Computer
Sciences
Department

Novel Uses for Machine Learning and Other Computational Methods for the Design and Interpretation of Genetic Microarrays

Michael N. Molla

Technical Report #1612

September 2007

UNIVERSITY OF
WISCONSIN
MADISON

NOVEL USES FOR MACHINE LEARNING
AND OTHER COMPUTATIONAL METHODS FOR
THE DESIGN AND INTERPRETATION OF GENETIC MICROARRAYS

by

Michael N. Molla

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2007

for Rebecca, Celia, and Eva

ACKNOWLEDGMENTS

First I would like to acknowledge my advisor, Jude Shavlik, for his patient guidance throughout. His careful attention and rigorous approach have not only helped to develop my work, but have also set the ideal example of how to perform meaningful professional research.

Next, I would like to thank the other members of my thesis committee: Fred Blattner, David Page, Mark Craven and Chuck Dyer. They all provided important suggestions and insights that improved the thesis dramatically.

I would also like to acknowledge the other members of the University of Wisconsin Machine Learning Research Group: Mark Rich, Ameet Soni, Lisa Torrey, Frank DiMaio, Jesse Davis, Sean McIlwain, Adam Smith, Yue Pan, Burr Settles, Mike Waddell, Louis Oliphant, Kieth Noto, Trevor Walker, Irene Ong, Rich Maclin, Soumya Ray, Joseph Bockhorst. They provided a community and an environment in which I could thrive.

Since 2001, NimbleGen Systems Inc. (now *Roche Nimblegen*) has provided me with not only a series of interesting problems to work on, and, through my consulting, the funding to work on them, but NimbleGen has also provided superb collaborators too numerous to mention. I would like to specifically acknowledge several who were of special help with regard to my thesis work: Roland Green, Emile Nuwaysir, Steven Smith, Tom Albert, Todd Richmond, Nan Jang, Jacob Kitzman and Matt Rosesch have all been very helpful along the way.

I would like to thank Peter Andrae for joining and substantially improving our text mining project; and John Tobler for his part in our joint work on microarray design. I would also like to thank my close friend Eric Haag for allowing me to join his work in compensatory evolution.

This work was supported by the following grants: NIH 2 R44 HG02193-02, NLM 1 R01 LM07050-01, NSF IRI-9502990, NIH 2 P30 CA14520-29, NIH 5 T32 GM08349. I also received three years of support from the Biotechnology Training Program (BTP, NIH Grant 5T32GM08349) and 3 months of support from the Computation and Informatics in Biology and Medicine (CIBM) Training Program (NLM Grant 5T15LM007359). I would also like to acknowledge Timothy Donohue and Beth Holden of the BTP and Louise Pape of the CIBM program for their kind assistance during and since my participation in these training programs.

Finally, I would like to acknowledge my family. My wife, Rebecca, and my children Celia and Eva have been constant sources of inspiration and support. My parents' love and generosity have helped to make this possible. I would also like to thank Rebecca's parents for their gracious encouragement and advice throughout.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Appendix

Page

Appendix

Page

DISCARD THIS PAGE

LIST OF TABLES

Table Page

DISCARD THIS PAGE

LIST OF FIGURES

Appendix
Figure Page

Appendix
Figure Page

NOVEL USES FOR MACHINE LEARNING AND OTHER COMPUTATIONAL METHODS FOR THE DESIGN AND INTERPRETATION OF GENETIC MICROARRAYS

Michael N. Molla

Under the supervision of Professor Jude W. Shavlik
At the University of Wisconsin-Madison

It is clear that high-throughput techniques, such as rapid DNA sequencing and gene chips are changing the science of genetics. Hypothesis-driven science is now strongly complemented by these newer data-driven approaches. Over the course of the past decade, DNA microarrays, also known as *gene chips*, have come into prominence for genetic-level analysis throughout the life sciences. Using these microarrays, a scientist is able to perform hundreds of thousands of experiments on the surface of a single one-inch-by-one-inch wafer in the space of a single afternoon, generating more data than an army of researchers could have a generation ago. This potential flood of data brings many informatic challenges in both analysis and design. It is well understood that computer science will play a crucial role in their development and application. This thesis presents novel applications of machine learning and other computational methods to central tasks in high-throughput biology. These tasks include gene-chip design, detection of genomic variation, and the interpretation of gene-expression patterns.

Jude W. Shavlik

Chapter 1

Introduction

My research focuses on applying machine learning to problems in genetics. High-throughput techniques, such as rapid DNA sequencing (Gilbert and Maxam 1973) and gene chips (Schena et al. 1995), are changing the science of genetics. Hypothesis-driven science is now strongly complemented by these newer data-driven approaches (Cristianini and Hahn 2006, Baldi and Brunak 2001). It is well understood that computer science will play a crucial role in their development and application (Hanson and Coontz 2001). Machine learning has been of particular value in this domain (Molla et al. 2004a). The specific challenge that I address in this work is the application of machine learning to the design and interpretation of gene-expresion microarrays or *gene chips*.

1.1 Gene-Chip Design

As will be described in the chapters to come, genetic microarrays, commonly known as *gene chips*, make it possible to simultaneously infer the rate at which a cell or tissue is expressing - translating into a protein - each of its thousands of genes. One can use these comprehensive snapshots of biological activity to infer regulatory pathways in cells, identify novel targets for drug design, and improve the diagnosis, prognosis, and treatment of patients suffering from disease. However, the amount of data this new technology produces is more than one can manually analyze. Hence, the need for automated analysis of gene-chip data offers an opportunity for machine learning to have a significant impact on biology and medicine.

Maskless array synthesis (Nuwaysir et al. 2002) is an efficient method for producing a gene chip from a design. Applying known machine-learning methods to the problem of gene-chip design (Tobler et al. 2002), we have shown that machine learning can be effectively applied to this problem. Our findings from this work are still in use today by a local gene-chip manufacturer, Nimblegen Systems, Inc. The ability to design gene chips is a major bottleneck between today's chief high-throughput genetics technologies: the DNA sequencer and the gene chip.

1.2 Genetic Variation

Single Nucleotide Polymorphisms (SNPs) are positions in a genome that contain single-base variations across the population. They are important in the diagnosis and prediction of many diseases such as Alzheimer's and certain types of cancer. Copy-number variations are regions of the genome that appear at different frequencies in different individuals. These are also common and can also indicate specific types of cancer. In collaboration with Nimblegen Systems, we have developed algorithmic methods to help identify both by improving the analysis of gene chips designed for this purpose (Molla et al. 2004b; Albert et al. 2005).

1.3 Gene-Expression Analysis

In addition to the vast amount of data being generated through the use of gene chips, the amount of annotated genomic data available in public databases is also rapidly increasing. Through collaboration with University of Wisconsin's *E. coli* lab, we developed automated methods for combining these two sources of information to produce insight into the operation of cells under various conditions. Our approach uses machine-learning techniques to identify words associated with genes that are up-regulated or down-regulated in a particular microarray experiment. (Molla et al. 2002; Molla et al. 2004c).

1.4 Thesis Statement

Over the course of the past decade, DNA microarrays, also known as “gene chips,” have come into prominence for genetic-level analysis throughout the life sciences. Using these microarrays, a scientist is able to perform hundreds of thousands of experiments on the surface of a single one-inch-by-one-inch wafer in the space of a single afternoon, generating more data than an army of researchers could have a generation ago. This potential flood of data brings many informatic challenges in both analysis and design. It is my thesis that novel applications of machine learning can help to solve varied and important problems in this domain.

1.5 Overview

In the next two chapters, I will present some important background information. This includes some relevant molecular biology and how it relates to the work in this thesis (Chapter 2) and provide an introduction to machine-learning methodology and the algorithms that I use in this work (Chapter 3). Chapter 4 will include my work on microarray design. I made use of this work in helping to develop a new technology called *Microarray-Based Genome Enrichment* which will be described in Chapter 5. The following two chapters describe algorithmic solutions to problems of interpreting chips specifically designed to discover instances of the two most common classes of genetic differences between individuals: *copy-number polymorphisms* (Chapter 6) and *Single Nucleotide Polymorphisms (SNPs)* (Chapter 7). Chapter 8 describes a generalization of the approach described in Chapter 7 so that it can be applied to other machine-learning problems. Chapter 9 describes my machine-learning approach to gene-expression analysis. Chapter 10 describes two more computational solutions to biological problems: the problem of efficiently finding unique probes sequences for use in microarrays and work that I did to model a specific evolutionary process to evaluate competing theories that attempt to describe it (Haag and Molla 2005). In Chapter 11, I discuss my overall contributions and possible future directions.

Chapter 2

Molecular-Biology Background

Here I give brief descriptions of the biological technologies involved in the studies that I have done. I also describe some of the previous work in these areas that is related to my own work. I also briefly mention some of my work, though it is described in much more detail in the chapters to come.

2.1 The *Central Dogma* of Molecular Biology

At each point in the life of a given cell, various proteins, the building blocks of life, are being produced. Each protein is encoded by a specific gene or group of genes. It is by turning on and off the production of specific proteins that an organism responds to environmental and biological situations, such as stress, and to different developmental stages, such as cell division.

Genes are contained in the DNA of the organism. The mechanism by which proteins are produced from their corresponding genes is a two-step process. The first step is the transcription of a gene from DNA into a temporary molecule known as RNA. During the second step - translation - cellular machinery builds a protein using the RNA message as a blueprint. Although there are exceptions to this process, these steps (along with DNA replication) are known as the central dogma of molecular biology.

One property that DNA and RNA have in common is that each is a chain of chemicals known as bases. In the case of DNA these bases are Adenine, Cytosine, Guanine and Thymine, commonly referred to as *A*, *C*, *G* and *T*, respectively. RNA has the same set of four bases, except that instead of Thymine, RNA has Uracil - commonly referred to as *U*.

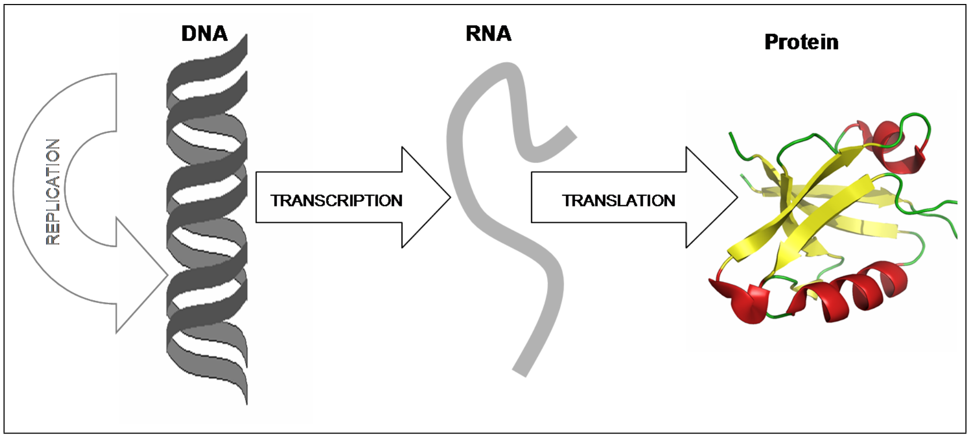


Figure 2.1 **The central dogma of molecular biology.** When a gene is expressed, it is first transcribed into an RNA sequence, and the RNA is then translated into a protein, a sequence of amino acids. DNA is also replicated when a cell divides, but my work only focuses on the DNA-to-RNA-to-Protein process.

2.2 Complementarity

Another property that DNA and RNA have in common is called *complementarity*. Each base only binds well with its complement: A with T (or U) and G with C. As a result of complementarity, a strand of either DNA or RNA has a strong affinity for what is known as its *reverse complement*. This is a strand of either DNA or RNA that has bases exactly complementary to the original strand, as Figure 2.2 illustrates.

Complementarity is central to the double-stranded structure of DNA and the process of DNA replication. It is also vital to transcription. In addition to its role in these natural processes, molecular biologists have, for decades, taken advantage of complementarity to detect specific sequences of bases within strands of DNA and RNA. One does this by first synthesizing a *probe*, a piece of DNA that is the reverse complement of a sequence one wants to detect, and then introducing this probe to a solution containing the genetic material (DNA or RNA) to be searched. This solution of genetic material is called the *sample*. In theory, the probe will bind to the sample if and only if the probe finds its complement in the sample (but as I later discuss in some detail, this does not always happen in practice and this imperfect process provides an excellent opportunity for machine learning). The act of binding between probe and sample is called *hybridization*. Prior to the

DNA	GTAAGGCCCTCGTTGAGTCGTATT
RNA	CAUUCCGGGAGCAACUCAGCAUAA

Figure 2.2 **Complementary binding between DNA and RNA sequences**

experiment, one labels the probes using a fluorescent tag. After the hybridization experiment, one can easily scan to see if the probe has hybridized to its reverse complement in the sample. In this way, the molecular biologist can determine the presence or absence of the sequence of interest in the sample.

2.3 Gene Chips

More recently, DNA probe technology has been adapted for detection of, not just one sequence, but hundreds of thousands simultaneously. This is done by synthesizing a large number of different probes and either carefully placing each probe at a specific position on a glass slide (so-called spotted arrays (Schena et al. 1995)) or by attaching the probes to specific positions on some surface (Fodor et al. 1991, Singh-Gasson et al. 1999). Figure 2.3 illustrates the latter case, which has become the predominant approach as the technology has matured. Such a device is called a *microarray* or *gene chip*.

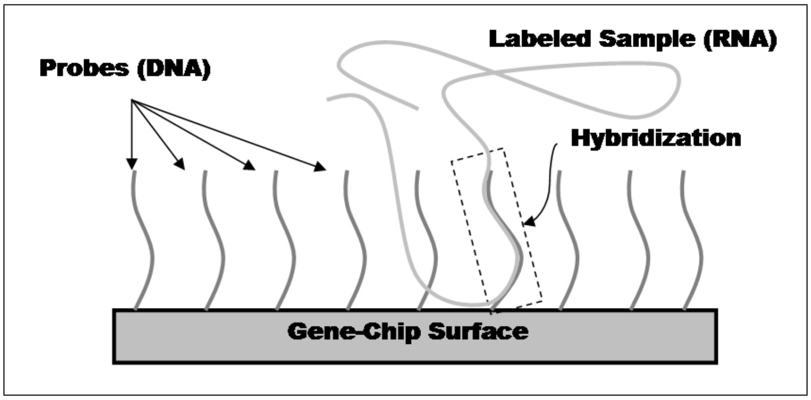


Figure 2.3 **Probes are typically between 25 and 100 bases long**, whereas samples are usually about 10 times as long, with a large variation due to the process that breaks up long sequences of RNA into small samples.

Utilization of these chips involves labeling the sample rather than the probe, spreading thousands of copies of this labeled sample across the chip, and washing away any copies of the sample that do not remain bound to some probe. Since the probes are attached at specific locations on the chip, if labeled sample is detected at any position on the chip, one can determine which probe has hybridized to its complement.

2.4 Gene-Expression Analysis

One common use of gene chips is to measure the expression level of various genes in an organism. An expression level measures the rate at which a particular gene is being transcribed. This is used as a proxy measure for the amount of corresponding protein being produced within an organism's cells at a given time.

Ideally, biologists would measure the protein-production rate directly, but doing so is currently very difficult and impractical on a large scale. So one instead measures the expression level of various genes by estimating the amount of RNA for that gene that is currently present in the cell. Since the cell degrades RNA very quickly, this level will accurately reflect the rate at which the cell is producing the corresponding protein. In order to find the expression level of a group of genes, one labels the RNA from a cell or a group of cells and spreads the RNA across a chip that contains probes for the genes of interest. A single gene chip can hold enough probes to monitor tens of thousands of genes.

Typically, the use of microarrays specifically designed to measure gene expression, known as *gene-expression arrays*, involves many experiments measuring the same set of genes under various circumstances (e.g., under normal conditions, when the cells are heated up or cooled down, or when some drug is added), at various time points (e.g., 5, 10, and 15 minutes after adding an antibiotic; due to the steps one needs to manually perform to produce an RNA sample, sub-minute resolution is not currently feasible) or in different cell types, individuals or developmental stages.

In order to make use of such an experiment, one must use the data derived from microarray experiments to build a hypothetical description of the processes underlying the observed pattern of expression. The development of microarrays and their associated large collections of experimental data have led to the need for automated methods that assist in the interpretation of microarray-based biomedical experiments.

One class of methods for interpreting these experiments is known as *clustering*. This is the partitioning of genes into groups based on a *distance metric*. The distance metric can vary from task to task. It defines the similarity between two genes based on information about the genes. This allows the partitioning to produce groups of similar genes.

Some recent approaches to clustering genes rely not only on the expression data, but also on background knowledge about the problem domain. Hanisch et al. (2002) present one such approach. They add a term to their distance metric that represents the distance between two genes in a known biological-reaction network. The BIOLINGUA system of Shrager et al. (2002) also uses a network graph describing a known biological pathway and updates it using the results of microarray experiments. Another source of data is the DNA sequence itself. Craven et al. (2000) use machine learning to integrate *E. coli* DNA sequence data, including geometric properties such as the spacing between adjacent genes and the predicted DNA binding sites of important regulatory proteins, with microarray expression data in order to predict operons. An operon is a set of genes that are transcribed together. Operons provide important clues to gene function because functionally related genes often appear together in the same operon. DNA sequence information is also used in a method that Segal et al. (2001) developed.

Another excellent source of supplementary material is the large amount of human-produced text about the genes on a microarray (and their associated proteins) that is contained in biomedical digital libraries and in the expert-produced annotations in biomedical databases. In a previous paper (Molla et al. 2002), we investigate using the text in the curated SwissProt protein database (Bairoch and Apweiler 2000) as the features characterizing each gene on an *E. coli* microarray. Using these text-based features, we employ a machine-learning algorithm to produce rules that "explain" which genes' expression levels increase when *E. coli* is treated with an antibiotic. See chapter 9 for more additional details.

A great deal of research has been done in text mining, much of which involves biomedical tasks. Hearst's LINDI system (1999) searches medical literature for text relating to a particular subject or problem and tries to make logical connections to form a hypothesis. One of the benefits of my approach is that researchers do not need to know what they are looking for in advance. Given expression data and textual descriptions of the genes represented in the expression data, this system makes an automated “first pass” at discovering what is interesting. The PubGene tool (Jenssen et al. 2001) also interprets gene-expression data based on textual data. One big difference is that PubGene compiles clusters of text ahead of time and tries to match the expression data to an already-made cluster. My tool is designed to allow the expression data itself to define its models. Masys et al. (2001) also use text associated with genes to explain experiments. However, they cluster expression values across experiments and then use the text to explain the clusters, whereas I can use the text directly during the learning process and can explain single experiments.

Since the publication of my work (Molla et al. 2002), others have attempted to use the scientific literature in order to help to automatically interpret gene expression data. One such experiment (Hvidsten et al. 2003) uses a similar rule-based approach using gene-ontology information (Ashburner et al. 2000), rather than free-form text, as features. Another (Glenisson et al. 2003) uses a technique called meta-clustering to combine free-form text data and expression data. Yet another (Raychaudhuri and Altman 2003) develops and tests a method called neighbor divergence per gene which uses natural language processing (NLP) of scientific literature to decide whether or not there is a functional relationship between genes in a microarray-expression experiment.

2.5 Microarray Design

So far I have been presenting the process of measuring gene-expression levels as simply creating one probe per gene and then computing how much RNA is being made by measuring the fluorescence level of the probe-sample hybrid. Not surprisingly, there are complications, and the remainder of this section summarizes the major ones.

Probes on gene chips (see Figure 2.3) are typically between 25 and 100 bases long, since synthesizing substantially longer probes is not generally practical. The protein-coding portions of genes are on the order of a 1000 bases long, and while it may be possible to find a unique 25-base-long probe to represent each gene, most probes do not hybridize to their corresponding sample as well as one would like. For example, a given probe might partially hybridize to other samples, even if the match is not perfect, or the sample might fold up and hybridize to itself. For these reasons, microarrays typically use about a dozen or so probes for each gene, and an algorithm combines the measured fluorescence levels for each probe in this set to estimate the expression level for the associated gene (Li and Stormo 2001, Nuwaysir et al. 2002).

Due to the nature of these experiments, including the fact that microarrays are still a nascent technology, the raw signal values typically contain a great deal of noise (Saiki et al. 1989, Wang et al. 1998, Nuwaysir et al. 2002). Noise can be introduced during the synthesis of probes, the creation and labeling of samples, or the reading of the fluorescent signals. So ideally the data will include replicated experiments. However, each gene-chip experiment can cost several hundred dollars, and so in practice one only replicates each experiment a very small number of times (and, unfortunately, often no replicated experiments are done).

Currently it is not possible to accurately estimate the absolute expression level of a given gene. One work-around is to compute the ratio of fluorescence levels under some experimental condition to those obtained under normal or control conditions. For example, one might compare gene expression under normal circumstances to that when the cell is heated to a higher than normal temperature (so called heat shock); experimenters may say such things as *when E. coli is heated, gene X is expressed at twice its normal rate.* When dealing with such ratios the problem of noise is exacerbated, especially when the numerator or denominator are small numbers. Michael Newton's group, (Newton et al. 2001) have developed a Bayesian method for more reliably estimating these ratios. Another approach is to partner each probe with one or more mismatch probes; these are probes that have different bases from the probe of interest in one or more positions. Each gene's expression score is then a function of the fluorescence levels of the dozen or so match and mismatch probes (Li and Wong 2001).

As previously described, one typically uses a dozen or so probes to represent one gene because the probe-sample binding process is not perfect (Breslauer et al. 1986). If one did a better job of picking good probes, one could not only use fewer probes per gene (and hence test for more genes per microarray), but also get more accurate results.

In a published paper we use machine learning to address the task of choosing good probes (Tobler et al. 2002). The results are encouraging. However, as I will describe in detail in Chapter 9, our experiments were only a partial solution to the important problem of choosing good probes.

Heuristics have been developed to discard probes based on based on knowledge about hybridization characteristics (Lockhart et al. 1996) such as self-hybridization and degenerate repeats. Others have attempted to use melting point (Tm) equations derived from genetic material in solution (Kutara and Suyama 1999; Li and Wong 2001). Kutara and Suyama (1999) also investigate the use of predictions of stable secondary structures and probe uniqueness to create criteria for selecting good probes. Our contribution is a successful empirical evaluation of three standard machine-learning methods applied to the task of learning to predict good probes. Others have since used machine-learning methodology to evaluate oligonucleotide probes (Antipova et al. 2002).

2.6 Identifying Common Forms of Genetic Variation

To date, the genomes of hundreds of organisms have been sequenced. For each of these organisms, a consensus or *reference sequence* has been deposited into a public database. Though this sequence matches the particular individual whose genome was sequenced, other individuals of this species will differ slightly from this reference sequence. One way to identify these differences is to completely sequence, from scratch, the genomes of other individuals of this species and then do a comparison. However, this is costly and generally impractical. A much more cost-effective approach is to use the reference sequence as scaffolding and identify variations from this sequence in various individuals. In addition to their value in gene-expression analysis, gene-chip-based methods have also proven effective in this process of *genome* analysis. Here we describe the state of the art in using gene chips to identify two of the most common types of genomic variation: *Single Nucleotide Polymorphisms (SNPs)* and *Copy-Number Polymorphisms*.

2.6.1 SNP Identification

Most of the genetic variation between individuals is in the form of Single Nucleotide Polymorphisms (SNPs; Altshuler et al., 2000). The process of identifying SNPs through comparison with a known reference sequence is known as resequencing (Saiki et al. 1989).

One method of resequencing that has shown significant results utilizes oligonucleotide microarray technology (Hacia 1999). In particular, this type of resequencing chip consists of a complete tiling of the reference sequence - that is, a chip containing one probe corresponding exactly to each 29-mer in the reference sequence - plus, for each base in this sequence, three mismatch probes: one representing each possible SNP at this position (see Chapter 7 for a more detailed description of this method). In theory, any time a SNP is present, the mismatch probe representing this SNP will have a higher intensity signal than the corresponding probe that matches the reference sequence. However, due to unpredictability in signal strength, varying hybridization efficiency, and various other sources of noise, this method typically results in many base positions whose identities are incorrectly predicted. For this reason, among all the cases where a mismatch probe has more signal intensity than that of the reference sequence's probe, I would like to accurately separate the true SNPs from the noisy, false positives.

Several approaches to this noise-reduction problem have been previously tried (Wang et al. 1998; Hirschhorn et al. 2000; Cutler et al. 2001). Originators of the overall process, the Wang and Hirshhorn groups, focused their attention on the development of the biological methods. In order to interpret the arrays, the Wang group simply took the difference between the two highest-intensity probes as an indication of the likelihood of a SNP at a given position. The Hirshhorn group used the ratio of these values.

In terms of chip interpretation, the most successful method to date has been that of the Cutler group in conjunction with Affymetrix Corporation. They use parametric statistical techniques that take into account the distribution of pixel intensities within each probe's scanned signal pattern. However, this approach presents a number of limitations. Principal among them is the fact that this method is very sensitive to changes in chemistry, scanner type, and chip layout. In order to overcome some of these problems, extensive parameter tuning is required. This involves the analysis of large amounts of data and needs to be re-run any time chemistry, light-gathering technology, or virtually any other experimental condition is changed. Another limitation is that, in order to have a single probe represented by a sufficient number of pixels, a high-resolution scanner must be used.

2.6.2 Copy-Number Variation

Another common form of genetic variation between individuals is know as *copy-number variation*. Copy-number variations are specific sequences that that appear at different frequencies in the genomes of different individuals. Comparative Genomic Hybridization (CGH) is a method for finding copy-number differences between two samples of genomic DNA. These differences, or copy-number polymorphisms, can be either amplifications or deletions and, in humans, are often indicative of specific diseases, including cancer. CGH involves the use of oligoneucleotide microarrays. Specifically, the arrays contain tilings of either specific regions of interest, entire chromosomes, or entire genomes. Identical microarrays are exposed to different samples of genomic DNA and the resulting variations in signal intensity between the two chips can be interpreted as copy-number differences.

The process of interpreting such a chip, i.e. finding the sections of the genome whose copy numbers vary between samples, is called *segmentation*. Unfortunately, due to nonuniformity in hybridization efficiency and other anomalies, the amount of noise accompanying the signal in a typical chip is far from negligible. As a result, the process of segmenting such a chip is not straightforward. In fact, many approaches to this problem are already in use.

Unfortunately, many were designed to work on far fewer datapoints and are, as a result, are much too slow to be run on data such as that produced by Nimblegen microarrays, which typically incorporate hundreds of thousands of features. Two popular algorithms that can be run on such data are *DNACopy* (Olshen et al. 2004) and *StepGram* (Lipson et al. 2006). One way that our method differs from *DNACopy* and *StepGram* is that, rather than greedily taking the best segment and potentially only reaching a local optimum, we use a dynamic-programming algorithm to efficiently arrive at a globally optimal solution in terms of squared error relative to the segment means, a standard statistical measure of segmentation quality (Lipson et al. 2006). However, unlike other dynamic-programming approaches, we start by using the *t-test* statistic to identify a set of candidate breakpoints, which dramatically increases the efficiency of the algorithm.

Other dynamic-programming algorithms have tried to limit the search space. Specifically, Huber et al. (2006) limit the search space by restricting the maximum segment length. However, since no such limitation on segment length exists in nature, it would be better not to have to make such a limitation.

In Chapter 6 I will describe our algorithm and show empirically that it outperforms both *DNA-Copy* and *StepGram* on synthetic and biological data.

Chapter 3

Machine-Learning Background

Here I give brief descriptions of the machine-learning problem formulations and techniques used in this work.

3.1 Machine-Learning Problem Formulations

Machine-learning methods fall into three main categories which will be described here: *supervised learning*, *unsupervised learning*, and *semi-supervised learning*.

3.1.1 Supervised Learning

Nearly all of the studies in my work employ what is know as *supervised learning* (Mitchell 1997). Supervised learning methods train on examples whose categories are known in order to produce a model that can classify new examples that have not been seen by the learner.

Evaluation of this type of learner is typically done through the use of a method called *N-fold cross-validation* (Mitchell 1997), a form of hold-out testing. In hold-out testing, some (e.g., 90%) of the examples are used as the training data for a learning algorithm, while the remaining (*held aside*) examples are used to estimate the future accuracy of the learned model. In N-fold cross validation, the examples are divided into N subsets, and then each subset is successively used as the held-aside test set, while the other $(N-1)$ subsets are pooled to create the training set. The results of all N test-set runs are averaged to find the total accuracy. The typical value for N is 10.

3.1.2 Unsupervised Learning

Unsupervised learning is learning about a set of examples from their features alone; no categories are specified for the examples. Examples of this type are commonly called *unlabeled examples*. In the context of gene chips, for example, this could mean mean learning models of biological processes and relationships among genes based entirely on their expression levels without being able to improve models by checking the learners' answers against some sort of externally provided ground truth.

3.1.3 Semi-Supervised Learning

Until recently, nearly all machine-learning methods could be divided into the aforementioned two categories: unsupervised learning and supervised learning. In order to simultaneously make use of both labeled and unlabeled examples one can use a relatively new class of algorithms: *semi-supervised learning* (Chapelle et al. 2006). This class of learning is particularly useful in domains where unlabeled data are abundant and labeled data are scarce. Evaluation of this type of method usually also involves cross validation.

3.2 Naive Bayes

Naive Bayes (Mitchell 1997) is a practical, successful machine-learning algorithm that assumes independence among the features for a given example conditioned on the class variable. Using this *independence assumption*, a simple ratio can be used to compute the relative likelihood that a test example with features $f_1, f_2, \ldots, f_N$ and feature values $v_1, v_2, \ldots, v_N$ should be labeled positive or negative:

$$NBratio = \frac{P(positive) \prod P(f_i = v_i | positive)}{P(negative) \prod P(f_i = v_i | negative)}$$

where $P(positive)$ and $P(negative)$ are the fraction of training examples labeled positive and negative, respectively. $P(f_i = v_i | positive)$ is estimated by simply counting the number of examples in the training dataset with output labeled positive (or negative for the terms in the denominator) and feature f_i equal to value v_i.

3.3 Decision Trees

Another algorithm used in this work is know as the *decision tree*. The algorithm most often used to generate decision trees is *ID3* (Quinlan 1986) or it successor *C4.5* (Quinlan 1996). This algorithm selects the next node to place in the tree by computing the *information gain* for all candidate features and then choosing the feature that gains the most information about the output category of the current example. Information gain is a measure of how well the given feature separates the remaining training examples, and is based on Shannon's (1948) information theory. Information gain is calculated as described in the following equations.

$$Entropy(S) = -P(negative)log_2P(negative) - P(positive)log_2P(positive)$$

where S is a set of examples, and $P(negative)$ and $P(positive)$ are estimated by computing the fractions of negative and positive labeled examples in S.

$$InfoGain(S, F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $Values(F)$ is the set of all possible values for feature F and S_v is the subset of S for which feature F has value v (Mitchell 1997).

3.4 Artificial Neural Networks (ANNs)

Another approach that I employ is the multi-layered *artificial neural network* (*ANN*) (Figure 3.1). The ANN is an algorithm inspired by the organization of the human brain. Feature values are fed into the *input nodes* as *activation levels* and conveyed to the *hidden nodes* via weighted connections. Likewise, the activation is conveyed from the hidden nodes to the *output nodes* through another set of weighted connections. For two-category classification, there can be two output nodes: one for the *positive* category and one for the *negative* category. Whichever output node has the higher activation as the result of a given example's input feature values defines the example's predicted category.

I train the ANN using *backpropagation* (Rumelhart and Williams 1986), the standard algorithm used for training neural networks. This algorithm attempts to minimize the squared error between

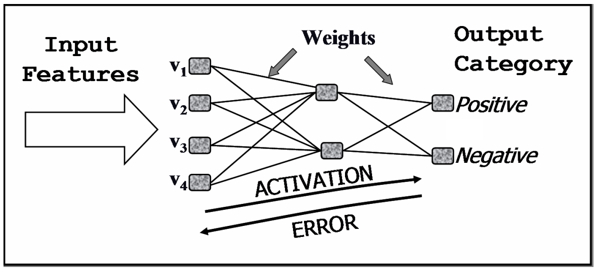


Figure 3.1 **An Artificial Neural Network (ANN),** with 3 layers, 4 input units, 2 hidden units and 2 output units.

the network output values and the target value for these outputs. The algorithm searches a weight space (defined by all possible weight values for each arc in the network) for an error minimum. Because a non-linear, multi-layered network is used, the algorithm is not guaranteed to find the globally minimal error, but rather it may find a local minimum (Mitchell 1997).

3.5 K-Nearest Neighbors Algorithm

In this method one plots examples in an *N*-dimensional space, where the dimensions are features of the examples. In order to interpret an example in this *feature space*, one looks at the K examples nearest to it in this space and uses their categories to predict the class of the example in question. The choice of K typically depends on the task at hand and one can select a good value for *k* by using a third set of data, called a *tuning set*.

Chapter 4

Improving Microarray Design via Machine Learning

Here we apply three well-established, machine-learning techniques to the problem of microarray probe selection, in order to judge how well probe quality can be predicted and which learning algorithm performs best on this important task. Specifically, we compare the performance of artificial neural network (ANN) training, the nave Bayes algorithm, and a decision-tree learner (see Chapter 3). The computational experiments reported here were done jointly by John Tobler and myself.

In order to do this, we frame the problem of probe selection as a category-prediction task. Each 24-base-pair-long (*24bp*) probe is called an example. An example's features are derived (as described below) from the sequence of 24 bases that make up the probe. The goal is to predict the quality of the probe strictly from these features. Our set of probes is the set of all possible 24-bp probes from each of eight genes in the *E. coli* and *B. subtilis* genomes (four genes from each genome), also known as a *tiling* of the genes. As a measure of probe quality, we use the measured fluorescence levels of our probes after they have been exposed to a sample in which all eight of these genes are highly expressed. To summarize, our task is to algorithmically learn how to perform the following:

Given: *a 24-bp probe from a gene not used during training*

Do: *predict if this probe's fluorescence level will rank in the top third of all the possible probes for this gene when exposed to a sample where this gene is highly expressed*

If accurately predicted, this information (when combined with other constraints) can prove highly beneficial when deciding which probes to use to represent a given gene when designing a microarray. If we can increase the probability that a chosen probe will bind strongly when the probe's gene is expressed in the current sample, then fewer probes will be needed per gene that one wishes to detect by the microarray and, hence, more genes can be measured by a gene chip of a given physical size. This process is also described in Tobler et al. (2002).

4.1 Datasets

Our experimental data are tilings of four genes from each of *E. coli* and *B. subtilis*, measured on *maskless* microarrays (Singh-Gasson et al. 1999) produced by NimbleGen, Inc. In order to standardize the data across the eight genes, the measured fluorescence intensity for each probe is normalized such that the top and bottom 2.5% of the measured intensities are labeled 0 and 1 respectively, and all other values are linearly interpolated between 0 and 1. Mapping the top and bottom 2.5% to 0 and 1 reduces the impact of outliers.

Supervised machine-learning algorithms like naive Bayes, decision trees, and ANNs typically require training on a set of data with *labeled* (i.e., categorized) examples; this set of data is called the training set. The classifier produced by the learning algorithm after training is then tested using another dataset, the test set. The predicted classifications of the trained machine-learning classifier are compared to the correct outputs in the test set in order to estimate the accuracy of the classifier. We used the microarray data for our eight genes to generate eight training-set and test-set pairs using the commonly applied leave-one-out method. In this method, each training set consists of the probes from the tilings of seven genes and the corresponding test set contains the remaining gene's probes.

We use only simple features to describe the 24-bp probes (also called *24-mers*) in each data set. Certainly much more expressive data descriptions are possible, but using these features allows us to determine how much useful information is contained in the basic 24-mer structure.

Using the normalized intensity, we chose to label each probe example with a discrete output of *low*, *medium*, or *high*. We sought to roughly evenly distribute the *low*s, *medium*s, and *high*s. The normalized intensity to discrete-output mappings we chose are: the interval [0.00-0.05] maps to *low* (45% of the examples), (0.05-0.15] (23% of the data) to *medium*, and (0.15-1.00] (32%) to *high*.

In all of our experiments, during training we discard all of the probes labeled medium, since those are arguably ambiguous. Of course it would not be fair to ignore the medium probes during the testing phase (and we do not do so), since the objective during testing is to predict the output value for probes not seen during training.

However, we did remove some probes from the test sets, namely those that had less than three mismatches with at least one other 24-bp region in the full genome containing the probe's gene. Such probes are not useful, since they are insufficiently indicative of a single gene. We did leave such *insufficiently unique* probes in the training sets since that may still provide useful information to the learning algorithms. In any case, the number of discarded (from the test sets) genes is very small, only a very small fraction of the probes had at least one close match in their genome.

4.2 Methods

As mentioned above, we evaluate the well-established machine learning algorithms of naive Bayes, decision trees, and artificial neural networks (ANNs) on our task. All of these algorithms are described in detail in the previous chapter. Here I focus on aspects specific to the task of probe selection.

4.2.1 Naive Bayes

As described in Chapter 3, Naive Bayes is a practical, successful machine-learning algorithm that assumes independence among the features for a given example. Recall that, where $P(high)$ and $P(low)$ are the number of training examples labeled *high* and *low*, respectively, $P(f_i = v_i|high)$ is estimated by simply counting the number of examples in the training dataset with output labeled *high* (or *low* for the terms in the denominator) and feature f_i equal to value v_i.

To avoid bias toward underestimating the probability of a given output when an occurrence of the feature value is not seen in the training data, we used the *m*-estimate of probability (Mitchell 1997). We use the following to actually estimate probabilities:

$$P(f_i = v_i | c = \{high, low\}) = \frac{n_c + mp}{n + m}$$

where *c* is either *low* or *high*, n_c is the number of occurrence that have feature value equal to v_i and have output *c*, *n* is the number of examples with output *c*. I choose *m* to equal *n* (I did not experiment with other settings), and $p = \frac{1}{k}$ for features with *k* discrete feature values.

We discretized the non-discrete features by binning them into five equally distributed bins. The naive Bayes algorithm is a fast algorithm for training and classification. Training and classification of a given train/test fold take on average less than 10 minutes on a standard desktop PC.

4.2.2 Decision Trees

The second classifier we evaluate for use in probe selection is a decision tree. The algorithm most often used to generate decision trees is ID3 (Quinlan 1986) or it successor C4.5 (Quinlan 1996).

We use the University of Waikato's Weka 3 Machine Learning Algorithms in Java package (*http://www.cs.waikato.ac.nz/ml/weka/index.html*) to run the decision-tree experiments. The Weka algorithm for decision-tree learning is named J48, but it uses the same algorithm as Quinlan's C4.5. We used Quinlan's reduced-error-pruning algorithm to avoid overfitting of the training data (Quinlan 1996); this procedure removes those portions of the initially induced decision tree that seem to be overfitting the data. At each leaf in the pruned tree, the fraction of the training set reaching that node that was *high* and *low* is recorded. We slightly modified the Weka code to report these fractions when classifying each test example; this provides a crude estimate of the probability that the current test-set example should be called high. Using the WEKA software package, training and classification of a decision tree for a given train-test pair takes under 5 minutes on the standard PC used in all of our experiments.

4.2.3 Neural Networks

The third approach we evaluate is to use a multi-layered ANN trained using backpropagation, which is the standard algorithm used for training neural networks.

The networks we train consist of 485 input units, produced by using one input unit for each real-valued feature and a *1-of-N encoding* for each discrete-valued feature (e.g., n_1). A 1-of-N encoding requires one Boolean-valued input unit for each possible feature value. When an example is input into the network, only the input unit representing that value for a given discrete-valued feature is set to 1 with the other inputs set to 0 (e.g., $n_1 = A$ would be represented by 1, 0, 0, and 0 as the values for the four input units associated with n_1). We also use a single layer of hidden units. In general, too many hidden units leads to overfitting the training data, while using too few hidden units can lead to underfitting the data. Hidden units free an ANN from the constraints of the feature set, and they allow for the network to discover intermediate, non-linear representations of the data (Mitchell 1997). Sarle (1995) and others assert that using large numbers of hidden units is necessary to avoid finding bad local minima. Thus we decided to use 161 hidden units ($\frac{1}{3}\times$ the number of input units), each employing the standard sigmoidal activation function. Finally, we use two output units with sigmoid activation functions. The two output units represent the estimated probability that the output is high and low respectively. The network is fully connected with each input unit connected to each of the hidden units, and each hidden unit connected to each of the output units. Each arc in the network is initialized with a random weight between -0.3 and 0.3, following standard practice.

Training consists of a maximum of 100 cycles through the training set. We use early stopping (**?**)searle.1995) to avoid overfitting the training data by training too long. To decide when to "stop" training, we first create a tuning set of data by randomly removing 10% of the training data, and after each cycle we measure the current accuracy of the ANN's predictions on the tuning set. The ANN's weight settings for the cycle that performs the best on the tuning set are the weight settings that we use to classify the test set. Each training and classification run takes over an order of magnitude longer than those for naive Bayes and decision trees.

4.3 Results

After each algorithm is run on training set *j*, we sort the predicted scores for each probe in test set *j*. For the naive Bayes and ANN classifiers, the sorting is done from the highest to lowest according to the ratio:

$$\frac{prob(label = high \text{ for testset example } x)}{prob(label = low \text{ for testset example } x)}$$

The decision-tree sorting is solely based on $prob(label = high$ for test-set example *x*), which, as mentioned above, is estimated from the distribution of high and low examples in the leaf of the pruned decision tree that is reached by test-set example *x*. By sorting in this manner, we produce an ordering of the best test-set examples as predicted by the machine-learning classifier. The question we would like to answer is the following: Assume we want to get at least *N good* probes for a gene, how far down my sorted list do we need to go? This question is similar to that asked of information-retrieval systems (e. g., search engines): in order to get *N* relevant articles, how many of the highest-scoring articles should be returned? We consider various definitions of a *good* probe in Figure 4.1. In Panel (a), we define *good* as measuring at or higher than 0.5 on my normalized [0-1] scale; about 13.5% of my probes have normalized measured intensities at or above 0.5. For example, when we look at the test-set probes with the 10 highest predicted scores, it turns out that for ANN and naive Bayes nearly all of them had normalized measured intensities at or above 0.5. The *ideal* curve is a 45-degree line: if all of the probes in the top *N* are considered *good*, then the results would fall on the ideal curve. Panels (c) through (d) report the same information for increasingly strict definitions of *good* (6.3% of the probes have normalized intensities at or above 0.75, 3.7% at or above 0.9, and, by construction, 2.5% normalize to 1.0). In all cases, the decision-tree learner performs very poorly. Neural networks and naive Bayes perform well, with the neural networks doing slightly better.

Included in Figure 4.1 is the curve that presents results from ordering probes simply by their predicted melting point. We calculated the melting points using the formula presented by Aboul-ela et al. (1985). As one can clearly see, predictors based on neural networks and naive Bayes

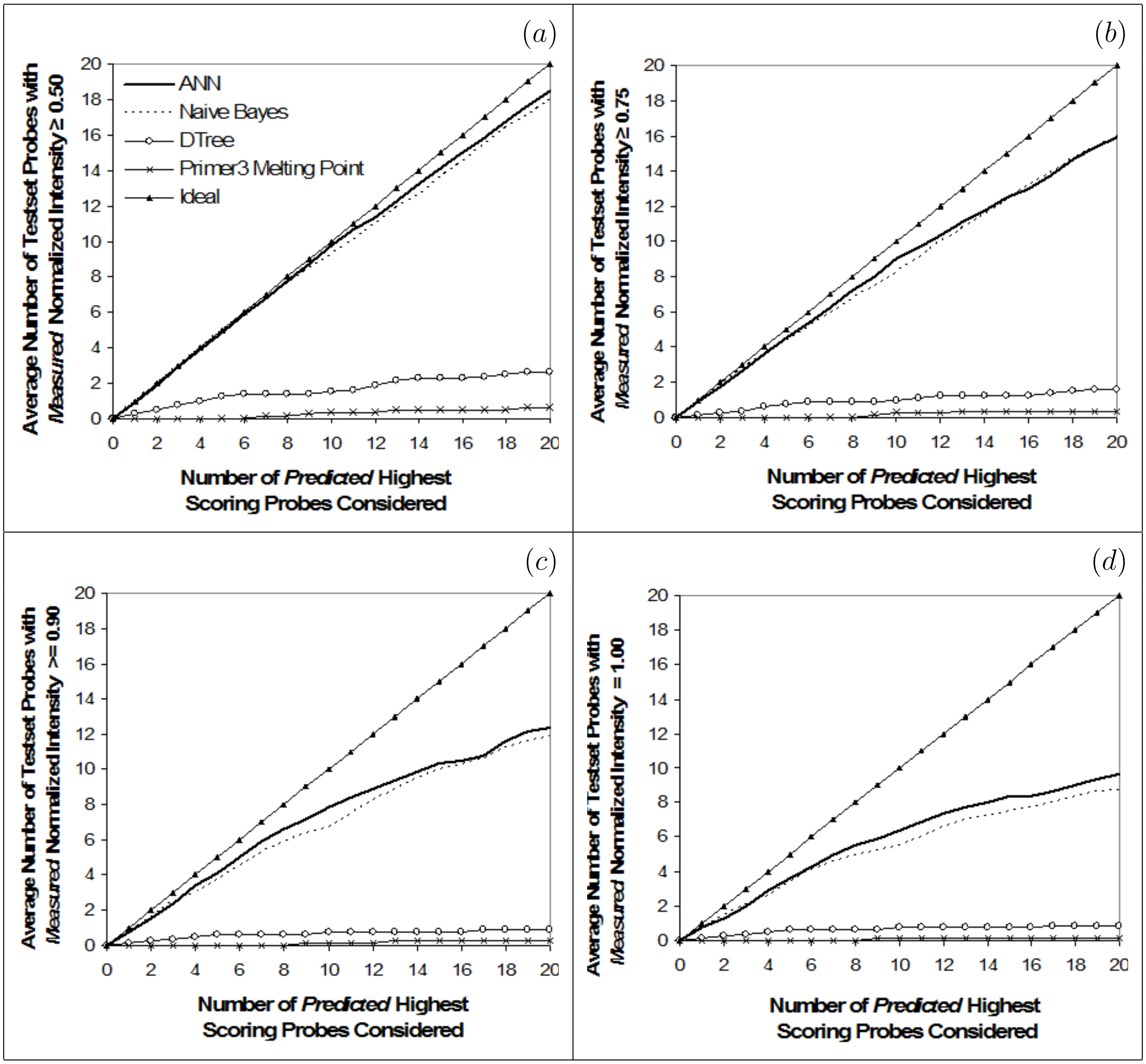


Figure 4.1 **Number of test-set probes in the *N* highest-scoring predictions that exceed a given threshold for their normalized measured intensity** (per learning algorithm and averaged over the eight test sets)

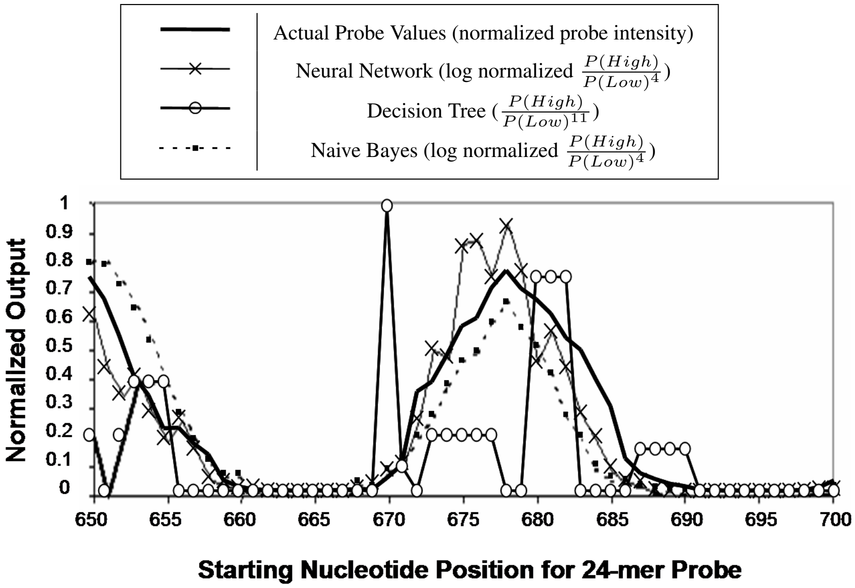


Figure 4.2 **Probe intensity and classifier output v. starting nucleotide for a typical DNA segment**

are substantially more accurate than simply using predicted melting point, at least according to our metric.

Figure 4.2 presents a visualization of the predicted intensities by the learning algorithms for a typical gene region. We generated these curves by manipulating the ratios used to determine the best probes reported in Figure 4.1. It is important to note that this figure is for visualization purposes only. The functions used to generate Figure 4.2's curves are partially fitted to the testing data, which invalidates their use in quantitative evaluation. These curves are generated as described below. However, to better visualize the predictions, we have found it useful to manipulate the predictions.

For naive Bayes and ANNs, we use the log of the ratio $\frac{prob(label=high)}{prob(label=low)}$ to create a predicted output for each probe for a given gene. Similarly, decision trees simply use $prob(label = high)$. We then compute the squared error between these predicted values and the normalized measured intensities, across all the probes in a given gene. For visualization purposes, we next consider raising the predicted values to increasing powers from 1 to 20, and the power with the minimum squared error is chosen for use in these visualization graphs; the selected power used to generate the curves is shown in the figure legend. In our quantitative experiments we are only interested in relative predicted values for the various probes (e.g., Figure 4.1).

The naive Bayes and ANN predicted curves closely fit the measured probe intensities over high and low intensity values. The decision-tree curve does not fit the actual probe curve nearly as well as the other two algorithms over all intensity ranges. While Figure 4.2 only shows a short, 50-bp region of probes, the results are typical of what is produced over all of our eight genes.

4.4 Discussion

Machine learning appears to do a good job on this task. Our results strongly suggest that *off the shelf* machine-learning methods can greatly aid the important task of probe selection for gene-expression arrays. One limitation of this work that I have tried to address with follow-up studies is whether or not we can predict, in addition to which probes will bind their target, which probes will remain vacant when their target is not present. This follow-up work is inconclusive to this point.

Chapter 5

Direct Genomic Selection Using Custom Gene Chips

The ability to efficiently design high-quality custom gene chips has presented us with myriad technological opportunities. Here I present one such technology that I helped to explore and develop which allows the gene chip, typically considered a sample-assay tool, to be used as a sample-preparation tool as well.

5.1 Motivation

The ability to discover novel variation in a genome is a key challenge in genomics, especially in the human. Screening for known variations can be done inexpensively through a number of existing techniques. These include genotyping approaches utilizing hybridization and single base extension (Steemers et al. 2006), *padlock probes* (Antson et al. 2000) and mass spectrophotometry (Tai et al. 2006).

However, discovering, in a particular individual, in a specific genomic region, previously unknown variations from a given genomic sequence is an expensive and labor-intensive process. A process known as *amplicon sequencing* (Deffernez et al. 2005) is a typical strategy used for targeted sequencing. This method uses polymerase chain reaction (PCR) to isolate and amplify a single DNA region and then uses low-cost, high-throughput sequencing to find the exact sequence and, therefore, any variations from the reference genomic sequence. Unfortuunately, the process used to isolate and amplify the single DNA region is expensive and does not scale well to multiple regions.

Another process, known as *direct genomic selection* (Bashiardes et al. 2005), utilizes either cDNA clones, or, more recently, bacterial artificial chromosome (BAC) clones, to select genomic regions for further analysis. Two rounds of hybridization, elution and amplification of fragmented genomic DNA to a biotinylated BAC clone, enrich the region targeted by the BAC 10,000-fold. Approximately 50% of sequenced clones from the enriched fraction correspond to the targeted region.

Sequencing the enriched fraction is a cost-effective method to discover new mutations in a targeted genomic region from many samples. This strategy, however, cannot be sufficiently targeted to functional elements since it relies on large genomic clones, at least 100kb, as the affinity matrix to enrich for homologous DNA. One cannot easily target specific genomic regions at a resolution level below the BAC cloning range. The method is also labor intensive to scale to large numbers of broadly dispersed loci across the entire genome.

An alternate strategy, capable of selecting all or nearly all human exons and known regulatory regions, coupled with current high-throughput sequencing technology, could revolutionize our ability to quickly discover mutations associated with cancer and other diseases.

5.2 Our Technique

To this end, we have developed a flexible and scalable method that uses NimbleGen high-density oligonucleotide microarrays to target and capture genomic elements that can then be readily sequenced using high-throughput sequencing. As described in the previous chapter, I have worked to improve the process of microarray design. In conjunction with NimbleGen Systems, using the lessons learned from my previous work, I have designed microarrays that can be used to substantially simplify the discovery of new genomic variations at a cost that is approximately 1/100th that of amplicon sequencing and 1,000 times the resolution of standard direct genomic selection.

Using much longer probes than in a typical microarray – approximately 100 bases long – chosen using the neural network described in the previous chapter and efficient uniqueness testing which will be described in Chapter 10, I have created arrays that can capture genomic sequences with such high efficiency that the microarray is no longer simply a sensor. Very dense tilings of such probes are able to actually capture a significant amount of the genomic sequence in the sample; making it available for subsequent sequencing by existing methods.

See Figure 5.1 for a high-level illustration of this process.

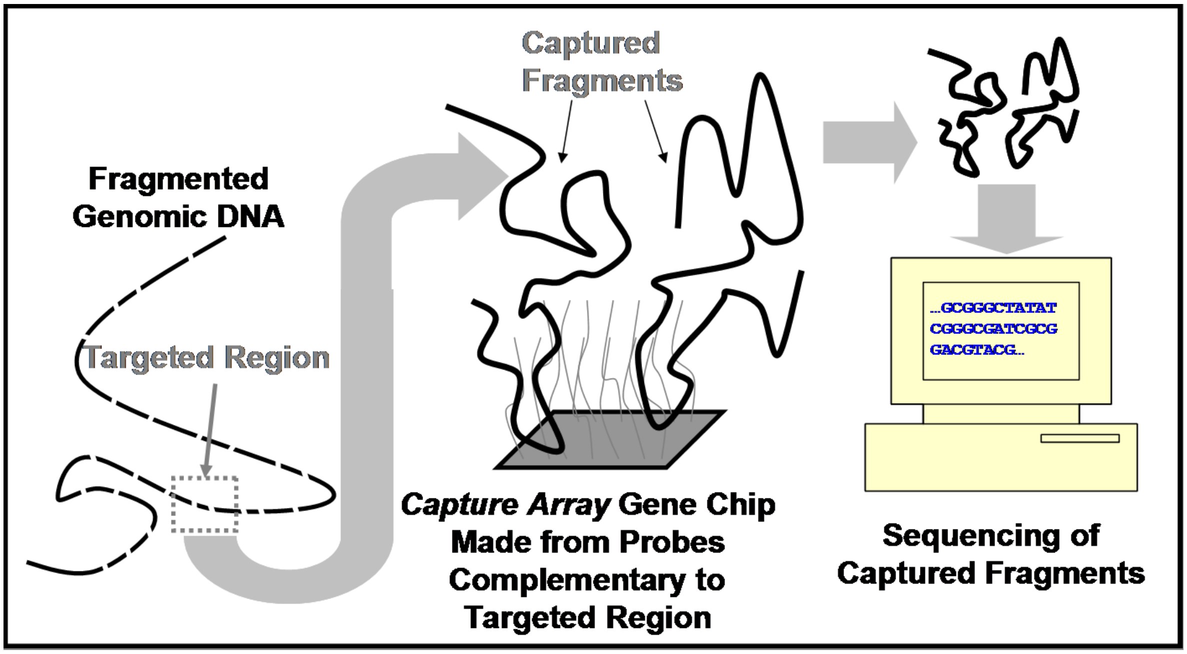


Figure 5.1 **Direct sequence capture using microarrays**

5.3 Experiments

I have designed various microarrays for this purpose. Of the six that I report on here, five of them are each designed to capture a different portion – from 200KB to 5MB in length – of the region surrounding a well-known genomic region known as the BRCA1 locus. The other one is designed to capture 6,726 genomic regions (minimum length 500 base pairs (pb), 5Mb of total sequence) that are spread widely across the genome and encompass the NHGRI Tumor Sequencing Program exon set (Collins and Barker 2007).

As described in the previous section, these microarray designs use long oligonucleotide probes to densely tile targeted genomic regions with a probe every 10pb on average. Genomic target regions were repeat masked, and individual probes were checked to be unique in the genome. To test the reproducibility of the capture system, TSP exons were captured from a Burkett Lymphoma cell line using three different microarrays. The BRCA1 locus was also captured from the same sample. Genomic DNA was whole-genome amplified, sonicated, linkers were ligated to the ends of DNA fragments, and fragments were then hybridized to capture arrays.

All of the fragments not attached to probes were then washed off of the array, leaving only the hybridized or *captured* sequences behind. Once this step is complete, these hybridized sequences are eluted from the array as well, and sequenced by a new high-throughput massively-parallel sequencing machine: 454 Life Sciences' GS FLX Instrument.

5.4 Results

Following in-silico removal of the linker sequence, I use BLAST (Altshul et al. 1990) to compare each of the sequencing reads to the entire hg17 version of the Human Genome. I use a cutoff score of $e = 10^{-48}$, tuned to maximize the number of unique hits. The reads that do not uniquely map back to the genome (between 10 and 20% of them) are discarded. The rest are considered *captured sequences*. The captured sequences that, according to the original BLAST comparison, map uniquely back to regions within the target regions are considered *sequencing hits*. The sequencing hits are reported as the *sequence coverage* of the target regions.

DNA sequencing of each of the three replicate *exonic* capture products on the 454 FLX instrument generated 63 Mb, 115 Mb, and 93 Mb of total sequence. BLAST analysis showed that 91%, 89%, and 91% of reads, respectively, mapped back uniquely to the genome, 75%, 65%, and 77% were from targeted regions and 96%, 93%, and 95% of target sequences contained at least one sequence read (Table 5.1). This represents an average enrichment of 432-fold. Figure 5.4 illustrates a detail of the read mapping for chromosome 12 from the three samples. The median per-base coverage for each sample was 5, 7 and 7-fold coverage, respectively.

There is considerable interest in the analysis of large contiguous genomic regions. Using the same DNA sample, NimbleGen tested capture microarrays targeting segments from 200kb - 5Mb surrounding the the human BRCA1 gene. As shown in Table 5.2 all capture targets performed

Table 5.1 **Series of 3 replicated capture chips, employed for direct selection of DNA**, from human genomic DNA 6,726 'Exonic' regions totaling 5Mb of genomic sequence

Fold Enrichment	**FLX -Yield (Mb)**	**Percentage of Reads Mapped Uniquely to the Genome**	**Percentage of Total Reads That Mapped to Selection Targets**	**Median Fold Coverage for Target Regions**
318	63	91%	75%	5
399	115	89%	65%	7
418	93	91%	76%	7

well, with up to 140 Mb of raw sequence generated in a single sequencing machine run, generating approximately 18-fold coverage, from a 5 Mb capture region. It is interesting to note that the percentage of reads that map to the target sequence increased with the size of the target region. This efficiency is captured visually in Figure 5.2 with a closeup view in Figure 5.3.

5.5 Discussion and Current Work

These data illustrate the power of microarray-based, direct-selection methods for enrichment of targeted sequences. In addition to the specificity of the assay, the high yields of the downstream DNA sequencing steps are consistently superior to the routine average performance using non-captured DNA sources. This is attributed to the capture-enrichment process providing a useful purification of unique sequences away from repeats and other impurities that can confound the first emulsion PCR step of the 454 sequencing process. The ease and scalability of the approach show that the method can be adapted for larger fractions of the genome and for analysis of many samples. Current efforts aim to produce a whole human exon capture array and sequence assay.

Though the probes were made using information gained from my study of probe quality from the previous chapter, I am in the process of performing a similar experiment in the context of sequence capture. The longer probes and different goal of this process could have a strong influence on the type of probe that is effective at this job.

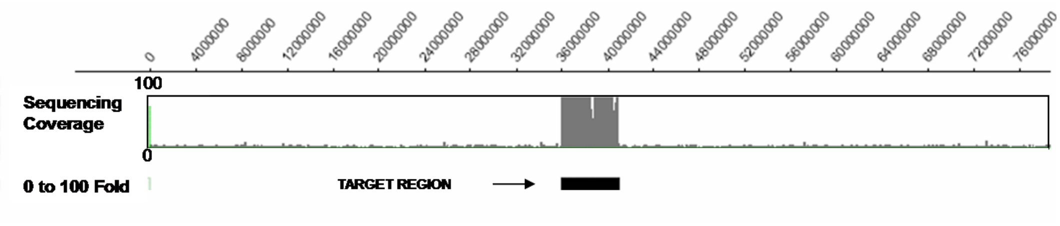
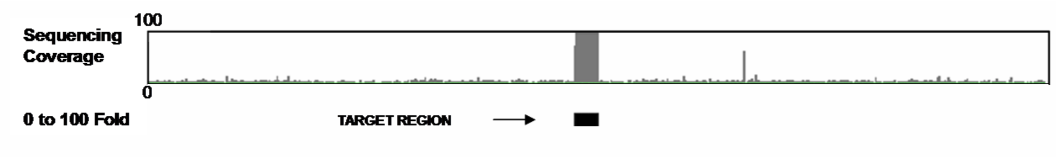
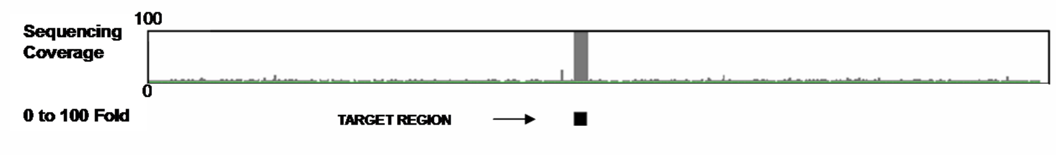
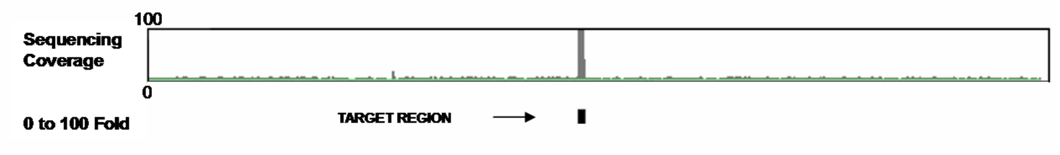
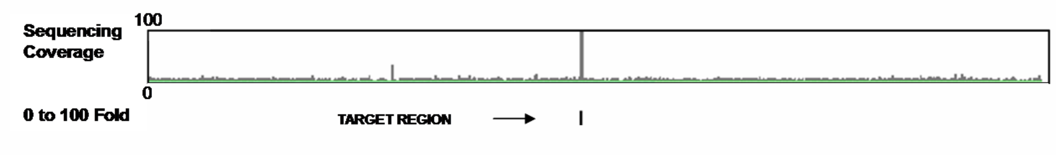
Target Length	Base Position on Chromosome 17
5 Megabases	
2 Megabases	
1 Megabase	
500 Kilobases	
200 Kilobases	

Figure 5.2 **BRCA1 capture results.** Black = Target, Gray = Captured sequence coverage depth.

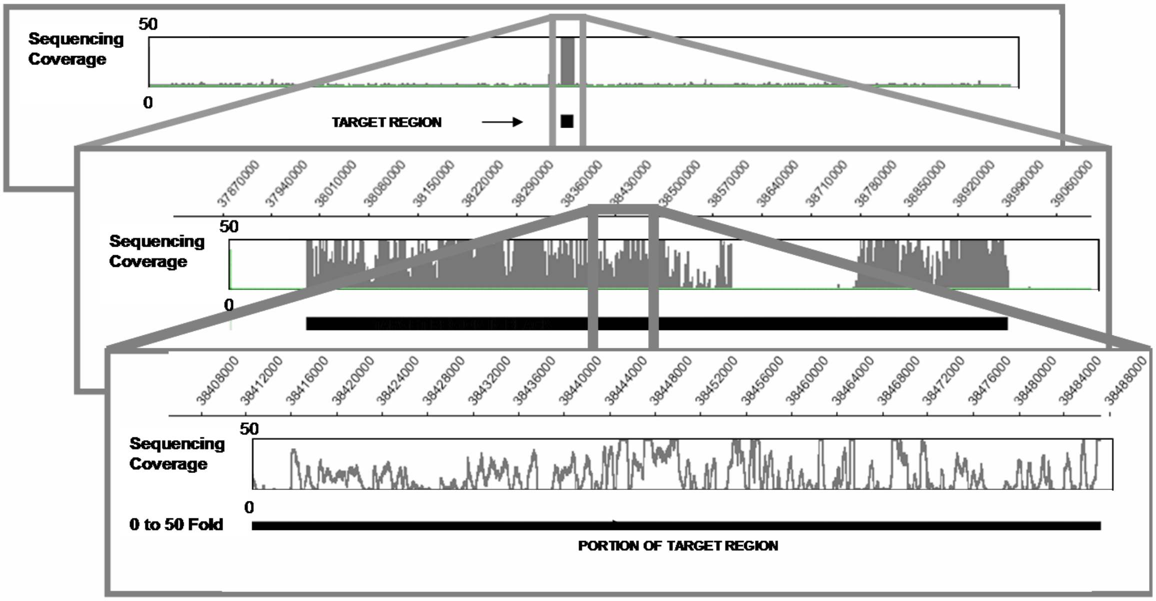


Figure 5.3 **One megabase BRCA1 region magnified to 76 kilobases**

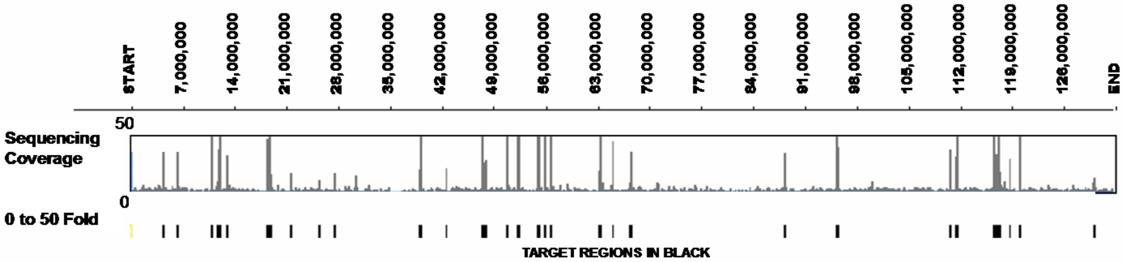


Figure 5.4 **Chromosome 12 with targets and coverage depth**

Table 5.2 **Regions of increasing size containing the human BRCA1 locus.** See text for more details. The individual DNA fragments were captured and sequenced with one run of the 454 FLX instrument.

Tiling Size (kb)	Average Selection Probe Tiling Density	FLX -Yield (Mb)	Percentage of Reads Mapped Uniquely to the Genome	Percentage of Total Reads That Mapped to Selection Targets	Median Fold Coverage for Target Regions
200	1bp	102	55%	14%	79
500	1bp	85	61%	36%	93
1,000	2bp	96	56%	35%	38
2,000	3bp	112	81%	60%	37
5,000	7bp	140	81%	64%	18

Chapter 6

Comparative Genomic Hybridization (CGH) Segmentation

Comparative Genomic Hybridization (CGH) is a method for finding copy-number differences between two samples of genomic DNA. These differences, or copy-number polymorphisms, can be either amplifications or deletions and, in humans, are often indicative of specific diseases including cancer.

6.1 Related Work

CGH involves the use of oligonucleotide microarrays. Specifically, the arrays contain tilings of either specific regions of interest, entire chromosomes, or entire genomes. Identical microarrays are exposed to different samples of genomic DNA and the resulting variations in signal intensity between the two chips can be interpreted as copy-number differences. Typically the unit of measurement is the log_2 ratio of the intensities of a given probe between the two chips.

The process of interpreting such a chip, i.e. finding the sections of the genome whose copy numbers vary between samples, is called *segmentation*. See Figure 6.1 for an example segmentation. Unfortunately, due to nonuniformity in hybridization efficiency and other anomalies, the amount of noise accompanying the signal in a typical chip is far from negligible. As a result, the process of segmenting such a chip is not straightforward. In fact, many approaches to this problem are already in use. Unfortunately, most of these algorithms were designed to work on far fewer datapoints and are, as a result, much too slow to be run on data such as that produced by Nimblegen microarrays which typically incorporate hundreds of thousands of features. Because of an inability to handle such a large number of databpoints methods such as *aCGH* (Pollack et al. 1999), and other dynamic-programming approaches (Picard et al. 2005; Huber et al. 2006) are not tested here.

Two popular algorithms that can be run on such data are *DNACopy* (Olshen et al. 2004) and *StepGram* (Lipson et al. 2006). One way that our method differs from DNACopy and StepGram is that, rather than greedily taking the best segment and potentially only reaching a local optimum, we use a dynamic-programming algorithm to efficiently arrive at a globally optimal solution in terms of squared error relative to the segment means, a standard statistical measure of segmentation quality (Lipson et al. 2006). However, unlike other dynamic programming approaches, we start by using the *t*-test statistic to identify a set of candidate breakpoints which dramatically increases the efficiency of the algorithm.

Other dynamic-programming algorithms have tried to limit the search space. Specifically, Huber et al. limit the search space by restricting the maximum segment length. However, since no such limitation on segment length exists in nature, it would be better not to have to make such a limitation.

In this chapter, I will describe our algorithm and show empirically that it outperforms both DNACopy and StepGram on synthetic and biological data.

6.2 Our Segmentation Algorithm

Our algorithm has three steps. I will describe all three parts in detail here. I will begin with a dynamic program that, given the set of log_2 intensity ratios between the probes of the two chips, the desired number of segments N, produces a segmentation: a set of $N - 1$ positions in the genome which define the extents of regions of constant copy number ratio between the samples. I will also prove that this segmentation will have the globally-maximal score with respect to the standard score function, the number of segments, N, and, if used, the list of candidate breakpoints. Next, I will describe our permutation-test-based method for choosing N. Finally I will describe the *t*-test-based method for determining candidate breakpoints and how these candidate breakpoints are used to cut the time bound from $O(n^2k)$ to $O(b^2k)$, where b is the number of candidate breakpoints and k is the maximum number of breakpoints I am willing to consider.

6.2.1 Definitions

Before I describe the algorithm, I will define some notation that I will use throughout the chapter:

1. $probe_i$ = The log_2 intensity ratio between the two chips for the i^{th} probe. In the text, I will also refer to this as the *probe value.*
2. A *vector* is a set of *n* probes numbered from 0 to $n - 1$
3. A $segment_{p,q}$ is the contiguous set of probes from the p^{th} probe to the q^{th} probe
4. A *segmentation* is contiguous set of segments.
5. $segmentation_j$ is the j^th segment of *segmentation*
6. A *k-length* segmentation is a segmentation with *k* breaks.
7. $segend_{k,t}$ is the probe number of the last probe in the k^{th} segment in segmentation *t*
8. $$score_{k,t} = \begin{bmatrix} (k = 0) : \frac{\left(\sum_{i=0}^{segend_{k,t}} probe_i\right)^2}{segend_{k,t}} \\ \\ (k > 0) : \frac{\left(\sum_{segend_{k-1,t+1}}^{segend_{k,t}} probe_i\right)^2}{segend_{k,t} - segend_{k-1,t}} \end{bmatrix}$$
9. $|segmentation|$ = the number of segments in *segmentation*
10. $||segmentation||$ = the score of *segmentation* = $\sum_{k=0}^{|segmentation|} score_{k,segmentation}$
11. $optimal_{p,q,k}$ is a segmentation with *k* preakpoints were the first segment starts at *p* and the last segment ends at *q* where $||optimal_{p,q,k}|| \geq ||t||$ for any other such segmentation *t*.

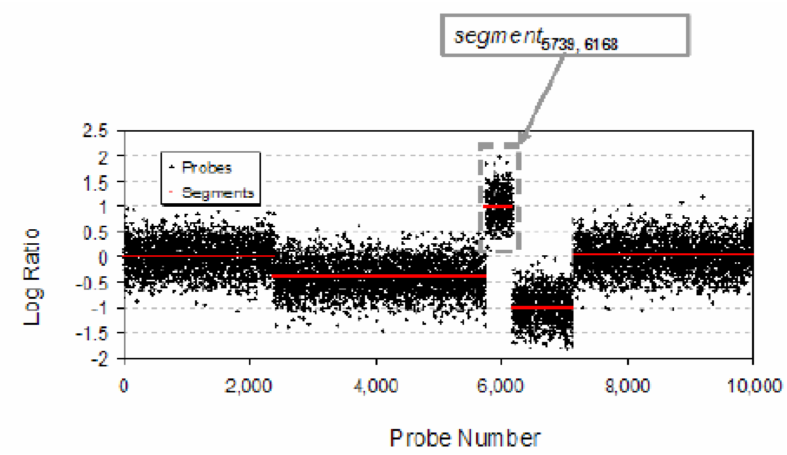


Figure 6.1 **A sample 4-length segmentation**

6.2.2 Our Task

In precise terms, our goal is as follows:

Given: *a vector of n probes and the number of segments:* k

Do: *Produce an array called* paths *where, for* $i = \{0, 1, 2, 3, ..., n\}$ *and* $j = \{0, 1, 2, 3, ..., k\}$

$$paths_{i,j} = segend_{j-1,optimal_{0,i,j}}$$

Once this is achieved, it is a simple matter to decode this array to get either $optimal_{o,n,k}$ or $optimal_{o,n,j}$ for any $j \leq k$ by using the segend values as indices into descending (*j*-labeled) rows of our paths array as is done in Table 6.1. Each entry in this *paths* array can be considered a pointer to the endpoint of the previous segment in the optimal *j*-length path from probe zero to probe *i*.

As will be described in the following subsection, we fill the *paths* array by the following recurrence relation on it and the *scores* array which eventually holds $||optimal_{0,i,j}||$ for all $i = \{0, 1, 2, 3, ..., n\}$ and $j = \{0, 1, 2, 3, ..., k\}$:

$$paths_{i,j} = \arg\max_{p}(scores_{p,j-1} + score_{(i-p),i})$$

$$scores_{i,j} = scores_{(paths_{i,j}),j-1} + score_{(i-paths_{i,j}),i}$$

exhaustively trying all values for p at each step. In Subsection 6.1.5 I describe a method to greatly reduce the number of values for p that need to be tried.

6.2.3 The Dynamic Program

To fill the paths array for our vector of n probes, we first compute the scores of all n of the zero-length (one-segment) segmentations that start at $probe_0$. These represent the scores of the segmentations in the $j = 0$ row of our paths array: $paths_{i,0}$ for i = 0 to n. These are $||optimal_{0,i,0}||$ for i = 0 to n. We use these to compute the $j = 1$ row. For each m from 1 to n, we find, by exhaustive search, the p that maximizes:

$$||optimal_{0,p,1}|| + \frac{\left(\sum_{i=p}^{m} probe_i\right)^2}{m-p}$$

This p will be the final break in $optimal_{0,m,1}$, otherwise known as $segend_{1,optimal_{0,m,1}}$. So we can set $paths_{m,1}$ to p for each m from 1 to n. In a similar manner, we can now compute the scores of all of the optimal j-length segmentations that start at probe 0 where j = 1,2,3, ..., k. Each j-length segmentation is computed from the scores of the $(j-1)$-length segmentations by exhaustively finding the p that maximizes:

$$||optimal_{j-1,p,j}|| + \frac{\left(\sum_{i=p}^{m} probe_i\right)^2}{m-p}$$

This p will be the final break in $optimal_{0,m,j}$ and will be placed into the paths array as $paths_{m,j}$. This is illustrated in Table 6.2.

6.2.3.1 Derivation of the Scoring Function

SVR is meant to define the fraction of the total variance in the sample that is within segments as opposed to the variance among segments. It is defined in standard terms from the Analysis of Variance (ANOVA) (Lindman 1974):

$$SVR = SSw/SSt$$

Where SSt = the total sum of squares of the differences between each of the probe values and the mean probe value in the entire vector; k = the number of segments; n_j is the number of probes in

Table 6.1 **Pseudocode for decoding the *paths* array**

```
Algorithm 6.2.1: DECODESEGMENTATION($numSegments, paths$)

Decode and return the resulting segmentation.

$nextPath \leftarrow numSegments - 1$
for $segmentNum \leftarrow numSegments - 1$ downto $0$
    $curSegmentEnd \leftarrow nextPath$
    $nextPath \leftarrow pathsnextPath, segmentNum$
    if $(segmentNum = 0)$
        then $curSegmentStart \leftarrow 0$
        else $curSegmentStart \leftarrow nextPath + 1$
    $segmentation_{segmentNum} \leftarrow segment_{curSegmentStart,curSegmentEnd}$
return $(segmentation)$
```

Table 6.2 **Pseudocode for the dynamic program without candidate breakpoints,** where n is the total number of probes and k is the maximum number of segments we are willing to consider

```
Algorithm 6.2.2: SEGMNT-ALLBREAKPOINTS($n, probeVector, k$)

First, compute the scores of all of the 0-length segmentations that start at probe 0

$curTotal \leftarrow 0$
for $m \leftarrow 0$ to $n$
    $curTotal \leftarrow curTotal + probeVector_m$
    $scores_{m,0} \leftarrow \frac{curTotal^2}{m}$

Now, compute the scores of all of the optimal j-length segmentations that start at probe 0 where j = 1,2,3, ..., k and put them into the paths array.

for $j \leftarrow 1$ to $k$
    Each j-length segmentation is computed from the scores of the (j-1)-length segmentations.
    for $m \leftarrow (n-1)$ downto $j$
        $curTotal \leftarrow 0; bestScore \leftarrow scores_{m,j-1}$
        Find the p that maximizes:
            $$scores_{p,j-1} + \frac{(\sum probe_i)^2}{m-p}$$
        . This p will be the final break in $scores_{i,j}$
        for $p \leftarrow m$ downto $j$
            $curTotal \leftarrow curTotal + probeVector_p$
            $curScore \leftarrow scores_{p,j-1} + \frac{curTotal^2}{m-p}$
            if $(curScore < bestScore)$
                $bestScore \leftarrow curScore$
                $bestIndex \leftarrow p$
        $paths_{m,j} \leftarrow bestIndex; optimal_{o,m,j} \leftarrow bestScore$
return $(paths)$
```

the j^{th} segment; $probe_{j,i}$ is the i^{th} probe of the j^{th} segment; $\overline{probe_{j,*}}$ is the average probe value in the j^{th} segment; and $\overline{probe_{*,*}}$ is the average probe value over all probes:

$$SSt = \sum_{j=1}^{k}\sum_{i=1}^{n_j}(probe_{j,i} - \overline{probe_{*,*}})^2$$

and SSw = the total sum of squares of the differences between each of the probe values and the mean probe value within that probe's segment:

$$SSw = \sum_{j=1}^{k}\sum_{i=1}^{n_j}(probe_{j,i} - \overline{probe_{j,*}})^2$$

Since SSt is constant with respect to a given vector, minimizing SSw also minimizes SVR.

$$\begin{aligned}
&SSw = \textstyle\sum_{j=1}^{k}\sum_{i=1}^{n_j}\left(probe_{j,i} - \overline{probe_j}\right)^2 \\
&= \textstyle\sum_{j=1}^{k}\sum_{i=1}^{n_j}\left(probe_{j,i}^2\right) + \sum_{j=1}^{k}\sum_{i=1}^{n_j}\left(\overline{probe_{j,i}^2} - 2\overline{probe_{j,*}}probe_{j,i}\right) \\
&= \textstyle\sum_{j=1}^{k}\sum_{i=1}^{n_j}\left(probe_{j,i}^2\right) + \sum_{j=1}^{k}\overline{probe_{j,*}}\left(\left(\sum_{i=1}^{n_j}\left(\overline{probe_{j,*}}\right)\right) - 2n_j\overline{probe_{j,*}}\right) \\
&= \textstyle\sum_{j=1}^{k}\sum_{i=1}^{n_j}\left(probe_{j,i}^2\right) + \sum_{j=1}^{k}\left(-n_j\overline{probe_{j,*}}^2\right) \\
&= \textstyle\sum_{j=1}^{k}\sum_{i=1}^{n_j}\left(probe_{j,i}^2\right) - \sum_{j=1}^{k}\left(\frac{\left(\sum_{i=1}^{n_j} probe_{j,i}\right)^2}{n_j}\right)
\end{aligned}$$

Note that $\sum_{j=1}^{k}\sum_{i=1}^{n_j}\left(probe_{j,i}^2\right)$ is the total sum of squares. This value remains constant for a given vector. So maximizing the quantity $\sum_{j=1}^{k}\left(\frac{\left(\sum_{i=1}^{n_j} probe_{j,i}\right)^2}{n_j}\right)$ with respect to the vector will minimize SSw and, therefore, SVR.

6.2.3.2 Proof by Induction That Our Algorithm Produces an Optimal Segmentation

Specifically, I intend to prove that by recursively finding the p that maximizes $||scores_{p,k-1}|| + \left(\sum_{i=p}^{m} probe_i\right)^2$ our algorithm produces $scores_{n,k}$.

For $k = 0$

We know that, for any $m = n$, the best segmentation from $probe_0$ to $probe_m$ with zero breaks, $optimal_{0,m,0}$, is the 1-segment path $segment_{0,m}$ since this is the only way to get from $probe_0$ to $probe_m$ with zero breaks.

For $k > 0$

Recall, from the above derivation, that, where k = number of segments; n_j is the number of probes in the j^{th} segment; $probe_{ij}$ is the i^{th} probe of the j^{th} segment, the score of $segmentation_{0,n,k}$ is:

$$\sum_{j=1}^{k} \left(\frac{\left(\sum_{i=1}^{n_j} probe_{j,i} \right)^2}{n_j} \right)$$

Note that the contribution of each segment to this final score is

$$\frac{\left(\sum_{i=1}^{n_j} probe_{j,i} \right)^2}{n_j}$$

In terms of some segment $segment_{p,m}$, this contribution is:

$$\frac{\left(\sum_{i=p}^{m} probe_i \right)^2}{m-p}$$

Given $optimal_{0,p,(k-1)}$ for some p where $k-1 = p = n$, we know that the best k-length segmentation from $probe_0$ to probem where $m > p$ and the last segment of the segmentation is $segment_{p,m}$ will be:

$$optimal_{0,p,(k-1)} \cup segment_{p,m}.$$

Since, by definition, there is no higher scoring length-(k-1) segmentation from $probe_0$ to $probe_p$ and I specified above that the last segment is $segment_{p,m}$. Note that this is not necessarily the best k-length segmentation from $probe_0$ to $probe_m$, just the best k-length segmentation from $probe_0$ to $probe_m$ that concludes with $segment_{p,m}$.

Given $optimal_{0,p,(k-1)}$ for all p where $k-1 = p = n$, our algorithm explicitly (by brute force) finds the p that maximizes

$$||optimal_{0,p,(k-1)}|| + \frac{\left(\sum_{i=p}^{m} probe_i \right)^2}{m-p}$$

for each m from k to n. Call this p'. We know that p' must be the last breakpoint in $optimal_{0,m,k}$. Therefore, for any given m and k,

$$optimal_{0,m,k} = optimal_{0,p',(k-1)} \cup segment_{p',m}$$

including the case where $n = m$. So,

$$optimal_{0,n,k} = optimal_{0,p',(k-1)} \cup segment_{p',n}$$

6.2.3.3 Time Complexity

The time complexity of the dynamic-programming algorithm described above is $O(n^2k)$ where n is the total number of probes and k is the number of segments. This is because, for each m, of which there are n, there are $n-m$ possible candidate values for p which are found by enumeration. This happens k times.

In section 6.1.5, I will describe our method for decreasing this time bound substantially by narrowing the list of candidate breakpoint positions.

6.2.4 Choosing the Number of Segments

In choosing the correct number of segments, we take advantage of the fact that, along with $optimal_{0,n,k}$, the $paths$ array contains $optimal_{0,n,j}$, for all $j < k$. So, we are able to set k to an arbitrarily high number representing the maximum number of segments we are willing to consider and evaluate all segmentations with no more than k segments. In the case of experiments in this chapter, k is set to 50, but using a higher number is completely feasible since the time and space complexity of the algorithm both grow linearly with regard to this parameter.

In order to evaluate each of these segmentations, we use what is known as a permutation test. This non-parametric statistical test is performed by randomly reordering the probes in the vector and trying to segment the resulting data. No matter what the order, our algorithm will return some segmentation and a score. If real segments exist in the original (nonreordered) data, the score for its segmentation should be much better than the scores for the permuted data. We perform 30 permutations per segmentation and segment all 30. The highest j for which the score of the non-reordered data is much better than that of the 30 permutations is returned as the real segmentation. The way we define much better is by assuming that the scores of the permuted data are drawn from a Gaussian distribution and estimating the mean and standard deviation of that distribution based on the results of the permutation test. We have a parameter (set to 95% for the experiments in this chapter) that defines how much of the resulting distribution needs to be lower-scoring than the segmentation of the nonreordered data in order for this score to be considered much better. This score is similarly compared to the scores for each h-length segmentation of the permutations where

$h < j$. The segmentation score from the nonreordered data needs to be much better than the scores for all of these permutations as well in order to be considered much better than permuted data. The maximum j for which the j-length segmentation of the nonreordered data is much better then the permuted data is chosen as the number of segments to return and the corresponding $optimal_{0,n,j}$ is returned as the resulting segmentation.

6.2.5 Identification of Candidate Breakpoints

In order to increase efficiency, a set of candidate breakpoints is be used. The probe values on either side of an actual breakpoint should come from measurably different distributions. So, we evaluate a genome position as to whether or not it is a candidate breakpoint by running a statistical *t*-test comparing the probe values of the p probes directly before the position to the p probes directly after the position. We run 24 *t*-tests for each position in the vector with p=2, p=3, p=4, p=5, through p=25. The minimum score (i.e. the minimum expectation that the two p-length sets of probe values would have been drawn from the same distribution) is returned as the candidate score. These scores are sorted and the lowest probability positions are returned as candidate breakpoints. For the experiments in this chapter, our default value of 1,000 candidate breakpoints are used. As shown in Table 6.3, The change to the dynamic program is that, instead of incrementing the score totals for each probe, the regions between candidate breakpoints are each summed once and the individual probe values are replaced by these sums in all further calculations.

This can dramatically increase the speed of the algorithm because the $O(n^2k)$ time bound becomes $O(b^2k)$, where b is the number of candidate breakpoints. Additionally, it does not substantially change the result. In order to test this, we ran our *t*-test-based method on a set of 900 randomly-generated synthetic chips. The synthetic data were generated to simulate actual chip data. Each has 10,000 probe values. They have between 2 and 10 underlying segments each with mean intensities ranging from -1 to 1 with a minimum difference of 0.2. Gaussian noise was added with variance between 1.0 and 10.0, but held constant for a given chip.

We measure the average distance between a real breakpoint and the nearest candidate breakpoint. The result is plotted in Figure 6.2. This distance is plotted as a function of the number of

Table 6.3 **Pseudocode for the dynamic program with candidate breakpoints,** where n is the total number of probes and k is the maximum number of segments we are willing to consider

```
Algorithm 6.2.3: SEGMNT-CANDIDATEBREAKPOINTS($n, probeVector,$
$k, numCandidates, candidateVector$)

  First, as before, compute the scores of all of the 0-length segmentations that start at probe 0. While doing this, also
  compute the totals array containing the total probe intensity between any two consecutive candidates.
  $curTotal \leftarrow 0; curCandidateIndex \leftarrow 0$
  for $m \leftarrow 0$ to $k$
    if ($n = candidates_{curCandidateIndex}$)
      $scores_{m,0} \leftarrow \frac{curTotal^2}{candidates_{curCandidateIndex}}$
      $curCandidateIndex++$
      $totals_{curCandidateIndex} \leftarrow 0$
    $totals_{curCandidateIndex} \leftarrow totals_{curCandidateIndex} + probeVector_m$
    $curTotal \leftarrow curTotal + probeVector_m$
  Now, compute the scores of all of the optimal j-length segmentations that start at probe 0 where j = 1,2,3, ..., k and put
  them into the paths array.
  for $j \leftarrow 1$ to $k$
    Each j-length segmentation is computed from the scores of the (j-1)-length segmentations.
    for $m \leftarrow (n-1)$ downto $j$
      $curLength \leftarrow 0; curTotal \leftarrow 0; bestScore \leftarrow scores_{m,j-1}$
      Find the p that maximizes: $scores_{p,j-1} + \frac{(\sum probe_i)^2}{m-p}$. This p will be the final break in $scores_{i,j}$
      for $p \leftarrow m$ downto $j$
        $curCandidate \leftarrow candidates_p; curScore \leftarrow scores_{p,j-1} + \frac{curTotal^2}{curLength}$
        $curLength \leftarrow curLength + candidates_{p+1} - candidates_p$
        $curTotal \leftarrow curTotal + totals_p$
        if ($curScore < bestScore$)
          $bestScore \leftarrow curScore$
          $bestIndex \leftarrow p$
      $paths_{m,j} \leftarrow bestIndex; optimal_{o,curCandidate,j} \leftarrow bestScore$
  return ($paths$)
```

candidate breakpoints. At approximately 500 candidate breakpoints, virtually every real breakpoint is included in the set of candidate breakpoints.

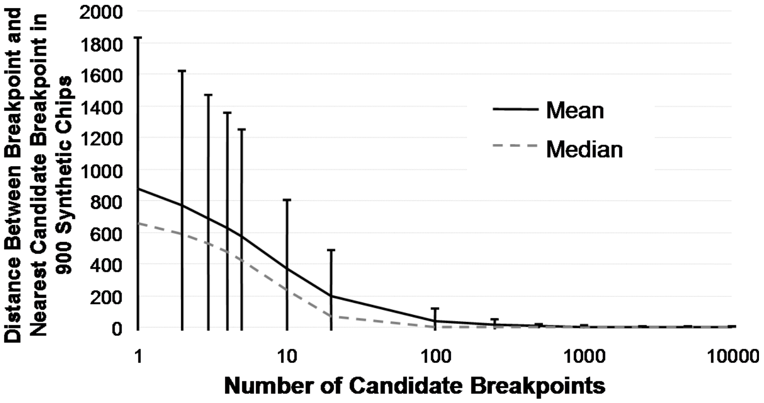


Figure 6.2 **Sufficient candidate breakpoints eliminate the need for an exhaustive search**

6.2.6 Adding Constraints

In order to further filter noise and to incorporate background information, if available, it is often useful to restrict the space of allowable segmentations in terms of either minimum segment length, minimum difference between the average probe values of two adjacent segments, or both. Other algorithms tend to approach this problem via a post-processing, or pruning, step where the resulting segmentation is modified through a series of heuristics that successively join or remove segments or breakpoints. In our view it makes more sense to optimize around these constraints in the original algorithm. This is done by constraining the choice of p when minimizing:

In other words, simply skip over any p that would produce a final segment the violates the constraint. In order to enforce a minimum segment length, p is constrained such that $(m - p)$ is the desired minimum segment length. In order to enforce a minimum average probe value difference, p is constrained such that the difference between the average probe value of $segment_{m,p}$ and that of the last segment of $optimal_{0,p,k-1}$ is the desired minimum probe value difference.

It is important to note, however, that, a k-length segmentation conforming to an arbitrary set of constraints may not exist. In this case, we substitute the optimal k-1 length segmentation. Note also that, though constraining the minimum segment length does not invalidate our proof of optimality, constraining the minimum segment intensity difference does. This is because we can no longer assert that the best k-length segmentation from $probe_0$ to probem where $m > p$ and the last segment of the segmentation is $segment_{p,m}$ will be

$$optimal_{0,p,(k-1)} + segment_{p,m}.$$

This is because $optimal_{0,p,(k-1)}$ + $segment_{p,m}$ may not be a legal segmentation with regard to the minimum segment intensity difference. There may be some $segmentaton_{0,p,(k-1)}$ to which adding $segment_{p,m}$ would have produced the real $optimal_{0,p,k}$. More generally, it violates dynamic programming's optimal substructure assumption (Cormen et al. 2001). In practice, however, when reasonable values are used, both constraints tend to produce good results.

6.3 Results

We empirically evaluate this algorithm against two well-known and algorithms known to have good performance: DNACopy (Olshen et al. 2004) and StepGram (Lipson et al. 2006). We use 900 chips of synthetic data whose exact underlying segments are known and 96 chips of real experimental data whose underlying segments have been verified by other biological methods.

The synthetic data were generated to simulate actual chip data. They were randomly generated and each have 10,000 probe values. They have between 2 and 10 segments each at intensities ranging from -1 to 1 with a minimum difference of 0.2. Gaussian noise was added with variance between 1.0 and 10.0, but held constant for a given chip. We segment each of these chips with segMNT, DNACopy and StepGram and measure the average error as defined by the average difference between the mean of the called segment containing a given probe and the mean of the underlying segment containing that probe before the introduction of the Gaussian noise. Figure 6.3 plots this average error for each of the algorithms as a function of the magnitude of the noise. Each point represents 90 segmentations whose average error has been averaged together. Though performance degrades as the noise level increase, segMNT consistently outperforms the other two algorithms.

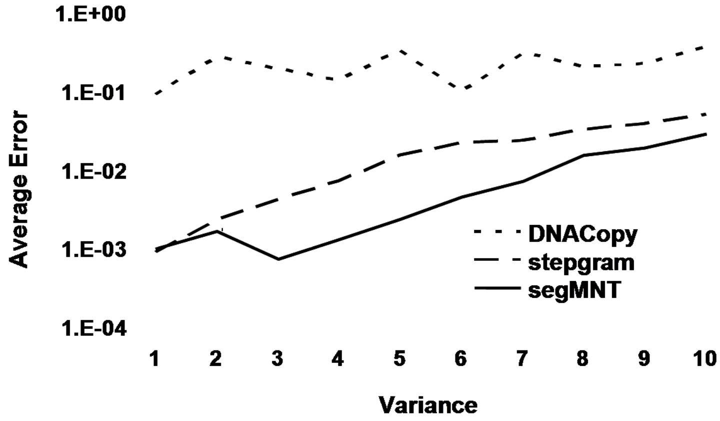


Figure 6.3 **SegMNT, StepGram and DNAcopy average error on synthetic data**

The biological data that we used were 96 chips representing 6 samples of human DNA, each measured by an 8-chip set representing the entire human genome. Each of these chips has 380,000 probes. Because the exact underlying correct segmentation is not known we used the Database of Genomic Variants (DGV) (Iafrate et al. 2004) as our source of ground truth. Unfortunately, this is not a straightforward task since any individual's DNA will only have a fraction of the possible genomic variants. Furthermore the DGV probably only contains a fraction of the possible sites of human genetic variation.

However, we still expect there to be some overlap between the variations in a given sample's DNA and the variations listed in the GDV. So, we use a concept from information retrieval, the f-measure (Mitchell 1997), to express the agreement breakpoints found by a particular segmentation and the set of known human breakpoints. We segmented all 96 chips with all three algorithms and calculated the f-measure of identified breakpoints compared to known breakpoints in the DGV.

Figure 6.4(a) is a scatter plot of the f-measure on the 96 biological chips between segMNT and StepGram. Each point represents one of the 96 chips. If the two algorithms were identical, all of the points would appear on the diagonal line. Points above the line represent chips on which segMNT does a better job than StepGram. Points below the line represent chips on which StepGram does a better job than segMNT. 75% of the points are above the line.

Likewise Figure 6.4(b) is a scatter plot of the f-measure on the 96 biological chips between segMNT and DNACopy. Points above the line represent chips on which segMNT does a better job than DNACopy. Points below the line represent chips on which DNACopy does a better job than segMNT. In this case, 80% of the points are above the line.

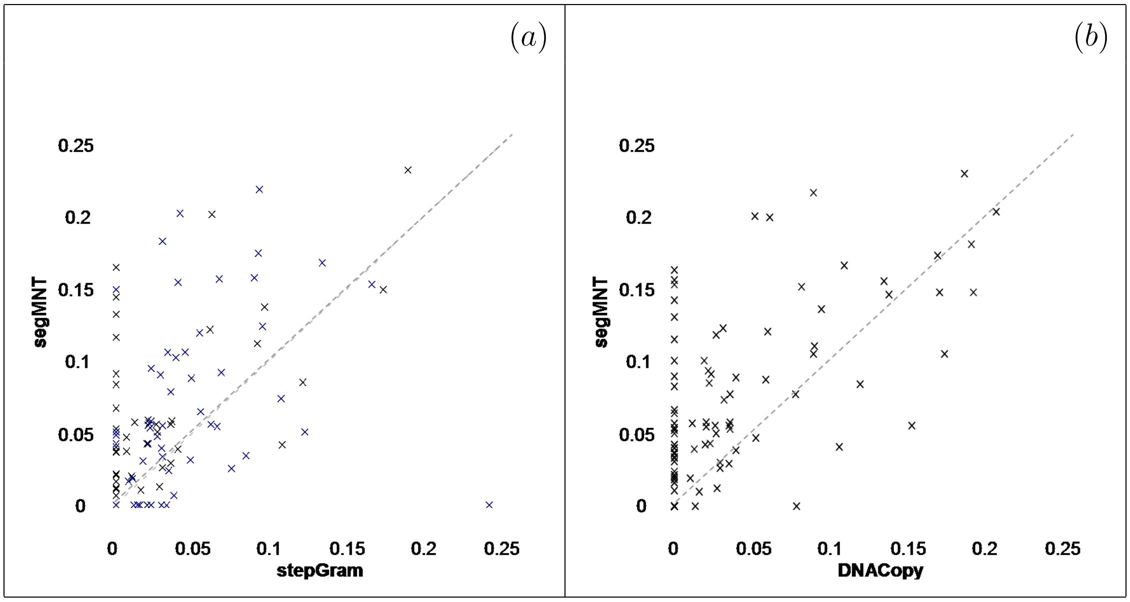


Figure 6.4 **F-measure comparison of segmentations with the Database of Genomic Variation**

6.4 Discussion

By using dynamic programming along with the *t*-test to confine the search space, we have created an algorithm that is efficient enough to run on very large datasets. I have shown empirically that this method is a very good method for the interpretation of CGH experiments. Our experiments also show that the default value of 1,000 candidate breakpoints does a good job of segmenting the very large real data sets to which we have access as well as the smaller synthetic data sets that we have used. The reason for this is probably that 1,000 is still orders of magnitude greater than the number of real breakpoints in a typical sample.

Furthermore, it can be tuned, by adjusting the number of candidate breakpoints, to produce results arbitrarily close to the globally optimal segmentation with regard to total variance from the segment means for a given number of segments. I have also shown that our permutation-test-based strategy can do a good job of determining the correct number of segments.

Chapter 7

SNP Identification

The task I address in this chapter is to identify SNPs (Single Nucleotide Polymorphisms) in the context of oligonucleotide-microarray-based DNA resequencing (Nuwaysir et al. 2002; Singh-Gasson et al. 1999). This type of resequencing consists of fully tiling (making probes corresponding to every 29-mer in) the reference sequence of an organism's DNA over a region of interest. For each of these probes, another three mismatch probes are generated. Each of these has a different base in its center position. For example, if the organism's reference DNA includes the sequence:

3'-CTGACATGCAGCTATGCATGCATGAA-5'

the corresponding reference probe will be its reverse complement and, therefore, be the sequence:

5'-GACTGTACGTCGATACGTACGTACTT-3'

and the corresponding mismatch probes will be the sequences:

5'-GACTGTACGTCGAAACGTACGTACTT-3'
5'-GACTGTACGTCGACACGTACGTACTT-3'
5'-GACTGTACGTCGAGACGTACGTACTT-3'

I call a group of probes such as this that represent all possible SNPs at a given position a position-group or, for short, a *p-group*. One can summarize the task of interpreting such a resequencing chip as follows:

Given: *The data from a single resequencing chip, representing either the complete genome of an organism, or some region or regions of interest in such a genome.*

Do: *Identify, from among the positions at which the sample sequence seems to differ from the reference sequence, which of these positions are likely to be real SNPs rather than noise and return these positions along with a confidence measure for each.*

7.1 Approach

After the chip has been exposed to the sample, each of the probes will have a resulting intensity. I also call each p-group's set of four such intensities an *example* (I use this term taken from machine learning because my solution is built upon a technique from machine learning). For most of these examples, the highest of the four intensities will be the reference probe, i.e., the probe with no mismatch base. I call examples for which this is the case *conformers* (Table 7.2, provides an illustration) since they conform with what I expect, given the reference sequence. When one of the mismatch probes has the highest intensity, I call the p-group a *non-conformer*.

Some of these non-conformers reflect actual SNPs in the DNA of the organism. However, most of them are the results of hybridization failures or other types of noise and do not represent an actual SNP in the sample. Note that, though the task of separating conformers from non-conformers is a trivial data-processing step, separation of the non-conformers that truly are SNPs from the non-conformers that arise from noise in the data is not.

I posit that one can perform the task of accurately separating the non-conformers that truly are SNPs from the noisy non-conformers by applying what is called the *nearest-neighbor method* (Mitchell 1997). In this method one plots examples in an *N*-dimensional space, where the dimensions are features of the examples. In order to interpret an example in this feature space, one looks at the K examples nearest to it in this space and uses their classifications to interpret the example in question.

In the traditional manner for applying the nearest-neighbors method (which I do not follow in this work), one would manually label a ”training set” of p-groups as being either true SNPs - non-conformers that arise from a one-base difference between the sample sequence and the reference sequence - or false SNPs, non-conformers that arise from noise in the microarray experiment. The nearest-neighbors algorithm would then use these labeled examples when it needed to categorize future non-conformers.

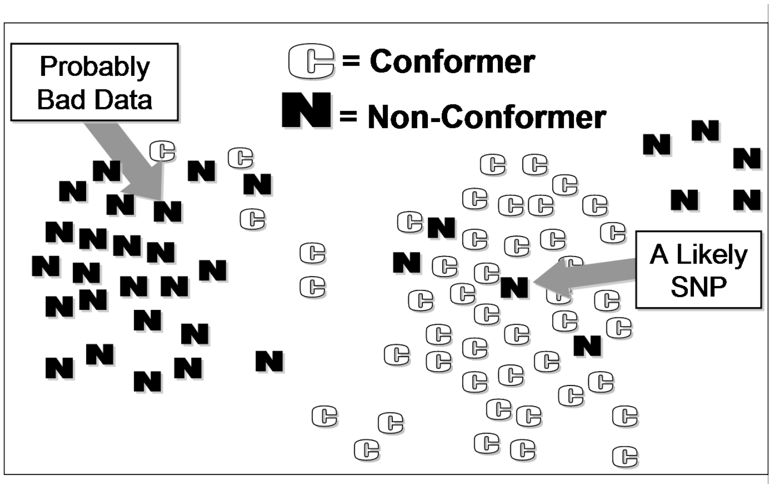
In this case, however, this approach would not be feasible. It would require that someone laboriously collect each of these training examples. Worse still, whenever the chip chemistry or any other laboratory condition changed, one would need to collect an entirely new set of training examples. This is because the underlying process that generated the noise would probably have changed.

Instead I apply the nearest-neighbor approach without needing human-labeled examples. The key idea is that examples involving bad microarray hybridizations will tend to group together in different portions of feature space than examples from good hybridizations. Once I have separated "noisy" examples from good examples, I can identify SNPs by simply finding examples where the highest-scoring base is not the base in the reference sequence. This is possible because of the nature of my particular task. Specifically, I rely on the following three assumptions, which have held true in all of the data I have looked at so far, including the data used in the experimental section of this chapter:

1) Examples resulting from proper probe-target hybridizations will be much nearer to each other in feature space than to examples resulting from hybridization failures.
2) The majority of non-conformers are due to noise in the data rather than SNPs. Hence, I can safely ignore, when looking for SNPs, those areas in feature space dense with non-conformers.
3) SNPs are relatively rare. Hence SNPs involved in successful hybrizations will fall in regions of feature space that are surrounded by conformers.

Given these assumptions it follows that, as illustrated in Figure 7.1, an area of feature space dense with conforming examples is unlikely to contain probes that are hybridization failures. In fact, the likelihood that any given example in an area is a hybridization error can be roughly estimated by the density of non-conformers in that area. By performing this estimation for each of the non-conformers, I find an approximate likelihood that it is the result of a hybridization error. Those non-conformers with low likelihood of being hybridization errors, and conversely high likelihood of being a correct reflection of the underlying sequence, I can predict to be SNPs.

Note that, though my approach makes use of labeled examples, my approach does not require a human to label any examples as being SNPs or not. Instead, my possible labels are conformer



FEATURE SPACE

Figure 7.1 **Interpreting conformers by looking at their neighbors in feature space**

and non-conformer, a distinction easily computed automatically. If a non-conformer is mainly surrounded by conformers in feature space, I classify it as a SNP since it looks like a successful hybridization and the non-conforming values are likely to have arisen due to a nucleotide change from the reference sequence (i.e., a SNP).

7.2 My Algorithm for Finding SNPs

Table 7.1 contains my algorithm for SNP-detection in microarrays. This *K*-nearest-neighbor algorithm involves plotting each example in feature space and then, for each of these examples, finding the K other examples nearest to it in this feature space. The categories of these K neighbors determine the prediction. If greater than some threshold of these neighbors are conformers, I infer that the example is not the result of a failed hybridization. I thus classify a non-conformer as a SNP. Should an insufficient number of neighbors be conformers, I view the example as being noisy. The fraction of conformers among the K neighbors can further be used as a measure of confidence in the prediction. (One could use different settings for K and $threshold$ for conformers than for non-conformers, but I have not done so.)

Table 7.1 **My algorithm for identifying SNPs**

```
Algorithm 7.2.1: SNPFINDER($K, threshold, dataset$)

In my experiments, except where otherwise noted, K = 100, threshold = 0.97

for each $example \in dataset$
    Find the K other members of dataset closest to example in feature space
    These are example's K nearest neighbors
    $P \leftarrow 0$
    for each $otherExample \in$ these K nearest neighbors
        if otherExample is a conformer
          then $P++$
    if $\frac{P}{K} > threshold$
        if the category of example is a conformer
          then classify example as a non-SNP
          else classify example as a candidate SNP
      else classify example as a non-call (i.e. possibly bad data)
return $(classifications)$
```

The appropriate value for K and threshold and appropriate definitions of nearness and feature space vary between learning tasks. In this case, my feature space - see Table 7.2 - is the five-dimensional space of examples, where four of the dimensions correspond to the intensities of the four probes in the example and the fifth dimension is the identity of the base in the reference sequence. I define nearness between two probes to be infinite in cases where the two examples differ in the fifth dimension. Otherwise, it is defined as:

$$nearness(example_j, example_k) = \sum_{i=1}^{4} |feature_i(example_j) - feature_i(example_k)|$$

where $example_j$ and $example_k$ are two p-groups, and $feature_i(example)$ is the intensity of the i^{th} most intense probe in $example$.

For purposes of evaluation, I compare my algorithm to a simple alternative, which I call my baseline algorithm. Table 7.3 contains this baseline algorithm, which simply compares the highest intensity probe to the second highest. If the ratio is above a threshold value, the algorithm assumes that the base represented by the highest intensity probe is the base in the sequence. If this p-group is a non-conformer, my baseline algorithm calls it a candidate SNP.

7.3 Evaluation

In order to evaluate my algorithm, I chose a useful, realistic task. One strain of the SARS virus (Ruan et al. 2003) has been completely sequenced via standard capillary sequencing. I was supplied with a different sample strain. This sample differed from the reference sequence to an unknown degree. My task was to identify candidate SNPs in this strain. My predictions would subsequently be evaluated using further capillary sequencing and various other “wet” laboratory methods (Wong et al. 2004).

Using the reference sequence, I designed a resequencing chip including both the forward and reverse strands of this virus. I then exposed this chip to the sample. After that I used my algorithm to predict the SNPs on this chip. Once these results were obtained, I combined the forward and reverse predictions for each possible SNP position by averaging the two predictions.

Table 7.2 **The features used to describe the *p*-groups**

Reference Sequence: `AGCGCTTTAAGCATATATCCATCCTAGCATACGATCTTTATACTTACATTACCCT`

Resequencing probes (reference probes are boxed)

p-group 7:	`TTTAAGCATATATCAATCCTAGCATACGA`	← Probe 7A
	`TTTAAGCATATATCCATCCTAGCATACGA`	← Probe 7C
	`TTTAAGCATATATCGATCCTAGCATACGA`	← Probe 7G
	`TTTAAGCATATATCTATCCTAGCATACGA`	← Probe 7T
p-group 8:	`TTAAGCATATATCGATCCTAGCATACGAT`	← Probe 8A
	`TTAAGCATATATCGCTCCTAGCATACGAT`	← Probe 8C
	`TTAAGCATATATCGGTCCTAGCATACGAT`	← Probe 8G
	`TTAAGCATATATCGTTCCTAGCATACGAT`	← Probe 8T
p-group 9:	`TAAGCATATATCGAACCTAGCATACGATC`	← Probe 9A
	`TAAGCATATATCGACCCTAGCATACGATC`	← Probe 9C
	`TAAGCATATATCGAGCCTAGCATACGATC`	← Probe 9G
	`TAAGCATATATCGATCCTAGCATACGATC`	← Probe 9T

Resulting Intensities (obtained by exposing the chip to the sample)

Probe	Intensity
...	...
7A	1543
7C	3354
7G	342
7T	737
8A	1456
8C	2432
8G	212
8T	334
9A	332
9C	456
9G	232
9T	2443
...	...

← The reference probe for *p*-group 7 is 7C. This is also the highest-intensity probe in this *p*-group. Hence, I call *p*-group 7 a *conformer*.

← Note that, though the reference probe from *p*-group 8 is 8A,
← the highest intensity probe from this *p*-group is 8C. I call such a *p*-group a *non-conformer*.

The Feature Set Each *p*-group produces one example. The features are the reference base and the four sorted intensities (note that the feature set contains no information about which actual probe has the highest intensity). The category of the example is either conformer or non-conformer, that is whether or not this *p*-group's highest intensity probe is the reference base.

Example	Reference Base	Intensity 1	Intensity 2	Intensity 3	Intensity 4	**Category**
...	...	...	...	...	...	**...**
7	C	3354	1543	737	342	**conformer**
8	A	2435	1456	334	212	**non-conformer**
9	T	2443	456	332	232	**conformer**
...	...	...	...	...	...	**...**

Table 7.3 **A baseline algorithm for identifying SNPs**

```
Algorithm 7.2.2: BASELINESNPFINDER($threshold, dataset$)

for each $example \in dataset$
    $maxIntensity \leftarrow$ intensity of the highest intensity base in example
    $secondIntensity \leftarrow$ intensity of the second highest intensity base in example
    $P \leftarrow 0$
    for each $otherExample \in$ these K nearest neighbors
        if otherExample is a conformer
            then $P++$
    if $\frac{maxIntensity}{secondIntensity} > threshold$

      then classify example as a non-call (i.e. possibly bad data)
      else
            if the category of example is conformer
              then classify example as a non-SNP
              else classify example as a candidate SNP
return ($classifications$)
```

7.4 Materials and Methods

A detailed description of the methods used to prepare and analyze the SARS samples has been previously published (Wong et al. 2004). The following is a brief overview.

7.4.1 Preparation and Hybridization of SARS Sample.

Total RNA is extracted from patient lung, sputum or fecal samples, or from Vero E cultured cells inoculated with SARS-CoV RNA. RNA is reverse-transcribed into double-stranded cDNA. Tissue samples are amplified using a nested-PCR strategy. For each sample, PCR-product fragments are pooled at an equimolar ratio, digested with DNase I (from Invitrogen, Carlsbad, CA) and end labeled with Biotin-N6 ddATP (Perkin Elmer, Wellesley, MA) using Terminal Deoxynucleotidyl Transferase (Promega, Madison, WI).

The arrays are synthesized as previously described (Nuwaysir et al. 2002; Singh-Gasson et al. 1999). The re-sequencing arrays are hybridized with biotinylated DNA overnight, then washed and stained with Cy3-Streptavidin conjugate (Amersham Biosciences, Piscataway, NJ). Cy3 signal is amplified by secondary labeling of the DNA with biotinylated goat anti-streptavidin (Vector Laboratories, Burlingame, CA).

7.4.2 Data Extraction and Analysis.

Microarrays are scanned at 5μm resolution using the Genepix 4000b scanner (Axon Instruments, Inc., Union City, CA). The image is interpolated and scaled up 2.5x in size using NIH Image software (http://rsb.info.nih.gov/nih-image/). Each feature on the microarray consists of 49 pixels; pixel intensities are extracted using NimbleScan Software (NimbleGen Systems, Inc. Madison, WI).

7.5 Results

Out of the 24,900 sequence positions represented by *p*-groups on this chip, 442 are nonconformers (i.e., *p*-groups where the highest-intensity probe was not from the reference sequence).

Of these 442, my algorithm identifies 36 as candidate SNPs. Subsequent laboratory experimentation by NimbleGen Systems produced 24 actual SNPs, all of which were identified by my algorithm.

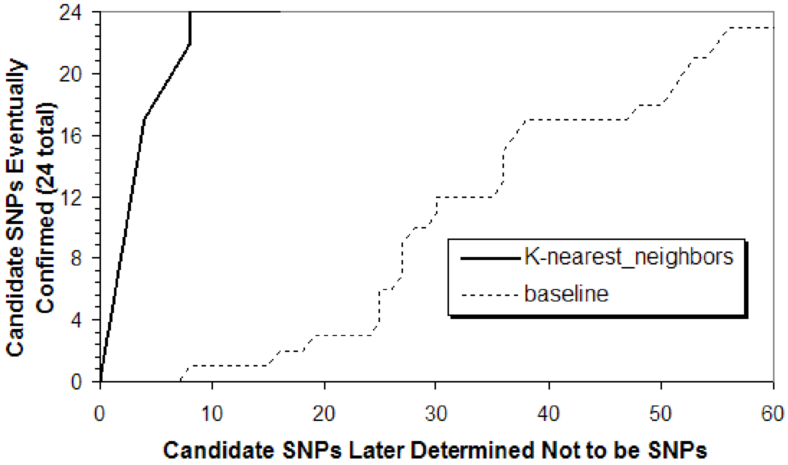


Figure 7.2 **ROC curve for SARS SNP detection**

Note, though, that in general it is possible for a conformer to truly be a SNP; however, my algorithm will not call these as SNPs, at best it will label this p-group as suspicious data. Since the SARS strain I used did not contain any "conforming" SNP's, I am unable to evaluate how well my approach does at labeling such SNPs as non-calls. Of the 24,458 conformers, my algorithm (using the same parameter settings as used for categorizing the non-conformers) only marked 3% as bad data.

In order to verify this result, five more identical SNP chips were generated and exposed to the same sample using the same values of K and $threshold$ (later in this section I discuss how I choose good values for K and $threshold$). The results varied only slightly. My algorithm found all 24 SNPs in each of the five cases. The number of false positives ranged from 6 to 13.

Figure 7.2 contains a Receiver-Operating-Characteristic (ROC) curve (Davis and Goadrich 2006) that further illustrates the performance of my algorithm, and compares it to my baseline algorithm. A ROC curve is a plot of true positives against false positives. It is typically obtained by running an algorithm at various thresholds. Recall that in the case of the K-nearest-neighbors algorithm, this threshold is the minimum percentage of neighbors that must be conformers in order for a non-conforming p-group to be classified a SNP (see Table 7.1). In my task, false positives are non-SNPs incorrectly classified as SNPs. True positives are SNPs correctly identified by my algorithm. A perfect algorithm's curve would immediately reach the upper-left corner, since that would mean that the algorithm is capable of identifying all of the true positives without producing any false positives. Though the curve for my algorithm does not quite reach this corner, note that it substantially dominates the baseline algorithm.

Based on the results of these experiments, it seems that my system is clearly superior to the baseline algorithm described above and is a reliable and efficient method for the identification of SNPs.

My algorithm is largely self-tuning, in that examples are compared to their neighbors in feature space and classifications are made according to the properties of the neighbors, as opposed to specific portions of feature space being pre-labeled as clean or noisy. However, I do have two parameters, K and $threshold$. Next, I describe some experiments that investigate the sensitivity of my algorithm to the particular settings of these parameters.

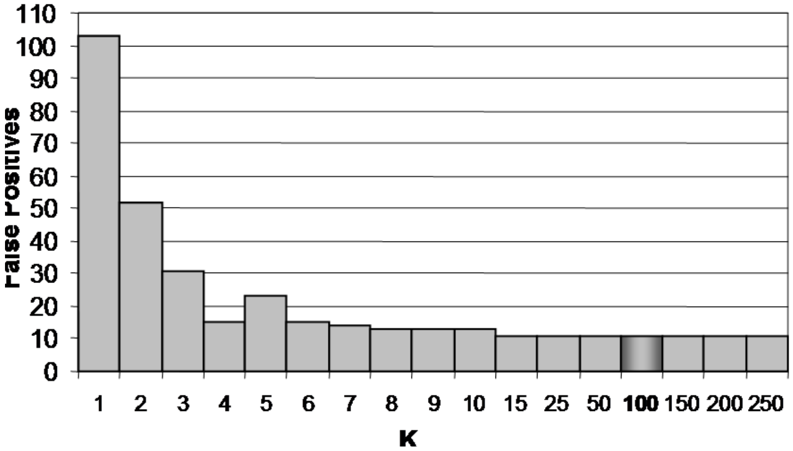


Figure 7.3 **The impact of K.** The Y-axis reports the number of false positives (noisy examples misclassified as SNPs) that result for the given value of K for the largest $threshold$ that allows my algorithm to detect all 24 true SNPs.

In order to choose an appropriate value for K, I tried various values between 1 and 250 to see how many false positives would result if one chose the largest $threshold$ that allowed my algorithm to detect all 24 of the true SNPs. The results of this experiment appear in Figure 7.3. Fortunately my approach is not overly sensitive to the particular value of K; I chose K=100 and hypothesize that this parameter setting will work well across a wide variety of organisms and strains.

Figure 7.4 presents the impact of varying $threshold$ (for K=100). It reports the number of true SNPs detected, as well as the number of false positives (non-SNPs incorrectly called SNPs). As can be seen, the algorithm's performance is not overly sensitive to the setting for $threshold$. I also anticipate that a single setting for $threshold$ (such as the 0.97 that I use) will work well across many organisms and strains, and hope that neither K nor $threshold$ need to be reset for each new dataset. Remember, however, that my approach classifies some p-groups as non-calls, namely those whose neighbors are predominantly non-conformers. The percentage of p-groups that are called (either SNP or non-SNP) is typically known as the *call rate*. If this rate is too low, the procedure is of much less use since the algorithm only interprets a small fraction of the data. In order to increase the call rate, one can lower the $threshold$ value. Using my chosen parameter settings I achieve a call rate of over 97%, while still identifying all of the SNPs in the samples I tested and misclassifying only a small number of non-SNPs.

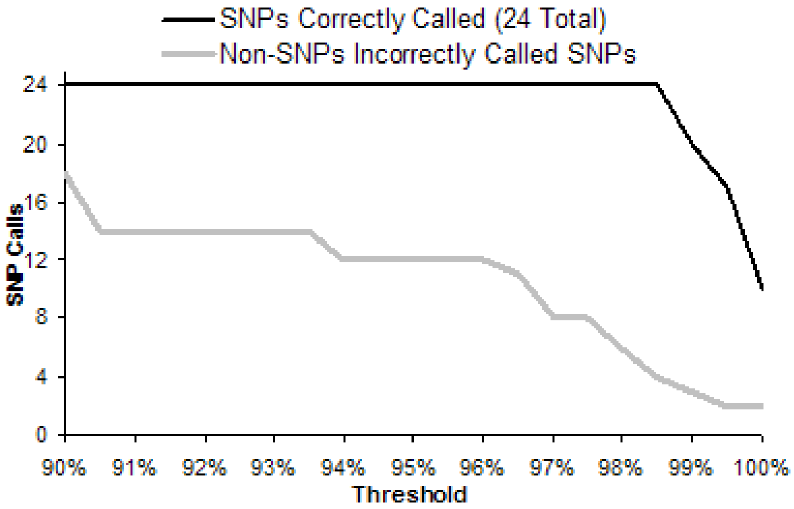


Figure 7.4 **The impact of the** $threshold$ **value.** The Y-axis reports the number of SNPs found and the number of false positives that result for the given $threshold$ with the value of K fixed at 100.

I am unable to directly compare against the haploid SNP calling accuracy of the current standard algorithm, ABACUS, from the Cutler group in conjunction with Affymetrix Corp. However, I believe my results to be comparable to those published by Cutler et al. (2001), while my approach has much less overhead due to tuning and does not require high-resolution scanning. Their published results indicate an emphasis on high-confidence SNPs, at the cost of having a low call rate. The Cutler group's reported accuracy is good. Of the 108 SNPs they predicted in the human *X* chromosome, all 108 were verified to be real. However, they report their call rate on the chip as a whole to only be approximately 80%. Though my method is currently geared more toward a high sensitivity to SNPs, I can change this by increasing my $threshold$ from 97% to 99%. My call rate drops from 97% to 81% and, though I only make 22 SNP calls at that level, only 2 of them are false positives (hence I only detect 20 of the 24 known SNPs). Of course, one should not closely compare results across species, but these numbers do at least suggest the accuracy of my algorithm is on par with that of the Cutler group.

7.6 Discussion

This work as proven to be quite useful in the field of SNP identification. Combined with the technology similar to that described in Chapter 6, this method is an efficient way to identify thousands of SNPs to date. The CGH process is used as a *filter* which can identify short regions that may contain SNPs, cutting down amount of sequence that needs to be interrogated by our SNP finding procedure. This process is used in Albert et al. (2005), Herring et al. (2006), Kane et al. (2007), and many other studies.

Chapter 8

Generalizing the SNP-Finding Method

In this chapter, I describe a possible further use for the SNP-finding algorithm of the previous chapter. Though developed for a specific purpose – to identify SNPs – it has a novel property that may be of more general use. As will be described in detail, the algorithm makes use of the distribution of examples with regard to an observed feature – *conformer* vs *non-conformer* – to infer the distribution examples with regard to an unobserved feature – *SNP*s vs *noise*. If put to use in a general setting, this could yield valuable results.

8.1 The Concept of a *Key Feature*

Consider the feature space pictured in Figure 8.1. Similar to Figure 7.1, a dense cluster of *conformers* indicates data that are likely reporting correct values, indicating that non-conformers among them are likely the result of actual SNPs. Conversely, a dense cluster of *non-conformers* indicate data that are likely noise; the non-conformers that make up this space are likely to be mostly bad data.

I am able to make this judgment because I have background information that *conformer* vs *non-conformer* is, in this sense, a *key* feature. This feature value has special meaning in examples in which its value is unexpected. It is important to note that whether or not a feature itself is predictive or unpredictive has very little bearing on whether or not it is a *key* feature. A *key* feature is an *observed feature* that represents the dimension of feature space that separates a dense relatively-easy-to-characterize cluster of data from sparser more-difficult-to-characterize data. The idea is that, as in the SNP-finding algorithm, I may be able to gain insights about the sparser data from

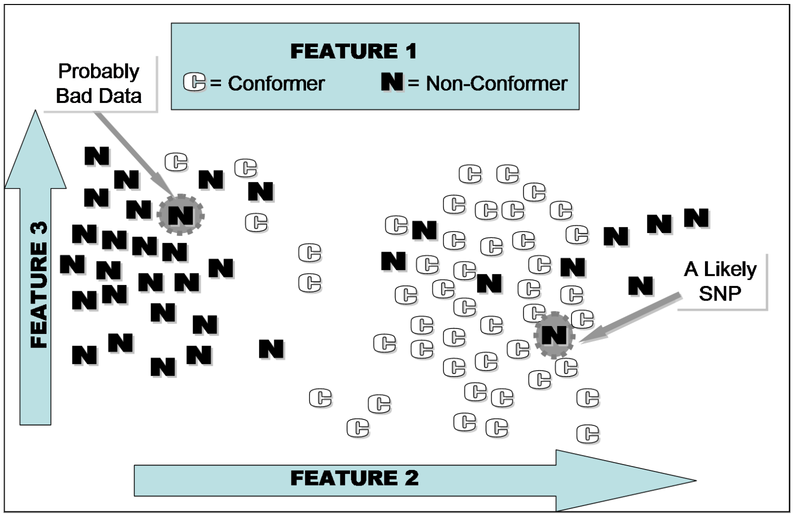


Figure 8.1 **Feature space similar to Figure 7.1**

the distribution of the denser data. One way to do this would be to simply ignore this separating *key* feature, collapsing the dense and sparse data into a single cluster. However, as is the case with the SNP data, sometimes this separating feature is, itself, of crucial importance.

Consider the situation where I have the same data as Figure 8.1, but no knowledge about *key* features. Figure 8.2 depicts this situation. The data is the same, but *conformer* vs *non-conformer*, rather than Feature 3, is displayed on the Y-axis. Feature 3 from Figure 8.1 is displayed as **H** vs **L** rather than being displayed on the Y-axis. Without knowing that *conformer* vs *non-conformer* is a *key* feature, the learner cannot use the procedure from the previous chapter. In order to use that procedure, we need to know which is the *key* feature in order to know which to attempt to predict from the others.

In addition to the fact that I do not know which, if any, feature is a key feature. I also do not know how a key feature should be interpreted. In the SNP data *conformer* vs *non-conformer* is the key feature. The way that I interpret it is that when the rest of the features strongly suggest *conformer* but the actual value is *non-conformer*, I call the example a SNP.

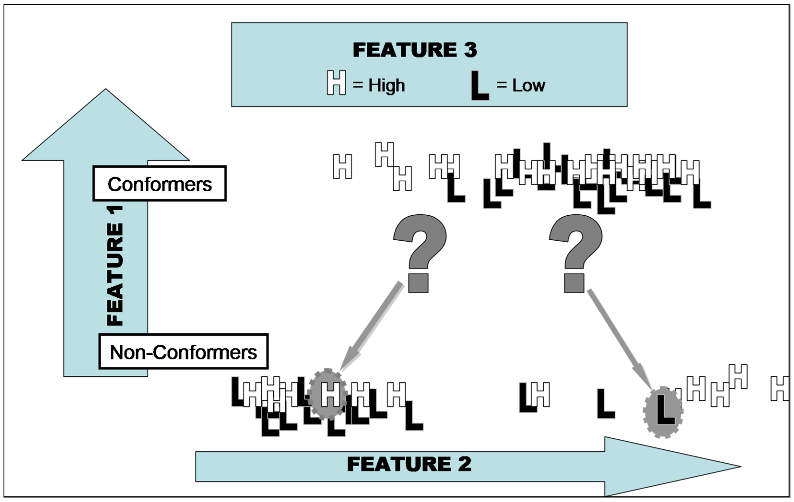


Figure 8.2 **The same data as Figure 8.1, plotted with features 1 and 3 reversed**

8.2 The *Pretraining* Algorithm

If the learner were able to identify *key* features and figure out how to use them, the principles underlying this procedure could be of general use.

So, for the general algorithm, I allow the learner to decide these things for us. Specifically, using a ten-fold cross-validation framework, I predict each feature from the others and add these predictions as additional features, as illustrated in Table 8.2. As opposed to the typical machine-learning prediction depicted in Table 8.1, my final predictions use these additional features as in Table 8.3. The pseudocode is in Table 8.4. The idea behind this is that, at prediction time, the learner has access to more of the information about whether or not a feature is a *key* feature. If the feature was wrongly predicted from the others, this information can be used in the predictive model for the category.

As described in Chapter 3, in machine learning, most datasets include *features* and *categories*. For example, a database of medical patients, designed to predict the presence or absence of a particular difficult-to-diagnose disease might have *features* that corresponding to particular diagnostic test results or attributes about the patient, and the *category* might be the ground truth about whether or not the patient actually had the disease as evidenced by some additional more-reliable diagnostic test or a future manifestation of the disease.

However, there is no reason that I could not restructure the examples so that one of the features becomes the category. For instance, say I want to predict the result of one of the diagnostic tests from the results of the others. I could simply relabel that feature the *category* and train and test on it. One existing application of this kind of relabeling is in the context of *feature selection* (Blum and Langley 1997). The theory is that, if a feature can be reliably predicted from the others, that feature must not be necessary since it carries no unique information. So, any well-predicted feature is discarded. In my algorithm, I am not concerned with predictability of features. Instead of discarding features, I use the predicted feature values as additional features themselves.

Specifically, if one considers the dataset contained in Table 8.1, where each feature is a column, with one extra column for the category, I add, for each feature one new column. To do this I do the following:

As in Panel 8.2(a):

1) Relabel *column 1* the *category* column.
2) Ignore the original category column.
3) Predict the values in column 1 from the other features.
4) Add the resulting column of predictions as a new feature column.
5) Replace column 1's original label.

As in Panels 8.2(b) and 8.2(c):

6) Repeat steps 1 through 5 for each of columns 2 through *N* where *N* is the number of features.

As in Table 8.3:

7) Use the *2N* feature columns (*N* original features + *N* columns of predictions) to predict the original category column.
8) Report the results of this final prediction.

For a more detailed pseudocode description see Table 8.4

It is important to note that, in order to participate in the *feature prediction* phase of this algorithm – that is, the phase where the *feature prediction* columns are added – an example does not need to be labeled with a category. All that is required are the feature values. So, it can make use of unlabeled data as well as labeled data. This is important in the common case that labeled data are scarce. In this way, the pretraining algorithm can be considered a semi-supervised learning algorithm (e.g. Chapelle et al. 2006).

8.3 Experiments

Here I present experiments to show that pretraining is able to make use of unlabeled data. These experiments are done using a support vector machine (SVM) (Scholkopf et al. 1999) in both feature prediction and final category prediction. This is because the SVM is a very successful standard machine-learning algorithm and is also known to perform well on high-dimensional data.

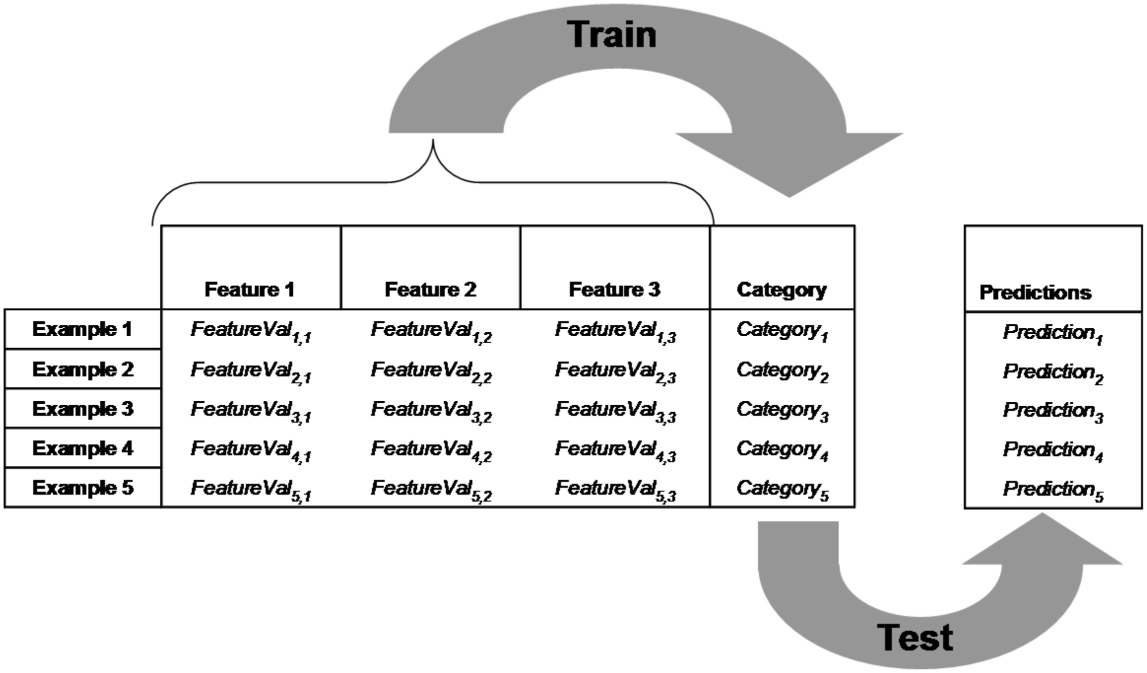
8.3.1 Unbalanced Data

In order to create a region of feature space dense with negative examples, I need the data to be unbalanced in order to have a few members of one class in a region of feature space dense with the other. In order to achieve this, we, to the extent possible given the sizes of the datasets, undersample the positive class. One exception to this is the 3-bit-parity dataset, in which I instead oversampled the negative class while being careful not to allow the same negative example to appear in the training and test sets for the same experiment. In the typical real-world dataset, this would probably not be required since very few real-world learning problems involve balanced data. In fact, the use of machine learning algorithms often requires that real-world data be filtered in order to create balanced datasets.

8.3.2 Datasets

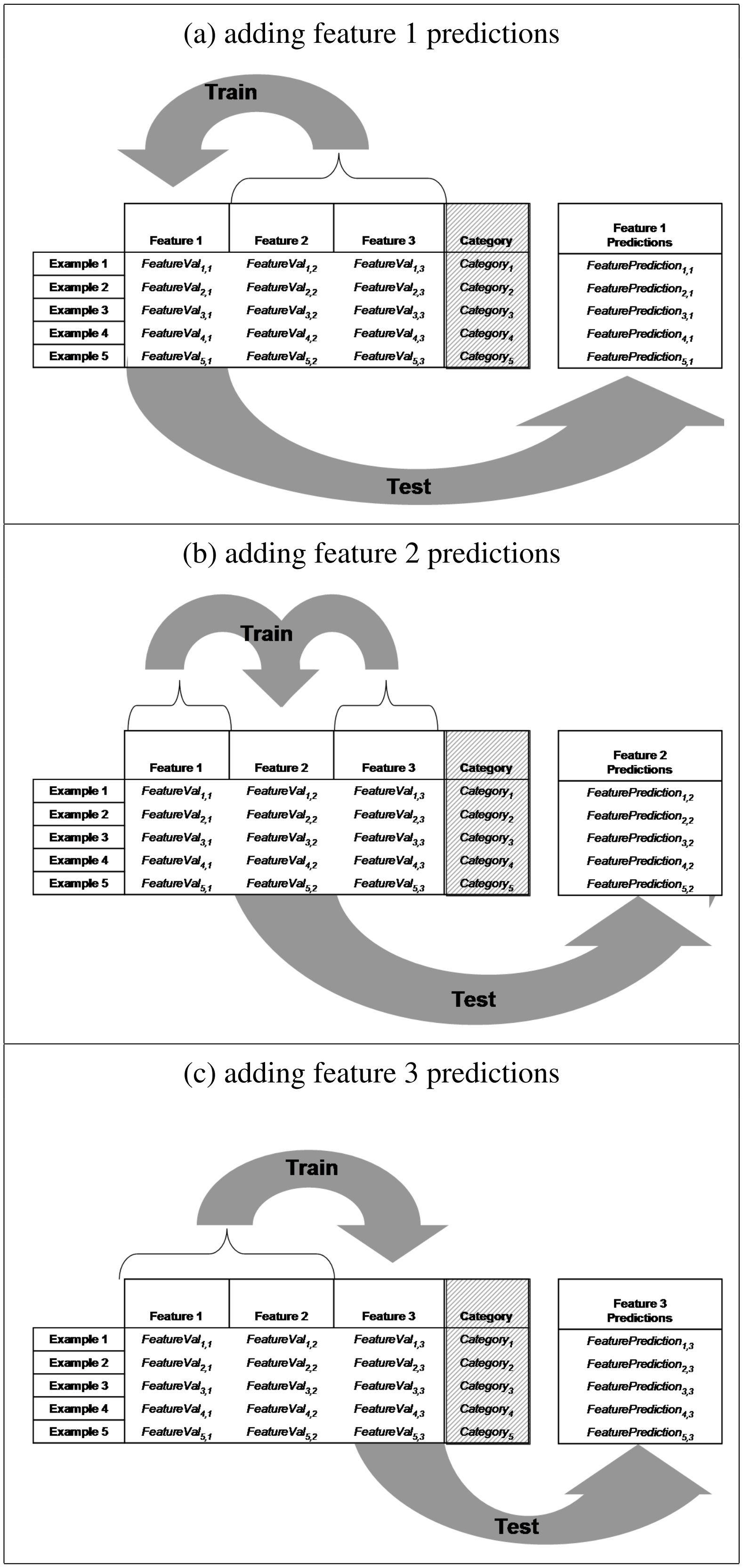
I test five real-world datasets from the UCI Machine Learning Repository (Asuncion, et al., 2007) and one synthetic dataset. The five UCI datasets are *thyroid disease* database (Quinlan, et al., 1987), the *primate splice-junction* database (Towell, et al., 1992), the *contraceptive method*

Table 8.1 **Category prediction in typical machine-learning methodology:** In typical machine-learning methodology, the learner is trained to predict a *Category* column from the features in the *Feature* columns. I have depicted the training phase here with the *Train* arrow from the *Feature* columns, whose values are used to make the prediction, to the *Category* column, whose values are being predicted. I have depicted the testing phase with the *Test* arrow from the *Category* column being predicted to the column of *Predictions* made by the learner.



	Feature 1	Feature 2	Feature 3	Category
Example 1	*FeatureVal$_{1,1}$*	*FeatureVal$_{1,2}$*	*FeatureVal$_{1,3}$*	*Category$_1$*
Example 2	*FeatureVal$_{2,1}$*	*FeatureVal$_{2,2}$*	*FeatureVal$_{2,3}$*	*Category$_2$*
Example 3	*FeatureVal$_{3,1}$*	*FeatureVal$_{3,2}$*	*FeatureVal$_{3,3}$*	*Category$_3$*
Example 4	*FeatureVal$_{4,1}$*	*FeatureVal$_{4,2}$*	*FeatureVal$_{4,3}$*	*Category$_4$*
Example 5	*FeatureVal$_{5,1}$*	*FeatureVal$_{5,2}$*	*FeatureVal$_{5,3}$*	*Category$_5$*

Predictions
Prediction$_1$
Prediction$_2$
Prediction$_3$
Prediction$_4$
Prediction$_5$



(a) adding feature 1 predictions

	Feature 1	Feature 2	Feature 3	Category
Example 1	*FeatureVal*$_{1,1}$	*FeatureVal*$_{1,2}$	*FeatureVal*$_{1,3}$	*Category*$_{1}$
Example 2	*FeatureVal*$_{2,1}$	*FeatureVal*$_{2,2}$	*FeatureVal*$_{2,3}$	*Category*$_{2}$
Example 3	*FeatureVal*$_{3,1}$	*FeatureVal*$_{3,2}$	*FeatureVal*$_{3,3}$	*Category*$_{3}$
Example 4	*FeatureVal*$_{4,1}$	*FeatureVal*$_{4,2}$	*FeatureVal*$_{4,3}$	*Category*$_{4}$
Example 5	*FeatureVal*$_{5,1}$	*FeatureVal*$_{5,2}$	*FeatureVal*$_{5,3}$	*Category*$_{5}$

Feature 1 Predictions
FeaturePrediction$_{1,1}$
FeaturePrediction$_{2,1}$
FeaturePrediction$_{3,1}$
FeaturePrediction$_{4,1}$
FeaturePrediction$_{5,1}$

(b) adding feature 2 predictions

	Feature 1	Feature 2	Feature 3	Category
Example 1	*FeatureVal*$_{1,1}$	*FeatureVal*$_{1,2}$	*FeatureVal*$_{1,3}$	*Category*$_{1}$
Example 2	*FeatureVal*$_{2,1}$	*FeatureVal*$_{2,2}$	*FeatureVal*$_{2,3}$	*Category*$_{2}$
Example 3	*FeatureVal*$_{3,1}$	*FeatureVal*$_{3,2}$	*FeatureVal*$_{3,3}$	*Category*$_{3}$
Example 4	*FeatureVal*$_{4,1}$	*FeatureVal*$_{4,2}$	*FeatureVal*$_{4,3}$	*Category*$_{4}$
Example 5	*FeatureVal*$_{5,1}$	*FeatureVal*$_{5,2}$	*FeatureVal*$_{5,3}$	*Category*$_{5}$

Feature 2 Predictions
FeaturePrediction$_{1,2}$
FeaturePrediction$_{2,2}$
FeaturePrediction$_{3,2}$
FeaturePrediction$_{4,2}$
FeaturePrediction$_{5,2}$

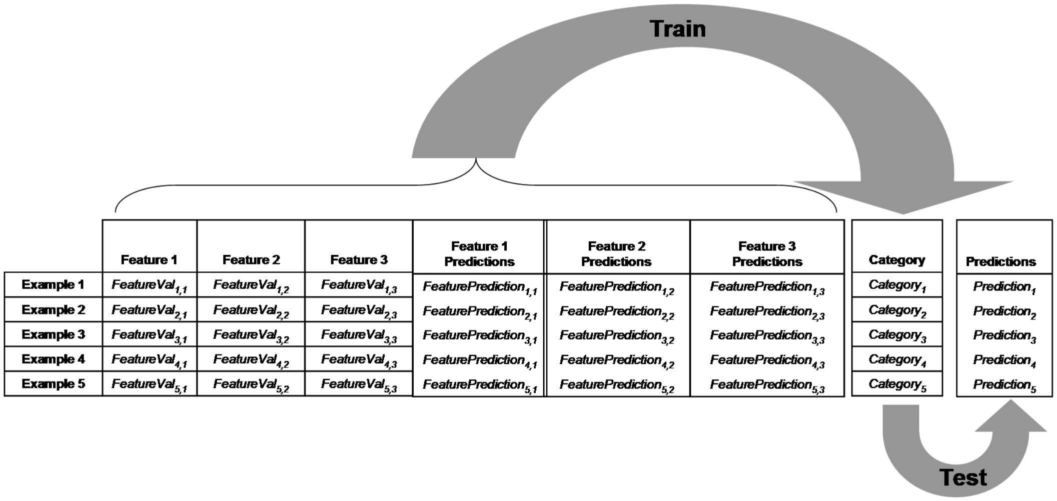
(c) adding feature 3 predictions

	Feature 1	Feature 2	Feature 3	Category
Example 1	*FeatureVal*$_{1,1}$	*FeatureVal*$_{1,2}$	*FeatureVal*$_{1,3}$	*Category*$_{1}$
Example 2	*FeatureVal*$_{2,1}$	*FeatureVal*$_{2,2}$	*FeatureVal*$_{2,3}$	*Category*$_{2}$
Example 3	*FeatureVal*$_{3,1}$	*FeatureVal*$_{3,2}$	*FeatureVal*$_{3,3}$	*Category*$_{3}$
Example 4	*FeatureVal*$_{4,1}$	*FeatureVal*$_{4,2}$	*FeatureVal*$_{4,3}$	*Category*$_{4}$
Example 5	*FeatureVal*$_{5,1}$	*FeatureVal*$_{5,2}$	*FeatureVal*$_{5,3}$	*Category*$_{5}$

Feature 3 Predictions
FeaturePrediction$_{1,3}$
FeaturePrediction$_{2,3}$
FeaturePrediction$_{3,3}$
FeaturePrediction$_{4,3}$
FeaturePrediction$_{5,3}$

Table 8.2 **Creating additional features in the** *pretraining* **algorithm:** For each feature in the original dataset, another feature is added. Each panel, as in Table 8.1, has a *Train* arrow from the *Feature* columns whose values are used to make the prediction to the column whose values are being predicted and a *Test* arrow from the column being predicted to the column of *Predictions* made by the learner. The difference is that the column being predicted is not the *Category* column. The columns being predicted are the *Feature* columns. The reason to do this is to produce columns of *Feature Predictions* that will be used in the prediction of the *Category column* (Table 8.3). Note that the *Category column* is ignored during this phase of the algorithm.

Table 8.3 **Final category prediction in the** *pretraining* **algorithm:** Much like typical machine-learning methodology (Table 8.1), the learner is trained to predict a *Category* column from the features in the *Feature* columns. The training phase is depicted by the *Train* arrow from the columns whose values are used to make the prediction to the *Category* column whose values are being predicted. The difference is that, in addition to the original features, there are also columns for the *Feature Prediction* columns that also function as features in this phase of the algorithm. The testing phase is depicted with the *Test* arrow from the *Category* column being predicted to the column of *Predictions* made by the learner.



	Feature 1	**Feature 2**	**Feature 3**	**Feature 1 Predictions**	**Feature 2 Predictions**	**Feature 3 Predictions**	**Category**	**Predictions**
Example 1	$FeatureVal_{1,1}$	$FeatureVal_{1,2}$	$FeatureVal_{1,3}$	$FeaturePrediction_{1,1}$	$FeaturePrediction_{1,2}$	$FeaturePrediction_{1,3}$	$Category_1$	$Prediction_1$
Example 2	$FeatureVal_{2,1}$	$FeatureVal_{2,2}$	$FeatureVal_{2,3}$	$FeaturePrediction_{2,1}$	$FeaturePrediction_{2,2}$	$FeaturePrediction_{2,3}$	$Category_2$	$Prediction_2$
Example 3	$FeatureVal_{3,1}$	$FeatureVal_{3,2}$	$FeatureVal_{3,3}$	$FeaturePrediction_{3,1}$	$FeaturePrediction_{3,2}$	$FeaturePrediction_{3,3}$	$Category_3$	$Prediction_3$
Example 4	$FeatureVal_{4,1}$	$FeatureVal_{4,2}$	$FeatureVal_{4,3}$	$FeaturePrediction_{4,1}$	$FeaturePrediction_{4,2}$	$FeaturePrediction_{4,3}$	$Category_4$	$Prediction_4$
Example 5	$FeatureVal_{5,1}$	$FeatureVal_{5,2}$	$FeatureVal_{5,3}$	$FeaturePrediction_{5,1}$	$FeaturePrediction_{5,2}$	$FeaturePrediction_{5,3}$	$Category_5$	$Prediction_5$

Table 8.4 **Pseudocode for the *pretraining* algorithm**

```
Algorithm 8.2.1: PRETRAIN($dataset, features, category$)

procedure PRETRAIN($dataset$)
    $n \leftarrow$ the number of features in dataset
    Randomly reorder dataset
    for $m \leftarrow 1$ to $n$
        $predFeatures_m \leftarrow$ CROSSVALIDATE( $dataset,$
                                                  $features\{1, 2, \cdots, m\text{-}1, m\text{+}1, \cdots, n\},$
                                                  $feature_m$ )
    $results \leftarrow$ CROSSVALIDATE( $dataset + predFeatures,$
                                       $features + predFeatures,$
                                       $category$ )
    Calculate accuracy by comparing results to category


procedure CROSSVALIDATE($dataset, features, category$)
    Divide dataset into 10 disjoint test sets: $testSet_1$ to $TestSet_{10}$
    for $i \leftarrow 1$ to $10$
        $trainingSet \leftarrow dataset - testSet_i$
        for each $example \in testSet_i$
            Given: feature values for example and trainingSet and categories for trainingSet
            Predict: category for example
            if category is numeric
              then record prediction - category in results
              else record ”CORRECT” or ”INCORRECT” in results
    return ($results$)
```

choice (Lim, et al., 1999) database, the *mushrooms* database (Schlimmer, 1987) and the *breast cancer Wisconsin* database (Street, et al., 1993). I modified most of these by undersampling the positive class. The ratios of *negative* to *positive* examples after undersampling are listed in Table 8.6. I also modified the *splice junctions* database by removing *exon/intron* sites, leaving only *intron/exon* sites and *negative* values, reducing it to a two-class problem. The synthetic dataset is called *3-bit parity*.

3-bit parity is the function:

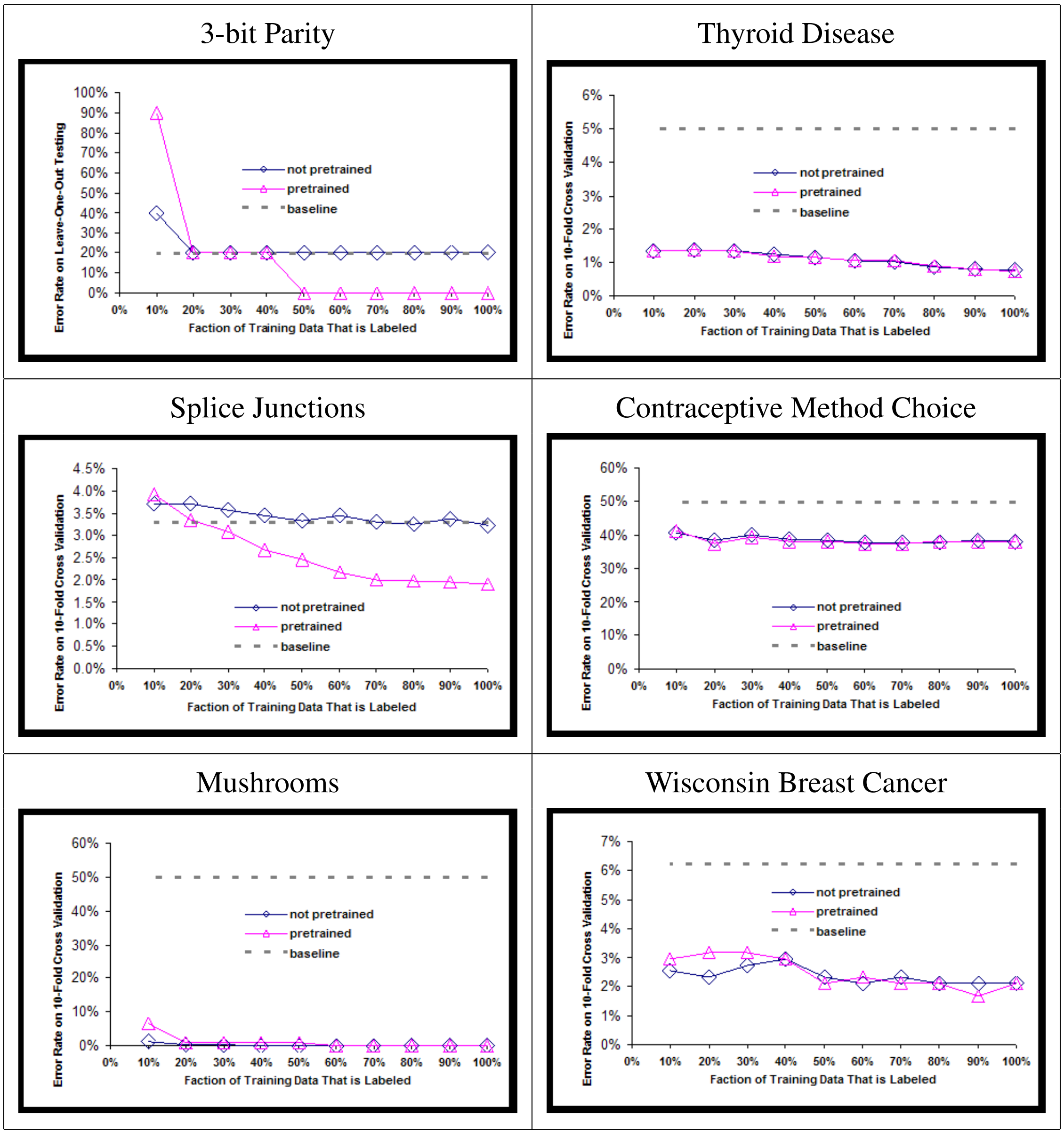
$$((Feature_1 \oplus Feature_2) \equiv Feature_3) \equiv Category$$

Each of the features and the category have values of either *TRUE* or *FALSE*. The truth table for this function is listed in Table 8.5.

Table 8.5 **The 3-bit parity truth table**

$Feature_1Value$	$Feature_2Value$	$Feature_3Value$	$CategoryValue$
$True$	$True$	$True$	**False**
$True$	$True$	$False$	**True**
$True$	$False$	$True$	**True**
$True$	$False$	$False$	**False**
$False$	$True$	$True$	**True**
$False$	$True$	$False$	**False**
$False$	$False$	$True$	**False**
$False$	$False$	$False$	**True**

All of the datasets are tested using 10-fold cross validation except for *3-bit parity*. Since *3-bit parity* has so few examples, leave-one-out testing is used. I also ran each experiment ten times with the data randomly reordered each time. The results reported here are the average of these runs.



3-bit Parity

Thyroid Disease

Splice Junctions

Contraceptive Method Choice

Mushrooms

Wisconsin Breast Cancer

Figure 8.3 **Learning curves for the six datasets.** As is standard practice in machine learning the *baseline* error rate is the error rate that would be achieved if the majority class were always chosen. In addition to the synthetic dataset, statistically significant results were also achieved in two of the UCI datasets: the *splice junction* dataset and the *Wisconsin breast cancer* dataset.

Table 8.6 **Positive:negative ratios for UCI datasets after undersampling**

UCI Dataset	Positive:Negative Ratio
thyroid disease (Quinlan, et al., 1987)	1:20
primate splice-junctions (Towell, et al., 1992)	1:30
contraceptive method choice (Lim, et al., 1999)	1:1
mushrooms (Schlimmer, 1987)	1:1
breast cancer Wisconsin (Street, et al., 1993)	1:15

8.4 Results

Figure 8.3 contains learning curves for the six datasets. Increasing amounts of the training data are labeled at each point on the curve.

The SVM does seem to sometimes make good use of the *feature prediction* columns. On my synthetic dataset and two of the UCI datasets, the these columns provide statistically significant improvement of the performance of the learning algorithm. It is surprising that, on the *splice junctions* dataset, the improvement *increases* as the amount of labeled data increases. This could be a result of predicted features that are only useful in the presence of sufficient labeled data.

8.5 Discussion

At the center of conventional semi-supervised learning algorithms is what is known as *the cluster assumption* (Chapelle et al. 2006). Simply stated, it is the assumption that two examples that appear in the same dense section of feature space are likely to be in the same class.

Most current semi-supervised learning methods rely on this assumption (Chapelle et al. 2006), but my SNP-finding method is different because it relies on density in another region of feature space in order to learn. This is similar to a feature-selection step that eliminates the feature that separates the example from the dense part of feature space, but with one important difference. This feature is not ignored. It plays a different role as the switch that helps to decide the class of the example.

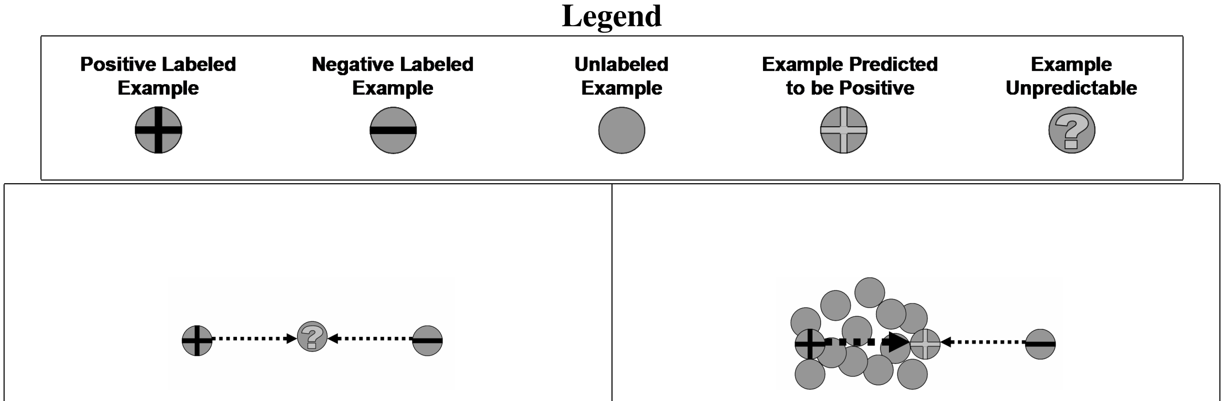
This algorithm, as it currently exits, has several weaknesses. Chief among them is the fact that, though it does not tend to decrease accuracy, it only seems to produce a statistically significant improvement in a small fraction of datasets. This may be a result of the scarcity of datasets that contain *key* features. The cost of running this algorithm is another serious issue. The training and testing time are effectively multiplied by the number of features. One way to mitigate this effect might be to use a Bayes network learner (Mitchell 1997) as the learner for the *feature prediction* phase of the algorithm. The Bayes Network representation has the advantage that, once the relationships among the features are learned, any feature can be predicted naturally from the others. No further training is needed.

An illustration of how this works can be found in Figure 8.4. Panels (a) and (b) illustrate the use of the cluster assumption in making use of unlabeled data. Figure 8.5 illustrates the situation where, under the *Nearby-Cluster Hypothesis*, a nearby cluster – differing in a single feature – can cause two examples – one labeled and another unlabeled – to have the same feature wrongly predicted. In this case, they will have that incorrect feature prediction in common. In this way, the labeled example passes information *through* the nearby cluster as in panel 8.5(a). This can even work if the cluster itself has the opposite label as in panel 8.5(b).

8.6 Current and Future Work

Empirical study has shown that not all real-world datasets respond well to the *pretraining* algorithm. This may be because not all real-world datasets contain *key* features. I would like to develop method for identifying datasets that have this property in order to exploit it.

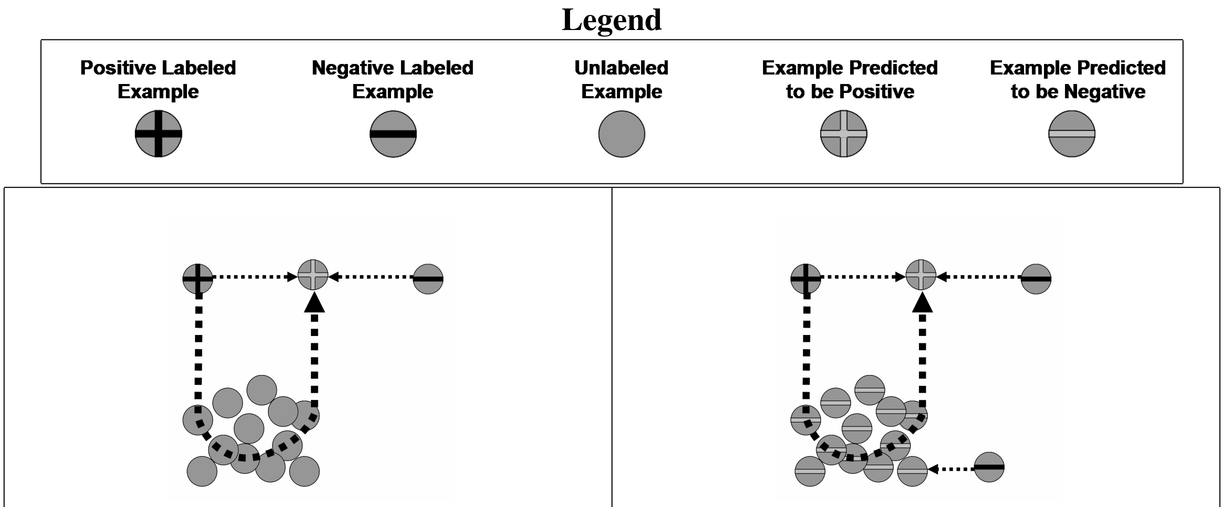
Another possible reason for this difficulty is that the algorithm needs to be improved. To this end, I have begun to develop an SVM kernel that tries to make use of *The Nearby Cluster Hypothesis* directly.



(a) **Learning in the absence of unlabeled data:** In *supervised learning* methodology, only labeled data is used. If a learner is given labeled training data that equally predict that an example is in two different classes (In this case, *positive* and *negative*) the learner does not have a good way to make a prediction one way or the other. A simple case of this is depicted here where only three examples exist, the *positive* example and the *negative* example are equidistant from the example to be predicted. In this case, both of the labeled examples have the same influence on the example to be predicted.

(b) **The cluster assumption:** Semi-supervised learning uses both *labeled* and *unlabeled* data to make predictions. Most semi-supervised learning algorithms depend on *the cluster assumption* (Chapelle et al. 2006) which state that examples in the same category will tend be nearer to each other in feature space than examples that are not, and its corollary of *low-density separation* (Chapelle et al. 2006) which states that category boundaries are less likely to appear in areas of feature space that are dense with examples than those that are sparse. This has the effect of increasing the influence of labeled examples acting through dense regions of feature space.

Figure 8.4 **The cluster assumption**



(a) **The nearby-cluster hypothesis:** I posit here that it is useful in some datasets, to allow the density of examples in feature space adjacent to the region between two examples to influence affect the interaction between those examples.

(b) **The nearby cluster can have a different label:** To illustrate how this differs from a feature-selection step that simply removes the feature that separates the two examples from the dense area of feature space, it is important to note that they do not need to have the same label or predicted label as the adjacent dense region of feature space.

Figure 8.5 **The nearby-cluster hypothesis**

Chapter 9

Literature-Based Expression Analysis

The development of microarrays and their associated large collections of experimental data have led to the need for automated methods that assist in the interpretation of microarray-based biomedical experiments. In this chapter, I present a method for creating partial interpretations of microarray experiments that combine the expression-level data with textual information about individual genes. These interpretations consist of models that characterize the genes whose expression levels were up- (or down-) regulated. The goal of the models is to assist a human scientist in understanding the results of an experiment. Our approach is to use machine learning to create models that are both accurate and comprehensible. I report here on experiments using actual *E. coli* microarray data, demonstrating the trade-offs between model accuracy and comprehensibility.

In order to make them comprehensible, my models are expressed in terms of English words from text descriptions of individual genes. I currently get these descriptions from the curated *SwissProt* protein database (Bairoch and Apweiler 2000). It contains annotations of proteins; I use the text associated with the protein generated by a gene as the description of that gene. My models consist of sets of words from these descriptions that characterize the up-regulated or down-regulated genes. Note that I can use the same text descriptions of the genes to generate interpretations of many different microarray experiments. In each experiment, different genes will be up-regulated or down-regulated, even though the text description associated with each gene is the same across all experiments.

The basic task can be described as follows:

Given:

a) *The (numeric) RNA-expression levels of each gene on a gene array under two conditions, before and after a particular event (e.g., antibiotic treatment)*, and

b) *For each gene on the microarray, the* SwissProt *text describing the protein produced by that gene.*

Produce: *A text-based model that accurately characterizes the genes that were up-regulated or down-regulated in response to the event.*

In my work, my models are sets of disjunctive IF-THEN rules of the form:

IF *Word1* and *Word2* appear in the gene's annotation and *Word3* and *Word4* are not present

THEN this gene is up-regulated.

For shorthand, in the remainder of this chapter I will only present the IF part of the rules and I always focus on the up-regulated group (an arbitrary choice). That is, I would list the above rule as:

Word1 and *Word2* and NOT *Word3* and NOT *Word4*

Since my rules are disjunctive, if any of the rules match a gene's annotation, my model characterizes that gene as *up-regulated*. If no rule matches, then my model characterizes that gene as *down-regulated*.

My work is related to several prior attempts to use machine learning to predict gene-regulation levels (e.g., Brown et al. 2000, Dudoit et al. 2000; Xing et al. 2001), but my focus is different in that my goal is not to predict gene-regulation levels, but to automatically generate human-readable characterizations of the *up-* or *down-regulated* genes to help scientists generate hypotheses to explain experiments.

I investigate herein a new rule-building algorithm of my own design against a standard successful algorithm from the machine-learning literature, evaluating how well each satisfies my desiderata of accuracy and comprehensibility. The standard approach to which I compare is PFOIL , (Mooney 1995) a rule learner based on propositional logic.

In my current set of experiments, I consider a gene up-regulated if its ratio of RNAafter to RNAbefore (the gene's expression ratio) is greater than 2; if this ratio is less than I consider it down-regulated. As is commonly done, I currently discard as ambiguous all genes whose expression ratio is between $\frac{1}{2}$ and 2. I train the learners only using the data set of up-regulated and down-regulated genes and do not attempt to model the ambiguous genes.

In a published paper (Molla et al. 2002) we describe experiments that pitted two successful, standard, machine-learning algorithms against each other at the same task of interpreting gene chip expression data using text annotating the genes. The two algorithms were PFOIL and Nave Bayes (Mitchell 1997). Here I introduce a new algorithm, GORB. I also propose an atypical evaluation method for machine-learning algorithms. Previously, I used cross-validation, the dominant technique for machine-learning evaluation. We now argue that a statistical method called the *permutation test* is actually a more appropriate metric for characterization tasks and that, though cross-validation is still a good measure for predictive accuracy, the permutation test could be applied to other learning tasks where a model or characterization of the data, rather than an accurate predictor, is the desired output.

I use this method to measure my first desired property: *accuracy*. I record the accuracy of the model in classifying the examples in the entire data set. This is in contrast to my previous work, which recorded the accuracy of models on a held-out test set to ensure that the accuracy measurement was unbiased. Instead of using a test set, in this paper, I repeatedly randomly permute the labels of the examples, train the learner and then record the accuracy of the resulting model each time. I only consider the model from the real data to be significant if the accuracy of the model is significantly better than the accuracy of the models from the permuted data sets. Section 9.1.3 further explains the permutation test.

My second desired property is *human comprehensibility*. As mentioned before, comprehensibility is the reason I express the rules in terms of English words. The actual comprehensibility of a particular model, however, is difficult to measure. I use a crude approximation by counting the number of distinct SwissProt words appearing in a given model.

Section 9.1 presents the machine-learning algorithms I investigate in my experiments. Section 9.2 further explains my experimental methodology and Section 9.3 presents and discusses experimental results obtained using data from the Blattner *E. coli* Laboratory at the University of Wisconsin.

9.1 Algorithm Descriptions

This section describes the two algorithms – PFOIL (Mooney 1995) – and GORB (General-purpose One-step-look-ahead Rule Builder), an algorithm devised by me. Both algorithms take as input a collection of training instances (in my case, genes), labeled as belonging to one of two classes (which I will call *up* and *down*), and described by a vector of Boolean-valued features. Each feature corresponds to a word being present or absent from the text description of the gene. Both algorithms produce a model that can be used to categorize a gene on the basis of its feature values (i.e., the words describing it).

9.1.1 PFOIL

PFOIL (Mooney 1995) is a propositional version of FOIL (Quinlan 1990), a rule-building algorithm that incrementally builds rules that characterize the instances of a class in a data set. FOIL builds rules for a first-order logic language, so that the rules are conjunctions of features (in this case, English words or their negation) that may contain logical variables (and may even be recursive) and must be interpreted by a first-order reasoning engine such as Prolog. PFOIL uses a simpler propositional language, and builds rules that are conjunctions of features. PFOIL rules can be interpreted straightforwardly – a rule covers an instance if each feature in the rule is true of the instance. In my domain, a rule specifies words that must or must not be present in a gene's annotation.

PFOIL builds a set of rules by constructing one rule at a time. It constructs each rule by adding one feature (or its negation) at a time to the current rule. At each step, it chooses the feature that maximizes the performance of the rule according to the *FOILGain* measure. It stops adding to a rule when either the rule covers only positive instances, or none of the remaining features have a positive FOILGain. When a rule is complete, the algorithm removes all of the positive instances covered by that rule from the data set, and then builds a new rule if there are any positive examples not yet covered by at least one learned rule.

FOILGain is a measure of the improvement that would be obtained by adding a new feature to a rule. It is a trade-off between the coverage of the new rule – the number of positive instances of the class that are covered by the rule – and the increase in precision of the rule – the fraction of the instances covered by the rule that are positive:

$$F\textsc{oil}Gain(rule, f) = p(log(\frac{p}{p+n})) - log(\frac{P}{P+N})$$

where P and N are the number of positive and negative instances covered by rule, and p and n are the number of positive and negative instances that are covered when feature f is added to rule.

As originally described by Mooney (1995), PFOIL does not prune its rule set. Because PFOIL keeps constructing rules until it has covered all the positive instances, a data set with noise is likely to result in a large set of rules, many of which may be specific to particular instances.

To address this problem, I have extended PFOIL to include a rule-pruning stage, along the lines of the pruning in FOIL. In the pruning stage, the algorithm repeatedly removes a single feature from one of the rules, choosing the feature whose removal results in the highest accuracy of the remaining rule set. When all the features are removed from a rule, the rule is removed from the rule set. Rather than halting the pruning when the accuracy peaks, in my experiments I continue the pruning until the rule set is empty in order to explore the trade-off between comprehensibility and accuracy.

9.1.2 GORB

The GORB algorithm (Table 9.1) is similar to that of PFOIL in terms of its input and output. It searches the identical hypothesis space of possible rules, but differs in how it searches this space.

Like PFOIL, GORB explores the hypothesis space by adding one feature at a time to an ever-expanding disjunction of conjunctive rules. One difference, however, is that instead of building the rules sequentially, one rule at a time, GORB considers adding a feature to any existing rule or starting a new rule. This is illustrated in Figure 9.2 with each letter representing one feature. At each step, the current rule set is illustrated.

The other difference is that, instead of using an information-gain-based heuristic to decide which feature to add, GORB computes the accuracy that will result from the addition of this feature. Though time-consuming, this method directly seeks to improve accuracy, which is my desired property.

As with my version of PFOIL, I have included a pruning phase. It works identically to my PFOIL pruning stage, repeatedly removing a single feature from one of the rules, choosing the feature whose removal results in the highest accuracy of the remaining rule set. It is worth noting that this phase, in both PFOIL and GORB, is essentially GORB's hypothesis-space search in reverse. Instead of searching the space of possible features for the one whose addition results in the best accuracy, the pruning algorithm searches the space of features included in the model for the one whose removal results in the best accuracy.

9.1.3 The Permutation Test

A property of both PFOIL and GORB is that they are guaranteed to find a model for any data set, regardless of whether there is any relationship between the descriptions and the labels, because the space of disjunctions of conjunctive rules is large enough to describe any set of instances with any labels. In the worst case, the algorithms could generate a distinct rule for each instance. In most data sets, especially if the instances have many features, the algorithms will be able to find more compact models that exploit possibly random associations between instances and labels. Therefore, the fact that a model is produced by one of the algorithms is not evidence that the model

Table 9.1 **Rule construction with GORB**

```
Algorithm 9.1.1: GORB($dataSet, featureSet$)

$ruleSet \leftarrow \emptyset, accuracy \leftarrow 0, prevAccuracy \leftarrow -1, curAccuracy \leftarrow 0$
while $accuracy > prevAccuracy$
  $prevAccuracy \leftarrow accuracy$
  for each $feature \in featureSet$
    $newRule \leftarrow$ a new rule consisting only of the current feature: eg.
                "IF feature then up-regulated"
    $curRuleSet \leftarrow ruleSet \cup newRule$
    $curAccuracy \leftarrow$ accuracy of curRuleSet on dataSet
    if $curAccuracy > accuracy$
      then $accuacy \leftarrow curAccuracy$
    for each $rule \in ruleSet$
      $newRule \leftarrow$ the current feature added to the current rule
      $curRuleSet \leftarrow ruleSet \cup newRule$
      $curAccuracy \leftarrow$ accuracy of curRuleSet on dataSet
      if $curAccuracy > accuracy$
        then $accuacy \leftarrow curAccuracy$
    if $accuracy > prevAccuracy$
      then Add the feature that generated accuracy to the rule in ruleSet (either new or
           existing) where accuracy was measured
return ($ruleSet$)
```

Table 9.2 **Hypothesis space search: PFoil vs. GORB**

	$Step1$	$Step2$	$Step3$	$Step4$	$Step5$	$Step6$
PFoil	$Rule1 : \mathbf{A}$	$Rule1 : A\mathbf{B}$	$Rule1 : AB$ $Rule2 : \mathbf{C}$	$Rule1 : AB$ $Rule2 : C\mathbf{D}$	$Rule1 : AB$ $Rule2 : CD\mathbf{E}$	$Rule1 : AB$ $Rule2 : CDE$ $Rule3 : \mathbf{F}$
GORB	$Rule1 : \mathbf{A}$	$Rule1 : A$ $Rule2 : \mathbf{B}$	$Rule1 : A\mathbf{C}$ $Rule2 : B$	$Rule1 : AC$ $Rule2 : B$ $Rule3 : \mathbf{D}$	$Rule1 : AC\mathbf{E}$ $Rule2 : B$ $Rule3 : D$	$Rule1 : ACE$ $Rule2 : B\mathbf{F}$ $Rule3 : D$

represents a meaningful relationship between the descriptions and the labels. To show that a model is significant, one must show that the algorithm is unlikely to have produced a model of the same quality as a result of random associations.

A standard way of showing the significance of a model in a classification or prediction task is to test its accuracy on a held-out test set, since a model that is merely the result of chance associations in the training set will perform poorly on the test set. This approach is appropriate for prediction tasks, since the accuracy of the model on the test set is also a good estimate of the accuracy on future instances, which is important in a prediction task. In a characterization task, there are no future instances to predict, and the training data is the complete data set. Therefore measuring accuracy on a test set is not a useful measure. Furthermore, the limited size of the data set means that holding any data out of the training set will likely reduce the quality of any models that are learned. A permutation test is a better way of measuring the significance for a characterization task since it does not require holding out any data and is unrelated to prediction.

A permutation test (Good 2001) is a statistical test to determine significance by comparing the results of an algorithm on a real data set to the results of the algorithm on permutations of the real data set in which the meaningful relationships have been lost. If the model from the real data set is no better than the models from the permuted data sets, then it is not considered meaningful. If the model from the real data set is much better than models from the permuted data sets, then the model must be taking advantage of semantically meaningful relationships in the data set and is considered meaningful.

In my application, the quality of a model is its accuracy on the data set given the size of the model. My permutation test compares the accuracy of a model on the real data set to the accuracy of the models of the same size produced on data sets obtaining by randomly permuting the labels (i.e., whether the gene is up- or down-regulated) of the real data set. The model from the real data set is considered significant if its accuracy is clearly higher than the accuracies of all the models from the permuted data sets. It is important in any permutation test that enough permutations are considered to obtain a statistically valid result. The number of permutations required depends on the data. I have found that 30 permutations are adequate for my particular application .

9.2 Experimental Methodology

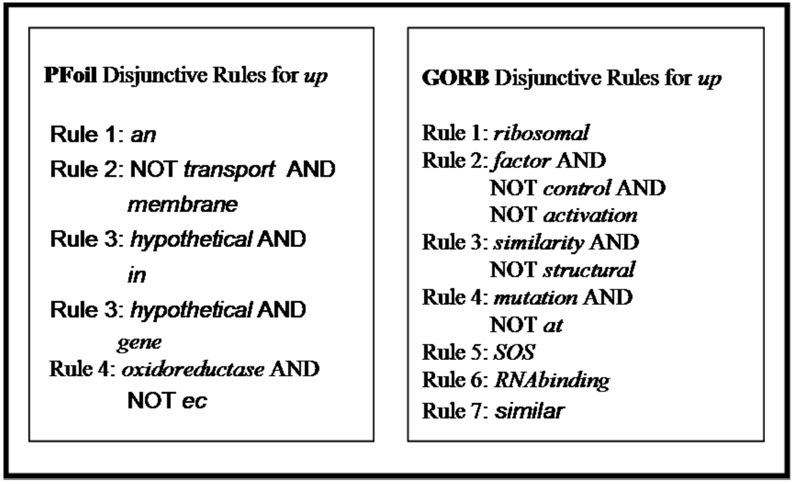
The data I am using are from microarray experiments performed by the Blattner *E. coli* Sequencing Laboratory at the University of Wisconsin. In my computational experiments, I used my methods on 43 different experiments that measure expression of approximately 4,200 genes in *E. coli* under various conditions. These conditions include heat and cold shock and various antibiotics for various periods of time. In order to measure the change in expression due to each condition, I compare these expression levels to the mean of those measured in six replicate microarrays under standard conditions . By this definition, each experiment includes, on average, 717 up-regulated genes, 352 down-regulated genes and 3221 unregulated genes.

To construct the text description for the genes, I use all words of all the text fields in the SwissProt database. These include the comment (CC) fields (with the exception of the database (CDB) and mass spectrometry (CMS) topics since these only contain ibnformation about the experimental techniques used, not about the protein itself), the description (DE) field, the Organism Classification (OC) field, the keyword (KW) field, and the reference title (RT) fields (with the exception of titles containing: The Complete Sequence of the *E. coli* K12 Genome).

9.3 Experimental Results

Figure 9.3 shows the rulesets of PFOIL and GORB on a typical run, this one from my cold-shock testbed, after the all features have been added and the rule sets have been pruned to 10 features each. As it shows, the format of the models is similar, but not the content. Some of the PFOIL rules seem plausible. For example, Rule 2: "NOT transport and membrane" can be interpreted as an indication that the non-transport-related membrane proteins are involved in the *E. coli*'s cold-shock response. Others seem to be spurious. Rule 1 is simply the word *an*. The GORB rules, for the most part, seem more reasonable. Rule 5 points predictably to *E. coli*'s "SOS" response, a well known *E. coli* stress response pathway. Rule 2 points to factors other than control factors and activation factors. Their up-regulation may provide a hint at what types of processes are involved in the *E. coli* cold shock response.

Table 9.3 **Sample small rule-sets built from the experimental cold-shock data**



PFoil Disjunctive Rules for *up*

Rule 1: *an*
Rule 2: NOT *transport* AND
membrane
Rule 3: *hypothetical* AND
in
Rule 3: *hypothetical* AND
gene
Rule 4: *oxidoreductase* AND
NOT *ec*

GORB Disjunctive Rules for *up*

Rule 1: *ribosomal*
Rule 2: *factor* AND
NOT *control* AND
NOT *activation*
Rule 3: *similarity* AND
NOT *structural*
Rule 4: *mutation* AND
NOT *at*
Rule 5: *SOS*
Rule 6: *RNAbinding*
Rule 7: *similar*

Figures 9.1.1 and 9.1.2 show the results of the permutation tests on the two algorithms on four of the experimental data sets and the results of each step of pruning on the two algorithms on the same data set. All data points for model sizes between one and twenty are plotted. The baseline accuracies for these experiments are also plotted; these are the accuracies that would be achieved if the learner always chose the most frequent category (i.e., up-regulated).

On both the real data and the permuted data, PFOIL tends to make much larger models. Though the accuracy of PFOIL's complete model on real data is comparable to the accuracy of GORB's, when models of similar size are compared, GORB is substantially more accurate. For example, in the case of heat shock, both complete rule sets are about 87% accurate. However, PFOIL's complete rule set contains 119 words while GORB's contains only 31. By the time PFOIL's rule set is pruned down to 31 rules, its accuracy has dropped to 68%.

Also striking is the fact that, though both algorithms tend to perform better on the real data than on permuted data, GORB's margin is much wider and remains so well into the range of comprehensibility (around 10 words). This is of crucial importance because, as explained earlier, this means that GORB is making models that rely on real patterns in the data since, when those patterns are removed, GORB performs much worse.

9.4 Discussion

Though it may seem surprising that GORB is able to exceed baseline accuracy on randomly permuted data, the fact that the data can be fit whether or not there exists a pattern is precisely the point of doing the permutation test. The permutation test is meant to show, by virtue of the fact that the accuracy is much better on the real data than on the permuted data, that the learner is picking up on real connections between the text and the biological behavior. When these connections are removed, the learner ceases to perform as well.

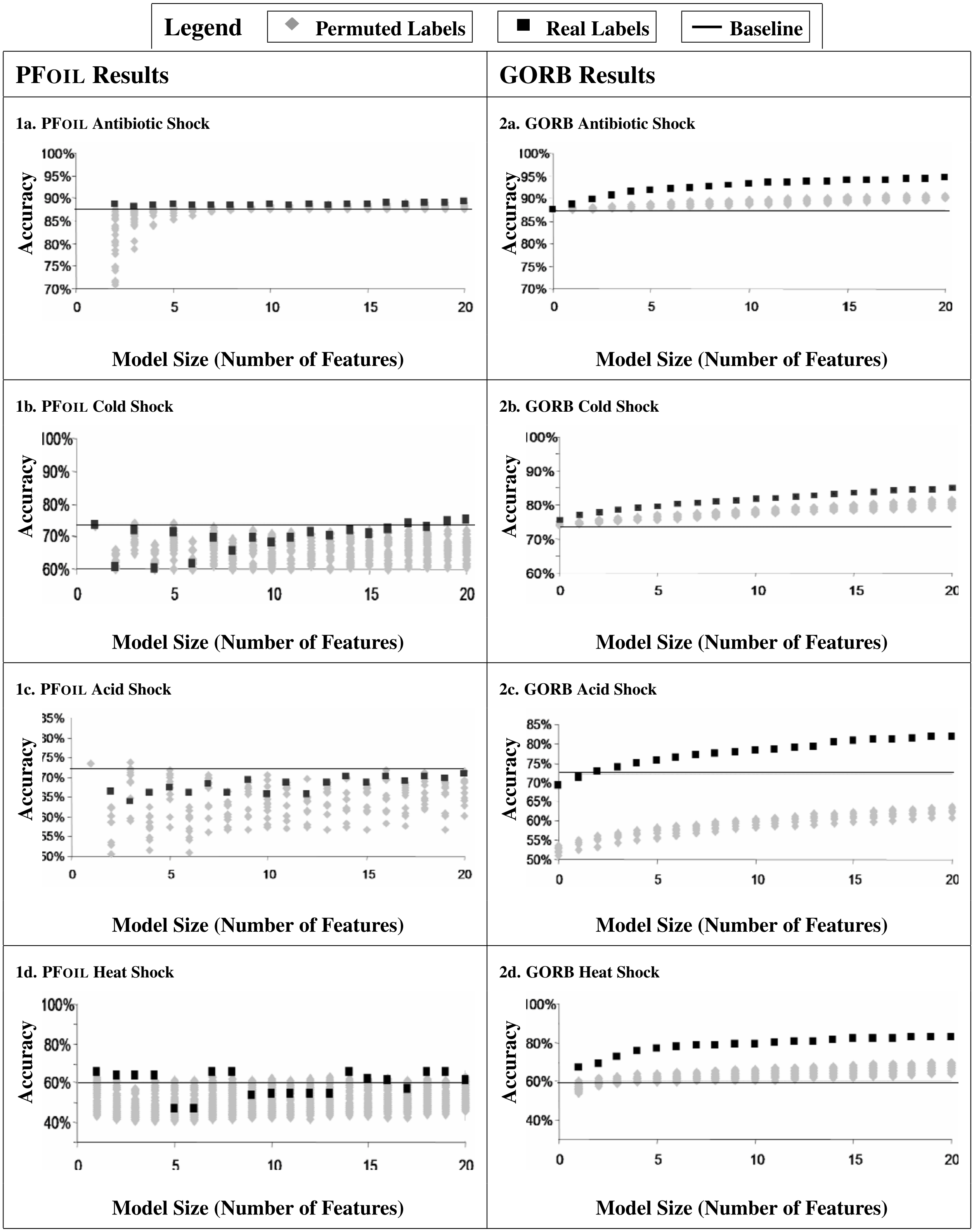
Conversely, it may seem strange that PFOIL does not always exceed the baseline accuracy, even on the non-permuted data. One thing to keep in mind, however, is the fact that the data points plotted in Figure 9.1.1 correspond to the performance of heavily pruned rule sets. The full rule sets exceed the baseline by a wide margin.

Though the GORB rule sets are more accurate and more significant than the PFOIL rule sets across the full range of rule set sizes, the difference in the heavily pruned models is most striking. GORB appears to be much better at characterizing the data when only allowed to use a few words in its models. This suggests that it is doing a better job at identifying the real regularities in the data.

Though more accurate, the GORB strategy does suffer from having a large number of rules that are only relevant in the context of the greater rule set. This occurs because they only specify a single infrequently occurring word that may or may not be descriptive on its own; that is on the whole set of genes, not just those not covered by other rules. One remedy for this may be an variant on GORB that is biased toward rules of similar length or individual rules that each have high precision even if they have only moderate coverage.

By my measure, the comprehensibility of both methods is good. Short models tend to be almost as good as longer ones. It is, however, also clear that, as mentioned earlier, length is a crude measure. Attainment of a sophisticated answer on comprehensibility will probably someday have to involve a large number of human testers.

This work in mining biomedical text for explanations of experiments is an early step in what is now a growing field (Ananiadou and Mcnaught 2005).



PFOIL Results

GORB Results

1a. PFOIL Antibiotic Shock

2a. GORB Antibiotic Shock

1b. PFOIL Cold Shock

2b. GORB Cold Shock

1c. PFOIL Acid Shock

2c. GORB Acid Shock

1d. PFOIL Heat Shock

2d. GORB Heat Shock

Figure 9.1 **PFOIL results vs. GORB results**

Chapter 10

Additional Computational Methods for Molecular Biology

In addition to direct applications of machine learning, I have also developed other algorithms and methods to solve specific problems in molecular biology. Here I will present two of these. The first is a novel extension to a tree-based approach to efficiently choosing microarray probes that are sufficiently unique within a genome. The second is a genetic simulation meant to test a long-held theory in evolutionary biology.

10.1 Probe Uniqueness Testing

When choosing probes for a gene chip, one crucial concern is the uniqueness of the probes. Not only should they be strictly unique in the genome being tested, there should not even be another similar region in this genome. This is because non-specific hybridization can severely confound the results of an experiment.

10.1.1 Suffix Trees

Suffix trees are generally used for this purpose (Kurtz et al. 2001). A suffix tree is an efficient tree data structure designed to hold string data. Each string is encoded as a path from the root of the tree down, with a sequential character of the string represented at each vertex. For the task of uniqueness testing, the tree only needs to be as deep as the longest probe to be tested. Figure 10.1 is an illustration of such a suffix tree holding every possible substring of a very short (15-base-long) DNA sequence. Characters terminating the DNA sequence are denoted by a **"$"**. Also, this particular suffix tree only extends to a depth of 5. Though real-world probes are typlically between

20 and 120 bases long, for simplicity, I have designed this fictitious suffix tree to test probes of length 5.

In this context, the suffix tree is typically used to count the number of mismatches from the most similar region of the genome. Probes are eliminated based on a simple threshold value for this count. Each path from the root is traversed until it has either accumulated enough mismatches to be disregarded or reached the end of the probe sequence. Each path that is disregarded in this way represents a genomic region that is sufficiently dissimilar from the probe sequence. Each path that is explored to the completion of the probe sequence represents a region of the genome that is similar to the probe within the threshold value. This process is illustrated in Figure 10.2.

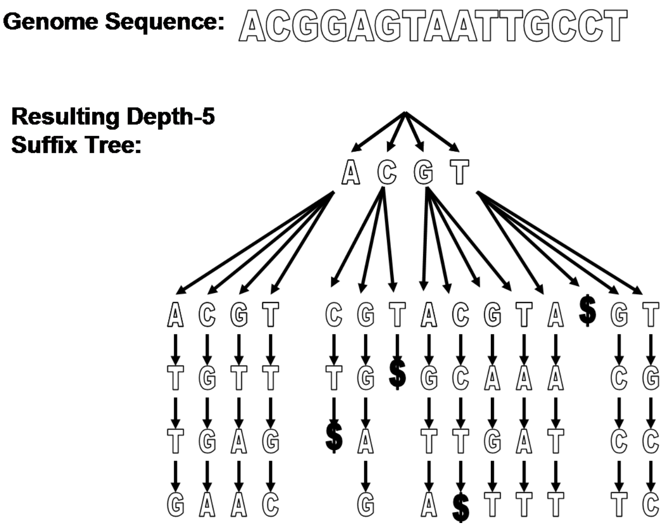

Genome Sequence: ACGGAGTAATTGCCT
Resulting Depth-5 Suffix Tree:
A C G T
A C G T C G T A C G T A $ G T
T G T T T G $ G C A A A C G
T G A G $ A T T G A T C C
G A A C G A $ T T T T C

Figure 10.1 **A suffix tree containing every substring of the genome sequence ACGGAGTAATTGCCT**, truncated, for simplicity, to a maximum depth of 5. The positions that represent the terminus of the DNA sequence are denoted by a **"$"**.

In collaboration with NimbleGen Systems, I have discovered that the bases in the probe do not contribute uniformly to the hybridization characteristics. Specifically, mismatches in the middle of a probe sequence contribute more to uniqueness than those at the ends. This effect can be seen in Figure 10.3 from Tobler et al. (2002). This is a graph of the *normalized information gain* attributed to the identity of each probe in a 24-base-long probe sequence. Though the task is different – we were measuring the contribution of each base to the predicted efficiency of the probe

– the information is still relevant. If the identity of the base at one position is more important than that at another, we assume that a mismatch at that position will also have more of an impact.

Because of this, I have developed a modification to the standard uniqueness-testing algorithm that weights bases differently based on their position in the probe and eliminates probes based on the accumulation of this weight or, more precisely, the lack thereof.

10.1.2 Sequence Alignment

Others have tried to account for this effect by adding a further step to measure the hybridization potential between the probes and the most similar sections of the genome. An alignment step can be used to further filter the probes based on Melting Point (T_m) (Kaderali and Schliep 2002). This will also up-weight bases in the middle of a probe because they will more substantially lower the T_m of the probe-target combination, also known as the *complex*. Unfortunately, this uses $O(n^2)$ time for each probe analyzed, where *n* is the length of the probe. This is not feasible for microarrays for which over 400,000 probes routinely need to be evaluated.

10.1.3 Our Uniqueness Testing Method

So, in collaboration with NimbleGen Systems, I have developed a method to take this into account in the suffix tree-traversal step itself. Instead of simply counting the number of mismatches (up to the threshold value) from every path in the suffix tree (Figure 10.2), we assign a weight to each position in the tree and add the appropriate weight at each node during the traversal of the tree. It is this accumulated weight that is compared to the threshold value to make a determination with regard to uniqueness (Figure 10.4).

Observe that we will never have to explore more than $\frac{threshold}{minWeightBase}$ paths, where $minWeightBase$ is the weight at the base position that contributes the least to the total weight, and that $threshold$ and $minWeightBase$ are both constants specified by the user. This implies that, given

$$minWeightBase > 0$$

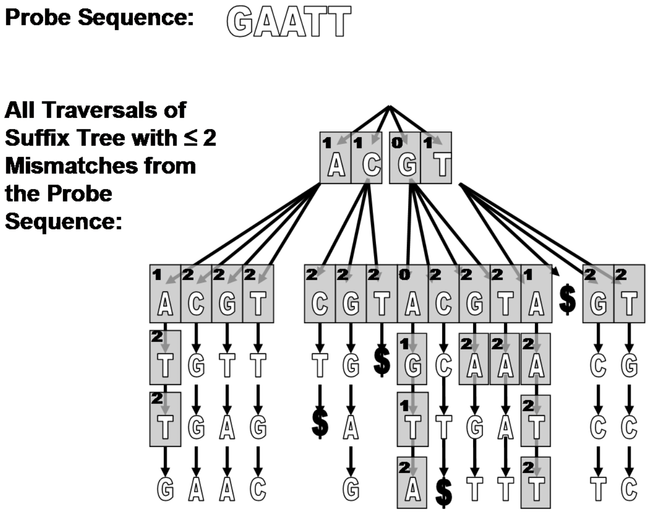

Probe Sequence:
GAATT
All Traversals of Suffix Tree with ≤ 2 Mismatches from the Probe Sequence:

Figure 10.2 **The result of a traversal of the suffix tree in Figure 10.1 by the standard method (Kurtz, 2001) in search of matches to the probe sequence: GAATT**. Each path from the root is traversed until it has either accumulated enough mismatches to be disregarded or reached the end of the probe sequence. Each path that is disregarded in this way represents a genomic region that is sufficiently dissimilar from the probe sequence. Each path that is explored to the completion of the probe sequence represents a region of the genome that is similar to the probe within the threshold value. The nodes that were visited are denoted by a gray box. In each gray box is a number. This specifies the number of mismatches from the probe sequence (**GAATT**) accumulated between the root node and the node whose gray box contains the number.

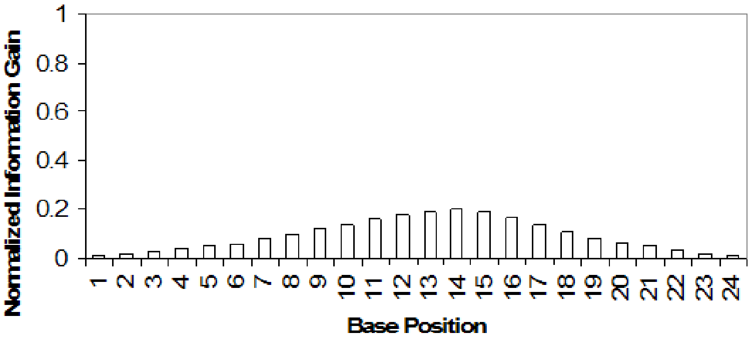


Figure 10.3 **The bases in the probe do not contribute uniformly to the hybridization characteristics.** Their influence seems to be position-dependent. From Tobler et al. (2002), this is a graph of the *normalized information gain* (Shannon, 1948) attributed to the identity of each probe in a 24-base-long probe sequence. We made this graph by dividing a set of probes, who's comparative hybridization efficiency and base sequence are known, into two classes: *good* and *bad* probes. Then we measured the value of each base's identity *A*,*G*,*C*, or *T* in predicting the class of a probe, in terms of *entropy*. For a more detailed explanation of this type of prediction, see Section 3.3.

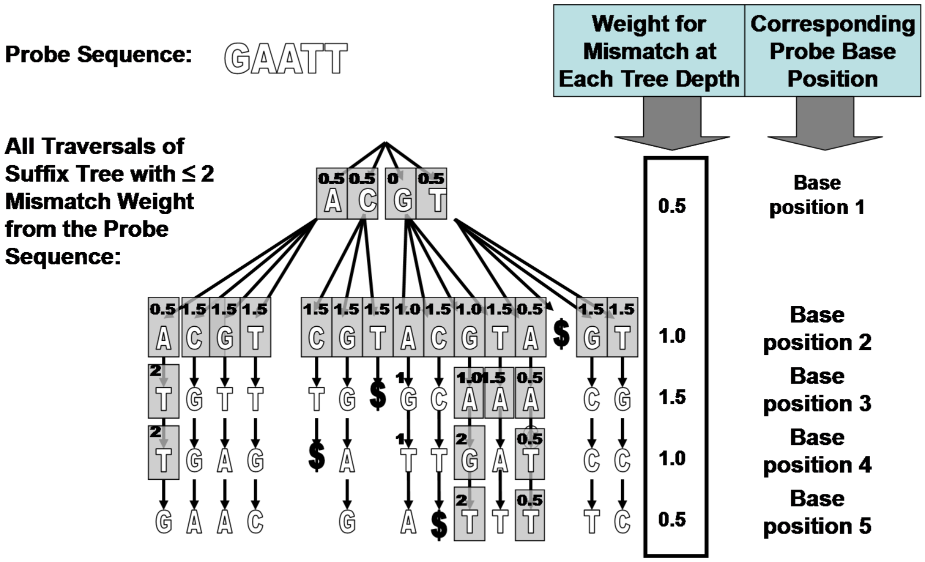

Probe Sequence:
GAATT
Weight for Mismatch at Each Tree Depth
Corresponding Probe Base Position
All Traversals of Suffix Tree with ≤ 2 Mismatch Weight from the Probe Sequence:
0.5 0.5 0 0.5
A C G T
0.5
1.0
1.5
1.0
0.5
Base position 1
Base position 2
Base position 3
Base position 4
Base position 5
$

Figure 10.4 **Instead of simply counting the number of mismatches (up to the threshold value) from every path in the suffix tree (Figure 10.2), we assign a weight to each position in the tree** and add the appropriate weight at each node during the traversal of the tree. It is this accumulated weight that is compared to the threshold value to make a determination with regard to uniqueness. At the right are the weight values for each base position.

, just as is the case with a simple thresholding of the number of mismatches, the time complexity is linear in the length of the probe for each probe being evaluated based. This can be proved in the following way:

- When exploring a path, we only branch in the case that we can *afford* a mismatch. That is, the *mismatch weight* accumulated prior to this point in the tree plus the mismatch weight at this level of the tree is less than or equal to the $threshold$ value.
- By definition, no position in the tree contributes less to total weight than $minWeightBase$.
- So, weight cannot be accumulated any faster than:

  $$minWeightBase(\text{the number of branching events on the path})$$

  .
- Since $threshold$ is the maximum amount of weight that can be accumulated before a path is discarded, a path can have no more than $\frac{threshold}{minWeightBase}$ branching events.
- Since there are only 4 bases to choose from, each branching event will create, at most, 4 new paths. So the breadth of the tree of traversals can never exceed $4^{(\frac{threshold}{minWeightBase})}$; because $\frac{threshold}{minWeightBase}$ is a constant, $4^{\frac{threshold}{minWeightBase}}$ is also a constant. Since the maximum breadth of this traversal tree is constant, the maximum size of this tree is linear in the depth of the traversal tree, which is the same as the length of the probe.

It may be observed that, though the algorithm is linear with regard to the length of the probe, it is exponential with regard to $\frac{threshold}{minWeightBase}$. This value is analogous to the maximum number of mismatches allowed by a simple shreshold in the more typical suffix-tree traversal method. In practice, however, this is not a major issue since, for reasonable values, $threshold$ tends to be exhausted quickly by base positions that exhibit more influence than $minWeightBase$.

This represents an efficiency improvement over the $O(n^2)$ alignment step, which is no longer needed when probe position is taken into account in the suffix-tree phase.

Though a comprehensive study of this approach has not been done, in practice, this method provides an effective substitute to the the time-consuming sequence-alignment step. Since I developed this method in 2001, NimbleGen Systems has been successfully using it as their method to assess probe uniqueness.

10.2 Genetic Simulation for Evolutionary Biology

In collaboration with Eric Haag, an evolutionary biologist at the University of Maryland, I wrote a population-simulation program that we used to simulate a phenomenon known as *compensatory evolution* (Kimura, et al. 1990). We adjusted the parameters to reflect different hypotheses about the phenomenon and, by comparing the results of our simulation to observations made in real organisms, were able to strongly support one hypothesis over the others (Haag and Molla 2005).

10.2.1 Compensatory Evolution

When changes occur in two genes that, together, are beneficial, but would each have been deleterious on their own, this is known as *compensatory evolution*. According to most annalytical models, this should be a very rare event.

Until recently, evolutionary-biology theory held that it could only occur in the presence of either tight linkage between the genes involved, high mutation rates, or very little evolutionary pressure against the deleterious single-gene changes (Crow and Kimura 1965, Gillespie 1984, Kimura 1985, Michalakis and Slatkin 1996, Phillips 1996). However, it has been shown empirically that compensatory evolution does happen quickly in the absence of these circumstances (Swanson and Vacquier 1995, Metz and Palumbi 1996, Hellberg and Vacquier 1999, Kachroo et al. 2001, Haag et al. 2002).

10.2.2 Pseudocompensation

In order to solve this conundrum, Professor Eric Haag developed the theory of *pseudocompensation* (Haag and Molla 2005). His idea was that there may be intermediate forms of the

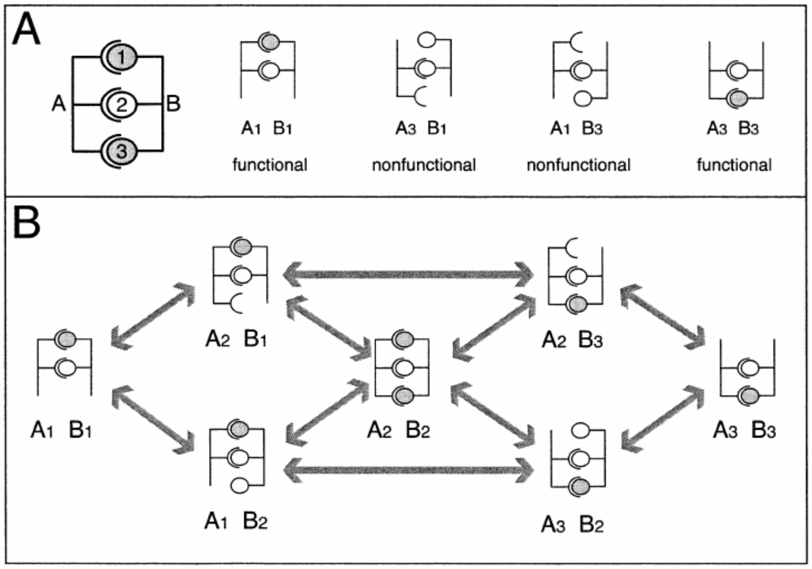


Figure 10.5 **Compensatory versus pseudocompensatory evolution of interacting gene products**, from Haag and Molla (2005). The products of two genes, A and B, interact via multiple bonds, numbered here 1-3. Any one of these bonds is dispensible. In both true compensatory evolution and in pseudocompensation, ancestral alleles allowing interaction via bond 1 could evolve into descendant alleles functioning via bond 3 (both indicated by the shaded ovals). Bond 2 does not change, but is included to represent the multipartite nature of macromolecular interactions. Panel (A) In traditional compensatory evolution, fully functional allele pairs, such as A1B1 or A3B3, are separated mutationally by nonfunctional mismatch genotypes that have only a single bond. With strong selection against mismatched alleles, evolution from A1B1 to A3B3 is possible only under a narrow range of conditions. Panel (B) In pseudocompensation, a multifunctional adaptor allele at each interacting locus (A2 or B2) is posited to exist that has the capacity to interact productively with the products of both ancestral and derived alleles. Since multiple alleles exist for each gene, arrows are included here to represent all possible direct mutational events under pseudocompensation. For simplicity, our models only consider the forward arrows (i.e. the arrows causing allele numbers to ascend rather than descend)

.

genes, known as *multifunctional intermediates*, that mitigate the cost of making the first single-gene change in the compensatory evolution process (Figure 10.5). For example, imagine a fictitious 2-gene organism where gene A can have either allele A_1 or allele A_3 and gene B can have either allele B_1 or allele B_3. The combination A_1B_1 produces a fit organism and the combination A_3B_3 produces an even fitter organism. However, the genotypes A_1B_3 and A_3B_1 are strongly selected against. It is also known that populations do mutate from A_1B_1 to A_3B_3 in far less time than it should take for both genes to, by chance, mutate in the same individual.

Professor Haag's *Pseudocompensation* idea was the idea that there may be an allele A_2 or B_2 that works with either of the other gene's alleles, working as a sort of *stepping stone* between the two genotypes. However, he had no way to test this theory.

10.2.3 Genetic Population Simulation Experiments

So, I developed software that simulated this process. It used parameters to specify the presence or absence of an intermediate allele, selective pressures against the various alleles, mutation rate, and population size. I used this software to simulate the evolution of populations over millions of generations. I used the University of Wisconsin-Madison's CONDOR system to run these simulations, varying the parameters to investigate the likelihood of a compensatory-evolution event under various conditions.

As described in Table 10.1, the first step in each simulation is to calculate the relative fitness of each of the nine (haploid) or 81 (diploid) possible genotypes given the particular fitness model specified. Each generation of the simulation consists of three phases: *mutation*, *survival*, and *mating*. In the mutation phase, each individual in the population is mutated with probability described by the given six forward mutation rates. The resulting population is then subjected to the survival phase in which the quantity of each of the possible genotypes is adjusted directly by its precomputed relative fitness. Finally, in the mating phase, a number of gametes equal to twice the population size are formed by pulling random pairs of A and B alleles (with replacement) from the population pool, and these are then randomly paired to form the next generation. The mutation, survival, and mating phases are then repeated. Note that, though the number of individuals in the population may diverge from the given population size during the survival phase, it will return to its original size during the mating phase. This ensures a population of consistent size at the start of each iteration.

The C program that performs the haploid simulations, synthPopHap, was compiled and run by me on a Sun cluster at the University of Wisconsin, Madison. The diploid version, synthPop, is much slower, and was compiled on a Linux cluster to run up to 50 simultaneous runs with the gracious assistance of Kai Zhang and the University of Maryland Institute for Advanced Computer Studies. Both sets of simulations were run using the Condor Software Program, developed by the Condor Team at the Computer Sciences Department of the University of Wisconsin, Madison (available at http://www.cs.wisc.edu/condor/downloads/v6.6.license.html). For each of the 30 unique combinations of selection strength, fitness model, and population size tested, 20 replicate runs were performed using the same 20 random seeds (1-20). To avoid potential artifacts due to random number generator periodicity, the seeds of simulations that had not reached at least semi-incompatibility by the 10 millionth generation were changed by adding a “1” in front of each (e.g., 13 becomes 113). This was only required for two runs of the 840 performed, both in the haploid adaptive bond gain simulations with $N = 10{,}000$ (seeds 15 and 19). Depending on population size, some diploid runs took as long as several weeks to complete. Data are output as tab-delimited files that note the frequency of each of the possible ordered genotypes at intervals of 200 generations.

10.2.4 Results

The results of this study are described in detail in Haag and Molla (2005). We showed that *pseudocompensation* does indeed explain the apparent disconnect between laboratory results and annalytical results. Figure 10.6 shows the interesting result of one of our experiments. A small number of the B2 allele appear after approximately 14,000 generations and the A3, and subsequently the B3, alleles take over and dominate the population.

Table 10.1 **Pseudocode for the *synthPop* algorithm**

```
Algorithm 10.2.1: SYNTHPOP($populationSize, mutationRates, fitnessModel$)

for each $genotype \in$ the 9 (haploid) or 81 (diploid) possible
    $fitness_{genotype} \leftarrow$ calculate fitness based on $fitnessModel$
$population \leftarrow populationSize$ individuals of genotype $A_1B_1; numGenerations \leftarrow 0$
repeat
    for each $individual \in population$
        MUTATION
            for each $gene \in individual$
                choose a random number, rand, where $0 < rand < 1$
                if $rand <$ the mutation rate for this allele specified in mutationRates
                    change this individual's genotype by mutating this gene
    $newPopulation \leftarrow \emptyset$
    for each $genotype \in$ the 9 (haploid) or 81 (diploid) possible
        SURVIVAL
            $curTotal =$ the number of individuals in populaton with this genotype
            $newTotal \leftarrow curTotal * fitness_{genotype}$
            add $newTotal$ individuals of $genotype$ to $newPopulation$
    $population \leftarrow newPopulation; newPopulation \leftarrow \emptyset$
    for each $i \leftarrow 1 \rightarrow populationSize$
        MATING
            choose, at random, one member of $population$
            record the A allele of this member
            choose, at random, another member of $population$
            record the B allele of this member
            combine the two chosen alleles to form an $individual$
            add this $individuals$ to $newPopulation$
    $population \leftarrow newPopulation; numGenerations++$
until the entire population is $A_3B_3$
return ($numGenerations$)
```

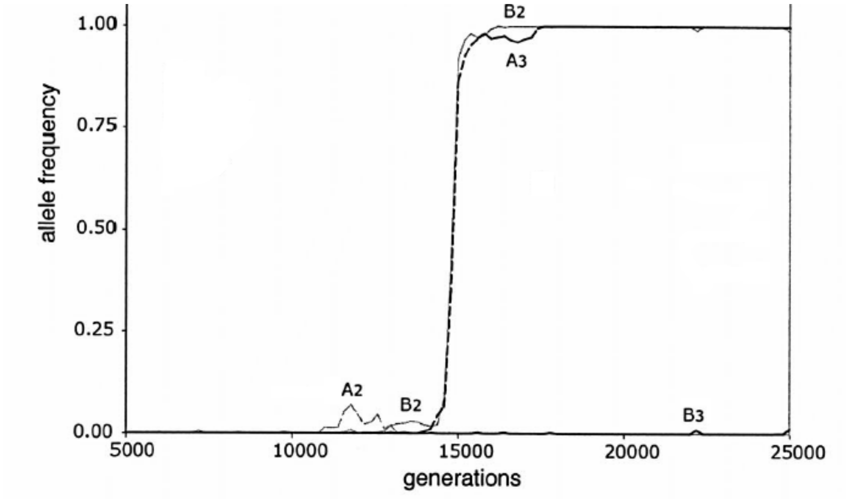


Figure 10.6 **An example of allele frequency dynamics in one of the experiments.** The total population size is normalized to 1.0. For simplicity, the A1 and B1 alleles are not plotted. Their abundance can be inferred from the difference between the other totals and the 1.0 level on the Y-axis. In this experiment, 3-type alleles can rapidly spread when even a small population of the other 2-type allele is present. In this case, a very small proportion of allele B2 has been produced by drift at around generation 14,000. The subsequent arising of the A3 mutation in this context leads to rapid fixation of both alleles.

10.3 discussion

These methods that I developed were both the result of current needs by biologists. The probelms were both interesting and required efficient computational methods to solve. Using sound programming concepts, the computer scientist can make a distinct contribution to the science of biology.

Chapter 11

Conclusion

I have focused on applying machine learning to problems in genetics, contributing to both. I have helped to enhance the strong synergy between the two disciplines. Here I will summarize my major contributions to genetics and to machine learning.

11.1 Contributions to Genetics

I have used machine learning and other computational methods to directly affect the process by which microarrays are designed and used. Most significantly, I have contributed directly to the following major areas of current microarray use.

11.1.1 Automated Analysis of Expression

As also described in Chapter 9 and in Molla et al. (2002), we developed an early system to use textual descriptions of genes and proteins to automatically generate descriptions of the phenomena underlying microarray-expression experiments. We followed this up with studies to actually peruse a published textbook (Ingraham and Neidhardt, 1987) for relevant strings of text and report them to the user, without any human intervention, based on the results of microarray experiments.

11.1.2 SNP Identification

As described in Chapter 7 and Molla et al. (2004), we developed a new method to interpret SNP-finding gene chips. As an alternative to the complex parametric statistical technique requiring a great deal of tuning that had been used previously (Cutler et al. 2001), we developed a machine-learning approach that requires virtually no tuning. This method drastically improved the efficiency of the process, is currently in use at Nimblegen Systems, and has identified thousands of SNPs over the time since it was developed.

11.1.3 Genomic Copy-Number Variation Analysis (CHG)

Human genome copy-number polymorphisms, either amplifications or deletions in human DNA, are often indicative of specific diseases including cancer. The most effective known method for identifying such polymorphisms involves oligonucleotide microarrays and is know as Comparative Genomic Hybridization (CGH). In Chapter 6, I described how we developed a dynamic-programming method for interpreting such an experiment which executes a standard benchmark more effectively and efficiently than any other known method. This method is also in current use at NimbleGen and had been used to identify thousands of such polymorphisms.

11.1.4 Direct Genomic Selection using Gene Chips

In Chapter 5, I described how I helped to pioneer a fourth and completely new use of gene chips. The ability to discover novel variation in a genome is a key challenge in genomics, especially in the human. I played a significant role in the development, with NimbleGen Systems, of a process described in an upcoming paper by which gene chip can be used as a *filter* for genomic DNA, allowing the sequencing of all of exons in any given human genome for well under $1,000.

11.2 Contributions to Machine Learning

In doing this work, I have also helped to advance the science of machine learning. Here I will describe two major contributions.

11.2.1 Text Mining

In addition to the vast amount of data being generated through the use of gene chips, the amount of annotated genomic data available in public databases is also rapidly increasing. Through collaboration with University of Wisconsin's *E. coli* lab, as described in Chapter 9 and in Molla et al. (2002, 2004c) developed automated methods for combining these two sources of information to produce insight into the operation of cells under various conditions. Our approach uses machine-learning techniques to identify words associated with genes that are up-regulated or down-regulated in a particular microarray experiment.

11.2.2 Semi-supervised Learning for Large-Scale Experiments

In order to make use of both labeled and unlabeled examples, one can use a relatively new class of algorithms: semi-supervised learning (Chapelle et al. 2006). This class of learning is particularly useful in domains where unlabeled data are abundant and labeled data are scarce. In Chapter 8, using the insights gained in the SNP-finding project (Chapter 7), I have developed a new algorithm for semi-supervised learning which I have tried, with some promising results, on both biological and non-biological datasets.

11.3 Future Work

Through continued collaboration with biologists, statisticians, and other computer scientists in both academia and private industry, I intend to continue to participate in the design and interpretation of high-throughput experiments, design algorithms to increase both throughput and comprehensibility, and add new dimensions to the automated interpretation of experiments. By doing so, I will not only participate in discovery in field of biology; I will also continue to uncover new computer science principles and techniques that can be of general use.

11.3.1 Data Mining

I plan to use an approach similar to my text-mining projects to combine other sources of data such as protein structures, functional classifications, and other chemical assays. Such methods could also be used for text mining in different experimental domains.

I also plan to improve my methods of text mining to produce results that can not only be comprehended, but can also be evaluated. In retrospect, the most difficult aspect of mining a piece of prose text like a book is the ability to reliably and algorithmically evaluate the results. Once such a method is created, searching the space of explanations for a phenomenon or experimental result can proceed more rapidly.

11.3.2 Semi-Supervised Learning

At present, over a dozen large-scale projects to gather genetic information are underway around the world . Some are sequencing DNA; others are measuring gene reactions to chemical compounds; still others are cataloging genetic variation. One thing that all of the projects have in common, however, is that they are collecting data at a much higher rate than can be thoroughly analyzed by researchers. In most cases, only a small fraction of the data has been closely investigated. This small fraction can be considered labeled data. The rest is unlabeled data. This is the perfect domain for the ongoing development of semi-supervised learning methods.

Empirical study has shown that not all real-world datasets respond well to Chapter 8's pretraining algorithm. This may be because not all real-world datasets contain key features. I would like to develop methods for identifying datasets that have this property in order to exploit it. Another possible reason for this difficulty is that my algorithm needs to be improved. To this end, I have begun to develop an SVM kernel that tries to make use of the *nearby cluster hypothesis* directly.

11.3.3 Microarray Technology

I will also continue to contribute to the science and technology of genetic microarrays. As I have shown most clearly, in chapter 5, with our *genome enrichment* project, this dynamic technology continues to be applied in surprising ways with profound results. I intend to remain on the forefront of this work by creating the means by which new types of gene chips are designed and used.

The ease and scalability of the *genome enrichment* approach show that the method can be adapted for larger fractions of the genome and for analysis of many samples. Current efforts aim to produce a microarray that can capture the whole human exon set for sequencing. Though the probes were made using information gained from my study of probe quality reported in chapter 4 and from Tobler et al. (2002), I am in the process of performing a similar experiment in the context of sequence capture. The longer probes and different goals of this process could have a strong influence on the type of probe that is effective at this job.

11.4 Final Remarks

It is clear that high-throughput techniques, such as rapid DNA sequencing and gene chips are changing the science of genetics. Hypothesis-driven science is now strongly complemented by these newer data-driven approaches. It is well understood that computer science will play a crucial role in their development and application. As I and others have shown, machine learning has been of particular value in this domain.

Bibliography

Aboul-ela, F., D. Koh, I. J. Tinoco, and F. Martin (1985). Base-base mismatches. Thermodynamics of double helix formation for dca3xa3g + dct3yt3g (x, y = a,c,g,t). *Nucleic Acids Research 13*, 4811–4824.

Albert, T., D. Dailidiene, G. Dailide, J. Norton, A. Kalia, T. Richmond, M. Molla, J. Singh, R. Green, and D. Berg (2005). Mutation discovery in bacterial genomes: Metronidazole resistance in H*elicobacter* P*ylori*. *Nature Methods 2*, 951–953.

Altshul, S., W. Gish, W. Miller, E. Myers, and D. Lipman (1990). Basic local alignment tool. *Journal of Molecular Biology 215*, 403–410.

Ananiadou, S. and J. Mcnaught (Eds.) (2005). *Text Mining for Biology And Biomedicine*. Norwood, MA: Artech House Publishers.

Antipova, A., P. Tamayo, and T. Golub (2002). A strategy for oligonucleotide microarray probe reduction. *Genome Biology 3*, research0073.1–research0073.4.

Antson, D., A. Isaksson, U. Landegren, and N. M. (2000). PCR-generated padlock probes detect single nucleotide variation in genomic DNA. *Nucleic Acids Res 28*, E58.

Ashburner, M., C. Ball, J. Blake, D. Botstein, H. Butler, J. Cherry, A. Davis, K. Dolinski, S. Dwight, J. Eppig, D. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. Matese, J. Richardson, M. Ringwald, G. Rubin, and G. Sherlock (2000). Gene Ontology: tool for the unification of biology. *Nature Genetics 25*, 25–29.

Bairoch, A. and R. Apweiler (2000). The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Research 28*, 45–48.

Baldi, P. and S. Brunak (2001). *Bioinformatics: The Machine Learning Approach* (second ed.). Cambridge, MA: MIT Press.

Bashiardes, S., R. Veile, C. Helms, E. Mardis, A. Bowcock, and M. Lovett (2005). Direct genomic selection. *Nature Methods 2*, 63–69.

Blum, A. and P. Langley (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence 97*, 245–271.

Breslauer, K., R. Frank, H. Blocker, and L. Marky (1986). Predicting DNA duplex stability from the base sequence. *Proceedings of the National Academy of Sciences, USA 83*, 3746–3750.

Brown, W., D. Lin, D. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler (2000). Knowledge-based analysis of microarray gene expression data using support vector machines. *Proceedings of the National Academy of Sciences, USA 97*, 262–267.

Chapelle, O., B. Scholkopf, and A. Zien (Eds.) (2006). *Semi-Supervised Learning*. Cambridge, MA: MIT Press.

Collins, F. and A. Barker (2007). Mapping the cancer genome: Pinpointing the genes involved in cancer will help chart a new course across the complex landscape of human malignancies. *Scientific American*.

Cormen, T., C. Leiserson, R. Rivest, and C. Stein (2001). *Introduction to Algorithms* (second ed.). Cambridge, MA: MIT Press.

Craven, M., D. Page, J. Shavlik, J. Bockhorst, and J. Glasner (2000). Using multiple levels of learning and diverse evidence sources to uncover coordinately controlled genes. In *Proceedings of the 17th International Conference on Machine Learning*, Palo Alto, CA. Morgan Kaufmann.

Cristianini, N. and M. Hahn (2006). *Introduction to Computational Genomics*. Cambridge, UK: Cambridge University Press.

Crow, J. and M. Kimura (1965). Evolution in sexual and asexual populations. *The American Naturalist 99*, 439–450.

Cutler, D., M. Zwick, M. Carrasquillo, C. Yohn, P. Tobin, C. Kashuk, D. Mathews, N. Shah, E. Eichler, J. Warrington, and A. Chakravarti1 (2001). High-throughput variation detection and genotyping using microarrays. *Genome Research 11*, 1913–1925.

Davis, J. and M. Goadrich (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA.

Deffernez, C., W. Wunderli, Y. Thomas, S. Yerly, L. Perrin, and L. Kaiser (2005). Amplicon sequencing and improved detection of human rhinovirus in respiratory samples. *Journal of Clinical Microbiology 43*, 3593.

Dudoit, S., Y. Yang, M. Callow, and T. Speed (2000). Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. Technical report, UC-Berkeley Statistics Dept.

Fodor, S., J. Read, M. Pirrung, S. L., A. Lu, and D. Solas (1991). Light-directed, spatially addressable parallel chemical synthesis. *Science 251*, 767–73.

Gilbert, W. and A. Maxam (1973). The nucleotide sequence of the lac operator. *Proceedings of the National Academy Science USA 70*, 3581–3584.

Gillespie, J. (1984). Molecular evolution over the mutational landscape. *Evolution 38*, 1116–1129.

Glenisson, P., J. Mathys, and B. De Moor (2003). Meta-clustering of gene expression data and literature-based information. *SIGKDD Explorations 5*, 101–112.

Good, P. (2001). *Resampling Methods A Practical Guide to Data Analysis* (second ed.). Boston: Birkhduser.

Haag, E. and M. Molla (2005). Compensatory evolution of interacting gene products through multifunctional intermediates. *Evolution 59*, 1620–1632.

Haag, E., S. Wang, and J. Kimble (2002). Rapid coevolution of the nematode sex-determining genes fem-3 and tra-2. *Current Biology 12*, 2035–2041.

Hacia, J. (1999). Resequencing and mutational analysis using oligonucleotide microarrays. *Nature Genetics 21*, (1 Suppl):42–7.

Hanisch, D., A. Zien, R. Zimmer, and T. Lengauer (2002). Co-clustering of biological networks and gene expression data. *Bioinformatics 18*, Suppl.1, S145–S1554.

Hanson, B. and R. Coontz (2001). Introduction to the special issue: A computer science odyssey. *Science Special Issue: Computers and Science 293*, 2021.

Hearst, M. (1999). Untangling text data mining. College Park, MD, pp. 443–449.

Hellberg, M. and V. Vacquier (1999). Rapid evolution of fertilization selectivity and lysin cDNA sequences in teguline gastropods. *Molecular Biology and Evolution 16*, 839–848.

Herring, C., A. Raghunathan, C. Honisch, T. Patel, K. Applebee, A. Joyce, T. Albert, F. Blattner, D. van den Boom, C. Cantor, and B. Palsson (2006). Comparative genome sequencing of E*scherichia coli* allows observation of bacterial evolution on a laboratory timescale. *Nature Genetics 38*, 1406 – 1412.

Hirschhorn, J., P. Sklar, K. Lindblad-Toh, Y. Lim, M. Ruiz-Gutierrez, S. Bolk, B. Langhorst, S. Schaffner, E. Winchester, and E. Lander (2000). SBE-TAGS: An array-based method for efficient single-nucleotide polymorphism genotyping. *Proceedings of the National Academy of Sciences, USA 97*, 12164–12169.

Huber, W., J. Toedling, and L. Steinmetz (2006). Gene expression transcript mapping with high-density oligonucleotide tiling arrays. *Bioinformatics 22*, 1963–1970.

Hvidsten, T., A. Lgreid, and J. Komorowski (2003). Learning rule-based models of biological process from gene expression time profiles using gene ontology. *Bioinformatics 19*, 1116–1123.

Iafrate, A., L. Feuk, M. Rivera, M. Listewnik, P. Donahoe, Y. Qi, S. Scherer, and C. Lee (2004). Detection of large-scale variation in the human genome. *Nature Genetics 36*, 945–951.

Jenssen, T.-K., A. Lgreid, J. Komorowski, and E. Hovig (2001). A literature network of human genes for high-throughput gene-expression analysis. *Nature Genetics 28*, 21–28.

Kachroo, A., C. Schopfer, M. Nasrallah, and J. Nasrallah (2001). Allele-specific receptor-ligand interactions in brassica self-incompatibility. *Science 293*, 1824–1826.

Kaderali, L. and A. Schliep (2002). Selecting signature oligonucleotides to identify organisms using DNA arrays. *Bioinformatics 18*, 1340–1349.

Kane, S., A. Chakicherla, S. Patrick, P. Chain, R. Schmidt, M. Shin, T. Legler, K. Scow, F. Larimer, S. Lucas, P. Richardson, and K. Hristova (2007). Whole-genome analysis of the methyl tert-butyl ether-degrading beta-proteobacterium M*ethylibium petroleiphilum* PM1. *Journal of Bacteriology 189*, 1931–45.

Kimura, M. (1985). Compensatory neutral mutations in molecular evolution. *Genetica 64*, 7–19.

Kurtz, A., J. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Geigerich (2001). Reputer: The manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Research 29*, 4633–4642.

Kutara, K. and A. Suyama (1999). Probe design for DNA chips. *Journal of the Japanese Society for Bioinformatics 10*, 225–226.

Li, C. and W. Wong (2001.). Model-based analysis of oligonucleotide arrays: Expression index computation and outlier detection. *Proceedings of the National Academy of Sciences, USA 98*, 31–36.

Li, F. and G. Stormo (2001). Selection of optimal DNA oligos for gene expression arrays. *Bioinformatics 17*, 1067–1076.

Lindman, H. (1974). *Analysis of Variance in Complex Experimental Designs*. San Francisco: W. H. Freeman and Co.

Lipson, D., Y. Aumann, A. Ben-Dor, N. Linial, and Z. Yakhini (2006). Efficient calculation of interval scores for DNA copy number data analysis. *Journal of Computational Biology, 13*, 215–228.

Lockhart, D., H. Dong, M. Byrne, M. Follettie, M. Gallo, M. S. Chee, M. Mittmann, C. Wang, M. Kobayashi, H. Horton, and E. Brown (1996). Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology 14*, 1675–80.

Masys, D., J. Welsh, J. Fink, M. Gribskov, I. Klacansky, and J. Corbeil (2001). Use of keyword hierarchies to interpret gene expression patterns. *Bioinformatics 17*, 319–326.

Metz, E. and S. Palumbi (1996). Positive selection and sequence rearrangements generate extensive polymorphism in the gamete recognition protein bindin. *Molecular Biology and Evolution 13*, 397–406.

Michalakis, Y. and M. Slatkin (1996). Interaction of selection and recombination in the fixation of negative-epistatic genes. *Genetical Research 67*, 257–269.

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

Molla, M., P. Andreae, J. Glasner, F. Blattner, and J. Shavlik (2002). Interpreting microarray expression data using text annotating the genes. *Information Sciences 146*, 75–88. Also appears in: Proceedings of the 4th Conference on Computational Biology and Genome Informatics, Durham, NC.

Molla, M., P. Andreae, and J. Shavlik (2004). Building genome expression models using microarray expression data and text. Machine Learning Research Group Working Paper 04-1, Department of Computer Sciences, University of Wisconsin.

Molla, M., J. Shavlik, T. Albert, T. Richmond, and S. Smith (2004). A self-tuning method for one-chip SNP identification. In *Proceedings of the IEEE Conference on Computational Systems Bioinformatics*, Stanford, CA, pp. 69–79.

Molla, M., M. Waddell, D. Page, and J. Shavlik (2004). Using machine learning to design and interpret gene-expression microarrays. *AI Magazine 25*(1), 23–44. Appears in the Special Issue on Bioinformatics.

Mooney, R. (1995). Encouraging experimental results on learning CNF. *Machine Learning 19*, 79–92.

Newton, M., C. Kendziorski, C. Richmond, F. Blattner, and K. Tsui (2001). On differential variability of expression ratios: Improving statistical inference about gene expression changes from microarray data. *Journal of Computational Biology 8*, 37–52.

Nuwaysir, E., W. Huang, T. Albert, J. Singh, K. Nuwaysir, A. Pitas, T. Richmond, T. Gorski, J. Berg, J. Ballin, M. McCormick, J. Norton, T. Pollock, T. Sumwalt, L. Butcher, D. Porter, M. Molla, C. Hall, F. Blattner, M. Sussman, R. Wallace, F. Cerrina, and R. Green (2002). Gene expression analysis using oligonucleotide arrays produced by maskless photolithography. *Genome Research 12*, 1749–1755.

Olshen, A., E. Venkatraman, R. Lucito, and M. Wigler (2004). Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics 5*, 557–572.

Phillips, P. (1996.). Waiting for a compensatory mutation: Phase zero of the shifting-balance process. *Genetical Research 67*, 271–283.

Picard, F., S. Robin, M. Lavielle, C. Vaisse, and J. Daudin (2005). A statistical approach for array CGH data analysis. *BMC Bioinformatics 6*.

Pollack, J., C. Perou, A. Alizadeh, M. Eisen, A. Pergamenschikov, C. Williams, S. Jeffrey, D. Botstein, and P. Brown (1999). Genome-wide analysis of DNA copy-number changes using cDNA microarrays. *Nature Genetics*, 41–46.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning 1*, 81–106.

Quinlan, J. (1990). Learning logical descriptions from relations. *Machine Learning 5*, 239–266.

Quinlan, J. (1996). *C4.5: Programs for Machine Learning*. San Mateo, CA.: Morgan Kaufman.

Raychaudhuri, S. and R. Altman (2003). A literature-based method for assessing the functional coherence of a gene group. *Bioinformatics 19*, 396–401.

Ruan, Y., C. Wei, A. Ee, V. Vega, H. Thoreau, S. Su, J. Chia, P. Ng, K. Chiu, L. Lim, T. Zhang, C. Peng, E. Lin, N. Lee, S. Yee, L. Ng, R. Chee, L. Stanton, P. Long, and L. E. (2003). Comparative full-length genome sequence analysis of 14 SARS coronavirus isolates and common mutations associated with putative origins of infection. *Lancet 361*, 1779–1785.

Rumelhart, D. and R. Williams (1986). *Learning Internal Representations by Error Propagation*, Volume 1, pp. 318–362. Cambridge, MA: MIT Press. From *Parallel Distributed Processing: Exploration in the Microstructure of Cognition vol. 1: Foundations*, MIT Press, Cambridge, MA. 318-362.

Saiki, R., P. Walsh, C. Levenson, and H. Erlich (1989). Genetic analysis of amplified DNA with immobilized sequence-specific oligonucleotide probes. *Proceedings of the National Academy of Sciences, USA 86*, 6230–6234.

Sarle, W. (1995). Stopped training and other remedies for overfitting. *Proceedings of the 27th Symposium of the Interface of Computing Science and Statistics*, 352–360.

Schena, M., D. Shalon, R. Davis, and P. Brown (1995). Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science 270*, 467–70.

Scholkopf, B., J. Burges, and A. Smola (Eds.) (1999). *Advances in Kernel Methods: Support Vector Learning*. Cambridge, MA: MIT Press.

Segal, E., B. Taskar, A. Gasch, N. Friedman, and D. Koller (2001). Rich probabilistic models for gene expression. *Bioinformatics 1*, 1–10.

Shannon, C. (1948). A mathematical theory of communication. Bell System Technical Journal.

Shrager, J., P. Langley, and A. Pohorille (2002). Guiding revision of regulatory models with expression data. *Proceedings of the Pacific Symposium on Biocomputing*, 486–497.

Singh-Gasson, S., R. Green, Y. Yue, C. Nelson, F. Blattner, M. Sussman, and F. Cerrina (1999). Maskless fabrication of light-directed oligonucleotide microarrays using a digital micromirror array. *Nature Biotechnology 17*, 974–978.

Steemers, F., W. Chang, G. Lee, D. Barker, R. Shen, and K. Gunderson (2006). Whole-genome genotyping with the single-base extension assay. *Nature Methods 3*, 31–33.

Swanson, W. and V. Vacquier (1995). Extraordinary divergence and positive Darwinian selection in a fusagenic protein coating the acrosomal process of abalone spermatozoa. *Proceedings of the National Academy of Sciences, USA 92*, 4957–4961.

Tai, A., W. Mak, P. Ng, D. Chua, M. Ng, L. Fu, K. Chu, Y. Fang, Y. Qiang Song, M. Chen, M. Zhang, P. Sham, and X. Guan (2006). High-throughput loss-of-heterozygosity study of chromosome 3p in lung cancer using single-nucleotide polymorphism markers. *Cancer Research 66*, 4133–4138.

Tobler, J., M. Molla, E. Nuwaysir, R. Green, and J. Shavlik (2002). Evaluating machine learning approaches for aiding probe selection for gene-expression arrays. *Bioinformatics, Special Issue Based on the Papers Presented at the Tenth International Conference on Intelligent Systems for Molecular Biology 18*, S164–S171.

Wang, D., J. Fan, C. Siao, A. Berno, P. Young, R. Sapolsky, G. Ghandour, N. Preking, E. Winchester, J. Spencer, L. Kruglyak, L. Stein, L. Hsie, T. Topaloglou, E. Hubbell, E. Robinson, M. Mittmann, M. Morris, N. Shen, D. Kilburn, J. Rioux, C. Nusbaum, S. Rozen, T. Hudson, R. Lipshutz, M. Chee, and E. Lander (1998). Large-scale identification, mapping, and genotyping of single-nucleotide polymorphisms in the human genome. *Science 280*, 1077–1082.

Wong, C., T. Albert, V. Vega, J. Norton, D. Cutler, T. Richmond, L. Stanton, E. Liu, and L. Miller (2004). Tracking the evolution of the SARS coronavirus using high-throughput, high-density resequencing arrays. *Genome Research 14*, 398–405.

Xing, E., M. Jordan, and R. Karp (2001). Feature selection for high-dimensional genomic microarray data. *Proceedings of the 18th International Conference on Machine Learning*, 601–608.