CS 189 Introduction to Machine Learning
Summer 2019 Marc Khoury & Brijen Thananjeyan HW7

Due: Monday, August 12 at 11:59 pm

Deliverables:

1. Submit a PDF of your homework, **with an appendix listing all your code**, to the Gradescope assignment entitled "Homework 7 Write-Up". In addition, please include, as your solutions to each coding problem, the specific subset of code relevant to that part of the problem. You may typeset your homework in LaTeX or Word (submit PDF format, **not** .doc/.docx format) or submit neatly handwritten and scanned solutions. **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.

   - In your write-up, please state with whom you worked on the homework.
   - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadvertently cheats.

     "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."

2. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "Homework 7 Code". Yes, you must submit your code twice: in your PDF write-up following the directions as described above so the readers can easily read it, and once in compilable/interpretable form so the readers can easily run it. Do **NOT** include any data files we provided. Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory. If your code cannot be executed, your solution cannot be verified.

1 Regularized and Kernel k-Means

Recall that in k-means clustering we attempt to minimize the objective

$$\min_{C_1,C_2,\ldots,C_k} \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2, \text{ where}$$

$$\mu_i = \mathrm{argmin}_{\mu_i \in \mathbb{R}^d} \sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2 = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j, \quad i = 1, 2, \ldots, k.$$

The samples are $\{x_1, \ldots, x_n\}$, where $x_j \in \mathbb{R}^d$. C_i is the set of sample points assigned to cluster i and $|C_i|$ is its cardinality. Each sample point is assigned to exactly one cluster.

(a) What is the minimum value of the objective when $k = n$ (the number of clusters equals the number of sample points)?

Solution: The value is 0, as every point can have its own cluster.

(b) (Regularized k-means) Suppose we add a regularization term to the above objective. The objective is now

$$\sum_{i=1}^{k} \left(\lambda \|\mu_i\|_2^2 + \sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2 \right).$$

Show that the optimum of

$$\min_{\mu_i \in \mathbb{R}^d} \lambda \|\mu_i\|_2^2 + \sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2$$

is obtained at $\mu_i = \frac{1}{|C_i|+\lambda} \sum_{x_j \in C_i} x_j$.

Solution:

Consider the function

$$f(\mu_i) = \left(\sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2 \right) + \lambda \|\mu_i\|_2^2.$$

Its gradient with respect to μ_i is

$$\nabla_{\mu_i} f(\mu_i) = \left(2 \sum_{x_j \in C_i} (\mu_i - x_j) \right) + 2\lambda \mu_i$$

$$= 2 \left((|C_i| + \lambda)\mu_i - \sum_{x_j \in C_i} x_j \right).$$

Setting it to zero, we have $\mu_i = \frac{1}{|C_i|+\lambda} \sum_{x_j \in C_i} x_j$. As the function f is convex, the minimum is obtained at $\mu_i = \frac{1}{|C_i|+\lambda} \sum_{x_j \in C_i} x_j$.

(c) Here is an example where we would want to regularize clusters. Suppose there are n students who live in a $\mathbb{R}^2$ Euclidean world and who wish to share rides efficiently to Berkeley for their final exam in CS189. The university permits k vehicles which may be used for shuttling students to the exam location. The students need to figure out k good locations to meet up. The students will then walk to the closest meet up point and then the shuttles will deliver them to the exam location. Let x_j be the location of student j, and let the exam location be at $(0, 0)$. Assume that we can drive as the crow flies, i.e., by taking the shortest path between two points. Write down an appropriate objective function to minimize the total distance that the students and vehicles need to travel. Hint: your result should be similar to the regularized k-means objective.

Solution:

The objective function that minimizes the total distance that the students and vehicles need to travel is

$$\min_{\mu_i \in \mathbb{R}^d} \sum_{i=1}^{k} \left(\|\mu_i\|_2 + \sum_{x_j \in C_i} \|x_j - \mu_j\|_2 \right). \tag{1}$$

Cathy Wu, Ece Kamar, Eric Horvitz. *Clustering for Set Partitioning with a Case Study in Ridesharing*. IEEE Intelligent Transportation Systems Conference (ITSC), 2016.

(d) (Kernel k-means) Suppose we have a dataset $\{x_i\}_{i=1}^n, x_i \in \mathbb{R}^\ell$ that we want to split into k clusters, i.e., finding the best k-means clustering *without the regularization*. Furthermore, suppose we know *a priori* that this data is best clustered in an impractically high-dimensional feature space $\mathbb{R}^m$ with an appropriate metric. Fortunately, instead of having to deal with the (implicit) feature map $\Phi : \mathbb{R}^\ell \to \mathbb{R}^m$ and (implicit) distance metric[1], we have a kernel function $\kappa(x_1, x_2) = \Phi(x_1) \cdot \Phi(x_2)$ that we can compute easily on the raw samples. How should we perform the kernelized counterpart of k-means clustering?

Derive the underlined portion of this algorithm, and show your work in deriving it. The main issue is that although we define the means μ_i in the usual way, we can't ever compute Φ explicitly because it's way too big. Therefore, in the step where we determine which cluster each sample point is assigned to, we must use the kernel function κ to obtain the right result. (Review the lecture on kernels if you don't remember how that's done.)

Algorithm 1: Kernel k-means

```
Require: Data matrix $X \in \mathbb{R}^{n \times d}$; Number of clusters $K$; kernel function $\kappa(x_1, x_2)$
Ensure: Cluster class class($j$) for each sample $x_j$.
  function KERNEL-K-MEANS($X, c$)
      Randomly initialize class($j$) to be an integer in $1, 2, \ldots, K$ for each $x_j$.
      while not converged do
          for $i \leftarrow 1$ to $K$ do
              Set $S_i = \{j \in \{1, 2, \ldots, n\} : \text{class}(j) = i\}$.
          for $j \leftarrow 1$ to $n$ do
              Set class($j$) = $\text{argmin}_k$ ________________________________
      Return $S_i$ for $i = 1, 2, \ldots, c$.
  end function
```

Solution: Given a clustering S_i, we define

$$\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} \Phi(x)$$

[1]Just as how the interpretation of kernels in kernelized ridge regression involves an implicit prior/regularizer as well as an implicit feature space, we can think of kernels as generally inducing an implicit distance metric as well. Think of how you would represent the squared distance between two points in terms of pairwise inner products and operations on them.

to minimize $\sum_{x\in S_i}\|\Phi(x)-\mu_i\|_2^2$. (But we don't explicitly compute μ_i!)

Given this choice of the μ_k's, the K-means clustering is found by assigning each x_i to the cluster $\arg\min_k f(i,k)$, where

$$\begin{aligned}
f(i,k) &= \|\Phi(x_i)-\mu_k\|^2\\
&= \Phi(x_i)\cdot\Phi(x_i) - 2\Phi(x_i)\cdot\mu_k + \mu_k\cdot\mu_k\\
&= \Phi(x_i)\cdot\Phi(x_i) - \frac{2}{|S_k|}\sum_{x_j\in S_k}\Phi(x_i)\cdot\Phi(x_j) + \frac{1}{|S_k|^2}\sum_{x_j\in S_k}\sum_{x_l\in S_k}\Phi(x_j)\cdot\Phi(x_l)\\
&= \kappa(x_i,x_i) - \frac{2}{|S_k|}\sum_{x_j\in S_k}\kappa(x_i,x_j) + \frac{1}{|S_k|^2}\sum_{x_j\in S_k}\sum_{x_l\in S_k}\kappa(x_j,x_l).
\end{aligned}$$

The first term does not vary with k, so we can drop it. Therefore, we write

$$\text{class}(i) = \arg\min_k\left(\frac{1}{|S_k|^2}\sum_{x_j,x_l\in S_k}\kappa(x_j,x_l) - \frac{2}{|S_k|}\sum_{x_j\in S_k}\kappa(x_i,x_j)\right). \tag{2}$$

(e) The expression you derived may have unnecessary terms or redundant kernel computations. Explain how to eliminate them; that is, how to perform the computation quickly without doing irrelevant computations or redoing computations already done.

Solution: First, we dropped the term $\kappa(x_i,x_i)$. Second, observe that the first summation above is the same for every sample point x_i, so we can compute it once for each cluster per iteration, and use it with every sample point. Third, observe that we need to compute a kernel computation for every pair of sample points (not necessarily distinct), but we only need to do it once at the beginning of the algorithm. This amounts to computing every entry of the kernel matrix, keeping in mind that the kernel function (and the kernel matrix) is symmetric to avoid redundant computations. Then the computed kernel values can be used multiple times per iteration, in both the first and second summations.

2 Low-Rank Approximation

Low-rank approximation tries to find an approximation to a given matrix, where the approximation matrix has a lower rank compared to the original matrix. This is useful for mathematical modeling and data compression. Mathematically, given a matrix M, we try to find $\hat{M}$ in the following optimization problem,

$$\underset{\hat{M}}{\operatorname{argmin}}\ \|M-\hat{M}\|_F \quad \text{subject to} \quad \operatorname{rank}(\hat{M})\le k \tag{3}$$

where $\|A\|_F = \sqrt{\sum_i\sum_j a_{ij}^2}$ is the Frobenius norm, i.e., the sum of squares of all entries in the matrix, followed by a square root.

This problem can be solved using Singular Value Decomposition (SVD). Specifically, let $M = U\Sigma V^\top$, where $\Sigma = \operatorname{diag}(\sigma_1,...,\sigma_n)$. Then a rank-$k$ approximation of M can be written as $\hat{M} = U\hat{\Sigma}V^\top$, where $\hat{\Sigma} = \operatorname{diag}(\sigma_1,...,\sigma_k,0,...,0)$. In this problem, we aim to perform this approximation method on gray-scale images, which can be thought of as a 2D matrix.

(a) Using the image `low-rank_data/face.jpg`, perform a rank-5, rank-20, and rank-100 approximation on the image. Show both the original image as well as the low-rank images you obtain in your report.

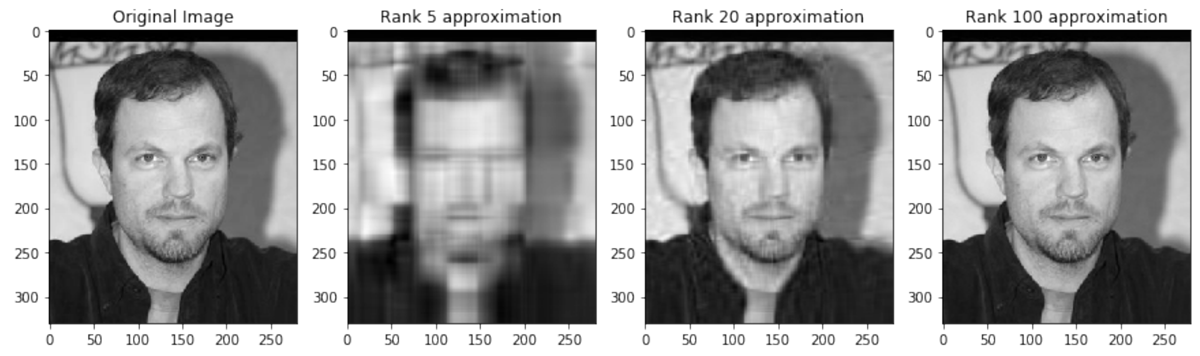
Solution:

```
import numpy as np
import matplotlib.pyplot as plt
from imageio import imread

def low_rank_approximation(X, rank):
    U, S, Vt = np.linalg.svd(X)
    Sk = np.zeros([U.shape[0], Vt.shape[0]])
    Sk[np.arange(rank), np.arange(rank)] = S[:rank]
    return U @ Sk @ Vt

face = imread("./data/face.jpg")
rank5_approx = low_rank_approximation(face, 5)
rank20_approx = low_rank_approximation(face, 20)
rank100_approx = low_rank_approximation(face, 100)

f, ax = plt.subplots(nrows= 1, ncols= 4, figsize = (15, 5))
ax[0].imshow(face, cmap='gray')
ax[1].imshow(rank5_approx, cmap='gray')
ax[2].imshow(rank20_approx, cmap='gray')
ax[3].imshow(rank100_approx, cmap='gray')
ax[0].set_title("Original Image")
ax[1].set_title("Rank 5 approximation")
ax[2].set_title("Rank 20 approximation")
ax[3].set_title("Rank 100 approximation")
```

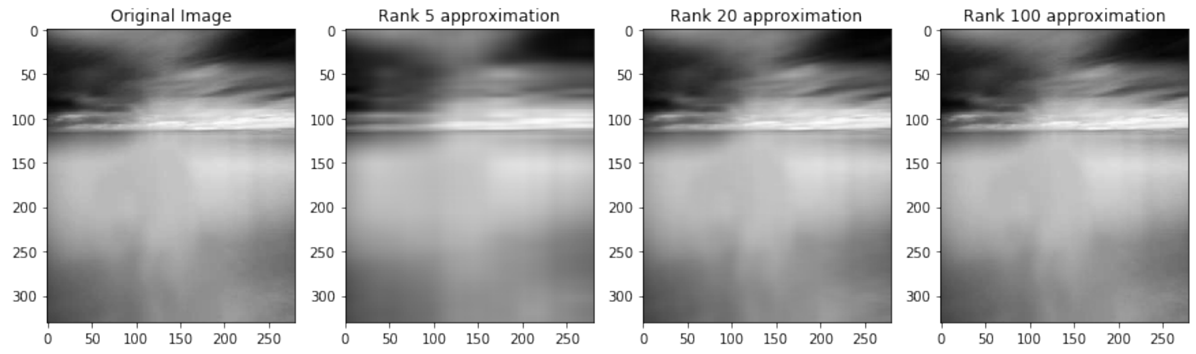


(b) Now perform the same rank-5, rank-20, and rank-100 approximation on `low-rank_data/sky.jpg`. Show both the original image as well as the low-rank images you obtain in your report.

Solution:

```
sky = imread("./data/sky.jpg")
rank5_approx = low_rank_approximation(sky, 5)
rank20_approx = low_rank_approximation(sky, 20)
rank100_approx = low_rank_approximation(sky, 100)

f, ax = plt.subplots(nrows= 1, ncols= 4, figsize = (15, 5))
ax[0].imshow(sky, cmap='gray')
ax[1].imshow(rank5_approx, cmap='gray')
ax[2].imshow(rank20_approx, cmap='gray')
ax[3].imshow(rank100_approx, cmap='gray')
ax[0].set_title("Original Image")
ax[1].set_title("Rank 5 approximation")
ax[2].set_title("Rank 20 approximation")
ax[3].set_title("Rank 100 approximation")
```



(c) In one plot, plot the Mean Squared Error (MSE) between the rank-k approximation and the original image for both `low-rank_data/face.jpg` and `low-rank_data/sky.jpg`, for k ranging from 1 to 100. Be sure to label each curve in the plot. The MSE between two images $I, J \in R^{w \times h}$ is

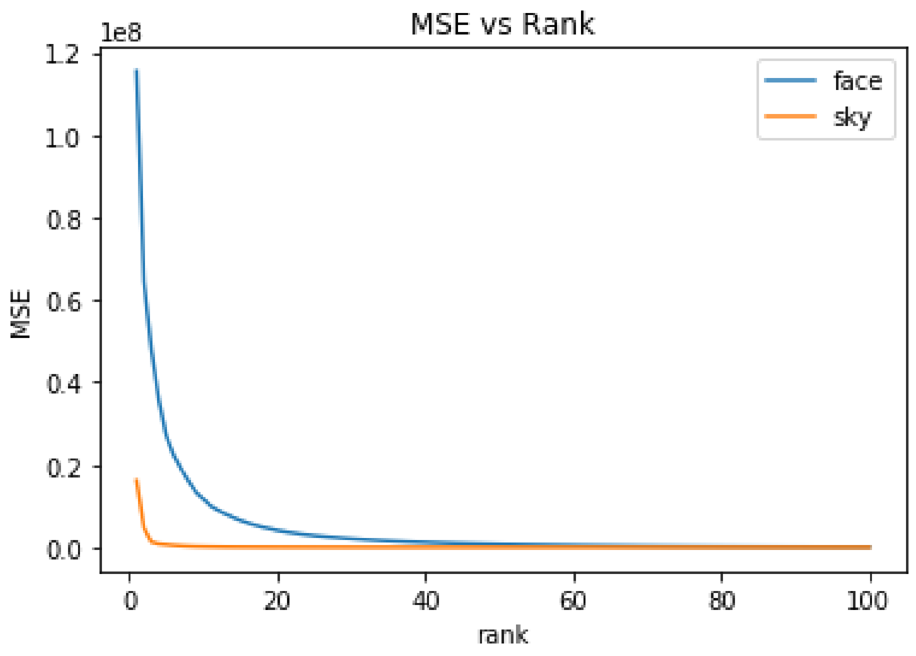
$$\mathrm{MSE}(I, J) = \sum_{i,j} (I_{i,j} - J_{i,j})^2. \tag{4}$$

Solution:

```
def mse(img1, img2):
    return np.sum((img1 - img2)**2)

face_mse, sky_mse = [], []
for i in range(1, 101):
    face_approx = low_rank_approximation(face, i)
    sky_approx = low_rank_approximation(sky, i)
    face_mse.append(mse(face_approx, face))
    sky_mse.append(mse(sky_approx, sky))

plt.plot(range(1, 101), face_mse, label='face')
plt.plot(range(1, 101), sky_mse, label='sky')
plt.title("MSE vs Rank")
plt.xlabel("rank")
plt.ylabel("MSE")
plt.legend()
```



(d) Find the lowest-rank approximation for which you begin to have a hard time differentiating the original and the approximated images. Compare your results for the face and the sky image. What are the possible reasons for the difference?