02-510/710: Computational Genomics, Spring 2022

HW2: RNA-seq data analysis

Version: 1
Due: 23:59 EST, Mar 2, 2022 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. Please use the latex template included in the handout.
- 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.

./S2022HW2.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. **[25 points] Normalization**

In this question, we will implement three 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 $\sim 9K$ 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) (15 points) Normalize the data using three normalization techniques, i) RPKM, ii) TPM, and iii) size factor normalization.

Both RPKM and TPM normalize 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*, *tpm*, *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
```

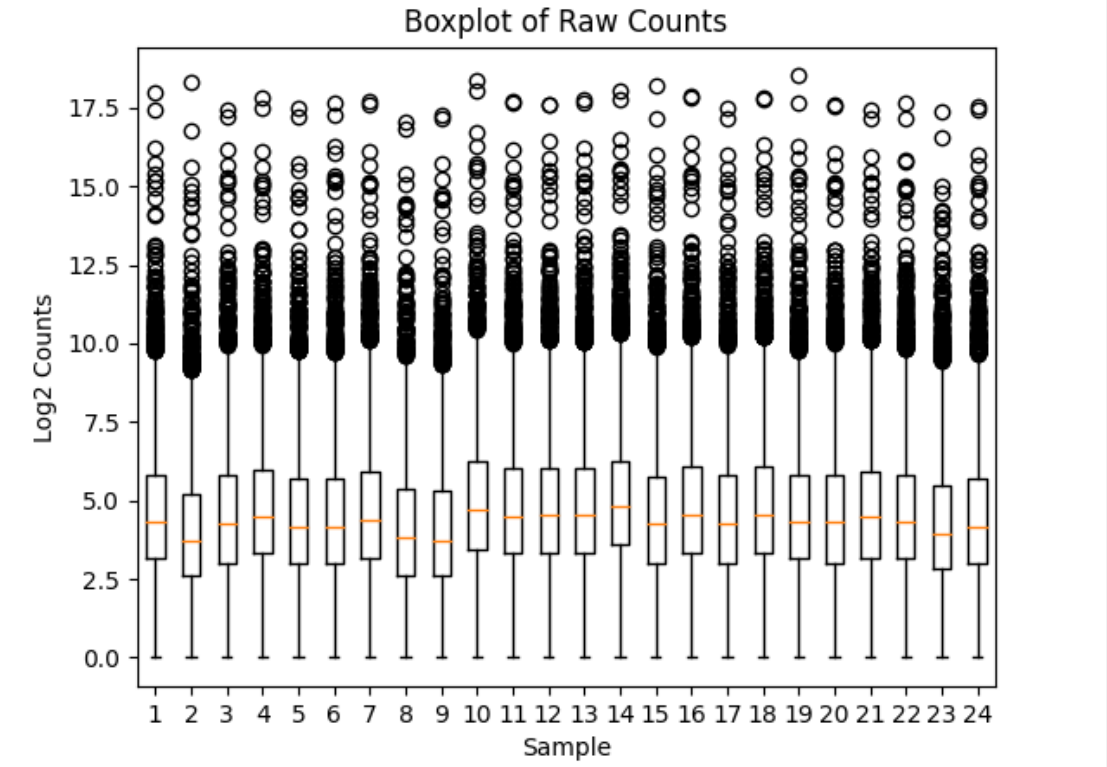
All the normalization functions should return a $n \times m$ array.

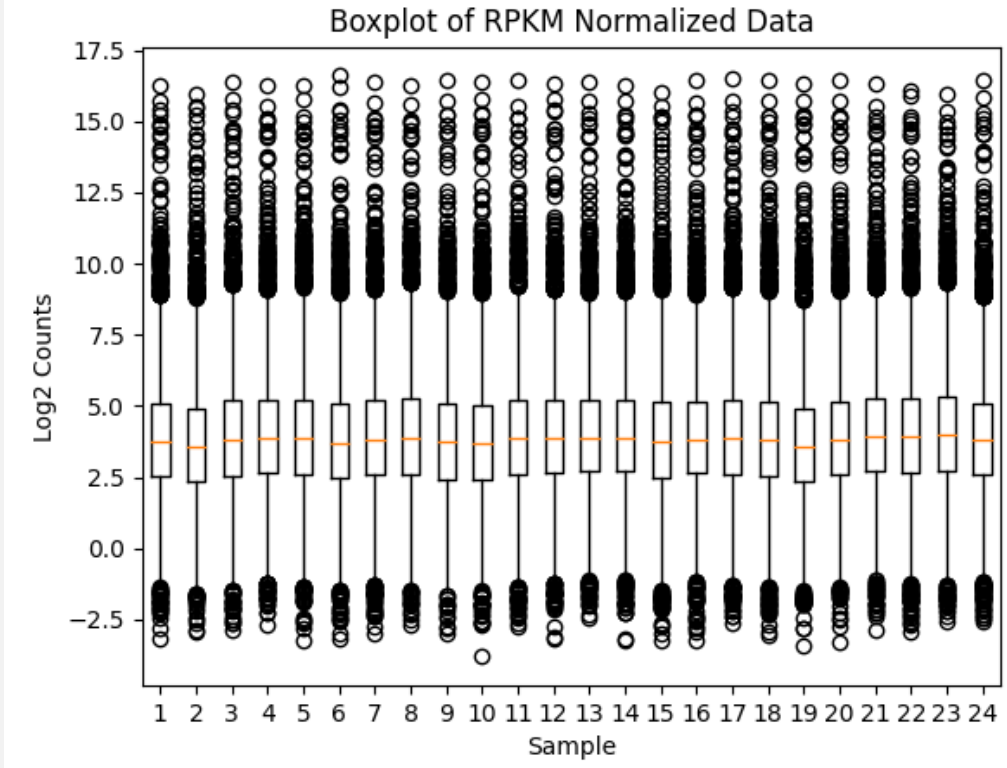
(b) (6 points) Submit the boxplots of the log2 converted count data for each sample for the i) raw counts, ii) RPKM normalized counts, iii) TPM normalized counts, and iv) size factor normalized counts.

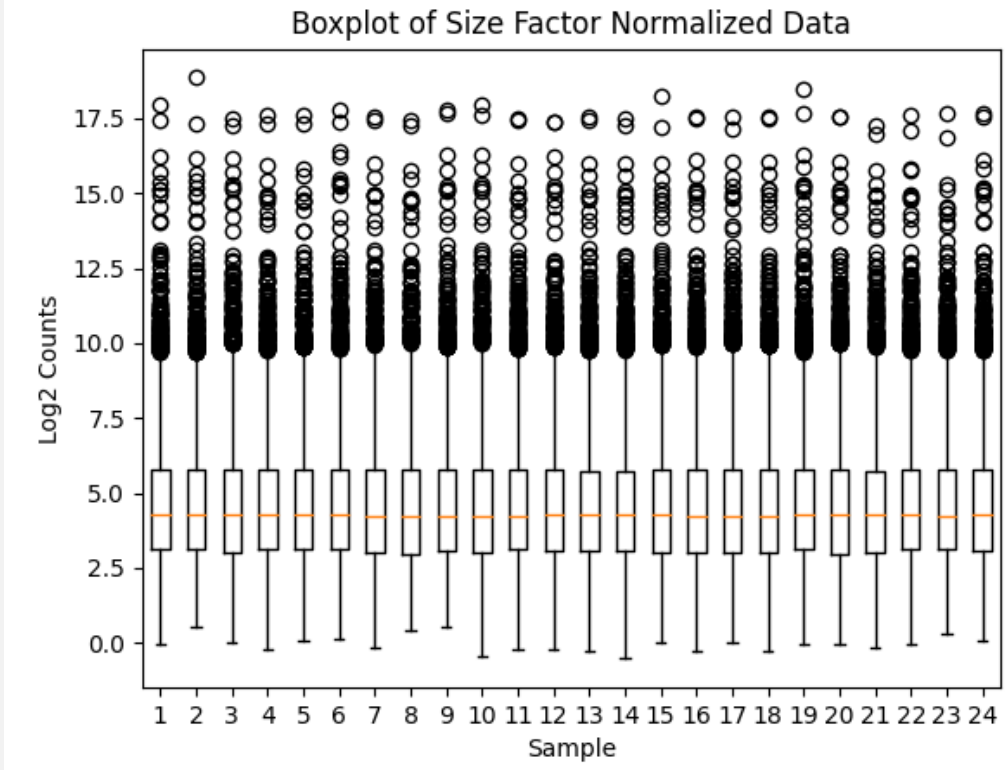
Note: Your plots will not be graded by the autograder. Include the plots in the pdf file.

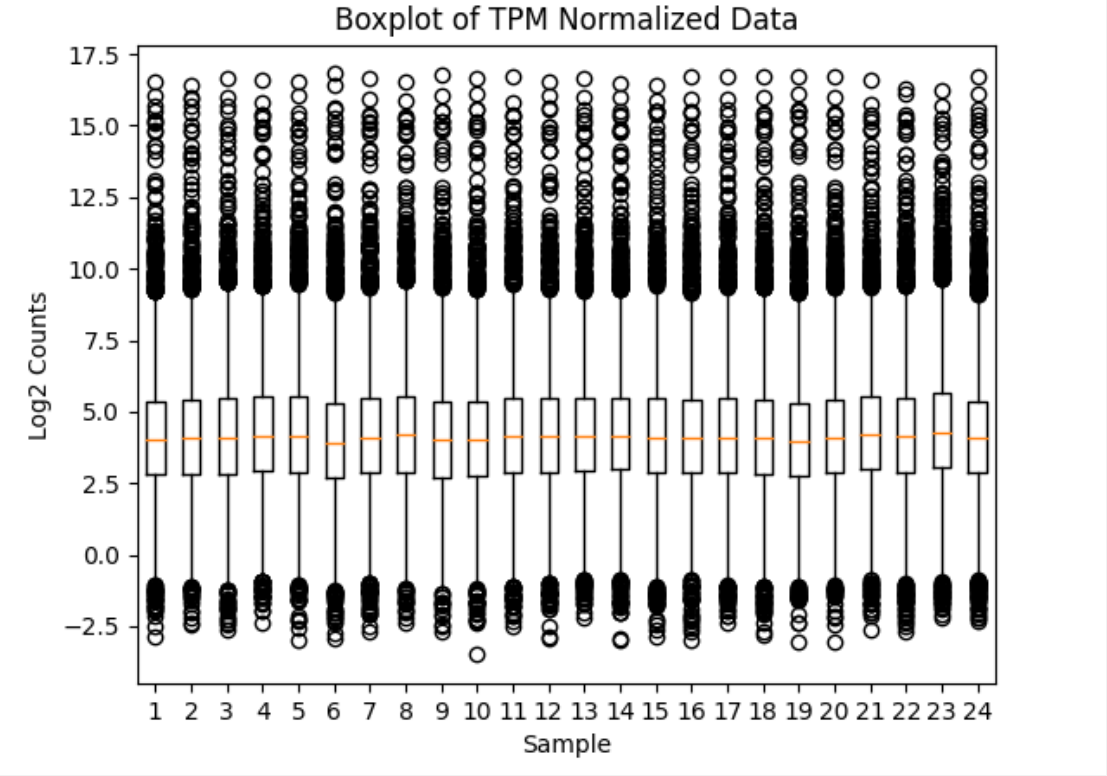
Note2: Save your output from iv) as a new file, *size_factor_normalized_counts.txt*

Solution




Boxplot of RPKM Normalized Data
Log2 Counts
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Sample


Boxplot of Size Factor Normalized Data
Log2 Counts
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Sample



(c) (4 points) Comparing the 4 boxplots, which normalization technique would you choose for this dataset? Explain.

Solution

Since the samples are from the same type, I would use the normalization technique in which the count distribution appears to be the same. The size factor normalized data appears to have consistent medians which suggests a similar count distribution.

2. **[22 points] Differentially expressed genes**

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.

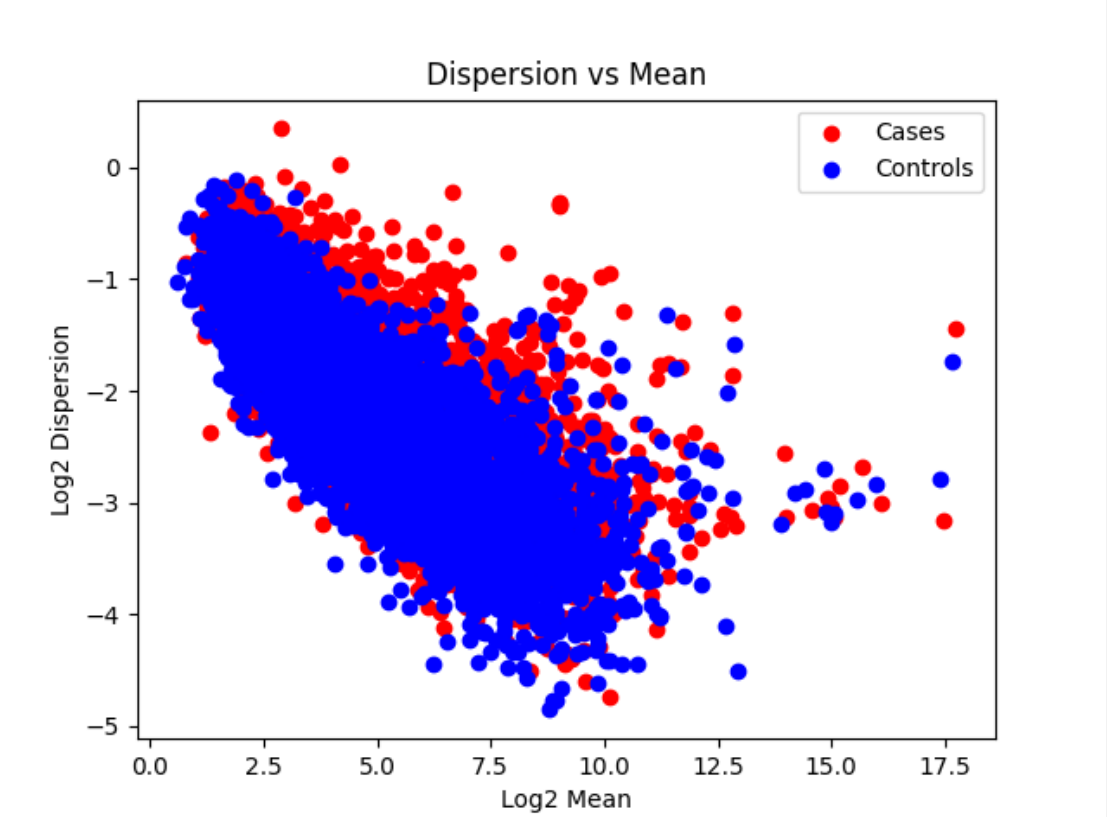
Your task

(a) (3 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?

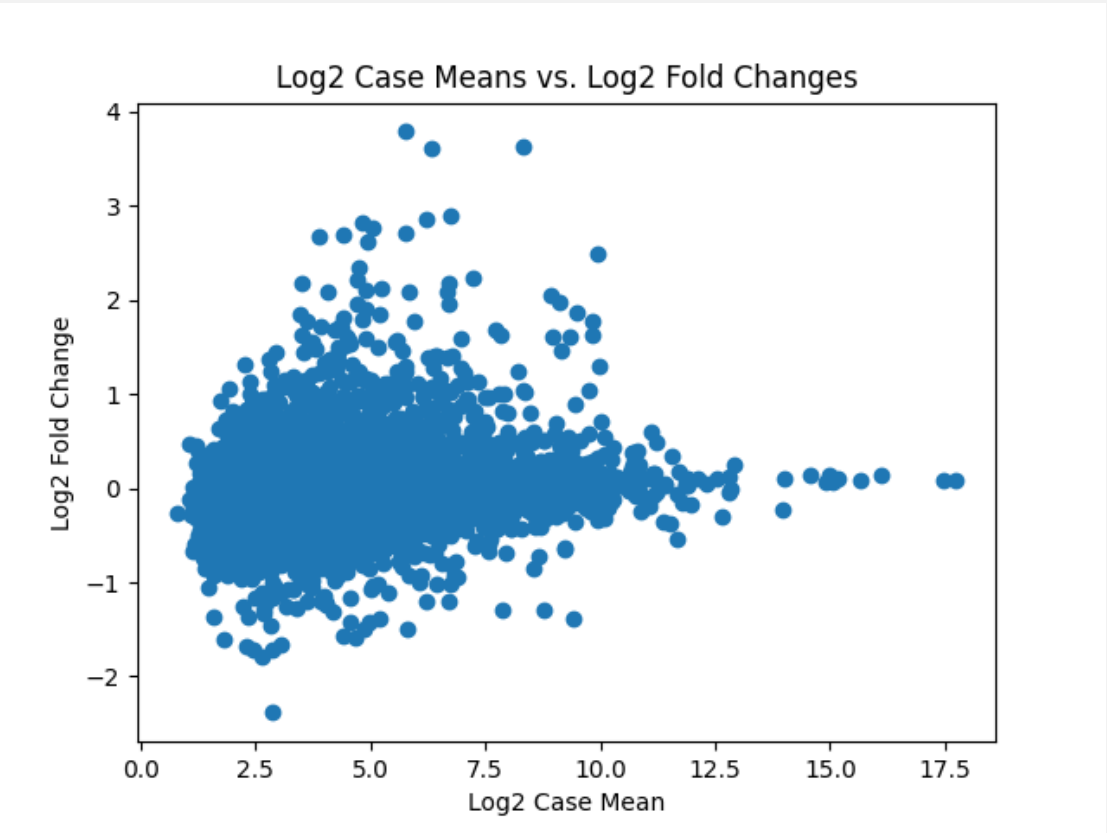
Solution



This plot shows that for most genes the standard deviation of the expression is less than the mean expression for both case and controls, except for one gene in the case samples.

(b) (3 points) Compute the log2 fold change of the genes in the two conditions. Submit a scatterplot of log2 foldchange vs log2 mean count in the two conditions. List the top 10 up and down regulated genes according to the fold change. Are these the most useful genes to investigate? Why or why not?

Solution



Top 10 Up-regulated genes: UNC5B, FKBP5, AL592146.1, TNFAIP8L3, ANPEP, FGD4, MT1X, JPH2, AL049839.2, ZFP36
Top 10 Down-regulated genes: DIO2, C6orf141, RND1, AL807757.2, AL606970.1, IER3-AS1, AC011446.2, AREG, NR5A2, CCL2
These genes might not be useful. Performing a p-test allows us to identify why genes from the data are useful.

(c) (6 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.

(d) (10 points)

In addition to Benjamini-Hochberg (BH) correction, we will discuss two other ways to correct for multiple hypotheses: i) permutation tests and ii) Bonferroni correction.

In permutation tests we run multiple tests by randomizing the labels of the experiments. We use the total number of permutation tests instead of the total number of genes for p-value correction.

Consider a situation in which we have conducted $50,000$ such permutation tests and we have $20,000$ genes. In the table below, we present the results of the permutation tests:

p-value	0.05	0.01	0.001
No. of times we found a gene with a lower p-value	4000	2500	600

i. What is the uncorrected p-value required for a corrected p-value of 0.05 according to the randomization correction method?

ii. Assume we have identified 4000 genes with a p-value < 0.1. What is the false discovery rate (FDR)? **Reminder**: by definition, in a multiple hypothesis test framework, FDR $= \mathbb{E}[V/R]$, where V is the number of null hypotheses that are false rejected and R is the total number of rejected null hypotheses.

Solution

i. A p-value of 0.001 is the uncorrected p-value required for a corrected p-value. If the uncorrected p-value is 0.05, we would expect 5% of the permutation tests that have no significance to have a p-value that is less than or equal to 0.05. 5% of 50,000 permutations is 2,500. Consequently, we would choose 0.001 as a p-value, because 600 of these tests were found to be significant which is much less than 2,500.

ii. We would expect 10% of the 20,000 genes to have no significance and have a p-value less than 0.1. Consequently, 2000 is the number we expect of null hypotheses that are falsely rejected. Since identified 4000 genes with a p-value < 0.1, our false discovery rate is $2000/4000 = 0.5$.

The motivation of Bonferroni correction is,

$$p(specific\, T_i\, passes\, H_0) < \frac{\alpha}{n}$$

$$p(some\, T_i\, passes\, H_0) < \alpha$$

where α is the p-value cut-off and n is the total number of p-values to be corrected.

iii. What is the uncorrected p-value required for a corrected p-value of 0.05 according to the Bonferroni correction?

iv. Suppose we identified 40 genes as significant using the p-value adjusted by Bonferroni correction that you found in part iii). What is the FDR?

Solution

iii. 2.5×10^6

iv. We would expect 0.05 false positives which would mean that the FDR is $0.05/400 = 0.000125$.

3. **[18 points] Learning from scRNA-seq data**
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 are in a Box folder here. 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 20499 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 and SciKit Learn installed on your machine. If you have trouble installing these packages, please come to office hours for help. In addition, please download the dataset from the above Box link. To do this, just open the link and click the Download button in the Box UI. Note that the data is somewhat large ($\sim$ 400 MB), so make sure to do this well before the homework due date.

Your task:

Note: Your code is graded by an autograder. You have been given a skeleton code spectral_clustering.py. You have to complete 4 functions (`get_top_gene_filter`, `reduce_dimensionality_pca`, `plot_transformed_cells`, and `train_and_evaluate_svm_classifier`) as well as part of the `__main__` method of the script. The details about the functions are given later. At the end, the script should be able to run with command line:

```
python classification.py path_to_dataset/train_gene_expression.npz \
path_to_dataset/test_gene_expression.npz path_to_dataset/train_labels.npy \
dataset/test_labels.npy "svm_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) (4 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. **Complete** the `get_top_gene_filter` function 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.

(b) (2 points) As another preprocessing step, we will use Principle 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

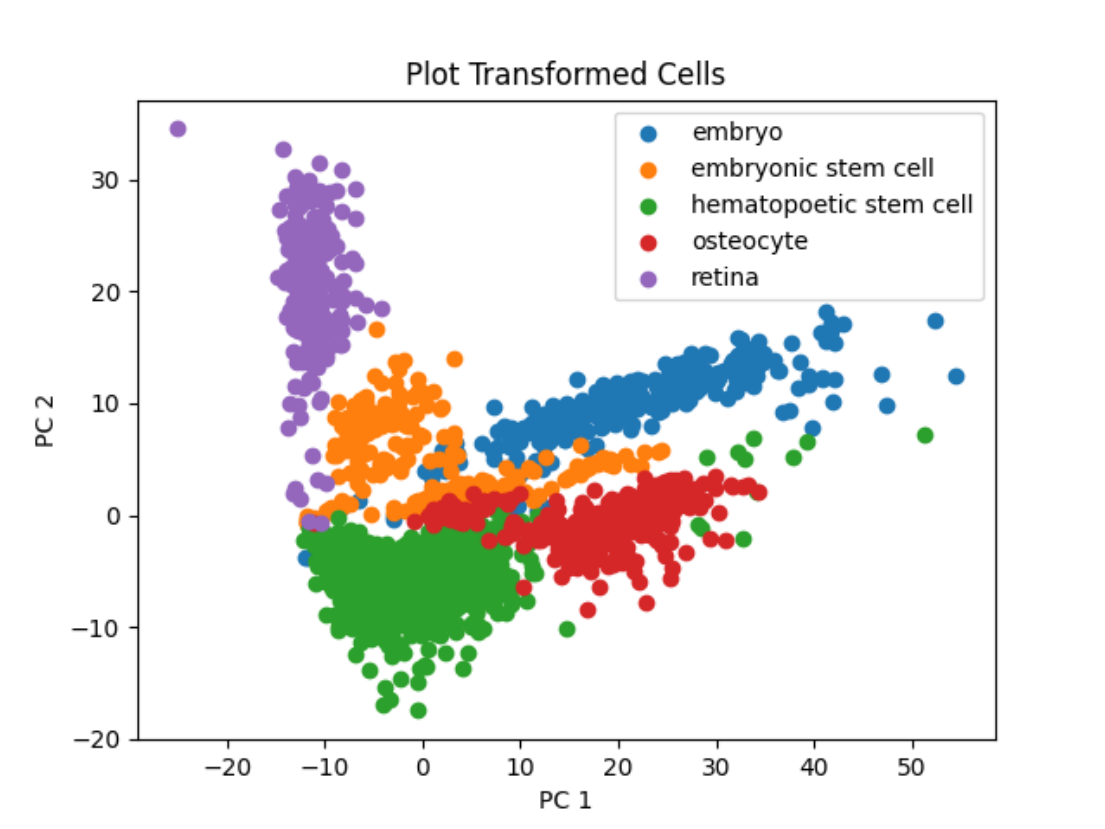
performs PCA on the filtered expression data and returns the transformed training data and test data. Please use the SciKit Learn implementation of PCA (see documentation here).

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

(c) (4 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



The differences between the cell types is defined with some overlap between embryo, embryo stem cells, hematopoetic stem cells, and osteocytes.

(d) (2 points) Next, we will implement a pipeline that uses a Support Vector Machine (SVM) with a soft margin to classify our cells. SVM is a supervised learning algorithm that finds a margin (that is, a "hyperplane" that divides our feature space into multiple regions) to maximize the distinction between different classes (in our case, cell types). In addition, using the "kernel trick," SVM is able to find non-linear decision boundaries when the data is not linearly separable (see the Wikipedia page for more details if you are curious). **Complete** the `train_and_evaluate_svm_classifier` function, which trains a simple SVM pipeline on the

PCA-reduced gene expressiond and returns the trained model as well as the classification accuracy on the test data. Note that this function has already been partially implemented for you. Please use the SciKit Learn implementation of the SVM classifier (see documentation here and here).

(e) (4 points) Now we will combine the above functions to generate a robust pipeline for cell-type classification. **Complete** the (`mode =="svm_pipeline"`) branch within the `__main__` method of the Python script, which combines the gene filtration, PCA dimensionality reduction and SVM classification steps and prints the final score of your classifier. Filtering for the top 2000 genes using the training dataset, using PCA with 20 principal components and using the default SVM hyperparameters, **report** the training and test accuracy of your classifier below.

Solution

train accuracy: 0.9947089947089947
test accuracy: 0.664179104477612