worksheet_10

March 3, 2024

1 Worksheet 10

Name: Youxuan Ma

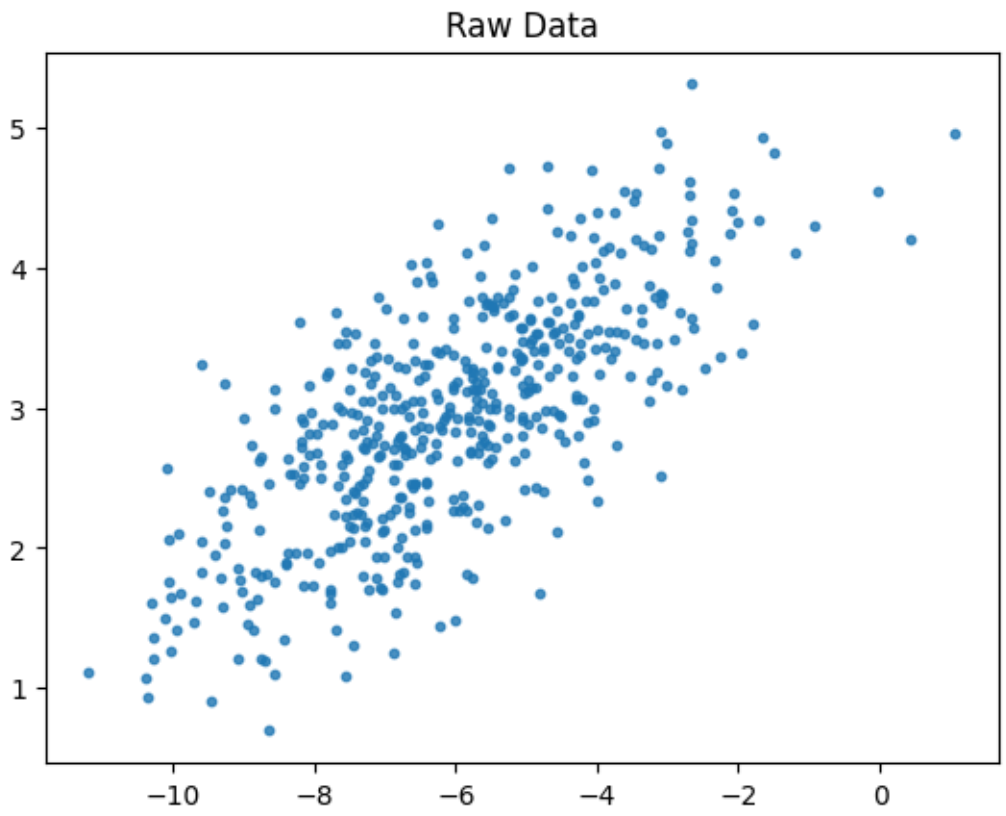
UID: U23330522

1.0.1 Topics

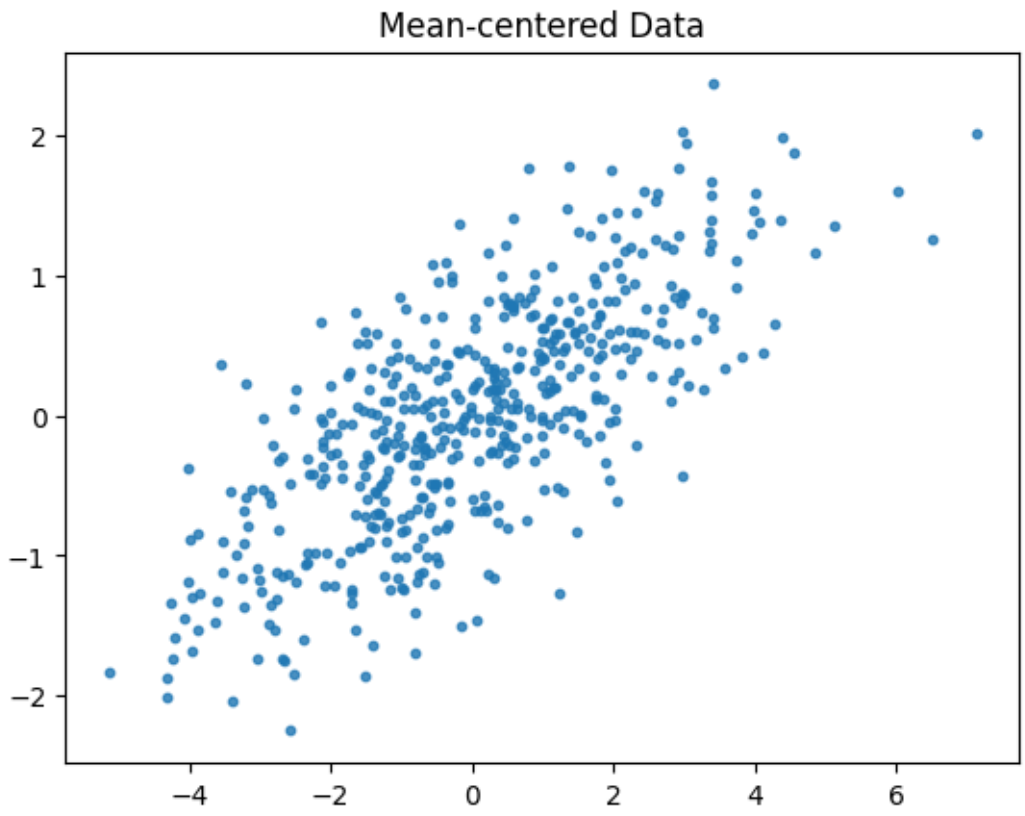
- Singular Value Decomposition

Feature Extraction SVD finds features that are orthogonal. The Singular Values correspond to the importance of the feature or how much variance in the data it captures.

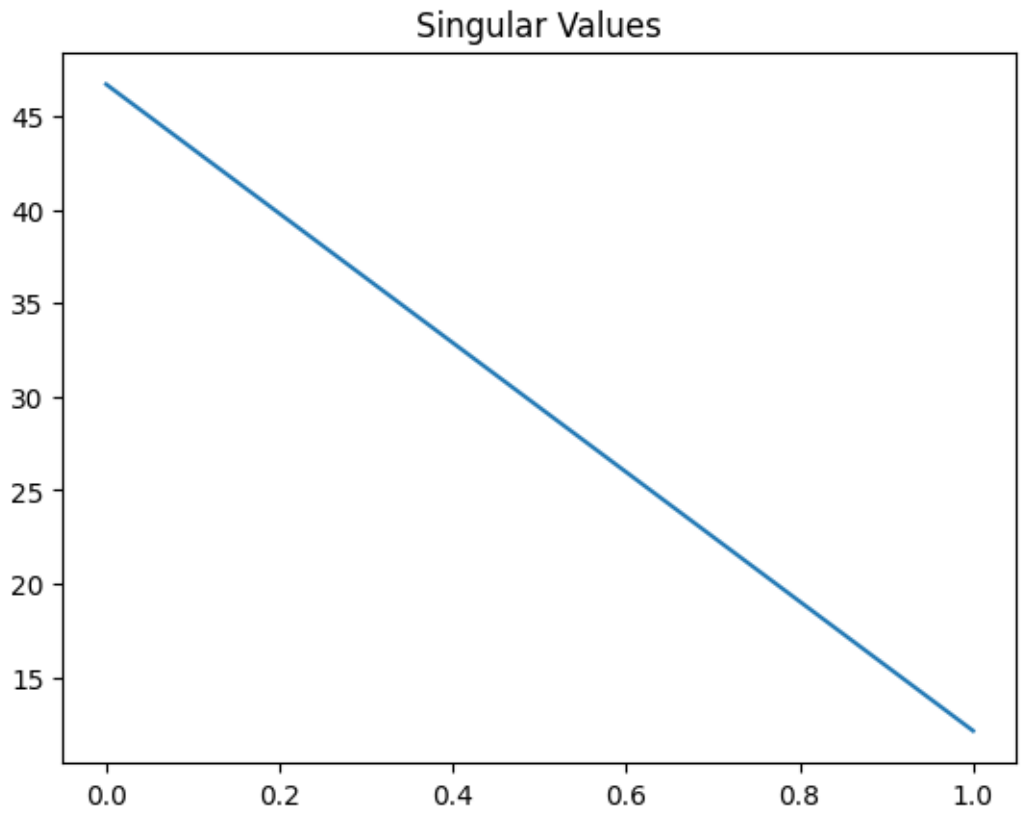
[1]:
```
import numpy as np
import matplotlib.pyplot as plt

n_samples = 500
C = np.array([[0.1, 0.6], [2., .6]])
X = np.random.randn(n_samples, 2) @ C + np.array([-6, 3])
plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.8)
plt.title("Raw Data")
plt.show()
```



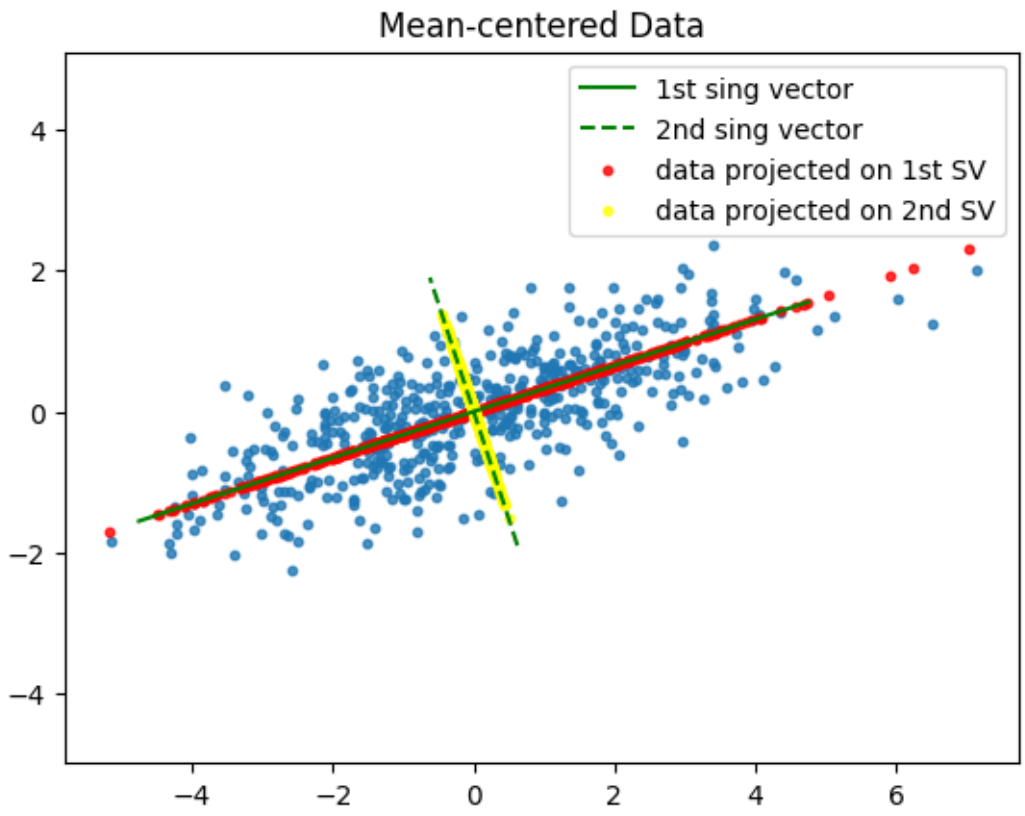
[2]:
```
X = X - np.mean(X, axis=0)
plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.8)
plt.title("Mean-centered Data")
plt.show()
```



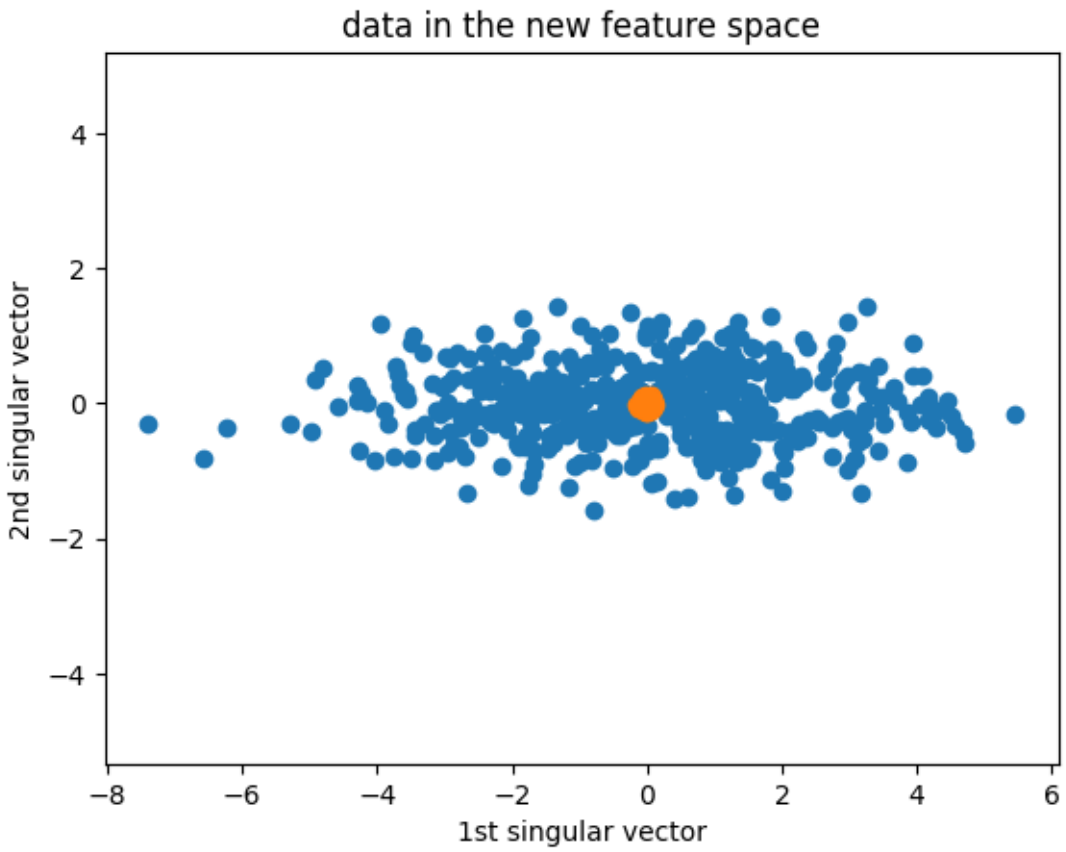
[3]:
```
u,s,vt=np.linalg.svd(X, full_matrices=False)
plt.plot(s) # only 2 singular values
plt.title("Singular Values")
plt.show()
```



[4]:
```
scopy0 = s.copy()
scopy1 = s.copy()
scopy0[1:] = 0.0
scopy1[:1] = 0.0
approx0 = u.dot(np.diag(scopy0)).dot(vt)
approx1 = u.dot(np.diag(scopy1)).dot(vt)
plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.8)
sv1 = np.array([[-5],[5]]) @ vt[[0],:]
sv2 = np.array([[-2],[2]]) @ vt[[1],:]
plt.plot(sv1[:,0], sv1[:,1], 'g-', label="1st sing vector")
plt.plot(sv2[:,0], sv2[:,1], 'g--', label="2nd sing vector")
plt.scatter(approx0[:, 0] , approx0[:, 1], s=10, alpha=0.8, color="red",␣
 ↪label="data projected on 1st SV")
plt.scatter(approx1[:, 0] , approx1[:, 1], s=10, alpha=0.8, color="yellow",␣
 ↪label="data projected on 2nd SV")
plt.axis('equal')
plt.legend()
plt.title("Mean-centered Data")
plt.show()
```

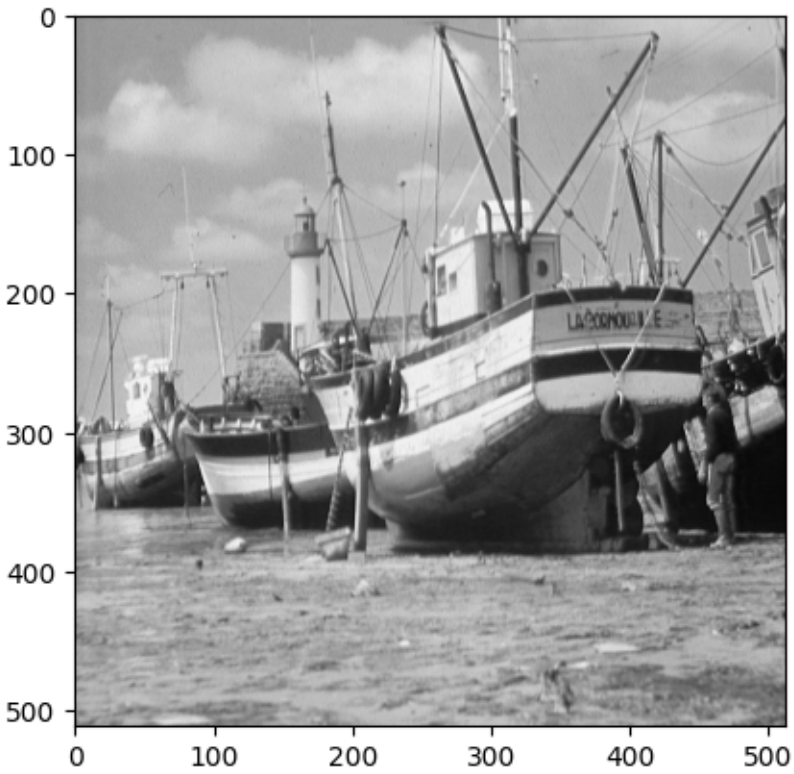


[5]:
```
# show ouput from svd is the same
orthonormal_X = u
shifted_X = u.dot(np.diag(s))
plt.axis('equal')
plt.scatter(shifted_X[:,0], shifted_X[:,1])
plt.scatter(orthonormal_X[:,0], orthonormal_X[:,1])
plt.xlabel("1st singular vector")
plt.ylabel("2nd singular vector")
plt.title("data in the new feature space")
plt.show()
```



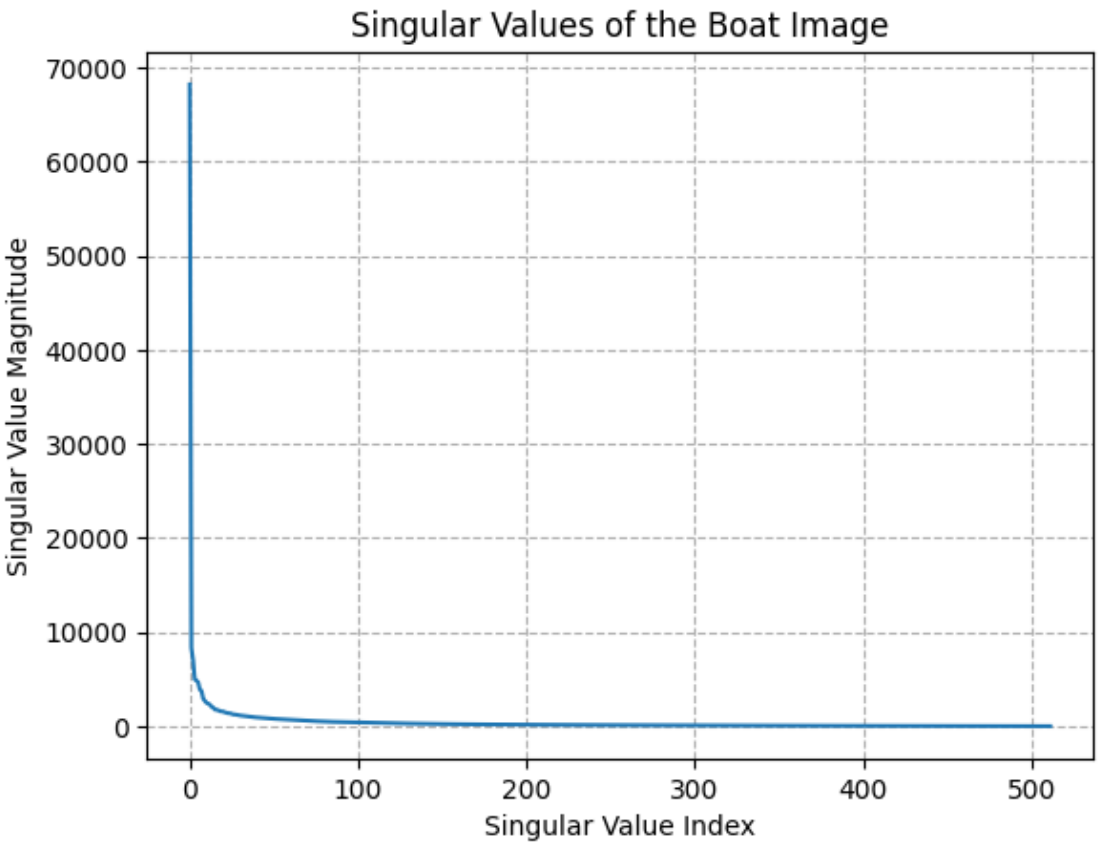
[6]:
```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

boat = np.loadtxt('./boat.dat')
plt.figure()
plt.imshow(boat, cmap = cm.Greys_r)
```

[6]: <matplotlib.image.AxesImage at 0x11c4bfef0>



a) Plot the singular values of the image above (note: a gray scale image is just a matrix).

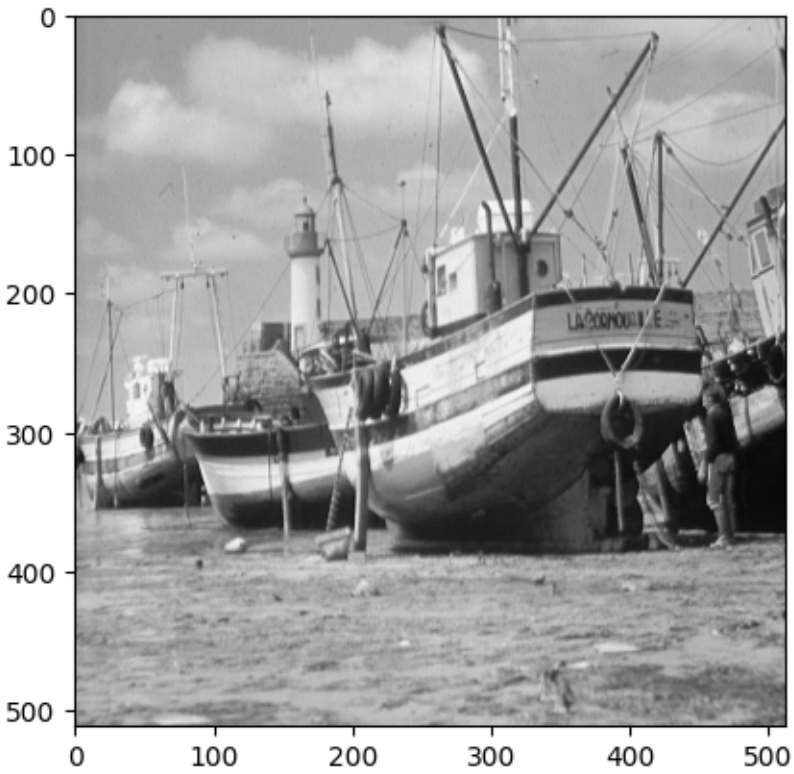
[7]:
```
u,s,vt=np.linalg.svd(boat,full_matrices=False)
plt.figure()
plt.plot(s)
plt.title('Singular Values of the Boat Image')
plt.xlabel('Singular Value Index')
plt.ylabel('Singular Value Magnitude')
plt.grid(True, which="both", ls="--")
plt.show()
```



Notice you can get the image back by multiplying the matrices back together:

[8]:
```
boat_copy = u.dot(np.diag(s)).dot(vt)
plt.figure()
plt.imshow(boat_copy, cmap = cm.Greys_r)
```

[8]:
```
<matplotlib.image.AxesImage at 0x11c5375c0>
```

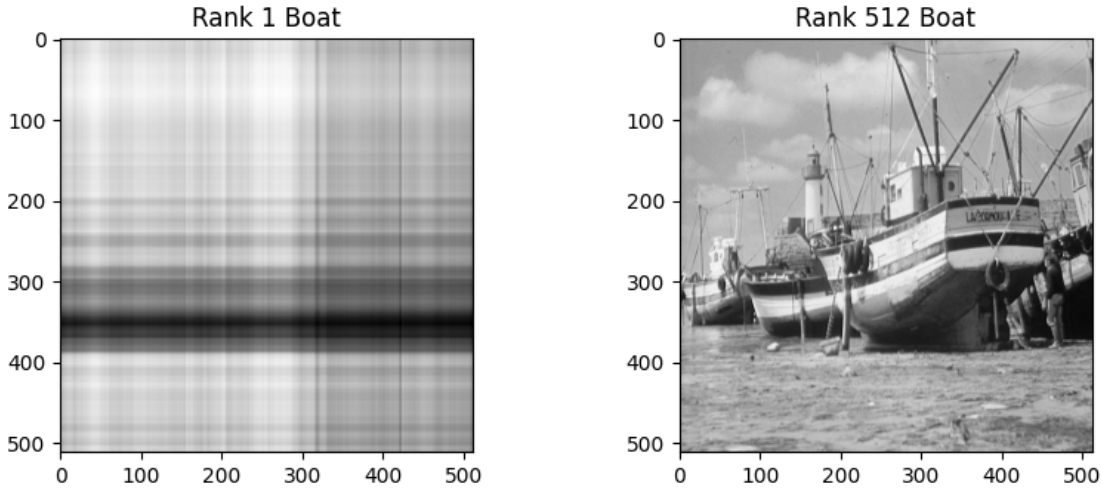


b) Create a new matrix `scopy` which is a copy of `s` with all but the first singular value set to 0.

[9]:
```
scopy = s.copy()
scopy[1:] = 0.0
```

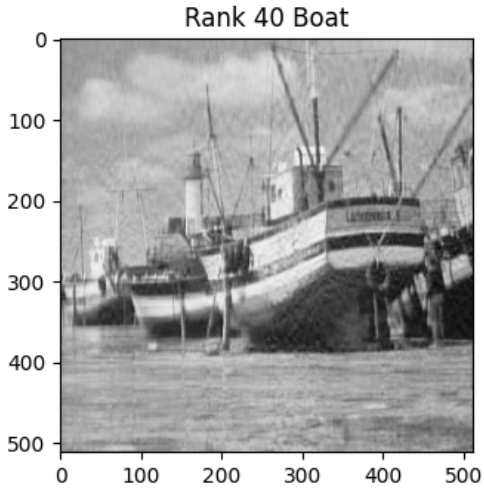
c) Create an approximation of the boat image by multiplying `u`, `scopy`, and `v` transpose. Plot them side by side.

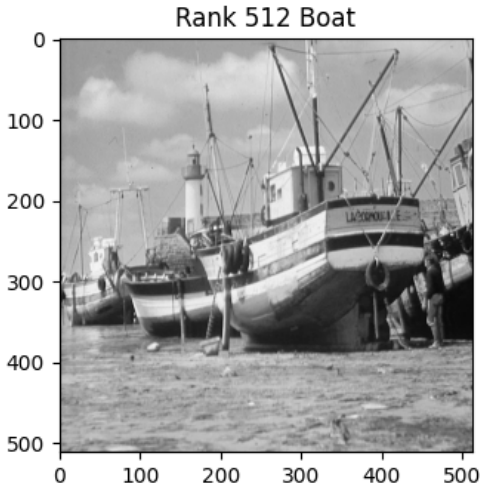
[10]:
```
boat_app = u.dot(np.diag(scopy)).dot(vt)

plt.figure(figsize=(9,6))
plt.subplot(1,2,1)
plt.imshow(boat_app, cmap = cm.Greys_r)
plt.title('Rank 1 Boat')
plt.subplot(1,2,2)
plt.imshow(boat, cmap = cm.Greys_r)
plt.title('Rank 512 Boat')
_ = plt.subplots_adjust(wspace=0.5)
plt.show()
```



d) Repeat c) with 40 singular values instead of just 1.

[11]:
```
scopy2 = s.copy()
scopy2[40:] = 0.0

boat_app2 = u.dot(np.diag(scopy2)).dot(vt)

plt.figure(figsize=(9,6))
plt.subplot(1,2,1)
plt.imshow(boat_app2, cmap = cm.Greys_r)
plt.title('Rank 40 Boat')
plt.subplot(1,2,2)
plt.imshow(boat, cmap = cm.Greys_r)
plt.title('Rank 512 Boat')
_ = plt.subplots_adjust(wspace=0.5)
plt.show()
```





1.0.2 Why you should care

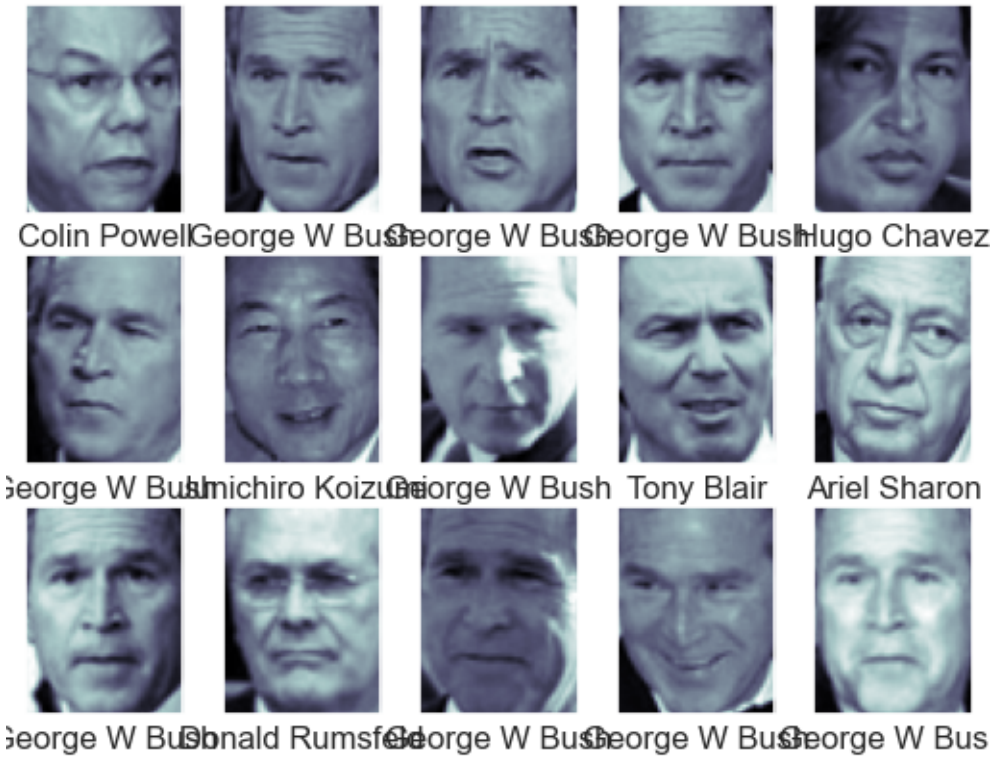
a) By using an approximation of the data, you can improve the performance of classification tasks since:

1. there is less noise interfering with classification
2. no relationship between features after SVD
3. the algorithm is sped up when reducing the dimension of the dataset

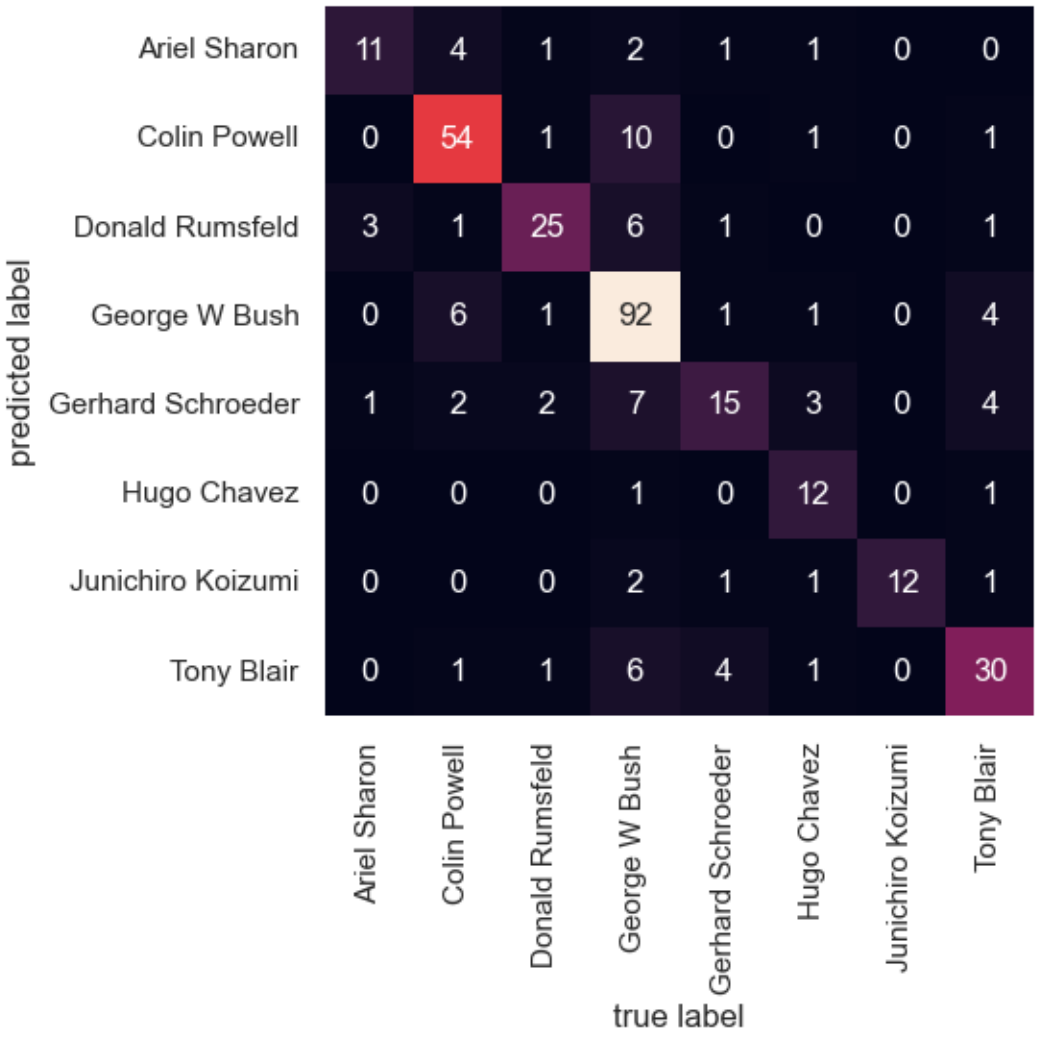
Below is some code to perform facial recognition on a dataset. Notice that, applied blindly, it does not perform well:

[12]:
```
import numpy as np
from PIL import Image
import seaborn as sns
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.datasets import fetch_lfw_people
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV, train_test_split

sns.set()

# Get face data
faces = fetch_lfw_people(min_faces_per_person=60)

# plot face data
fig, ax = plt.subplots(3, 5)
```

```
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel=faces.target_names[faces.target[i]])
plt.show()

# split train test set
Xtrain, Xtest, ytrain, ytest = train_test_split(faces.data, faces.target,␣
 ↪random_state=42)

# blindly fit svm
svc = SVC(kernel='rbf', class_weight='balanced', C=5, gamma=0.001)

# fit model
model = svc.fit(Xtrain, ytrain)
yfit = model.predict(Xtest)

fig, ax = plt.subplots(6, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14)
plt.show()

mat = confusion_matrix(ytest, yfit)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()

print("Accuracy = ", accuracy_score(ytest, yfit))
```


Colin Powell
George W Bush
George W Bush
George W Bush
Hugo Chavez
George W Bush
Junichiro Koizumi
George W Bush
Tony Blair
Ariel Sharon
George W Bush
Donald Rumsfeld
George W Bush
George W Bush
George W Bus


Predicted Names; Incorrect Labels in Red
Bush
Bush
Schroeder
Bush
Sharon
Sharon
Bush
Bush
Blair
Rumsfeld
Bush
Koizumi
Koizumi
Bush
Bush
Bush
Blair
Koizumi
Bush
Bush
Bush
Bush
Powell
Blair
Blair
Bush
Koizumi
Bush
Blair
Chavez
Bush
Rumsfeld
Powell
Bush
Rumsfeld
Schroeder
Bush
Bush

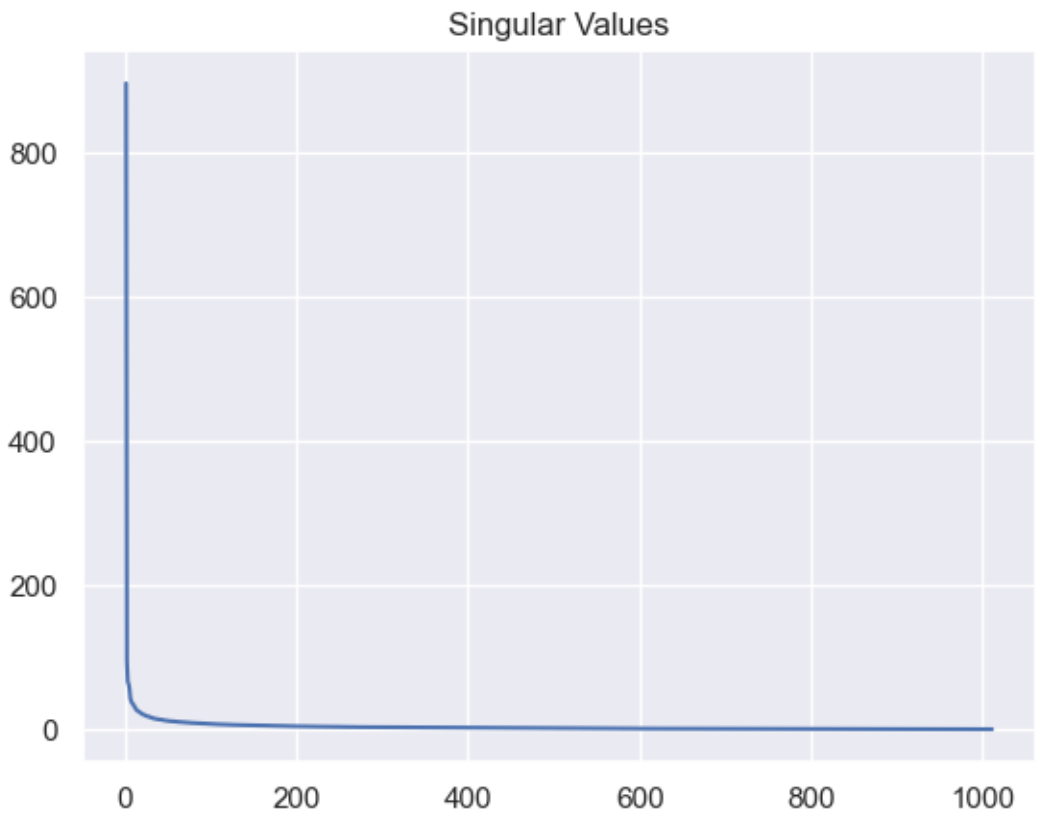



```
Accuracy =  0.744807121661721
```

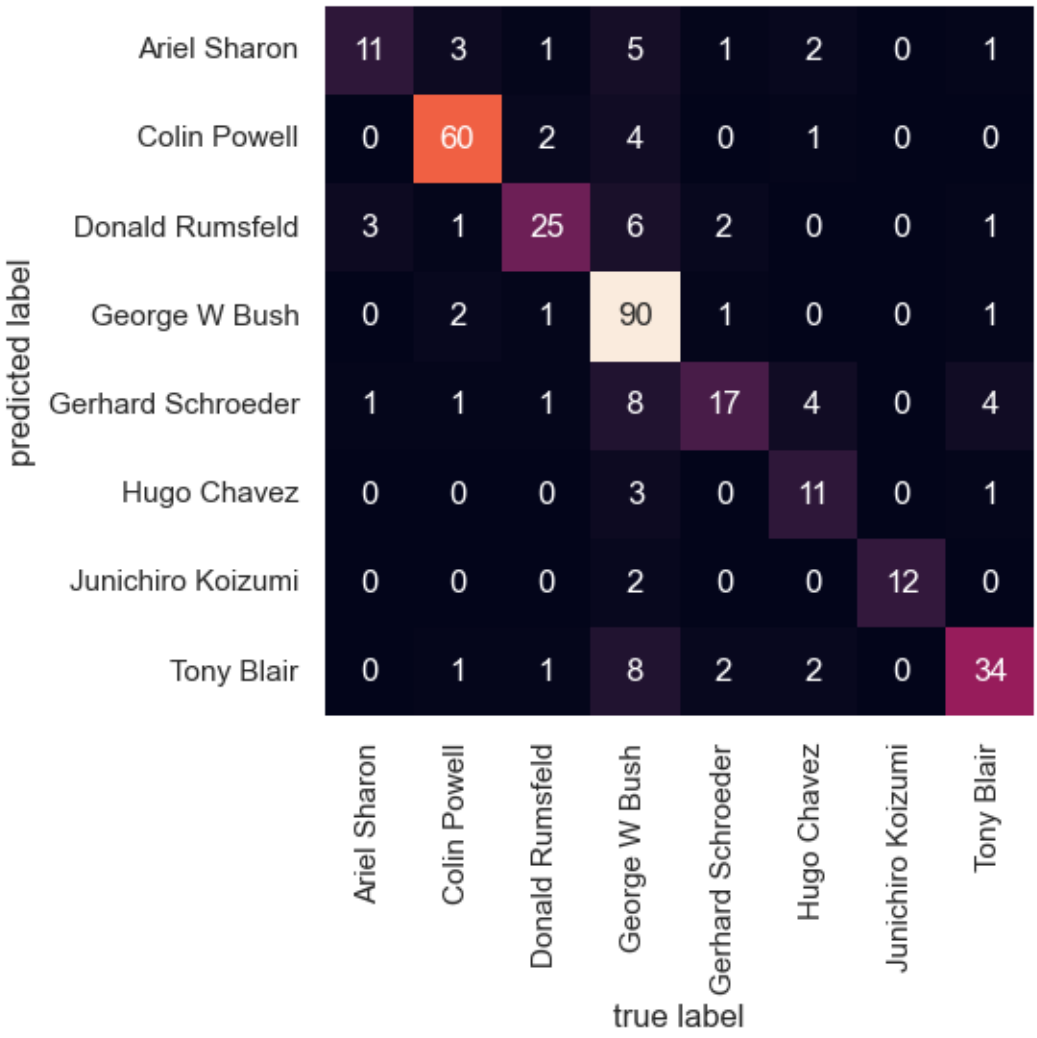
By performing SVD before applying the classification tool, we can reduce the dimension of the dataset.

[13]:
```
# look at singular values
_, s, _ = np.linalg.svd(Xtrain, full_matrices=False)
plt.plot(range(1,len(s)+1),s)
plt.title("Singular Values")
plt.show()

# extract principal components
pca = PCA(n_components=50, whiten=True)
svc = SVC(kernel='rbf', class_weight='balanced', C=5, gamma=0.001)
svcpca = make_pipeline(pca, svc)
```

```
model = svcpca.fit(Xtrain, ytrain)
yfit = model.predict(Xtest)

fig, ax = plt.subplots(6, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14)
plt.show()

mat = confusion_matrix(ytest, yfit)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()

print("Accuracy = ", accuracy_score(ytest, yfit))
```




Predicted Names; Incorrect Labels in Red
Bush
Koizumi
Schroeder
Bush
Sharon
Sharon
Bush
Rumsfeld
Bush
Koizumi
Bush
Bush
Bush
Bush
Rumsfeld
Bush
Bush
Schroeder
Blair
Koizumi
Bush
Koizumi
Powell
Blair
Blair
Bush
Bush
Powell
Blair
Chavez
Rumsfeld
Powell
Bush
Rumsfeld
Chavez
Bush
Bush




```
Accuracy =  0.771513353115727
```

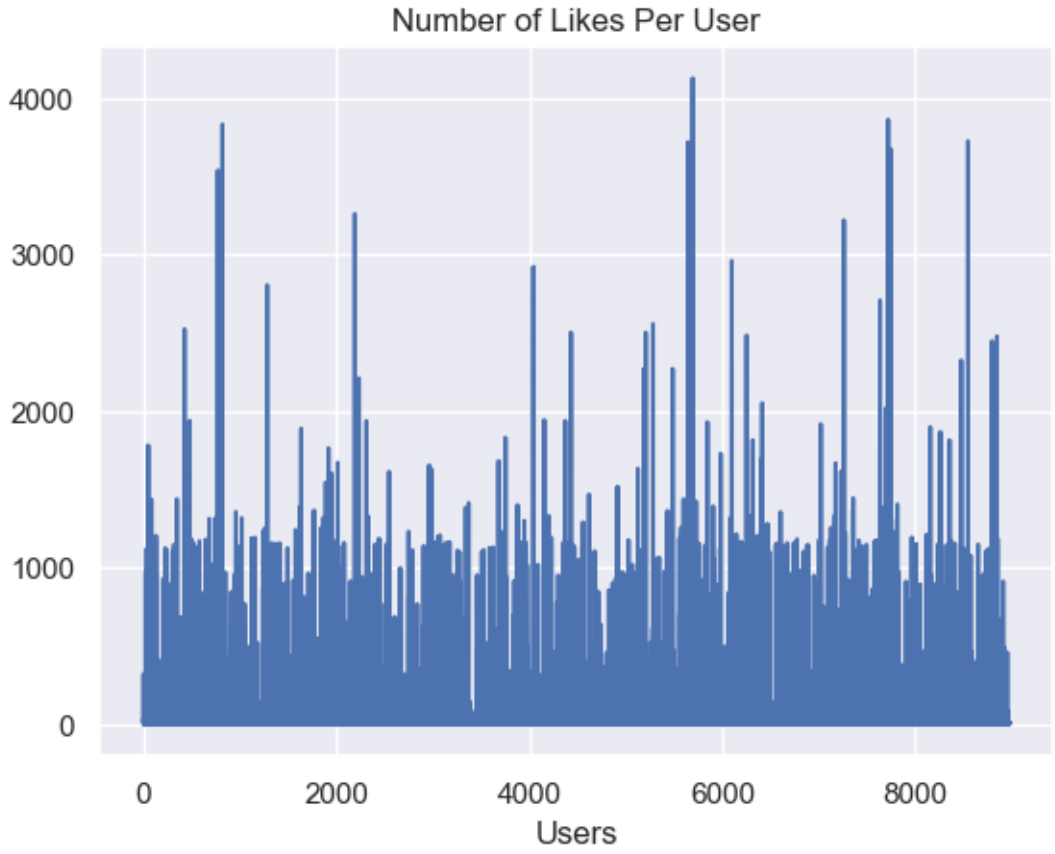
Similar to finding k in K-means, we're trying to find the point of diminishing returns when picking the number of singular vectors (also called principal components).

b) SVD can be used for anomaly detection.

The data below consists of the number of 'Likes' during a six month period, for each of 9000 users across the 210 content categories that Facebook assigns to pages.

[14]:
```
data = np.loadtxt('spatial_data.txt')

FBSpatial = data[:,1:]
FBSnorm = np.linalg.norm(FBSpatial,axis=1,ord=1)
plt.plot(FBSnorm)
plt.title('Number of Likes Per User')
```

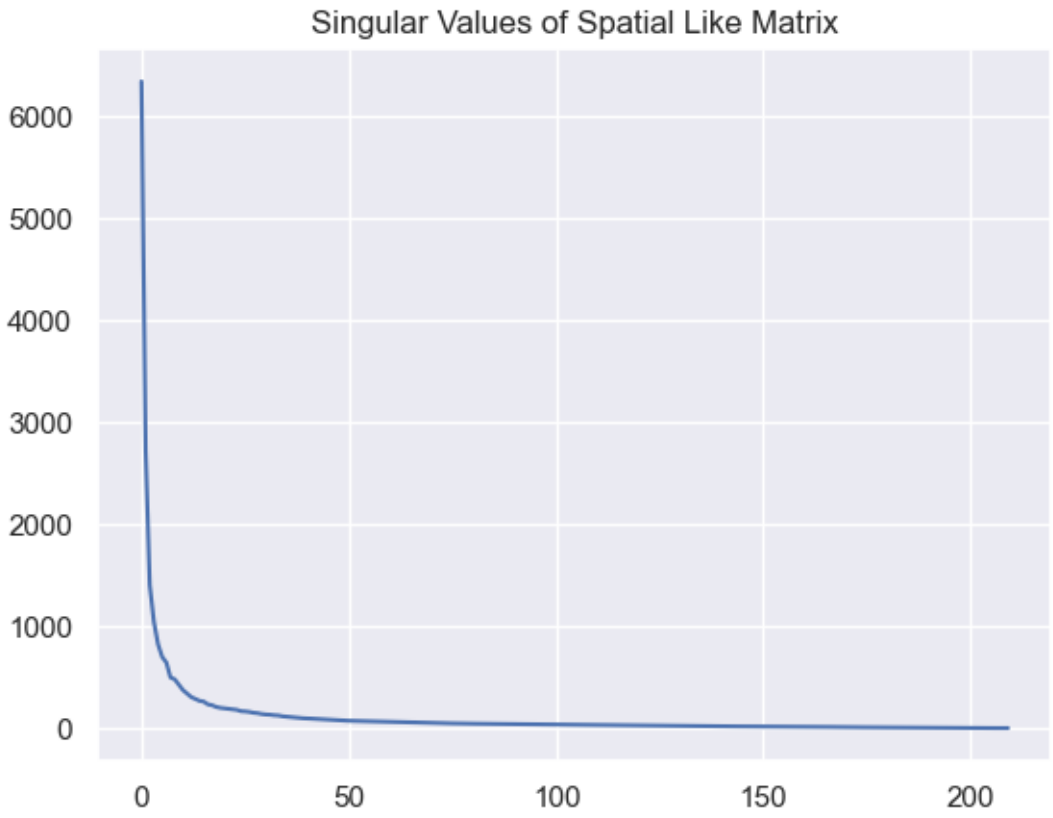
```
_ = plt.xlabel('Users')
plt.show()
```

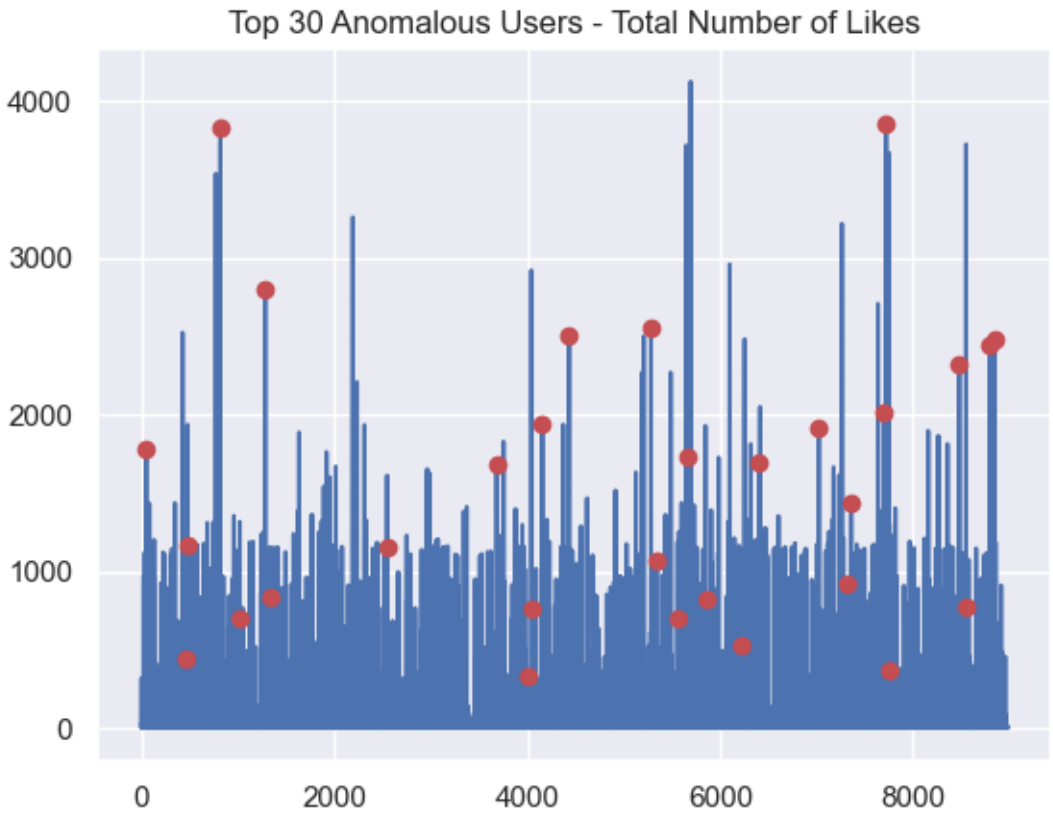


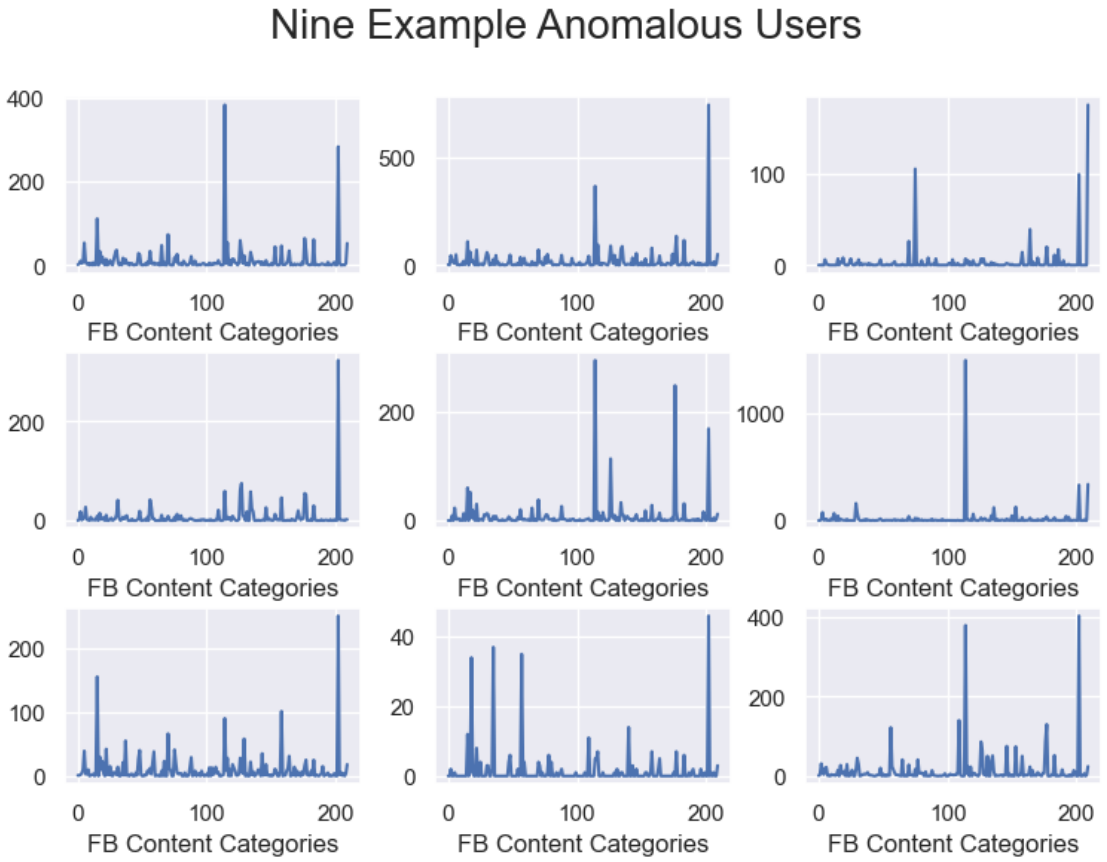
How users distribute likes across categories follows a general pattern that most users follow. This behavior can be captured using few singular vectors. And anomalous users can be easily identified.

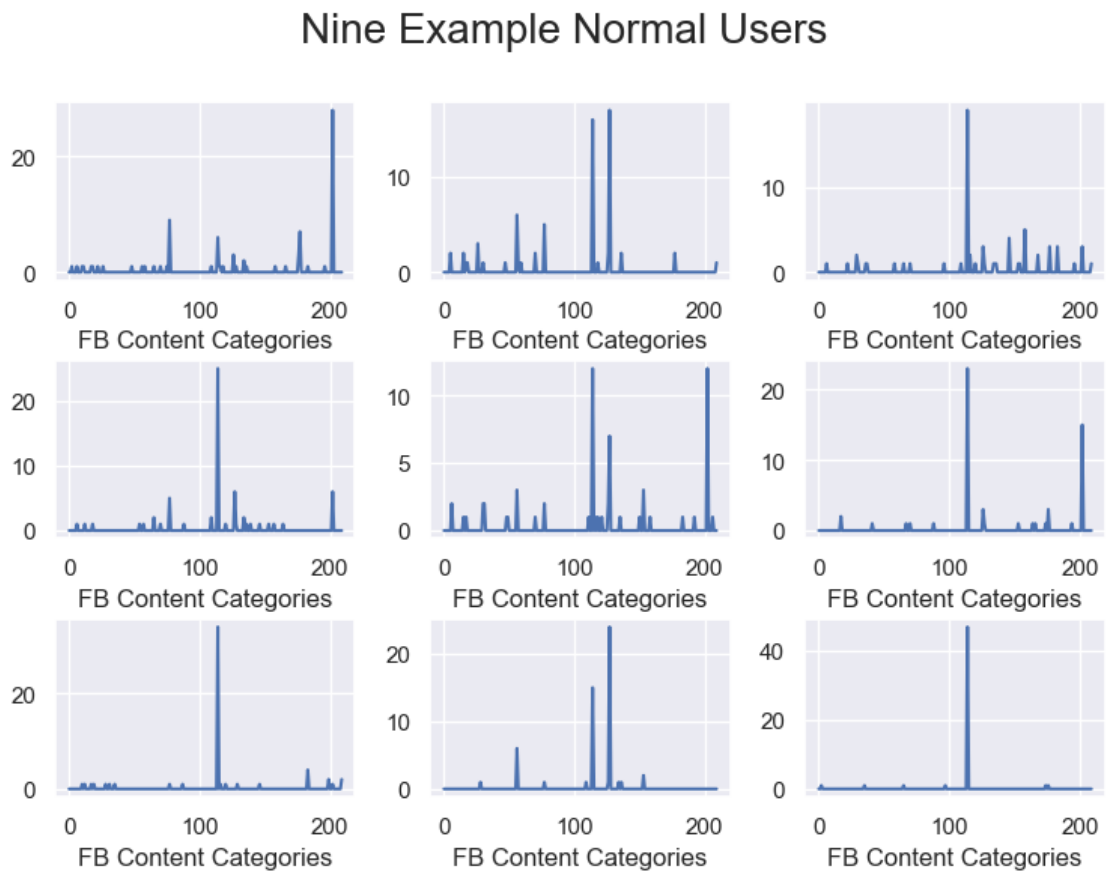
[15]:
```
u,s,vt = np.linalg.svd(FBSpatial,full_matrices=False)
plt.plot(s)
_ = plt.title('Singular Values of Spatial Like Matrix')
plt.show()

RANK = 20
scopy = s.copy()
scopy[RANK:] = 0.
N = u @ np.diag(scopy) @ vt
O = FBSpatial - N
Onorm = np.linalg.norm(O, axis=1)
anomSet = np.argsort(Onorm)[-30:]
# plt.plot(Onorm)
# plt.plot(anomSet, Onorm[anomSet],'ro')
```

```
# _ = plt.title('Norm of Residual (rows of O)')
# plt.show()

plt.plot(FBSnorm)
plt.plot(anomSet, FBSnorm[anomSet],'ro')
_ = plt.title('Top 30 Anomalous Users - Total Number of Likes')
plt.show()

# anomalous users
plt.figure(figsize=(9,6))
for i in range(1,10):
    ax = plt.subplot(3,3,i)
    plt.plot(FBSpatial[anomSet[i-1],:])
    plt.xlabel('FB Content Categories')
plt.subplots_adjust(wspace=0.25,hspace=0.45)
_ = plt.suptitle('Nine Example Anomalous Users',size=20)
plt.show()

# normal users
set = np.argsort(Onorm)[0:7000]
# that have high overall volume
max = np.argsort(FBSnorm[set])[::-1]
plt.figure(figsize=(9,6))
for i in range(1,10):
    ax = plt.subplot(3,3,i)
    plt.plot(FBSpatial[set[max[i-1]],:])
    plt.xlabel('FB Content Categories')
plt.subplots_adjust(wspace=0.25,hspace=0.45)
_ = plt.suptitle('Nine Example Normal Users',size=20)
plt.show()
```


Singular Values of Spatial Like Matrix
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200


Top 30 Anomalous Users - Total Number of Likes
4000
3000
2000
1000
0
0
2000
4000
6000
8000

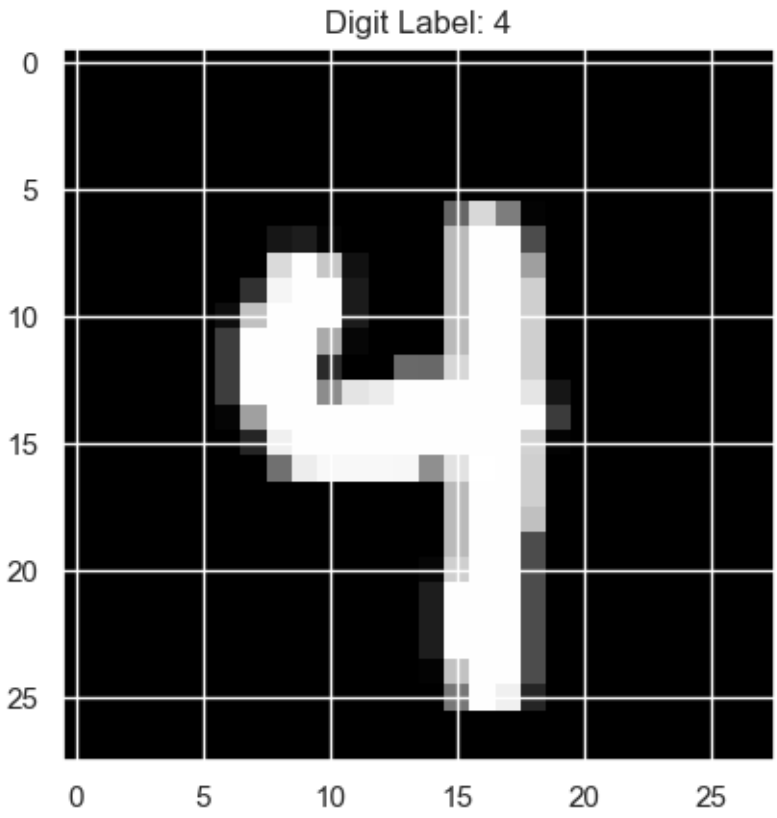

Nine Example Anomalous Users
400
200
0
500
0
100
0
0
100
200
FB Content Categories
200
0
200
0
1000
0
200
100
0
40
20
0
400
200
0



1.1 Challenge Problem

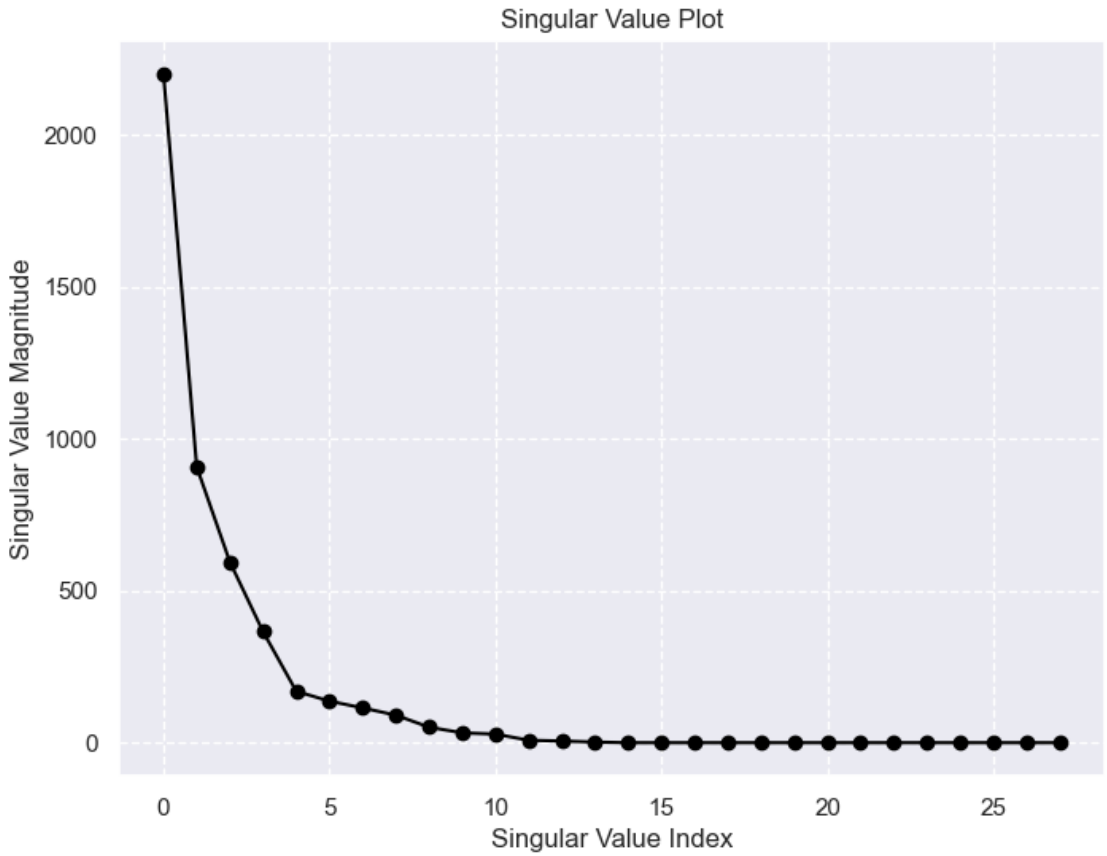
a) Fetch the "mnist_784" data. Pick an image of a digit at random and plot it.

[16]:
```
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml

X, y = fetch_openml(name="mnist_784", version=1, return_X_y=True,␣
 ↪as_frame=False)

# your code here
random_index = np.random.randint(0, X.shape[0])

image_data = X[random_index].reshape(28, 28)

plt.imshow(image_data, cmap = cm.Greys_r)
plt.title(f"Digit Label: {y[random_index]}")
plt.show()
```



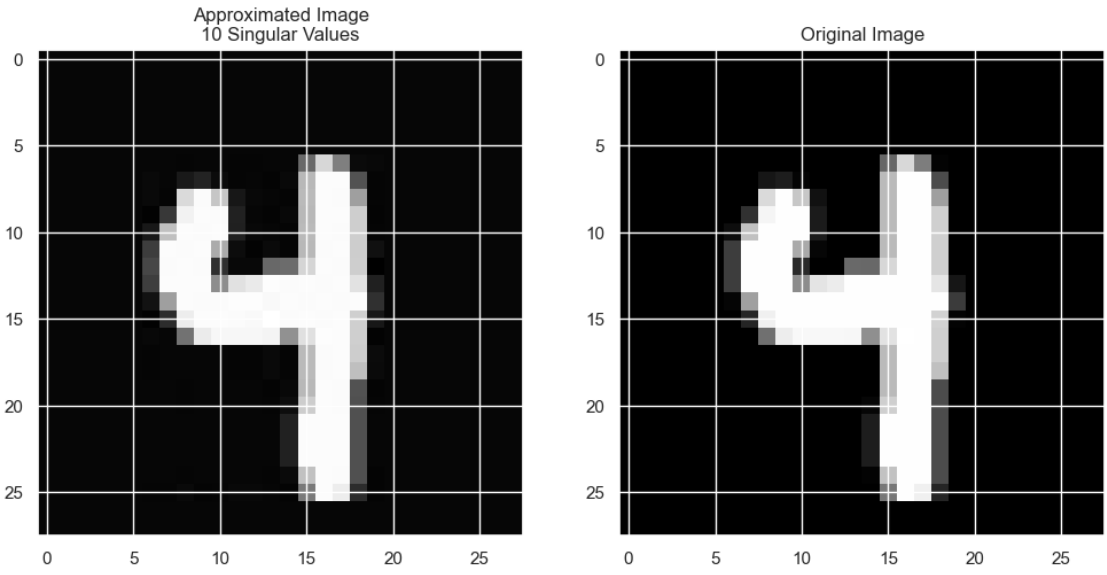
b) Plot its singular value plot.

[17]:
```
U, S, V = np.linalg.svd(image_data, full_matrices=False)

plt.figure(figsize=(8, 6))
plt.plot(S, '-o', color='black')
plt.title('Singular Value Plot')
plt.xlabel('Singular Value Index')
plt.ylabel('Singular Value Magnitude')
plt.grid(True, which="both", ls="--")
plt.show()
```



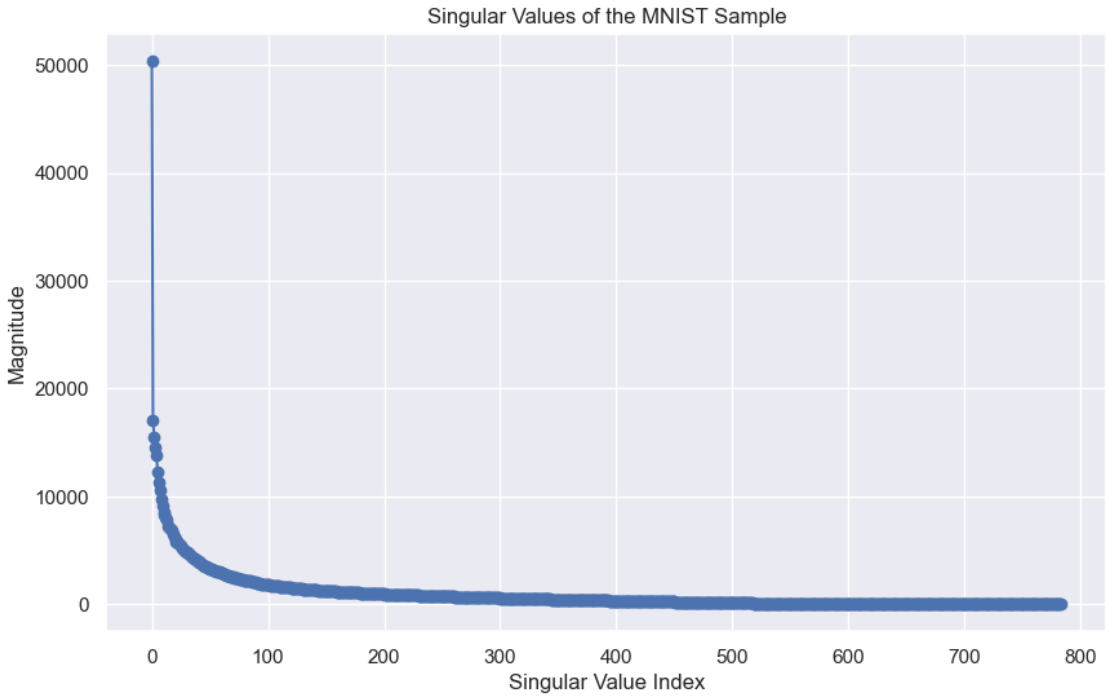
c) By setting some singular values to 0, plot the approximation of the image next to the original image

[18]:
```
k = 10
S_approx = np.zeros(S.shape)
S_approx[:k] = S[:k]

image_approx = U.dot(np.diag(S_approx)).dot(V)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 2)
plt.imshow(image_data, cmap=cm.Greys_r)
plt.title('Original Image')

plt.subplot(1, 2, 1)
plt.imshow(image_approx, cmap=cm.Greys_r)
plt.title(f'Approximated Image\n{k} Singular Values')

plt.show()
```



d) Consider the entire dataset as a matrix. Perform SVD and explain why / how you chose a particular rank. Note: you may not be able to run this on the entire dataset in a reasonable amount of time so you may take a small random sample for this and the following questions.

[19]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.utils import resample

X, y = fetch_openml(name="mnist_784", version=1, return_X_y=True,␣
 ↪as_frame=False)
sample_X, sample_y = resample(X, y, n_samples=1000, random_state=42)

U, S, V = np.linalg.svd(sample_X, full_matrices=False)

plt.figure(figsize=(10, 6))
plt.plot(S, '-o')
plt.title('Singular Values of the MNIST Sample')
plt.xlabel('Singular Value Index')
plt.ylabel('Magnitude')
plt.grid(True)
plt.show()

energy_total = np.sum(S**2)
energy_cumulative = np.cumsum(S**2) / energy_total
rank_choice = np.argmax(energy_cumulative >= 0.9) + 1

print(f"Chosen rank to capture 90% of the energy: {rank_choice}")
```




```
Chosen rank to capture 90% of the energy: 49
```

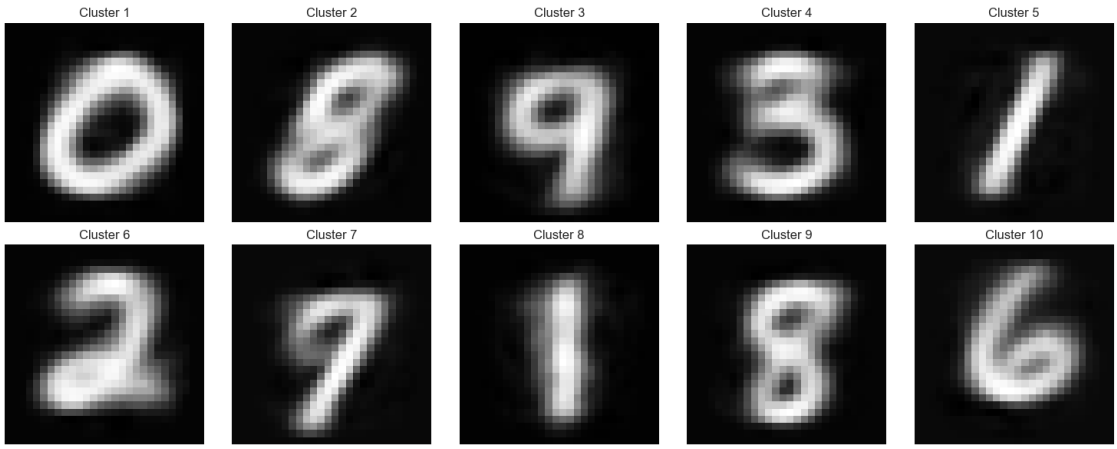
I decided to choose the rank that captures a significant portion, 90%, of the dataset's energy, which is often measured as the sum of the squared singular values.

Through this way, we can preserve as much information as possible while reducing the dataset's complexity.

e) Using Kmeans on this new dataset, cluster the images from d) using 10 clusters and plot the centroid of each cluster. Note: the centroids should be represented as images.

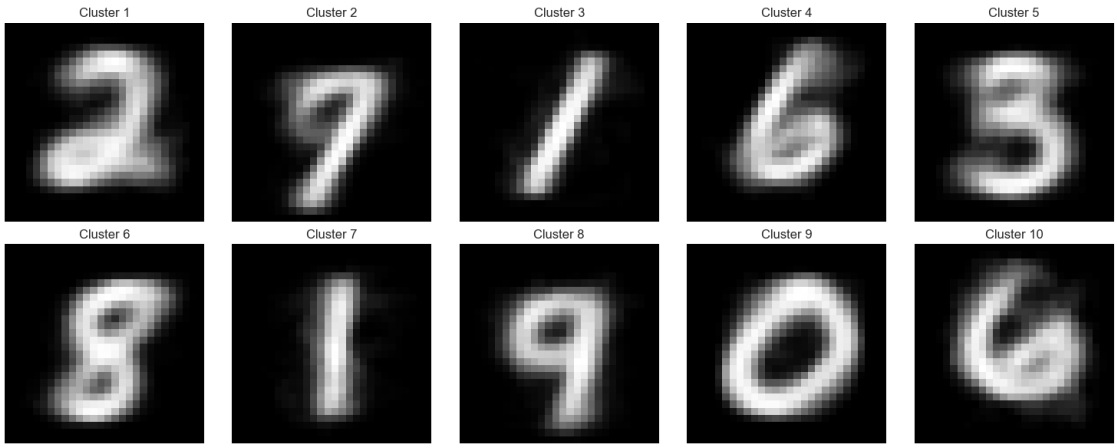
[20]:
```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

pca = PCA(n_components=rank_choice)
X_reduced = pca.fit_transform(sample_X)

kmeans = KMeans(n_clusters=10, init='k-means++', n_init=10, random_state=42)
cluster_labels = kmeans.fit_predict(X_reduced)
centroids_reduced = kmeans.cluster_centers_

centroids = pca.inverse_transform(centroids_reduced)

fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(centroids[i].reshape(28, 28), cmap=cm.Greys_r)
```

```
    ax.set_title(f'Cluster {i+1}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```



f) Repeat e) on the original dataset (if you used a subset of the dataset, keep using that same subset). Comment on any differences (or lack thereof) you observe between the centroids created here vs the ones you created in e).

[21]:
```
kmeans = KMeans(n_clusters=10, init='k-means++', n_init=10, random_state=42)
kmeans.fit_predict(sample_X)

centroids = kmeans.cluster_centers_

fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(centroids[i].reshape(28, 28), cmap=cm.Greys_r)
    ax.set_title(f'Cluster {i+1}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```



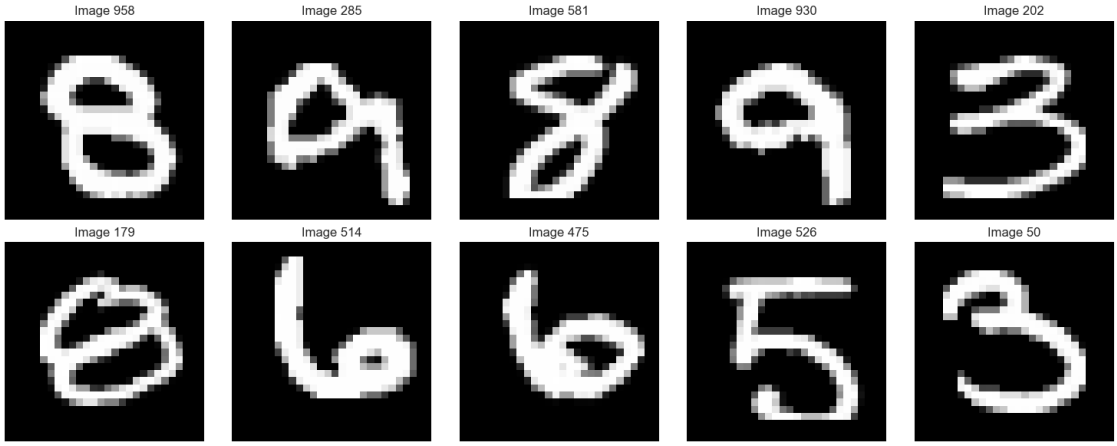
Based on the results, I don't see much clear differences between the centroids created here vs the ones I created in e).

g) Create a matrix (let's call it `O`) that is the difference between the original dataset and the rank-10 approximation of the dataset. i.e. if the original dataset is `A` and the rank-10 approximation is `B`, then `O = A - B`

[22]:
```
pca = PCA(n_components=10)
A = sample_X
A_reduced = pca.fit_transform(A)
B = pca.inverse_transform(A_reduced)

O = A - B
```

h) The largest (using euclidean distance from the origin) rows of the matrix `O` could be considered anomalous data points. Briefly explain why. Plot the 10 images (by finding them in the original dataset) responsible for the 10 largest rows of that matrix `O`.

[23]:
```
distances = np.linalg.norm(O, axis=1)

indices_of_largest = np.argsort(distances)[-10:]

fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for i, ax in enumerate(axes.flatten()):
    image = A[indices_of_largest[i]].reshape(28, 28)
    ax.imshow(image, cmap=cm.Greys_r)
    ax.set_title(f'Image {indices_of_largest[i]+1}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```



The 10 images are plotted above.

They could be considered anomalous data points, since the rows in `O` with the largest Euclidean distances represent data points whose characteristics are least represented in the rank-10 approximation. These points deviate significantly from the patterns captured by the principal components used in the approximation. In the context of data analysis, such deviations could indicate anomalies or outliers, which are data points that differ significantly from the main data distribution.