CSE 547: Machine Learning for Big Data **Winter 2023**

Problem Set 2

Please read the homework submission policies.

Assignment Submission All students should submit their assignments electronically via GradeScope. No handwritten work will be accepted. Math formulas **must** be typeset using LaTeX or other word processing software that supports mathematical symbols (E.g. Google Docs, Microsoft Word). Simply sign up on Gradescope and use the course code V84RPB. Please use your UW NetID if possible.

For the non-coding component of the homework, you should upload a PDF rather than submitting as images. We will use Gradescope for the submission of code as well. Please make sure to tag each part correctly on Gradescope so it is easier for us to grade. There will be a small point deduction for each mistagged page and for each question that includes code. Put all the code for a single question into a single file and upload it. Only files in text format (e.g. .txt, .py, .java) will be accepted. **There will be no credit for coding questions without submitted code on Gradescope, or for submitting it after the deadline**, so please remember to submit your code.

Coding You may use any programming languages and standard libraries, such as NumPy and PySpark, but you may not use specialized packages and, in particular, machine learning libraries (e.g. sklearn, TensorFlow), unless stated otherwise. Ask on the discussion board whether specific libraries are allowed if you are unsure.

Late Day Policy All students will be given two no-questions-asked late periods, but only one late period can be used per homework and cannot be used for project deliverables. A late-period lasts 48 hours from the original deadline (so if an assignment is due on Thursday at 11:59 pm, the late period goes to the Saturday at 11:59pm Pacific Time).

Academic Integrity We take academic integrity extremely seriously. We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions and the code independently. In addition, each student should write down the set of people whom they interacted with.

Discussion Group (People with whom you discussed ideas used in your answers):

On-line or hardcopy documents used as part of your answers:

I acknowledge and accept the Academic Integrity clause.

*(Signed)*____

1 Principal Component Analysis and Reconstruction (40 points)

Principal Component Analysis (PCA) is a dimensionality reduction algorithm that represents the data matrix as a set of principal components that are orthogonal to each other. Each of these principal components captures a source of variation in the original data matrix such that we can reduce the dimensions of the data while preserving as much information as possible. Principal components are defined as the eigenvectors of the covariance matrix of the data; thus, we will be implementing PCA using eigendecomposition.

This question will help you understand the basics of eigendecomposition. By examining the dimensionality reduction and reconstruction of images, you will think and learn more about informative representations of data.

Let's do PCA and reconstruct pictures of faces in the PCA basis. As we will be making many visualizations of images, plot these images in a reasonable way (e.g. make the images smaller).

(a) Matrix Algebra Review [7 pts]

The trace of a square matrix $\boldsymbol{M}$, denoted, by $Tr(\boldsymbol{M})$ is defined as the sum of the diagonal entries of $\boldsymbol{M}$.

1. [**5 pts**] Now we prove a few claims that will be helpful in the next problem. Define $\boldsymbol{\Sigma} := \frac{1}{n}\mathbf{X}^{\mathrm{T}}\mathbf{X}$, where $\mathbf{X}$ is the $n \times d$ data matrix. Let $\mathbf{x}_i$ be the i-th row of $\mathbf{X}$ (so $\mathbf{x}_i$ is a d-dimensional vector). Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ be the eigenvalues of $\boldsymbol{\Sigma}$. Show the following two properties:

$$Tr(\boldsymbol{\Sigma}) = \sum_{i=1}^{d} \lambda_i \tag{1}$$

$$Tr(\boldsymbol{\Sigma}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i\|_2^2 \tag{2}$$

   Hint 1: Symmetric matrices are orthogonally diagonalizable. That is, a symmetric matrix $\boldsymbol{A}$ can always be written as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$, where $\boldsymbol{D}$ is a diagonal matrix with the eigenvalues of $\boldsymbol{A}$ along the diagonal and $\boldsymbol{U}$ is an orthogonal matrix, whose columns are the eigenvectors of $\boldsymbol{A}$.

   Hint 2: $Tr(\boldsymbol{A}\boldsymbol{B}^{\mathrm{T}}) = Tr(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{A})$ for two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of size $n \times d$.

2. [**2 pts**] Now we have a $d \times k$ projection matrix $\mathbf{P}$ with $\mathbf{P}^T\mathbf{P} = \mathbf{I}_k$. We have the linear projection from $\mathbb{R}^{n\times d}$ to $\mathbb{R}^{n\times k}$ $\mathbf{X} \mapsto \mathbf{XP}$. We have the following result:

$$\underset{\mathbf{P}\in\mathbb{R}^{d\times k}, \mathbf{P}^T\mathbf{P}=\mathbf{I}}{\arg\max} \operatorname{Tr}\left(\frac{1}{n}\mathbf{P}^T\mathbf{X}^T\mathbf{X}\mathbf{P}\right) = \mathbf{P}^*$$

where each column of $\mathbf{P}^*$ is the corresponding eigenvector u_i for λ_i with $i = 1, ..., k$. Please prove $Tr(\frac{1}{n}\mathbf{P}^{*T}\mathbf{X}^T\mathbf{X}\mathbf{P}^*) = \sum_{i=1}^{k} \lambda_i$.

With the result obtained in part 1, we have $1 - \frac{\lambda_1+\cdots+\lambda_k}{\lambda_1+\cdots+\lambda_d} = 1 - \frac{\sum_{i=1}^{n}\left\|\mathbf{P}^{*T}\mathbf{x}_i\right\|^2}{\sum_{i=1}^{n}\|\mathbf{x}_i\|^2}$. This metric is called the PCA fractional reconstruction error of using the top k out of d directions.

(b) PCA [10 pts]

For this question we will use the dataset `faces.csv` from `pca/data` (adapted from the Extended Yale Face Database B), which is composed of facial pictures of 38 individuals under 64 different lighting conditions such as lit from the top, front, and side (some lighting conditions are missing for some people). In total, there are 2414 images, where each image is 96 pixels high and 84 pixels wide (for a total of 8064 pixels per image). Each row in the dataset is a single image flattened into a vector in a column-major order. The images are in grayscale, so each pixel is represented by a single real number between 0 (black) and 1 (white).

Define $\mathbf{\Sigma}$, a 8064×8064 matrix, as follows:

$$\mathbf{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}}$$

where the $\boldsymbol{x}_i$'s are points in our dataset as **column** vectors and $n = 2414$ is the number of points in the dataset. Now compute the top 50 PCA dimensions; these are the 50 dimensions which best reconstruct the data.

We will be implementing PCA using eigendecomposition. Thus, you may use library functions for obtaining the eigenvalues and eigenvectors of a matrix (e.g. `numpy.linalg.eig` in Python and `eig` or `eigs` in MATLAB). You should **not** use functions which directly compute the principal components of a matrix. Please ask on Ed regarding other (non-basic) linear algebra functions you may be interested in using.

1. [**3 pts**] What are the eigenvalues λ_1, λ_2, λ_{10}, λ_{30}, and λ_{50}? Also, what is the sum of eigenvalues $\sum_{i=1}^{d} \lambda_i$? *(Hint: use the answer from the previous question).*

2. [**5 pts**] From part (a) we know that fractional reconstruction error of using the top k out of d directions is $1 - \frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$ (this means that when $k = d$, our reconstruction error is 0, which basically means that we use all the principal components and simply reconstruct the original matrix). Plot this fractional reconstruction error for each of the first 50 values of k (i.e. k from 1 to 50). So the X-axis is k and the Y-axis is the fractional reconstruction error. Make sure your plots are legible at reasonable zoom and the axes are labeled in your write-up.

3. [**2 pt**] Using 1-3 sentences, explain what the first eigenvalue captures, and why you think λ_1 is much larger than the other eigenvalues.

(c) Visualization of the Eigen-Directions [10 pts]

Now let us get a sense of the what the top PCA directions are capturing (recall these are the directions which capture the most variance).

1. [**5 pts**] Display the first 10 eigenvectors as images. Label the images to denote which eigenvector each image denotes.

   Hint 1: If the images appear like random lines, try reshaping differently and then transposing.

   Hint 2: The eigenvectors you obtain are normalized to have length 1 in the L_2 norm, thus their entries are extremely small. To avoid getting all-black images, make sure to re-normalize the image. in Python, `matplotlib.pyplot.imshow` does this by default. If you are using `imshow` in MATLAB, you should pass an extra empty vector as an argument, e.g. "`imshow(im, []);`".

2. [**5 pt**] Provide a brief interpretation (4-6 sentences) of what you think each eigenvector captures (*Hint:* Look at each image and think about what features of the image the eigenvector is capturing).

(d) Visualization and Reconstruction [13 pts]

1. [**8 pt**] We will now observe the reconstruction using PCA on a sample of images composed of the following images:

   (a) image 1 (row 0).

   (b) image 24 (row 23).

   (c) image 65 (row 64).

   (d) image 68 (row 67).

   (e) image 257 (row 256).

   In the previous parts, we used eigendecomposition to extract the top k eigenvectors of Σ. Now, we will examine reconstruction using PCA.

   For each of these images, plot the original image and plot the projection of the image onto the top k eigenvectors of Σ for $k = 1, 2, 5, 10, 50$. In particular, if U is the $d \times k$ matrix of the top k eigenvectors, the reconstruction matrix will be UU^{T}.

   Specifically, you should obtain a 5×6 table where in each row corresponds to a single image, the first cell in every row is the original image and the following cells are the reconstructions of the image with the 5 different values of k.

   Hint: In this problem, we are observing (partial) combinations of images that already have a meaningful scale. Therefore, we want to keep the scale of the results and not re-normalize the images (in contrast to part (c)). If you are using `matplotlib.pyplot.imshow` in Python, you should pass the additional arguments vmin=0, vmax=1, e.g. `imshow(im, vmin=0, vmax=1)`. If you are using `imshow` in MATLAB, the default is not re-normalizing, so you should **not** pass additional arguments, e.g. simply use "`imshow(im);`".

2. [**5 pt**] Provide a brief interpretation (4-6 sentences), in terms of your perceptions of the quality of these reconstructions. Explain the results in light of your answers for part (c). Discuss specific examples regarding pairs of images and eigenvector set (i.e. how does including a set of eigenvectors affect specific images?)

What to submit:

(i) Proofs for the claims in 1(a).

(ii) The values of λ_1, λ_2, λ_{10}, $\lambda_{30}, \lambda_{50}$ and $\sum_i \lambda_i$, a plot of the reconstruction error vs. k and an explanation for 1(b).

(iii) Plots of the first 10 eigenvectors as images and their interpretation [1(c)].

(iv) Table of original and reconstructed images and their interpretation [1(d)].

(v) Upload the code to Gradescope.

2 k-means on Spark (30 points)

Note: This problem requires substantial computing time. Don't start it at the last minute. Also, you should **not** use the Spark MLlib clustering library for this problem.

This problem will help you understand the nitty gritty details of implementing clustering algorithms on Spark. In addition, this problem will also help you understand the impact of using various distance metrics and initialization strategies in practice. Let us say we have a set $\mathcal{X}$ of n data points in the d-dimensional space $\mathbb{R}^d$. Given the number of clusters k and the set of k centroids $\mathcal{C}$, we now proceed to define various distance metrics and the corresponding cost functions that they minimize.

Given two points $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$, such that $\boldsymbol{a} = [a_1, a_2 \cdots a_d]$ and $\boldsymbol{b} = [b_1, b_2 \cdots b_d]$ the *Euclidean distance* or L^2 *distance* between $\boldsymbol{a}$ and $\boldsymbol{b}$ is defined as:

$$\|\boldsymbol{a} - \boldsymbol{b}\|_2 = \sqrt{\sum_{i=1}^{d} |a_i - b_i|^2} \tag{3}$$

The *Manhattan distance* or L^1 *distance* between a and b is defined as:

$$\|\boldsymbol{a} - \boldsymbol{b}\|_1 = \sum_{i=1}^{d} |a_i - b_i| \tag{4}$$

The k-**means algorithm** aims to partition $\mathcal{X}$ to k clusters by defining a set $\mathcal{C} = \{\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(k)}\}$ of k centroids and partitioning $\mathcal{X}$ to k clusters $\mathcal{P} = \{P_1, \ldots, P_k\}$, so to minimize the sum of **squared** L^2-distances between every point and the centroid associated with its cluster. Formally, k-means tries to solve the following minimization problem:

$$\min_{\mathcal{C},\mathcal{P}} \Phi(\mathcal{C}, \mathcal{P}) = \min_{\mathcal{C},\mathcal{P}} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in P_i} \|\boldsymbol{x} - \boldsymbol{c}^{(i)}\|_2^2$$

where $\Phi(\mathcal{C}, \mathcal{P}) = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in P_i} \|\boldsymbol{x} - \boldsymbol{c}^{(i)}\|_2^2$ is the "cost" associated with the set of centroids $\mathcal{C}$ and the partition $\mathcal{P}$.

For a fixed set of centroids $\mathcal{C}$, the cost function Φ is minimized by assigning every point to the centroid closest to it (in L^2). Thus, the cost $\phi(\mathcal{C})$ associated with $\mathcal{C}$ is:

$$\phi\left(\mathcal{C}\right) = \min_{\mathcal{P}} \Phi(\mathcal{C}, \mathcal{P}) = \sum_{x \in \mathcal{X}} \min_{\boldsymbol{c} \in \mathcal{C}} \|\boldsymbol{x} - \boldsymbol{c}\|_2^2 \tag{5}$$

For a fixed partition $\mathcal{P}$, the cost function Φ is minimized by setting the centroid of every cluster to be the mean of all the points in that cluster. Thus, the algorithm operates as follows: Initially, k centroids are initialized. Thereafter, alternately, given the set of centroids, each point is assigned to the cluster associated with the nearest centroid; and the centroids are recomputed based on the assignments of points to clusters. The above process is executed for several iterations, as is detailed in Algorithm 1.

Algorithm 1 k-means Algorithm

```
procedure k-MEANS
    Select $k$ points as initial centroids of the $k$ clusters.
    for iteration := 1 to MAX_ITER do
        for each point $x$ in the dataset do
            Cluster of $x \leftarrow$ the cluster with the closest centroid to $x$
        end for
        for each cluster $P$ do
            Centroid of $P \leftarrow$ the mean of all the data points assigned to $P$
        end for
        Calculate the cost for this iteration.
    end for
end procedure
```

Instead of minimizing the cost function Φ, which is defined using the squared L^2 distance, we can minimize a cost function defined using the L^1 distance, namely:

$$\Psi(\mathcal{C}, \mathcal{P}) = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in P_i} \|\boldsymbol{x} - \boldsymbol{c}^{(i)}\|_1$$

Note that in this case the distance is **not** squared.

Still, for a fixed set of centroids $\mathcal{C}$, the cost function Ψ is minimized by assigning every point to the centroid closest to it (but this time in L^1). Thus the cost $\psi(\mathcal{C})$ associated with $\mathcal{C}$ is:

$$\psi\left(\mathcal{C}\right) = \min_{\mathcal{P}} \Psi(\mathcal{C}, \mathcal{P}) = \sum_{\boldsymbol{x} \in \mathcal{X}} \min_{\boldsymbol{c} \in \mathcal{C}} \|\boldsymbol{x} - \boldsymbol{c}\|_1 \tag{6}$$

In contrast to Φ, for a fixed partition $\mathcal{P}$, the cost function Ψ is minimized setting the centroid of every cluster to be the **median** in every dimension of the points in that cluster. Thus, we obtain a variation on k-means, which is known as **k-medians**.

Please use the dataset from `kmeans/data` within the bundle for this problem.

The folder has 3 files:

1. `data.txt` contains the dataset which has 4601 rows and 58 columns. Each row is a document represented as a 58 dimensional vector of features. Each component in the vector represents the importance of a word in the document.
2. `c1.txt` contains k initial cluster centroids. These centroids were chosen by selecting $k = 10$ random points from the input data.
3. `c2.txt` contains initial cluster centroids which are as far apart as possible. (You can do this by choosing the first centroid $\boldsymbol{c}^{(1)}$ randomly, and then finding the point $\boldsymbol{c}^{(2)}$ that is farthest from $\boldsymbol{c}^{(1)}$, then selecting $\boldsymbol{c}^{(3)}$ which is farthest from $\boldsymbol{c}^{(1)}$ and $\boldsymbol{c}^{(2)}$, and so on).

Set the number of iterations (`MAX_ITER`) to 20 and the number of clusters k to 10 for all the experiments carried out in this question. Your driver program should ensure that the correct amount of iterations are run.

Note: Please ensure that your solution works correctly with duplicate data points. If not, your computation might be slightly off.

(a) Exploring initialization strategies with Euclidean distance [15 pts]

Implement k-means using Spark and compute the cost function $\phi(i)$ (Equation 5) after every iteration $i = 0, 1, \ldots$[1]. This means that for your 0^{th} iteration, you'll be computing the cost function using the initial centroids located in one of the two text files. Run the k-means on `data.txt` using `c1.txt` and `c2.txt`.

Hint: Note that you do not need to write a separate Spark job to compute $\phi(i)$. You should be able to calculate costs while partitioning points into clusters.

Hint 2: The cost after 5 iterations starting from the centroids in `c1.txt` is approximately 460,538,000.

[1] We are overloading the notation here. The more precise way to express $\phi(i)$ would be $\phi(\mathcal{C}_i)$, where $\mathcal{C}_i$ is the set of centroids after iteration i.

1. **[8 pts]** Generate a graph where you plot the cost function $\phi(i)$ as a function of the number of the iteration i=0,. . . ,20 for `c1.txt` and also for `c2.txt`. Label your axes and add a legend for the subgraphs for `c1.txt` and `c2.txt`.

2. **[7 pts]** By how many percent does the cost change over the first 10 iterations of k-means when initializing using `c1.txt` and when using `c2.txt`? (e.g. if the cost before the first iteration was 10 and after the 10^{th} iteration it is 7, then it decreased by 30%).

   Explain which initialization method is better in terms of the cost $\phi(i)$, and your intuition as to why.

   Hint: The cost for c1 decreases by ∼26% after 10 iterations.

(b) Exploring initialization strategies with Manhattan distance [15 pts]

Implement k-medians using Spark and compute the cost function $\psi(i)$ (Equation 6) after every iteration $i = 0, 1, \ldots$ [2]. This means that, for your 0^{th} iteration, you'll be computing the cost function using the initial centroids located in one of the two text files. Run the k-medians on `data.txt` using `c1.txt` and `c2.txt`.

Hint: This problem can be solved in a similar manner to that of part (a).

Hint 2: The cost after 5 iterations starting from the centroids in ***`c1.txt`*** *is approximately* $412,012$.

1. **[8 pts]** Generate a graph where you plot the cost function $\psi(i)$ as a function of the number of the iteration i=0,. . . ,20 for `c1.txt` and also for `c2.txt`. Label your axes and add a legend for the subgraphs for `c1.txt` and `c2.txt`.

2. **[7 pts]** By how many percent does the cost change over the first 10 iterations of k-medians when initializing using `c1.txt` and when using `c2.txt`?

   Explain which initialization method is better in terms of cost $\psi(i)$, and your intuition as to why.

   Hint: The cost for c1 decreases by ∼26% after 10 iterations.

What to submit:

(i) Upload the code for 2(a) and 2(b) to Gradescope (Your solution may be in either one or two files).

(ii) A plot of cost vs. iteration for two initialization strategies for 2(a).

(iii) Percentage of change and your explanation for 2(a).

(iv) A plot of cost vs. iteration for two initialization strategies for 2(b).

(v) Percentage of change values and your explanation for 2(b).

[2] Same as footnote 1

3 Implicit Feedback Recommendation System (30 points)

In many real-world applications, it's expensive to collect explicit rating data. However, it's cheap to collect implicit feedback data such as clicks, page views, purchases and media streams at a large and fast scale.

In this problem, you will learn more about implicit feedback recommendation systems by focusing on Alternating Least Square (ALS) algorithm. Deriving the formula yourself by following all the steps of the question and trying the algorithm on a real dataset will help you understand how implicit feedback data can be used to learn more about users and items.

Let $\mathcal{U}$ be a set of m users, and $\mathcal{I}$ be a set of n items. For every user $u \in \mathcal{U}$ and item $i \in \mathcal{I}$, let's define such observable implicit feedback as r_{ui}:

$$r_{ui} = \text{The number of times user } u \text{ interacted with item } i$$

Note that r_{ui} could be allowed to have non-integer values; e.g. $r_{ui} = 0.5$ may indicate that user u watched half of movie i. We cannot observe the true *preference* p_{ui} of user u for item i. For simplicity, we assume p_{ui} can only take the values 1 (like) or 0 (dislike).

Following the latent factor model in lecture, we assume p_{ui} is the dot product of a user vector $\boldsymbol{x}_u \in \mathbb{R}^f$ and an item vector $\boldsymbol{y}_i \in \mathbb{R}^f$ for each user u and item i:

$$p_{ui} \approx \boldsymbol{x}_u^T \boldsymbol{y}_i$$

If user u has interacted with item i, we have reason to believe that $p_{ui} = 1$ with some confidence. The more the user interacts with that item, the more confident we are that $p_{ui} = 1$. To capture this idea, we try to minimize the following heuristic cost function over possible assignments to the user matrix $\boldsymbol{X} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_m]^T \in \mathbb{R}^{m \times f}$ and to the item matrix $\boldsymbol{Y} = [\boldsymbol{y}_1 \cdots \boldsymbol{y}_n]^T \in \mathbb{R}^{n \times f}$:

$$C_{\text{implicit}}(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{u,i \in \mathcal{U} \times \mathcal{I}} c_{ui} \left(p_{ui} - \boldsymbol{x}_u^T \boldsymbol{y}_i \right)^2 + \lambda \left(\sum_u \|\boldsymbol{x}_u\|^2 + \sum_i \|\boldsymbol{y}_i\|^2 \right), \tag{7}$$

where

$$p_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 0 \\ 0 & \text{if } r_{ui} = 0 \end{cases},$$

$c_{ui} = 1 + \alpha r_{ui}$ is our confidence in p_{ui}. Empirical evidence suggests setting hyperparameter α to the sparsity ratio $= \frac{\#\text{nonzero } r_{ui}}{\#\text{zero } r_{ui}}$.

We apply an algorithm known as Alternating Least Square (ALS) to minimize C_{implicit}. The basic idea of ALS is: first hold the user vectors fixed and solve for the minimum in the item variables, then hold the item vectors fixed and solve for the minimum in the user variables, and repeat until convergence.

Algorithm 2 ALS

```
procedure ALS
    for $iteration \leftarrow 1$ to MAX_ITER do
        for each item $i$ do                                          ▷ Step 1: update item matrix $Y$
            $\boldsymbol{p}_i \leftarrow$ all the preferences for item $i$          ▷ $\boldsymbol{p}_i \in \mathbb{R}^m$
            $\boldsymbol{C}_i \leftarrow$ the diagonal $m \times m$ matrix, where $[\boldsymbol{C}_i]_{uu} = c_{ui}$.
            $\boldsymbol{y}_i \leftarrow \left(\boldsymbol{X}^T\boldsymbol{C}_i\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{C}_i\boldsymbol{p}_i$
        end for
        for each user $u$ do                                          ▷ Step 2: update user matrix $X$
            $\boldsymbol{p}_u \leftarrow$ all the preferences of user $u$           ▷ $\boldsymbol{p}_u \in \mathbb{R}^n$.
            $\boldsymbol{C}_u \leftarrow$ the diagonal $n \times n$ matrix, where $[\boldsymbol{C}_u]_{ii} = c_{ui}$.

            $\boldsymbol{x}_u \leftarrow \left(\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{Y} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{p}_u$
        end for
    end for
end procedure
```

(a) **[3 pts]** Explain why we give $p_{ui} = 0$ (or $r_{ui} = 0$) the lowest confidence weight ($c_{ui} = 1$).

(b) **[6 pts]** Treat $\boldsymbol{y}_i$ as fixed for all $i \in \mathcal{I}$. Show that the optimal $\boldsymbol{x}_u$ with respect to the cost function (7) can be expressed as:

$$\boldsymbol{x}_u = \left(\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{Y} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{p}_u,$$

where $\boldsymbol{Y} = [\boldsymbol{y}_1 \cdots \boldsymbol{y}_n]^T \in \mathbb{R}^{n\times f}$, $\boldsymbol{C}_u = diag(c_{u1}, \ldots, c_{un}) \in \mathbb{R}^{n\times n}$, and $\boldsymbol{p}_u = [p_{u1}, \ldots, p_{un}]^T \in \mathbb{R}^n$.

Hint: You may use the following in your derivation. For any two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$, we have:

$$\frac{\partial \boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}} = \boldsymbol{b} \qquad\qquad \frac{\partial\|\boldsymbol{a}\|_2^2}{\partial\boldsymbol{a}} = \frac{\partial\boldsymbol{a}^T\boldsymbol{a}}{\partial\boldsymbol{a}} = 2\boldsymbol{a}.$$

(c) **[5 pts]** For the calculation of $\boldsymbol{x}_u$, a computational bottleneck is computing $\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{Y}$, where the naive solution requires time $\mathcal{O}(nf^2)$. It's because $\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{Y}$ has f^2 entries and each entry requires n times summation. A trick is to re-express $\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{Y} = \boldsymbol{Y}^T\boldsymbol{Y} + \boldsymbol{Y}^T\left(\boldsymbol{C}_u - \boldsymbol{I}\right)\boldsymbol{Y}$. Now, $\boldsymbol{Y}^T\boldsymbol{Y}$ is independent of u and can be precomputed. Denote n_u as the number of items u iteracted with.

How many nonzero entries does $\boldsymbol{C}_u - \boldsymbol{I}$ have?

Assuming that $\boldsymbol{C}_u - \boldsymbol{I}$ is represented as a sparse matrix, derive that the time complexity of computing $\boldsymbol{Y}^T\left(\boldsymbol{C}_u - \boldsymbol{I}\right)\boldsymbol{Y}$ for a single user u is $\mathcal{O}(n_uf^2)$. The real data set typically has $n_u \ll n$. We could see this trick significantly improves the naive solution which requires $\mathcal{O}(nf^2)$. Similarly, we could use the same trick for $\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{p}_u$, where $\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{p}_u = \boldsymbol{Y}^T\left(\boldsymbol{C}_u - \boldsymbol{I}\right)\boldsymbol{p}_u + \boldsymbol{Y}^T\boldsymbol{p}_u$

(d) **[16 pts]** We have provided a real dataset (`user_artists.txt`) in `implicit_feedback/data` folder containing the listening history of 3000 artists from 1882 users in Last.fm[3]. The file contains tab-separated triplets (one triplet per line) of the form $< u, i, r_{ui} >$, where u is a user label, i is an artist label and r_{ui} is the number of time user u interacted with artist i (e.g. listened to him). The file `artists.txt` contains tab-separated pairs (one pair per line) of the form $< i, s_i >$, where i is an artist and s_i is the name of the artist. We also provide a smaller dataset (`user_artists_small.txt`) for sanity check, which contains 100 artists and 100 users.

For the two datasets, calculate the sparsity ratio:

$$\alpha = \frac{\sum_{ui \in \mathcal{U} \times \mathcal{I}} \mathbb{1}[\![r_{ui} > 0]\!]}{\sum_{ui \in \mathcal{U} \times \mathcal{I}} \mathbb{1}[\![r_{ui} = 0]\!]}$$

Then, implement the implicit feedback recommendation system via ALS for this dataset. Here we assume $f = 3$ and $\lambda = 0.01$. For initialization, set $\boldsymbol{X}^{(0)}$ as a matrix with all elements $= 0.5$ and $\boldsymbol{Y}^{(0)}$ as a zero matrix. You have been provided a skeleton in the file `ALS.py` in the folder `implicit_feedback/code`. Complete the function `ALS`. Note that the template does not include code for other functions, such as reading the file, calculating the objective function, running iterations. You need to implement all the other necessary functions yourself. Answer the following questions:

Run your program for 100 iterations. What's initial C_{implicit}? What's C_{implicit} after the 1st iteration? Plot the value of the objective function C_{implicit} as a function of the number of iterations. Report top 2 favorite artists for user 0, 20 after the 1st and 100th iterations. See below for additional instructions on what to report.

Additional Instructions:

We found that there might be subtle variations in the final values based on the implementation and runtime environment. For minimizing these variations, please follow these instructions:

(1) Please do not round off values at any stage including the sparsity ratio (compute it programmatically). Also, avoid using explicit floating point precision typecasting such as **dtype=np.float32**.

(2) In your implementation, please store $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{C}_i - I$ and $\boldsymbol{C}_u - I$ as sparse matrix. You may use *csr_matrix* in library *scipy.sparse*. And please use *scipy.sparse.linalg.spsolve* to solve and update $\boldsymbol{x_u}$ and $\boldsymbol{y_i}$.

Note: Taking these subtle variations into account, we defined a longer list of potential top 3 recommendations for each user. Your predictions for each user are expected to be within these lists even though the ordering of the top recommendations might show small variations depending on the implementation and runtime environment.

[3] This dataset was built by the Information Retrieval group at Universidad Autonoma de Madrid (http://ir.ii.uam.es).

Hint 1: For the `user_artists_small.txt` dataset, after 1 iteration, C_{implicit} is between 1400 and 1600, after 100 iterations, the top 5 favorite artists for user 1 include 30 and 95.

Hint 2: Use the trick from part (c) to speed up your implementation. The expected run time for the the `user_artists.txt` dataset is 10-15 minutes.

What to submit:

(i) Answer to 3(a).

(ii) Proof for optimality of $\boldsymbol{x}_u = \left(\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{Y} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{Y}^T\boldsymbol{C}_u\boldsymbol{p}_u$.

(iii) The number of nonzero entries in $\boldsymbol{C}_u - \boldsymbol{I}$ and the proof for the $\mathcal{O}(n_u f^2)$ time complexity of computing $\boldsymbol{Y}^T\left(\boldsymbol{C}_u - \boldsymbol{I}\right)\boldsymbol{Y}$ for each u.

(iv) The sparsity ratio α .

(v) Initial C_{implicit} and C_{implicit} after the 1st iteration.

(vi) The top 2 artist recommendations for user 0, 20 after the 1st and 100th iterations.

(vii) A plot of C_{implicit} vs. the iteration over the execution of your implementation of ALS.

(viii) Upload your implementation for ALS to Gradescope.