Assignment 2
CS289: Algorithmic Machine Learning, Fall 2016
Due: October 26, 10PM

Guidelines for submitting the solutions:

- The assignments need to be submitted on Gradescope. Make sure you follow all the instructions - they are simple enough that exceptions will not be accepted.
- To save my eyes (and perhaps, more importantly, a few of your points) please use a good scanner and/or digitize the scan as per the instructions on the class web page.
- Start each problem or sub-problem on a separate page even if it means having a lot of white-space and write/type in large font.
- The solutions need to be submitted by 10 PM on the due date. No late submissions will be accepted.
- Please adhere to the code of conduct outlined on the class page.

1. Exercise 3.4 from [BHK]. [2 points]

2. The goal of this exercise is to partially show one of the remarkable properties of SVD that we discussed in class. Let A be a $n \times d$ matrix and let $A = U\Sigma V^T$ be its singular-value decomposition where Σ is a $r \times r$ diagonal matrix with $\Sigma_{11} \geq \Sigma_{22} \geq \ldots \geq \Sigma_{rr}$. Let $\sigma_i = \Sigma_{ii}$. Show that $\max_{v:\|v\|=1} \|Av\| \leq \sigma_1$. [2 points]
   (Hint: Choose an appropriate orthonormal basis for $\mathbb{R}^d$ and use the representation of vectors v in this basis to bound $\|Av\|$.)

3. Start by taking a picture of something around you with resolution about 1024x1024. You can view the image as 3 matrices (one for each color Red, Green, Blue). Now, compute the best rank-k approximation for these matrices and render the resulting image. Do this for $k = \{10, 20, 50, 100, 200\}$ (as I showed in class for a specific image). Your submission should be a) A screenshot of your code and b) The original image followed by the approximations (in order of increasing values of k). You can use MATLAB, R, Numpy/Scipy, or Julia for this. [2 points]

4. Let n be a large number (say $n = 10,000$) and k a small number (say 20). Let A be a sparse $n \times n$ matrix with density p (say 0.01). MATLAB (and other numerical analysis software) can exploit the sparsity of A to compute the top singular vector quite efficiently. Next, let U be a $n \times k$ matrix U and let $B = UU^T$. Computing the first right singular vector of B

directly on MATLAB would be quite slow. But, the first right singular vector of B is the same as the first left singular vector of U which you can compute much more efficiently.

Now, suppose we had to compute the first singular vector of $C = A + UU^T$ (such situations arise often in practice). Doing so directly would be quite expensive.

(a) Give an algorithm based on the power iteration to approximate the top singular vector of C. Show that you can do this without even computing C explicitly and prove a bound on the per-iteration running-time (in terms of n, p, k) of the power iteration. [1 point]

(b) You can use MATLAB, R, Numpy/Scipy, or Julia for this exercise. Generate a random $n \times n$ sparse matrix with density p for $n = 10,000$, $p = 0.01$. (In MATLAB, you can do this by sprand(n,n,p).) Generate a random $n \times k$ matrix U. Compute the top singular vector/value of $C = A + UU^T$ using the in-nuilt MATLAB command and time the operation. Implement the power iteration to compute the top singular vector of $C = A + UU^T$ (say for $10, 20, 30, ..., 100$ iterations) and time the algorithm. Is there a difference in speed?

Your submission should be the following items. [3 points]

i. A screenshot of your code for the power iteration (only the important part - this should only take a few lines).

ii. A table or figure showing the run-times of the in-built MATLAB command and the run-times for different number of iterations obtained by averaging over 10 runs (as A, U are random - it is better to consider multiple runs).

iii. A figure plotting the number of iterations (on the x-axis) with the error on the y-axis averaged over 10 runs.