Unsupervised Evidence Integration

Philip M. Long PLONG@CS.COLUMBIA.EDU
Vinay Varadan VARADAN@EE.COLUMBIA.EDU
Sarah Gilman SRG2104@COLUMBIA.EDU
Mark Treshock MAT140@COLUMBIA.EDU
Rocco A. Servedio ROCCO@CS.COLUMBIA.EDU
Columbia University

Abstract

Many biological propositions can be supported by a variety of different types of evidence. It is often useful to collect together large numbers of such propositions, together with the evidence supporting them, into databases to be used in other analyses. Methods that automatically make preliminary choices about which propositions to include can be helpful, if they are accurate enough. This can involve weighing evidence of varying strength.

We describe a method for learning a scoring function to weigh evidence of different types. The algorithm evaluates each source of evidence by the extent to which other sources tend to support it. The details are guided by a probabilistic formulation of the problem, building on previous theoretical work. We evaluate our method by applying it to predict protein-protein interactions in yeast, and using synthetic data.

1. Introduction

Motivation. An ongoing major international effort has targeted the problem of identifying which pairs of proteins interact [12, 8, 10, 3]. There are many fundamentally different indications of protein-protein interaction. These include indirect indications, such as a tendency to be synthesized under the same conditions, as well as various high-throughput experimental techniques that directly test whether pairs of proteins "like" to bind. Jansen, et al [5] showed that it is possible to learn an accurate predictor of protein-protein interactions in yeast using five attributes of a candidate protein pair, corresponding to a mix of direct and indirect evidence as described above.

Other bioinformatics problems take a similar form. There are many candidate propositions, and a number of fundamentally different sources of evidence that can be evaluated for each. One example is pairing genes in a newly sequenced genome with their counterpart human genes.

While databases have been automatically constructed using multiple sources of evidence (see [12]), it has not been obvious how best to automatically weigh the evidence of different types. A natural approach is to learn the relationship between numerical scores formalizing the strength of evidence arising from different sources, and a binary class designation indicating whether the statement is true or not. Jansen et al [5] learned to predict protein-protein interactions in yeast this way. However, it is often not possible to get sufficient training data consisting of examples of this relationship,[1] as is the case at present regarding protein-protein interactions in higher organisms [11].

Even when a few "gold standard" designations are available, they may be biased in unpredictable ways, and these biases could skew the results of algorithms that tried to use them in a supervised analysis. For example, a curated database may tend to be enriched for conclusions that can be drawn on the basis of the most easily applied and popular technology. If so, most supervised algorithms would overvalue the attribute corresponding to this "popular" evidence source.

Thus, we are faced with the problem of learning a rule for predicting class membership using training data without class membership information. This can

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005.

[1] For the case in which the class designation *can* be observed, sophisticated methods have also been developed (see [4, 2, 7]).

also be thought of as the problem faced by clustering algorithms; however, the applications driving this work present opportunities and challenges not found in many clustering problems.

The first opportunity arises from the fact that the value of each of the variables can be interpreted as the strength of evidence of a certain type supporting the proposition. Consequently, we can assume a priori that the larger the value of a variable, the more likely it is that the proposition holds. Furthermore, it is often the case that a number of unrelated sources of evidence are available. The approximate independence of the prediction errors resulting from such measures can be exploited by a learning algorithm. (This was exploited by the supervised algorithm for protein-protein interaction prediction in yeast in the Jansen, et al paper.) Finally, usually each of the sources of evidence is known to have some utility, so that deciding which to use and which to discard ("feature selection") is unnecessary.

The term "clustering" arises from the fact that in most clustering applications, members of each class tend to be grouped together away from the other classes. For some of the problems targeted in this work, however, the distributions of the values of the variables in the two classes overlap significantly. For example, as observed by Atschul [1], the distribution of BLAST scores of orthologous pairs of genes is approximately uniform from 0 up to a maximum, whereas the distribution of BLAST scores for other pairs of genes is approximately exponential.[2]

Because of these special characteristics, we prefer not to think of the problems studied here as clustering problems; if they are, they form an important type of clustering problem that merits special attention. In any case, we compare the method we propose with established approaches to clustering.

Our approach. In this paper, we describe a practical method of unsupervised evidence integration inspired by a theoretical analysis.

In the applications targeted in this work, scientists have sources of evidence that look for fundamentally different kinds of indications that a proposition might hold, and thus it is reasonable to expect that these evidence sources are approximately independent given the class designation. Conditional independence is used in our analysis in order to conclude that what association there is between pairs of variables is due to their common association with the class designation, and therefore strong associations are evidence that the variables are useful. If significant associations among variables are possible due to causes other than the common class designation, it would appear impossible to perform class prediction in an unsupervised manner, which is our goal. We note that (as will be seen in Section 2) our method includes measures that "shrink" estimates toward a default; this may be viewed in part as hedging against limited violations of independence. Still, the view that the main source of dependence among variables is their common association with the class designation is central to the design of our algorithm. While this assumption may not hold exactly true in real-world data sets, our experimental results reported in Section 3 show that our method does perform very well on natural biological data.

The details of our approach are based on a theoretical analysis that builds on earlier theoretical work by Mitina and Vereshchagin [9]. In their analysis, in addition to assuming that the values of $X_1, ..., X_n$ are conditionally independent given Y, they also assumed that

- the observed variables $X_1, ..., X_n$ are $\{0, 1\}$-valued,
- each variable is equal to the hidden class designation Y more often than not,
- for each i, $\Pr(X_i \neq Y | Y = y)$ does not depend on y.

This last property was identified by Mitina and Vereshchagin as crucial for their analysis. If independent training examples are generated from a source with these properties, they gave a polynomial-time algorithm that learns a rule for predicting the value of Y that approaches the accuracy of the optimal such rule.

Some aspects of our application are not captured in Mitina and Vereshchagin's setting. For example, some of the variables in our applications are continuous-valued. Even if we were to treat each X_i as a binary-valued variable by thresholding, it is difficult to do so in such a way that whether this binary-valued variable equals Y is (approximately) independent of the value of Y. For these and other reasons, we needed to build substantially on Mitina and Vereshchagin's work to address our applications.

Our algorithm, like Mitina and Vereshchagin's, takes advantage of the fact that a variable is valuable for prediction to the extent that it tends to agree with

[2]Extreme value distributions [6] provide a more detailed model.

other variables. We call our new method *peer ranking*.[3]

Evaluation. We compare peer ranking with three possible alternatives. The first is an algorithm that (i) finds the line with the property that projecting the examples onto that line maximizes the variance, and (ii) then sorts examples according to their positions on that line. The second is an algorithm (described in Section 3) based on k-means clustering. The third uses EM, a standard incremental optimization technique, to search for the maximum likelihood probability model from a set of "naive bayes" models like those implicitly used by our method. The applications targeted in this work give rise to inputs with tens of millions of examples, and possibly more. Thus, direct application of kernel-based or spectral clustering methods appears infeasible.

Although the peer ranking method is intended to be applied in an unsupervised setting, to evaluate how well it works on natural data, it is necessary to apply it in a situation where a gold standard *is* available. For this, we use five sources of evidence of protein-protein interaction from Jansen, et al's paper [5]. We also evaluated the peer algorithm on synthetic randomly generated data. On both kinds of data, peer ranking also performs substantially better than the other methods.

2. The peer ranking algorithm

In our problem, an algorithm is given a collection of examples, each of which is a tuple $(X_1, ..., X_n)$, and outputs a scoring function g that is used to rank the examples. The goal is for examples for which a hidden variable Y takes the value 1 to be high on the list.

As mentioned in the introduction, the peer ranking approach is based on a view that the values $X_1, ..., X_n$ of the variables, together with the hidden class designation Y, are generated collectively by a joint probability distribution. The peer algorithm behaves as if $X_1, .., X_n$ complement one another, in the sense that they are conditionally independent given Y. It also behaves as if $\Pr(Y = 1|X_i > x)$ is nondecreasing in x for all i.

2.1. Outline

The outline of the peer ranking algorithm is as follows:

1. It estimates conditional probabilities relevant to the predictive ability of binary-valued discretizations of $X_1, ..., X_n$ in a number of rounds: in round number ℓ, it
   a. chooses a value β_ℓ and thresholds $b_{\ell,1}, ..., b_{\ell,n}$ so that for each i, $\Pr(X_i \geq b_{\ell,i}) \sim \beta_\ell$ (how β_ℓ is chosen, and how exactly $b_{\ell,1}, ..., b_{\ell,n}$ are chosen, are described in Section 2.3),
   b. for each i, estimates $\Pr(X_i \geq b_{\ell,i}|Y = 1)$ and $\Pr(X_i \geq b_{\ell,i}|Y = 0)$ (this is the key step in the algorithm – how it does this is explained in Section 2.2).
2. It constructs a scoring function g using the estimates as follows: given an example $(X_1, ..., X_n)$ to be scored, it
   a. for each i, finds the largest member b of $B_i = \{b_{1,i}, b_{2,i}, ...\}$ for which $X_i \geq b$ (call it $\mathrm{floor}_{B_i}(X_i)$), and
   b. sets g to be an estimate of
   $$\Pr(Y = 1|X_1 \geq \mathrm{floor}_{B_1}(X_1) \wedge ... \wedge X_n \geq \mathrm{floor}_{B_n}(X_n))$$
   obtained from the previously constructed conditional probability estimates (how this is done is described in Sections 2.7 and 2.8).

[3] Thanks to Vinsensius Vega for suggesting this term.

2.2. The basic estimates

In this subsection we concentrate on a particular iteration of step 1, i.e. a particular value of ℓ. Since ℓ is fixed, we drop it from all notation.

Define $U_1, ..., U_n$ by letting U_i be 1 if $X_i \geq b_i$, and 0 otherwise. Each value U_i can be viewed as a (very rough) prediction of Y, since by assumption the examples with large values of X_i are those examples that the ith source of evidence views as most likely to have $Y = 1$. The value of b_i is chosen so that $\Pr(U_i = 1) = \Pr(X_i \geq b_i)$ is approximately β (how this is done is described in Section 2.3), but if, for example, the distribution of X_i has an accumulation point (that is, many examples have the same value for X_i), then this probability may not actually be very close to β. Note that if we have a lot of training data, the fraction of examples in our training set for which $U_i = 1$ provides an accurate estimate of the true value of $\Pr(U_i = 1)$.

We emphasize the difference between X_i and U_i: X_i is a (possibly continuous) value that is interpreted as the strength of evidence of the ith type, and U_i is a binary-valued prediction obtained by examining whether X_i is large enough.

Suppose for a moment that we know $\Pr(Y = 1)$ (how to estimate it will be treated in Section 2.4). Since $\Pr(U_i = 1|Y = 0)$ equals

$$\frac{\Pr(U_i = 1) - \Pr(Y = 1)\Pr(U_i = 1|Y = 1)}{1 - \Pr(Y = 1)}, \qquad (1)$$

if we have an estimate of $\Pr(U_i = 1|Y = 1)$, we can use it to obtain an estimate of $\Pr(U_i = 1|Y = 0)$.

If, in addition, the number n of variables is at least 3, then in fact we can estimate $\Pr(U_i = 1|Y = 1)$ by looking at the extent to which pairs of variables agree that $Y = 1$. This is because for any three distinct indices i, j, k we have

$$\begin{aligned}&\Pr(U_i = 1|Y = 1)\\ &= a_i + \sqrt{\left(\frac{1-\Pr(Y = 1)}{\Pr(Y = 1)}\right)\frac{(a_{i,j}-a_i a_j)(a_{i,k}-a_i a_k)}{a_{j,k}-a_j a_k}}\end{aligned} \tag{2}$$

where a_i denotes $\Pr(U_i = 1)$ and $a_{i,j}$ denotes $\Pr(U_i = 1 \ \& \ U_j = 1)$. The straightforward but tedious derivation of (2) is in Appendix A.

The most natural estimate of $a_{i,j} = \Pr(U_i = 1 \ \& \ U_j = 1)$ is the fraction of training examples for which both $U_i = 1$ and $U_j = 1$. Indeed this, and the corresponding estimate of $a_{i,k}$, are plugged into the numerator in (2) to estimate $\Pr(U_i = 1|Y = 1)$ in our algorithm. However, estimation of the probability $a_{j,k} = \Pr(U_j = 1 \ \& \ U_k = 1)$ in the denominator requires special treatment. Note that underestimates of this quantity can have a radical effect on the resulting estimate of $\Pr(U_i = 1|Y = 1)$, as they can bring the denominator close to 0. Thus our algorithm adjusts the estimate of $a_{j,k}$ upwards by the width of a 95% confidence interval around the empirical estimate, to make it unlikely that $a_{j,k}$ is underestimated, and to reduce the extent of such underestimation when it occurs. Specifically, if $q_{j,k}$ is the fraction of training examples for which $U_j = 1$ and $X_k \geq b_k$, then our estimate $\hat{a}_{j,k}$ is given by

$$\hat{a}_{j,k} = q_{j,k} + 1.96\sqrt{\frac{q_{j,k}(1 - q_{j,k})}{m}}, \tag{3}$$

where m is the number of training examples. This adjustment also has the effect of bringing $\Pr(U_i = 1|Y = 1)$ closer to $\Pr(U_i = 1)$, that is, moderating any optimism about utility of the ith variable.

Estimates of $\Pr(U_i = 1|Y = 1)$ can be constructed in this way using any other pair $\{j, k\}$ of variables. The role that a particular pair plays is determined in part by the fractions q_j and q_k of training examples for which $U_j = 1$ and $U_k = 1$ respectively. The estimates obtained from other pairs are combined as follows:

- for all pairs $\{j, k\}$ of other variables for which

$$q_{j,k} > q_j q_k, q_{i,j} > q_i q_j \text{ and } q_{i,k} > q_i q_k, \tag{4}$$

  and estimate of $\Pr(U_i = 1|Y = 1)$ is constructed using using variables j and k as outlined above, and
- the median of the resulting estimates is used as the final estimate.

This further enhances the robustness of the method against departures from the conditional independence assumption. In the unlikely but conceivable event that no pair $\{j, k\}$ satisfies (4), then $\Pr(U_i = 1|Y = 1)$ is estimated to be $\Pr(U_i = 1)$ (that is, U_i is estimated to be useless for prediction).

2.3. Choosing values for β and b

In round ℓ, the peer algorithm chooses thresholds $b_{\ell,1}, ..., b_{\ell,n}$ so that, for each i, the fraction of examples for which $X_i \geq b_{\ell,i}$ is as close as possible to β_ℓ subject to the constraint that the fraction is at least β_ℓ. (Note that many examples might share the same value of X_i.)

As described above, the event that $X_i \geq b_{\ell,i}$ can be viewed as a rough prediction of the event that $Y = 1$; in order for this prediction to be as accurate as possible we would like to have $\Pr[X_i \geq b_{\ell,i}] \approx Pr[Y = 1]$. Since, in many applications, $\Pr(Y = 1)$ is small, the values of β_ℓ are chosen according to a geometric progression, so that the region near 0 is populated more densely. Specifically, the value of β is doubled each time, so that $\beta_\ell = \beta_1 2^{\ell-1}$.

We set $\beta_1 = 1/m$ because we cannot measure probabilities at a finer granularity from m examples.

2.4. Estimating $\Pr(Y = 1)$

The ordering output by the algorithm should be expected to be insensitive to its estimated value of $\Pr(Y = 1)$: the main effect of changing its value will be to shift all estimates of conditional probabilities up or down, leaving the order largely unchanged. The value of $\Pr(Y = 1)$ is therefore estimated in a rough way, by examining the consequences of plugging different estimates into (1) and (2).

If an estimate is too small, then plugging into (2) will result in estimates of $\Pr(X_i \geq b_i|Y = 1)$ that are too large. In extreme cases, they can be greater than 1. On the other hand, using estimates of $\Pr(Y = 1)$ that are too large with (1) can result in estimates of $\Pr(X_i \geq b_i|Y = 0)$ that are too small, and, in extreme cases, less than 0. The value of $\Pr(Y = 1)$ is chosen to maximize the margin by which both of these illogical outcomes is avoided. This maximum margin might be viewed as a crude, but computationally feasible, approximation to maximum likelihood.

2.5. Clipping

Intuitively, the training data can only provide information about probabilities at a granularity inversely proportional to the number of training examples. Conditional probability can be informed only at a coarser granularity. Thus, we clip all probability estimates to fall between $1/m$ and $1-1/m$ (and any other estimates are replaced with the closest value in that range).

2.6. Enforcing monotonicity

Recall that, in the intended applications, we expect that $\Pr(Y = 1|X_i \geq b)$ is nondecreasing in b. On the other hand, our adjustment (3) results in a bias toward conservatism. While, overall, this helps, one unfortunate side effect is that our estimates of $\Pr(Y = 1|X_i \geq b)$ might not be monotone in b. An estimate of $\Pr(Y = 1|X_i \geq b)$ for large b might be based on only a few cases, and so the algorithm would conservatively adjust this estimate downward. There might be $b' < b$ for which, due to a larger number of cases for which $X_i \geq b'$, the algorithm might be able to confidently assert a large value of $\Pr(Y = 1|X_i \geq b')$. But in such cases, the monotonicity implies that the algorithm can conclude that $\Pr(Y = 1|X_i \geq b)$ is at least as big.

This reasoning is incorporated into the peer ranking algorithm in a postprocessing step, in which pairs of estimates that conflict with the monotonicity assumption are resolved in favor of the more confident estimate.

2.7. Preliminary scoring function

Recall from Section 2.1 that the scoring function is obtained by evaluating an estimate of

$$\Pr(Y = 1|X_1 \geq \mathrm{floor}_{B_1}(X_1) \wedge ... \wedge X_n \geq \mathrm{floor}_{B_n}(X_n))$$

where each B_i is made up of the thresholds $b_{\ell,i}$ for the useful rounds ℓ. Exploiting the conditional independence assumption in a standard way, we can see that the above conditional probability is monotone in

$$\sum_i \log \frac{\Pr(X_i \geq b_i|Y = 1)}{\Pr(X_i \geq b_i|Y = 0)}. \tag{5}$$

Since our ultimate goal is to order the examples based on their values of $X_1, ..., X_n$, any two scoring functions are equivalent if they order the examples in the same way, and thus we can perform any monotone transformation on our scoring function. The derivation of (5) is standard (omitted due to space constraints).

Plugging our estimates of $\Pr(X_i \geq b_i|Y = 1)$ and $\Pr(X_i \geq b_i|Y = 0)$ for various i into (5), we obtain our preliminary scoring function.

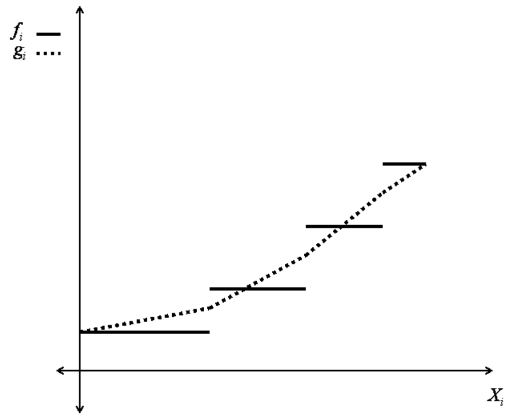

Figure 1. An example of the definition of the function g_i mapping the value of a variable X_i to its contribution to the overall score: g_i is a continous, piecewise linear approximation to the function f_i.

2.8. Final, smoothed, scoring function

A drawback of the scoring function f described in Section 2.7 is that, when used for sorting, it can result in many ties (cases where two propositions are assigned the same score). Note that (5) expresses f as a sum over functions that depend on individual variables; let us denote by f_i the function that depends on variable i. Each of $f_1, \ldots, f_n$ is monotone and piecewise constant. The final scoring function g used by the peer ranking algorithm is obtained by replacing each f_i with a corresponding piecewise linear function g_i and letting

$$g(X_1, ..., X_n) = \sum_i g_i(X_i).$$

A representative example of how g_i is generated from f_i is shown in Figure 1: The value of g_i at a threshold is the average of the values of f_i on either side, and g_i interpolates linearly between.

2.9. Discussion – why not use bins?

An arguably more principled approach would be to use the members of each B_i to divide the real line into bins, and then design the scoring function by evaluating the conditional probability that Y is 1 given the bin memberships of the values of the variables. However, the individual conditional probabilities that values fall in bins given that Y is 1 and 0 respectively are estimated with too high a variance for this to work. The probability that an example falls in a bin is the difference between the probability that it falls to the right of the right endpoint, and the probability that if falls to the right of the left endpoint. The estimate of the bin probability is thus sensitive to fluctuations in *both* these probability estimates. We have found that the approach we suggest works better, we suspect in large part because only *one* of these probabilities is estimated.

We note since the peer algorithm chooses thresholds geometrically, many of the examples for which $X_i \geq b_{\ell,i}$ also satisfy $X_i \in [b_{\ell,i}, b_{\ell-1,i}]$. Thus, it is reasonable to view $\frac{\Pr(X_i \geq b_i | Y=1)}{\Pr(X_i \geq b_i | Y=0)}$ as an approximation to the corresponding ratio for the bin with left endpoint b_i.

Alternatively, note that, if a scientist is able to identify a null distribution that matches the conditional distribution given that $Y = 0$, then $\Pr(X_i \geq b | Y = 0)$ is the p-value associated with the case where $X_i = b$. So the peer method may be viewed as going beyond combining p-values to also estimate the corresponding conditional distributions given that $Y = 1$.

3. Experiments

We evaluated the peer ranking algorithm by applying it to predict protein-protein interactions in yeast, and on artificial data. The results are summarized in Table 1.

3.1. Other algorithms

We compare the peer algorithm with three standard methods that are feasible with large datasets like those that arise in the targeted applications.

The first algorithm ("k-means algorithm")

1. performs k-means clustering with $k = 2$,
2. defines the $Y = 1$ cluster to be the one with the "larger" center (the center with the larger value on the majority of the variables),
3. ranks propositions by how much closer they are to the center of the $Y = 1$ cluster than the other cluster center.

The second algorithm ("EM algorithm") proceeds as follows:

1. It discretizes the data by dividing the values of the variables into bins roughly as in the peer algorithm: for each variable i, if the examples are sorted in order of the values of variable j, then the jth bin boundary is placed after example number $100 \cdot 2^j$. If the next example takes the same value, the boundary is placed after the last example with that value. (As in the peer algorithm, more boundaries are placed near the top because the minority class is small, and large values of the variables are associated with membership in the minority class.)
2. It learns a Naive Bayes probability model for the discretized data: i.e., the model includes a single binary-valued hidden variable, and regards the discretized observed variables as conditionally independent given the hidden variable. It uses EM to attempt to find the maximum likelihood model in this class.
3. It defines the $Y = 1$ cluster to be the one for which most variables are more likely to take their largest value.
4. It ranks propositions by evaluating the conditional probability that $Y = 1$ given the discretized values of their variables.

The third algorithm ("eigen algorithm")

1. estimates the covariance matrix C of $X_1, ..., X_n$,
2. finds the eigenvector $(w_1, ..., w_n)$ of C with the largest eigenvalue, and
3. ranks propositions by $w_1 X_1 + ... + w_n X_n$.

Because the datasets targeted in this work are too large to fit in memory, the run-times of the algorithms are dominated by the number of passes required over the data. The eigen algorithm makes two passes over the data, one to estimate the covariance matrix, and one to compute the scores. The peer algorithm needs to sort the values of each attribute, then make another pass to collect statistics, followed by a pass to compute scores. The k-means and EM algorithms need to make a pass over the data for each iteration, making them much slower than the eigen and peer algorithms. It was infeasible for us to perform our experiments while allowing these algorithms to run all the way to convergence: we limited the number of iterations of each algorithm to 20, still allowing the iterative algorithms much more time than the peer and eigen algorithms.

3.2. Protein-protein data

To evaluate algorithms, we need data with gold-standard indications of the truth or falsehood of the propositions they are predicting. It is worth emphasizing that the purpose of this analysis is to evaluate the algorithms, and not to discover new biology.

We used data provided with a paper by Jansen, et al [5] on predicting protein-protein interactions in yeast.

Their analysis used the following sources of evidence of interaction: (a) the outcome of yeast two-hybrid experiments, (b) the outcome of in-vivo pulldown experiments, (c) the tendency to be co-expressed across a variety of conditions, (d) similar functional annotations, and (e) the tendency to be essential in the same circumstances.

We used the raw variables from the Jansen, et al study exactly as we found them, with the following exceptions: (a) We summed the indicator variables corresponding to the results of two yeast two-hybrid screens, obtaining a single $\{0, 1, 2\}$-valued yeast two-hybrid variable, (b) we analogously summed the two in-vivo pulldown variables, and (c) we took the reciprocals of the two functional annotation raw variables (to obtain values that increase with the likelihood of interaction, and are on a similar scale to the other variables) and added them to obtain a single functional annotation variable. Missing values were replaced with 0.

We downloaded the gold-standard indications of interaction from the web site of the Jansen, et al paper: these were obtained from the curated MIPS data. Jansen, et al, also determined a list of proteins regarded as not interacting by finding pairs that are annotated as localizing to different parts of the cell in the GO database. Of the 24229603 protein pairs in the dataset, 2713970 are given gold standard designations this way, 8250 annotated as interacting and the rest not. (In the raw data from the Jansen, et al web site, 124 protein pairs appeared both in the list of interacting proteins, and in the list of negative examples arising from the localization evidence. These protein pairs were regarded as interacting in our evaluation. This appears the most logical choice, but due to their small number, this choice must have an insignificant effect on our evaluation anyway.)

We applied the peer algorithm, and the four algorithms we compare with, to the set of 24229603 protein pairs with data from that paper, and evaluated them by computing ROC scores (also known as the AUC, for Area Under the Curve) on the 2713970 gold-standard examples. The results are tabulated in Table 1. Because the iterative algorithms use random starting configurations, their results are averaged over 5 runs.

For a given value a of the AUC, $1 - a$ can be viewed as an error probability: it is the probability that a randomly chosen true proposition is ranked below a randomly chosen false proposition. On the protein-protein data, the peer algorithm reduces this error probability by more than a factor of two compared with the other algorithms. With 8250 positive examples, this difference is highly statistically significant.

Algorithm	Protein-Protein	Synthetic
Peer	0.947	0.977
Eigen	0.862	0.899
k-means	0.862	0.724
EM	0.848	0.911

Table 1. AUC scores on protein-protein and synthetic data.

3.3. Synthetic data

We also tried out the algorithms on artificially generated data. An obvious advantage of such data is that the correct class of an item can be determined without question. In our data, $X_1, ..., X_n$ are conditionally independent given Y. The probability distributions governing the individual variables were inspired by the high-level description of BLAST scores described by Altschul [1]: when $Y = 1$, X_i is distributed uniformly on $[0, 1]$, and when $Y = 0$, X_i is distributed according to an exponential distribution that has been truncated at 1 and normalized (so that $\Pr(Y = 1|X \geq b)$ is monotone in b). To generate variables with varying strengths, the means of the exponential variables were chosen uniformly at random from [0,1/2]. The value 1/2 was chosen to generate variables with strong but not overwhelming association with Y.

We generated 100 random sources with five variables and $\Pr(Y = 1) = 0.01$ this way. For each source, we generated 100000 examples, tried all the algorithms, and calculated the AUC for each algorithm. For each algorithm, we then averaged the AUC scores.

3.4. Code and data

The following is available by request: source code for all the algorithms, the protein-protein data used in our analysis, scripts to generate the synthetic data, and a script to reproduce the results of this section.

Acknowledgments

We warmly thank Edison Liu, Yoav Freund, Vinsensius Vega, K. R. Krishna Murthy, Anshul Kundaje, John Rice, Prasanna Kolatkar, Peter Bickel, Drew Bagnell, Naoki Abe, David Haussler and Bin Yu for valuable conversations.

References

[1] S. Altschul. Assessing the accuracy of database search methods,and improving the performance

of psi-blast, 2002. Invited talk at ISMB.

[2] M. Deng, T. Chen, and F. Sun. An integrative analysis of protein function prediction. *RECOMB*, pages 95–103, 2003.

[3] C. Alfarano et al. The biomolecular interaction network database and related tools. *Nucleic Acids Res.*, 33:D418–D424, 2005.

[4] Y. Freund, R. Iyer, R.E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In *Proc. 15th International Conference on Machine Learning*, 1998.

[5] R. Jansen, H. Yu, D. Greenbaum, Y. Kluger, N. J. Krogan, S. Chung, A. Emili, M. Snyder, J. F. Greenblatt, and M. Gerstein. A bayesian networks approach for predicting protein-protein interactions from genomic data. *Science*, 302:449–453, 2003.

[6] S. Karlin and S. F. Altschul. Method for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87:2264–2268, 1990.

[7] G. R. G. Lanckriet, M. Deng, N. Cristianini, M. I. Jordan, , and W. S. Noble. Kernel-based data fusion and its application to protein function prediction in yeast. *PSB*, 2003.

[8] E. M. Marcotte, M. Pellegrini, H. L. Ng, D. W. Rice, T. O. Yeates, and D. Eisenberg. Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285(5428):751–3, 1999.

[9] O. Mitina and N. Vereshchagin. How to use several noisy channels with unknown error probabilities. *Information and Computation*, 184(2):229–241, 2003.

[10] I. M. Nooren and J. M. Thornton. *EMBO J.*, 22(14):3486–92, 2003.

[11] C. Von Mering, R. Krause, B. Snel, M. Cornell, S. G. Oliver, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417:399–403, 2002.

[12] I. Xenarios, D. W. Rice, L. Salwinski, M. K. Baron, E. M. Marcotte, and D. Eisenberg. DIP: The database of interacting proteins. *Nucleic Acids Research*, 28:289–91, 2000.

A. Derivation of equation (2)

Define

- $\theta_i = \Pr(U_i = 1 | Y = 1)$, and
- $p = \Pr(Y = 1)$.

By breaking down into the cases in which $Y = 1$ and $Y = 0$, we get

$$\begin{aligned} a_{i,j} &= \Pr(U_i = U_j = 1 | Y = 0) \Pr(Y = 0) \\ &\quad + \Pr(U_i = U_j = 1 | Y = 1) \Pr(Y = 1). \end{aligned} \tag{6}$$

Exploiting the conditional independence, we get

$$\begin{aligned} a_{i,j} &= \Pr(U_i = 1 | Y = 0) \Pr(U_j = 1 | Y = 0) \Pr(Y = 0) \\ &\quad + \Pr(U_i = 1 | Y = 1) \Pr(U_j = 1 | Y = 1) \Pr(Y = 1). \end{aligned} \tag{7}$$

Equation (1) can be rewritten using this section's notation as follows:

$$\Pr(U_i = 1 \mid Y = 0) = \frac{a_i - p\theta_i}{1 - p}. \tag{8}$$

Combining (7) and (8) with the fact that $\Pr(Y = 0) = 1 - \Pr(Y = 1)$, and simplifying, we get

$$a_{i,j} = \frac{(a_i - p\theta_i)(a_j - p\theta_j)}{1 - p} + \theta_i \theta_j p. \tag{9}$$

Similarly,

$$a_{i,k} = \frac{(a_i - p\theta_i)(a_k - p\theta_k)}{1 - p} + \theta_i \theta_k p \tag{10}$$

$$a_{j,k} = \frac{(a_j - p\theta_j)(a_k - p\theta_k)}{1 - p} + \theta_j \theta_k p. \tag{11}$$

Solving (9) for θ_j, we get

$$\theta_j = \frac{a_j(\theta_i p - a_i) + a_{i,j}(1 - p)}{p(\theta_i - a_i)}. \tag{12}$$

Replacing the occurrence of θ_j in (11) with the right-hand-side of (12) and simplifying yields

$$a_{j,k} = \frac{a_{i,j} a_k - a_j a_k \theta_i - a_{i,j} \theta_k + a_i a_j \theta_k}{a_i - \theta_i}. \tag{13}$$

Solving (13) for θ_k and simplifying, we get

$$\theta_k = \frac{-a_i a_{j,k} + a_{i,j} a_k + a_{j,k} \theta_i - a_j a_k \theta_i}{a_{i,j} - a_i a_j} \tag{14}$$

Replacing the θ_k in (10) with the right-hand-side of (14), and simplifying, yields

$$\begin{aligned} a_{i,k} = \big(& a_i^2 (a_{j,k} p - a_j a_k) + (a_{j,k} - a_j a_k) p \theta_i^2 \\ &+ a_i (a_{i,j} a_k (1 - p) - 2(a_{j,k} - a_j a_k) p \theta_i)\big) / \\ &\big((a_{i,j} - a_i a_j)(1 - p)\big). \end{aligned} \tag{15}$$

Solving for θ_i and simplifying completes the proof.