Assignment Goals

- Use mutual information to reconstruct gene expression networks
- Evaluate classifier predictions
- Control for multiple hypothesis testing with *q*-values
- Include epigenetic information in motif finding
- Predict target genes of enhancers

Instructions

- To submit your assignment, log in to the biostat server `mi1.biostat.wisc.edu` or `mi2.biostat.wisc.edu` using your biostat username and password.
- Copy all relevant files to the directory `/u/medinfo/handin/bmi776/hw2/<USERNAME>` where `<USERNAME>` is your biostat username. Submit all of your Python source code and test that it runs on `mi1.biostat.wisc.edu` or `mi2.biostat.wisc.edu`. Do not test your code on `adhara.biostat.wisc.edu`.
- For the rest of the assignment, compile all of your answers in a single file and submit as `solution.pdf`.
- Write the number of late days you used at the top of `solution.pdf`.
- For the written portions of the assignment, show your work for partial credit.

Part 1: Mutual information in regulatory networks

In class, we saw how FIRE uses mutual information to detect relationships between sequence motifs and gene expression levels. Mutual information is also a popular technique for reconstructing transcriptional regulatory networks from gene expression datasets[1]. After measuring gene expression levels for all genes in a sufficient number of biological conditions, mutual information can detect some types of pairwise dependencies that may suggest one gene is a regulator (transcription factor) and another is its target. Fluctuations in the regulator's expression can influence the expression levels of the target gene. We can create an undirected gene-gene network by computing mutual information for all pairs of genes. Thresholding the mutual information produces a set of gene-gene edges.

[1] http://link.springer.com/article/10.1186%2F1471-2105-7-S1-S7

For this assignment, we will use the DREAM3[2] network inference challenge dataset. The file `data.txt` has a 21 time point simulation of the gene expression levels for ten genes over four simulated replicates. The first line of the file provides the column labels. The first column is the time point, which you will not need. The other columns are the expression levels of the gene named in the first line, where genes are represented with a numeric index. The file is in a tab-delimited format.

***1A**: Mutual information implementation*

Write a program `CalcMI.py` that takes as input the expression data for a set of genes over a number of conditions and reconstructs pairwise gene-gene dependencies using mutual information. The program will output the list of gene-gene dependencies and their mutual information. It should only consider the dependencies between unique genes, not the mutual information of a gene and itself (the entropy of that gene's expression).

Compute mutual information by discretizing the gene expression values, mapping continuous gene expression into discrete bins. For a pair of genes, for example G1 and G2, construct a count matrix that tracks the number of times G1's expression is in some bin a and G2's expression is in some bin b. Add a pseudocount of 0.1 to all entries in the count matrix. From this count matrix, you can compute the terms $P(G1 = a)$, $P(G2 = b)$, and $P(G1 = a, G2 = b)$ needed to calculate mutual information.

You can run your program from the command line as follows:

```
CalcMI.py --dataset=<dataset> --bin_num=<bins> --out=<out>
```

where

- `<dataset>` is the name of the text file that contains the gene expression data formatted according to the description above.
- `<bin_num>` is an integer that represents the number of bins that should be used to discretize the continuous gene expression data when calculating the mutual information. In this part, you should use a *uniform* binning of the gene expression range. For example, if a gene has expression values in the range [1, 11] and `bin_num = 4`, the bins

[2] http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0009202

would be [1, 3.5), [3.5, 6), [6, 8.5), and [8.5, 11]. Later, you will consider an alternative strategy so make your code flexible.

- `<out>` is the name of the text file into which the program will print *all* unique gene pairs and their mutual information values in decreasing order. Round mutual information to three decimal places. Break ties based on the index of the first gene and then the index of the second gene if needed, sorting genes indexes in ascending order[3]. Each line in the file should contain the undirected gene-gene edge and its mutual information separated by a tab '\t'. The file should be formatted as follows:

```
(6,9)       0.817
(3,10)      0.633

...         ...

(5,8)       0.480
(3,7)       0.463

...         ...

(8,9)       0.427
```

***1B**: Plot the receiver operating characteristic (ROC) curve*

Write another program `plot.py` to plot the ROC curve[4] for the output of `CalcMI.py`. The gold standard edges, that is, the edges in the true gene-gene network, are tabulated in a file called `network.txt` as two columns. Each line represents an undirected edge between two different genes. The genes in an edge are always listed with the smaller index first.

You will have to plot a graph using Python. A plotting example is given as `plot_ex.py`, where you can learn the basic usage of `matplotlib`, a package for plotting in Python. For the detailed API, refer to http://matplotlib.org/users/tutorials.html

Your program should also calculate the area under the ROC curve (AUC ROC). This summarizes the entire curve, where 1.0 is perfect classification and 0.5 is random classification. You should calculate the points on the

[3] See https://docs.python.org/3/howto/sorting.html for sorting tips
[4] The ROC curve is defined in the assigned reading Lever et al. (2016). More detailed instructions for computing it are on slide 24 of the CS 760 slides http://pages.cs.wisc.edu/~dpage/cs760/evaluating.pdf

curve yourself but may use `numpy` to compute the area under the curve, as shown in `plot_ex.py`.

You can run your program from the command line as follows:

```
plot.py --MI=<MI> --gold=<gold_network> --name=<name>
```

where

- `<MI>` is the path of the output file from `CalcMI.py`.
- `<gold_network>` is the gold standard network file for calculating TPR and FPR.
- `<name>` is the name for your plot image. `plot.py` will automatically append `.png` so only provide a name without the file type extension.

***1C**: Try a different binning strategy*

In this part, you will try a different binning strategy for the gene expression data. Add a new argument `bin_str` to implement use the equal density binning strategy. This uses a percentile-based assignment to assign expression values to bins. For example, with `bin_num = 2` and `bin_str = density`, the lowest 50% of a gene's expression values would be mapped to bin 0 and the highest 50% would be mapped to bin 1. `bin_str` should accept `uniform` and `density` as valid arguments.

You can run your program from the command line as follows:

```
CalcMI.py --dataset=<dataset> --bin_num=<bins> --bin_str=density --out=<out>
```

After you compute the mutual information, plot the ROC curves and calculate the AUC ROC scores for these two argument combinations:

```
bin_num = 8       bin_str = uniform
bin_num = 7       bin_str = density
```

Include the plots in your handin directory and the AUC ROC scores in your solution PDF. Which combination gave the best AUC ROC?[5]

[5] Note that in a real research problem it would be improper to select the hyperparameters (the number of bins and binning strategy) based on the AUC ROC on the gold standard network. Also note that AUC ROC is not actually an appropriate metric for this task because the gold standard has few positive edges.

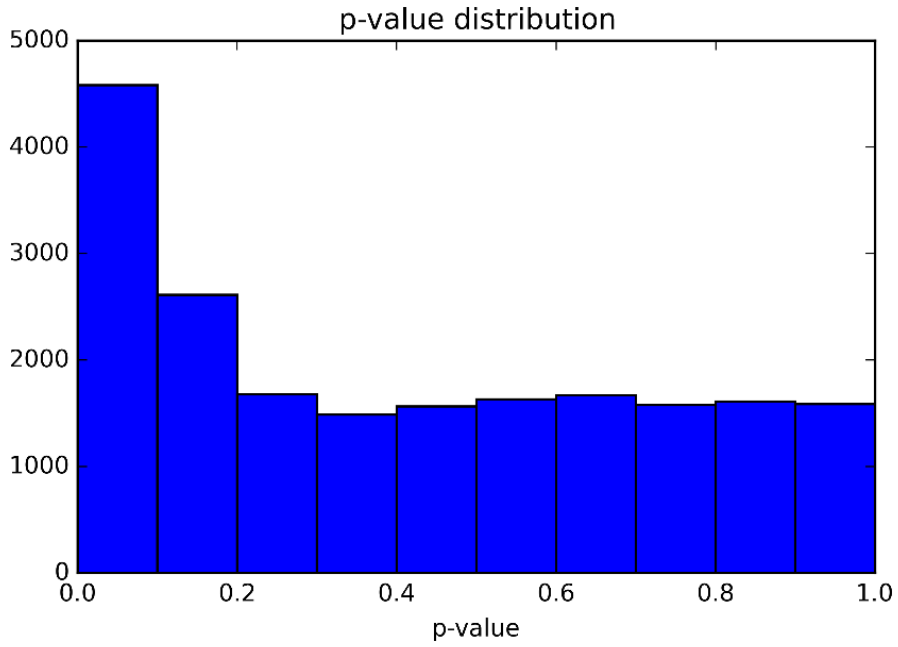
Input files and template Python files can be downloaded from
https://www.biostat.wisc.edu/bmi776/hw/hw2_files.zip

Part 2: Calculating *q*-values

You will manually calculate *q*-values from a *p*-value distribution.

2A: Estimating λ

First, use the histogram of the *p*-value distribution below to estimate λ visually as in Storey and Tibshirani 2003. The distribution contains *p*-values for 20000 features. Estimate λ to the nearest 0.1 and report the value you estimated.



2B: Estimating $\hat{\pi}_0(\lambda)$

Use the table below to estimate $\hat{\pi}_0(\lambda)$ for the value of λ that you selected. Report the $\hat{\pi}_0$ you estimated rounded to two decimal places.

λ	$\#\{p_i > \lambda; i = 1 \ldots m\}$
0.0	**20000**
0.1	**15418**
0.2	**12805**
0.3	**11126**
0.4	**9641**
0.5	**8076**
0.6	**6445**
0.7	**4776**
0.8	**3196**
0.9	**1586**

***2C**: Calculating* q-*values*

Although 20000 features have been tested, only the top 10 features ranked by *p*-value are listed below:

Rank	p-value
1	**0.000025**
2	**0.000048**
3	**0.000048**
4	**0.000060**
5	**0.000069**
6	**0.000076**
7	**0.000085**
8	**0.000132**
9	**0.000154**
10	**0.000159**

Calculate the *q*-value for these 10 features rounded to three decimal places. You may assume that the *q*-values for the remaining features not shown in the list above do not affect the *q*-values of these 10 features.

Part 3: Epigenetic information in motif finding

Suppose you are studying a transcription factor and suspect that its DNA binding preferences are affected by DNA methylation. You would like to include DNA methylation status into a motif-finding algorithm. Specifically, you have base-pair resolution methylation data that tells you whether each cytosine (C) nucleotide is in the unmethylated form or the methylated form 5-methylcytosine (5mC).

Briefly describe how to extend MEME to account for cytosine methylation. You may assume that your input sequences follow the OOPs model.

Part 4: Enhancer targets

We saw in class that DNA looping can bring proteins bound to enhancer sites into contact with proteins bound to promoter sites, enabling the enhancer-bound proteins to influence the transcription of genes that are up to 1 million base pairs away. Hi-C data provide a direct way to detect which distant genes may be influenced by an enhancer site, which we will call *enhancer targets*. However, as a bioinformatician you are often forced to work with the data types that are available instead of the data that would be ideal to address a biological question. Here we will explore how to use other types of data to predict enhancer targets.

For each of the two scenarios below, briefly outline an algorithm that would enable you to predict which genes in a 2 million base pair window (1 million upstream and 1 million downstream of the enhancer site) might be regulated by an enhancer site. The outline does not need to contain full algorithmic details, only the major steps. For instance, an acceptable outline for PIQ would be:

- Scan PWMs to identify candidate binding sites
- Estimate background DNase-Seq read density with a Gaussian process
- Build motif-specific DNase profiles using the top-scoring candidate binding sites
- Iteratively update estimated mean and variance of the DNase-Seq profiles for each condition, the predicted binding sites, and the functions mapping motif match strength and read counts to priors on binding

***4A**: Genetic variation*

Suppose you have data from a resource like GTEx that provides genotypes and tissue-specific gene expression for a large number of individuals. How could you use this information to predict enhancer targets? It is okay if your approach cannot make predictions for all enhancers or all genes.

***4B**: Histone modifications*

The histone modification H3K27ac indicates active enhancer sites. Suppose you have H3K27ac and gene expression data from a resource like ENCODE that covers hundreds of cell lines. How would you predict enhancer targets?