CMSC 25300 / 35300
Homework 5

1. In this problem you will work with an analyze the dasaset `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 of jokes. 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 overdetermined). *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 underdetermined. 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 sprectrum 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 principle 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 principle component for large datasets like this is the so-called power method (see `http://en.wikipedia.org/wiki/Power_iteration`). 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.

2. Consider the face emotion classification problem from HW 2. Design and compare the performances of the classifiers proposed in **a** and **b**, below. In each case, divide the dataset into 8 equal sized subsets (e.g., examples $1-16$, $17-32$, etc). Use 6 sets of the data to estimate

$\boldsymbol{w}$ for each choice of the *regularization parameter*, select the best value for the regularization parameter by estimating the error on one of the two remaining sets of data, and finally use the $\boldsymbol{w}$ corresponding to the best value of the regularization parameter to predict the labels of the remaining "hold-out" set. Compute the number of mistakes made on this hold-out set and divide that number by 16 (the size of the set) to estimate the error rate. Repeat this process 56 times (for the 8×7 different choices of the sets used to select the regularization parameter and estimate the error rate) and average the error rates to obtain a final estimate.

a) Truncated SVD solution. Use the pseudo-inverse $\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T$, where $\boldsymbol{\Sigma}^{-1}$ is computed by inverting the k largest singular values and setting others to zero. Here, k is the regularization parameter and it takes values $k = 1, 2, \ldots, 9$; i.e., compute 9 different solutions, $\widehat{\boldsymbol{w}}_k$.

b) Regularized LS. Let $\widehat{\boldsymbol{w}}_\lambda = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$, for the following values of the regularization parameter $\lambda = 0, 2^{-1}, 2^0, 2^1, 2^2, 2^3$, and 2^4. Show that $\widehat{\boldsymbol{w}}_\lambda$ can be computed using the SVD and use this fact in your code.

Here is some code to get you started:

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

d = sio.loadmat('face_emotion_data.mat')
X = d['X']
y = d['y']

n, p = np.shape(X)

# error rate for regularized least squares
error_RLS = np.zeros((8, 7))
# error rate for truncated SVD
error_SVD = np.zeros((8, 7))

# SVD parameters to test
k_vals = np.arange(9) + 1
param_err_SVD = np.zeros(len(k_vals))

# RLS parameters to test
lambda_vals = np.array([0, 0.5, 1, 2, 4, 8, 16])
param_err_RLS = np.zeros(len(lambda_vals))
```

c) Use the original dataset to generate 3 new features for each face, as follows. Take the 3 new features to be random linear combination of the original 9 features. This can be done for instance with the Matlab command `X*randn(9,3)` and augmenting the original matrix $\boldsymbol{X}$ with the resulting 3 columns. Will these new features be helpful for

classification? Why or why not? Repeat the experiments in (a) and (b) above using the 12 features.

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?

4. PageRank. Let us focus on a web of n pages. Define an $n \times n$ adjacency matrix M, corresponding to n pages as

$$M_{ij} = \begin{cases} 1, & \text{if page } j \text{ has a link to page } i \\ 0, & \text{otherwise} \end{cases}$$

Let N_j be the number of pages that page j points to. Then we can define the matrix A via

$$A_{ij} = \begin{cases} \frac{1}{N_j}, & \text{if } M_{ij} = 1 \\ 0, & \text{otherwise} \end{cases}$$

However, the above definition can be problematic when there are "dangling nodes" (when page j has no links point to other pages, i.e. $N_j = 0$). If j is a dangling node, we further define $A_{ij} = \frac{1}{n}$ for all $i = 1, \ldots, n$. Then we could compute the PageRank vector $\boldsymbol{\pi} \in \mathbb{R}^n$ by repeatedly multiplying by A until convergence:

$$\boldsymbol{\pi}^{(1)} = A\boldsymbol{\pi}^{(0)},\ \boldsymbol{\pi}^{(2)} = A\boldsymbol{\pi}^{(1)},\ \ldots, \boldsymbol{\pi}^{(t)} = A\boldsymbol{\pi}^{(t-1)}$$

The elements of the converged vector $\boldsymbol{\pi}$ are the PageRank of corresponding pages, and the matrix A is also called "Google matrix".

Note that if we can find the converged page rank vector $\boldsymbol{\pi}$, $\boldsymbol{\pi}$ must be an eigenvector of the matrix A with eigenvalue one.

a) Given any matrix A, A and A^T have the same set of eigenvalues. Use this fact to show that if the columns of A sum to one, one must be one of the eigenvalues of the "Google matrix" A.

b) **Damping** In real application, Google found it useful to add a "damping factor" α s.t. $0 < \alpha < 1$. The modified "Google matrix" G is then defined to be

$$G = \alpha A + (1 - \alpha)\boldsymbol{u}\mathbb{1}^T$$

where $\mathbb{1} \in \mathbb{R}^n$ is all-ones vector and $\boldsymbol{u}$ models the likelihood of users clicking a page (whether or not there is a link to it on current page). Theoretically, we could use any vector $\boldsymbol{u} \in \mathbb{R}^n$, but the columns of G might not sum to one. What condition on $\boldsymbol{u}$ would we need to keep the columns of G still sum to one? Hint: in practice, the "Google matrix" is defined as

$$G = \alpha A + (1 - \alpha)\frac{1}{n}\mathbb{1}\mathbb{1}^T$$

c) Now let's stick to the definition of G as $G = \alpha A + (1 - \alpha)\frac{1}{n}\mathbb{1}\mathbb{1}^T$. Intuitively, you can think of the damped PageRank as following: a user who clicks a random link on the current page with probability α, but with probability $1 - \alpha$ the user would choose a uniformly random page (whether or not there is a link to it). Consider a world of n pages, in which every page (including Facebook) links to Facebook, and no page links anywhere else. Facebook is page 1, and the other $n - 1$ pages are page 2 to n. With damping parameter α, what is the adjacency matrix M in this case? And the Google matrix G? If the PageRank vector $\boldsymbol{\pi}$ is a probability vector in $\mathbb{R}^n$ (elements sum up to 1), what is the PageRank of Facebook? What about the rank of the rest $n - 1$ pages? Express these two ranks in terms of damping parameter α and total number of pages n.

5. PCR you will experiment PCR on MNIST dataset, a well known handwritten digit recognition dataset. A small data subset is provided in *mnist.mat*. It contains **two** types of digits and each digit has 100 training images and 100 testing images. Each image $\mathbf{X}$ has size of 28×28 and target $\mathbf{Y} \in \{-1, 1\}$. Since raw data is in high dimension, we're trying to use PCA to first compress data into lower dimensions where the decision boundary is easy to draw.

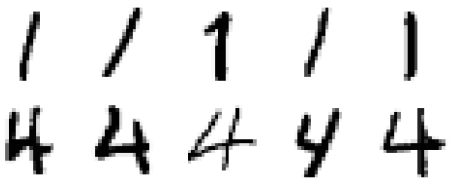


Figure 1: Image samples in training set

a) Plot the curve of reconstruction accuracy vs. number of principle components (2,4,8,10,12,14) used for reconstruction.

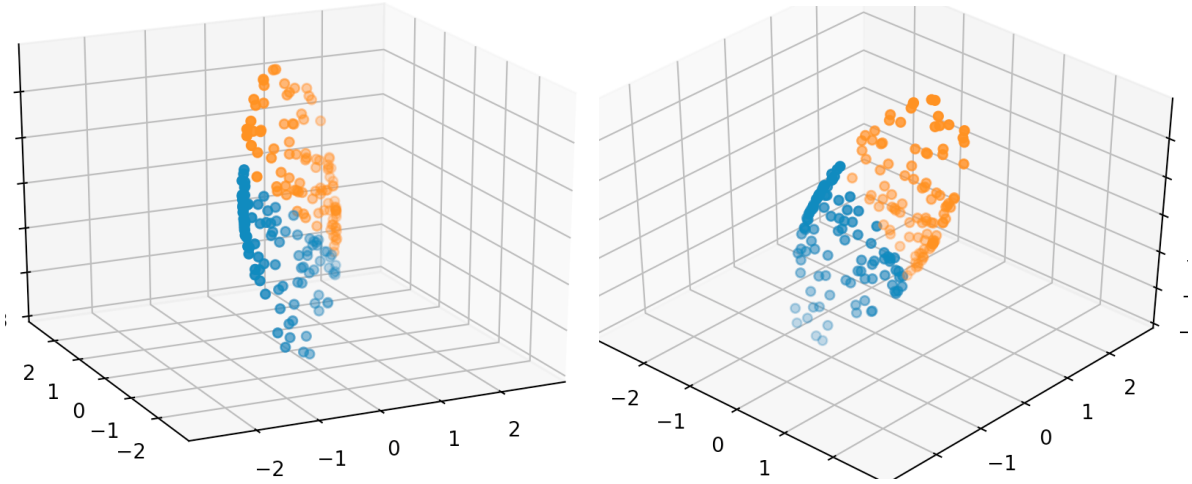
Take x number rows from V

b) Use the truncated SVD $(v1, v2)$ to project training data into 2d space and visualize the projected points. Mark different digit classes with different colors.

c) Fit a linear regression model that predict the digit type in two dimensional space. Report the training accuracy and the testing accuracy.

d) Plot the classification decision boundary in 2d space.

6. PCA In the following plots, a set of 3d points belonging to two classes are given in `pca_3d.mat`. In this question you need to write your own code to compute PCA to project the 3d data into lower dimensions. Note: you can **only** use subroutines like svd, eig for decomposition.



a) Compute all PCA axes and check how many of them are necessary, i.e., with nonzero singular values?

b) Project the data into 1 and 2 dimensional space, visualize the projected points

c) Whitening PCA: A whitening transformation is an important pre-processing step for many algorithms, which transforms a data matrix (X) into a new matrix $(\tilde{X})$ with $\tilde{X}^T\tilde{X} = I$. If X^TX has eigendecomposition $Q\Lambda Q^T$, then the whitening PCA matrix could be written in the following form:

$$W^{PCA} = \Lambda^{-1/2}Q^T, \text{ where } X^TX = Q\Lambda Q^T$$

and then we could form $\tilde{X}$ as $\tilde{X} = W^{PCA}X$. Now, after whitening transformtaion, project and visualize the reconstructed data into 2 dimensional space.

7. **Matrix Completion (OPTIONAL)** Recall the matrix completion algorithm in the lecture note. The **nuclear norm** of matrix A is defined as the sum of the singular values of A, i.e. $||A||_* = \sum_{i=1}^{r} \sigma_i(A)$, where the $\sigma_1(A), ..., \sigma_r(A)$ are the singular values of A.

   a) Now we define **Trace** of a matrix $A \in \mathbb{R}^{n \times n}$ as the sum of the diagonal entries, i.e. $\mathbf{Tr}(A) = \sum_{i=1}^{n} A_{ii}$. Show that if matrix A is symmetric positive semi-definite (i.e. SVD of A can be written as $A = U\Sigma U^{-1}$), the nuclear norm of A is equal to its trace. (Nuclear norm is sometimes called trace norm.) Hint: $\mathbf{Tr}(AB) = \mathbf{Tr}(BA)$

   b) Recall the iterative singular value thresholding. We test the convergence based on the **Frobenius norm** $|| \cdot ||_F$. The Frobenius norm of a matrix $A \in \mathbb{R}^{n \times n}$ can be defined as $||A||_F = \sqrt{\sum_{i,j} |A_{ij}|^2}$. Show that

$$||A||_F = \sqrt{\mathbf{Tr}(A^T A)}$$

   c) Show that if U and V are orthogonal, then $||UA||_F = ||AV||_F = ||A||_F$. In other word, the Frobenius norm is not changed by a pre- or post- orthogonal transformation.

   d) Show that $||A||_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$, where $\sigma_1, \ldots, \sigma_r$ are the singular values of A. Then show that $\sigma_{max}(A) \leq ||A||_F \leq \sqrt{r}\sigma_{max}(A)$.