U N I V E R S I T Y O F T A R T U

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Institute of Computer Science

Chair of Software Systems

Konstantin Tretyakov

A Linear Model of Genetic Transcription Regulation that Combines Microarray and Genome Sequence Data

Bachelor's thesis (10 cp)

Supervisor: Jaak Vilo, Ph.D.

Autor:	"....."	mai	2005
Juhendaja:	"....."	mai	2005
Õppetooli juhataja:	"....."	....	2005

TARTU 2005

Contents

Introduction

The cell of a living organism is perhaps one of the most studied and at the same time the least understood objects on our planet. Processes that take place in this tiny piece of matter are strikingly complex, yet concurring in harmony together they influence the lifecycle of the whole organism. Understanding these processes is therefore one of the ultimate goals of contemporary medical and biological studies, that may lead to better treatment of diseases and new discoveries.

Current biological knowledge suggests that most of the regulatory processes in the cell are related to certain molecules, mainly proteins. Despite the diversity of different proteins (hormones, enzymes, transport proteins, etc), on the molecular level their structure is generally the same: each protein can be costructed as a linear chain of amino-acids. These linear chains are encoded by the DNA molecules, stored in the chromosomes of the cell. Studying the DNA and the relationships among the proteins may therefore provide the keys to the inner workings of the cell.

Microbiological research during the last years has produced a considerable amount of data concerning DNA sequences and mutual influences among proteins. Of particular interest are the results of the *microarray hybridization experiments*. This *microarray data*, however, consists of indirect measurements and therefore sophisticated methods are required to infer biologically meaningful conclusions from it. In fact, quite a small number of techiques are currently capable of extracting plausible biological hypotheses from microarray data [1]. Even less methods manage to combine this data with additional external information, such as DNA sequences. That's why the search for new computational methods for complex analysis of microarray data is currently a rather hot topic in the bioinformatics community (some examples are [2, 3, 4, 5, 6, 7, 8]).

This thesis proposes a novel analysis technique based on a simple linear model that combines microarray measurements with DNA sequence data. The approach can be regarded as both a predictive model for the expression values of the genes, as well as a descriptive model, that can potentially provide some insight into genetic transcriptional regulation. The model can be applied in the context of both linear regression and linear classification, and in both cases it should be capable of extracting useful information.

The main text of the thesis consists of four chapters. The first chapter gives a brief biological background required for the understanding of the thesis, the second provides basic introduction to the topics of machine learning and linear models. The reader familiar with the subject may skip immediately to the third chapter, where a specific linear model for gene expression data is proposed and its applications to the analysis of microarray data are discussed. The last chapter illustrates the method using a toy example, and examines its performance on a large artificial dataset.

Chapter 1

Biological Background

People are DNA's way of making more DNA.

Edward O. Wilson

1.1 The DNA

The cell is the fundamental unit of any living organism. The smallest organisms such as bacteria might consist of a single cell, larger beings may contain millions of different cells organised in organs and tissues. Despite the small size, a cell is a very complex system on its own and is composed of yet smaller functional units called organelles. Most of the regulation of the work of the organelles, as well as inter-cellullar communication is performed by protein molecules. Proteins vary greatly in their functions, yet they all are constructed in the same way: as linear chains of amino-acids. The exact order of amino-acids is therefore enough to specify any protein. This order is encoded in the strands of the DNA molecules, which form the genetic material of the cell.

The DNA (desoxyribose nucleic acid) is a long spiral-shaped molecule consisting of two complementary strands of nucleotides. Each nucleotide consists of a phosphate group, sugar desoxyribose and a nitrogenous base: adenine, thymine, cytosine or guanine. Depending on the base, nucleotides are usually denoted by letters `A`, `T`, `C` and `G` respectively. A DNA strand can be therefore encoded as a string of these four letters. The second, complementary strand of the same molecule is uniquely defined by the first one. The order of the nucleotides in the DNA determines the order of the amino-acids in the proteins of the cell. It is known that each amino-acid is encoded by a certain triple of nucleotides, and there are certain nucleotide triples that usually denote the start and the end of the coding region (e.g. `ATG`, `TAG`).

The DNA is believed to store all the information relevant for the organism. DNA is replicated together with the cells, and, in case of heterosexual organisms, the DNA in the cells of the offspring is usually a mixed version of the DNAs of the parents. The DNA of every organism is different, but for a given species the differences between the DNA's are localized to a very small subset that determines some basic traits only. We may therefore say that every species has its unique DNA code. The full DNA has been sequenced for many different species.

DNA is usually divided into regions referred to as *genes*. The set of all genes in a cell is called the cell's *genome*. Each gene encodes one or more proteins and the set of all proteins synthesized from a genome is called *proteome*. Some regions of the genome are devoted to control mechanisms, and a substantial amount of genomes of higher-order organisms appears to serve no purpose at all (the so-called junk DNA) [1]. The coding regions of the DNA sequence are usually referred to as *exons* and the non-coding regions — as *introns* (when within genes) or *intergenic* regions (outside of genes).

1.2 Transcription and Translation, Gene Expression

The proteins encoded in the DNA are produced in two stages: *transcription* and *translation*. In the process of transcription, a large molecular complex called *RNA-polymerase* attaches itself to the segment of DNA containing a certain *promoter sequence*, and starts copying DNA from that point producing an equivalent RNA molecule. RNA (ribose nucleic acid) molecule is very similar to the DNA: it is also a chain of nucleotides, the only difference is that the sugar ribose is used, and base thymine is replaced by its cousin uracil. Contrary to the DNA, however, RNA molecules are much less stable and get hydrolised in the cell within minutes, this is an important feature of the RNA.

The transcription process stops when a certain terminator-sequence is found, and releases an RNA encoding exactly the same information as that on the corresponding region of the DNA. Not all of this information might encode a protein, there might be *introns* inside. These introns get removed with the help of even more complex molecular machinery (a process known as *splicing*), and in the end an RNA strand ready to be translated to proteins is produced. This RNA is often called *mRNA* (short for *matrix-RNA*).

Next another large molecular complex, the *ribosome*, attaches itself to a certain sequence on the mRNA, and starts reading the mRNA producing the corresponding protein. This process is known as *translation*.

Together, the production of proteins from genes is referred to as *gene expression*. Not all genes of the genome are expressed in a cell at once. The regulation of which gene will be expressed and which will not is a very complex process. An important role in expression regulation is played by certain proteins, called *transcription factors* or *TF*-s, that attach themselves to certain sequences (*motifs*) in the DNA and provoke (or prevent) RNA-polymerase to bind there, thus inducing (or suppressing) the transcription of the corresponding gene. Much more complex transcription regulation patterns take place: certain TF-s will attach to the DNA only in presence of some other TF-s, and certain TF-s, when attached, may block other TF-s from binding there.

Transcription regulation is not the only part of expression regulation. After being transcribed to mRNA, the gene will be translated to proteins only under certain circumstances, when all the machinery (i.e. proteins) needed for splicing and translation is there. Depending on the conditions, the mRNA of the gene may be translated to one or another protein. At last, the same protein molecule may sometimes (although rarely) obtain different *tertiary structure* (i.e. 3D shape) in different conditions, and therefore have different function.

In general, the production of any given protein by the cell depends on the amount of other proteins as well as the environmental conditions (heat, acidity, etc). The relationships among proteins, that govern gene expression, are often called *genetic networks*, and the problem of determining them is crucial to explaining cell's lifecycle.

1.3 Microarray Experiments

DNA hybridization microarrays are a relatively recent development that allows to study genome-wide patterns of gene expression. A typical experiment is the following: first an array is prepared, which is a glass or membrane support with a set of DNA fragments attached to it. The fragments are carefully arranged in *spots*: each spot contains the fragments of a certain gene only. Next, two cell cultures are selected: a "reference" culture, and a culture of cells under certain stress or environmental conditions ("test" culture). All the RNA from the cells is reverse-transcribed to create strands of fluorescent-labeled (or radioactive) complementary DNA. Labeling of the cDNA differs for the two cultures. The obtained cDNA is then purged onto the array where it hybridizes with the DNA in the spots of the array. As a result, all the cDNA-s should end up on the spots with matching DNA fragments. Due to the labeling of the cDNA, the colour (or radiation spectrum) of each spot will indicate the ratio of the cDNA amounts from the first and the second culture attached to it, hence the ratio of RNA amounts of the corresponding gene produced in the different cultures. The amount of RNA for a gene may be regarded as an indicator of the expression of that gene. Of course, as noted above, transcription is not the only aspect of gene expression, so this is a rather rough indicator indeed. Despite that, a microarray experiment measures the "expression levels" of many genes at the same time.

1.4 Microarray Data

Ultimately, a microarray experiment reports a single real value for each gene. Data from several arrays is usually represented as a matrix with rows corresponding to genes and columns to arrays. Each entry specifies the expression of a gene in a given array. Microarray experiments are rather difficult to set up, however every experiment measures the expression levels of thousands of genes at the same time. It is therefore typical for the matrix to have thousands of rows, but only about a hundred or so columns. It is this *expression matrix*, that is in the focus of this thesis.

1.5 Genome Analysis

Although a decent amount of microarray data is available together with some fully sequenced genomes, no universally good analysis methods for this data exist. The main problem is that we still don't understand all aspects of gene expression at the molecular level. Several mathematical formalisms have been applied to genetic networks: boolean networks, continuous recurrent networks, bayesian networks , but none of them appears to capture all the dimensions of gene regulation [9]. The available data is too scarce to allow inferring complex models from it, but the few examples of regulatory circuits for which detailed information is available, all appear to be complex.

To the moment, machine learning techniques seem to provide some of the best results in inferring information from microarray data. This thesis documents yet another attempt of that kind.

Chapter 2

Linear Models

If you ask somone a harmless question, and they scream at you, then it is a reasonable metaphor to say they "went non-linear" because your input created a disproportionate output.
Rob MacLachlan

This chapter provides basic introduction to machine learning in general and linear regression and classification in particular. We present a brief tour of the methods for fitting common linear models. Only the most basic ideas are provided, avoiding detailed explanations as this would require hundreds of pages, far more than is in the scope of this work. For a more thorough introduction on the topic the reader is referred to textbooks on the subject, such as [10, 11, 12, 13, 14].

2.1 Machine Learning

The task of inferring functional relationships between the variables in the data is often referred to as "machine learning". A common formulation for the task is the following:

> Given a *training set* of labeled data examples $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, find the function $f : \mathcal{X} \to \mathcal{Y}$ that describes the relation between $\mathcal{X}$ and $\mathcal{Y}$ "best".

The function f may then be used to predict values of y for previously unseen values of x (*regression*, *classification*). Some machine learning algorithms construct a human-comprehensible model of f (e.g. a decision tree), that may be used to "explain" the relationship and find its underlying factors. Finally, the machine learning task can also be formulated as a *density estimation* problem:

> Given a sample S generated i.i.d from some unknown probability distribution F on $\mathcal{X} \times \mathcal{Y}$, estimate this distribution.

It can be shown that it is impossible to construct f using only the data and making no assumptions about the underlying distribution [11]. The assumptions made by the algorithm are called its *inductive bias*. For most algorithms the inductive bias can be expressed in terms of a parameterized model: given that model, the algorithm searches for such parameter values, with which the model fits the data best.

Perhaps the most simple, but at the same time the most stable (and often the most robust) is the linear model. In the following, we describe some algorithms for fitting linear models, that will be used in the analysis further on.

2.2 Linear Regression

Suppose $\mathcal{X} = \mathbb{R}^m$ and $\mathcal{Y} = \mathbb{R}$. The idea of linear regression is, given the training set $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$[1], find a *linear* function f that fits the data best. By a linear function we mean here a function of the form[2]

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{j=1}^{m} w_j x_j \quad .$$

The problem consists in finding the best parameter vector $\mathbf{w}$.

Which function is the best? Intuitively, the best function is the one which has the least error on the training data, for example, the one that minimizes the sum of error squares:

$$\mathcal{E}(f) = \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2$$

Minimizing this sum is indeed a sensible choice and it can be demonstrated by a more formalised argument. Suppose that $\mathbf{x}$ are sampled from a uniform distribution on some closed subset of $\mathbb{R}^m$, and for each $\mathbf{x}$ the corresponding y is obtained as $y = \boldsymbol{\omega}^T\mathbf{x} + \varepsilon$, where ε denotes zero-mean gaussian noise that is independent of $\mathbf{x}$. In this case the parameter $\mathbf{w}$ that minimizes sum of error squares

$$\mathcal{E}(\mathbf{w}) = \sum_{i=1}^{n} (\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

gives an unbiased estimate of true $\boldsymbol{\omega}$.

[1]In the following we denote elements of $\mathbb{R}^m$ by bold letters ($\mathbf{x}$, $\mathbf{w}$, etc) and treat them as if they were $n \times 1$ matrices. $\mathbf{x}^T$ denotes matrix transpose and $\mathbf{x}^T\mathbf{y}$—matrix multiplication of $\mathbf{x}^T$ and $\mathbf{y}$, i.e. the inner product of $\mathbf{x}$ and $\mathbf{y}$. Elements of $\mathbf{x}$ are denoted as x_1, x_2, etc.

[2]Note that a linear function is often defined as a function of the form $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$. However, the intercept term b can always be simulated by adding one more constant coordinate to all the $\mathbf{x}$-s so we prefer to ignore it here. This is standard practice in multivariate statistics, see [10] for example.

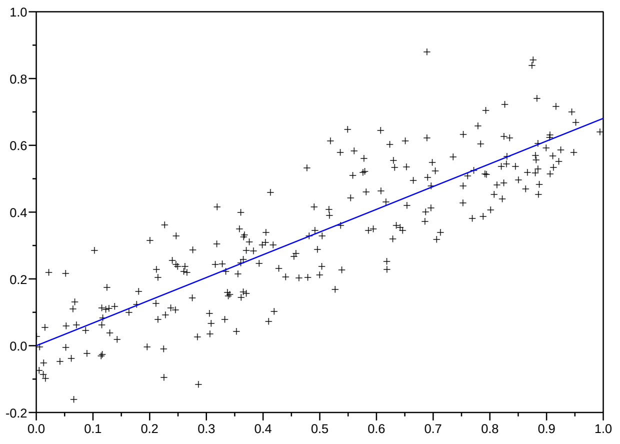

Figure 2.1: Linear regression. The points denote training data (one-dimensional variable x and the corresponding y). The line is a linear function $f(x) = wx$ that minimizes the sum of error squares for the training points.

Denote by $\mathbf{X}$ the matrix that contains vectors $\mathbf{x}_i^T$ as its rows (the *design matrix*, as it is commonly called) and by $\mathbf{y}$ the column vector $(y_1, y_2, \ldots, y_n)^T$. Then, for a given $\mathbf{w}$ the sum of error squares can be expressed as $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$. By minimizing this expression with respect to $\mathbf{w}$, we obtain the following solution to the linear regression problem:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

which is always defined when $\operatorname{rank}(\mathbf{X}) = m$.

The coefficients of the obtained linear regression model have a simple interpretation: a large value for w_j for some j indicates that the variable x_j has a certain significant "influence" on the result y. If $w_j > 0$, this influence is positive (larger values of x_j indicate larger values of y) and if $w_j < 0$ then the variable x_j has a negative effect on y.

2.3 Linear Classification

Classification can be considered as a particular case of regression, where $\mathcal{Y}$ is a finite set. Here we shall deal with *binary* classification only, that is, $|\mathcal{Y}| = 2$. It is particularly convenient to choose $\mathcal{Y} = \{-1, +1\}$. *Linear (binary) classification* then refers to finding a function $f : \mathbb{R}^m \to \{-1, +1\}$ of the form[3]

$$f(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^T\mathbf{x} + b) = \operatorname{sign}\left(\sum_{j=1}^{m} w_j x_j + b\right)$$

that has the best precision on the training data.

Having trained a linear classifier, we can use it to interpret the importance of variables in exactly the same way as in the case of a trained linear regression model: variables with larger coefficients should be considered more important than variables with coefficients closer to zero.

It is convenient to think of a linear classifier as a hyperplane in $\mathbb{R}^m$, that separates the points of the two classes well. Indeed, the set of points $\mathbf{x}$, satisfying $\mathbf{w}^T\mathbf{x}+b = 0$ forms a hyperplane in $\mathbb{R}^m$. Points, for which $\mathbf{w}^T\mathbf{x}+b > 0$ are located to one side of that hyperplane, and those, for which $\mathbf{w}^T\mathbf{x} + b < 0$ —to the other side. By minimizing the error rate of the classifier we find the hyperplane that separates the points of the two classes best. In the following we shall therefore sometimes use the term *separating hyperplane* instead of *linear classifier*.

There are several algorithms for learning linear classifiers: Rosenblatt's perceptron [15], Fisher's determinant, logistic regression and support vector machines [16] are the most famous of them. This thesis deals primarily with support vector machines because they have a firm advantage over the others from the standpoint of performance. This type of classifiers has guaranteed bounds on generalization error and is supported by Vapnik's *statistical learning theory* [17]. They are easily representable as a *kernel method* and this allows to apply them more conveniently on the particular type of data we use. The only disadvantage is the nearly quadratic computational complexity of SVM training. In the following we shall discuss this method in a bit more detail.

2.4 Support Vector Machines

2.4.1 Maximal Margin Classifier

Let $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in \{-1, +1\}$ denote, as before, the training set. Suppose for the moment, that the data is *linearly separable*. That means, there exists some linear classifier that can classify all the training points perfectly:

$$\exists(\mathbf{w}, b)\,\forall i,\ 1 \le i \le n \quad \operatorname{sign}(\mathbf{w}^T\mathbf{x}_i + b) = y_i$$

[3]Note that unlike the case of linear regression, it is more convenient to explicitly include the intercept term b here.

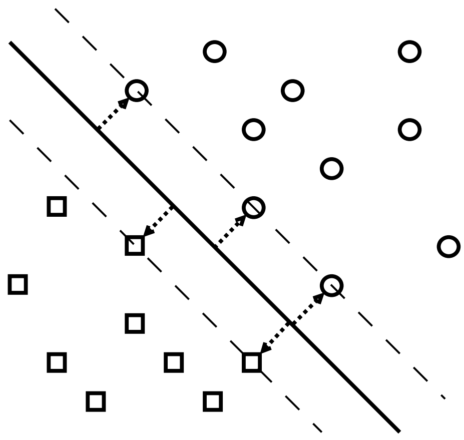

Figure 2.2: Maximal margin separating hyperplane. The figure displays the points of two classes and the maximal margin classifier separating them. Points that have the least margin m_i (the *support vectors*) are marked.

The idea of the support vector machine method is to find a linear classifier that not only separates the classes perfectly, but also has the largest *margin*. By margin we denote the distance to the closest training point. More formally, for a fixed hyperplane $(\mathbf{w}, b)$, denote the margin m_i of a training sample $(\mathbf{x}_i, y_i)$ as the distance of the point $\mathbf{x}_i$ to the hyperplane:

$$m_i = \operatorname{dist}(\mathbf{x}, (\mathbf{w}, b)) = \frac{|\mathbf{w}^T \mathbf{x}_i + b|}{\|\mathbf{w}\|}$$

where the latter fraction is a well-known formula for the distance of a point to a plane.

The *margin* m of the separating hyperplane with the respect to the whole training set S is the smallest margin of an instance of the training set:

$$m = \min_i m_i$$

Finally, the *maximal margin separating hyperplane* for a training set S is the separating hyperplane having the maximal margin with respect to this training set. In case of linearly separable data, training of SVM consists of finding this maximal margin separating hyperplane.

2.4.2 Maximal Margin Classifier Training

In order to find the maximal margin separating hyperplane for a given training sample we have to find the parameters $(\mathbf{w}, b)$ that maximize the margin m. It is easy to see that the hyperplane given by parameters $(\mathbf{w}, b)$ is the same as the hyperplane $(k\mathbf{w}, kb)$, therefore we may constrain our search by only considering *canonical* hyperplanes: those, for which

$$\min_i |\mathbf{w}^T\mathbf{x}_i + b| = 1$$

It is also possible to show, that the canonical hyperplane with the maximal margin has minimal $\|\mathbf{w}\|$. Finding the maximal margin classifier therefore reduces to the following constrained minimization problem:

Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ under the conditions

$$g_i(\mathbf{w}, b) \leq 0, \quad i = 1, 2, \ldots, n \tag{2.1}$$

where $g(\mathbf{w}, b) := 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$.

It turns out, that the solution of this minimization problem can be found in the *saddle point* of the *Lagrangian function*

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \boldsymbol{\alpha}_i g_i(\mathbf{w}, b)$$

The saddle point of interest is a point $(\mathbf{w}^*, b^*, \boldsymbol{\alpha}^*)$ where L is maximized with respect to $\boldsymbol{\alpha} \geq 0$ and minimized with respect to $(\mathbf{w}, b)$. In order to see that, consider the function

$$f(\mathbf{w}, b) = \max_{\boldsymbol{\alpha}\geq\mathbf{0}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = \begin{cases} \frac{1}{2}\mathbf{w}^T\mathbf{w} & \text{if } \mathbf{w} \text{ satisfies the constraints} \\ \infty & \text{otherwise} \end{cases}$$

The sought parameters $(\mathbf{w}, b)$ minimize $f(\mathbf{w}, b)$, and we can therefore express them as a saddle point:

$$(\mathbf{w}, b) = \operatorname*{argmin}_{\mathbf{w},b} \max_{\boldsymbol{\alpha}\geq\mathbf{0}} L(\mathbf{w}, b, \boldsymbol{\alpha})$$

It follows from the *Kuhn-Tucker* theorem [18], that the saddle point $(\mathbf{w}^*, b^*, \boldsymbol{\alpha}^*)$ is precisely the point, for which the following relations hold true:

$$\frac{\partial L(\mathbf{w}^*, b^*, \boldsymbol{\alpha}^*)}{\partial \mathbf{w}} = \mathbf{0} \tag{2.2}$$

$$\frac{\partial L(\mathbf{w}^*, b^*, \boldsymbol{\alpha}^*)}{\partial b} = 0 \tag{2.3}$$

$$\alpha_i^* g_i(\mathbf{w}^*, b^*) = 0, \qquad i = 1, \ldots, m \tag{2.4}$$

$$g_i(\mathbf{w}^*, b^*) \leq 0, \qquad i = 1, \ldots, m \tag{2.5}$$

$$\alpha_i^* \geq 0, \qquad i = 1, \ldots, m\ . \tag{2.6}$$

By applying the first two equations we get

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \qquad 0 = \sum_{i=1}^{m} \alpha_i y_i \tag{2.7}$$

and

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j = \mathbf{f}^T \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha}$$

where $\mathbf{f} = (1, 1, \ldots, 1)^T$ and $(\mathbf{H})_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$.

As a result, we eliminated the variables $\mathbf{w}$ and b, and obtained a quadratic function that has to be maximised with respect to $\boldsymbol{\alpha} > 0$, $\sum_{i=1}^{m} \alpha_i y_i = 0$. This is a standard *quadratic programming* problem with linear constraints, that can be solved by well-known interative techniques (e.g. interior point methods [19]).

To summarize, the maximal margin classifier for a linearly separable training set can be found by solving the following maximization problem:

$$\max_{\boldsymbol{\alpha}} \left(\mathbf{f}^T \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} \right), \qquad \boldsymbol{\alpha} \geq 0, \qquad \sum_{i=1}^{m} \alpha_i y_i = 0\ . \tag{2.8}$$

Having found $\boldsymbol{\alpha}$, we can calculate $\mathbf{w}$ and b using the relations (2.4) and (2.7).

Usually, most of the coefficients α_i are equal to zero, because of the relation (2.4) and the fact that for many training points the constraints (2.1) hold with strict inequalities. The data points i, for which $\alpha_i \neq 0$ are called *support vectors* (see figure 2.2). The small number of support vectors is what makes the maximal margin separating hyperplane very stable and robust in practical classification problems.

2.4.3 Soft-margin Classifier

The solution derived above is only applicable to linearly separable data, but it is easily extended to the linearly nonseparable case by introducing *slack variables* ξ_i. Instead of requiring that $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$ for each i, we allow the classifier to violate the constraint by ξ_i:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i\ .$$

We also incorporate the slack variables in the function to be minimized: instead of minimizing $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ we minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \xi_i$. The parameter C controls the "cost" of errors. The resulting classifier is called *soft margin classifier*. It turns out that in order to find the soft margin classifier the following maximization problem must be solved:

$$\max_{\boldsymbol{\alpha}} \left(\mathbf{f}^T \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} \right), \qquad C \geq \boldsymbol{\alpha} \geq 0, \qquad \sum_{i=1}^{m} \alpha_i y_i = 0\ . \tag{2.9}$$

The only difference from the hard-margin case (2.8) is the additional upper bound constraint C on all the α_i.

2.4.4 The Kernel Trick

Note that in order to find the maximal margin or the soft-margin classifiers we only need the *inner products* $\mathbf{x}_i^T\mathbf{x}_j$ between the data points (in the maximization problems the points $\mathbf{x}_i$ are only present in the entries of the matrix $\mathbf{H}$ in the form of pairwise inner products). Also, the solution vector $\mathbf{w}$ is represented as a linear combination of the data points: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$. These two observations allow to apply an interesting idea, commonly called the *kernel trick*.

Let $\phi : \mathbb{R}^m \to \mathbb{R}^k$ be a nonlinear function that maps the training points to a very high-dimensional space $\mathbb{R}^k$. Denote by $K(\mathbf{x}, \mathbf{y})$ the inner product $\phi(\mathbf{x})^T\phi(\mathbf{y})$ and suppose that we can calculate the function K efficiently, perhaps without the need of explicitly mapping the points $\mathbf{x}$ to their corresponding images $\phi(\mathbf{x})$. As a simple example consider the function $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$:

$$K(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{m} x_i y_i\right)^2 = \sum_{i,j=1}^{m} x_i y_i x_j y_j = \sum_{i,j} (x_i x_j)(y_i y_j) = \phi(\mathbf{x})^T\phi(\mathbf{y})$$

where $\phi(\mathbf{x}) := (x_1x_1, x_1x_2, x_1x_3, \dots, x_2x_1, x_2x_2, \dots, x_mx_m)$, i.e. ϕ maps a given vector to a vector of pairwise products of its elements. The function K is called a *kernel*.

Given such a map ϕ and a corresponding kernel K, we can efficiently apply the support vector machine training algorithm on the data points transformed by this map, $\phi(\mathbf{x}_i)$. Indeed, let us substitute $\phi(\mathbf{x}_i)$ for $\mathbf{x}_i$ everywhere in the algorithms above. That will change the definition of $(\mathbf{H})_{ij}$ to $y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, therefore we shall still be able to calculate this matrix efficiently. The vector $\mathbf{w}$ will become $\sum_{i=i}^{m} y_i \phi(\mathbf{x}_i)$, and it will no longer be possible to calculate $\mathbf{w}$ efficiently, but that is not a problem because in order to perform classification it is enough to know the variables α_i only:

$$\begin{aligned}\operatorname{sign}(\mathbf{w}^T\phi(\mathbf{x}) + b) &= \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x}) + b\right) \\ &= \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right) .\end{aligned}$$

It is this *kernelized* version of the *soft-margin classifier* that is usually meant under the term *support vector machine (SVM)*. Here is a quick summary of SVM training and classification procedures:

- *Training*

  - Given: training set S, kernel function K
  - Find: $\boldsymbol{\alpha}$ and b by solving (2.9) and applying (2.4)

- *Classification*
  - Given: training set S, kernel function K, trained classifier $(\boldsymbol{\alpha}, b)$, a point to be classified $\mathbf{x}$.
  - Return: the classification y of point $\mathbf{x}$ calculated as

$$y = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right) .$$

It is interesting to note that the function ϕ is present in the algorithm only indirectly, via the kernel K. We may therefore choose a kernel function K without even knowing what is the corresponding feature map ϕ. We only need to check that K satisfies the condition of the *Mercer's theorem*, that states that a symmetric function $K(\mathbf{x}, \mathbf{y})$ can be expressed as an inner product $\phi(\mathbf{x})^T \phi(\mathbf{y})$ for some ϕ iff $K(\mathbf{x}, \mathbf{y})$ is positive semidefinite, i.e.

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{y} \geq 0 \qquad \text{for any } g .$$

This condition is equivalent to the requirement that for any set of points $\{\mathbf{x}_i \mid i = 1 \ldots n\}$ the *kernel matrix* $\mathbf{K}$, $(\mathbf{K})_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ is positive semidefinite. The condition has been verified for several common *kernel-functions*, such as the *polynomial kernel* $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^k$, the *gaussian kernel* $K(\mathbf{x}, \mathbf{y}) = \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$ and some other [14, 13].

2.4.5 SVM in Practice

As already mentioned, the main problem of the SVM method is its computational complexity. Training has quadratic memory requirements (an $n \times n$ matrix needs to be stored) and the solution of the quadratic programming problem is nearly quadratic in complexity even with SVM-specific optimization algorithms like SMO (*sequential minimal optimization*, [20]) . This limits this method to the training sets of not much more than about 10000 elements.

There is certainly more that can be said on the subject of support vector machines and kernel methods, but we stop here and refer the reader to the books [12, 13, 14] for further information.

Chapter 3

A Linear Model for Gene Expression

I just got lost in thought. It was an unfamiliar territory.

This chapter introduces the main result of this thesis: a linear model for gene expression analysis that incorporates both microarray data and DNA sequence information. First, some motivation for such analysis is presented, that is followed by a more precise statement of the problem. Next the design of a specific model is discussed and an algorithm is derived for finding a least-squares fit for it.

3.1 Motivation

Microarray experiments produce a vast amount of data. The analysis of this data is, however, not straightforward, as the biological mechanisms underlying gene expression are not completely understood. Moreover, the goal of microarray data analysis is often seen in the *determination* of these mechanisms.

In particular, some insight may be obtained by learning predictive models on the data, that is, models that can predict the expression level of a given gene from the expression levels of other genes. Such a model may be used to identify genes playing important role in transcription regulation in at least two ways. Firstly, the model itself may be descriptive and explicitly specify which genes are important for prediction. The classical example of such a model is a decision tree. Indeed, attempts were made to build decision trees on microarray data, reporting satisfactory results [4, 5]. Another way to infer information from a model is to examine its prediction performance when trained on different sets of predictor genes. The set of genes that shows the best performance should be considered biologically important.

However, the statistical significance of a model built on solely the expression data is questionable. If we want to predict expression of a gene from expressions of other genes we have to consider the *columns* of the expression matrix as the instances (training points) given to the algorithm. The number of columns in a typical expression matrix is often prohibitively small (about hundred or so), and it is not clear whether a machine learning algorithm will be able to infer meaningful results from such scarce data. Adding some external information, such as DNA sequence data, might improve the situation.

There exist different possibilities for the analysis of data, that combines microarray measurements with DNA sequences. Examples are combinatorial analysis of motif correlation [3], two-dimensional regression trees [2] and ADT-trees [6]. In the following we propose yet another approach, based on a specific linear model. Exposition follows in a top-down manner, first presenting the problems and ideas and concluding with the specification of the method itself.

3.2 Notation

In the following we differentiate between two kinds of genes: the ones that correspond to actual or putative transcription factors and all the rest. We call the former *TF*-s (or *predictor genes*) and designate them as t_k, $k \in \{1, 2, \ldots, n_t\}$ where n_t denotes the number of TF-s. To the remaining genes we refer simply as *genes* and denote them as g_i, $i \in \{1, 2, \ldots, n_g\}$ where n_g is the number of *genes*.

Microarray experiment data is usually represented as an *expression matrix* $\mathbf{E}$ with rows corresponding to genes, and columns to arrays (or conditions). Denote by n_a the number of arrays in this matrix and the arrays themselves by a_j, $j \in \{1, 2, \ldots, n_a\}$. We group all the rows of the matrix $\mathbf{E}$ corresponding to *TF*-s in a separate *TF expression matrix*, $\mathbf{T}$, and all the remaining rows in a *gene expression matrix*, $\mathbf{G}$. So G_{ij} denotes the expression level of gene g_i in the array a_j and T_{kj} is the expression level of t_k in a_j.

Besides microarray data we also have the DNA sequences. Although it is not impossible to use raw character sequences in the analysis, it is much more convenient to work with numeric data, so we transform each sequence to a vector of counts of certain *motifs* in that sequence. A *motif* is such a subsequence of the DNA for which it is known (or supposed) that certain TF-s attach there[1]. In our analysis we fix a certain set of motifs: m_l, $l \in \{1, 2, \ldots, n_m\}$ and for each gene g_i and motif m_l we calculate the number of times M_{il} this motif is present in the promoter region of that gene. This results in a *motif matrix* $\mathbf{M}$.

The matrices $\mathbf{T}$, $\mathbf{G}$ and $\mathbf{M}$ make up the data to be analyzed. Figure 3.1 shows

[1]In practice motifs are usually represented as probabilistic objects by means of *position weight matrices* [21] or *hidden markov models* [9]

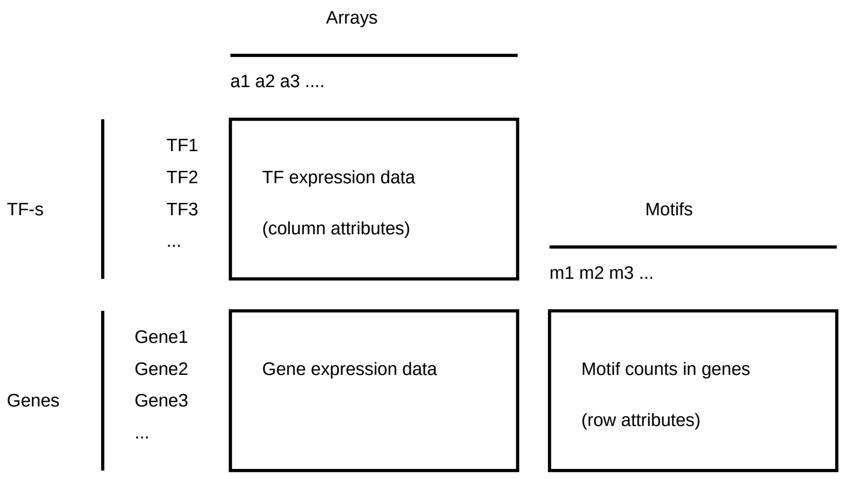

Figure 3.1: The matrices **T** (top), **G** (bottom left) and **M** (bottom right). Each row of **G** corresponds to a certain gene, as does each row of **M**. Each column of **G** corresponds to a certain array, as does each column of **T**. The rows of **M** therefore provide "additional information" about the rows of **G** and the columns of **T** provide additional information about the columns of **G**.

a convenient way to visualize these matrices. Following [2] we call the rows of the motif matrix *row attributes* of **G**, and the columns of **T** — *column attributes* of **G**.

In the derivation of the algorithm we need to refer to separate rows and columns of the matrices **M** and **T**. We denote the i-th row of the matrix **M** by M_{i*} and treat it as a row vector. The l-th column of the matrix **M** is denoted by M_{*l} and is treated as a column vector. Analogously, we refer to the k-th row and j-th column of the matrix **T** as T_{k*} and T_{*j} respectively.

3.3 Modeling the Expression of Genes

Now we can formulate our task more precisely. We shall try to find a model that can predict the expression G_{ij} of gene g_i on the array a_j, given the description of the gene as a vector of its motif counts $M_{i*} = (M_{i1}, M_{i2}, \ldots, M_{in_m})$, and the specification of the array as a vector of the expression levels of the TF-s on that array: $T_{*j} = (T_{1j}, T_{2j}, \ldots, T_{n_tj})^T$. In other words, we shall be predicting the value in a cell of **G** given the corresponding row of **M** (row-attributes) and the corresponding column of **T** (column-attributes). Such formulation was borrowed from [2] and [6].

In order to design the model we make some assumptions concerning the processes that underlie microarray experiments:

1. We assume that gene expression is regulated by TF proteins. Each gene's expression may be expressed as a function of the expressions of its regulators.

2. We assume that TF-s perform their regulatory function by *attaching* to certain motifs in the promoter region of the gene. A TF, when attached to a motif, may thus induce or suppress the transcription of the corresponding gene. Some genes require an ensemble of TF-s to be attached to their promoter in order to get expressed. If the promoter region of the gene does not contain a motif, to which a given TF might attach itself, this TF does not participate in the regulation of the gene.

If we knew all the underlying regulatory rules, the relations between motifs and TF-s, and the states of all the regulators, we could in principle precisely predict the expression state of all the genes. That is, if we knew the rules, we could calculate the matrix **G** given **M** and **T**. The problem under consideration is, however, precisely the inverse: knowing the matrices **G**, **T** and **M** we attempt to restore the regulatory relations.

3.4 Design of the Model

As noted above, we use the assumption that the expression level of each gene can be expressed as a rule depending on the expression levels of the TF-s. Let us also assume for now that there are two expression states: *upregulated* and *downregulated*, and use the predicate UP to denote expression state of a gene. That is, we say that UP(g) is true when g is upregulated and false otherwise. A regulatory rule for a gene g might then look for example like that:

$$\textsc{up}(g) = (\textsc{up}(t_1)\,\&\,\neg\textsc{up}(t_2))\,|\,\textsc{up}(t_3)\ .$$

Let T_1, T_2, T_3 and G be real numbers such that:

$$G = \begin{cases} 1, & \text{if } \textsc{up}(g) \\ 0, & \text{otherwise} \end{cases} \qquad T_k = \begin{cases} 1, & \text{if } \textsc{up}(t_k) \\ 0, & \text{otherwise} \end{cases} \qquad k \in \{1,2,3\}\ .$$

It is then easy to see that the above rule is equivalent to

$$G = H(T_1\,\mathrm{not}(T_2) + T_3)$$

where $\mathrm{not}(x) = 1 - x$ and H is the Heaviside step function, i.e. $H(x) = 1$ if $x > 0$ and 0 otherwise.

Due to our second assumption, there must exist a binding site m_l for each TF t_k that participates in the regulation of the gene, and the rule may be trivially rewritten as

$$\begin{aligned}\textsc{up}(g) = (&\textsc{up}(t_1)\,\&\,\textsc{hasmotif}(g, m_1)\,\&\,\textsc{binds}(t_1, m_1)\,\& \\ &\&\,\neg\textsc{up}(t_2)\,\&\,\textsc{hasmotif}(g, m_2)\,\&\,\textsc{binds}(m_2, t_2))\,| \\ &|\,(\textsc{up}(t_3)\,\&\,\textsc{hasmotif}(g, m_3)\,\&\,\textsc{binds}(m_3, t_3))\end{aligned}$$

where $\textsc{hasmotif}(g, m)$ denotes that gene g has the motif m, and $\textsc{binds}(m, t)$ denotes that TF t binds to the motif m. Let M_{il} be the numeric representation of $\textsc{hasmotif}(g_i, m_l)$ (i.e. $M_{il} = 1$ if the corresponding predicate is true nad $M_{il} = 0$ otherwise), and let α'_{lk} be the numeric representation of $\textsc{binds}(m_l, t_k)$. The above rule is then equivalent to

$$G = H(\alpha'_{11}M_1T_1\alpha'_{22}M_2\,\mathrm{not}(T_2) + \alpha'_{33}M_3T_3)$$

It is clear that we may express the regulatory rule for any gene in the same way, which is in its most general form

$$G = H\left(\sum_{K=\{k_1,\ldots,k_c\}\subset\{1,\ldots,n_t\}}\ \sum_{F=(f_1,\ldots,f_c)\in\{\mathrm{id},\mathrm{not}\}^c} d(K, F)\sum_{l=1}^{n_m}\prod_{i=1}^{c}\alpha'_{lk_i}M_l f_i(T_{k_i})\right)$$

where id is the identity function and d is a parameter of the model. Such model, although biologically motivated, is far too complex to be fitted using the available data. In order to reduce the complexity we drop all the terms from the outermost sum where $|K| > 1$ and obtain

$$G = H\left(\sum_{k=1}^{n_t}\sum_{l=1}^{n_m}(d(k, \mathrm{id})\alpha'_{lk}M_lT_k + d(k, \mathrm{not})\alpha'_{lk}M_l\,\mathrm{not}(T_k))\right)$$

which is equivalent to

$$G = H\left(\sum_{k=1}^{n_t}\sum_{l=1}^{n_m}\alpha_{lk}M_lT_k + \sum_{l=1}^{n_m}\alpha_{l0}M_l + b\right)$$

for some choice of α_{lk} and b.

Note, that although the variables α_{lk} here do not directly correspond to the predicate $\textsc{binds}$ anymore, they still express the importance of the relation between a motif and a TF. Also note, that the presence of H and the discretization of expression states are not necessarily needed. Finally, we may drop the sum[2] $\sum_{l=1}^{n_m}\alpha_{l0}M_l$. It will still make sense to search for parameters α_{lk} and b that describe the (continuous) expression value G as a linear combination of pairwise products M_lT_k.

$$G = \sum_{l,k}\alpha_{lk}M_lT_k + b\ . \tag{3.1}$$

[2] Or we may simulate it by adding a TF with a constant expression level of 1

where T_k are continuonus TF expression values.

It is this model that we shall be fitting. The model is descriptive in the sense that we can search for putative relations between TF-s and motifs by examining the values of the coefficients α_{lk}. The model is also predictive as it can be used to predict the gene expression values of previously unseen genes. The model is linear with respect to the products $M_l T_k$ and has a reasonably small number of parameters, so that we may hope that it won't overfit the data too much.

It should also make sense to apply the model in the context of linear classification, where the expression levels are discretized to $+1$ and -1. In this case the model equation (3.1) takes the form of

$$G = \operatorname{sign}\left(\sum_{l,k} \alpha_{lk} M_l T_k + b\right) \tag{3.2}$$

We shall refer to this variation as the *classification version* and to the variation defined by equation (3.1) as the *regression version*.

It is clear from the preceding discussion that neither version of the model takes into account the "crosstalk" of different TF-s, so the regulatory rules containing lots of conjunctions of the form $\textsc{up}(t_1)\,\&\,\textsc{up}(t_2)$ will probably cause problems for the model. Rules with such conjunctions could be perhaps better modeled by including the products of the form $T_1 T_2$ in the design, but in this case the number of model parameters would grow in an unmanageable manner.

The parameters of the model may be fitted using standard techniques like linear regression or support-vector machines. In the following we also derive an especially efficient way of calculating the least-squares fit for the parameters of the regression version of the model.

3.5 Least-Squares Fit for the Model

Let $\mathbf{A}$ denote the matrix of the parameters α_{lk}, that is, $(\mathbf{A})_{lk} = \alpha_{lk}$. We assume here without loss of generality that the data is centered, so $b = 0$. The problem is to find the *least-squares* fit for the parameters α_{lk}, given the matrices $\mathbf{G}$, $\mathbf{M}$ and $\mathbf{T}$.

Let us fix some value for the parameter matrix $\mathbf{A}$. Consider a gene g_i and an array a_j. The gene g_i is in our setting described by its motif vector M_{i*} and the array t_j is given by the corresponding TF expression vector T_{*j}. Given this data, the model would predict the expression value $\hat{G}_{ij}$ of gene g_i in array a_j as

$$\hat{G}_{ij} = \sum_{l,k} \alpha_{lk} M_{il} T_{kj} \ .$$

By using the fact, that for any matrix $\mathbf{M}$ and column-vectors $\mathbf{x}$, $\mathbf{y}$ of the appropriate size, the equation $\mathbf{x}^T\mathbf{M}\mathbf{y} = \sum_{i,j} (\mathbf{M})_{ij} x_i y_j$ holds true, we may conveniently rewrite the former formula in a more compact way:

$$\hat{G}_{ij} = \sum_{l,k} \alpha_{lk} M_{il} T_{kj} = M_{i*} \mathbf{A} T_{*j} \ .$$

Next we group the equations for all possible i and j in one single matrix equation

$$\hat{\mathbf{G}} = \mathbf{MAT} \ .$$

Finding the least squares fit for the parameter matrix means finding a matrix $\mathbf{A}$ that minimizes the sum of squares of errors in the predictions $\hat{G}_{ij}$

$$\mathcal{E}(\mathbf{A}) = \sum_{i,j} (G_{ij} - \hat{G}_{ij})^2 \ .$$

In other words, we wish to find the matrix $\mathbf{A}$ for which the predicted expression matrix $\hat{\mathbf{G}}$ is as close as possible to the true matrix $\mathbf{G}$. This idea allows to express the task in a simple manner:

Find $\mathbf{A}$ for which

$$\mathbf{G} \approx \mathbf{MAT}$$

This equation suggests the codename *G-MAT* for the model.

In order to find the minimum of $\mathcal{E}(\mathbf{A})$ (which is unique here, as $\mathcal{E}$ is a quadratic function) we set the partial derivatives of $\mathcal{E}$ to zero and get:

$$\frac{\partial \mathcal{E}(\mathbf{A})}{\partial \alpha_{lk}} = 2 \sum_{i,j} (G_{ij} - M_{i*} \mathbf{A} T_{*j})(-M_{il} T_{kj}) = 0 \qquad \forall\, l,k$$

$$\sum_{i,j} M_{i*} \mathbf{A} T_{*j} M_{il} T_{kj} = \sum_{i,j} G_{ij} M_{il} T_{kj} \qquad \forall\, l,k \ .$$

This corresponds to the matrix equation

$$M_{*l}^T \mathbf{MAT} T_{k*}^T = M_{*l}^T \mathbf{G} T_{k*}^T \qquad \forall\, l,k \ .$$

Grouping the equations for all l and k together into one matrix equation, we get

$$\mathbf{M}^T \mathbf{MATT}^T = \mathbf{M}^T \mathbf{GT}^T$$

from which it follows that

$$\mathbf{A} = (\mathbf{M}^T \mathbf{M})^{-1} \mathbf{M}^T \mathbf{GT}^T (\mathbf{TT}^T)^{-1} \ . \tag{3.3}$$

The number of rows of the matrix $\mathbf{M}$ is usually much greater than the number of its columns ($n_g > n_m$). Therefore, rank($\mathbf{M}$) $= n_m$, which is the necessary and sufficient condition for $\mathbf{M}^T\mathbf{M}$ to be invertible. The matrix $\mathbf{TT}^T$, however, will often be singular, because the number of TF-s n_t might be greater than the number of arrays n_a. In order to make it invertible, we may either reduce the number of TF-s in the analysis, or regularize $\mathbf{T}\mathbf{T}^T$ by adding a small number λ to the diagonal elements (a trick known as ridge regression, [22]). The solution (3.3) then obtains the form

$$\mathbf{A} = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{G}\mathbf{T}^T(\mathbf{T}\mathbf{T}^T + \lambda\mathbf{I})^{-1}\ . \tag{3.4}$$

The time complexity of the calculation of $\mathbf{A}$ depends linearly on n_g and n_a, and is cubic in $\max(n_t, n_m)$. Memory requirements are quadratic in $\max(n_m, n_t)$ and linear in n_g. This is very efficient when compared to the naïve attempt of applying standard techniques on the dataset of $n_g \times n_a$ points (one data point per each cell of $\mathbf{G}$) and $n_m \times n_t$ attributes (one attribute per pair $(\text{motif}, \text{TF})$).

3.6 Support Vector Machine Version

It is common to discretize the expression levels into two or three states (*upregulated/downregulated/normal*). In this case the classification version (3.2) of the model is more applicable.

We shall apply the support-vector machine classifier algorithm to fit the parameters of the model. Unfortunately, SVM training is computationally demanding: for a training set of size n it requires solving a quadratic programme with a $n \times n$ Hessian matrix. In our case the size of the training set would be $n_g n_a$ as there is one training instance per each entry of the matrix $\mathbf{G}$. Solving such a large quadratic problem is completely unfeasible, and we didn't find a way to adapt the SVM algorithm to our special case to achieve any significant speedup. We shall therefore have to use a smaller set of size n_{train} to train the algorithm. That means, we shall choose a set I of pairs of indices (i, j) refering to the elements of the matrix $\mathbf{G}$:

$$I \subset \{1, \ldots, n_g\} \times \{1, \ldots, n_a\}, \qquad |S| = n_{\text{train}}$$

and for each element $s = (i, j)$ of I we define the corresponding training sample $(\mathbf{x}_s, y_s)$ as

$$\mathbf{x}_s = (M_{i1}T_{1j}, M_{i1}T_{2j}, \ldots, M_{i2}T_{1j}, M_{i2}T_{2j}, \ldots, M_{in_m}T_{n_t j}), \quad y_s = G_{ij}\ .$$

We then feed these training instances to the SVM training algorithm.

There is one optimization we can make, however. The SVM algorithm does not require the training samples $\mathbf{x}_s$ as input, but rather their *inner products*. It turns out that we may calculate inner products of the training instances in a way that is more efficient than the straightforward approach. Let $s_1 = (i_1, j_1)$, $s_2 = (i_2, j_2)$. Consider the inner product $\mathbf{x}_{s_1}^T \mathbf{x}_{s_2}$:

$$\begin{aligned}\mathbf{x}_{s_1}^T \mathbf{x}_{s_2} &= \sum_{l,k} M_{i_1 l} T_{k j_1} M_{i_2 l} T_{k j_2} = \left(\sum_l M_{i_1 l} M_{i_2 l} \right) \left(\sum_k T_{k j_1} T_{k j_2} \right) \\ &= (M_{i_1 *} M_{i_2 *}^T)(T_{* j_1}^T T_{* j_2}) \ .\end{aligned}$$

Besides being more efficient to calculate, such representation allows to plug in any kernels instead of the inner products $M_{i_1 *} M_{i_2 *}^T$ and $T_{* j_1}^T T_{* j_2}$. In particular, the inner product of the motif vectors might be replaced by a string kernel calculated on the corresponding DNA sequences.

3.7 Interpretation

As already noted, both the regression and the classification versions of the model have a rather clear interpretation. Namely, a large positive or negative value of $\boldsymbol{\alpha}_{lk}$ should indicate the possibility for some biological connection to exist between the TF t_k and the motif m_l. It is not easy to derive a precise statistical test on the significance of a particular value of α_{lk}, but we may estimate significance using randomization tests (introduced later).

Finally, the error of the model $\mathcal{E}(\mathbf{A})$ might indicate the "goodness" of the chosen set of predictor genes. We may optimize this "goodness" and thus find a set of genes that should afterwards be examined for biological significance.

Chapter 4

Experiments

Experience is something you don't get until just after you need it.

In this chapter we first present a toy example illustrating the ideas of the method, and then attempt to apply the technique on a larger artificial dataset that simulates, to some extent, the real-life situation.[1].

4.1 A Toy Example

Let us design a small example of the "regulatory network" that works in accordance with the proposed model, and examine the capabilities of the method to restore the regulatory relations given the "microarray measurements" and "sequence data".

Suppose that there exist 5 transcription factors t_k, 5 motifs m_l, and that the following relations hold among them:

- TF t_1 is a transcriptional activator that binds to motif m_1.
- t_2 is a strong transcriptional suppressor, it binds to motif m_2.
- The TF t_3 also binds to m_2, but unlike t_2 acts as an activator.
- t_4 can bind to m_3 and m_4 and acts as an activator when bound to m_3 and as a suppressor when bound to m_4.
- t_5 is an activator that can bind to both m_4 and m_5, but its activity is stronger when it is bound to m_5.

[1]Unfortunately, a thorough analysis of the performance of the method on a real dataset was not completed in this work due to lack of timely access to data, late-to-discover bugs in the analysis and some other mistakes of the author.

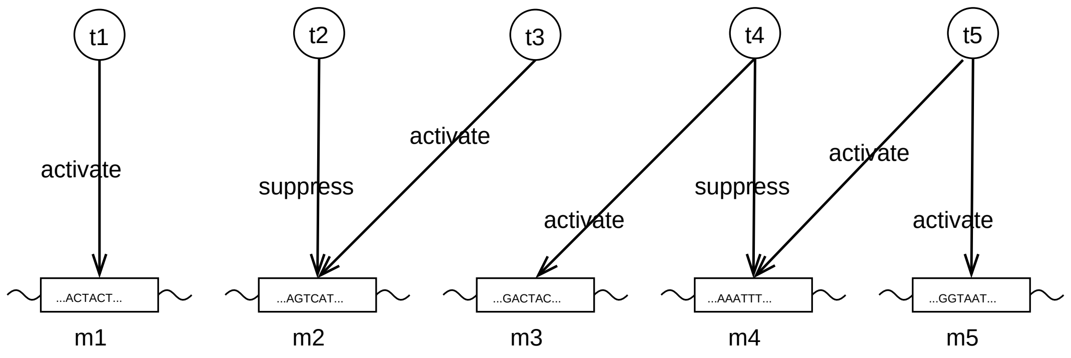

Figure 4.1: The regulatory rules of the toy example. Arrows indicate which TF-s bind to which motifs and what effect they have on transcription regulation.

These relations are depicted in the figure 4.1.

The relations are described by the following matrix $\mathbf{A}_{\text{true}}$:

$$\begin{array}{c|ccccc} & t_1 & t_2 & t_3 & t_4 & t_5 \\ \hline m_1 & 1 & 0 & 0 & 0 & 0 \\ m_2 & 0 & -2 & 1 & 0 & 0 \\ m_3 & 0 & 0 & 0 & 1 & 0 \\ m_4 & 0 & 0 & 0 & -1 & 1 \\ m_5 & 0 & 0 & 0 & 0 & 2 \end{array}.$$

A positive entry $\alpha_{lk} = (\mathbf{A}_{\text{true}})_{lk}$ indicates that TF t_k induces transcription when bound to motif m_l. A negative entry denotes the suppressive effect of the corresponding TF. Larger values indicate stronger effect.

Suppose we have a set of 5 genes, described by the following motif matrix $\mathbf{M}$:

$$\begin{array}{c|ccccc} & m_1 & m_2 & m_3 & m_4 & m_5 \\ \hline g_1 & 1 & 1 & 0 & 0 & 0 \\ g_2 & 0 & 0 & 1 & 1 & 0 \\ g_3 & 0 & 1 & 0 & 1 & 0 \\ g_4 & 0 & 0 & 1 & 0 & 0 \\ g_5 & 0 & 0 & 1 & 0 & 1 \end{array}.$$

That is, the promoter region of gene g_1 contains motifs m_1 and m_2, the promoter region of g_2 contains m_3 and m_4, etc.

Imagine that we performed 4 microarray experiments a_1, a_2, a_3, a_4, and the expression levels of the TF-s in these experiments are given by the following TF expression matrix $\mathbf{T}$:

	a_1	a_2	a_3	a_4
t_1	1	0	1	0
t_2	0	1	0	1
t_3	1	1	0	1
t_4	0	0	1	0
t_5	0	1	0	0

.

It means that TF t_1 was upregulated in the experiments a_1 and a_3, TF t_2 was upregulated in the experiments a_2 and a_4, etc.

At last, suppose that genetic regulation indeed follows the proposed model: the expression of a gene can be written as a sum of the "effects" of the TF-s that bind to the gene's promoter. The effect of each TF t_k depends on its expression level T_k and the motif m_l to which it binds. The parameters α_{lk} describe the effect of a TF t_k bound to motif m_l.

Having all the information, we can determine the expression of each g_i in each experiment a_j. For example, the gene g_1 contains motifs m_1 and m_2 and is therefore regulated by the TF-s t_1, t_2 and t_3. In the experiment a_1 the TF-s t_1 and t_2 are upregulated. Each of them has an effect of 1 on the gene. Therefore the total expression level of g_1 in a_1 is $1 + 1 = 2$. As another example, consider the expression of g_1 in a_2. Here the TF-s t_2, t_3 and t_5 are upregulated. TF t_2 has the effect of -2 on the expression level of g_1, the effect of t_3 is 1, and the TF t_5 cannot bind to g_1 and thus has no influence on it. The total expression level of g_1 is therefore $-2 + 1 = -1$. Continuing in the same manner we can calculate the full expression matrix $\mathbf{G}$:

	a_1	a_2	a_3	a_4
g_1	2	-1	1	-1
g_2	0	1	0	0
g_3	1	0	-1	-1
g_4	0	0	1	0
g_5	0	2	1	0

.

.

We shall now illustrate how the proposed analysis method can restore the "regulatory relations" $\mathbf{A}_{\text{true}}$, from the matrices $\mathbf{G}$, $\mathbf{M}$ and $\mathbf{T}$. We cannot use the equation (3.3) here because the number of TF-s is greater than the number of arrays and the matrix $\mathbf{TT}^T$ is thus singular. It means that there is not enough information about the matrix $\mathbf{A}_{\text{true}}$ in the data, and the best we can do is attempt the regularized equation (3.4). We apply that formula with $\lambda = 0.01$ and obtain the following solution **A**:

$$\mathbf{A} = \begin{array}{c|rrrrr} & t_1 & t_2 & t_3 & t_4 & t_5 \\ \hline m_1 & 0.75 & -0.25 & 0.25 & 0.25 & 0 \\ m_2 & 0.74 & -1.23 & -0.24 & -0.73 & -0.01 \\ m_3 & 0.25 & 0.24 & -0.25 & 0.74 & 0 \\ m_4 & -0.25 & -0.23 & 0.25 & -0.74 & 0.98 \\ m_5 & 0 & 0.01 & 0 & 0 & 1.96 \end{array} .$$

Due to incomplete data, the obtained result is not precisely equal to the true $\mathbf{A}_{\text{true}}$ underlying our genetic network. To obtain more interpretable results we round the values to closest integers and obtain

$$\mathbf{A} = \begin{array}{c|rrrrr} & t_1 & t_2 & t_3 & t_4 & t_5 \\ \hline m_1 & 1 & 0 & 0 & 0 & 0 \\ m_2 & 1 & -1 & 0 & -1 & 0 \\ m_3 & 0 & 0 & 0 & 1 & 0 \\ m_4 & 0 & 0 & 0 & -1 & 1 \\ m_5 & 0 & 0 & 0 & 0 & 2 \end{array} .$$

It is easy to confirm that this matrix is rather close to the original $\mathbf{A}_{\text{true}}$. There are some mistakes, however. For example, the regulatory role of TF t_3 was not detected. The reason might lie in the fact that in the generated "experiments" this TF was too often upregulated together with t_2. The two TF-s bind to the same motif m_2, but the suppressing effect of t_2 is stronger than the inducing effect of t_3. This is probably also the reason why t_2 was detected as only a weak suppressor. One more mistake is the spurious identification of the pair (m_2, t_4). Despite these errors, the precision of the obtained results is in general quite satisfactory.

If we add one more experiment a_5 to the analysis, the problem becomes well-defined and we may use the formula (3.3) to restore $\mathbf{A}_{\text{true}}$. In this case the restored matrix **A** is, of course, precisely equal to the original $\mathbf{A}_{\text{true}}$.

In the following we shall attempt to apply the method on a larger dataset, that is not generated using the same model that we are fitting.

4.2 The Synthetic Dataset

We examine the ability of the model to detect TF-motif relations "hidden" in a synthetic dataset, created in the following manner:

- There are 100 TF-s, of which 80 are assumed to participate in the regulatory process (call them *important*).
- There are 100 motifs, of which 80 are considered important.

- The relationship of the 80 *important* TF-s and motifs is described by a randomly initialized "binding matrix" B. A nonzero entry B_{lk} identifies that a TF t_k may bind to motif m_l. About 2% of entries in the matrix are nonzero.
- A set of 3000 genes is used, of which 2500 are regulated via certain rules. A motif vector is initialized randomly for each gene.
- For each of the 2500 correctly regulated genes a random regulatory rule is created in the following manner
  - A random subset of *important* TF-s that can bind to the gene is chosen.
  - Each TF in the chosen subset is assigned either *positive influence* or *negative influence*.
  - The rule is the following: a gene is *upregulated* if there are more upregulated TF-s with positive influence than there are TF-s with negative influence; the gene is *downregulated* if there are more upregulated TF-s with *negative influence*; otherwise the gene is in *normal state*.
- 200 arrays are simulated. In each array the expression levels of the TF-s are chosen at random, the expression levels of the 2500 *regulated* genes are calculated using the corresponding rules, and the expression levels of the remaining genes are selected at random.
- At last, data of 20 *important* TF-s and 20 *important* motifs is removed.

Such a dataset corresponds, to some extent, to our assumptions about the transcription regulation processes, but at the same time it is not generated using the same model that we shall be fitting. There are no complex rules that could completely confuse a linear model but there is some data missing and a considerable amount of noise present. The point of interest is the ability of the model to restore the information in the *binding matrix*. We assume that the model's parameter matrix $\mathbf{A}$ will provide this information.

4.3 Linear Regression

We start by fitting the linear version of the model using the equation (3.3). After obtaining the matrix $\mathbf{A}$ we need to analyze which entries of $\boldsymbol{\alpha}_{lk}$ indicate significant connection between the corresponding TF and motif. In order to do that we use a *randomization test*: we shuffle the rows of the motif matrix $\mathbf{M}$ and the columns of the TF expression matrix $\mathbf{T}$ and try to fit the model on this randomized data obtaining the matrix $\tilde{\mathbf{A}}$. Due to the random shuffling of motifs

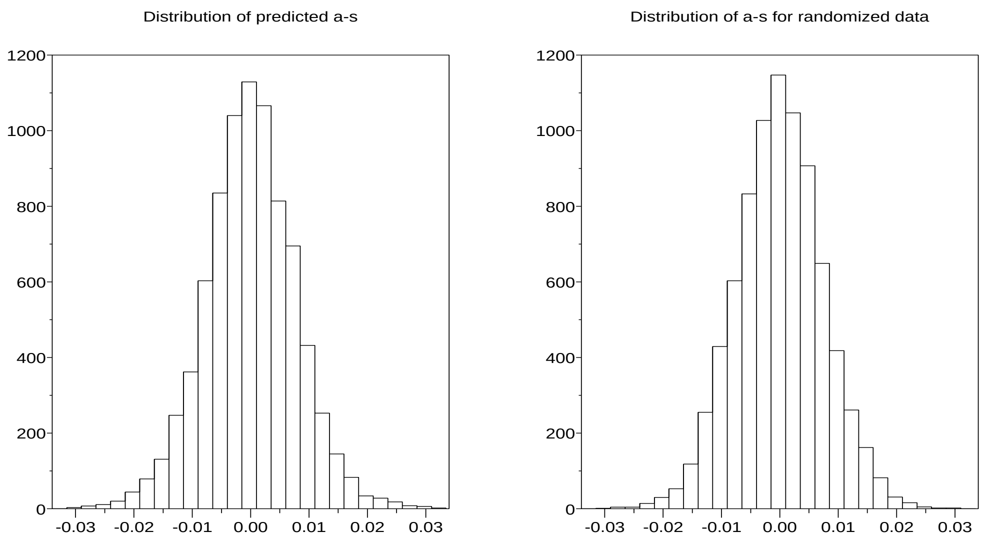

Figure 4.2: Distributions of the entries of the matrix $\mathbf{A}$ for a linear regression model fitted on a synthetic dataset (left) and on the same data with motifs and TF-s randomized (right).

and TF-s there is no "biological" information in the data anymore, therefore we might consider entries of the matrix $\tilde{\mathbf{A}}$ as purely random. By comparing the distributions of the entries of the matrices $\mathbf{A}$ and $\tilde{\mathbf{A}}$ (figure 4.2) we see that the former has slightly heavier "tails". The difference is tiny indeed, but its presence can be verified by repeating the experiment many times. The tails of the distribution of true $\boldsymbol{\alpha}_{lk}$ always turn out to be a bit heavier. We assume that these "tails" contain the values that we should consider significant.

More precisely, let q_l and q_h be the 0.5 and the 99.5 quantiles of the distribution of $\tilde{\mathbf{A}}$ correspondingly. It means that in the random case, 99% of entries fall into the interval $[q_l, q_h]$. We consider as significant all the values of $\mathbf{A}$, that are not in this interval. That is, if for some l and k, $\alpha_{lk} \notin [q_l, q_h]$, we state that the corresponding entry of the binding matrix is nonzero.

We consider two indicators for the obtained result:

- *Specificity:* The percentage of the predicted entries α_{lk}, that turned out to be correct (i.e. for which the corresponding entry of the binding matrix is nonzero).
- *Sensitivity:* The percentage of the entries of the binding matrix, that were correctly detected by the model.

In the performed experiments the specificity turned out to be about 10-15%

on average, and the sensitivity about 20-25%. Although the number is not very impressive, it is still a rather good result. It means, that every eighth relation predicted by the method holds true, and that the method managed to determine 20% of all relations. Due to the way the data was generated, the probability of finding a related pair (m_l, t_k) by making a random choice is less than 2%. Therefore finding 20% of such pairs with 10% precision is indeed a reasonable achievement.

4.4 Support Vector Machine Classification

The support vector machine classifier version of the model did not do as well in the same kind of analysis. Both sensitivity and specificity of the predictions of the binding matrix were less than 5%. The reason for such performance lies probably in the fact that we had to select a relatively small training sample for the analysis—less than 2000 instances out of the maximum possible set of $3000 \times 200 = 600000$ instances. Otherwise the training just takes too much time.

In order to still be able to examine the usefulness of the SVM version of the model we created a really small dataset consisting of 100 genes, 10 arrays, 30 TF-s and 30 motifs, and filling about 7% of the entries in the binding matrix. We performed the analysis using the SVM classifier on this dataset. Unfortunately, the results were unsatisfactory: both sensitivity and the specificity of the trained model were under 10%. In the case of this small dataset it is not significantly better than a random selection of pairs.

Conclusion

A conclusion is the place where you got tired of thinking.

To summarize, a new method for analysis of microarray data has been developed, that combines microarray data with DNA sequence data. The idea of the method is to fit a certain linear model to the data and examine its coefficients. The method was tested on a synthetic dataset where it reported satisfactory performance.

There are basically two variations of the method: one based on linear regression and the other — on linear classification. In the first case, an efficient algorithm for fitting the model was developed. In the second case no such algorithm has yet been found. An attempt to use the SVM algorithm for fitting the classifier model failed due to the complexity of this algorithm. However, there are still several possibilities that can be tested such as the Fisher's determinant and logistic regression.

A Linear Model of Genetic Transcription Regulation that Combines Microarray and Genome Sequence Data

Bachelor thesis

Konstantin Tretyakov

Abstract

The thesis proposes a novel method for the analysis of microarray data based on fitting a specific linear model that combines microarray data with DNA sequence information. The model is both descriptive and predictive: its coefficients provide insight into the structure of the genetic regulatory networks, and its predictive performance may be used to find a set of genes that play important role in transcription regulation (transcription factors). An efficient algorithm is proposed for calculating the least-squares fit for the parameters of the model.

The proposed method is tested on a synthetic dataset and the results indicate that the approach is capable of detecting interesting relations in the data.

Lineaarne mudel geeni ekspressiooni andmete analüüsi jaoks

Bakalaureusetöö (10 ap)

Konstantin Tretjakov

Resümee

DNA mikrokiipide abil saadud geeni ekspressiooni andmete analüüsitehnikate väljatöötamine on praegu üks olulisematest uurimissuundadest bioinformaatikas. Antud töö pakub uue meetodi geeni ekspressiooni andmete analüüsimiseks, mis põhineb teatud lineaarse mudeli sobitamises geeni ekspressiooni ja DNA järjendite andmetele.

Vaatleme geenide hulka $\{g_i \,|\, i = 1 \ldots n_g\}$, transkriptsioonifaktorite (TF-ide) hulka $\{t_k \,|\, k = 1 \ldots n_t\}$, eksperimentide hulka $\{a_j \,|\, j = 1 \ldots n_a\}$ ning DNA-motiivide hulka $\{m_l \,|\, l = 1 \ldots n_m\}$. Kasutame järgmisi tähistusi:

- $\mathbf{G}$ tähistab geenide ekspressiooni maatriksit, mille read vastavad geenidele g_i, veerud — eksperimentidele a_j, ning iga väärtus G_{ij} näitab geeni g_i ekspressioonitaset eksperimendis a_j.
- $\mathbf{T}$ tähistab tranksriptsioonifaktorite ekspressiooni maatriksit, mis on analoogne $\mathbf{G}$-le, kuid read vastavad TF-idele, mitte geenidele.
- $\mathbf{M}$ on motiivide maatriks, kus M_{il} näitab kui mitu korda esineb motiiv m_l geeni g_i promootor piirkonnas.

Meetodi idee on leida lineaarne regressioon, mis suudab ennustada iga geeni ekspressioonitaset G teades iga motiivi m_l jaoks selle motiivi esinemiste arvu M_l selle geeni promootoris ning iga TF t_k jaoks tema ekspressioonitaset T_k. Mudel omab kuju

$$G = \sum_{l=1}^{n_m} \sum_{k=1}^{n_k} \alpha_{lk} M_l T_k \,.$$

Alternatiivselt võib üritada sobitada ka klassifitseerija-mudelit

$$G = \operatorname{sign}\left(\sum_{l=1}^{n_m} \sum_{k=1}^{n_k} \alpha_{lk} M_l T_k \right) \,.$$

mis ennustab geeni jaoks kas see on “sisse” või “välja” lülitatud.

Töös esitatakse mitteformaalsed põhjendused, miks sellise mudeli otsimine võib omada mõtet bioloogia seisukohast.

Mudeli parameetrite α_{lk} jaoks saab regressiooni puhul suhteliselt efektiivselt leida vähimruutude hinnangu andmetest $\mathbf{G}$, $\mathbf{M}$ ja $\mathbf{T}$. Kui $\mathbf{A}$ tähistab parameetrite maatriksit $((\mathbf{A})_{lk} = \alpha_{lk})$ siis osutub, et parameetrite hinnang avaldub

$$\mathbf{A} = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{G}\mathbf{T}^T(\mathbf{T}\mathbf{T}^T)^{-1}\,.$$

Klassifitseerija-põhise mudeli sobitamiseks efektiivset algoritmi käesolevas töös kahjuks ei leitud.

Treenitud mudeli parameetrid on lihtsasti interpreteeritavad. Juhul kui mõni parameeter α_{lk} omab suurt absoluutväärtust, tähendab see, et motiivi m_l ja TF-i t_k vahel võib olla bioloogiline seos.

Töö lõpus hinnatakse mudeli headust lihtsa sünteetilise andmestiku peal. Osutus et regressioonil põhinev mudel suudab teatud määral tuvastada andmetes peidetud seosed. Klassifitseerija-mudeli sobitamist prooviti tugivektor masina (SVM) algoritmi abil. See katse kukkus kahjuks läbi eelkõige selle algoritmi ajalise keerukuse tõttu.

References

[1] Daniel P. Berrar, Werner Dubitzky, and Martin Granzow, editors. *A Practical Approach to Microarray Data Analysis*. Kluwer Academic Publishers, 2003. 384 pages.

[2] Jianhua Ruan and Weixiong Zhang. A two-dimensional regression tree approach to the modeling of gene expression regulations. http://cic.cs.wustl.edu/bdtree/ (Last visited: May 1, 2005).

[3] Y. Pilpel, P. Sudarsanam, and G. M. Church. Identifying regulatory networks by combinatorial analysis of promoter elements. *Nat Genet*, 29(2):153–9, Oct 2001.

[4] L. A. Soinov. Supervised classification for gene network reconstruction. *Biochem Soc Trans*, 31(Pt 6):1497–502, Dec 2003.

[5] Lev A Soinov, Maria A Krestyaninova, and Alvis Brazma. Towards reconstruction of gene networks from expression data by supervised learning. *Genome Biol*, 4(1):R6, 2003.

[6] Manuel Middendorf, Anshul Kundaje, Chris Wiggins, Yoav Freund, and Christina Leslie. Predicting genetic regulatory response using classification. *Bioinformatics*, 20 Suppl 1:I232–I240, Aug 2004.

[7] Thomas Schlitt, Kimmo Palin, Johan Rung, Sabine Dietmann, Michael Lappe, Esko Ukkonen, and Alvis Brazma. From gene networks to gene function. *Genome Res*, 13(12):2568–76, Dec 2003.

[8] Kimmo Palin, Esko Ukkonen, Alvis Brazma, and Jaak Vilo. Correlating gene promoters and expression in gene disruption experiments. *Bioinformatics*, 18 Suppl 2:S172–80, Oct 2002.

[9] Pierre Baldi and Søren Brunak. *Bioinformatics: The Machine Learning Approach*. The MIT Press, second edition, 2001. 400 pages.

[10] Wolfgang Härdle and Leopold Simar. *Applied Multivariate Statistical Analysis*. Springer-Verlag, 2003. 486 pages.

[11] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997. 414 pages.

[12] John Shawe-Taylor and Nello Christianini. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1st edition, 2000. 189 pages.

[13] John Shawe-Taylor and Nello Christianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004. 476 pages.

[14] B.Scholkopf and A. Smola. *Learning with Kernels*. The MIT Press, 1st edition, 2001. 644 pages.

[15] R. Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan Books, Washington, DC, 1962.

[16] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1998. 842 pages.

[17] V. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. 736 pages.

[18] H.W.Kuhn and A.W.Tucker. Nonlinear programming. In J.Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, 1950.

[19] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 11:451–484, 1999.

[20] J. Platt. *Advances in Kernel Methods - Support Vector Learning*, chapter Fast Training of Support Vector Machines using Sequential Minimal Optimization, pages 41–65. The MIT Press, 1998.

[21] Triinu Tasa. Kaalumaatriksite rakendusi bioinformaatikas, 2005. semestritöö.

[22] Vojislav Kecman. *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. The MIT Press, Cambridge, MA, USA, 2001. 608 pages.