- Compute the eigenvalue decomposition of $M^T M$ (*Use scipy.linalg.eigh function in Python*). The function returns two parameters: a list of eigenvalues (let us call this list *Evals*) and a matrix whose columns correspond to the eigenvectors of the respective eigenvalues (let us call this matrix *Evecs*). Sort the list *Evals* in descending order such that the largest eigenvalue appears first in the list. Also, re-arrange the columns in *Evecs* such that the eigenvector corresponding to the largest eigenvalue appears in the first column of *Evecs*. What are the values of *Evals* and *Evecs* (after the sorting and re-arranging process)?

- Based on the experiment and your derivations in part (c) and (d), do you see any correspondence between V produced by SVD and the matrix of eigenvectors *Evecs* (after the sorting and re-arranging process) produced by eigenvalue decomposition? If so, what is it?
  Note: The function scipy.linalg.svd returns V^T (not V).

- Based on the experiment and the expressions obtained in part (c) and part (d) for $M^T M$, what is the relationship (if any) between the eigenvalues of $M^T M$ and the singular values of M? Explain.
  Note: The entries along the diagonal of Σ (part (e)) are referred to as singular values of M. The eigenvalues of $M^T M$ are captured by the diagonal elements in Λ (part (d))

What to submit:

(i) Written solutions to questions 1(a) to 1(e) with explanations wherever required

(ii) Upload the code via Gradescope [1(e)]

2 k-means on Spark (20 points)

Note: This problem should be implemented in Spark. You should **not** use the Spark MLlib clustering library for this problem. You may store the centroids in memory if you choose to do so.

* * *

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.

Euclidean distance Given two points A and B in d dimensional space such that $A = [a_1, a_2 \cdots a_d]$ and $B = [b_1, b_2 \cdots b_d]$, the Euclidean distance between A and B is defined as:

$$||a - b|| = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \tag{1}$$

The corresponding cost function ϕ that is minimized when we assign points to clusters using the Euclidean distance metric is given by:

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} ||x - c||^2 \tag{2}$$

Note, that in the cost function the distance value is squared. This is intentional, as it is the squared Euclidean distance the algorithm is guaranteed to minimize.

Manhattan distance Given two random points A and B in d dimensional space such that $A = [a_1, a_2 \cdots a_d]$ and $B = [b_1, b_2 \cdots b_d]$, the Manhattan distance between A and B is defined as:

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

The corresponding cost function ψ that is minimized when we assign points to clusters using the Manhattan distance metric is given by:

$$\psi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} |x - c| \tag{4}$$

Iterative k-Means Algorithm: We learned the basic k-Means algorithm in class which is as follows: k centroids are initialized, each point is assigned to the nearest centroid and the centroids are recomputed based on the assignments of points to clusters. In practice, the above steps are run for several iterations. We present the resulting iterative version of k-Means in Algorithm 1.

Iterative k-Means clustering on Spark: Implement iterative k-means using Spark. Please use the dataset from `q2/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. The ID to download `data.txt` into a Colab is 1E-voIV2ctU4Brw022Na8RHVVRGOoNkO1

2. `c1.txt` contains k initial cluster centroids. These centroids were chosen by selecting $k = 10$ random points from the input data. The ID to download `c1.txt` into a Colab is 1yXNlZWMqUcAwDScBrkFChOHJwR1FZXmI

Algorithm 1 Iterative k-Means Algorithm

```
1: procedure ITERATIVE k-MEANS
2:     Select k points as initial centroids of the k clusters.
3:     for iterations := 1 to MAX_ITER do
4:         for each point p in the dataset do
5:             Assign point p to the cluster with the closest centroid
6:         end for
7:         Calculate the cost for this iteration.
8:         for each cluster c do
9:             Recompute the centroid of c as the mean of all the data points assigned to c
10:        end for
11:    end for
12: end procedure
```

3. `c2.txt` contains initial cluster centroids which are as far apart as possible, using Euclidean distance as the distance metric. (You can do this by choosing 1st centroid c1 randomly, and then finding the point c2 that is farthest from c1, then selecting c3 which is farthest from c1 and c2, and so on). The ID to download `c2.txt` into a Colab is 1vfovle9DgaeK0LnbQTH0j7kRaJjsvLtb

Tip: To download the datasets in Colab, you can follow the set-up instructions at the beginning of Colab2 and replace the ids correspondingly.

Set number of iterations (`MAX_ITER`) to 20 and 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.

(a) Exploring initialization strategies with Euclidean distance [10 pts]

1. **[5 pts]** Using the Euclidean distance (refer to Equation 1) as the distance measure, compute the cost function $\phi(i)$ (refer to Equation 2) for every iteration i. This means that, for your first 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`. Generate a graph where you plot the cost function $\phi(i)$ as a function of the number of iterations i=1..20 for `c1.txt` and also for `c2.txt`. You may use a single plot or two different plots, whichever you think best answers the theoretical questions we're asking you about.

   (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.)

2. **[5 pts]** What is the percentage change in cost after 10 iterations of the K-Means algorithm when the cluster centroids are initialized using `c1.txt` vs. `c2.txt` and the distance metric being used is Euclidean distance? Is random initialization of k-means using `c1.txt` better than initialization using `c2.txt` in terms of cost $\phi(i)$? Explain your reasoning.

(Hint: to be clear, the percentage refers to (cost[0]-cost[10])/cost[0].)

(b) Exploring initialization strategies with Manhattan distance [10 pts]

1. **[5 pts]** Using the Manhattan distance metric (refer to Equation 3) as the distance measure, compute the cost function $\psi(i)$ (refer to Equation 4) for every iteration i. This means that, for your first 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`. Generate a graph where you plot the cost function $\psi(i)$ as a function of the number of iterations i=1..20 for `c1.txt` and also for `c2.txt`. You may use a single plot or two different plots, whichever you think best answers the theoretical questions we're asking you about.

   (Hint: This problem can be solved in a similar manner to that of part (a). Also note that It's possible that for Manhattan distance, the cost do not always decrease. K-means only ensures monotonic decrease of cost for squared Euclidean distance. Look up K-medians to learn more.)

2. **[5 pts]** What is the percentage change in cost after 10 iterations of the K-Means algorithm when the cluster centroids are initialized using `c1.txt` vs. `c2.txt` and the distance metric being used is Manhattan distance? Is random initialization of k-means using `c1.txt` better than initialization using `c2.txt` in terms of cost $\psi(i)$? Explain your reasoning.

What to submit:

(i) Upload the code for 2(a) and 2(b) to Gradescope

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

(iii) Percentage improvement values and your explanation [2(a)]

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

(v) Percentage improvement values and your explanation [2(b)]

3 Latent Features for Recommendations (35 points)

Note: Please use native Python (Spark not required) to solve this problem. It usually takes several minutes to run, however, time may differ depending on the system you use.

* * *