SVD and Random Forest

Compare The data set contains images of hand-written digits: 10 classes where each class refers to a digit. Goal: Compare the accuracy of Random Forest model when it is trained with the original iamge (8x8=64 features) and When it is used with reduced feature space obtained from singular value decomposition(SVD)

- Study the dimentionality reduction feature of SVD.
- Note: Using Out of bag score in random forest accuracy measure

About the Date:

Preprocessing programs made available by NIST were used to extract normalized bitmaps of handwritten digits from a preprinted form. From a total of 43 people, 30 contributed to the training set and different 13 to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of 4x4 and the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0..16. This reduces dimensionality and gives invariance to small distortions.

Import Libs

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Load Data

In [3]:
```
from sklearn.datasets import load_digits
X_org, y_org = load_digits(return_X_y = True)
```

In [ ]:
```
```

In [4]:
```
# about X
print(type(X_org))
print(X_org.shape)
X_org[0, :]
```

```
<class 'numpy.ndarray'>
(1797, 64)
```

Out[4]:
```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

In [5]:
```
# about y
print(y_org.shape)
y_org[0:30]
```

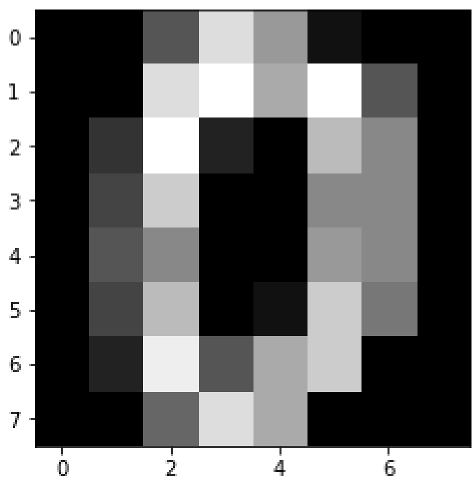
```
(1797,)
```

Out[5]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1,
       2, 3, 4, 5, 6, 7, 8, 9])
```

Visulaize one of the input image

In [6]:
```
index = 0
Image = X_org[index, :]
plt.imshow(Image.reshape(8,8), cmap='gray')
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x20cda06cc88>
```



Build the Random Forest Model

In [7]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [8]:
```
def get_scoreRF(In, out):
    model = RandomForestClassifier(oob_score=True)
    model.fit(In, out)
    return model.oob_score_
```

To ignore warning

In [9]:
```
import warnings
warnings.simplefilter('ignore')
```

Random Forest score on Original Image

In [10]:
```
print('score=%.2f'%(get_scoreRF(X_org, y_org)))
```

```
score=0.88
```

Reduceding the features by SVD

For two cases:

- reduce the image feaure to 2
- reduce the image feature to 16

The goal is to see whether the Random Forest would be able to predict the value well or not!? Reconstruct Iamge with 2 features

In [11]:
```
from sklearn.decomposition import TruncatedSVD
```

In [12]:
```
def reduceFeatures(In, n_components):
    svd_model = TruncatedSVD(n_components=n_components)
    In_reduced = svd_model.fit_transform(In)
    score = get_scoreRF(In_reduced, y_org)
    return score,svd_model, In_reduced[0,:]
```

Reucing the features to 2:

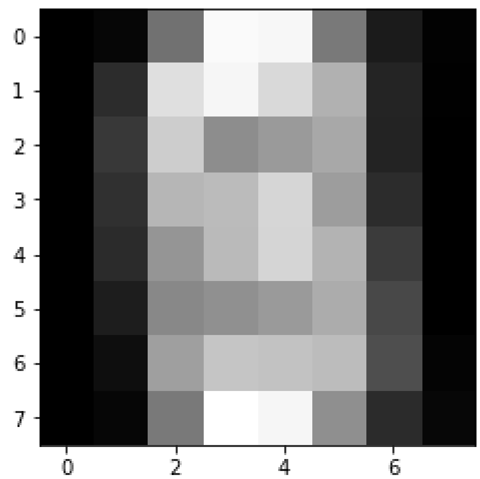
In [13]:
```
score,model_2, Image_reduced = reduceFeatures(X_org, n_components=2)
print('score=%.2f'%(score))
```

```
score=0.38
```

```
In [14]: ## visualize an example
         Image_rec = model_2.inverse_transform(Image_reduced.reshape(1,-1))
         print(Image_rec.shape)
         plt.imshow(Image_rec.reshape(8,8), cmap='gray')
```

```
(1, 64)
```

```
Out[14]: <matplotlib.image.AxesImage at 0x20cdb1cdf60>
```



As we see, with 2 features it is very hard to distinguish waht the digit is.

```
In [15]: model_2.explained_variance_ratio_
```

```
Out[15]: array([0.02870851, 0.1489005 ])
```

```
In [16]: model_2.explained_variance_ratio_.sum()
```

```
Out[16]: 0.17760900817197903
```

Reucing the features to 16:

```
In [17]: score,model_16, Image_reduced = reduceFeatures(X_org, n_components=16)
         print('score=%.2f'%(score))
```

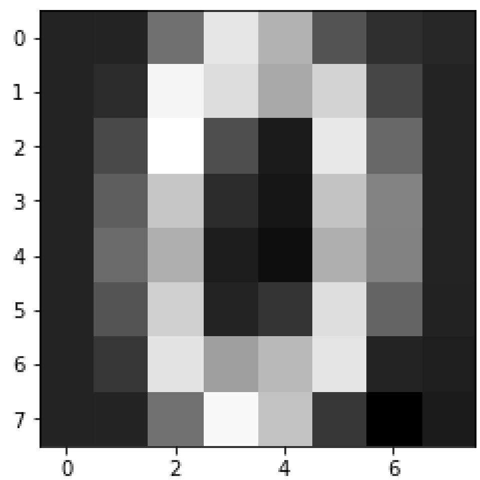
```
score=0.86
```

With using 16/64=25% of the data, the acuuracy is comparable to the original one.

In [18]:
```
## visualize an example
Image_rec = model_16.inverse_transform(Image_reduced.reshape(1,-1))
print(Image_rec.shape)
plt.imshow(Image_rec.reshape(8,8), cmap='gray')
```

```
(1, 64)
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x20cdb22fd68>
```



In [19]:
```
model_16.explained_variance_ratio_
```

Out[19]:
```
array([0.02870851, 0.1489005 , 0.13605748, 0.11771282, 0.0838876 ,
       0.0577855 , 0.04752737, 0.04225609, 0.03619554, 0.03339511,
       0.02421063, 0.02327175, 0.01994783, 0.01797682, 0.01598064,
       0.01414377])
```

In [20]:
```
model_16.explained_variance_ratio_.sum()
```

Out[20]:
```
0.8479579535436155
```

Selecting the best number of TSVD

In [21]:
```
# Create and run an TSVD with one less than number of features
tsvd = TruncatedSVD(n_components=X_org.shape[1]-1)
X_tsvd = tsvd.fit(X_org)
```

In [22]:
```
# List of explained variances
tsvd_var_ratios = tsvd.explained_variance_ratio_
```

In [23]:
```python
# Create a function
def select_n_components(var_ratio, goal_var: float) :
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0

    # For the explained variance of each feature:
    for explained_variance in var_ratio:

        # Add the explained variance to the total
        total_variance += explained_variance

        # Add one to the number of components
        n_components += 1

        # If we reach our goal level of explained variance
        if total_variance >= goal_var:
            # End the loop
            break

    # Return the number of components
    return n_components
```

In [24]:
```python
# Run function
best_n_components = select_n_components(tsvd_var_ratios, 0.95)
print(best_n_components)
```

```
29
```

In [25]:
```python
## Reduce the features to the best selecetd one
```

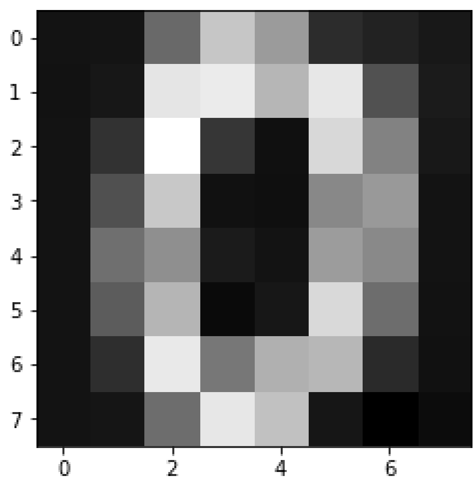
In [26]:
```python
score,best_model, Image_reduced = reduceFeatures(X_org, n_components=best_n_co
mponents)
print('score=%.2f'%(score))
```

```
score=0.85
```

In [27]:
```
## visualize an example
Image_rec = best_model.inverse_transform(Image_reduced.reshape(1,-1))
print(Image_rec.shape)
plt.imshow(Image_rec.reshape(8,8), cmap='gray')
```

```
(1, 64)
```

Out[27]:
```
<matplotlib.image.AxesImage at 0x20cdc9b1b70>
```



In [28]:
```
best_model.explained_variance_ratio_.sum()
```

Out[28]:
```
0.9547505374819848
```