Zach Hoisington

BMI 203
Winter 2018
Final Project

Due Friday 03/13/2020 by 11:59pm
PST

You are to construct a machine learning system that will distinguish real binding sites of a transcription factor (RAP1) from other sequences. The standard methodology for learning and representing TF binding sites is based on positive data (examples of known binding sites of a TF) alone. One generally constructs a linear position weight matrix based on the positive examples along with some model of the background probabilities of observing various sequences. There are limitations with this method, among them being that the linear model cannot address interdependencies among the different positions in the TF motif. There are cases where people have identified TF motifs where positional interdependencies exist. Interestingly, some of these cases are isomorphic to the XOR problem that killed perceptrons a few decades ago. Your work will address this problem as well as provide a way around the explicit requirement of a background model for sequence probabilities.

The data provided include both positive examples of TF binding sites for the yeast transcription factor Rap1 along with negative examples comprised of a few thousand yeast promoters to which Rap1 is not known to bind. The specific input data files are the following:

1) 137 true positive binding sites from the yeast Rap1 transcription factor, each 17 base pairs long (rap1-lieb-positives.txt), and

2) Yeast genomic sequence as the negative training data. (yeast- upstream-1k-negative.fa) Each of the approximately 3000 fasta sequence contains 1000 bases upstream of a yeast ORF. By chance, there may be some examples of subsequences from the negative set that correspond exactly to one of the positive example sequences. If this occurs, you probably should not include the example in your training!

Your tasks for the assignment follow. Please answer the indented questions in your writeup.

1) Implement a feed-forward, three-layer, neural network with standard sigmoidal units. Your program should allow for variation in the size of input layer, hidden layer, and

output layer. You will need to write your code to support cross-validation. We expect that you will be able to produce fast enough code to be of use in the learning task at hand. You will want to make sure that your code can learn the 8x3x8 encoder problem prior to attempting the Rap1 learning task.

If your code is too slow or otherwise unworkable for the Rap1 learning task, you will need to show results that indicate your code worked adequately to learn the 8x3x8 encoder problem. In this case, after showing adequacy for the 8x3x8 encoder, you may obtain other neural network code, preferably in a language that runs quickly (C, Matlab, etc.).

As an alternative, after showing that your NN code can run the 8x3x8 encoder, you may choose to implement or obtain code for another machine learning method (e.g. support vector machines or KNN).

The encoder test case shows this

NOTE: If you are unable to successfully implement the neural network code directly yourself, you should not stop working. While it will be reflected in your project grade, you should continue with the Rap1 learning task using code for the core learning method obtained elsewhere. In this case, you should still document your effort toward making a NN that can learn the 8x3x8 encoder.

2) Set up the learning procedure to allow DNA sequences as input and to produce an output of the likelihood that an input is a true Rap1 binding site. You will want the output value to be a real number no matter what method you are using, since your performance on the blind test set will be judged based on the area under an ROC curve. So, please DO NOT threshold your output to produce a binary value.

- As mentioned above, you may use your ANN implementation. Many very successful projects have done so in the past. However, you may also use KNN, SVM, random forests, or any other approach that you see fit.

- Describe the machine learning approach in detail. This will include, for an ANN, a description of the network structure of your encodings of inputs and output. For an SVM, you will need to discuss input encoding as well as kernel function choices, etc... This will also include a description of the representation you have chosen for the input DNA sequences.

My neural network is, at a glance, pretty basic. I has an input layer, a hidden layer, and an output layer. There is also an architecture function where, for example the AutoEncoder at 8x3x8, you can decide the structure of the neural network preemptively. When running the NeuralNetwork without a predefined architecture, the input size is the length of the input. In this project, there are 137 sequences input. The compression size was chosen to be 6, a standard number. The output size is the size of y in this case. For the autoencoder, x and Y are the same size. The neural network works by accepting positive and negative inputs. The inputs are encoded using OneHot encoding. For each position in the 17 bases, there is a distribution of how many of those are a certain nucleotide for both the positive and negative sequences. It is fully connected and takes both positive negative inputs and trains weights. This is done using my “forward” function, which begins with random and normally distributed weights. The inputs are multiplied by the weights and transformed twice, once each layer, to give these probabilities for each nucleotide occurring. Weights It then back propogates and updates the weights as necessary. This is done using mean squared error and chain rule. When it sees novel test data, it finds the same aspects in the data and uses its predefined test weights to determine if the input is a true positive or negative, which it outputs.

The data was split into a 70% training set and a 30% testing set. Even though there were a lot more negatives to work with, an equal number of positives and negatives were used. The negatives were checked for positives which were then removed. the Network runs for 500 iterations and then stops. For K folds validation, there was a ratio of 100:1 Negatives to positives used to prevent these problems.

overweights the negative data, your system will probably not converge (it will just call everything a non-Rap1 site). Your system should be able to quickly learn aspects of the Rap1 positives that allow elimination of much of the negative training data. Therefore, you may want to be clever about caching negative examples that form "interesting" cases and avoid running through all of the negative data all of the time.

MORE NOTES: You don't have to use the full 17 bases of each binding site. You can use a subset if you think performance can be improved. For the negative training data, you do not have to use the reverse complement of all the sequences, but it may improve performance. **I did use all 17 bases in my algorithm.**

In the real world, ideally negative data would be in the minority. However, a 50/50 data distribution would cause unforseen biases. Therefore, by overweighting the negatives to positives 100:1 when training, the weights will instead have a bias towards positive data.

- How was your training regime designed so as to prevent the negative training data from overwhelming the positive training data?

- What was your stop criterion for convergence in your learned parameters? How did you decide this?

I removed one array from the learning that was used as a test against the newly trained weights. The model is evaluated against this after each epoch. If the performance of the model against the validation dataset begins to degrade, the learned parameters are saved and the model does not continue to overtrain.

4) Given your learning system, perform cross validation experiments first to see whether it is working, and then see how well your system performs.

- Describe how you set up your experiment to measure your system's performance.

The performance was cross validated and tested using the AUROC and accuracy and then graphed for all 10 k's. The better the AUC, the better the prediction
The accuracy was calculated using sklearn metrics.accuract which takes into account true and false positives and returns a percentage.

- What set of learning parameters works the best? Please provide sample output from your system.

- What are the effects of altering your system (e.g. number of hidden units or choice of kernel function)? Why do you think you observe these effects?

- What other parameters, if any, affect performance?

Some other performance affecting parameters are the amount of samples input or alpha.

5) To provide an objective measure of your neural network's ability to recognize binding sites, a test dataset has been provided (rap1-lieb-test.txt). There are no class labels on these sequences.

Apply your training regime using the parameters and procedure that you optimized for your cross-validation results to all the available training data. Run the trained system on the test dataset. For each sequence, output the sequence and its output value from the network, separated by a tab, as follows:

Through K Folds I attempted two different types of ensembling methods to find the best model that I should use to make my final predictions for the submission. The two types of ensembles include:
1) Taking the average unseen data predictions for each k folds model and average these prediction
2) Taking the average weights of the k folds cross validation and make a "best" weight
I kept a small subset of the data out to test so that I could then again plot the accuracy of the predictions when using the model on this test set after a training.

ACATCCGTGCACCATTT 0.927
AAAAAAACGCAACTAAT 0.123

If you do not use the full 17 bases, output the sequence subset as an additional column (column 3), and in your writeup specify what portion of the sequence you used. In the above example, the first sequence is a bona fide Rap1 site and the second is not. They are correctly ranked relative to one another.

6) **To complete this project:**

- Comment code: Intuitive descriptions showing you understand what all steps are doing (and why) must be included.

- Email a single pdf (name = JaneSmith_BMI203_FinalProject.pdf) to sali@salilab.org, miriam.goldman@ucsf.edu, and Laura.Gunsalus@ucsf.edu with a cogent and complete description of the experiments you undertook to optimize your system's performance on the training data. In particular, answer the questions listed above. In addition, you will turn in the output of your program on the test data in a file named predictions.txt, formatted as described in (5). This will be used to evaluate your trained network's performance.

- A link to your Github repository

    - Make sure there is a link to the Travis build results for your repo in the README file

    - Note that only commits prior to the due date will be considered!

Your goal is to maximize the separation in scores for the true Rap1 binding sites as compared with the non-sites. You should design your cross-validation experiments carefully so that you can identify a set of training conditions that maximizes your system's performance.

There will be a $50 cash prize awarded to the best ROC area performance on the test data, subject to the belief that the results obtained reflect an honest application of the learning procedures described in your writeup. There will be an additional $50 bonus if the winning entry exceeds the best prior ROC area (0.98386 in 2016 from Seth Axen).