02-510/710: Computational Genomics, Spring 2025

HW2: RNA-seq data analysis

Version: 1
Due: 23:59 EST, Feb 21, 2025 on Gradescope

Topics in this assignment:

1. Normalization
2. Differential gene expression analysis
3. Cell type classification

What to hand in.

- One write-up (in pdf format) addressing each of following questions.
- All source code. If the skeleton is provided, you just need to complete the script and send it back. Your code is tested by autograder, please be careful with your main script name and output format.

Submit the following file which contain the completed code and the pdf file to gradescope separately.

./S2025HW2.pdf
./normalization.py
./de_genes.py
./classification.py

Please note that all the solutions and code must be your own. We will check for plagarism after the final submission.

1. **[40 points] Normalization**

In this question, we will implement two different normalization techniques on real RNA-seq data.

Data Description

We provide data from a study in which a human cell line (A549) was treated with a specific drug (dexamethasone). Samples were collected following treatment to create a time course. For this question you would use 24 samples for which raw RNA-seq data is provided (file: ReadCounts.txt). The table is of size $n \times m$ where n is the total number of genes and m is the total number of samples. We have $\sim 8K$ genes in the matrix.

You have also been provided with the lengths of each gene present in the data matrix (file: GeneLengths.txt).

Your task

(a) (10 points) Normalize the data using two normalization techniques, i) RPKM, ii) size factor normalization.

RPKM normalizes data by the total number of reads and the length of each transcript. This normalization technique, however, is not always effective since few, very highly expressed genes can dominate the total count and skew the expression analysis.

Another normalization technique consists of computing the effective library size by considering a size factor for each sample. By dividing each sample's counts by the corresponding size factors, we bring all the count values to a common scale, making them comparable. Intuitively, if sample A is sequenced N times deeper than sample B, the read counts of non-differentially expressed genes are expected to be, on average, N times higher in sample A than in sample B, even if there is no difference in expression.

If we expect that most genes are not different between the samples, we can use the median ratio to estimate these factors. Thus, to estimate the size factors for each sample, we take the median of the ratios of observed counts k_{ij} to their geometric mean.

$$s_j = \underset{i}{\text{median}} \frac{k_{ij}}{(\prod_{v=1}^{m} k_{iv})^{\frac{1}{m}}}$$

After computing the size factors, we can normalize the read counts by dividing each sample by their corresponding size factors.

$$\hat{k_{ij}} = \frac{k_{ij}}{s_j}$$

You have been provided with a skeleton python script *normalization.py*. Complete the functions *rpkm*, $size\_factor$ in the *normalization.py* script.

Note: Your code is graded by an autograder. The script should be able to run with command line: `python normalization.py ReadCounts.txt GeneLengths.txt` All the normalization functions should return a $n \times m$ array.

(b) (10 points) Submit the boxplots of the log2 converted count data for each sample for the i) raw counts, ii) RPKM normalized counts, iii) size factor normalized counts.

Note: Your plots will not be graded by the autograder. Include the plots in the pdf file.

Solution

(c) (10 points) Comparing the 3 boxplots, which normalization technique would you choose for this dataset? Explain.

Solution

(d) (10 points) Let $R(g_1)$ and $R(g_2)$ be the raw read counts (number of reads mapped to a gene) for genes g_1 and g_2 in a sample. And $N(g_1)$ and $N(g_2)$ be the normalized read counts for genes g_1 and g_2 in a sample. Suppose $R(g_1) > R(g_2)$, for the following questions choose ALL answers that could be correct and briefly explain your answer.

i. If we use RPKM normalization then:

A. $N(g_1) > N(g_2)$

B. $N(g_1) = N(g_2)$

C. $N(g_1) < N(g_2)$

Solution

ii. If we use scale factor normalization then:

A. $N(g_1) > N(g_2)$

B. $N(g_1) = N(g_2)$

C. $N(g_1) < N(g_2)$

Solution

2. **[30 points] Differential gene expression analysis**

In this exercise we will complete steps to identify differentially expressed genes from the size factor normalized read count data you saved as 'size_factor_normalized_counts.txt', from Q1. We will also learn how to perform multiple hypothesis testing to identify significant differentially expressed genes.

You have been supplied with the numerical sample labels in *labels.txt*. There are two types of samples, 1) cell treated with dexamethasone(case) and 2) cell treated with ethanol (control). The gene names have been provided in *GeneNames.txt* file.

Note: Your code for this problem is graded by an autograder. The script should be able to run with command line:

```
python de_genes.py
```

Include requested plots in the pdf file.

Your task

(a) (10 points) Dispersion of gene i in condition C, $C \in \{Case, Control\}$ can be computed as

$$disp_i^C = \frac{std(\hat{k}_{i,C})}{mean(\hat{k}_{i,C})}$$

Submit a log2-log2 scatterplot of dispersion vs mean in the size factor counts dataset, for the two conditions (but with all the data on one plot). Use different colors for case and control condition. How would you interpret this plot, and why?

Solution

(b) (10 points) Compute the log2 fold change of the genes in the two conditions. Select top 10 upregulated genes, and top 10 downregulated genes, and submit a gene expression heatmap for the selected genes. Place both upregulated and dowregulated genes on the same heatmap. Are these the most useful genes to investigate? Why or why not?

Solution

(c) (10 points) Determining whether the gene expressions in two conditions are statistically different consists of rejecting the null hypothesis that the two data samples come from distributions with equal means. To do this, we can calculate a p-value for each gene.

However. thresholding P-values to determine what fold changes are more significant than others is not appropriate for this type of data analysis, due to the multiple hypothesis testing problem. When performing a large number of simultaneous tests, the probability of getting a significant result simply due to chance increases with the number of tests. In order to account for multiple testing, perform a correction (or adjustment) of the P-values so that the probability of observing at least one significant result due to chance remains below the desired significance level. We can use The Benjamini-Hochberg (BH) adjustment.

The BH is defined as:
Let $p_1 \leq ... \leq p_n$ be ordered p-values. Define

$$k = i : p_i \leq \frac{i}{n}\alpha$$

and reject $H_0^1 ... H_0^k$. α is the false discovery rate(FDR) to be controlled. If no such i exists, then no hypothesis will be rejected.

Implement the BH correction function in `de_genes.py`. The implementation must be your own.

3. **[30 points] Cell type classification**

While cells from the same individual share (roughly) the same DNA sequence, different cells use different subsets of these genes, and at different levels. Single-cell RNA sequencing (scRNA-seq) measures the activity levels of a set of annotated genes in a single cell, a vector with continues values that is termed 'gene expression profile'. The type of a cell (e.g., lung cell, brain cell, skin epidermis) is closely connected with its gene expression profile.

In this problem you will use a combination of unsupervised and supervised learning techniques to explore and classify cells from a large single-cell RNA sequencing (scRNA-seq) dataset. This is an important goal. For example, when taking a cancer biopsy the sample contains several different types of cells and the ability to accurately determine the cell composition can have important impact on the type of treatment prescribed.

Data Description We will implement several learning algorithms in Python and test them on an RNA-seq dataset. All the required data is in the `dataset/` folder. The two files `train_gene_expression.npz` and `test_gene_expression.npz` are sparse SciPy/Numpy-formatted gene expression matrices. Each row of the matrix is a cell, with its expression value for each gene in a separate column; there are a total of 22289 genes (columns) in each matrix. In addition, the two files `train_labels.npy` and `test_labels.npy` contain cell types labels for each row in the training and testing datasets, respectively; there are a total of five cell types present in both the train and test data.

Before starting: make sure you have Numpy, Matplotlib, Pandas, Seaborn, SciPy, SciKit Learn and PyTorch (the CPU version is sufficient) installed on your machine. If you have trouble installing these packages, please come to office hours for help.

Your task:

Note: Your code is graded by an autograder. You have been given a skeleton code clustering.py. You have to complete 3 functions (`reduce_dimensionality_pca`, `plot_transformed_cells`, and `train_and_evaluate_rf_classifier`). The details about the functions are given later. At the end, the script should be able to run with command line:

```
python classification.py dataset/train_gene_expression.npz \
dataset/test_gene_expression.npz dataset/train_labels.npy \
dataset/test_labels.npy "rf_pipeline"
```

The formats of arguments and return values of each function are explicitly described in the skeleton code. For the hyperparameters, please use the **default** values specified in the skeleton code.

(a) (10 points) Not all genes are useful for cell type classification. To save time and improve accuracy we can cluster using only a subset of the genes. One common criteria to select genes is dispersion, which is defined as the variance divided by mean. The `get_top_gene_filter` function is to compute a mask that selects only the columns of the expression matrix that are the top 2000 most dispersed across all cells in the training dataset.

As another preprocessing step, we will use Principal Component Analysis (PCA) to reduce the dimensionality of the filtered genes. PCA is an unsupervised learning technique that computes linear combinations of features that best capture the axes of variance in the high-dimensional feature space. **Complete** the function `reduce_dimensionality_pca`, which performs PCA on the filtered expression data and returns the transformed training data and test data. Please use the SciKit Learn implementation of PCA (see documentation here).

(**Hint:** for best results, concatenate the filtered training and expression data together and input the combined data for training the PCA model.)

(b) (10 points) To visualize the appropriateness of PCA embedding for the cell classification task, we can plot our cells in the 2D plane spanned by the top 2 principal components. **Complete** the `plot_transformed_cells` function, which takes as input the transformed gene expression data and plots the cells using only the first two principal components represented in the transformed data. Your plot should color the cells by their cell type label. Apply the completed function to the PCA-transformed *training* gene expression data, and **attach an image** of your plot below. **Comment** on how clear the difference between cell types is in your plot.

(**Hint:** consider using Pandas + Seaborn for plotting and labeling.)

Solution

(c) (10 points) Next, we will implement a pipeline that uses a Random Forest classifier to classify our cells. The Random forests algorithm is an ensemble learning method that operates by constructing a multitude of decision trees at training time. **Complete** the `train_and_evaluate_rf_classifier` function, which trains a simple Random forests pipeline on the PCA-reduced gene expressions and returns the trained model as well as the classification accuracy on the test data. Please use the SciKit Learn implementation of the Random forests classifier (see documentation here and here).

Last, we will combine the above functions to generate a pipeline for cell-type classification. The (`mode =="rf_pipeline"`) branch within the `__main__` method of the Python script has been provided for you, which combines the gene filtration, PCA dimensionality reduction and Random forests classification steps and prints the final score of your classifier. **Report** the training and test accuracy of your classifier below.

Solution