CS/ECE/ME532 Activity 13

Estimated time: 25 minutes for Q1 and 35 minutes for Q2.

1. We've previously considered several low rank approximations for matrices based on the SVD. Let an n-by-p matrix $\boldsymbol{X}$ with $n \leq p$ be expressed as

$$\boldsymbol{X} = \sum_{i=1}^{n} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

where σ_i is the ith singular value with left singular vector $\boldsymbol{u}_i$ and right singular vector $\boldsymbol{v}_i$. The rank-r approximation is

$$\boldsymbol{X}_r = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

where $r \leq n$. Define the error between $\boldsymbol{X}$ and the rank-r approximation as $\boldsymbol{E}_r = \boldsymbol{X} - \boldsymbol{X}_r$.

a) Find the SVD of $\boldsymbol{E}_r$ in terms of the σ_i, $\boldsymbol{u}_i$, and $\boldsymbol{v}_i$. $E_r = \sum_{i=r+1}^{n} \sigma_i u_i v_i^T$

b) Suppose $\boldsymbol{X}$ is full rank. What is the rank of $\boldsymbol{E}_r$? $\text{Rank}(E_r) = \text{Rank}(X) - r$

c) Find the operator norm (which is also called the 2-norm of a matrix) of the error matrix $||\boldsymbol{E}_r||_{op}$ in terms of the SVD parameters for $\boldsymbol{X}$. $||E_r||_{op} = \max_{x \neq 0} \frac{||E_r x||_2}{||x||_2} = \sigma_{r+1}$

d) Explain the conditions under which $\boldsymbol{X}_r$ will be a "good" approximation to $\boldsymbol{X}$.

We would like to minimize $||E_r||_{op}$, which will occur when the largest singular values are in the rank-r approximation, so the error is left with the smallest singular values.

2. Image compression. A digital image can be represented with a matrix, where each element of the matrix represents a pixel in the image. A low-rank approximation to the matrix is one way to compress the image, as explored in this problem. A data file contains a matrix $\boldsymbol{A} \in \mathbb{R}^{600 \times 400}$ of grayscale values scaled to lie between 0 and 1. A helper script loads the data and displays the corresponding image. There are three lines of code that require completion before you can run the code: one in the section labeled "Bucky's Singular Values" and two in the section labeled "Low-Rank Approximation".

a) Take the SVD of $\boldsymbol{A}$ by completing the code. Inspect the singular value spectrum. What do you conclude about the approximate rank of $\boldsymbol{A}$? Why is it useful to plot the logarithm of the singular values?

Rank of A is the number of non-zero singular values.
The Rank of A appears to be about 400, as the log of a negative number would be undefined.

Plotting the log makes the size of the singular value more apparant, as the singular values will get very small and close to Ø very fast, making them / the trend within them less obvious.

b) Approximate $\boldsymbol{A}$ as a rank r matrix $\boldsymbol{A}_r$ by only keeping the r largest singular values and making the rest zero. Try this for $r \in \{10, 20, 50, 100\}$ and plot the corresponding low-rank images. Also find the fractional squared error

$$e = \frac{||\boldsymbol{A} - \boldsymbol{A}_r||_F^2}{||\boldsymbol{A}||_F^2}$$

The quality of the approximation increases from having a lot of lines to just looking grainy.

Comment on the how the quality of the approximation changes as r increases.

c) Compare the space required to store the full $\boldsymbol{A}$ matrix with the space required to store the rank r SVD approximation of $\boldsymbol{A}$; how many times smaller is the storage requirement for $r \in \{10, 20, 50, 100\}$? You may assume that storage space requirements are proportional to the number of numbers that must be stored. e.g. a 10×10 matrix contains 100 numbers. Full A stores 600×400 nums = 240000

r=10 → 2400 times
r=20 → 600 times
r=50 → 96 times
r=100 → 24 times

r=10 → 10×10=100 num r=20 → 20×20 = 400num r=50 → 50×50 = 2500 num r=100 → 100×100= 10000num

d) Use the last section of the code to find the rank of the low-rank approximation that minimizes the sum of the bias squared and variance for a noisy version of Bucky. Note that since the "Bias-Variance Tradeoff in Low-Rank Approximations" lecture assumes an N-by-M matrix with $N < M$, we work with the transpose of $\boldsymbol{A}$ so that $M = 600$ and $N = 400$.

i. Assume the variance of each row $\sigma_g^2 = \sum_{j=1}^{M} g_{ij}^2$ in the "Bias-Variance Tradeoff in Low-Rank Approximations" lecture is $\sigma_g^2 = 10$. Best rank = 68

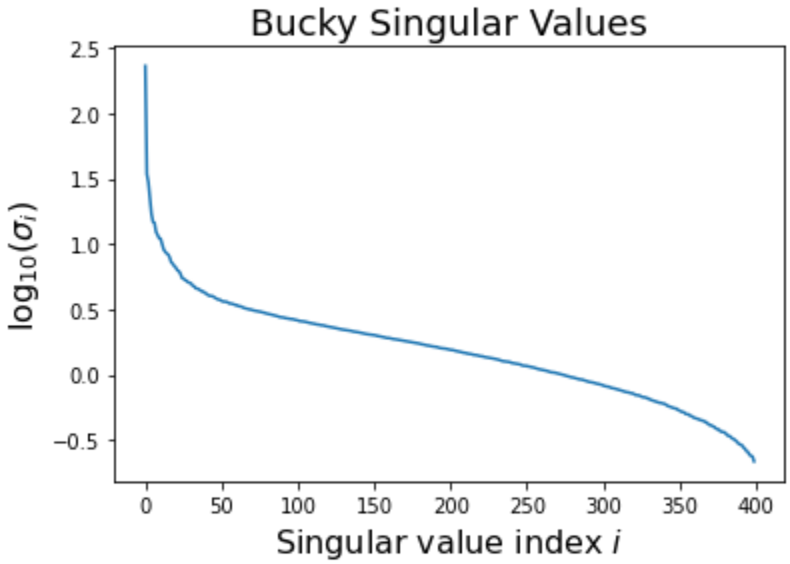
ii. Assume the variance of each row $\sigma_g^2 = \sum_{j=1}^{M} g_{ij}^2$ in the "Bias-Variance Tradeoff in Low-Rank Approximations" lecture is $\sigma_g^2 = 50$. Best rank = 18

e) Optional. Simulate the noisy case by performing low-rank approximations to a noisy version of $\boldsymbol{A}$. You may create this using the command `Anoise = A + np.sqrt(sigma2/600)* np.random.randn(np.shape(A)[0], np.shape(A)[1])` in Python, where `sigma2` corresponds to σ_g^2. Note that the division by $M = 600$ is necessary because `randn` creates random matrices where each element (not the row) has unit variance.

As the sigma2 in the noise calculation increases, the r∈{10, 20, 50, 100} images get less clear and the best rank for low rank approximations increases

```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Load data for activity
#
in_data = loadmat('bucky.mat')
A = in_data['A']

##

# Load data for activity: Another option
# A = imageio.imread("Whateveryoulike.png")
# A = np.average(A[:,:,0:3], axis=2)/256

rows, cols = np.array(A.shape)


# Display image
fig = plt.figure()
ax = fig.add_subplot(111)

ax.imshow(A,cmap='gray')
ax.set_axis_off()
plt.show()
```

```
# Bucky's singular values

# Complete and uncomment line below
U,s,VT = np.linalg.svd(A)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(np.log10(s))
ax.set_xlabel('Singular value index $i$', fontsize=16)
ax.set_ylabel('$\log_{10}(\sigma_i)$', fontsize=16)
ax.set_title('Bucky Singular Values', fontsize=18)
plt.show()
```




```
# Find and display low-rank approximations

r_vals = np.array([10, 20, 50, 100 ])
err_fro = np.zeros(len(r_vals))

# display images of various rank approximations
for i, r in enumerate(r_vals):

    # Complete and uncomment two lines below
    sing = np.pad(np.diag(s[:r]), ((0,len(U)-r),(0,len(s)-r)), mode='constant')
    Ar = U@sing@VT

    Er = A-Ar
    err_fro[i] = np.linalg.norm(Er,ord='fro')

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.imshow(Ar,cmap='gray',interpolation='none')
    ax.set_axis_off()
    ax.set_title(['Bucky Rank =', str(r_vals[i])], fontsize=18)
    plt.show()

# plot normalized error versus rank
norm_err = err_fro/np.linalg.norm(A,ord='fro')

fig = plt.figure()
ax = fig.add_subplot(111)
ax.stem(r_vals,norm_err)
ax.set_xlabel('Rank', fontsize=16)
ax.set_ylabel('Normalized error', fontsize=16)
plt.show()
```

['Bucky Rank =', '10']

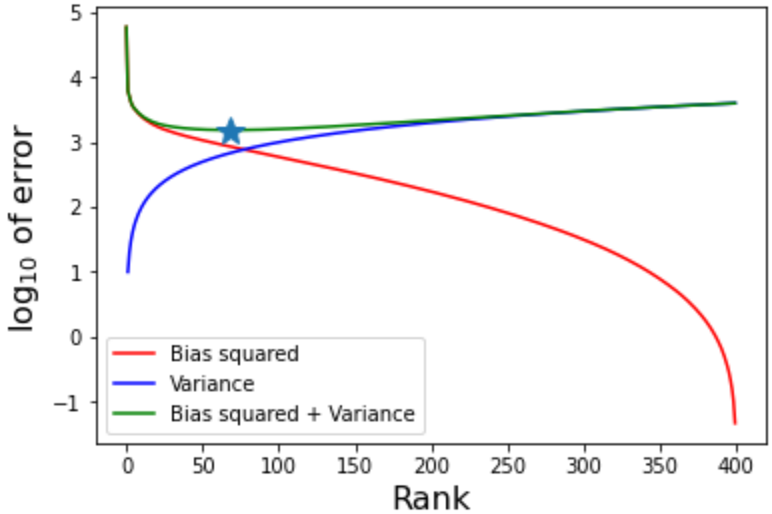
['Bucky Rank =', '20']

['Bucky Rank =', '50']

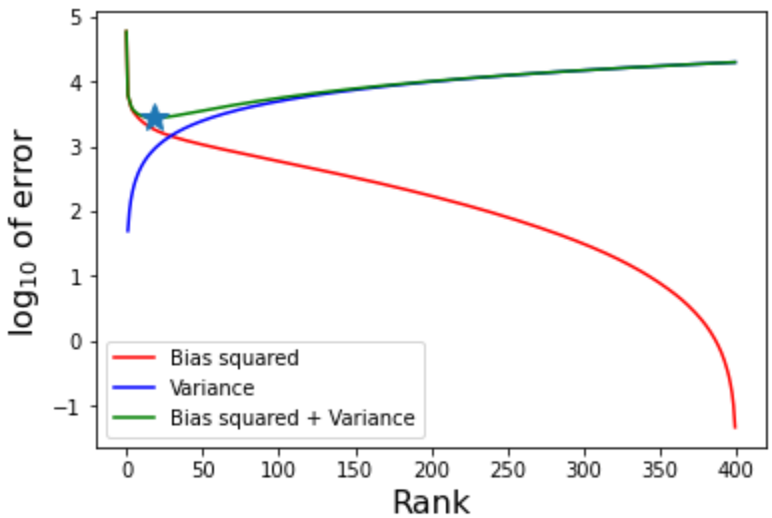
['Bucky Rank =', '100']


```
# bias-variance tradeoff
num_sv = min(rows, cols)
bias_2 = np.zeros(num_sv)
ranks = np.arange(num_sv)

for r in range(num_sv):
    bias_2[r] = np.linalg.norm(s[r:num_sv])**2

sigma2 = 10
var = sigma2*ranks
#print(var)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(ranks,np.log10(bias_2),'r',label='Bias squared')
ax.plot(ranks[1:],np.log10(var[1:]),'b', label = 'Variance')
ax.plot(ranks,np.log10(bias_2+var),'g', label='Bias squared + Variance')
min_bias_plus_variance_index = np.argmin(np.log10(bias_2+var))
ax.plot(ranks[min_bias_plus_variance_index], np.log10(bias_2+var)[min_bias_plus_variance_index
ax.set_xlabel('Rank', fontsize=16)
ax.set_ylabel('$\log_{10}$ of error', fontsize=16)
ax.legend()
plt.show()

print("Best r for variance of 10:", ranks[min_bias_plus_variance_index])



sigma2 = 50
var = sigma2*ranks
#print(var)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(ranks,np.log10(bias_2),'r',label='Bias squared')
ax.plot(ranks[1:],np.log10(var[1:]),'b', label = 'Variance')
ax.plot(ranks,np.log10(bias_2+var),'g', label='Bias squared + Variance')
min_bias_plus_variance_index = np.argmin(np.log10(bias_2+var))
ax.plot(ranks[min_bias_plus_variance_index], np.log10(bias_2+var)[min_bias_plus_variance_index
ax.set_xlabel('Rank', fontsize=16)
ax.set_ylabel('$\log_{10}$ of error', fontsize=16)
ax.legend()
plt.show()
```

```
print("Best·r·for·variance·of·50:",·ranks[min_bias_plus_variance_index])
```




```
Best r for variance of 10: 68
```



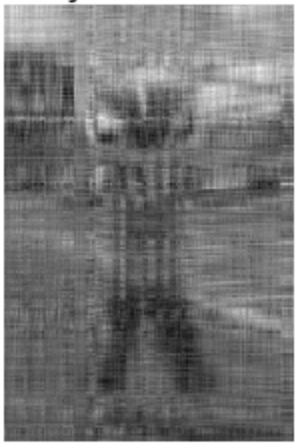

```
Best r for variance of 50: 18
```

Optional Part C

```
sigma2 = 50 # as sigma increases, the bucky picture gets less clear

Anoise = A + np.sqrt(sigma2/600)*np.random.randn(np.shape(A)[0],np.shape(A)[1])
U,s,VT = np.linalg.svd(Anoise)

# Find and display low-rank approximations

r_vals = np.array([10, 20, 50, 100 ])
err_fro = np.zeros(len(r_vals))

# display images of various rank approximations
for i, r in enumerate(r_vals):

    # Complete and uncomment two lines below
    sing = np.pad(np.diag(s[:r]), ((0,len(U)-r),(0,len(s)-r)), mode='constant')
    Ar = U@sing@VT
```

```
    Er = Anoise-Ar
    err_fro[i] = np.linalg.norm(Er,ord='fro')

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.imshow(Ar,cmap='gray',interpolation='none')
    ax.set_axis_off()
    ax.set_title(['Bucky Rank =', str(r_vals[i])], fontsize=18)
    plt.show()

# plot normalized error versus rank
norm_err = err_fro/np.linalg.norm(Anoise,ord='fro')

fig = plt.figure()
ax = fig.add_subplot(111)
ax.stem(r_vals,norm_err)
ax.set_xlabel('Rank', fontsize=16)
ax.set_ylabel('Normalized error', fontsize=16)
plt.show()
```

['Bucky Rank =', '10']

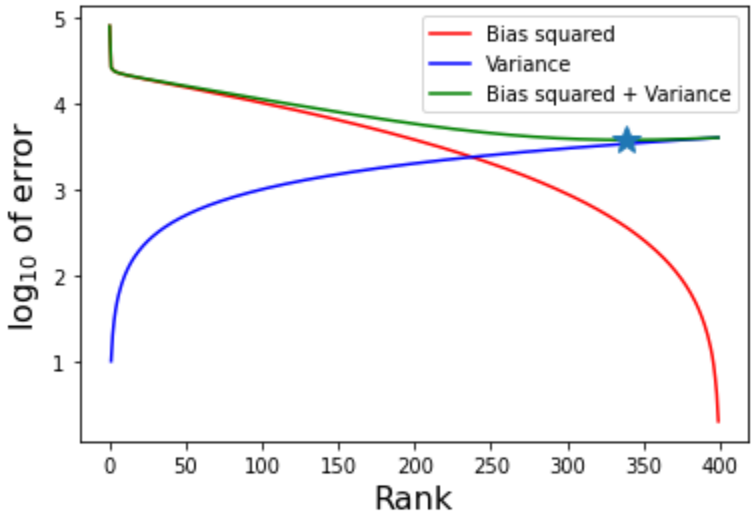


['Bucky Rank =', '20']

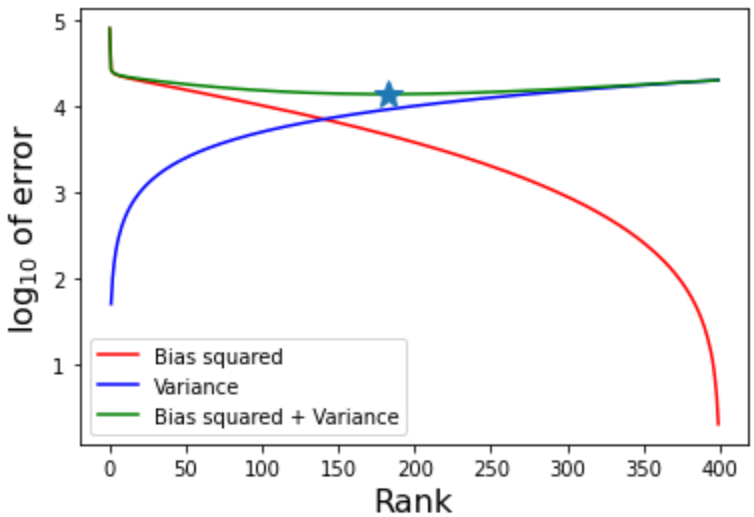
['Bucky Rank =', '50']

```
# bias-variance tradeoff
num_sv = min(rows, cols)
bias_2 = np.zeros(num_sv)
ranks = np.arange(num_sv)

for r in range(num_sv):
    bias_2[r] = np.linalg.norm(s[r:num_sv])**2

sigma2 = 10
var = sigma2*ranks
#print(var)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(ranks,np.log10(bias_2),'r',label='Bias squared')
ax.plot(ranks[1:],np.log10(var[1:]),'b', label = 'Variance')
```

```
ax.plot(ranks,np.log10(bias_2+var),'g', label='Bias squared + Variance')
min_bias_plus_variance_index = np.argmin(np.log10(bias_2+var))
ax.plot(ranks[min_bias_plus_variance_index], np.log10(bias_2+var)[min_bias_plus_variance_index
ax.set_xlabel('Rank', fontsize=16)
ax.set_ylabel('$\log_{10}$ of error', fontsize=16)
ax.legend()
plt.show()

print("Best r for variance of 10:", ranks[min_bias_plus_variance_index])


sigma2 = 50
var = sigma2*ranks
#print(var)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(ranks,np.log10(bias_2),'r',label='Bias squared')
ax.plot(ranks[1:],np.log10(var[1:]),'b', label = 'Variance')
ax.plot(ranks,np.log10(bias_2+var),'g', label='Bias squared + Variance')
min_bias_plus_variance_index = np.argmin(np.log10(bias_2+var))
ax.plot(ranks[min_bias_plus_variance_index], np.log10(bias_2+var)[min_bias_plus_variance_index
ax.set_xlabel('Rank', fontsize=16)
ax.set_ylabel('$\log_{10}$ of error', fontsize=16)
ax.legend()
plt.show()

print("Best r for variance of 50:", ranks[min_bias_plus_variance_index])
```




```
Best r for variance of 10: 339
```




```
Best r for variance of 50: 182
```