CS246: Mining Massive Data Sets **Winter 2022**

Problem Set 2

Please read the homework submission policies at http://cs246.stanford.edu.

1 Singular Value Decomposition and Principal Component Analysis (20 points)

In this problem we will explore the relationship between two of the most popular dimensionality-reduction techniques, SVD and PCA, at a basic conceptual level. Before we proceed with the question itself, let us briefly recap the SVD and PCA techniques and a few important observations:

- First, recall that the eigenvalue decomposition of a *real*, *symmetric*, and *square matrix* B (of size $d \times d$) can be written as the following product:

  $$B = Q\Lambda Q^{\mathrm{T}}$$

  where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$ contains the eigenvalues of B (which are always real) along its main diagonal and Q is an orthogonal matrix containing the eigenvectors of B as its columns.

- Principal Component Analysis (PCA): Given a data matrix M (of size $p \times q$), PCA involves the computation of the eigenvectors of MM^T or M^TM. The matrix of these eigenvectors can be thought of as a rigid rotation in a high dimensional space. When you apply this transformation to the original data, the axis corresponding to the principal eigenvector is the one along which the points are most "spread out." More precisely, this axis is the one along which the variance of the data is maximized. Put another way, the points can best be viewed as lying along this axis, with small deviations from this axis. Likewise, the axis corresponding to the second eigenvector (the eigenvector corresponding to the second-largest eigenvalue) is the axis along which the variance of distances from the first axis is greatest, and so on.

- Singular Value Decomposition (SVD): SVD involves the decomposition of a data matrix M (of size $p \times q$) into a product: $U\Sigma V^T$ where U (of size $p \times k$) and V (of size $q \times k$) are column-orthonormal matrices[1] and Σ (of size $k \times k$) is a diagonal matrix. The entries along the diagonal of Σ are referred to as singular values of M. The key to understanding what SVD offers is in viewing the r columns of U, Σ, and V as representing concepts that are hidden in the original matrix M.

For answering the questions below, let us define a real matrix M (of size $p \times q$) and let us assume this matrix corresponds to a dataset with p data points and q dimensions.

[1] A matrix $U \in \mathbb{R}^{p\times q}$ is column-orthonormal if and only if $U^TU = I$ where I denotes the identity matrix

(a) [3 points]

Are the matrices MM^T and M^TM symmetric, square and real? Explain.

(b) [5 points]

Prove that the nonzero eigenvalues of MM^T are the same as the nonzero eigenvalues of M^TM. You may ignore multiplicity of eigenvalues. Are their eigenvectors the same?

(c) [2 points]

Given that we now understand certain properties of M^TM, write an expression for M^TM in terms of Q, Q^T and Λ where $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_d)$ contains the eigenvalues of M^TM along its main diagonal and Q is an orthogonal matrix containing the eigenvectors of M^TM as its columns?
Hint: Check the definition of eigenvalue decomposition provided in the beginning of the question to see if it is applicable.

(d) [5 points]

SVD decomposes the matrix M into the product $U\Sigma V^T$ where U and V are column-orthonormal and Σ is a diagonal matrix. Given that $M = U\Sigma V^T$, write a simplified expression for M^TM in terms of V, V^T and Σ.

(e) [5 points]

In this question, let us experimentally test if SVD decomposition of M actually provides us the eigenvectors (PCA dimensions) of M^TM. We strongly recommend students to use Python and suggested functions for this exercise.[2] Initialize matrix M as follows:

$$M = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 4 \\ 4 & 3 \end{bmatrix}$$

- Compute the SVD of M (*Use scipy.linalg.svd function in Python and set the argument* `full_matrices` *to False*). The function returns values corresponding to U, Σ and V^T. What are the values returned for U, Σ and V^T? *Note: Make sure that the first element of the returned array Σ has a greater value than the second element.*

[2]Other implementations of SVD and PCA might give slightly different results. Besides, you will just need fewer than five python commands to answer this entire question

- Compute the eigenvalue decomposition of M^TM (*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^TM, what is the relationship (if any) between the eigenvalues of M^TM 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^TM 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.