University of Wisconsin-Madison	Spring 2019
BMI/CS 776: Advanced Bioinformatics	Homework #2
Prof. Colin Dewey	Due: Mon, Mar 11, 2019 11:59 PM

Assignment goals

- Use mutual information to reconstruct gene expression networks
- Evaluate classifier predictions
- Resolve ambiguity in RNA-seq quantification
- Reason about probabilistic splice graphs
- Use the Cufflinks algorithm to assemble transcripts

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`** without error. 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.

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.

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

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

***1A**: Mutual information via discrete binning*

Complete the program **`CalcMI.py`** using the provided template 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 pairs of 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, as in HW0. 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 G1:G2 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 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. After rounding 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:

[3] See https://docs.python.org/3/howto/sorting.html for sorting tips

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

...       ...

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

...       ...

(8,9)     0.427
```

***1B**: Implement a different binning strategy*

In this part, you will try a different binning strategy for the gene expression data. Add a new argument **`str`** to implement the equal density binning strategy. This uses a percentile-based assignment to assign expression values to bins, as in HW0. For example, with **`bin_num = 2`** and **`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.

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

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

***1C**: Mutual information via kernel density estimation*

The definition of mutual information naturally extends to continuous random variables. In this part, you will calculate mutual information of gene pairs on the same data using a kernel density estimator with a Gaussian kernel. Kernel density estimation is a nonparametric method for learning a smooth probability density function (pdf) from observed data. A kernel density estimator for n training points $x_1, \cdots, x_n$ is defined to be

$$\widehat{f_n}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where the *kernel* K enables the smoothing effect and the *bandwidth* h determines how influential a point is in its neighborhood.

To abstract away the technical details of the kernel, you are *required* to use the **`SciPy`** function **`gaussian_kde`** for estimating the pairwise joint probability distribution of gene expression levels[4]. Continuous mutual information is defined with a double integral over both variables. To avoid the integration, you will sample the estimated pdf on a 100×100 grid of points uniformly distributed in the region [-0.1, 1.1]×[-0.1,1.1] and use the sampled densities to approximate the estimated pdf[5]. Add 0.001 to the density at each sampling location in the grid to avoid precision error. The marginal probability of a single gene expression level can then be estimated via approximate integration over the other dimension (that is, a finite sum over 100 intervals defined by the grid). Similarly, mutual information of a gene pair can be computed via approximate integration over the region [-0.1, 1.1]×[-0.1,1.1] (that is, a finite sum over 100×100 squares defined by the grid).

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

```
CalcMI.py --dataset=<dataset> --str=kernel --out=<out>
```

[4,5] See https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.stats.gaussian_kde.html for examples, including **`np.mgrid`** syntax. The integration will also be discussed on Piazza.

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

Complete another program **`plot.py`** to plot the ROC curve[6] 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. Code for generating the plot and computing the area under the ROC curve (AUROC) is provided. You task is to add code for calculating the points on the curve.

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`**, formatted and sorted as in ***1A***
- **`<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.

***1E**: Evaluate your predictions*

After you compute the mutual information, plot the ROC curves and calculate the AUROC scores for the three following scenarios:

```
bin_num = 7          str = uniform
bin_num = 9          str = density
                     str = kernel
```

Include the mutual information output files and plots in your handin directory and the AUROC scores in your solution PDF. Which combination gave the best AUROC?[7]

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

[6] 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

[7] 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 AUROC on the gold standard network. Also note that AUROC is not actually an appropriate metric for this task because the gold standard has few positive edges.

Part 2: RNA-seq Rescue Algorithm

The full RSEM algorithm is too complicated to execute manually, but we can use the RNA-seq rescue method presented in class to approximate one iteration of expectation maximization. The bipartite graph (Figure 1) contains two types of nodes: transcripts and read groups. The transcript nodes contain a transcript id and the transcript length in base pairs (bp). The read nodes contain the read counts for a group of reads that all align to the same transcripts. Transcript-read group edges designate the transcripts to which each read group aligns.

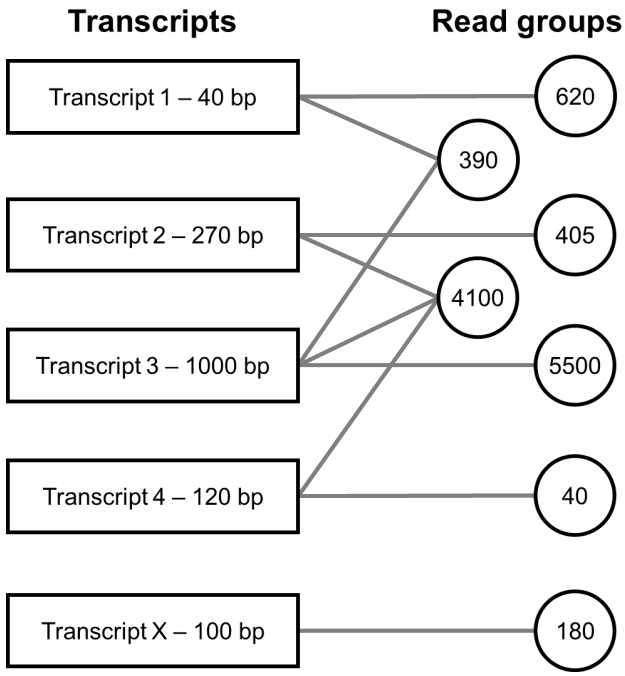

Figure 1: RNA-seq bipartite graph

2A: *Estimating relative abundance*

Use the rescue method to calculate the relative abundance for the five transcripts to three decimal places. Show your work for partial credit.

2B: *Estimating absolute abundance*

Transcript X is a RNA spike-in. 1000 copies of transcript X were mixed into the experimental sample when preparing the sample for RNA-Seq, meaning its absolute abundance is 1000. Use the relative abundances you calculated above to calculate the absolute abundances for the other four transcripts, rounded to the nearest whole number. Show your work for partial credit.

Hint: Review the "Issues with relative abundance measures" slide from the RNA-seq lecture. Given the relative abundances for all six genes and the absolute abundance of Gene 6, you can derive the absolute abundances of Genes 1 through 5.

Part 3: RNA-seq transcript assembly

In this problem you are to determine how Cufflinks would assemble the aligned RNA-seq reads shown in Figure 2. The shown read data are from single-end sequencing and the dotted lines indicate gaps in the alignment of a read to the genome, which would imply the presence of an intron.

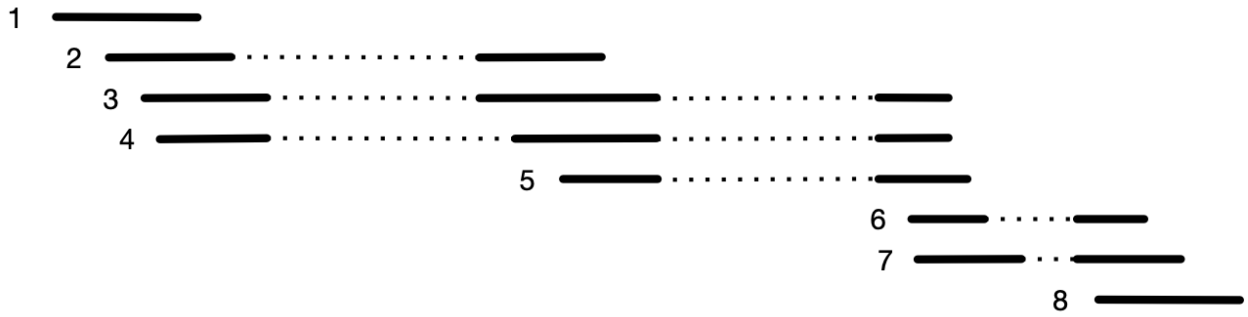

Figure 2: Aligned RNA-seq reads for Part 3

3A: *Constructing the read overlap graph*

Draw the read overlap graph for the reads in Figure 2.

3B: *Transitive reduction*

Draw the transitive reduction of your read overlap graph from part A.

3C: *Constructing the reachability graph*

Draw the bipartite reachability graph that Cufflinks would construct based on your read overlap graph from part B.

3D: *Maximum matching and minimum vertex cover*

In your reachability graph from part C, find (i) a maximum matching and (ii) a minimum vertex cover, which should have equal cardinality by König's theorem.

3E: *Largest antichain and minimum chain partition*

Given your minimum vertex cover from part D, determine (i) the reads that form the largest antichain and (ii) a partition of the reads into the smallest number of chains (by Dilworth's theorem, the size of your antichain in (i) should equal the number of chains in (ii)).

3F: *Assembled transcript structures*

Given the chains you found in 3E, give the structures of the transcripts that would be assembled by Cufflinks.

Part 4: Probabilistic splice graphs

In this problem we will examine how probabilistic splice graphs may produce a more compact representation for the possible isoform structures of a gene and their frequencies. Shown in Figure 3 are the six possible isoform structures for a gene.

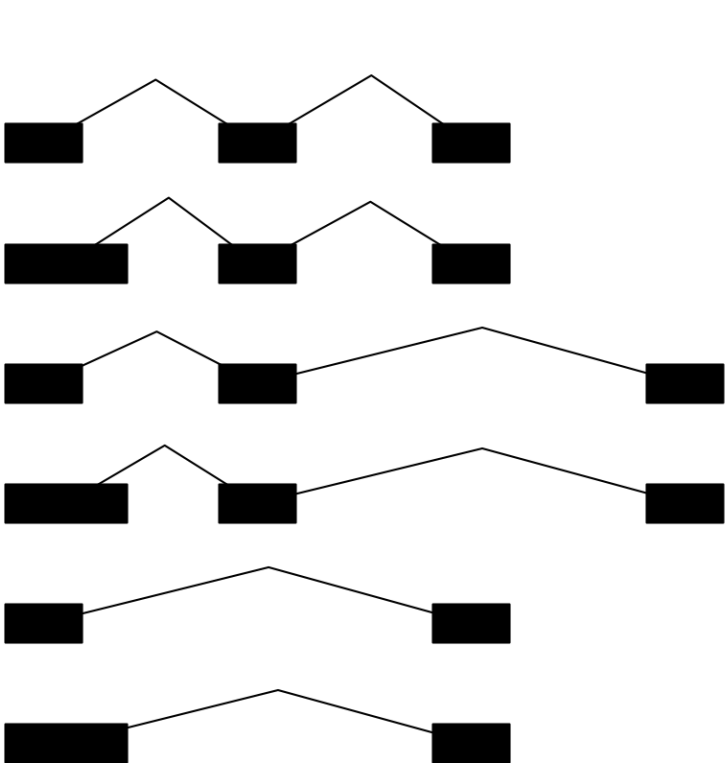

Figure 3: Six isoform structures of a gene and the frequencies of those structures in two scenarios A & B.

4A: For the frequencies of the isoforms given in scenario A in Figure 3, give the most compact probabilistic splice graph representation for this gene.

4B: For the frequencies of the isoforms given in scenario B in Figure 3, give the most compact probabilistic splice graph representation for this gene.