Due date: October 5, 2020.

Type your solution using Word or LaTeX,
submit a PDF document on canvas
Handwritten solutions will not be graded.

In this problem set, you will train to be a biostatistician: you will help a biologist make sense of several datasets in order to identify clusters of genes that are predictive of the clinical presence of a disease. Such statistical tools can be used for medical diagnostics and the prediction of patient outcome.

For each disease, you are given a matrix $\boldsymbol{X}$ composed of vertical columns that represent individuals,

$$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & \cdots & X_{1,j} & \cdots & X_{1,n} \\ \vdots & & \vdots & & \vdots \\ X_{i,1} & \cdots & X_{i,j} & \cdots & X_{i,n} \\ \vdots & & \vdots & & \vdots \\ X_{d,1} & \cdots & X_{d,j} & \cdots & X_{d,n} \end{bmatrix} \tag{1}$$

For each individual j the row i corresponds to a specific gene. The matrix entry $X_{i,j}$ quantifies how much the specific gene i is expressed (present) in patient j. The additional information about the clinical behavior of patient j can be used to learn the association between the disease and each gene i.

The challenge is that the matrix $\boldsymbol{X}$ is almost always tall (several thousands genes) and skinny (less than hundred columns – or subjects). Unless you use dimension reduction, you will not be able to devise reliable statistical machine learning algorithms.

In this problem set, you need to convince your colleague – who is a medical doctor and spent several years collecting the precious data – that the best way to reduce dimensions is to throw data away, by randomly sampling the matrix. Of course your M.D. colleague does not believe you, and you will provide evidence that your method works.

1 Notations

In this problem set, we will consider a dataset formed by a $d \times n$ matrix $\boldsymbol{X}$ composed of n columns of size d. Each column $j = 1, \dots, n$ corresponds to a subject, and each row $i = 1, \dots d$ corresponds to a gene (a feature).

We denote by $\boldsymbol{X}(:,j)$ the column j of the matrix $\boldsymbol{X}$. We can think of subject j (column j) as being characterized by the corresponding genes along that column vector,

$$\boldsymbol{X}(:,j) = \begin{bmatrix} X_{1,j} \\ \vdots \\ X_{d,j} \end{bmatrix}. \tag{2}$$

Because the number of genes d is much larger than the number of subjects n, we will reduce the dimensionality by combining in a judicious way the genes to create a much smaller number, k, of features. The dataset after dimension reduction will still have the same number of subjects (columns), but will have much fewer rows.

We will use linear methods, and define the reduced dataset to be

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X}, \tag{3}$$

where $\boldsymbol{A}$ is a $k \times d$ matrix.

2 The Data

This problem set is using six datasets collected at the Broad Institute. These datasets are well known and are used to benchmark algorithms in genomics and bioinformatics. Each dataset contains the gene expression levels for d genes of n patients. Each matrix $\boldsymbol{X}$ is a $d \times n$ matrix of real numbers. A ZIP archive of the six datasets can be retrieved here: click to download.

You simply need to load the dataset in MATLAB. For instance,

```
>> load data1.mat
>> whos
  Name        Size                Bytes  Class     Attributes

  ans         1x19                   38  char
  data     5597x42              1880592  double
```

The dataset data1 is composed of $n = 42$ microarray (one for each subject) of $d = 5597$ gene expression profiles. We describe below the size of each dataset.

- data1: $d = 5597 \times n = 42$
- data2: $d = 2000 \times n = 62$
- data3: $d = 3571 \times n = 72$
- data4: $d = 4026 \times n = 62$
- data5: $d = 6033 \times n = 52$
- data6: $d = 2308 \times n = 63$

3 How do we compare the original dataset to the reduced dataset?

Throughout this problem set, you will evaluate the performance of several algorithms to reduce the dimensionality of the original dataset $\boldsymbol{X}$ using the following steps.
In the following we assume that

- $\boldsymbol{X}$ is an $d \times n$ matrix, where $n \approx 100$, and $d \gg n$
- $\boldsymbol{Y}$ is a $k \times n$ matrix, where $k \ll d$
- the reduced dataset is obtained by projecting the column of $\boldsymbol{X}$ onto the k-dimensional subspace defined by the rows of A,

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X} \tag{4}$$

Our goal is to verify experimentally that the pairwise distance between points remains (approximately) the same after projection.

3.1 Step 1: scatterplot the pairwise distances between the subjects

The first step is used to check whether the pairwise distances in the reduced dataset are correlated with the corresponding distances in the original space.

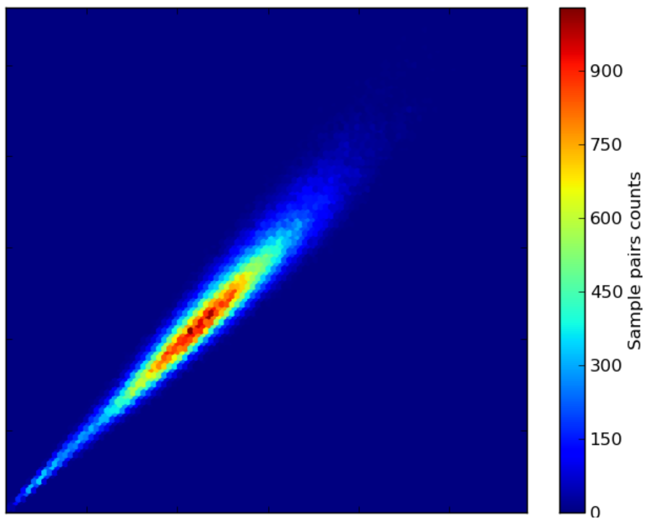

Figure 1: Scatterplot of the pairs $(D_{j,l}, C_{j,l})$, for $1 \leq j < l \leq n$. The intensity of the plot indicates the density of points. The pairwise distances in the reduced space depend linearly on the corresponding distances in the original space (the linear function appears to be $y = x$). We also note that large pairwise distances (toward the upper right) are more distorted than the smaller ones (close to the origin) in the reduced space.

Compute the pairwise distance between all the columns of $\boldsymbol{X}$, and save these distances in the $n \times n$ matrix $\boldsymbol{D}$.

$$D_{j,l} = \|\boldsymbol{X}(:,j) - \boldsymbol{X}(:,l)\| = \sum_{i=1}^{d} (X_{i,j} - X_{i,l})^2, \tag{5}$$

where $1 \leq j < l \leq n$. Compute the pairwise distance between all the columns of $\boldsymbol{Y}$, and save these distances in the $n \times n$ matrix $\boldsymbol{C}$.

$$C_{j,l} = \|\boldsymbol{Y}(:,j) - \boldsymbol{Y}(:,l)\| = \sum_{i=1}^{d} (Y_{i,j} - Y_{i,l})^2, \tag{6}$$

1. Display the scatterplot of the points $(D_{j,l}, C_{j,l})$, for $1 \leq j < l \leq n$.

   See Fig. 1 for an example where the pairwise distances in the reduced space depend linearly on the corresponding distances in the original space. While the linear function appears to be the identity, we note that large pairwise distances are more distorted than the smaller ones (close to the origin) in the reduced space.

3.2 Step 2: histogram of the pairwise distances

The second step is used to check whether the distortion introduced by the dimension reduction varies significantly across the dataset: some pairs of points remain at the same relative distance, while other pairs that were close are now very distant (or vice versa).

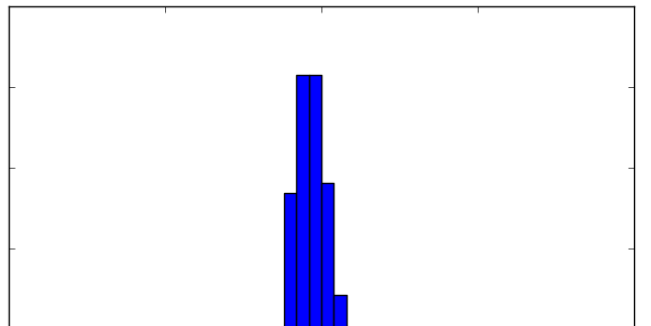

Figure 2: Histogram of the distortion ratio $r_{j,l}$, for $1 \leq j < l \leq n$. The pairwise distances are well preserved for most points in the reduced space: the histogram of the distortion is narrow and centered at 1.

For every pair of subjects $j < l$ compute the distortion ratio

$$r_{j,l} = C_{j,l}/D_{j,l}. \tag{7}$$

2. Display the histogram of the entries $r_{j,l}$, where $1 \leq j < l \leq n$. See Fig. 2 for an example where pairwise distances are well preserved for most points in the reduced space: the histogram of the distortion is narrow and centered at 1.

3.3 Step 3: fraction of the 5 nearest neighbors

The final step is the most relevant step for machine learning applications. While the dimension reduction may distort distances, and therefore deform the geometry of the dataset $\boldsymbol{X}$, this global deformation may have little to no effect on the actual machine learning task at hand.

Indeed, in most cases the only relevant information is whether the local distances are preserved. Specifically, one may be only interested in the labels of the nearest neighbors of each subject j. In other words, if the gene expression are useful to characterize a disease, we expect that if patient j suffers from that disease, other subjects with the same disease will cluster in the same neighborhood part of the space.

It is therefore important to test that the labels (column indices) of the neighbors of each subject remain the same after dimension reduction.

3. For each dataset $\boldsymbol{X}$,

- Compute $\boldsymbol{Y}$
- For every column $\boldsymbol{X}(:, j)$, compute the five nearest (according to the Euclidean norm) columns, $\boldsymbol{X}(:, j_1), \dots, \boldsymbol{X}(:, j_5)$.

  This can be easily performed using the following steps.

  (a) sort all the entries $D_{j,l} = \|\boldsymbol{X}(:, j) - \boldsymbol{X}(:, l)\|, \quad l = 1, \dots, n$
  (b) keep the five smallest entries.
  (c) save the five integers $j_1, \dots, j_5$ such that

  $$0 = D_{j,j} < D_{j,j_1} \leq \ldots \leq D_{j,j_5}. \tag{8}$$

- For every column $\boldsymbol{Y}(:, j)$, compute the five nearest columns, as explained above.
  You will get five integers $l_1, \dots, l_5$ such that

  $$0 = C_{j,j} < C_{j,l_1} \leq \ldots \leq C_{j,l_5}. \tag{9}$$

- compute the number of indices `nn(j)` in $\{l_1, \dots, l_5\}$ that are in the set $\left\{j_1, \dots, j_5\right\}$.
- divide `nn(j)` by 5. This is the fraction of nearest neighbors that are correctly identified after dimension reduction.
- In a separate figure for each dataset, display the four histograms (for $k = 25, 100, 225, 400$) of the set of values $\left\{nn(j), j = 1, \dots, n\right\}$.

4 The Data Science: Dimension Reduction (Random and Slow)

We have seen in class that we can use a $k \times d$ matrix $\boldsymbol{A}$ to project the columns $\boldsymbol{X}(:, j)$ onto k dimension. The entries of the matrix are independent realizations of zero-mean unit variance Gaussian. Algorithm 1 describes the pseudo code of the Johnson-Lindenstrauss transform.

Algorithm 1 Johnson-Lindenstrauss transform

```
 1: procedure JL($X, Y, k$)          // Apply a $k \times d$ random Gaussian matrix to the dataset
                                      // Input: $X$, $k$; output: $Y$ the reduced datatset
 2:     for $i \leftarrow 1, k$ do          // build the matrix $A$: vectorize these loops!
 3:         for $j \leftarrow 1, d$ do
 4:             $A(i, j) \leftarrow \mathcal{N}(0, 1)$          // zero-mean unit variance Gaussian
 5:         end for
 6:     end for

 7:     for $j \leftarrow 1, n$ do          // Apply $A$ to all columns of $X$
 8:         $Y(:, j) \leftarrow AX(:, j)$
 9:     end for

10:     $Y \leftarrow \frac{1}{\sqrt{k}} Y$          // Return $Y$ the reduced datatset
11: end procedure
```

Assignment [200 = 6 x 3 x 10 + 20]

Implement the function `JL` described in algorithm 1. Your function will return the amount of CPU time necessary to compute $\boldsymbol{Y}$ for each dataset.

1. For the six datasets described in Section 2, evaluate the performance of `JL` using the three criteria described in Section 3. You will use $k = 25, 100, 225, 400$.

   For Step 2, you will compare the theoretical guarantees provided by the Johnson Lindenstrauss theorem (accuracy ε, and probability of success $1 - \eta$) with your experimental findings.

   For Step 3, you will plot the fraction of nearest neighbors as a function of the dimension k. You will use the same figure for the six datasets.

2. Plot the CPU time as a function of k for each dataset using the same figure.

Algorithm 2 Fast Johnson-Lindenstrauss transform (Achlioptas, 2003)

```
 1: procedure FJL($X, Y, k$)            // Apply a sparse $k \times d$ random matrix to the dataset
                                        // Input: $X$, $k$; output: $Y$ the reduced datatset
 2:     for $i \leftarrow 1, k$ do      // build the matrix $A$ using a few coin flips
 3:         for $j \leftarrow 1, d$ do
 4:             $A(i,j) \leftarrow \begin{cases} 1 & \text{with probability} 1/6 \\ -1 & \text{with probability} 1/6 \\ 0 & \text{with probability} 2/3 \end{cases}$     // roll a die!
 5:         end for
 6:     end for

 7:     for $j \leftarrow 1, n$ do      // Apply $A$ to all columns of $X$
 8:         $Y(:,j) \leftarrow AX(:,j)$
 9:     end for

10:     $Y \leftarrow \sqrt{\frac{3}{k}} Y$     // Return $Y$ the reduced datatset

11: end procedure
```

5 The Data Science: Dimension Reduction (Random and Fast)

There are several problems with the function JL. First, generating random Gaussian number is a rather slow process. Second, the matrix A is dense: all entries are small but not equal to zero. Therefore the matrix vector multiplication of algorithm 1 costs about $k \times n$ flops. For our small datasets, this is already about of order 100,000 operations.

Fortunately there exists an algorithm that can solve both problems: most entries in A are zero, and the entries are constant. You will implement this algorithm and compare it to the traditional Johnson-Lindenstrauss transform. Algorithm 2 describes the pseudo code of the fast Johnson-Lindenstrauss transform invented by Achlioptas.

The first observation is that two third of the entries in A are zero, and thus require no work (provided sparse structures are used). The second observation is that the random Gaussian numbers are replaced by a coin flipping (± 1).

Assignment [200 = 6 x 3 x 10 + 20]

Implement the function `FJL` described in algorithm 2. Your function will return the amount of CPU time necessary to compute $\boldsymbol{Y}$ for each dataset.

In the following questions you will evaluate the performance of algorithm 2. In order to get all the points, your evaluation should focus on the comparison between the functions `JL` and `FJL`, described in algorithms 1 and 2 respectively.

3. For the six datasets described in Section 2, evaluate the performance of `JL` using the three criteria described in Section 3. You will use $k = 25, 100, 225, 400$.

   For Step 3, you will plot the fraction of nearest neighbors as a function of the dimension k. You will use the same figure for the six datasets.

4. Plot the CPU time as a function of k for each dataset using the same figure.