CMSC 25300 / 35300
Homework 5

1. **The informative SVD**. Let $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ with $\text{rank}(\boldsymbol{X}) = k < \min\{n, p\}$. Each row represents one training sample, and each column represents one feature. Let $\boldsymbol{y} \in \mathbb{R}^n$ be the response vector for each sample. The full SVD of $\boldsymbol{X}$ is given by $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{u}_1, \ldots, \boldsymbol{u}_k, \ldots, \boldsymbol{u}_n \end{bmatrix}, \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\sigma}_1 & & & & \\ & \ddots & & & \\ & & \boldsymbol{\sigma}_k & & \\ & & & 0 & \\ & & & & \ddots \end{bmatrix}, \qquad \boldsymbol{V}^T = \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_k^T \\ \vdots \\ \boldsymbol{v}_p^T \end{bmatrix}.$$

   a. Decompose $\boldsymbol{X}$ to be a sum of rank-one matrices.

   b. Think of each row of $\boldsymbol{X}$ as a point in $\mathbb{R}^p$. Based on the full SVD, find a basis for the best 1d subspace (line through origin) fit to these n data points.

   c. What are the SVDs of $\boldsymbol{X}^T$, $\boldsymbol{X}\boldsymbol{X}^T$, and $\boldsymbol{X}^T\boldsymbol{X}$?

   d. Now we focus on the least squares problem, and try to find a weight vector $\boldsymbol{w}$ by minimizing $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$.

      i) Find a basis for all weight vectors that satisfy $\boldsymbol{X}\boldsymbol{w} = 0$. (If $\boldsymbol{X}\boldsymbol{w} = 0$, then, for any $\hat{\boldsymbol{w}}$ such that $\boldsymbol{y} = \boldsymbol{X}\hat{\boldsymbol{w}}$, we have $\boldsymbol{y} = \boldsymbol{X}(\hat{\boldsymbol{w}} + \boldsymbol{w})$.)

      ii) Prove the least squares problem is equivalent to minimizing $\|\tilde{\boldsymbol{y}} - \boldsymbol{\Sigma}\tilde{\boldsymbol{w}}\|_2^2$, for $\tilde{\boldsymbol{y}} = U^T\boldsymbol{y}$ and $\tilde{\boldsymbol{w}} = V^T\boldsymbol{w}$, and find the optimal $\tilde{\boldsymbol{w}}$ with the smallest norm.

      iii) Find the optimal solution to the original least squares problem with the smallest norm.

2. In this problem you will work with and analyze the dataset `jesterdata.mat`. The dataset contains an $m = 100$ by $n = 7200$ dimensional matrix X. Each row of X corresponds to a joke, and each column corresponds to a user. Each of the users rated the quality of each joke on a scale of $[-10, 10]$.

   a. Suppose that you work for a company that makes joke recommendations to customers. You are given a large dataset X of jokes and ratings. It contains p reviews for each joke. The reviews were generated by p users who represent a diverse set of tastes. Each reviewer rated every movie on a scale of $[-10, 10]$. A new customer has rated $n = 25$ of the jokes, and the goal is to predict another joke that the customer will like based on her n ratings. Use the first $p = 20$ columns of X for this prediction problem (so that the problem is over-determined). *Specifically, we will think of the new customer's ratings as a weighted sum of the other $p = 20$ customers' ratings, and we will use the $n = 25$ joke ratings by these $p = 20$ customers plus the $n = 25$ joke ratings by the new customer to*

learn the weights. The new customer's ratings are contained in the file `newuser.mat`, also on canvas, in a vector b. The jokes she didn't rate are indicated by a (false) score of -99. Compare your predictions to her complete set of ratings, contained in the vector `truey`. Her actual favorite joke was number 29. Does it seem like your predictor is working well?

Here is some code to help you get started:

```
#python
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt

# load the data matrix X
d_jest = sio.loadmat('jesterdata.mat')
X = d_jest['X']

# load known ratings y and true ratings truey
d_new = sio.loadmat('newuser.mat')
y = d_new['y']
true_y = d_new['truey']

# total number of joke ratings should be m = 100, n = 7200
m, n = X.shape

# train on ratings we know for the new user
train_indices = np.squeeze(y != -99)
num_train = np.count_nonzero(train_indices)

# test on ratings we don't know
test_indices = np.logical_not(train_indices)
num_test = m - num_train

X_data = X[train_indices, 0:20]
y_data = y[train_indices]
y_test = true_y[test_indices]

# solve for weights

# compute predictions

# measure performance on training jokes


# display results
ax1 = plt.subplot(121)
sorted_indices = np.argsort(np.squeeze(y_data))
ax1.plot(
    range(num_train), y_data[sorted_indices], 'b.',
    range(num_train), y_hat_train[sorted_indices], 'r.'
)
ax1.set_title('prediction␣of␣known␣ratings␣(trained␣with␣20␣users)
```

```
    ↪ ')
ax1.set_xlabel('jokes␣(sorted␣by␣true␣rating)')
ax1.set_ylabel('rating')
ax1.legend(['true␣rating', 'predicted␣rating'], loc='upper␣left')
ax1.axis([0, num_train, -15, 10])

print("Average␣l_2␣error␣(train):", avgerr_train)

# measure performance on unrated jokes


# display results
ax2 = plt.subplot(122)
sorted_indices = np.argsort(np.squeeze(y_test))
ax2.plot(
    range(num_test), y_test[sorted_indices], 'b.',
    range(num_test), y_hat_test[sorted_indices], 'r.'
)
ax2.set_title('prediction␣of␣unknown␣ratings␣(trained␣with␣20␣
    ↪ users)')
ax2.set_xlabel('jokes␣(sorted␣by␣true␣rating)')
ax2.set_ylabel('rating')
ax2.legend(['true␣rating', 'predicted␣rating'], loc='upper␣left')
ax2.axis([0, num_test, -15, 10])

print("Average␣l_2␣(test):", avgerr_test)

plt.show()
```

b. Repeat the prediction problem above, but this time use the entire X matrix. Note that now the problem is under-determined. Explain how you will solve this prediction problem and apply it to the data. Does it seem like your predictor is working? How does it compare to the first method based on only 20 users?

c. Propose a method for finding one other user that seems to give the best predictions for the new user. How well does this approach perform? Now try to find the best two users to predict the new user.

d. Use the Matlab function `svd` with the 'economy size' option or the Python function `numpy.linalg.svd` with the `full_matrices` option set to 'false' to compute the SVD of $X = U\Sigma V^T$. Plot the spectrum of X. What is the rank of X? How many dimensions seem important? What does this tell us about the jokes and users?

e. Visualize the dataset by projecting the columns and rows on to the first three principal component directions. Use the `rotate` tool in the Matlab plot (or in Python make the figure interactive via

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
```

```
%matplotlib notebook

fig = ...
```

) to get different views of the three-dimensional projections. Discuss the structure of the projections and what it might tell us about the jokes and users.

f. One easy way to compute the first principal component for large datasets like this is the so-called power method (see `http://en.wikipedia.org/wiki/Power_iteration`). Specifically, let $A = XX^\top$; then A is "diagonalizable" and the first "eigenvector" of A is the first left singular vector of X, while the first "eigenvalue" of A is the squared first singular value of X (i.e. σ_1^2). Explain the power method and why it works. Write your own code to implement the power method in Matlab (or other language) and use it to compute the first column of U and V in the SVD of X. Does it produce the same result as Matlab's (or other language) built-in `svd` function?

g. The power method is based on an initial starting vector. Give one example of a starting vector for which the power method will fail to find the first left and right singular vectors in this problem.

3. Many sensing and imaging systems produce signals that may be slightly distorted or blurred (e.g., an out-of-focus camera). In such situations, algorithms are needed to deblur the data to obtain a more accurate estimate of the true signal. The Matlab code `blurring.m` or Python code `blurring.py` generates a random signal and a blurred and noisy version of it, similar to the example shown below. The code simulates this equation:

$$\boldsymbol{y} \;=\; \boldsymbol{X}\boldsymbol{w} \;+\; \boldsymbol{e}\;,$$

where $\boldsymbol{y}$ is the blurred and noisy signal, $\boldsymbol{X}$ is a matrix that performs the blurring operation, $\boldsymbol{w}$ is the true signal, and $\boldsymbol{e}$ is a vector of errors/noise. The goal is to estimate $\boldsymbol{w}$ using $\boldsymbol{y}$ and $\boldsymbol{X}$.

a. Implement the standard least squares, truncated SVD, and regularized least squares (i.e., ridge regression) methods for this problem.

b. Experiment with different averaging functions (i.e., different values of k in the `blurring` code) and with different noise levels (σ in the `blurring` code). How do the blurring and noise level affect the value of the regularization parameters that produce the best estimates?