Predicting Protein Interactions with Motifs

Jessica Long
Chetan Sharma
Lekan Wang

December 12, 2008

1 Background

Proteins are essential to almost all living organisms. They are comprised of a long, tangled chain of amino acids, of which there are twenty of the organic molecules. Proteins also tend to "fold" into distinctive shapes with certain idiosyncratic structures, with names such as "β-sheets," "α-helix," and "β-hairpin," which we typically call secondary structures. Combinations of these structures give rise to "motifs" in proteins, and some of these motifs have functional properties, such as mineral binding, protein-protein interactions, and enzymatic effects. The property we will consider is protein-protein interactions. Understanding which proteins interact is an important part of understanding biological processes, as many bodily functions are triggered by protein signaling pathways–series of proteins that interact, each causing the subsequent protein to re-fold in a slightly different way that allows it to interact with the next protein, and that protein will re-fold and interact with the next, and so on until the final protein is activated to do something biologically useful.

2 Data

In our problem, we deal with two matrices:

The matrix indicating the presence of each motif in each protein which is a subset of $R^{m \times n}$, where
n = number of proteins
m = number of motifs
This matrix has values A_{ij}={1 if protein i interacts with protein j, 0 otherwise}

We used a dataset of 4481 proteins and 8089 motifs. Both the associations matrix, A, and the interaction matrix, B, were extremely sparse matrices. Most proteins interact meaningfully with few other proteins, and most proteins have few of the motifs. The proteins, on average, interact with 35.9 other proteins, and contain 2.626 motifs. This made it difficult to find a relevant subset of the data to reduce the features for classification or to take repeated samples for bagging. A relevant set of proteins and motifs means that each protein shares a motif with one othe protein, and also that the set of proteins have multiple interactions.

This difficulty with finding a smaller relevant bootstrap sample, as well as the sheer size of the matrices made our bagging algorithm require prohibitively large amounts of computation time.

3 Boosted Decision Trees

Finding a complex rule in classification is often extremely computationally difficult. However, it is often easier to find many simple rules of thumb and let them work in tandem rather than finding a single complex rule. This is the inspiration behind boosting, in which many simple rules are found in multiple rounds, and in which the samples that are classified incorrectly in each round are then given more weight in the next round.

3.1 Basic Boosting Algorith

The basic boosting algorithm is as follows (Schapire, 2003):

Initialize $D_1(i) = 1/m$
for $t = 1, \ldots, T$:

- Train base learner using distribution D_t
- Get base classifier $h_t : X \to \Re$
- Choose an $\alpha_t \in \Re$
- Update $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Q_t}$ where Q_t normalizes $D_{t+1}$to a distribution.

Once trained, the classifier can output prediction with $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

With the AdaBoost algorith, we choose $\alpha_t = \frac{1}{2}\ln\left(\frac{w^+}{w_t^-}\right)$, where $w^+$is the number of positive samples, and $w^-$is the number of negative samples.

Since we are using boosted decision trees, we need to determine the relative position of the new node. We will do this by finding $Z_i = w_0 + 2\sqrt{w^+ \cdot w^-}$.

3.2 Upper Bound using Z_{pure}

One of the innovative parts of our procedure is the idea of Z_{pure} (Kunjade, Anshul). As captured by our algorithm, calculating Z_i runs in $O(n^2)$ time, where n is the number of samples. To make the algorithm more efficient, we calculate a best possible Z at each leaf, called Z_{pure}. $Z_{pure}$is the value that Z_i takes on when a particular rule classifies all negative examples, or all positive examples correctly.

We know that:

(1) in the case where the rule classifies all negative samples correctly:
$w_0' = w_0 + w^{+\prime} = 0;\ w^{-\prime} = w^-$

(2) in the case where the rule classifies all positive samples correctly:

$w_0' = w_0 + w^-;\ w^{+\prime} = w^{+\prime} w^{-\prime} = 0$

So calculating Z_{pure} is computationally quite simple as compare to calculating all other values of Z_i.

Ordinarily, it is necessary to calculate Z_{min} at every leaf to find the absolute minimum Z_{min}. However, if the current Z_{min} is lower than Z_{pure}, then we know that the lowest possible value of Z at that particular leaf cannot be less than our current Z_{min}. This means that we do not need to calculate Z_{min} at the Z_{pure} leaf, thus saving computational time.

3.3 AdaBoost Results

Using our adaboost algorithm with 100 iterations, we found the training loss to be 0.791067%, and the test loss to be 0.877136%, which means all but a few of the positive protein interactions were misclassified. In each iteration, the optimal motif pair is chosen and its weighted classifications added to the tree. In some iterations, even the optimal motif pair results in a loss in training error because of the weightings of the respective positive and negative labels. In this case, the weight of the positive interactions for the motif pair overweigh the negative interactions, and make the training loss increase. However, the error generally decreases with iterations of the algorithm. However, due to the sparsity of the matrices, the model takes a long time to converge, as each motif pair covers few protein combinations.

We ran the algorithm again with a reduced set of features. Instead of using all 8089 motifs, we used the 569 motifs that occurred in two or more different proteins, and re-classified with those. This required much less computational power, and didn't significantly increase the error. This is expected, as the boosted decision tree would have only used the more commonly associated motif pairs to predict, resulting in the same motifs being chosen. If we had the computational power to run many iterations, then the error would be different as the algorithm reaches motif pairs that occur in fewer and fewer proteins.

The AdaBoost algorithm will likely work better if other features were used in addition to motifs (secondary structures, surface charge, etc), or if we could find a data set with more motif commonalities. More training data, in the form of more proteins or more motifs, would also help the boosted algorithm find more commonalities between the proteins on which to predict.

See figures at the end of the document for further information.

4 Naive Bayes

4.1 The algorithm

Because boosting with so many rounds took a significant amount of time, we also implemented Naive Bayes as a way to get some quick results. Given a distribution of whether or not two proteins interact, conditioned on the the joint distribution of the two vectors of motifs associated with the two proteins, we have:

$$
\begin{aligned}
p(I_{\vec{m}\vec{n}}|\vec{m},\vec{n}) &= \frac{p(\vec{m},\vec{n}|I_{\vec{m},\vec{n}})p(I_{\vec{m},\vec{n}})}{p(\vec{m},\vec{n}|I_{\vec{m},\vec{n}}=0)+p(\vec{m},\vec{n}|I_{\vec{m},\vec{n}}=1)} \\
&= \frac{p(m_1,n_1|I_{\vec{m},\vec{n}})p(m_1,n_2|I_{\vec{m},\vec{n}})\cdots p(m_j,n_{k-1}|I_{\vec{m},\vec{n}})p(m_j,n_k|I_{\vec{m},\vec{n}})\cdots p(I_{\vec{m},\vec{n}})}{(p(m_1,n_1|I_{\vec{m},\vec{n}}=0)+p(m_1,n_1|I_{\vec{m},\vec{n}}=1))\,(p(m_j,n_k|I_{\vec{m},\vec{n}}=0)+p(m_j,n_k|I_{\vec{m},\vec{n}}=1))\cdots.}
\end{aligned}
$$

where $|m| = j$ and $|n| = k$. The second line is from the Naive Bayes assumption that individual features (the motifs) are independent.

We built up a matrix of counts, counting the presence of each motif in each pair of motifs present at each interaction or non-interaction. Then, we take the margins of every row and column of counts, and, after applying LaPlace smoothing, find the joint distributions of all $\vec{m}, \vec{n}.$, and each $I_{\vec{m},\vec{n}}$. Then, to predict, we simply calculate both $p(I_{\vec{m}\vec{n}} = 0|\vec{m},\vec{n})$ and $p(I_{\vec{m}\vec{n}} = 1|\vec{m},\vec{n})$, and compare which one is larger. Because the products in both the numerator and denominator risk underflow, we use the log-probability, $\log p(I_{\vec{m}\vec{n},}|\vec{m},\vec{n})$, in the prediction phase.

4.2 Results

Because Naive Bayes required calculating over each motif pairing of each protein pair, the problem was $O(m^2n^2)$, using the definitions of m and n given in section 2. With nearly 5000 proteins and 10000 motifs, we couldn't efficiently process the entire data set, so as in boosting, we selected some of the proteins and motifs that had the least sparse data, and ran Naive Bayes over those data. The result was similar to AdaBoost, with a 0.781%train error and a 0.895%test error, so very few of the positive protein interactions were classified correctly. Again, we attribute this to the overwhelming majority of non-interactions over interactions.

5 Conclusions

In the end, we tried several different algorithms, but had limited success due to the sparseness of our data. Ideally, we would have liked to make the data more concentrated. We tried to limit our data set, but it was difficult to find sparse subsets within the data and more difficult to predict whether these would be representative subsets of proteins and motifs.

Our primary algorithm was moderately successful, yielding a test loss that was almost half as much as random chance. It's possible that running more iterations of the algorithm would have given us better results, but throughout the first fifty iterations, our test loss and train loss remained completely stable. Concerned by the lack of updating on successive iterations, we turned to Naïve Bayes and Adaboost to give us a better sense of how traditional machine learning algorithms handled our particular data set. Unsurprisingly, the results from both algorithms were fairly poor, given the small number of interactions on which to condition future predictions.

We believe that better results could be attained in the future by using data with more of the protein interactions already filled in.

6 References

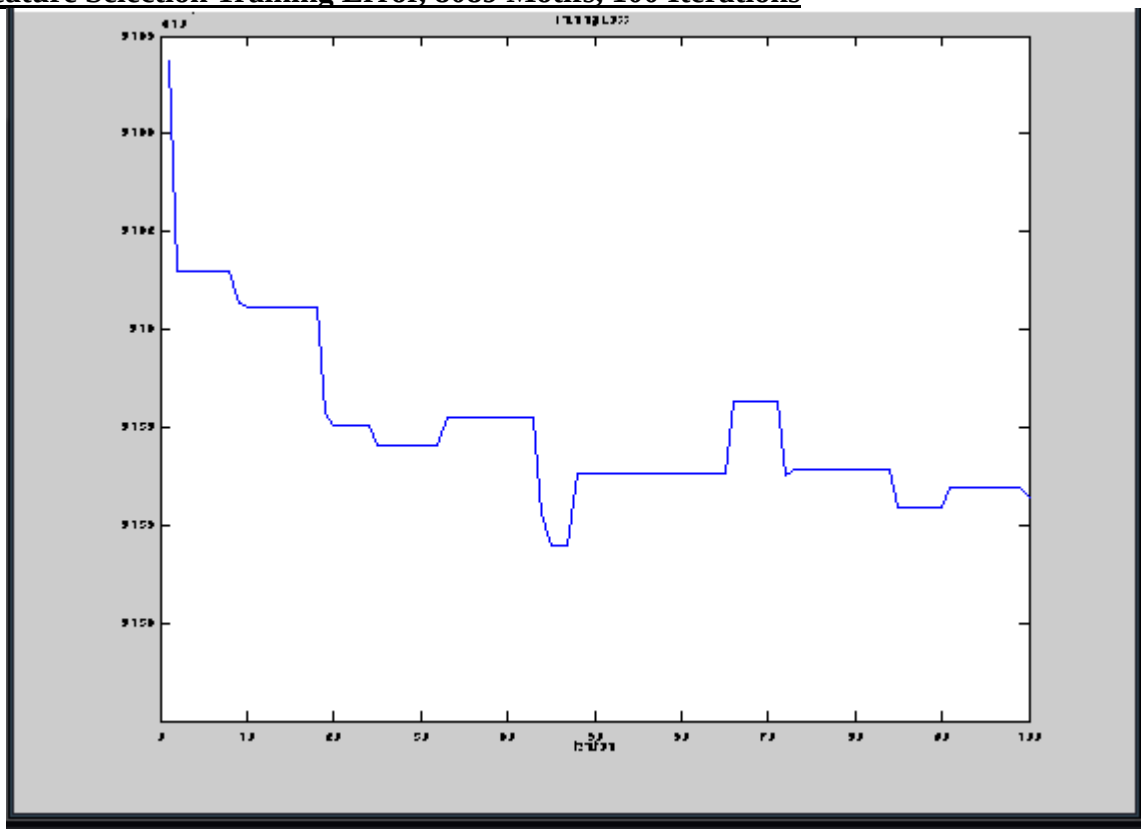
Jothi, Raja, and Przytycka, Teresa M., "Computational approaches to predict protein-protein and domain-domain interactions," National Center for Biotechnology Information.

Ng, Andrew, "CS 229 Notes," 2008, http://www.stanford.edu/class/cs229/materials.html

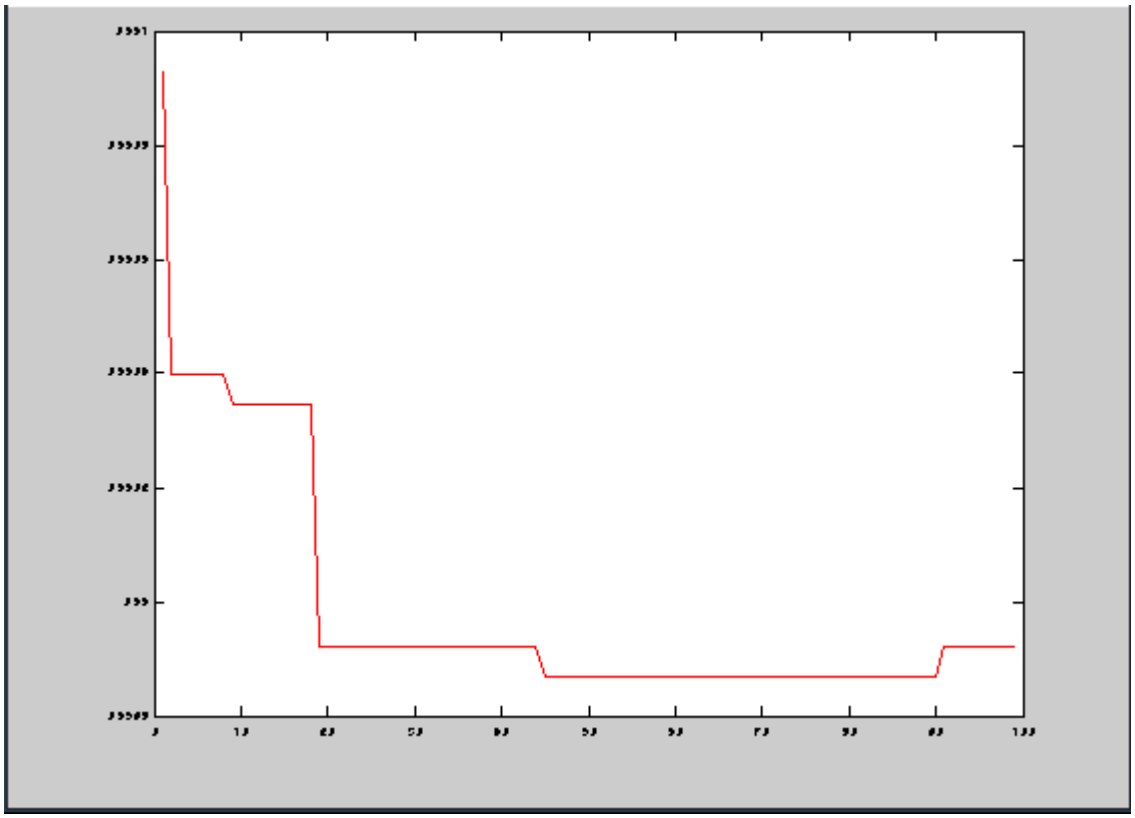
Schapire, Robert E., "The Boosting Approach to Machine Learning: An Overview," Nonlinear Estimation and Classification, Springer, ATT Labs, 2003.

Acknowledgment to Anshul Kundaje, Postdoc under Serfim Batzoglou and Arend Sidow, Department of Computer Science, Stanford University.

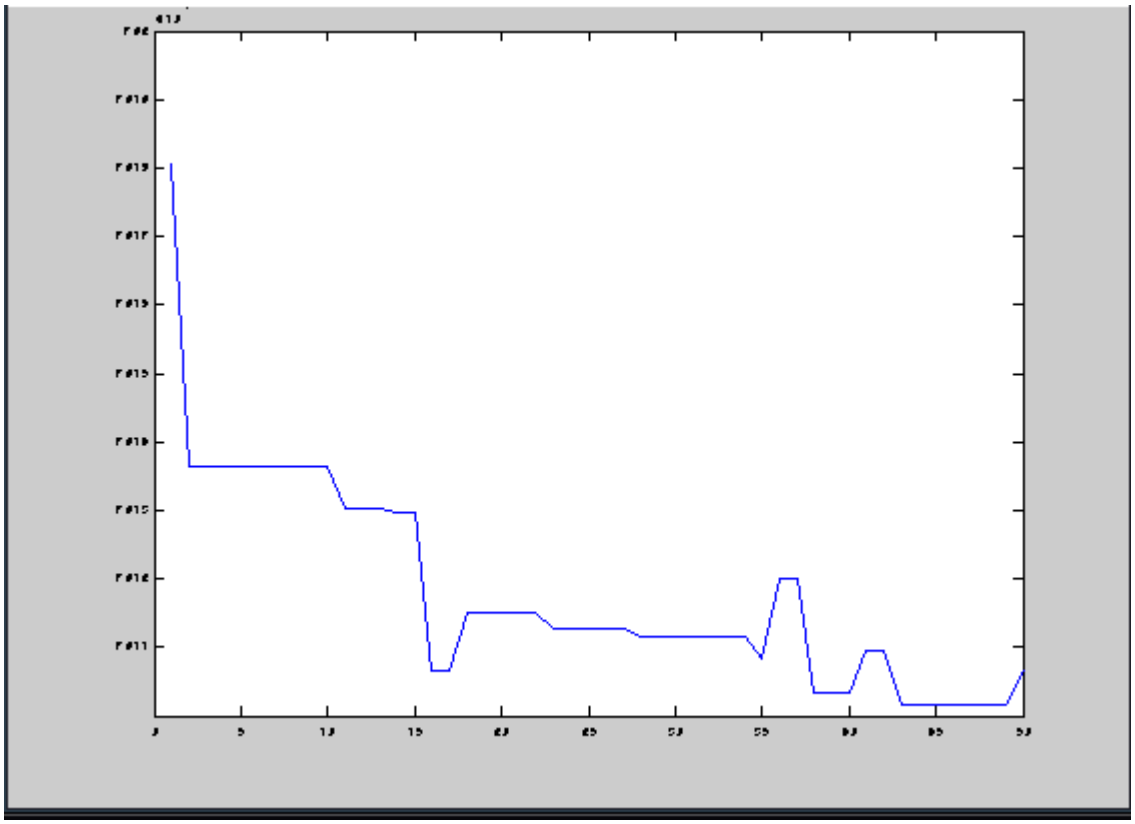
Full Feature Selection Training Error, 8089 Motifs, 100 Iterations



Full Feature Selection Test Error, 8089 Motifs, 100 Iterations



Smaller Feature Selection Training Error, 569 Motifs, 50 Iterations



Smaller Feature Selection Test Error, 569 Motifs, 50 Iterations