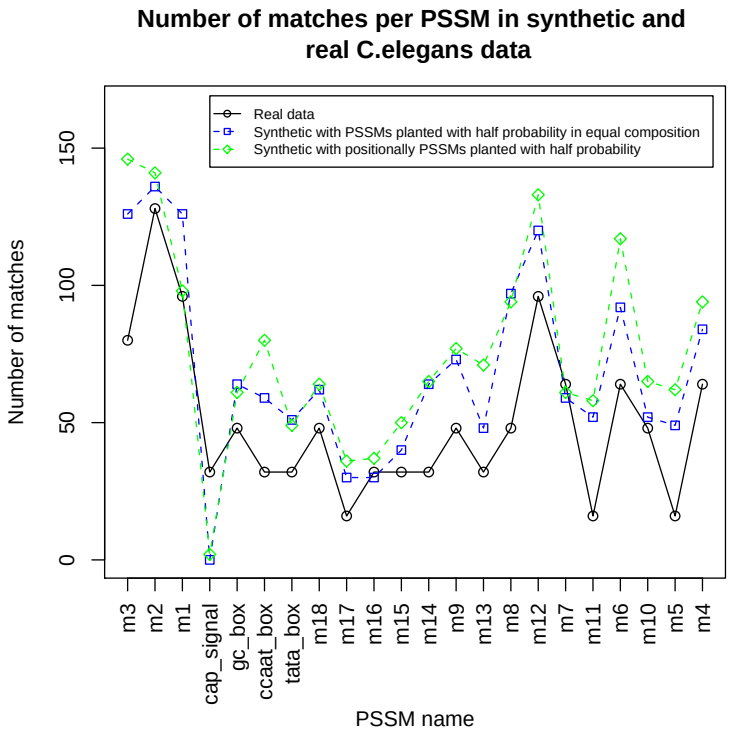

Figure 4.3: Comparison of the number of PSSM matches in *C.elegans* and synthetic data with PSSMs planted in equal distribution background sequences and half the probability. The number of matches for synthetic sequences was averaged over 50 runs.

4.1.4 Human Data

The human data that we have extracted from the work of Palii et al. contained two types of sequences: cancerous jurkat and normal erythroid sequences. The authors have identified a number of motifs in both sequence types, however they report only one significant relational rule: the distance between E-box and Gata sites is 8-10 nt in 6% of jurkat and 10% of erythroid sequences. The difficulty of finding any rule, describing the data, is that the scientist has to know what to look for, setup the experiment and run it. With the amount of data analyzed in Palii et al., testing one hypothesis is time consuming, which constricts the scientist to pick only a few of the most likely hypotheses. With ILP being able to discover new, unexpected relationships, we feel that this dataset is a good match for our method.

To further analyze this data, we first matched the discovered motifs (presented in IUPAC strings) to known PSSMs from Jaspar and Transfac databases. The matching was performed by TOMTOM algorithm [18]. Discovered PSSMs were used as input for the Patser algorithm. The list of the PSSMs, used in the experiment, is presented in Additional Information B.1.

The study already presents us with positive (jurkat) and negative (erythroid) examples. However, if like in the original study, our approach will not be able to find any significant rules distinguishing these two sets, we would like to examine each set with positive-only learning. To do this, we would need to contrast each set with synthetic data.

To determine the best synthetic data for both jurkat and erythroid sequences, we have repeated the same experiment as performed for *C.elegans* data above. The results were found to match that of *C.elegans*, therefore we present only the final selection graphs for jurkat and erythroid on figures 4.4 and 4.5 respectively.

4.2 Evaluation Methods

The advantage of ILP over other machine learning methods is that it is not only able to produce a classification, but can also induce a theory describing the classified data. The theory could be very beneficial to the researcher seeking to analyze the data, even if the classification accuracy is not high, since it can facilitate further research and/or revision

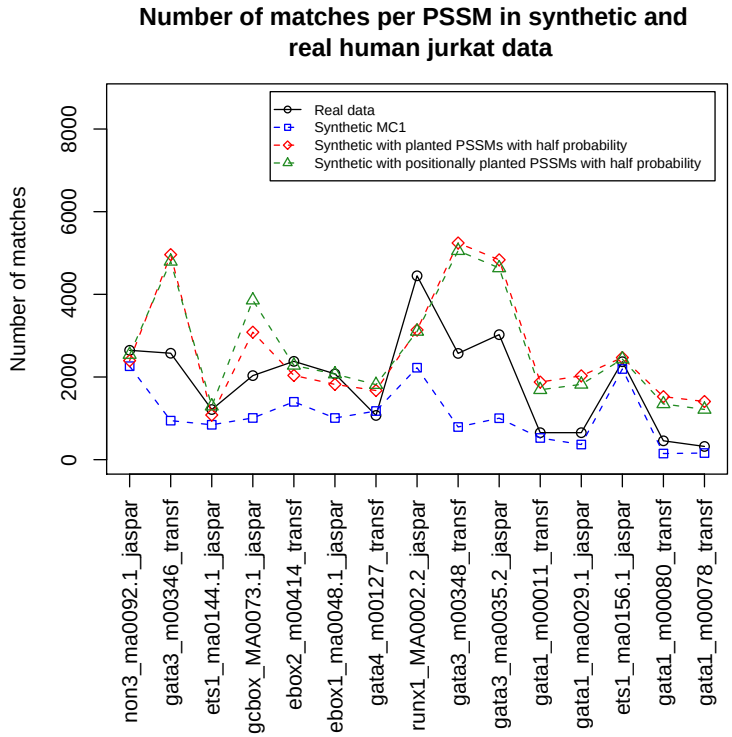

Figure 4.4: Comparison of the number of PSSM matches in human jurkat and 3 types of synthetic data: generated with Markov chain order 1, planted PSSMs and positionally planted PSSMs. Planting was done with half the probability. The number of matches for synthetic sequences was averaged over 50 runs.

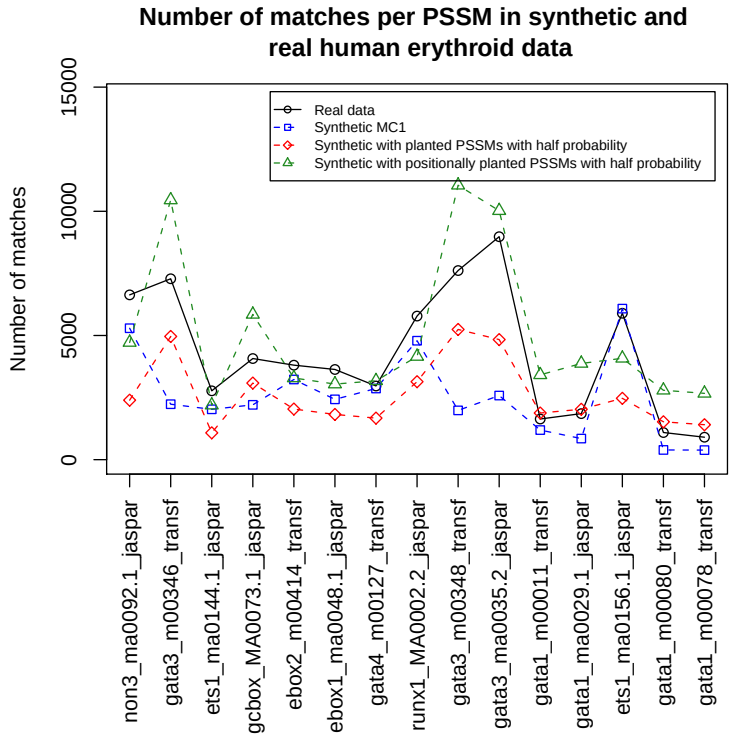

Figure 4.5: Comparison of the number of PSSM matches in human erythroid and 3 types of synthetic data: generated with Markov chain order 1, planted PSSMs and positionally planted PSSMs. Planting was done with half the probability. The number of matches for synthetic sequences was averaged over 50 runs.

of the quality of the data. Therefore to measure the performance of our method we used a hierarchy of different indicators, which collectively capture both the ability of our method to classify data, as well as produce a useful data description. These indicators could be divided into 2 classes: measures of the results of classification and measures of the quality of the theory. In this section we justify the selection of our evaluation metric as well as describe the test methods (cross-validation techniques) used to capture the performance of our method.

4.2.1 Measures of the Results of Classification

A wealth of techniques for sampling the data and measuring the resulting performance is available from the field of machine learning. When we talk about measuring the performance of the result of classification, we mean to apply some of these methods to measure how well our method performs at classifying new data based on the rules learned on the training data. Therefore we treat our ILP engine as any other machine learning classifier, where we divide the available data into training and test sets, produce a classifier as a set of rules, induced by ILP and then test this classifier on the test data.

Evaluation metric

Selecting an evaluation metric is an important issue, since it will not only measure the strength, but highlight the deficiencies of our system and will make comparison to other methods easier. The basis of all evaluation metric in machine learning lies in a contingency table (alternatively called confusion matrix). A description of this matrix is presented in Table 4.1. The most widely used evaluation metric in machine learning is accuracy. However it has been argued that it does not present a descriptive value of the classification, since describing a classifier in one number does not provide full information on its performance and it is independent of the cost of misclassification.

Instead, the graphical measure called ROC (Receiver Operating Characteristic) curve[45] with its accompanying AUC (area under curve) measure has been commonly used. In this method, the classifier performance is plotted on False positive rate (FPR) to true positive rate (TPR) 4.1, taking the cost of positive and negative classification into consideration. The shortcoming of presenting the classification results as ROC curves is that when comparing two distinct classifiers it is not immediately obvious which classifier is better for specific class distribution and cost. Since data from biological experiments

often produces much less positive examples then negative, it would be beneficial to use a metric that will enable us to easily compare the result of our method to other machine learning methods on such dataset. Davis and Goadrich [11] have shown that Precision-Recall curves are a better measure of classifiers for imbalanced data, while Cost Curves, introduced by Drummond et al.[12] have been shown to be more visually descriptive than ROC curves, as well as offer a better way of comparing classifiers and selecting specific classifiers based on class distribution and cost[11].

Ideally, we would have used Cost Curves as evaluation metric for our tests. However, using cost curves (or precision-recall or ROC curves) requires a score for misclassification, which our non-probabilistic ILP implementation does not provide. There are, of course, ways of turning discrete classifiers, like decision trees or in this case, our ILP implementation, into scoring ones[14]. One of such ways is to generate a score for each rule of the resulting theory, based on the amount of positive examples, covered by that rule. Aleph does provide this information for the training instances, however it does not provide it for test instances, making it impossible to extract the score without modifying Aleph implementation. There is also a way of generating a score by aggregating discrete classifiers into an ensemble of learners that vote. However, when only one run of our ILP method on a small *C.elegans* dataset takes about 5 hours of runtime, this method is deemed unfeasible.

Cost and class distribution are not the only factors in selecting the metric for evaluating our method. As described in [30], ILP is classified as both predictive and descriptive knowledge discovery algorithm. So far we have been looking at predictive knowledge discovery metric, however in descriptive knowledge discovery, the novelty as well as support and confidence of discovered association rules are important factors. In [30] Lavrač et al. introduce a weighted relative accuracy (*WRAcc*) measure, which is designed to take these factors into account. This measure, presented in Table 4.1, quantifies the novelty of a rule by taking into account the improvement of accuracy relative to the default rule, which classifies all instances into one set. *WRAcc* also provides a trade-off between accuracy and other predictive measures such as sensitivity, as well as being a descriptive measure, since it incorporates all values from the contingency table. Even though *WRAcc* was designed to measure the quality of the rule and not the whole theory, we have adopted this measure for the evaluation of our theory because of 2 reasons. First, the theory

can be viewed as a conjunction of rules, therefore a rule as well. Second, in a study by Todorovski et al.[52], *WRAcc* has been adopted in the rule induction algorithm and the rule sets, resulting from such improvement, have been shown to be more comprehensible with only a small drop in accuracy. Therefore we feel that there is a value in applying *WRAcc* to measure the novelty, comprehensiveness and generality of a theory. We have adopted this measure as part of our metric.

To make our results more available for bioinformatics and more general machine learning community, we also present a subset of descriptive evaluation metric. We have selected precision and recall, since they cover class imbalances and have been used extensively in machine learning community for text categorization [23]. The two measures complement each other, with recall measuring the proportion of the actual positive examples classified as positive, while precision measuring the proportion of examples, classified as positive which are also actual positive examples. In addition, the bioinformatics community usually reports its results in terms of sensitivity (also recall) and specificity. These two measures also complement each other, with specificity being the proportion of actual negative examples which are classified as negative. Therefore we have added specificity to the set of our evaluation metric. Formulas for our selected metric can be found in Table 4.1.

The expected values for the four selected metric are as follows. In the case of the perfect classification (no false positive or false negative results), recall, precision, and specificity will be equal to 1. With real data such result is hardly possible, therefore values over 0.8 are considered as good result, and over 0.9 as a very good result. Unlike recall, precision, and specificity, the expected value of *WRAcc* can never be 1, since even for a perfect classification on a balanced dataset the maximum value *WRAcc* can reach is 0.25. Therefore a result with a value of over 0.09 is considered as novel and interesting.

Cross-Validation

In accordance with the best practice in machine learning, we needed to select a data sampling method that would properly sample the data to exercise it for imbalance, noise, etc. That is why we have used two types of cross-validation techniques (5-by-2 and 10-fold), combined with presence and absence of stratification.

Contingency table:

		Predicted	
		T	F
Actual	T	**TP**	**FN**
	F	**FP**	**TN**

$$Accuracy = \frac{TP + TN}{N}$$

$$Recall = Sensitivity = TPR = \frac{TP}{TP + FN}$$

$$Specificity = 1 - FPR = \frac{TN}{TN + FP}$$

$$Precision = \frac{TP}{TP + FP}$$

$$WRAcc = \frac{TP + FP}{N} \times (\frac{TP}{TP + FP} - \frac{TP + FN}{N})$$

Table 4.1: Evaluation metric. The measures are based on a contingency table or confusion matrix. TP (true positive) and TN (true negative) correspond to the number of instances that classifier correctly accepted and rejected respectively. FP (false positive) is a number of instances the classifier has incorrectly accepted (type I error); FN (false negative) - number of instances the classifier incorrectly rejected (type II error). N is a total number of instances, or TP+FP+FN+TN. TPR and FPR are true and false positive rate respectively. WRAcc is weighted relative accuracy.

5-by-2 Cross-Validation with Random Sampling. This test consists of 5 iterations, where in each iteration we randomly divide the available data into training and test sets of the same size. Because the selection is random, we have to make sure that at least some positive and some negative examples are selected. Therefore we always pick one random positive and negative example, before continuing to randomly select examples, with no regard to whether they belong to positive or negative set. Because the selection is not stratified, the test will exercise slight imbalances in the data, even if the original dataset is balanced. These is a chance that some examples may not be selected for the tests, however this test has a benefit of having an even train-to-test data coverage. The results are measured in terms of the theory performance, i.e. how well the theory, induced on the training set, is able to classify the test set.

Stratified 10-Fold Cross-Validation. 10-fold cross-validation is a special case of a $k-$fold cross-validation and is a standard way of measuring the performance of a learning algorithm. It consists of selecting 10% of both positive and negative examples as the test set and run the test 10 times to achieve full data coverage. Because the ratio of positive to negative examples stays constant, we will only exercise imbalance in the data if the original dataset is imbalanced. Unlike 5-by-2 test, 10-fold has an uneven train-to-test data representation, however it is more consistent, because it covers whole dataset and stratification reduces the variation in fold selection. The results are measured in terms of the theory performance.

4.2.2 Measures of the Quality of the Theory

Because the primary goal of our method is to explain the data at hand, as opposed to classify the future data, the performance of our method should not be evaluated based solely on its ability to classify. Therefore we need another set of measures that would scrutinize the quality of the theory produced.

Evaluation metric

Measuring how easy it is to understand the classification is a task that still eludes machine learning community. Here we can argue that the set of rules, produced by our ILP engine can be easily translated into natural language and therefore are very easy to understand. However, for the purpose of our tests, we need a concrete metric that can compare the theories, resulting from different test runs. As described in Section 1.3,

ILP orders clauses by $\theta-$subsumption[43], when searching for a most specific theory, describing the data. So it seems that using a combined number of substitutions, needed for one theory to $\theta-$subsume another would be a good choice of metric. However, the $\theta-$substitution problem is known to be NP-complete for first order logic [26]. Therefore we have developed a set of lexical measures to compare the theories. The measures that we have used are rule and term similarity, which measure of how similar is the evaluated theory (ET) to the benchmark theory (BT).

$$Rule\ similarity = \frac{number\ of\ rules\ in\ ET\ that\ are\ the\ same\ as\ in\ BT}{number\ of\ rules\ in\ BT}$$

$$Term\ similarity = \frac{number\ of\ terms\ in\ ET\ that\ are\ the\ same\ as\ in\ BT}{number\ of\ terms\ in\ BT}$$

When we say that the term is the same in ET and BT, we mean that:

- if term is a function, function names are equal;
- constants are equal and in the same order;
- variables don't have to be equal.

For example, two terms $before(A, m1, m2)$ and $before(Z, m1, m2)$ are the same, since variables can be substituted. While $before(A, m1, m2)$ and $before(A, m2, m1)$ are not the same, since according to the first rule, PSSM *m1* comes before *m2*, while the second rule states the opposite (*m2* before *m1*).

We use both rule and term similarity, because rule similarity is a strict measure, which requires the whole rule to be matched. Since rules consist of terms, matching individual terms gives a more detailed picture of the theory similarity. The expected values in the case of a perfect theory similarity would be $Rule\ similarity = Term\ similarity = 1$. However, 100% term similarity does not necessarily mean 100% rule similarity, because the terms may belong to different rules and in different order.

Cross-Validation

The purpose of these tests is to determine how reliable is the hypothesis, induced by our method. We do it by testing it for stability on positive and negative examples. The

details of the two tests are outlined below.

Positive Example Test

With this test we want to demonstrate that slight changes to the number of positive examples does not drastically change resulting theory. We do this by using a method that is similar to $k-$fold cross-validation test. We set our benchmark theory by inducing it on a full dataset. Then we proceed to iteratively remove a small percentage of positive examples (usually 1-10%) and compare the resulting theory with the benchmark. The test is repeated as many times as needed to exhaustively exercise all positive examples. The examples that where removed on the previous iteration are placed back and different set of examples is removed. Therefore, at each iteration the number of positive examples in evaluated theory is smaller by a set percent then benchmark, while the number of negative examples stays the same.

Negative Example Test

With this test we want to explore the effect imbalanced data has on the theory. We start with the largest number of negative examples and proceed by decreasing them until negative example size equals to positive. How large is the beginning negative set is determined in individual test runs. The results of the runs are averaged over 10 executions. The theories are compared on rule and term similarity.

Rule Induction Test

This is a basic test to check if our method is able to uncover the rules present in the data. This test can be thought of as an equivalent of a unit test for ILP, where we run our method on a set of data which was designed to contain a certain rule and then test if the resulting theory contains this rule. Designing such a test is, unfortunately, not as straightforward as a classic unit test. The first challenge is the data. Using independent and identically distributed model, we can calculate that a word of 6 letters (usual PSSM length) will occur roughly once every 5000 characters. which means that creating a background sequences, in which to plant PSSMs according to some rule, will produce noisy data. Our synthetic data experiments in Section 4.1 confirms this suspicion. The second challenge lies in the heuristic of learning itself. When the ILP engine traverses a tree of possible term combinations in search of a suitable theory, it may terminate before reaching the rule we have planted if another rule, or combination of rules, meets its noise and

accuracy requirements. For example, if we randomly plant one PSSM before another in hopes of getting a theory that contains a *before* rule, we might get the *distance_interval* rule instead, because, by chance, the PSSMs might have been planted at some particular distance. Or if both PSSM sequences are rare and found very infrequently in the negative set, the learning process might terminate at *has_feature* rule for one of the PSSMs, since it sufficiently covers the positive set.

Keeping these challenges in mind, we have designed an experiment to test the theory generation of our method. Using the data from our synthetic data selection experiment 4.1, we have selected two PSSMs, which we will call *pssm1* and *pssm2*, that are least likely to occur in sequences with equal nucleotide distribution. We then have created 10 positive background sequences of length 200 (usual length of an upstream sequence) with equal nucleotide distribution. By having only 10 positive sequences, we will keep the noise associated with the change occurrence of one of the two selected PSSMs to a minimum. Then we have randomly planted *pssm1* before *pssm2* in every positive sequence. The negative examples consisted of 15 sequences, generated the same way as the positive background sequences, only without the planting. The input to our program also consisted of 8 random PSSMs from *C.elegans* and human data experiments, apart from the two planted PSSMs.

The success of this test depends if the originally planted rule $before(A, pssm1, pssm2)$ is found in the resulting theory.

Chapter 5

Results

In this chapter we present the results obtained with our method using the techniques described in Chapter 4.

5.1 Rule Induction Results

Before starting the cross-validation tests, which were described in Section 4.2, we need to make sure that our approach is working and able to find significant rules in data.

We have repeated the rule induction experiment, described in Section 4.2.2, 60 times and found that 57% of times the induced theory contained *before* term with the two planted PSSMs, with 37% of theories consisting of only the *before* rule. The 43% of theories that did not contain *before* term, contained instead a *distance_interval* rule with the planted PSSMs. This is because the randomly generated data for these theories placed the PSSMs at a particular distance, which is not surprising, given the small amount of sequences that we tested. Therefore we can confidently assume that our method can uncover significant rules in the data.

We have also repeated this experiment for the larger amount of positive and negative examples, to investigate how our method would perform on a more realistic dataset. Running it with 600 positive and 600 negative examples of length 200 a total of 20 times (only 20, to keep the execution time reasonable), we have found that 41% of the runs contained a planted *before* term, while 14% contained the whole rule. The 58% contained either *distance_interval* term, similar to the previous experiment, or *has_feature* term with one of the two planted PSSMs. Thus we also obtain confidence that introducing

noise to the data, by means of larger number of sequences, our approach is still able to discover significant rules.

Having obtained confidence in the ability of our method to discover rules, we have proceeded with formal testing.

5.2 Synthetic Data Results

Biologists often dispute the value of evaluating a bioinformatics method on a set of synthetic data. However we feel that synthetic dataset allows us to perform the most rigorous testing of our method, since we are limited to neither the number of positive or negative examples, nor the length of example sequences. Thus we can assess the impact of various factors, like noise and data imbalance, on the performance of our method in a systematic and controlled way, which will gives us confidence in the results achieved on a real dataset.

To make our synthetic data as realistic as possible, we have selected to use a positionally planted PSSMs method, described in Section 4.1.2, for positive sequence generation. This method takes positional probabilities of the PSSM location in model data into consideration, when planting same PSSMs into the background sequences. This makes it the most data over-fitting method among the ones we have considered. The data we have selected as a model for positive sequence generation is *C.elegans*, because, unlike human ChIP-Seq data, it provides long input sequences (2000 nt long). Having longer model sequences provides for more realistic long sequence generation in the case of data with positionally planted PSSMs. The negative sequences were generated using Markov chain order 1 (MC1) model with *C.elegans* nucleotide frequencies. MC1 is the least data over-fitting method out of the 3 methods considered.

The following tests were executed to see how our method will perform under different noise and data balance conditions.

5.2.1 5-by-2 Cross-Validation

For the test description, see Section 4.2.1. This test gives us a general idea of how our method will perform on a randomly balanced dataset. We have performed it on a set

Recall	Specificity	Precision	WRAcc
0.88 (0.072)	0.8 (0.16)	0.85 (0.082)	0.15 (0.005)

Table 5.1: Results of a 5-by-2 cross-validation test on a synthetic dataset. The values are presented with their variance in brackets.

of 5000 positive and 5000 negative sequences, with sequence length set to 1000 and the results are presented in Table 5.1. The test achieves a high sensitivity and specificity numbers, as well as good weighted relative accuracy, which shows that the theory is novel and interesting. However, since this test is unstratified, we can not assume that our method will perform as well on any type of data. Therefore we proceed with 10-fold validation.

5.2.2 10-fold Cross-Validation

For the test description, see Section 4.2.1. This test preserves original data balance, therefore we proceed with testing the following hypotheses.

Stability on the sequence length and number of examples

Here we test if our approach is stable for the different number of positive and negative examples and different lengths of the sequences. We want to establish if our method favors larger datasets and if it performs better on longer sequences. This is beneficial, because it may determine what kind of biological experiments would be best analyzed with our method. For instance, in *C.elegans* experiment, where we are looking for the CRM in the upstream regions of the genes, the length of sequences are relatively large (2000 nt), while the number of sequences is small (18 sequences). While in ChIP-Seq experiments, the sequence length is usually small (around 200 nt), while the number of sequences is very large.

The test we have designed, runs repeated 10-fold cross-validation on a varying number of positive examples (*seqNum*) and varying number of sequence lengths (*seqLen*). Since we are not testing for imbalance of data, the number of positive and negative examples are kept equal. We kept the number of characters in dataset the same through all the runs (i.e. $seqNum \times seqLen = 10000$). This way we model the data to contain approximately the same number of PSSM matches, therefore the ILP engine will have the

same amount of data to induce the theory. We have repeated this test 5 times, varying $seqNum \times seqLen$ ratio from 50 sequences of length 200 to 400 sequences of length 25. We did not select a larger dataset size in the interest of time, since ILP is a computationally intensive process and we need to perform a total of 50 inductions for this test. The results of the test are presented in Figure 5.1. The difference in values between test runs for specificity, precision and WRAcc are not statistically significant (based on variance). The values for recall show a slight preference (order of 0.06) of our method for a larger ratio of $seqNum/seqLen$, however the average value of recall for the 5 test runs is much smaller then the average recall in 5-by-2 test. This could be due to several reasons. A larger dataset could have resulted in fewer false negative classifications and therefore improved the recall. It is also possible that the 5-by-2 test result is higher then actual, since this test is less rigorous for the lack of stratification and because it was repeated only 5 times. However in this test the focus is not on achieving high classification result, but on comparing the classification performance on various sequence lengths. The results show no statistically significant difference for much larger vs much smaller sequences (opposite ends of the recall graph). Therefore, based on this test, we can not conclude that our method performs better with longer or shorter sequences.

Stability on the dataset size

Here we test the hypothesis that larger dataset size improves the quality of the theory.

The test set up is as follows. We keep the length of the sequences the same and equal to 150 nt. We then gradually increase the number of positive and negative examples to 2000. Since we are not testing for imbalance of data, the number of positive and negative examples are kept equal. Similarly to the previously described test, we do not test with larger number or longer length sequences in the interest of time. The results of the test are presented in Figure 5.2.

Visual inspection of the results shows that there is a simultaneous spike in the specificity, precision and recall around the 150 number of examples. The results also reflect the same tendency, as discussed in the previous test, when performance number for these 3 values are slightly better when sequence length is approximately the same as the number of examples. After the spike, however, specificity, precision and weighted relative accuracy show no statistical difference based on variance. Perhaps generating more sequences with the same method does not contribute to the improvement in data

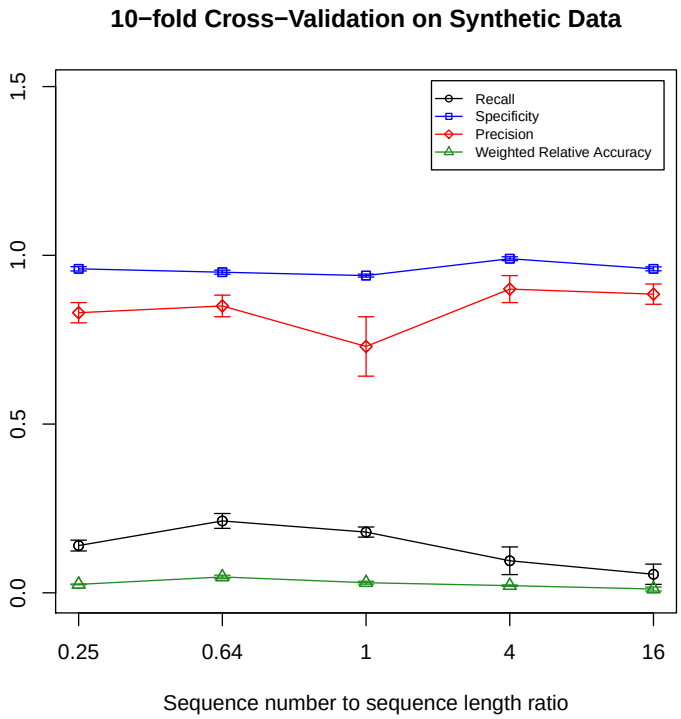

Figure 5.1: 10-Fold cross-validation results on synthetic data with varying sequence length and dataset size.The x axis represent the ratio of dataset size to sequence length, therefore the further left the point is from the middle of the graph, the larger the sequence length and the smaller the dataset size. Each test run is presented with its variance interval.

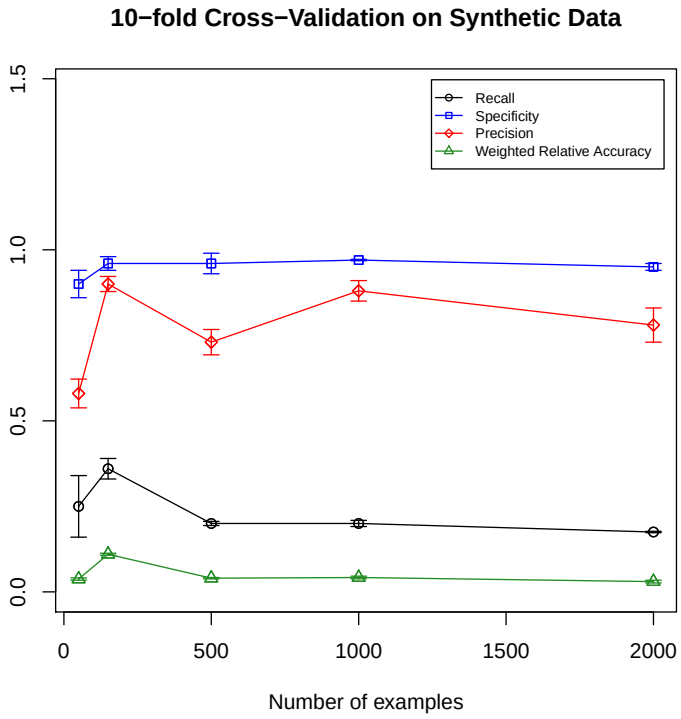

Figure 5.2: 10-Fold cross-validation results on synthetic data with varying dataset size.The x axis represent the number of positive and negative examples. The length of the sequences are 150 nt. Each test run is presented with its variance interval.

quality and therefore does not result in the increased performance. However, similar to the finding in the previous 10-fold validation test, we see that the average recall for this test is much smaller, then the one reported in 5-by-2 test. Perhaps continuing to increase the dataset size would result in the increased performance. But based on the available results we can not conclude that larger dataset size improves the quality of the theory.

Stability on imbalanced dataset

Here we test the behavior of our system with increasing number of negative examples. Our hypothesis is that the more negative examples are added, the higher the specificity and the lower the precision, recall and accuracy. We expect that when the data is extremely imbalanced, with many more negative examples than positive, the classifier assigns all instances to the negative set, therefore specificity becomes almost 1. While

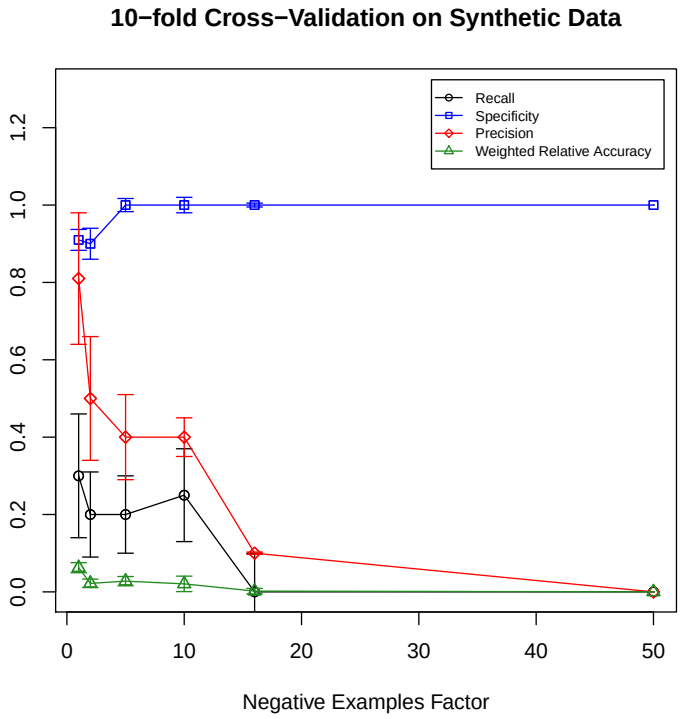

Figure 5.3: 10-Fold cross-validation results on synthetic data with varying size of negative examples. The x axis represents the number of times negative examples size is larger then positive. Each test run is presented with its variance interval.

the small number of positive examples become indistinguishable, the recall (sensitivity) and precision become close to 0. Since WRAcc measures the difference between the default rule and the theory, when all instances are classified as negative, this measure also becomes 0.

We have executed 10-fold cross-validation on progressively bigger negative examples size, while keeping the positive example set the same. Sequence length was selected at 200 and positive example set was 100. The results are presented in Figure 5.3.

Visual inspection of the results shows that they confirm our hypothesis. With 50 times more negative examples than positive, specificity becomes 1 with no variance, while recall, precision and weighted relative accuracy become almost 0 with negligible variance.

5.2.3 Positive Example Test

Here we test the hypothesis that removing a small number of positive examples does not drastically change the resulting theory. Because our method was designed to withstand 0.5% noise and consider rules with at least 90% accuracy, we expect that removing 5% of positive examples (as described in Section 4.2.2) should produce theories that have very high rule and term similarity.

After executing the test on an set of 1000 sequences of length 300 we found that mean rule similarity was 0.95 while mean term similarity was 0.975. This result confirms that our method is able to withstand small changes in the number of positive examples.

5.2.4 Negative Example Test

Here we explore the effect that changing the number of negative examples has on the resulting theory. Our hypothesis is that gradual change in the number of negative examples will introduce a gradual change in the theory similarity. We designed this test to see how large the imbalance has to be for the theories to be different. We start with the most imbalanced dataset, with 16 times more negative then positive examples. We have selected this number based on the previously described test for stability on imbalanced dataset 5.2.2, since at this point the system was still able to classify some positive examples correctly. The theory, induced on this dataset, is our benchmark theory and we proceed by decreasing the number of negative examples and comparing the theories using rule and term similarity measures (see Section 4.2.2 for details). The test was repeated 10 times and the results were averaged. The results are presented in Figure 5.4.

The results confirm our hypothesis. From the slope of the graph we can see that gradual change in the number of negative examples makes the theories gradually more different. We see that with negative example factor drop from 16 times to 4, rule similarity becomes 0 with 0 variance. However, the results are still good, since theories on balanced and 16 times imbalanced datasets still contained some of the same terms in 8 out of 10 test runs and the number of rules in all theories was the same as in benchmark theory (2 rules).

We also compared each theory during the run with its preceding theory (i.e one factor decrease in number of negative examples) and found that both similarity measures are high and stable for each run. This result shows that, similar to the positive example

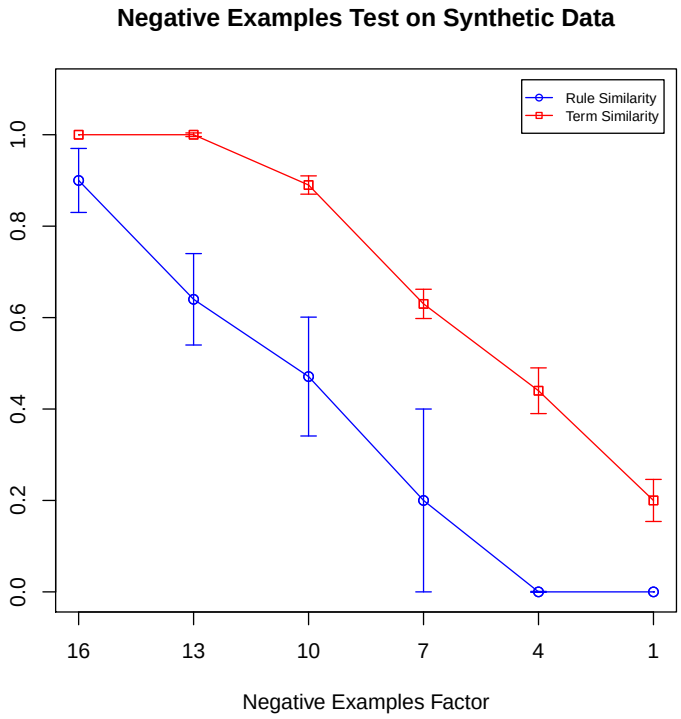

Figure 5.4: Decreasing number of negative examples test. The x axis represents the number of times negative examples size is larger then positive. The length of the sequences are 200 nt.

test, increasing the number of negative examples by a small factor produced a consistent theory.

5.3 *C.elegans* Results

For *C.elegans* data, we have a fixed amount of positive sequences (18) and the set of PSSMs, described in Section 4.1.3. However, since the negative sequences were not provided in the study, for positive-only learning we have experimented with 3 types of synthetic sequences: Markov chain order 1 (MC1), synthetic with randomly planted PSSMs and synthetic with positionally planted PSSMs. The results are presented below.

	Recall	Specificity	Precision	WRAcc
MC1	0.27 (0.046)	0.77 (0.13)	0.6 (0.004)	0.014 (0.0012)
Randomly Planted PSSMs	0.74 (0.065)	0.78 (0.0088)	0.84 (0.003)	0.124 (0.0003)
Positionally Planted PSSMs	0.681 (0.14)	0.921 (0.01)	0.958 (0.0012)	0.137 (0.0005)

Table 5.2: Results of a 5-by-2 cross-validation test on a *C.elegans* dataset with 3 types of negative examples. The values are presented with their variance in brackets.

5.3.1 5-by-2 Cross-Validation

This test, as described in Section 4.2.1, performs sampling of training and test data without stratification. Which means that we can not predict the positive to negative example ratio for each test run. This gives us a general idea of how the our method will perform on a similar balanced dataset.

The results of the test are presented in Table 5.2. We have used variance to compare the results of learning with three types of synthetic sequences. Markov chain order 1 performs worse on all metrics, but specificity. In our case, high specificity is not as desirable, as high sensitivity (or recall), because we would rather have a classification that includes more correctly identified positive examples, then exclude negative. The two methods that generate negative examples by planting PSSMs show a statistically better performance on recall, precision and weighted relative accuracy. The last measure is especially important, since it takes into account the novelty and descriptiveness of the theory. Out of two planted PSSMs methods, positionally planted can be said to perform better on specificity, precision and weighted relative accuracy, without a statistically significant difference in the performance based on recall. This makes it a negative example of choice, based on a 5-by-2 test.

5.3.2 10-Fold Cross-Validation

Stratified 10 fold validation, described in Section 4.2.1, is a more widely used test in machine learning, than 5-by-2, because it systematically covers all positive and negative examples in the same proportion, as the original dataset. Therefore we were content to find that the results of this test, presented in Table 5.3, are similar to the 5-by-2. We again used variance to compare the results for the 3 types of negative examples and found that positionally planted PSSMs sequences perform better on recall and weighted rela-

	Recall	Specificity	Precision	WRAcc
MC1	0.56 (0.089)	0.75 (0.5)	0.68 (0.12)	0.055 (0.0063)
Randomly Planted PSSMs	0.38 (0.063)	0.88 (0.41)	0.56 (0.2)	0.032 (0.0009)
Positionally Planted PSSMs	0.69 (0.18)	0.875 (0.049)	0.7 (0.17)	0.14 (0.052)

Table 5.3: Results of a 10-fold cross-validation test on a *C.elegans* dataset with 3 types of negative examples. The values are presented with their variance in brackets.

tive accuracy then any of the other two types of sequences. It has better specificity than MC1, while statistically the same specificity as randomly planted. Regarding precision, positionally planted method performs better than randomly planted, while no distinction in performance can be made with MC1 method. Therefore we can say that positionally planted negative example sequences perform better based on the 10-fold cross-validation test.

The results of the two cross-validation tests also confirm our expectation that our method performs well on finding relational dependencies in the data. Planting PSSMs in negative sequences depending on their position in positive sequences will inherently force our learning algorithm to find more relational rules, than rules of class membership. The fact that our algorithm is able to find more interesting rules (based on WRAcc) as well as perform better on classification shows that our method has merit.

5.3.3 Positive Example Test

Just like in Section 5.2.3, we would like to test the hypothesis that our theory will not be drastically different when we remove a small amount of positive examples. Since *C.elegans* data only has 18 positive sequences, we ran this test as a leave-one-out cross-validation test, removing only one positive sequence in each run. The results, presented in Table 5.4 confirm our hypothesis.

Based on the variance, we can say that the results for 3 types of negative sequences are not significantly different. Both rule and term similarity are high, confirming our hypothesis.

	Rule Similarity	Variance	Term Similarity	Variance
MC1	0.94	0.044	0.967	0.028
Randomly Planted PSSMs	0.92	0.02	0.98	0.04
Positionally Planted PSSMs	0.92	0.067	0.96	0.03

Table 5.4: Results of a positive example test on a *C.elegans* dataset with 3 types of negative examples.

5.3.4 Negative Example Test

Just like in Section 5.2.4, we would like to test the hypothesis that our theory will not be drastically different when we gradually increase the number of negative examples. Because of the length of *C.elegans* sequences and the running time, we have started with 8 times more negative than positive sequences. We have also selected to run the test on the positionally planted dataset only, because it has been shown to perform better, based on the 5-by-2 and 10-fold cross-validation. The results are presented on Figure 5.5.

From the visual inspection of the results we can see that the theory similarity decreases gradually with the number of negative examples, which confirms our hypothesis.

5.3.5 Comparison to the Original Study

With the previous tests we have shown that our method achieves good accuracy at classifying and describing *C.elegans* data. Now we would like to compare the results of our method with the original study. Presented below is the theory that our method produced on *C.elegans* data, described in Section 4.1.3 contrasted with synthetic data with positionally planted PSSMs.

```
[theory]

[Rule 1] [Pos cover = 10 Neg cover = 1
positive(A) :-
   has_feature(A, m1, 'R'), distance_interval(A, m9, m16, 20, 100).

[Rule 2] [Pos cover = 5 Neg cover = 0]
positive(A) :-
```

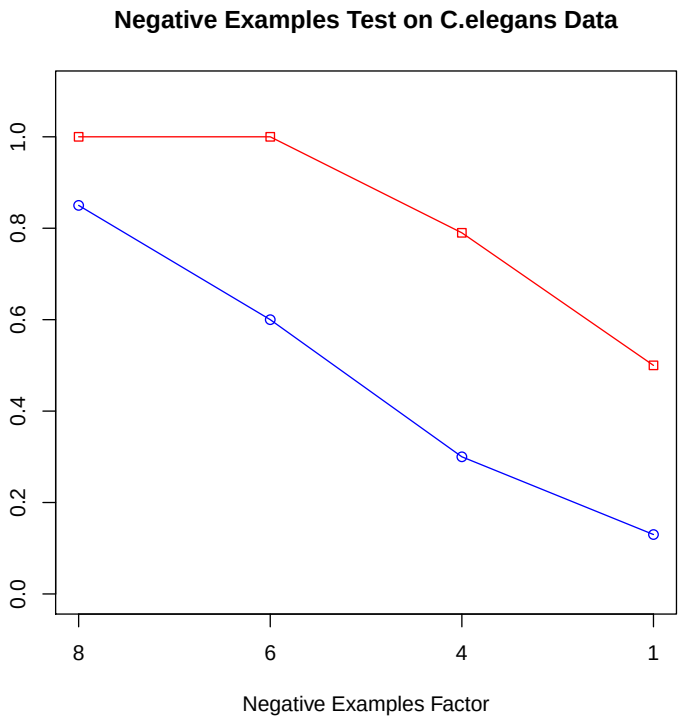

Figure 5.5: Decreasing number of negative examples test on a *C.elegans* dataset with positionally planted PSSMs method of negative sequence generation. The x axis represents the number of times negative examples size is larger then positive.

```
distance_interval(A, m4, m3, 40, 10).
```

In the original study, the authors achieved good results on this dataset, by identifying muscle-specific CRMs based on clustering of motifs. The cluster was identified as muscle-specific regulatory module if it contained at least 2 motifs, if the distance between motifs was no more than 40 nt and if it contained at least 2 muscle-specific binding sites. Our method was able to recover these assumptions. The rules cover a total of 5 motifs (`m1`, `m9`, `m16`, `m4`, `m5`). Four out of these motifs (`m1`, `m3`, `m4` and `m16`) correspond to the top 5 muscle-specific motifs, identified in the study. The Rule 2 identifies the distance between `m3` and `m4` to be 40 nt +/- 10 nt. However the results of our method show much more then assumptions in the original study. We see that in 10 out of 16 sequences motifs `m1`, `m9`, and `m16` always occur together and the distance between `m16` and `m9` is no more then 120. We also see that the distance of 40 nt is only significant for motifs `m3` and `m4`. These findings could be useful for refining the assumption on which the CRM predictions are based or at guiding future experiments to determine how motifs are organized to form muscle-specific regulatory modules.

5.4 Human Genome Results

The results of our testing on synthetic and *C.elegans* dataset have shown that our approach is able to induce high quality theory on a mostly balanced dataset, disregarding the length of sequences or the size of the dataset. This has made it a good candidate to analyze the data from the recent study by Palii et al. [39]. As described in Section 4.1.4, the study was able to determine one relation rule: 8-10 nt distance between E-box and Gata transcription factor binding sites. Our hypothesis is that our method will be able to uncover the same rule as well as find new rules, describing the data.

5.4.1 Jurkat and Erythroid vs. Control

Because the original study found the distance rule for E-box and Gata transcription factor binding sites in jurkat and erythroid sequences separately, contrasting them with control, to repeat the experiment with our method we performed 2 runs:

- jurkat vs. synthetic with positionally planted PSSMs;
- erythroid vs. synthetic with positionally planted PSSMs.

We have selected negative examples as synthetic sequences with positionally planted PSSM based on our findings with *C.elegans* data, presented earlier. The resulting theories for jurkat and synthetic runs are presented in Additional Information sections B.3 and B.4 respectively. None of the runs reproduced the result of Palii et al. and did not contain a rule with a distance 8-10 nt between Gata and E-box in 6% jurkat and 10% erythroid sequences is not confirmed. One of the reasons for this could be that in original study, the authors looked for an exact match to a motif sequence to find E-box and Gata, while we matched PSSMs, found by TOMTOM, based on the motif sequence. Since we have selected on average two PSSMs per each motif, this extra step could have contributed to more matches for the same TFBS, thus making the rule less distinguishable. Another reason is that the ILP search for this run has taken a direction that found other representative rules first, thus making the distance rule unnecessary for the final hypothesis.

However, the induced theories confirm several other findings in the study. For instance, the authors have discovered that despite expecting to see E-box as the most overrepresented motif in in both jurkat and erythroid sequences, Gata has ranked top for erythroid and Runx for jurkat sequences. If we examine the combined coverage of rules which contain Runx in jurkat theory and Gata in erythroid, we see that our results confirm this finding.

Another finding that is confirmed by our result is over-representation of Runx in jurkat sequences compared to erythroid. The combined coverage of rules, containing Runx in jurkat sequences is 37%, compared to 17% in erythroid. In addition, our results show that in at least 17% of the sequences, Runx is found at least twice in each sequence. This is evident from the rules like `before(A,'runx1','runx1')`, when both transcription factors are the same. Also, in at least 5% of the sequences, the distance between the double occurrence of Runx is 11 +/− 15 nt. This trend has not been observed in erythroid sequences. To our knowledge, this transcription factor is not known to form dimers with itself, therefore it would be an interesting avenue for a biologist to explore.

The jurkat and erythroid theories also offer a number of interesting findings, not discussed in the article. One of the interesting trends is that a motif, which was not matched to any known TFBS in the original study is present in rules that cover 18% of jurkat and 30% erythroid sequences. The motif in question was identified as Hand1 transcription factor

by TOMTOM. Hand1 is a basic Helix-Loop-Helix (bHLH) transcription factor, critical for heart development and contributing to the development of multiple cellular lineages linked to heart [59]. Down-regulation or silencing of this transcription factor has been linked to repressed cell growth in several types of cancer, including colorectal [25, 62] and thyroid [22]. To our knowledge, it has not been linked to the T-cell leukaemia, which was the cancer studied in Palii et al. In addition, in 15% of jurkat sequences, Hand1 TFBS is found on the same sequence where Runx binds twice. Perhaps this contributes to the down-regulation of Hand1 in jurkat sequences.

Another interesting trend is that many rules in both theories contain TFBS matched to the reverse strand. In particular, the rules with the highest coverage in jurkat (rule 2, 9%) and erythroid (rule 2, 14%) theories require all of participating transcription factors (Gata and Runx for jurkat; Gata and Hand1 for erythroid) to be on the reverse strand. We could not find any explanation to this observation in the literature. Perhaps a closer examination by a biology expert might shed light on this finding.

5.4.2 Jurkat vs. Erythroid

In original study the authors report that Runx was a top ranking over-represented motif in jurkat sequences compared to erythroid. The rule of Gata/E-box TFBS distance being 8-10 nt was not found to be over-represented when jurkat was contrasted with erythroid sequences. To confirm this finding and to discover other rules, not examined in the study, we have performed a run of our method on jurkat vs. erythroid sequences. The resulting theory is presented in Additional Information Section B.2

First we would like to mention that the resulting theory contained no rules with significant coverage. Increasing the number of positive examples, covered by each rule (see 3.2.3) resulted in no rules in the theory, while using Aleph default for this parameter failed to generalize most of the rules (i.e. theory consisted of rules stating that jurkat sequence N belonged to a positive set). This is in line with the findings of the computational analysis by Palii et al., which could not uncover any significant rules contrasting these two datasets. However, when examining a theory as a whole, some trends emerge. One trend confirms the original finding of Runx over-representation, since almost every rule in the theory contains Runx. Another interesting trend is that Runx is found at

least twice in more than 19% of the sequences. This is slightly higher than the finding in jurkat vs. synthetic theory, however, since ILP is not an exhaustive statistical method, more testing needs to be done to determine statistical significance of this finding. Also, similar to the findings in jurkat vs. synthetic theory, frequently the double occurrences of Runx were found to be close together.

Another interesting finding involves Hand1 transcription factor, which we discussed in the previous section. 141 rules out of 366 rules in the theory contain a reference to this TFBS. In addition, in most of the cases, the rule that contained Hand1 binding also contained Runx. Based on these findings, if we were to design a deterministic computational analysis experiment, we would examine Hand1 binding and its relationship to Runx binding in detail.

Chapter 6

Conclusion

In this work we have used ILP to find relationships between biological markers in a set of related sequences. This problem is broadly defined and can include a variety of important biological questions, like finding *cis*-regulatory modules, studying the impact of transcription factors binding on the gene transcription, or analyzing the results of ChIP-Seq experiments. Our approach was shown to have merit in two respects: as a predictive tool, able to accurately classify the data and predict which class the future instances of data should belong; as well as a knowledge mining tool, able to find new rules, describing the data. As recently noted by Muggleton et al. [36], ILP is difficult to use for a non-ILP specialist. With our ModuleInducer solution we introduce an intermediary interface and data management layer which hides the complexity of ILP and makes the method accessible to a wide range of users. We have also shown that using the method of positionally planted PSSMs to generate synthetic data for negative examples is more beneficial in positive-only learning with ILP than the widely used Markov chain of order 0 or 1 model.

One of the limitations of our approach is the lack of widely-accepted evaluation framework for ILP. Popular machine learning evaluation techniques that are based on cost of classification, like ROC curves and cost curves, could not be applied in our case due to the missing cost information and long execution time. Other single value metrics, like sensitivity and specificity, do not assess such important aspects of ILP performance as the novelty and comprehensibility of the theory. We address this issue in two ways. First, we develop the rule and term similarity measures to lexically compare theories and apply them on a set of specially designed tests, executed on the synthetic dataset. Second, we adapt a weighted relative accuracy metric to measure the novelty of the theory. However,

there are some disadvantages to this approach. It is based on the contingency table, and therefore, like other metrics that are based on the results of the classification, does not capture comprehensibility of the theory. Also, this metric was originally developed to asses the novelty of a rule and its applicability on a rule set has not been determined yet.

Another limitation of our work is a lack of comparison of our method with other methods on the same dataset. However, because ILP represents data in terms of relationships, applying any propositional classifier on the ILP dataset without the use of propositionalization is not possible. One of the future developments of this research could be applying propositionalization on the datasets, described in this work, to compare the accuracy and the running time of different methods. It would also be possible to apply ILP on a propositional dataset, omitting any relational rules from the background knowledge, however this will take away the main advantage of ILP and other propositional learners should be used in this case.

Another limitation of our method is the long running time, which is the result of a computational complexity of ILP. The factors that contribute to this are the number and length of analyzed sequences, number of PSSMs, number of rules in the background knowledge and Aleph search parameters, described in Section 3.2.3. Since the size of the dataset is determined by the biological experiment, the ways to address the execution time may include:

- Including fewer rules in the background knowledge. Since each biological experiment has its unique goals, the expert may decide to eliminate and/or replace some of the suggested background knowledge rules with the ones tailored for that specific experiment.
- Modifying Aleph search parameters. To achieve acceptable classification accuracy while keeping the execution time reasonable, we have determined a set of default Aleph search parameters, described in Section 3.2.3. However, these parameters can be modified to decrease the running time, when accuracy of the classification is not as important as finding novel rules with high coverage.

Other way to improve the execution time is to implement experiment-specific pruning, to decrease Aleph's search space. We can also suggest future improvements to the Aleph engine by parallelization of the search procedure, such as described in Srinivasan et al. [50]

An interesting research direction could be to study the effect of selecting fewer examples. We can see two possible benefits of this direction - shorter execution time and fewer, more general rules in the resulting theory. Input sequences could be rated based on the presence or absence of binding sites of interest. Simple distance measures, such as the Hamming distance could then be used to classify the examples for the purpose of selecting one representative (centroid) element per cluster. This clustering procedure can also improve the performance of the system on an unbalanced dataset, when applied to the bigger dataset (usually negative examples, since they are easier to come by).

Very often what interests biologists is not the accurate classification of the experiment's data, but what rules are enriched in the positive set, compared to negative. In this respect, our method could be improved by using the Annotation Concept Synthesis and Enrichment Analysis (ACSEA) approach, developed by Jiline et al. [24], instead of traditional ILP, used in Aleph. ACSEA uses statistical inference, instead of compression, to decide on the best clause to select during the search for the best theory. This technique has shown good results on microarray and genetic interaction datasets. As an added benefit, ACSEA engine reports a $P-$value for each induced rule, making it easier for a human expert to decide on the significance of the theory, as well as making our method amendable to cost curve analysis.

The application of our method on jurkat and erythroid cell lines from Palii et al. [39] did not produce any rules with significant coverage. One of the reasons for this could be that our search space was not rich enough. It might be possible to enrich it by using DREME [4] or other motif finding algorithm to find diverse enriched motifs. Perhaps incorporating information other than motifs, such as DNA methylation or histone modification may increase the chances of finding a significant difference between the two cell lines.

One of the contributions of our approach is the end-to-end analysis cycle, complete with a Web interface and a data pre-processing, which is easy to use by a researcher in biology. However, the interface that has been developed so far does not queue multiple requests. Since ILP is a computationally expensive process, we had to limit the size of the dataset that could be submitted for analysis through the Web. As a future development, we plan to implement user request distributed system to allow for a larger dataset size. A

number of other future development to the user interface could be suggested, such as:

- Exposing some of the available system parameters to the user. The parameters may include Patser cut-off score for accepting a PSSM match and ILP search parameters (accuracy, noise, etc. 3.2.3). Modifying these parameters for the same dataset and comparing the resulting theories may be beneficial for accurate interpretation of the experiment results.

- Translating the resulting theory into English for simpler interpretation of the results, as well as visual aids to assist the analysis of the results.

When working with human data from Palii et al., where biological markers of interest were presented in the form of IUPAC strings, we realized the benefit of having different methods of matching motifs to a sequence. Since we designed the architecture of our application with the concepts of object oriented programming in mind, it is not difficult to add new data processing modules to our system. Therefore we plan to add a regular expression matcher module, in addition to the existing Patser PSSM matcher.

In addition, we feel that verifying the results of our method with the experts, who designed the original biological experiment could provide us with many more ideas on how to improve our method.

Finally, the methods was tested on synthetic and two real world datasets, further experiments will help better understand the limitations of this framework.

Appendix A

Glossary of Terms and Abbreviations

A, C, G, T - abbreviations of the four bases of the DNA, which correspond to: A - adenine ; C - cytosine; G - guanine; T - thymine.

Gata, E-box, Runx, TAL1 transcription factor binding sites found in the study by Palii et al.[39].

ILP inductive logic programming. For definition, see Section 1.3.

ChIP-Seq ChIP-Sequencing. For description, see Section 1.2.2.

CRM *cis*-regulatory module. For definition, see Section 1.2.1.

FOL first order logic.

IUPAC sequence . Here refers to a systematic naming convention by IUPAC (International Union of Pure and Applied Chemistry) of a DNA sequence, which apart from the usual A, C, G, T alphabet, also contains 11 other characters, like N (= any character), R (= A or G), etc .

ML machine learning.

nt nucleotide. Typically referred to as a letter from a DNA sequence.

PSSM position-specific scoring matrix. Means to represent a motif in a DNA sequence.

SVM support vector machine. One of the propositional classifiers in machine learning.

TF transcription factor. For description, see Section 1.2.

TFBS transcription factor binding site. For description, see Section 1.2.

TSS transcription start site. Marks the first nucleotide at which transcription starts.

Appendix B

Additional Information

B.1 PSSMs Used with Human Data

PSSMs, presented here, were selected fromJaspar and Transfac databases. We matched the sequences, reported in the article by Palii et al.[39], using TOMTOM.

Table B.1: PSSM: ebox1_ma0048.1_jaspar

A	13	13	3	1	54	1	1	1	0	3	2	5
C	13	39	5	53	0	1	50	1	0	37	0	17
G	17	2	37	0	0	52	3	0	53	8	37	12
T	11	0	9	0	0	0	0	52	1	6	15	20

Table B.2: PSSM: ebox2_m00414_transf

A	3	1	5	0	12	0	0	0	0	0	3	3
C	5	4	0	12	0	11	11	0	0	1	1	6
G	4	2	7	0	0	0	1	0	12	5	5	2
T	0	5	0	0	0	1	0	12	0	6	3	1

Table B.3: PSSM: ets1_ma0144.1_jaspar

A	20	13	38	6	321	8	6	585	606	191
C	19	10	552	541	21	0	2	21	1	15
G	25	129	9	1	148	605	592	7	5	393
T	549	461	14	65	123	0	13	0	1	14

Table B.4: PSSM: ets1_ma0156.1_jaspar

A	3	0	2	0	0	18
C	5	0	0	0	0	0
G	4	19	18	19	20	2
T	8	1	0	1	0	0

Table B.5: PSSM: gata1_m00011_transf

A	16	0	15	16	0	16	0	15	12
C	0	16	0	0	0	0	0	0	1
G	0	0	0	0	16	0	0	0	3
T	0	0	1	0	0	0	16	1	0

Table B.6: PSSM: gata1_m00078_transf

A	7	0	13	0	12	13	0	13	0	11	13	0	13	0	11	11
C	0	0	0	5	0	0	0	0	3	0	0	0	0	1	1	1
G	1	13	0	1	0	0	13	0	0	0	0	13	0	1	0	0
T	5	0	0	7	1	0	0	0	10	2	0	0	0	11	1	1

Table B.7: PSSM: gata1_m00080_transf

A	23	0	27	0	26	27	0	27	0	26	19
C	0	0	0	2	0	0	0	0	0	0	1
G	0	27	0	0	0	0	27	0	0	0	4
T	4	0	0	25	1	0	0	0	27	1	3

Table B.8: PSSM: gata1_ma0029.1_jaspar

A	14	20	0	27	1	27	26	0	27	0	24	23	6	15
C	2	1	1	0	10	0	0	0	0	3	1	0	7	6
G	6	2	25	0	0	0	1	27	0	0	0	4	7	3
T	5	4	1	0	16	0	0	0	0	24	2	0	7	3

Table B.9: PSSM: gata3_m00346_transf

A	3	2	4	0	10	0	10	10	2	6
C	2	6	2	0	0	0	0	0	6	2
G	4	1	0	10	0	0	0	0	2	1
T	1	1	4	0	0	10	0	0	0	1

Table B.10: PSSM: gata3_m00348_transf

A	15	7	17	0	31	0	30	29	4	17
C	5	7	5	0	0	0	1	0	9	3
G	7	11	0	31	0	0	0	0	17	7
T	4	6	9	0	0	31	0	2	1	4

Table B.11: PSSM: gata3_ma0035.2_jaspar

A	1423	708	2782	0	4000	27	3887	3550	799	1432	1487
C	560	1633	31	0	0	29	0	4	681	897	829
G	1242	1235	10	4000	0	109	6	383	2296	1360	1099
T	775	424	1177	0	0	3835	107	63	224	311	585

Table B.12: PSSM: gata4_m00127_transf

A	4	1	1	2	3	0	12	0	0	8	1	3	3	2
C	1	1	2	2	0	0	0	0	0	1	4	4	1	4
G	2	3	4	2	2	12	0	0	0	3	4	3	7	4
T	0	2	0	1	2	0	0	12	12	0	3	2	1	2

Table B.13: PSSM: gcbox_MA0073.1_jaspar

A	3	1	3	0	7	9	8	4	0	11	4	1	3	4	2	4	4	4	1	4
C	8	10	8	11	4	2	3	6	11	0	7	10	8	6	9	5	5	6	7	4
G	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	3	2
T	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	1	1	0	1

Table B.14: PSSM: non3_ma0092.1_jaspar

A	4	10	2	0	0	0	0	9	16	5
C	8	0	2	28	0	0	3	14	0	4
G	10	15	1	0	0	29	25	1	3	4
T	7	4	24	1	29	0	1	5	10	16

Table B.15: PSSM: runx1_MA0002.2_jaspar

A	287	234	123	57	0	87	0	17	10	131	500
C	496	485	1072	0	75	127	0	42	400	463	158
G	696	467	149	7	1872	70	1987	1848	251	81	289
T	521	814	656	1936	53	1716	13	93	1339	1325	1053

B.2 Theory Induced on Jurkat vs. Erythroid Data

Since the induced theory contained over 350 rules, here we present a subset of the theory, consisting of the top high-coverage rules.

```
[theory]

[Rule 4] [Pos cover = 27 Neg cover = 2]
positive(A) :-
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',11,10),
   distance_interval(A,ebox2_m00414_transf,'non3_ma0092.1_jaspar',57,30),
   distance_interval(A,'non3_ma0092.1_jaspar','runx1_MA0002.2_jaspar',109,30).

[Rule 80] [Pos cover = 20 Neg cover = 1]
positive(A) :-
   distance_interval(A,ebox2_m00414_transf,ebox2_m00414_transf,4,5),
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',14,30).

[Rule 6] [Pos cover = 19 Neg cover = 2]
positive(A) :-
   distance_interval(A,'runx1_MA0002.2_jaspar',ebox2_m00414_transf,81,5),
   distance_interval(A,'runx1_MA0002.2_jaspar','non3_ma0092.1_jaspar',154,15).

[Rule 1] [Pos cover = 18 Neg cover = 1]
positive(A) :-
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',173,10),
   distance_interval(A,'non3_ma0092.1_jaspar','ets1_ma0156.1_jaspar',408,100).

[Rule 146] [Pos cover = 18 Neg cover = 1]
positive(A) :-
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',98,5),
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',161,15),
   pos_lteq(A,'runx1_MA0002.2_jaspar',-96).

[Rule 23] [Pos cover = 17 Neg cover = 1]
```

```
positive(A) :-
   has_feature(A,'ebox1_ma0048.1_jaspar','D'),
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',57,5),
   distance_interval(A,'runx1_MA0002.2_jaspar','non3_ma0092.1_jaspar',38,10).

[Rule 162] [Pos cover = 16 Neg cover = 0]
positive(A) :-
   before(A,'runx1_MA0002.2_jaspar','ets1_ma0144.1_jaspar'),
   has_feature(A,'runx1_MA0002.2_jaspar','R'),
   distance_interval(A,'ebox1_ma0048.1_jaspar',ebox2_m00414_transf,55,30),
   distance_interval(A,'ets1_ma0144.1_jaspar','runx1_MA0002.2_jaspar',121,15),
   pos_lteq(A,'ebox1_ma0048.1_jaspar',-24).

[Rule 98] [Pos cover = 16 Neg cover = 1]
positive(A) :-
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',53,5),
   pos_lteq(A,gata4_m00127_transf,-77), pos_lteq(A,'runx1_MA0002.2_jaspar',37).

[Rule 22] [Pos cover = 16 Neg cover = 1]
positive(A) :-
   before(A,ebox2_m00414_transf,ebox2_m00414_transf),
   distance_interval(A,'ebox1_ma0048.1_jaspar','ebox1_ma0048.1_jaspar',213,10),
   distance_interval(A,'ets1_ma0156.1_jaspar','ebox1_ma0048.1_jaspar',62,100).

[Rule 11] [Pos cover = 16 Neg cover = 0]
positive(A) :-
   distance_interval(A,ebox2_m00414_transf,ebox2_m00414_transf,16,5),
   distance_interval(A,'runx1_MA0002.2_jaspar','ebox1_ma0048.1_jaspar',43,30),
   pos_lteq(A,'ebox1_ma0048.1_jaspar',-61).

[Rule 167] [Pos cover = 15 Neg cover = 0]
positive(A) :-
   distance_interval(A,'ebox1_ma0048.1_jaspar','ets1_ma0144.1_jaspar',96,15),
   distance_interval(A,ebox2_m00414_transf,'runx1_MA0002.2_jaspar',13,15).
```

```
[Rule 75] [Pos cover = 15 Neg cover = 1]
positive(A) :-
   distance_interval(A,'ebox1_ma0048.1_jaspar','ebox1_ma0048.1_jaspar',119,15),
   distance_interval(A,'ebox1_ma0048.1_jaspar','runx1_MA0002.2_jaspar',9,15),
   pos_lteq(A,ebox2_m00414_transf,-38).

[Rule 38] [Pos cover = 15 Neg cover = 1]
positive(A) :-
   distance_interval(A,'ebox1_ma0048.1_jaspar',ebox2_m00414_transf,28,5),
   distance_interval(A,'runx1_MA0002.2_jaspar',ebox2_m00414_transf,46,15),
   distance_interval(A,'ets1_ma0156.1_jaspar','ebox1_ma0048.1_jaspar',25,100).

[Rule 28] [Pos cover = 15 Neg cover = 0]
positive(A) :-
   distance_interval(A,'runx1_MA0002.2_jaspar','gcbox_MA0073.1_jaspar',65,10),
   distance_interval(A,'runx1_MA0002.2_jaspar','gcbox_MA0073.1_jaspar',25,30),
   pos_lteq(A,ebox2_m00414_transf,-57).

[Rule 190] [Pos cover = 14 Neg cover = 0]
positive(A) :-
   before(A,'runx1_MA0002.2_jaspar','gcbox_MA0073.1_jaspar'),
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',65,10),
   distance_interval(A,'runx1_MA0002.2_jaspar','non3_ma0092.1_jaspar',38,15),
   pos_lteq(A,'non3_ma0092.1_jaspar',-1).

[Rule 36] [Pos cover = 14 Neg cover = 0]
positive(A) :-
   distance_interval(A,'ets1_ma0144.1_jaspar','gcbox_MA0073.1_jaspar',65,10),
   distance_interval(A,'non3_ma0092.1_jaspar','runx1_MA0002.2_jaspar',51,10).

[Rule 20] [Pos cover = 14 Neg cover = 0]
positive(A) :-
   distance_interval(A,ebox2_m00414_transf,ebox2_m00414_transf,283,30),
   distance_interval(A,'runx1_MA0002.2_jaspar','ebox1_ma0048.1_jaspar',279,30).
```

B.3 Theory Induced on Jurkat vs. Synthetic Data

```
[theory]

[Rule 1] [Pos cover = 162 Neg cover = 18]
positive(A) :-
   before(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar'),
   distance_interval(A,'non3_ma0092.1_jaspar','ebox1_ma0048.1_jaspar',
      56,30).

[Rule 2] [Pos cover = 233 Neg cover = 21]
positive(A) :-
   has_feature(A,'gata3_ma0035.2_jaspar','R'),
   has_feature(A,'runx1_MA0002.2_jaspar','R'),
   distance_interval(A,'runx1_MA0002.2_jaspar','gata3_ma0035.2_jaspar',
      42,15).

[Rule 4] [Pos cover = 142 Neg cover = 10]
positive(A) :-
   has_feature(A,'non3_ma0092.1_jaspar','R'),
   distance_interval(A,'runx1_MA0002.2_jaspar','non3_ma0092.1_jaspar',
      109,100),
   distance_interval(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar',
      11,15).

[Rule 8] [Pos cover = 136 Neg cover = 7]
positive(A) :-
   before(A,'runx1_MA0002.2_jaspar','runx1_MA0002.2_jaspar'),
   before(A,'ets1_ma0156.1_jaspar','ets1_ma0156.1_jaspar'),
   has_feature(A,ebox2_m00414_transf,'R').

[Rule 12] [Pos cover = 73 Neg cover = 5]
positive(A) :-
   has_feature(A,'gata3_ma0035.2_jaspar','R'),
   distance_interval(A,'non3_ma0092.1_jaspar','gata3_ma0035.2_jaspar',
```

```
      69,30),
   pos_lteq(A,'non3_ma0092.1_jaspar',78),
   pos_gteq(A,'ets1_ma0144.1_jaspar',6).

[Rule 13] [Pos cover = 89 Neg cover = 5]
positive(A) :-
   has_feature(A,'gata3_ma0035.2_jaspar','R'),
   distance_interval(A,gata3_m00346_transf,'non3_ma0092.1_jaspar',40,100),
   distance_interval(A,ebox2_m00414_transf,'ets1_ma0156.1_jaspar',25,100).

[Rule 16] [Pos cover = 155 Neg cover = 0]
positive(A) :-
   pos_lteq(A,'runx1_MA0002.2_jaspar',-190).

[Rule 27] [Pos cover = 115 Neg cover = 5]
positive(A) :-
   has_feature(A,'runx1_MA0002.2_jaspar','R'),
   distance_interval(A,'runx1_MA0002.2_jaspar',ebox2_m00414_transf,
      136,100),
   distance_interval(A,'ets1_ma0144.1_jaspar','ets1_ma0156.1_jaspar',
      49,100).
```

B.4 Theory Induced on Erythroid vs. Synthetic Data

```
[theory]

[Rule 1] [Pos cover = 623 Neg cover = 56]
positive(A) :-
   has_feature(A, 'ets1_ma0156.1_jaspar', 'R'),
   distance_interval(A, 'ets1_ma0156.1_jaspar',
      'gata3_ma0035.2_jaspar', 236, 100).

[Rule 2] [Pos cover = 887 Neg cover = 77]
positive(A) :-
   has_feature(A, gata3_m00348_transf, 'R'),
   has_feature(A, 'gata3_ma0035.2_jaspar', 'R'),
   has_feature(A, 'non3_ma0092.1_jaspar', 'R'),
   pos_lteq(A, 'gata3_ma0035.2_jaspar', -8).

[Rule 3] [Pos cover = 616 Neg cover = 0]
positive(A) :-
   pos_lteq(A, 'runx1_MA0002.2_jaspar', -126).

[Rule 4] [Pos cover = 286 Neg cover = 11]
positive(A) :-
   distance_interval(A, ebox2_m00414_transf, 'ets1_ma0156.1_jaspar',
      267, 100),
   distance_interval(A, 'ets1_ma0156.1_jaspar', ebox2_m00414_transf,
      55, 100).

[Rule 8] [Pos cover = 566 Neg cover = 0]
positive(A) :-
   pos_lteq(A, 'ets1_ma0156.1_jaspar', -139).

[Rule 9] [Pos cover = 392 Neg cover = 0]
positive(A) :-
   pos_lteq(A, ebox2_m00414_transf, -133).
```

```
[Rule 11] [Pos cover = 477 Neg cover = 11]
positive(A) :-
   distance_interval(A, 'runx1_MA0002.2_jaspar',  'non3_ma0092.1_jaspar',
      274, 100),
   pos_lteq(A, 'non3_ma0092.1_jaspar', 74).

[Rule 24] [Pos cover = 587 Neg cover = 0]
positive(A) :-
   pos_lteq(A, 'non3_ma0092.1_jaspar', -134).

[Rule 30] [Pos cover = 420 Neg cover = 7]
positive(A) :-
   pos_lteq(A, gata3_m00346_transf, -120).

[Rule 63] [Pos cover = 258 Neg cover = 13]
positive(A) :-
   has_feature(A, gata1_m00011_transf, 'R'),
   has_feature(A, gata3_m00346_transf, 'R'),
   has_feature(A, 'gata3_ma0035.2_jaspar', 'R'),
   distance_interval(A, gata3_m00348_transf, gata1_m00011_transf,
      11, 100).
```

B.5 User Interface Screen Captures

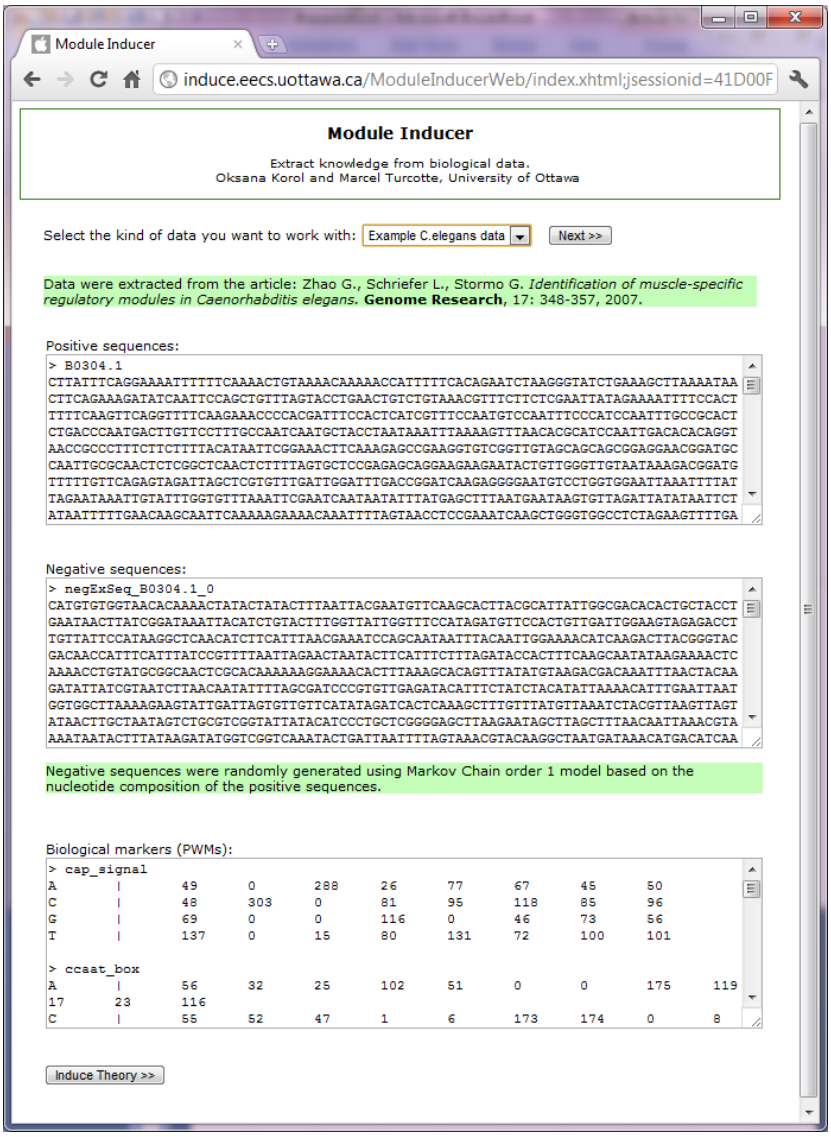

Figure B.1: User Interface Screen 1: Initial Data Entry

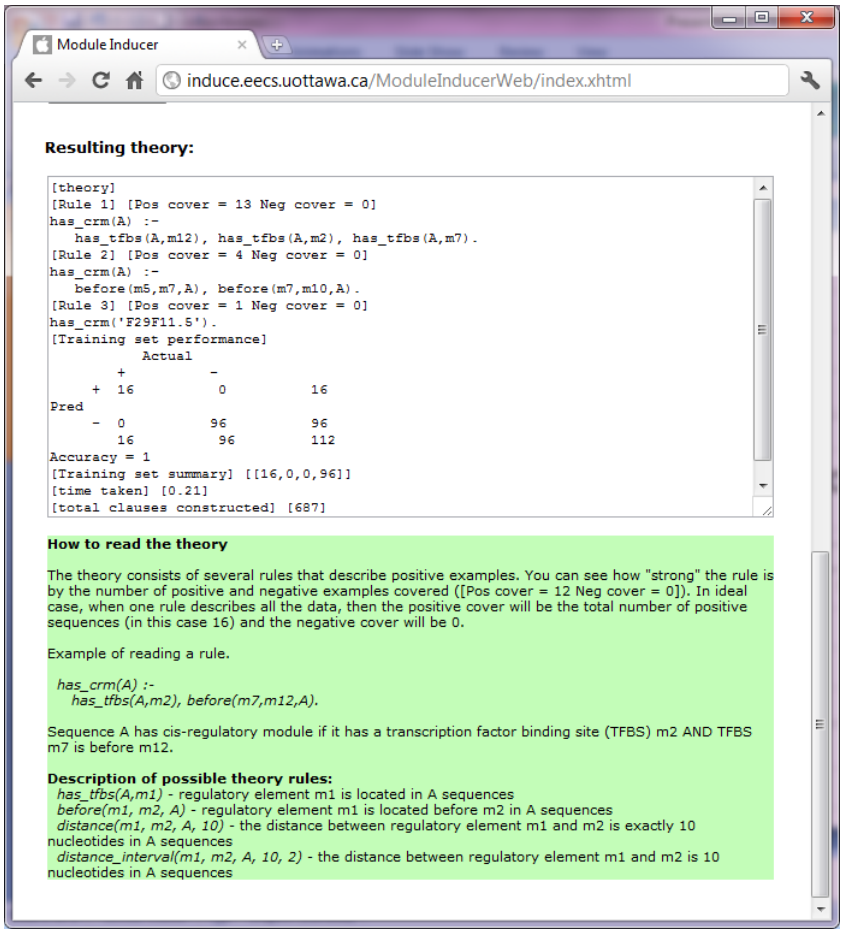

Figure B.2: User Interface Screen 2: Displaying Results

Bibliography

[1] Stein Aerts, Peter Van Loo, Gert Thijs, Yves Moreau, and Bart De Moor. Computational detection of *cis*-regulatory modules. *Bioinformatics*, 19:ii5ii14, 2003. ModuleSearcher tool.

[2] B. Alberts, A. Johnson, J. Lewis, and et al. *Molecular Biology of the Cell*. Garland Science, 2002.

[3] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[4] Timothy L. Bailey. DREME: Motif discovery in transcription factor ChIP-Seq data. *Bioinformatics*, 2011.

[5] Robert G. Beiko and Robert L. Charlebois. GANN: Genetic algorithm neural networks for the detection of conserved combinations of features in dna. *BMC Bioinformatics*, 6, 2005.

[6] Juliana S. Bernardes, Alessandra Carbone, and Gerson Zaverucha. A discriminative method for family-based protein remote homology detection that combines inductive logic programming and propositional models. *BMC Bioinformatics*, 12, 2011.

[7] Vitoantonio Bevilacqua, Patrizia Chiarappa, Giuseppe Mastronardi, Filippo Menolascina, Angelo Paradiso, and Stefania Tommasi. Identification of tumor evolution patterns by means of inductive logic programming. *Genomics, Proteomics and Bioinformatics*, 6, 2008.

[8] Ed Burns and Chris Schalk. *JavaServer Faces 2.0: The Complete Reference*. McGraw-Hill, 2010.

[9] Francis S. Collins and Victor A. McKusic. Implications of the Human Genome Project for medical science. *Journal of American Medical Association*, 285:2001.

[10] C.B. Congdon, C.W. Fizer, N.W. Smith, H.R. Gaskins, J. Aman, G.M. Nava, and C. Mattingly. Preliminary results for GAMI: A genetic algorithms approach to motif inference. *Computational Intelligence in Bioinformatics and Computational Biology*, 2005.

[11] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[12] Chris Drummond and Robert C. Holte. Explicitly representing expected cost: an alternative to roc representation. *Proceeding of Knowledge Discovery and Datamining*, 2000.

[13] Eric S. Lander et al. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[14] Tom Fawcett. ROC graphs: Notes and practical considerations for researchers. *Machine Learning*, 2004.

[15] Martin C. Frith, Michael C. Li, and Zhiping Weng. Cluster-Buster: finding dense clusters of motifs in DNA sequences. *Nucleic Acids Research*, 31:3666–3668, 2003.

[16] Valer Gotea and Ivan Ovcharenko. DiRE: identifying distant regulatory elements of co-expressed genes. *Nucleic Acids Research*, 36:W133W139, 2008. DiRE.

[17] Mayetri Gupta and Jun S. Liu. De novo *cis*-regulatory module elicitation for eukaryotic genomes. *PNAS (Proceedings of the National Academy of Sciences of the United States of America)*, 102:7079–7084, 2005.

[18] Shobhit Gupta, John A. Stamatoyannopoulos, Timothy L. Bailey, and William Stafford Noble. Quantifying similarity between motifs. *Genome Biology*, 8, 2007.

[19] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11, 2009.

[20] Outi Hallikas, Kimmo Palin, Natalia Sinjushina, Reetta Rautiainen, Juha Partanen, Esko Ukkonen, and Jussi Taipale. Genome-wide prediction of mammalian enhancers based on analysis of transcription-factor binding affinity. *Cell*, 124:47–59, 2006. EEL.

[21] Gerald Z. Hertz and Gary D. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15:563–577, 1999.

[22] J. Martinez Hoyos, A. Ferraro, S. Sacchetti, S. Keller, I. De Martino, E. Borbone, P. Pallante, M. Fedele, D. Montanaro, F. Esposito, P. Cserjesi, L. Chiariotti, G. Troncone, and A. Fusco. HAND1 gene expression is negatively regulated by the high mobility group A1 proteins and is drastically reduced in human thyroid carcinomas. *Oncogene*, 28:876885, 2009.

[23] Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms*. Cambridge University Press, 2011.

[24] Mikhail Jiline, Stan Matwin, and Marcel Turcotte. Annotation concept synthesis and enrichment analysis: a logic-based approach to the interpretation of high-throughput experiments. *Bioinformatics*, 27:2391–2398, 2011.

[25] Bilian Jin, Bing Yao, Jian-Liang Li, C. Robert Fields, Amber L. Delmas, Chen Liu, and Keith D. Robertson. DNMT1 and DNMT3B modulate distinct polycomb-mediated histone modifications in colon cancer. *Cancer Research*, 69, 2009.

[26] Jörg-Uwe Kietz and Marcus Lübbe. An efficient subsumption algorithm for inductive logic programming. *Machine learning: proceedings of the eleventh International Conference*, 1994.

[27] Ross D. King, Stephen H. Muggleton, Richard A. Lewis, and Michael J. E. Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *PNAS (Proceedings of the National Academy of Sciences of the United States of America)*, 89:11322–11326,, 1992.

[28] Ross D. King, Stephen H. Muggleton, Ashwin Srinivasan, and Michael J. E. Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *PNAS (Proceedings of the National Academy of Sciences of the United States of America)*, 93:438–442, 1996.

[29] Stefan Kramer and Eibe Frank. Bottom-up propositionalization. *Proceedings of the ILP*, page 156162, 2000.

[30] Nada Lavrač, Peter Flach, and Blaz Zupan. Rule evaluation measures: A unifying view. *Springer Lecture Notes in Computer Science*, 1634:174–185, 1999.

[31] Michael Levine and Robert Tjian. Transcription regulation and animal diversity. *Nature*, 424:147–151, 2003.

[32] Angelica Lindlf, Marcus Brutigam, Aakash Chawade, Olof Olsson, and Bjrn Olsson. In silico analysis of promoter regions from cold-induced genes in rice (Oryza sativa L.) and Arabidopsis thaliana reveals the importance of combinatorial control. *Bioinformatics*, 25:13451348, 2009.

[33] Huma Lodhi, Stephen Muggleton, and Mike J. E. Sternberg. Multi-class mode of action classification of toxic compounds using logic based kernel methods. *Molecular Informatics*, 29:655 664, 2010.

[34] Alain-Pierre Manine, Erick Alphonse, and Philippe Bessires. Learning ontological rules to extract multiple relations of genic interactions from text. *International Journal of Medical Informatics*, 78:e31e38, 2009.

[35] John S. Mattick. Non-coding RNAs: the architects of eukaryotic complexity. *EMBO Reports*, 2:986991, 2001.

[36] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. Ilp turns 20 - biography and future challenges. *Machine Learning*, pages 1–21, 2011.

[37] Stephen Muggleton, Huma Lodhi, Ata Amini, and Michael J. E. Sternberg. Support vector inductive logic programming. *Proceedings of the Eighth International Conference on Discovery Science*, 3735, 2005.

[38] Stephen H. Muggleton. Inductive logic programming. *New Generation Computing*, 8:295318, 1991.

[39] Carmen G. Palii, Carolina Perez-Iratxetaand Zizhen Yao, Yi Cao, Fengtao Dai, Jerry Davidson, Harold Atkins, David Allan, F. Jeffrey Dilworth, Robert Gentleman, Stephen J Tapscott, and Marjorie Brand. Differential genomic targeting of the transcription factor TAL1 in alternate haematopoietic lineages. *The EMBO Journal*, page 116, 2010.

[40] Adrian Paschke and Michael Schrder. Inductive logic programming for bioinformatics in Prova. *VLDB*, 2007.

[41] Giulio Pavesi, Giancarlo Mauri, and Graziano Pesole. An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics*, 17:s207–s214, 2001.

[42] Anthony A. Philippakis, Fangxue Sherry He, and Martha L. Bulyk. ModuleFinder: a tool for computational discovery of cis regulatory modules. *Pacific Symposium on Biocomputing*, pages 519–530, 2005.

[43] Gordon D. Plotkin. A note on inductive generalisation. *Machine Intelligence*, 5:153–163, 1970.

[44] The Sequence Ontology Project. Generic feature format version 3. `http://www.sequenceontology.org/gff3.shtml`, 2010.

[45] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. *Proceeding of the 15th International Conference on Machine Learning*, pages 445–453, 1998.

[46] Attila Reményi, Hans R. Schöler, and Matthias Wilmanns. Combinatorial control of gene expression. *Nature Structural and Molecular Biology*, 11:812815, 2004.

[47] Eran Segal and Roded Sharan. A discriminative model for identifying spatial *cis*-regulatory modules. *Journal of Computational Biology*, 12:822–834, 2005.

[48] Roded Sharan, Ivan Ovcharenko, Asa Ben-Hur, and Richard M. Karp. CREME: a framework for identifying *cis*-regulatory modules in human-mouse conserved segments. *Bioinformatics*, 19:i283i291, 2003.

[49] Ashwin Srinivasan. The Aleph manual. version 4 and above. `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/aleph.html`, 2007.

[50] Ashwin Srinivasan, Tanveer A. Faruquie, and Sachindra Joshi. Data and task parallelism in ilp using mapreduce. *Machine Learning*, pages 1–28, 2011.

[51] Michael J. E. Sternberg and Stephen H. Muggleton. Structure activity relationships (SAR) and pharmacophore discovery using inductive logic programming (ILP). *QSAR and Combinatorial Science*, 22, 2003.

[52] Ljupko Todorovski, Peter Flach, and Nada Lavrač. Predictive performance of weighted relative accuracy. *Lecture Notes in Computer Science*, 1910:182–211, 2000.

[53] Martin Tompa. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. *AAAI*, 1999.

[54] Marcel Turcotte, Stephen H. Muggleton, and Michael J. E. Sternberg. Automated discovery of structural signatures of protein fold and function. *Journal of Molecular Biology*, 306:591–605, 2001.

[55] Marcel Turcotte, Stephen H. Muggleton, and Michael J. E. Sternberg. The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Machine Learning*, 43:8195, 2001.

[56] J. van Helden, B. Andre, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology*, 281:827–842, 1998.

[57] Peter Van Loo, Stein Aerts, Bernard Thienpont, Bart De Moor, Yves Moreau, and Peter Marynen. ModuleMiner - improved computational detection of *cis*-regulatory modules: are there different modes of gene regulation in embryonic development and adult tissues? *Genome Biology*, 9, 2008. ModuleMiner.

[58] Peter Van Loo and Peter Marynen. Computational methods for the detection of *cis*-regulatory modules. *Briefings in Bioinformatics*, 10(5):509–524, 2009.

[59] Joshua W. Vincentz, Ralston M. Barnes, and Anthony B. Firulli. Hand factors as regulators of cardiac morphogenesis and implications for congenital heart defects. *Birth Defects Research*, 91:485–494, 2011.

[60] Wyeth W. Wasserman and Albin Sandelin. Applied bioinformatics for the identification of regulatory elements. *Nature Reviews Genetics*, 5:276–287, 2004.

[61] Zhi Wei and Shane T. Jensen. GAME: detecting *cis*-regulatory elements using a genetic algorithm. *Bioinformatics*, 22:15771584, 2006.

[62] Koichi Yagi, Kiwamu Akagi, Hiroshi Hayashi, Genta Nagae, Shingo Tsuji, Takayuki Isagawa, Yutaka Midorikawa, Yoji Nishimura, Hirohiko Sakamoto, Yasuyuki Seto, Hiroyuki Aburatani, and Atsushi Kaneda. Three DNA methylation epigenotypes in human colorectal cancer. *Clinical Cancer Research*, 16:21–33, 2010.

[63] Guoyan Zhao, Lawrence A. Schriefer, and Gary D. Stormo. Identification of muscle-specific regulatory modules in caenorhabditis elegans. *Genome Research*, 17:348–357, 2007.