INS 6632
12 August 2002 Type
ARTICLE IN PRESS
No. of Pages 14, DTD = 4.3.1
SPS, Chennai

ELSEVIER
Information Sciences xxx (2002) xxx–xxx
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Interpreting microarray expression data using text annotating the genes

Michael Molla [a,*], Peter Andreae [a,1], Jeremy Glasner [b], Frederick Blattner [b], Jude Shavlik [a,c]

[a] *Department of Computer Sciences, University of Wisconsin—Madison, Madison, WI, USA*
[b] *Department of Genetics, University of Wisconsin—Madison, Madison, WI, USA*
[c] *Department of Biostatistics and Medical Informatics, University of Wisconsin—Madison, Madison, WI, USA*

Received 1 June 2001; received in revised form 2 November 2001; accepted 1 February 2002

Abstract

Microarray expression data is being generated by the gigabyte all over the world with undoubted exponential increases to come. Annotated genomic data is also rapidly pouring into public databases. Our goal is to develop automated ways of combining these two sources of information to produce insight into the operation of cells under various conditions. Our approach is to use machine-learning techniques to identify characteristics of genes that are up-regulated or down-regulated in a particular microarray experiment. We seek models that are (a) accurate, (b) easy to interpret, and (c) stable to small variations in the training data. This paper explores the effectiveness of two standard machine-learning algorithms for this task: Naïve Bayes (based on probability) and PFoil (based on building rules). Although we do not anticipate using our learned models to predict expression levels of genes, we cast the task in a predictive framework, and evaluate the quality of the models in terms of their predictive power on genes held out from the training. The paper reports on experiments using actual *E. coli* microarray data, discussing the strengths and weaknesses of the two algorithms and demonstrating the trade-offs between accuracy, comprehensibility, and stability.
© 2002 Published by Elsevier Science Inc.

* Corresponding author.
E-mail addresses: molla@cs.wisc.edu (M. Molla), pondy@cs.wisc.edu (P. Andreae), jeremy@genome.wisc.edu (J. Glasner), fred@genome.wisc.edu (F. Blattner), shavlik@cs.wisc.edu (J. Shavlik).
[1] On leave from Victoria University of Wellington, NZ.

1. Introduction

RNA is the medium by which an organism's genes produce specific proteins, which are the building blocks of life. An understanding of how an organism regulates the production of specific RNA sequences is crucial to an understanding of the mechanism by which that organism functions. The *expression level* of a gene is a measure of the amount of RNA being produced by that gene at a particular time. *Microarrays* are a way to quickly and inexpensively measure the expression levels of thousands of genes simultaneously. Microarrays employ fluorescently labeled fragments of RNA that bind to known locations on the microarray's surface. A scanning laser measures the intensity of fluorescence at each point on that surface. The levels of expression of specific RNAs can be inferred from the intensity values measured by the laser. Microarrays are often used to measure expression levels before and after a specific physical event to see which genes changed their expression levels in response to that event. Genes whose expression levels increased are said to be *up-regulated*, while those whose levels decreased are said to be *down-regulated*.

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. We present a method for creating interpretations of microarray experiments that combines 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 (a) accurate, (b) comprehensible, and (c) stable to small changes in the microarray experiment.

In order to make them comprehensible, our models are expressed in terms of English words from text descriptions of individual genes. We currently get these descriptions from the curated SwissProt protein database [1]. It contains annotations of proteins; we use the text associated with the protein generated by a gene as the description of that gene. Our models consist of sets of words from these descriptions that characterize the up-regulated or down-regulated genes. Note that we 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.

Our work is related to several recent attempts to use machine learning to predict gene-regulation levels (e.g., [2,4,12]), but our focus is different in that our goal is not to *predict* gene-regulation levels, but to automatically generate human-readable characterizations of the up- or down-regulated genes to help a human scientist generate hypotheses explaining an experiment.

A key aspect of our approach for interpreting microarray experiments is that we evaluate the accuracy of potential models of the experimental data by using a statistical technique called *cross-validation* (described later). Although it is not our goal to make predictions about new genes, one of our criteria for what makes a good explanatory model is that if some of the genes' expression levels had not been measured in the experiment, then our model would have accurately predicted these expression levels.

We investigate two standard and successful algorithms from the machine-learning literature, evaluating how well they satisfy our three desiderata of accuracy, comprehensibility, and stability. We also present and evaluate some variants and combinations of these two approaches. One standard approach we investigate is Naïve Bayes [9], which is based on probability; the other is PFoil (Mooney, 1995), a rule learner based on propositional logic. Naïve Bayes has proven successful on a wide range of text problems [7] but produces models that are not particularly human-readable, while a major strength of rule learners is that they do tend to produce human-readable models.

Before describing the learning algorithms we evaluate, we first describe the basic task that we assign to the learners. The input to each learner consists of two parts: (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) the SwissProt text describing the protein produced by each gene on the microarray. The output of each learner is a model that accurately characterizes the genes that were up-regulated (or down-regulated) in response to the event.

In our current set of experiments, we consider a gene up-regulated if its ratio of RNA_{after} to RNA_{before} (the gene's expression ratio) is greater than 2; if this ratio is less than $1/2$ we consider it down-regulated. As is commonly done, we currently discard as ambiguous all genes whose expression ratio is between $1/2$ and 2.

In cross-validation, some (e.g., 80%) of the gene-regulation examples are used as the training data for a learning algorithm, while the remaining ("held aside") examples are used to estimate the 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.

Our second desired property is human *comprehensibility*. This is very difficult to measure and we use a crude approximation by counting the number of distinct SwissProt words appearing in a given model.

Our third important property is *stability*. Ideally, the models learned would not change much if the gene-array experiment were repeated under slightly different conditions. We do not currently have actual replicates of a given microarray experiment, so we instead measure stability by repeatedly creating models from random samples of the genes and then comparing these models to each other. For example, if a microarray contains 1000 genes, we would several times randomly select 900 of these genes, and then learn a model for each random sample. We then apply our stability measure (defined later) to this set of learned models.

The next section presents the machine-learning algorithms we investigate in our experiments. Section 3 further explains our experimental methodology and Section 4 presents and discusses experimental results obtained using data from the Blattner *E. coli* Laboratory at the University of Wisconsin. Section 6 describes some related work. The final section describes some of our planned follow-up research and summarizes the lessons learned so far in our work to create a tool that uses English-text protein annotations to assist in the interpretation of microarray experiments.

2. Algorithm descriptions

This section describes the two algorithms—Naïve Bayes [9] and PFoil (Mooney, 1995)—and the variants that we are exploring. Both algorithms take as input a collection of training instances (in our case, genes), labeled as belonging to one of two classes (which we will call up and down), and described by a vector of Boolean-valued features, *W*. 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 an unlabeled gene on the basis of its feature values (i.e., the words describing it).

2.1. Naïve Bayes

The Naïve Bayes algorithm works by computing estimates of conditional probabilities from the input data, and using those probabilities in Bayes' rule to determine the most probable class of an unlabeled instance. The algorithm counts the occurrences of words in all the instances of the positive and negative classes to compute four probabilities for each word. These probabilities are combined into two likelihood ratios for each word that represent the significance of the presence or the absence of that word:

$$\mathrm{lr}_i(w_i \text{ present}) = p(w_i \text{ present}|\text{up})/p(w_i \text{ present}|\text{down})$$
$$\mathrm{lr}_i(w_i \text{ absent}) = p(w_i \text{ present}|\text{up})/p(w_i \text{ absent}|\text{down})$$

It uses a standard Laplace estimator to deal with infrequent words; that is, 1 is added to each numerator and denominator used to estimate probabilities, thereby preventing estimating probabilities of zero.

The algorithm will classify an unlabeled instance (gene) as an up instance if

$$\prod_i \mathrm{lr}_i(w_i \text{ is } X)\frac{p(\text{up})}{p(\text{down})} > 1$$

where w_i is the ith word in the vocabulary and X is either *present* or *absent* depending on whether the word is present or absent in the instance. The algorithm is called "naïve" because we ignore any dependencies among the words when calculating these probabilities. This assumption of independence (of the features given the class) has proven effective in many practical problems, especially those involving English words as features [9].

A major problem with the Naïve Bayes classifier is the lack of comprehensibility of its characterization of the positive class: for a vocabulary of 5000 words, the characterization involves 10,000 likelihood ratios. The variant of Naïve Bayes that we explore in our experiments improves the comprehensibility by pruning the characterization to just the most significant words and their likelihood ratios. Note that this pruning could reduce the accuracy; the experiments described in Section 4 explore the trade-off between accuracy and comprehensibility.

Our Pruned Naïve Bayes algorithm has two phases:

1. Construct a subset of the vocabulary consisting of the n words with the highest score according to an *information-gain* [9] measure.
2. Prune this subset further by incrementally removing words one at a time.

The result is a characterization of the class in terms of a much smaller set of words. A pseudocode version of the algorithm is given in Table 1.

The information-gain measure used in the first phase is the standard measure of the effectiveness of a word at separating the up and down classes, as used in many decision-tree algorithms [9]. All the words are scored according to their information-gain, and the n words with the highest scores are selected.

Table 1
Pruned Naïve Bayes algorithm

```
Disjointly split data set into training and testing sets.
Disjointly subdivide training set into train' set and tuning set.
For each word $w_i$ in the train' set: Compute $InfoGain_i$
Set $W$ to the set of $n$ words with the largest $InfoGain_i$
While $W$ is not empty
  For each $w_i \in W$
    Remove $w_i$ from $W$
    Construct Naïve Bayes classifier on train', using only words in $W$
    Evaluate classifier accuracy on tuning set
    Return $w_i$ to $W$
  Remove from $W$ the $w_i$ that resulted in the best tuning-set accuracy
  Report current $W$ and tuning and testing set accuracies of classifier
```

For the second phase of the Pruned Naïve Bayes algorithm, the training set is separated into a train′ set and a tuning set (we use 25% of the data for the tuning set). To choose a word to prune, the algorithm constructs each subset that can be obtained by removing just one word from the current subset of words and generates a Naïve Bayes classifier on the train′ set with the instance descriptions restricted to remaining words, and determines the average accuracy of the resulting classifier on the tuning set. On each iteration, it chooses the word that results in the most accurate classifier and prunes it from the current subset. Since we are interested in the trade-off between accuracy and comprehensibility, the algorithm reports the current subset and its accuracy on the test set at each iteration, until the subset is empty.

2.2. *PFoil*

PFoil (Mooney, 1995) is a prepositional version of Foil [11], 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 literals 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 our domain, a rule specifies words that must or must not be present in a gene's annotation.

PFoil builds a rule set by constructing one rule at a time. It constructs each rule by adding one feature 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 the positive instances covered by the rule from the data set, and starts to build a new 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:

$FoilGain(rule, f) = p[\log(p/(p+n)) - \log(P/(P+N))]$ 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 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

Table 2
PFOIL with pruning

```
Disjointly split data set into training and test sets.
Disjointly subdivide training set into train′ and tuning sets.
Construct PFOIL rule set that completely fits train′.
While the rule set is not empty
  For each feature in each rule in current rule set
    Temporarily remove feature from rule.
    Evaluate rule set accuracy on tuning set.
    Return the feature to the current rule set.
  Remove from rule set the feature that resulted in the best accuracy.
  Remove any empty rule from the rule set.
  Report current rule set and its accuracy on train′, tuning, and test sets.
```

be very specific to particular instances. Even though individual PFOIL rules are more comprehensible than the vectors of likelihood ratios produced by Naïve Bayes, a large rule set is not particularly comprehensible.

To address this problem, we have extended PFOIL to include a rule pruning stage, along the lines of the pruning in FOIL (Table 2).

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. As in the pruning version of Naïve Bayes, we separate the training data set into a train′ set and a tuning set, using the tuning set to evaluate the predictive accuracy of the rules learned on the train′ set. Rather than halting the pruning when the tuning-set accuracy peaks, in our experiments we continue the pruning until the rule set is empty in order to explore the trade-off between comprehensibility and accuracy.

2.3. Improving stability of PFOIL

The Naïve Bayes algorithm is fairly stable to small variations in the data since it is based on estimated probabilities, but PFOIL—being a greedy, rule-based algorithm—is not so stable. The set of rules it produces can differ considerably with just a small change in the data. We have constructed a variant of PFOIL that restricts the set of features from which it will build its rules in order to encourage greater stability. The algorithm first runs PFOIL on a number of random subsets of the training set and produces a set of rules for each subset. It collects those words that are used in at least *m* of these rule sets. It then reruns PFOIL (with rule-set pruning) on the whole training set, but restricted to building rules containing only words found in the first stage. The pseudocode is given in Table 3.

Table 3
Pseudocode of stabilized PFoil

```
Five times disjointly split data set into training and test sets.
Generate Set of Words for Training Set i:
  Initialize counts on each word to zero.
  Repeat n times
    Choose a random 80% of training set for the train' set.
    Run PFoil on train' set (with no pruning).
    Increment count on each word used in any rule in rule set.
  Set W to the words with count > m
Generate Rule sets Using Only the Words in W:
  Disjointly split training set into train' and tuning sets.
  Run PFoil (with pruning) on train' set, restricted to words in W.
  Record rule set and accuracy on test set.
Compute stability and average accuracy of rule sets across all five folds.
```

To evaluate this algorithm, we do a fivefold cross-validation and compute a measure of the similarity of the rule sets from each of the five folds. We use a coarse measure that only considers the words used in the rule sets and ignores the structure of the rules. The measure is a root-mean-square (RMS) average of the number of times a word is duplicated across the five rule sets:

$$\text{stability} = \sqrt{\frac{\sum_{w_i} \in U(\text{count}(w_i) - 1)^2}{|U|} \frac{1}{N-1}}$$

where U is the set containing those words that appear in any of the rule sets, N is the number of rule sets, and $\text{count}(w_i)$ is the number of rule sets in which the word W_i appears. Using an RMS average weights words that occur in more data sets more heavily.

For example, if all N sets of words were identical, $\text{count}(w_i)$ would equal N for each $w_i \in U$. So, stability would be 1. If the N sets were completely disjoint, $\text{count}(w_i)$ would equal 1 for each $w_i \in U$. In this case, stability would be 0.

3. Experimental methodology

The data we are using are from microarray experiments performed by the Blattner *E. coli* Sequencing Laboratory at the University of Wisconsin. In our current computational experiments, we use the biological event of antibiotic treatment; the "after" gene expressions are measured 20 min after treatment. We use fivefold cross-validation, five times training on 80% of the data and then evaluating our models on the "held aside" 20%. We use all of 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), 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*).

As is common when using text, we convert all words in the annotations to their root form by using the Porter [10] stemming algorithm. Brackets ("[ ]") are used in this paper to signify that a word has been stemmed. For example: residu[al|e] refers to both residual and residue. Other than the Porter stemmer, we implemented the algorithms ourselves in C.

4. Experimental results and discussion

The first set of experiments explores the trade-off between accuracy and comprehensibility of the two algorithms. Fig. 1 shows the result of a fivefold cross-validation experiment running PFoil and Naïve Bayes, both with pruning. The average error rate is plotted against the number of words left in the model. The baseline error rate for this data set (predicting the most frequent class) is 38%. Both algorithms perform reasonably well. Naïve Bayes is consistently better than PFoil and both are consistently better than the baseline.

After a brief initial decline the PFoil error rate increases as words are pruned from the rule set, but the increase is surprisingly slight until only four words remain. Unexpectedly, over most of the experiment, pruning words from the Naïve Bayes model decreases the error rate to a minimum at 23 words. The error rate then increases again. The error rate of unpruned Naïve Bayes, i.e., using all of the SwissProt words, is only 12%—the higher error rate (26%) at $x = 100$ in the graph is the result of the initial pruning phase, where only the 100 most informative features are used. For both PFoil and Naïve Bayes, the accuracy is still high for models that have been pruned to a very comprehensible size.

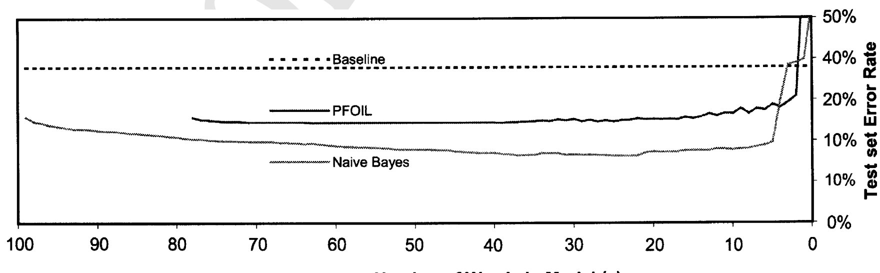

Fig. 1. Test set error as a function of model size.

Table 4
Sample small models of the experimental data

PFoil (disjunctive rules for up)		
Rule 1	NOT map	AND
	NOT biosynthesis	AND
	NOT proteins	AND
	NOT gene	
Rule 2	NOT biosynthesis	AND
	NOT map	AND
	NOT with	AND
	NOT encoded	
Rule 3	Hypothetical	
Rule 4	570-kb	
Naïve Bayes		
Word (*w*)	lr (*w* present)	lr (*w* absent)
Flagellar	0.07	1.09
Deriv[ative|ation]	0.09	1.03
Ubiquinone	0.09	1.03
Nuo	0.09	1.03
Mitochondri[a|al]	0.12	1.04
Aspart[ate|yl]	0.14	1.04
Nadh	0.15	1.03
Complex[|es|ed]	0.19	1.15
Biosynthesis	0.29	1.05
Permeas[e]	0.31	1.05

Table 4 shows sample models from both PFoil and Naïve Bayes. The PFoil section contains the disjunctive rules. The Naïve Bayes section shows the likelihood ratios. A high likelihood ratio for present or absent suggests that the word's presence or absence, respectively, is associated with a gene being up-regulated. Low likelihood ratios are associated with a gene being down-regulated. The models shown were chosen when the pruning had reduced the models to 10 words. They share only one word, which might seem surprising given that they are characterizing the same experiment. That word, "biosynthesis", likely reflects an underrepresentation of basic biosynthetic genes among those up-regulated. Note also that the PFoil rules are dominated by words being absent, whereas the Naïve Bayes model contains no words with a high likelihood ratio for the word being absent. It appears that the two algorithms find quite different kinds of characterizations of the data.

The second set of experiments explored the trade-off between accuracy and stability for the PFoil algorithm. Fig. 2 shows the averaged fivefold result of running Stabilized PFoil on various values of m between 1 and 50 on the same data set as Fig. 1; $m = 1$ represents the least constrained—all words that oc-

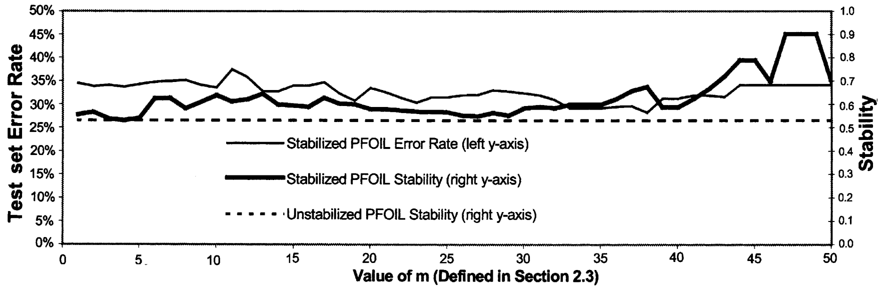

Fig. 2. Trade-off between rule set stability and test set error eate for PFoil.

curred in a rule set from any of the 50 runs (an average of 396 words) were available for the second phase; $m = 50$ represents the most constrained—only words that occurred in all 50 rule sets were available for the second phase. The stability measure represents the most constraint—only words that occurred in all 50 rule sets were available for the second phase. The stability measure represents the similarity of the five rule sets. The maximum observed stability (0.9) could result from 55% of the words having appeared in all five rule sets and the rest appearing in four of the five rule sets. The minimum stability (0.4) could result from half of the words appearing in three of the five rule sets and the other half appearing in only two.

The stability of the rule sets increased significantly as the number of words available was reduced. Stabilized PFoil is also clearly more stable than the original version. The error rate does not change very much as a result of the stabilizing algorithm. However, the error rate is considerably worse than for the unstabilized version, which suggests that forcing stability has a significant accuracy cost.

5. Discussion

The pruning experiments suggest that, using our greedy algorithm, the trade-off between accuracy and comprehensibility is not very severe. In fact decreasing the rule set size can be beneficial and does not significantly hamper accuracy until well into the range of comprehensible rules (<10 words). The information-gain-based pruning, however, that we use to prune the Naïve Bayes model from thousands of words to one hundred words does seem to have an adverse affect: The error rate increases from 12% to nearly 26% over the course of that pruning but drops back down to 16% with greedy pruning.

The fact that PFoil's error rate is higher than that of Naïve Bayes suggests that PFoil is not effective at discovering the real regularities in the gene data. We speculate that this is because PFoil's hill climbing search strategy depends on evaluating the benefit of adding features one at a time. Dependencies between features cause valleys in the search space that mislead PFOIL. A direction for future work is to explore ways of smoothing the search space.

Naïve Bayes does not even attempt to identify dependencies between the features, treating each feature as completely independent. Its model, even when pruned to a manageable number of words, is not helpful in identifying dependencies. In the model in Table 4, both Permeas[e] and Flagellar are strongly correlated with up-regulated genes, but the model does not say, for example, whether the genes characterized by Permeas[e] are the same or different from the genes characterized by Flagellar. In spite of this, the Naïve Bayes model consistently gets a higher accuracy than the PFOIL models. We speculate that one reason is that the PFOIL rules are "crisp" in the sense that an instance must have every feature specified in the rule to satisfy the rule, whereas the Naïve Bayes model is "softer" and allows a range of alternative features to contribute to the probability of being up-regulated. A direction for future work is to explore the use of rule-building algorithms that allow softer rules, for example, allowing *M* of *N* rules [3].

As noted above, the two algorithms produce very different characterizations of the data. This is partly due to the different "default rules" in the algorithms. When attempting to characterize the *up* genes, PFOIL's default is to classify any gene not covered by rule as a *down*. One possible explanation for the abundance of NOT's in the rules is mat PFOIL attempts to find rules that exclude down genes, which is most easily achieved by excluding words that are common in down genes. Pruned Naïve Bayes, on the other hand, has a bias towards the larger class so its default in this data set is the up genes. When pruning, there is little advantage in keeping words that pick out up genes, since it tends to classify them correctly. Like PFOIL, it retains words that will pick out the down genes, but only when they are very infrequent in the up genes. That means that lr_i(present) will be much smaller than 1.

We note also that although Naïve Bayes does not take dependencies between words into account, the pruning will tend to retain only one of a group of dependent words, since there will generally be little accuracy advantage in keeping the other words.

Both of the algorithms initially produce incomprehensible models containing many more features than a human can make sense of. However, our experiments show that heavy pruning can result in much more comprehensible models, with only moderate loss of accuracy. The results for Naïve Bayes are particularly striking, where a model containing only 10 words has only a moderate loss of accuracy, but a 500-fold increase in comprehensibility. Since comprehensibility is essential for this task, these results are encouraging.

PFOIL produces unstable models that can vary significantly with particular genes in the training data. The experiments have shown that our algorithm for improving the stability of PFOIL is effective at producing much more stable rule sets, though at a cost of reduced accuracy. This is a promising result, and we intend to use this technique with more sophisticated rule-building algorithms.

6. Related work

A great deal of research has been done in text mining, much of which involves biomedical tasks. Hearst's LINDI system [5] 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 our 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 [6] 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. Our tool is designed to allow the expression data itself to define its models. Masys et al. [8] 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 we use the text directly during the learning process and can explain single experiments.

7. Future directions and conclusions

We have presented an approach for aiding in the interpretation of microarray experiments that is based on machine learning and uses SwissProt text as its representation of the microarray's genes. We argue that there are at least three important properties for such a computational tool: it should produce models that are (1) accurate, (2) readable, and (3) stable to small changes in the microarray data. We empirically studied two widely successful algorithms—Naïve Bayes and PFoil—on an *E. coli* microarray experiment, evaluating these two approaches and some variants with respect to our desiderata. We have shown that both algorithms are able to find characterizations of the data with reasonable accuracy and that by adding pruning to the algorithms, comprehensibility can be achieved with only a minor reduction in accuracy. We also presented a modification to PFoil that has increased stability, but at a cost of decreased accuracy. We noted that the algorithms construct different kinds of characterizations. We intend to explore ways of combining the desirable properties of Naïve Bayes and PFoil, and to develop new algorithms for the task.

Another current limitation to this approach is that it relies only on text, though the sequence data are almost always also available. We plan to explore methods that make use of both the sequence data and the text annotating the genes. Another enhancement would be to increase the textual data we use. Abstracts from appropriate articles would be a logical next step. Finally, we plan to empirically evaluate our approach on additional microarray experiments.

Acknowledgements

This research was funded by grants NLM 1 R01 LM07050-01, NSF IRI-9502990, NIH 2 P30 CA14520-29, and NIH 5 T32 GM08349.

References

[1] A. Bairoch, R. Apweiler, The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000, Nucl. Acids Res. 28 (2000) 45–48.
[2] W. Brown, D. Lin, D. Cristianini, C. Sugnet, T. Furey, M. Ares, D. Haussler, Knowledge-based analysis of microarray gene expression data using support vector machines, Proc. Natl. Acad. Sci. 97 (1) (2000) 262–267.
[3] M. Craven, J. Shavlik, Learning symbolic rules using artificial neural networks, in: Proc. Intl. Conf. on Machine Learning, Morgan Kaufmann, 1993, pp. 73–80.
[4] S. Dudoit, Y. Yang, M. Callow, T. Speed, Statistical Methods for Identifying Differentially Expressed Genes in Replicated cDNA Microarray Experiments, UC-Berkeley Statistics Dept. TR #578, 2000.
[5] M. Hearst, Untangling text data mining, in: Proc. 37th Annual Meeting of Assoc. for Computational Linguistics, 1999.
[6] T.-K. Jenssen, A. Lægreid, J. Komorowski, E. Hovig, A literature network of human genes for high-throughput gene-expression analysis, Nat. Genet. 28 (2001) 21–28.
[7] C.D. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, The MIT Press, 1999.
[8] D. Masys et al., Use of keyword hierarchies to interpret gene expression patterns, Bioinformatics 17 (2001) 319–326.
[9] T. Mitchell, Machine Learning, McGraw-Hill, 1997.
[10] M. Porter, An algorithm for suffix stripping, Program 14 (1980) 130–137.
[11] J.R. Quinlan, Learning logical descriptions from relations, Mach. Learn. 5 (1990) 239–266.
[12] E. Xing, M. Jordan, R. Karp, Feature selection for high-dimensional genomic microarray data, in: Proc. Intl. Conf. on Machine Learning, Morgan Kaufmann, 2001, pp. 601–608.