# Welcome to the final project!

In this notebook you will be asked to use singular value decomposition and SVM to classify images. We will be working with the MNIST numbers dataset, where training data consist of pictures of digits, and the target value is the digit itself.

First, we import the necessary libraries.

In [1]:
```
import numpy as np
from numpy.linalg import svd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

Now, we read both training and test dataset as arrays.

In [2]:
```
data = np.load('mnist.npz')
X_test_total, X_train_total, y_train_total, y_test_total = data['x_test'], da
```

Let's select two digits that we will be learning to separate, for example 3 and 8.

In [3]:
```
num1, num2 = 3, 8
```

Let us form the lists of indices i such that the target of i-th object of our training data is either num1 or num2. Do the same for the test dataset.

In [4]:
```
train_indx = [y == num1 or y == num2 for y in y_train_total]
test_indx = [y == num1 or y == num2 for y in y_test_total]
```

Form new arrays consisting of the data with the target values num1 and num2 only.

In [5]:
```
X_train, y_train = X_train_total[train_indx], y_train_total[train_indx]
X_test, y_test = X_test_total[test_indx], y_test_total[test_indx]
```

The following two cells ensure automatic grading.

In [6]:
```
# import sys
# sys.path.append("..")

# import grading
# grader = grading.Grader(assignment_key="5QcKcr06RZWNXOR6Zubz0g",
#                        all_parts=["EGrPV", "LtYil", "otUqA", "o4nIb", "rZkTW
```

In [7]:
```
# # token expires every 30 min
# COURSERA_TOKEN = # YOUR COURSERA TOKEN HERE (can be found in Programming se
# COURSERA_EMAIL = # YOUR COURSERA EMAIL HERE
```

# Looking at the data

Let us check the sizes of the datasets and the shape of one image.

In [8]:
```
print('Data shapes: ')
print('X_train: ', X_train.shape)
print('y_train: ', y_train.shape)
print('X_test: ', X_test.shape)
print('y_test: ', y_test.shape)
```

```
Data shapes:
X_train:  (11982, 28, 28)
y_train:  (11982,)
X_test:  (1984, 28, 28)
y_test:  (1984,)
```

In [9]:
```python
n_train = X_train.shape[0]
n_test = X_test.shape[0]

n_train, n_test
```

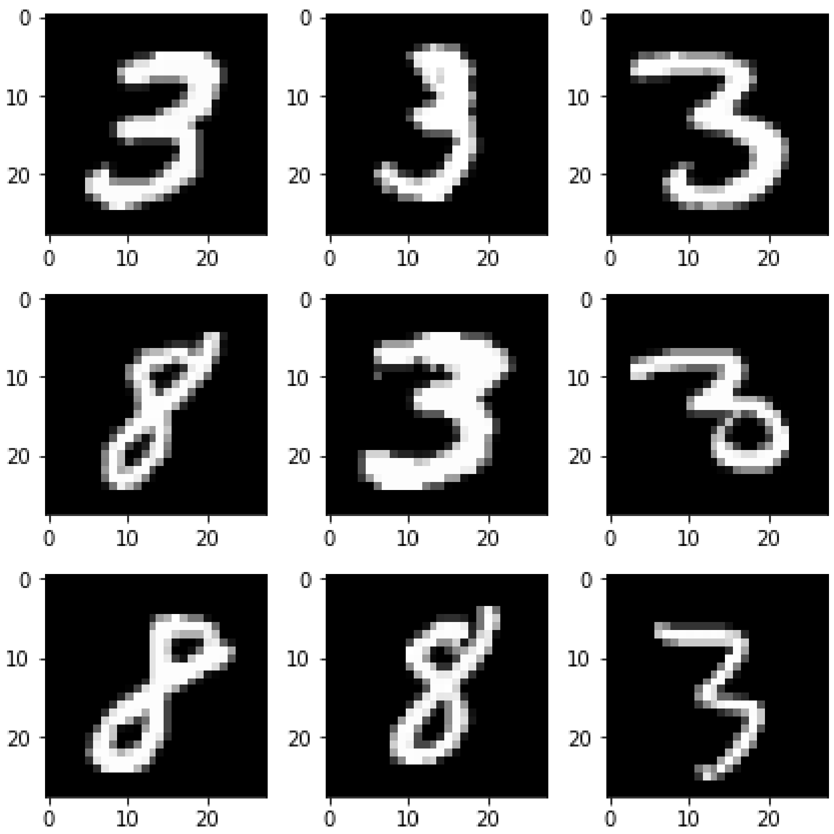
Out[9]:
```
(11982, 1984)
```

In [10]:
```python
print('Shape of one item: ')
print(X_train[0].shape)
```

```
Shape of one item:
(28, 28)
```

Train data are images of digits.

In [11]:
```python
plt.figure(figsize=(6,6))
a, b = 3, 3
for i in range(a*b):
    plt.subplot(b, a, i+1)
    plt.imshow(X_train[i], cmap='gray')
plt.tight_layout()
plt.show()
```



Target values are numbers.

In [12]:
```python
y_train[:9]
```

Out[12]:
```
array([3, 3, 3, 8, 3, 3, 8, 8, 3], dtype=uint8)
```

# Task 1 (1 point)

Now our data is 3-dimensional of shape (number of images, n_pixels, n_pixels). To work with PCA and SVM we need to flatten the images by turning each of them into an array of shape (n_pixels x n_pixels, ).

In [13]:
```
def flatten_image(X):
    d1, d2= X.shape
    X_flatten = X.reshape(d1*d2)

    return X_flatten
```

In [14]:
```
X_train_flat = np.array([flatten_image(img) for img in X_train]) # np.array([
X_test_flat = np.array([flatten_image(img) for img in X_test]) # your code he
X_test_flat.shape, X_test_flat.shape
```

Out[14]:
```
((1984, 784), (1984, 784))
```

PCA works best when the data is scaled (think, why?), so let's scale our data. We will use StandartScaler for it. Note, that scaling replaces a collection of vectors x by the collection of the vectors $x' = (x-M)/D$, where $M$ is the mean vector of the sample, $D$ is the vector of standard deviations of all components of the vectors, and the division is component-wise. So, the scaled collection has the same size as the original one, and each column has 0 mean and unit standard deviation.

In [15]:
```
scaler = StandardScaler()
X_train_flat = scaler.fit_transform(X_train_flat)
X_test_flat = scaler.transform(X_test_flat)
```

# Question 1

Please write your answer on the impact of scaling below. Why does scaling help PCA? If your idea need some computer experiments for confirmation (say, training and accuracy calculations with non-scaled data), please provide the code here as well.

***Your answer here.***

In [16]:
```
#your code here
# svm_linear = LinearSVC()

# svm_linear.fit(X_train_flat, y_train)
# predictions = svm_linear.predict(X_test_flat)
# print('Result for scaled data', accuracy_score(y_true=y_test, y_pred=predic


X_train_non_scaled = np.array([flatten_image(img) for img in X_train]) # np.a
X_test_non_scaled = np.array([flatten_image(img) for img in X_test])

svm_linear = LinearSVC()
svm_linear.fit(X_train_non_scaled, y_train)
predictions = svm_linear.predict(X_test_non_scaled)
accuracy_score(y_true=y_test, y_pred=predictions)
print('Result for original data', accuracy_score(y_true=y_test, y_pred=predic
```

```
Result for original data 0.9480846774193549
/home/tair/anaconda3/envs/data_science/lib/python3.8/site-packages/sklearn/sv
m/_base.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Now, we call PCA and reduce the number of components for each vector.

In [17]:
```
pca = PCA(n_components=128, random_state=42)
X_train_flat = pca.fit_transform(X_train_flat)
```

In [18]:
```
X_test_flat = pca.transform(X_test_flat)
```

In [19]:
```
X_test_flat.shape, X_test_flat.shape
```

Out[19]:
```
((1984, 128), (1984, 128))
```

## Question 2

What is the ratio of the memory used for the data `compressed' by PCA and the one used for the original data?

***Your answer here.***

Now, we use SVM with linear kernel to separate the two classes.

In [20]:
```
%%time

# What is the ratio of the memory used for the data `compressed' by PCA and t
clf = LinearSVC(random_state=42)
clf.fit(X_train_non_scaled, y_train)
```

```
CPU times: user 7.09 s, sys: 89.6 ms, total: 7.18 s
Wall time: 8 s
/home/tair/anaconda3/envs/data_science/lib/python3.8/site-packages/sklearn/sv
m/_base.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[20]:
```
LinearSVC(random_state=42)
```

In [21]:
```
%%time
clf = LinearSVC(random_state=42)
clf.fit(X_train_flat, y_train)
```

```
CPU times: user 8.33 s, sys: 12.4 ms, total: 8.34 s
Wall time: 8.47 s
/home/tair/anaconda3/envs/data_science/lib/python3.8/site-packages/sklearn/sv
m/_base.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[21]:
```
LinearSVC(random_state=42)
```

Now, let us make the predictions and calculate the accuracy, that is, the ratio of the true predictions to the test sample size. Use accuracy score as the quality metric here.

$$accuracy(y\_true, y\_pred) = \frac{1}{n}\sum_{i=1}^n [y\_true_i=y\_pred_i],$$
where $[a=b]=1$, if $a=b$, and $0$ otherwise.

In [22]:
```
y_pred = clf.predict(X_test_flat) # your code here
acc = accuracy_score(y_test, y_pred) # your code here
print("Test accuracy: ", acc)
```

```
Test accuracy:  0.9667338709677419
```

In [23]:
```
# ## GRADED PART, DO NOT CHANGE!
# grader.set_answer("EGrPV", acc)
```

In [24]:
```
# # you can make submission with answers so far to check yourself at this sta
```

```
# grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

## Try it from your own input

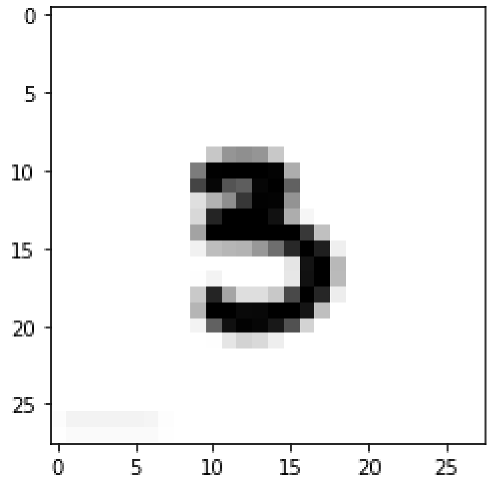
Try to make your own dataset. You can either make a photo image of an ink-written digit or draw a digit using a graphical editor of your computer or smartphone. Note that the input picture has to be a white number on a black background, like the numbers in the MNIST dataset. It can be either in png or jpeg format. Replace the sample striwith your file name.

In [25]:
```
from scipy import misc
from PIL import Image
```

In [26]:
```
image = Image.open('3.jpeg').convert('L')
new_image = image.resize((28, 28))
custom = np.array(new_image)
custom.shape
```

Out[26]:
```
(28, 28)
```

In [27]:
```
plt.imshow(custom, cmap='gray')
plt.show()
```




Re-shape your image and make a prediction.

In [28]:
```
custom = flatten_image(custom).reshape(1, -1)
custom = scaler.transform(custom)
custom.shape
custom = pca.transform(custom)
custom.shape
```

Out[28]:
```
(1, 128)
```

In [29]:
```
clf.predict(custom)
```

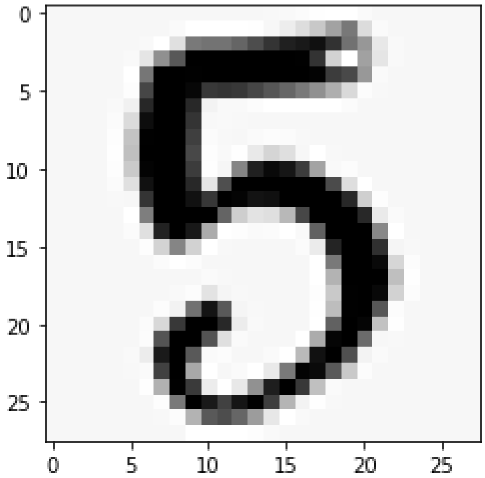
Out[29]:
```
array([3], dtype=uint8)
```

## Question 3

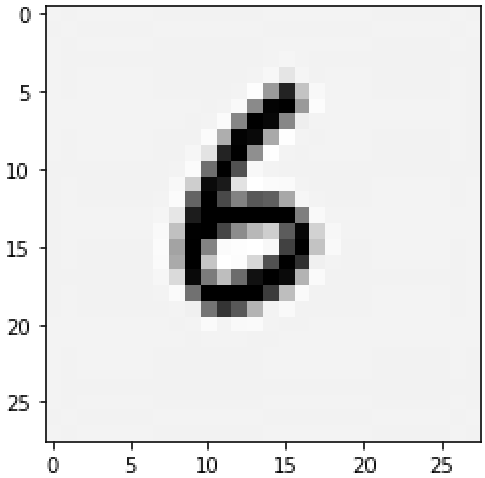
Repeat the above digit recognition procedure with other 5 to 10 hand-written images. Do your experiments confirm the above accuracy estimate? How do you think, why they confirm (or not confirm) it?

***Your answer here.***

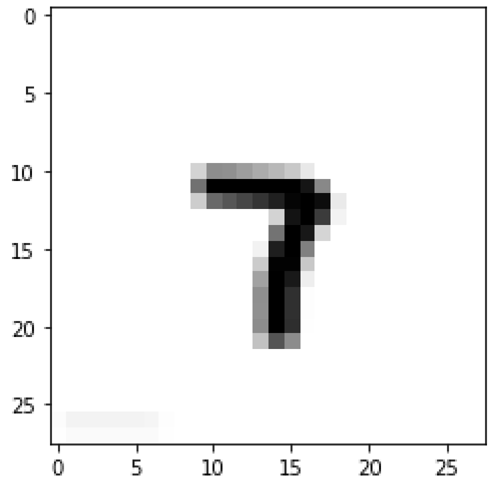
In [30]:
```python
ans = 0
for n in range(5,9):
    image = Image.open(f'{n}.jpeg').convert('L')
    new_image = image.resize((28, 28))
    custom = np.array(new_image)
    plt.imshow(custom, cmap='gray')
    plt.show()
    custom = flatten_image(custom).reshape(1, -1)
    custom = scaler.transform(custom)
    custom.shape
    custom = pca.transform(custom)
    print(clf.predict(custom))
    if int(clf.predict(custom)) == n:
        ans += 1
print('Accuracy on 5-8 digits is', ans/4)
```



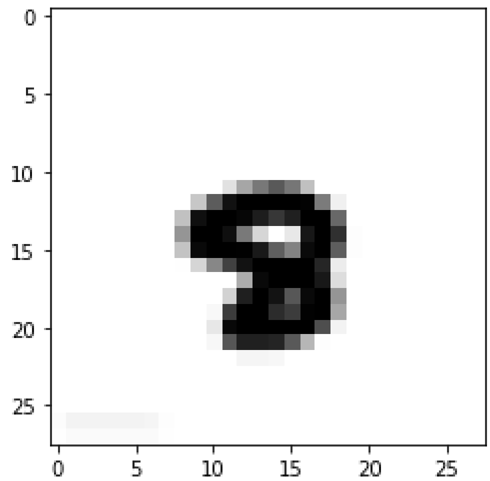

```
[3]
```




```
[3]
```



```
[3]
```



```
[3]
Accuracy on 5-8 digits is 0.0
```

# Task 2

Now let's try another approach explained here in Section 3. For each digit, we create a new matrix $A$ whose columns are flattened images of this digit. The first several (say, 10) columns of the matrix $U$ from SVD decomposition of $A$ represent a collection of "typical" images of this digit. Given an unrecognized flatten image, among average typical flattened images we find the closets one. Its target value is considered as a prediction for the target of the unrecognized image.

# SVD refesher

As you may recall from the lectures, SVD of a matrix $A$ is a decomposition: $A = U \Sigma V^T,$ where $U$ and $V$ are orthogonal matrices. In this method we will be utilizing some properties of SVD. Please note that due to large shapes of matrices the operations might take a while.

In [31]:
```
X_train_total[:,:10].shape
y_train
X_train_total.shape
```

Out[31]:
```
(60000, 28, 28)
```

In [32]:
```
def getSingularVectorsLeft(matrix, number=10): # let's take first 10 numbers
```

```
    u, s, vh = np.linalg.svd(matrix, full_matrices=False)
#     print('u', u.shape)
#     print(u[:10])
    return u[:,:10]
    # return first _number_ columns of U from SVD of _matrix_
```

In [33]:
```
def getSingularImage(X_train, y_train, number):
    A = []
    # find images whose target is _number_
    select_images = X_train[np.array(np.where(y_train==number)[0])]
    # iteratively append new column to form matrix A
    for image in select_images:
        image = flatten_image(image).reshape(1, -1)
        A.append(image[0])
    A = np.array(A)
    A = A.T
#     plt.imshow(A[:,3].reshape(28,28))
#     print(A.shape)
    left_basis = getSingularVectorsLeft(A, 10)
    # left_basis = # get left singular vectors

    return left_basis
```

In [34]:
```
# A = []
# select_images = X_train_total[np.array(np.where(y_train_total==2)[0])]

# for image in select_images[:10]:
#     image = flatten_image(image).reshape(1, -1)
#     A.append(image[0])
# A = np.array(A)
# A = A.T
# A.shape
# plt.imshow(A[:,3].reshape(28,28))
```

Try it first on "0".

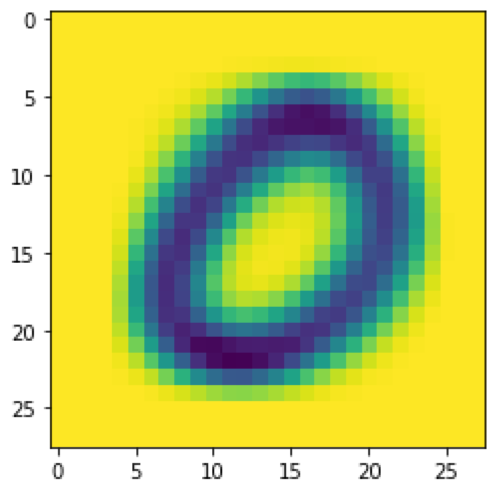
In [35]:
```
left_basis = getSingularImage(X_train_total, y_train_total, 0)

# assert left_basis.shape, (784, 10)
```

In [36]:
```
print(left_basis.shape)
plt.imshow(left_basis[:,0].reshape(28,28))
```

```
(784, 10)
```

Out[36]: `<matplotlib.image.AxesImage at 0x7f4f3a1c0670>`

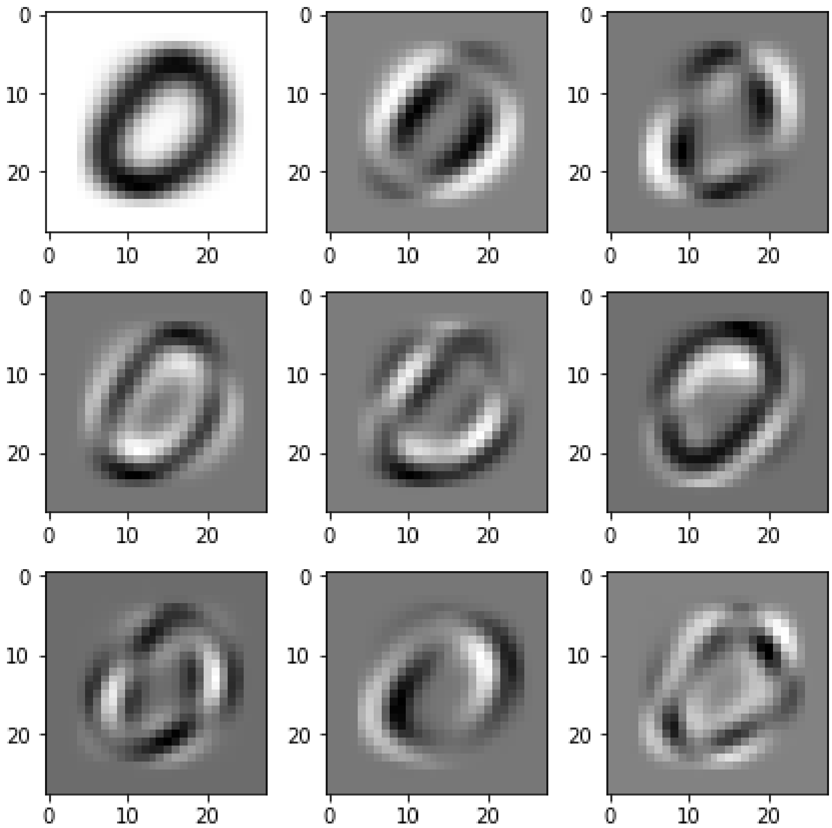



## Task 2.1 (1 point)

Plot first 9 singular images of the digit 0 taking columns of matrix U and reshaping them back into images 28x28. Use `numpy.reshape`.

In [37]:
```
#singular images
plt.figure(figsize=(6,6))
a, b = 3, 3
for i in range(a*b):
    plt.subplot(b, a, i+1)
    img = getSingularImage(X_train_total, y_train_total, 0)[:,i].reshape(28,2
    plt.imshow(img, cmap='gray')

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




In [38]:
```
# ## GRADED PART, DO NOT CHANGE!
# #9th image will be graded:
# grader.set_answer("LtYil", img[:, 5:7].flatten())
```

In [39]:
```
# # you can make submission with answers so far to check yourself at this sta
# grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

## Question 4

Reflect on properties of the columns of $U_k$. What properties do you think are contained in each of them? Draw more singular images to help you make conclusions.

***Your answer here.***

# $Uk$ is Unitary matrix having left singular vectors as columns.

Now let's move on and obtain singular images for all numbers. The matrices $U_k$ from the article are represented as `number_basis_matrices[k]`. This might take a while to finish, feel free to add debug print in your function to know the progress.

In [40]:
```
number_basis_matrices = np.array([getSingularImage(X_train_total, y_train_tot
```

In [41]:
```
number_basis_matrices[0].shape
```

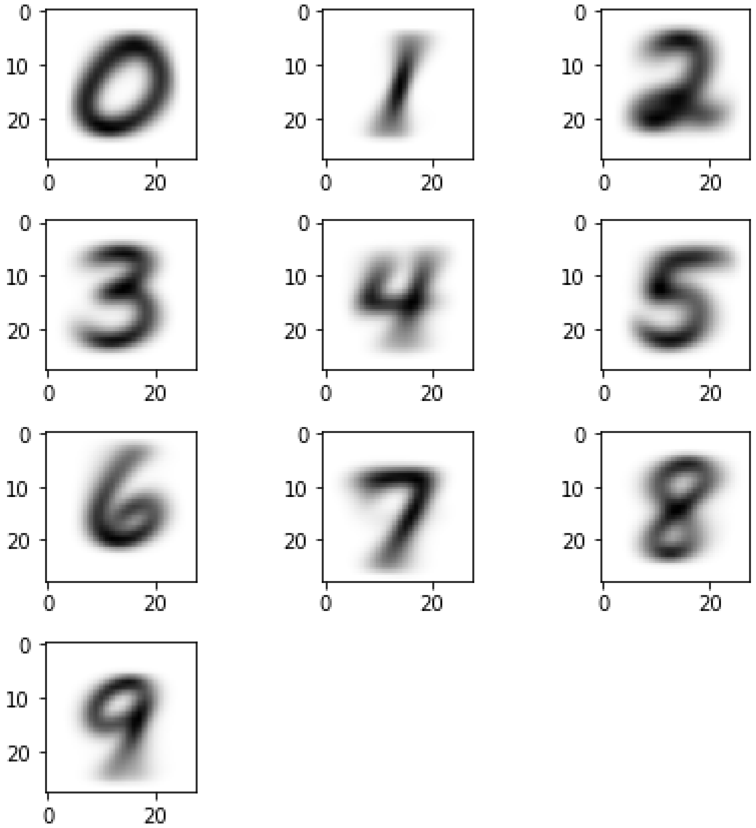
Out[41]:
```
(784, 10)
```

### Task 2.2 (1 point)

Plot the first singular image of each digit, similarly to the previous task.

In [42]:
```
plt.figure(figsize=(6,6))
a, b = 3, 4
for i in range(10):
    plt.subplot(b, a, i+1)
    img = number_basis_matrices[i][:,0].reshape(28,28) # first column of U_k
    plt.imshow(img, cmap='gray')

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



In [43]:
```
# ## GRADED PART, DO NOT CHANGE!
# #last image (of digit 9) will be graded:
# grader.set_answer("otUqA", img[:, 5:7].flatten())
```

In [44]:
```
# # you can make submission with answers so far to check yourself at this sta
# grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

In [45]:
```
dim = number_basis_matrices[0].shape[0]
dim
```

Out[45]:
```
784
```

## Task 2.3 (1.5 points)

Here we calculate the new projection matrix for each \$U_k\$ to apply later in testing: \$pr = (I - U_k \cdot U_{k}^{T})\$. Use `numpy.matmul` for matrix multiplication and `numpy.identity` to create an identity matrix. Please note that this operation might also take some time to finish.

In [46]:
```
# create an array of pr for each number
numeric_values = np.array([np.identity(dim) - np.matmul(number_basis_matrices
print(len(numeric_values))
numeric_values[0].shape
```

```
10
```

Out[46]:
```
(784, 784)
```

In [ ]:

In [47]:
```
# ## GRADED PART, DO NOT CHANGE!
# k = np.array([n[3:5, 3:13] for n in numeric_values])
# grader.set_answer("o4nIb", k.flatten())
```

In [48]:
```
# # you can make submission with answers so far to check yourself at this sta
# grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

## Task 2.4 (1.5 points)

Implement function utilizing `numeric_values` matrices to predict labels for unknown images. Use `numpy.norm` and enumerate to iterate over numeric values.

In [49]:
```
def find_closest(test_value, numeric_values):
    if test_value.shape[0] == 28:
#         print(test_value.shape)
        test_value = flatten_image(test_value).reshape(1, -1)
    values = []
    for U in numeric_values:
        values.append(np.linalg.norm(np.dot(U,test_value)))
    return values.index(min(values))
#     return index
```

In [50]:
```
X_test_SVD = np.array([flatten_image(img) for img in X_test_total])
y_pred = [find_closest(value, numeric_values) for value in X_test_SVD] # finc
```

In [51]:
```
# y_pred[220:230]
```

In [52]:
```
acc = accuracy_score(y_test_total, y_pred)
acc
```

Out[52]:
```
0.9485
```

In [53]:
```
# ## GRADED PART, DO NOT CHANGE!
# grader.set_answer("rZkTW", acc)
```

In [54]:
```
# # you can make submission with answers so far to check yourself at this sta
# grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

# Additional task (2 points)

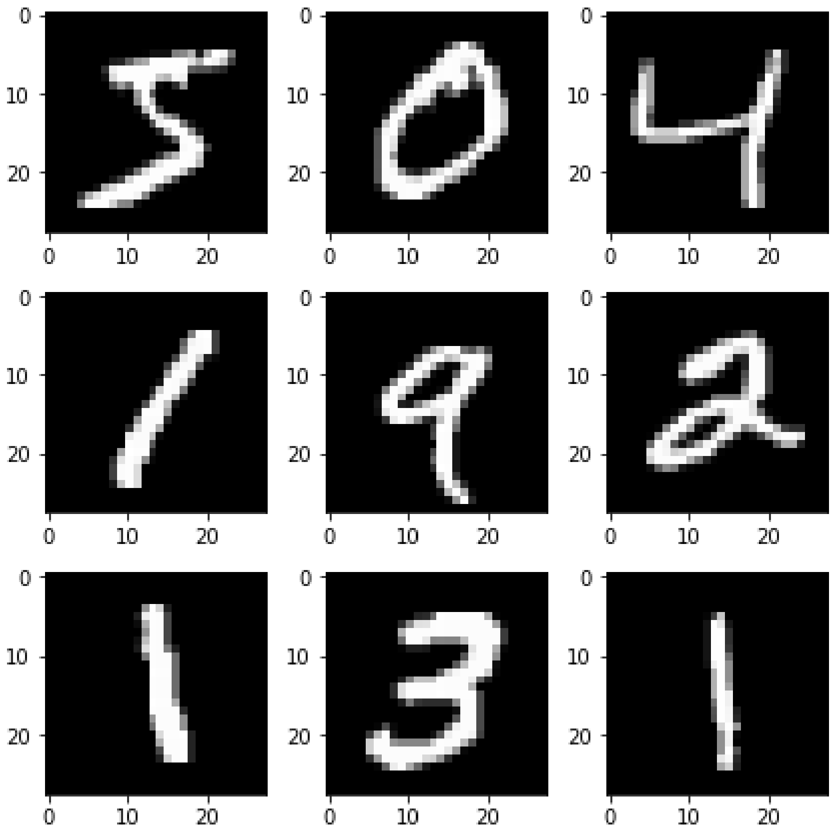
In the first task we trained Linear SVM to separate 3s and 8s. Here you can implement multiclass classification for *all* numbers in `MNIST`. Use the same function LinearSVC for "one-vs-the-rest" multi-class strategy, see the documentation. Follow the same steps from task 1: scaling, feature selection, training and testing. Is the accuracy of this method greater then the one calculated above?

**Note:** Use `random_state=42` for `PCA` and `LinearSVC`. Training `LinearSVC` on all the data might take a while, that's normal.

In [55]:
```
# flatten
# Scandart Scaler
# PCA
# LinearSVC
```

In [56]:
```
data = np.load('mnist.npz')
X_test_total, X_train_total, y_train_total, y_test_total = data['x_test'], da
```

In [57]:
```
plt.figure(figsize=(6,6))
a, b = 3, 3
for i in range(a*b):
    plt.subplot(b, a, i+1)
    plt.imshow(X_train_total[i], cmap='gray')
plt.tight_layout()
plt.show()
```



In [58]:
```
y_train_total[:9]
```

Out[58]:
```
array([5, 0, 4, 1, 9, 2, 1, 3, 1], dtype=uint8)
```

In [59]:
```
def flatten_image(X):
    d1, d2= X.shape
    X_flatten = X.reshape(d1*d2)

    return X_flatten
```

In [60]:
```python
X_train_flat = np.array([flatten_image(img) for img in X_train_total]) # np.a
X_test_flat = np.array([flatten_image(img) for img in X_test_total]) # your c
X_test_flat.shape, X_test_flat.shape
```

Out[60]:
```
((10000, 784), (10000, 784))
```

PCA works best when the data is scaled (think, why?), so let's scale our data. We will use StandartScaler for it. Note, that scaling replaces a collection of vectors x by the collection of the vectors $x' = (x-M)/D$, where $M$ is the mean vector of the sample, $D$ is the vector of standard deviations of all components of the vectors, and the division is component-wise. So, the scaled collection has the same size as the original one, and each column has 0 mean and unit standard deviation.

In [61]:
```python
scaler = StandardScaler()
X_train_flat = scaler.fit_transform(X_train_flat)
X_test_flat = scaler.transform(X_test_flat)
```

Now, we call PCA and reduce the number of components for each vector.

In [62]:
```python
pca = PCA(n_components=128, random_state=42)
X_train_flat = pca.fit_transform(X_train_flat)
```

In [63]:
```python
X_test_flat = pca.transform(X_test_flat)
```

In [64]:
```python
X_test_flat.shape, X_test_flat.shape
```

Out[64]:
```
((10000, 128), (10000, 128))
```

In [65]:
```python
%%time
clf = LinearSVC(random_state=42)
clf.fit(X_train_flat, y_train_total)
```

```
CPU times: user 3min 41s, sys: 0 ns, total: 3min 41s
Wall time: 3min 42s
```

```
/home/tair/anaconda3/envs/data_science/lib/python3.8/site-packages/sklearn/sv
m/_base.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[65]:
```
LinearSVC(random_state=42)
```

In [66]:
```python
y_pred = clf.predict(X_test_flat) # your code here
acc = accuracy_score(y_test_total, y_pred) # your code here
print("Test accuracy: ", acc)
```

```
Test accuracy:  0.9079
```

In [67]:
```python
## GRADED PART, DO NOT CHANGE!
grader.set_answer("keYiw", acc)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-67-f6b9c1f36494> in <module>
      1 ## GRADED PART, DO NOT CHANGE!
----> 2 grader.set_answer("keYiw", acc)

NameError: name 'grader' is not defined
```

In [ ]:
```python
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

In [ ]:
```python

```