SVD FOR IMAGE CLASSIFICATION

Computer Science and Engineering

Name, Surname: Mustafa Aktas

Date: 27.02.2019

Problem Definition

The aim of this experiment was to use our knowledge about matrices on the real-life problem and to learn some new topics about matrices. For this reason, we were given the set of handwritten 8 and 1 and wanted to classify test data by using SVD.

Solution

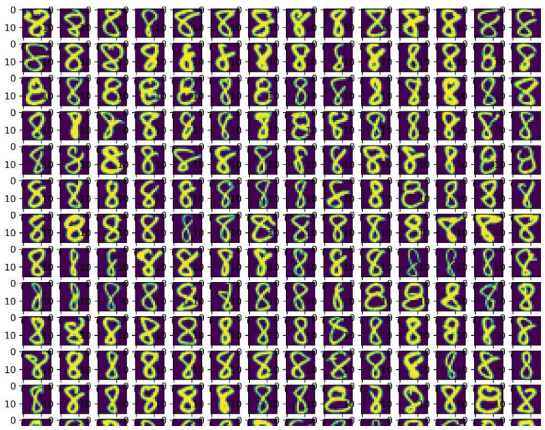
Before explaining the solution, I would like to mention some system requirements for the project. I have used Python 3.5.5 version and here are needed libraries, scipy.io, pandas, matplotlib.pyplot, numpy, and scipy.

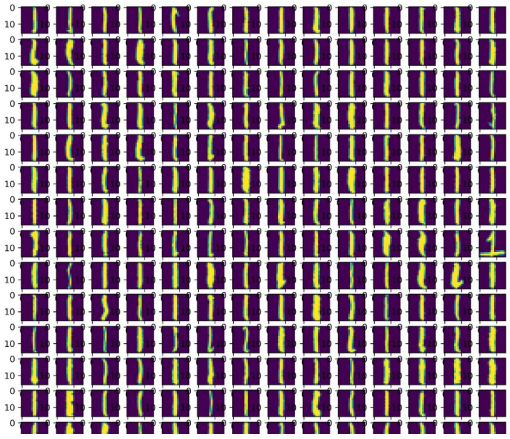
```
#Some required libraries
import scipy.io as sio
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from scipy import optimize
```

The data set was given in the format of .mat. I opened these data set as shown right side. Now, data_train_8 and data_train_1 have our training data set as matrices. Their size is (200, 256) which means 200 samples and each sample has 256 pixels.

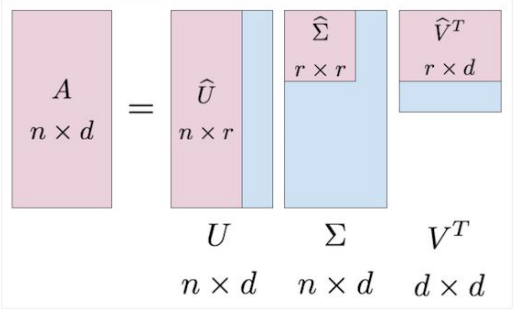
```
#dataset for 8
data_train_8 = sio.loadmat('data_train_8')
data_train_8 = data_train_8['data_train_8']
data_train_8 = np.asarray(pd.DataFrame(data_train_8).values)

#dataset for 1
data_train_1 = sio.loadmat('data_train_1')
data_train_1 = data_train_1['data_train_1']
data_train_1 = np.asarray(pd.DataFrame(data_train_1).values)
```

Then, if we display them by converting from (1, 256) to (16, 16), it would seem like,





It is time to factorize our matrices into U, S, and V. The idea of the singular value decomposition is to reduce the size of the matrices and factorize them into more informative versions. For example, let suppose that our matrix A is (n, d), it is possible to factorize it to U (n, d), S (n, d), and V(d, d). Then we are going to get exactly same matrix A. However, in this case, we try to factorize A into a more informative and reduced form. So, after factorize A to U, S, and V, their sizes will be, U (n, r), S (r ,r), V(r, d). You may wonder the rest of the matrices. I mean, what did happen to matrices then they reduced their size? Did not they lose some information? Yes, they did lose some information, but now we have more informative and reduced matrices which make our problem easy. Finally, our data_train_1 and data_train_8 turned into to U, S, and V. Their sizes are, U (200, 200), S (200, 256), V(256, 256), but do not forget that our core matrices' sizes are smaller than these sizes. If we remove the zeros from S which has eigenvalues on the diagonal, its size will be about (98, 98).

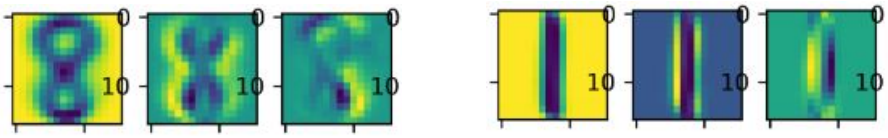


Another information for svd is, V has some bases for the row space of our matrix. However, there is a confusion for this part, in MATLAB, the columns of V are the bases for the matrix, while its row in Python. So, I took the rows of the V as bases for the row spaces. The bases are important for us to determine the core of the handwritten digits 1 and 8. I mean, we asked a question to the matrix A, what is the 'core of the

data set'? Then its respond will be the 'bases'. Thankfully, V sorts the basis form the most informative to the least informative. So, we can directly take the first three bases.

However, another question comes up with that part. Why did we take only first three-part instead of taking all of them? The reason for it is we try to generalize the data or try to find the core of the data. If we took all of the bases, we would create a very strict prediction model, because our prediction model would be included extremely bad samples. Even people are going to have a hard time to determine that is 8. Our model should omit the samples like shown right side.

Then we can get the most informative three bases,



As mentioned before, it is not possible to get exactly same matrix by multiplying S, U, and V. So, the idea behind the bases is it should be possible to find some coefficients that will give us an approximation matrix.

Let's suppose that F is the test handwritten digit number, our aim is to determine the coefficients that are going to be multiplied with bases, then calculate the error values for 8 and 1. For that part, some arrays were created and the test data was imported.

$$F = w_1 p_1 + w_2 p_2 + \cdots + w_k p_k,$$

```
data_test = sio.loadmat('data_test')
data_test = data_test['data_test']
data_test = np.asarray(pd.DataFrame(data_test).values)

data_labels = sio.loadmat('data_labels')
data_labels = data_labels['data_labels']
data_labels = np.asarray(pd.DataFrame(data_labels).values)
```

```
#Some arrays were created to store some values.
error_1 = np.zeros((1147,1)) #This for 3 usage together
error_8 = np.zeros((1147,1))
error_1_2 = np.zeros((1147,1)) #This for 2 usage together
error_8_2 = np.zeros((1147,1))
error_1_1 = np.zeros((1147,1)) #This for 1 usage together
error_8_1 = np.zeros((1147,1))
reconstruction_errors_3 = np.zeros((1147,1))#This for 3 usage together
reconstruction_errors_2 = np.zeros((1147,1))#This for 2 usage together
reconstruction_errors_1 = np.zeros((1147,1))#This for 1 usage together
labels_val = np.zeros((1147,1))
```

Another important part was to write the code for calculating the best coefficients and minimum error,

```
#The class for calculating the minimum error.
from scipy import optimize  #Import the optimize to use 'fmin' function

class findMinError(): #The class that calculates the minimum error for data set

    def __init__(self, basis, sample): #The constructor method takes the basis/
        self.basis = basis             #for data set and the sample that is/
        self.sample = sample           # going to be predicted.
        self.parameters = [[] for i in range(len(self.basis))]  #Empty array to store the constant to
                                                                #calculate the predicted matrix.
    def errfn(self, p): #This method calculates the error between the sample and the
        for i in range(len(p)):
            self.parameters[i] = p[i]
        return sum(((self.objective() - self.sample)**2).flatten())

    def objective(self): #It reconstruct the image
        recons_image = np.zeros((1,256))
        for i in range(len(self.basis)):
            recons_image += self.parameters[i]*self.basis[i]
        return recons_image

    def minErrorRepresentation(self): #This is for calculating the minimum error
        x0 = [[] for i in range(len(self.basis))]
        for i in range(len(self.basis)):
            x0[i] = 0
        min_error = optimize.fmin(self.errfn, x0, disp=False)
        return min_error
```

For the final step, the error values were calculated by comparing the prediction and the labels set. After some calculations, the results became like,

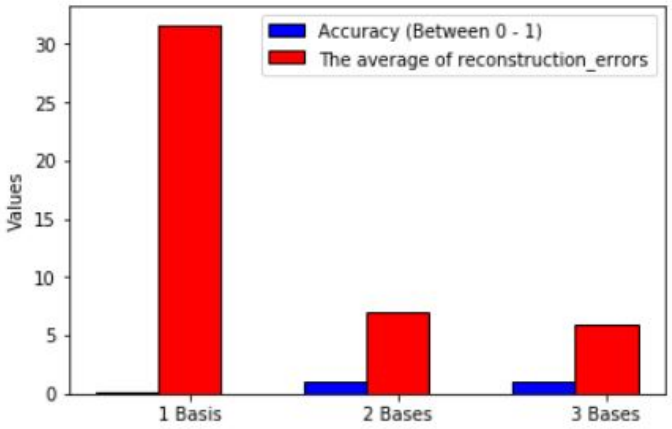
```
#It is for easy to compare the results
Results = {'Accuracy': [round((accuracy1), 2), round((accuracy2), 2), round((accuracy3), 2)],
           'False' : [false1, false2, false3],
           'True' : [true1, true2, true3],
            ' ' : ["Base 1", "Base 2", "Base 3"]}

data_pandas = pd.DataFrame(Results)
display(data_pandas)
```

```
3 bases were used
True: 1137  False: 10
Accuracy: 0.99128160418483

2 bases were used
True: 1139  False: 8
Accuracy: 0.993025283347864

1 basis was used
True: 69  False: 1078
Accuracy: 0.06015693112467306
```

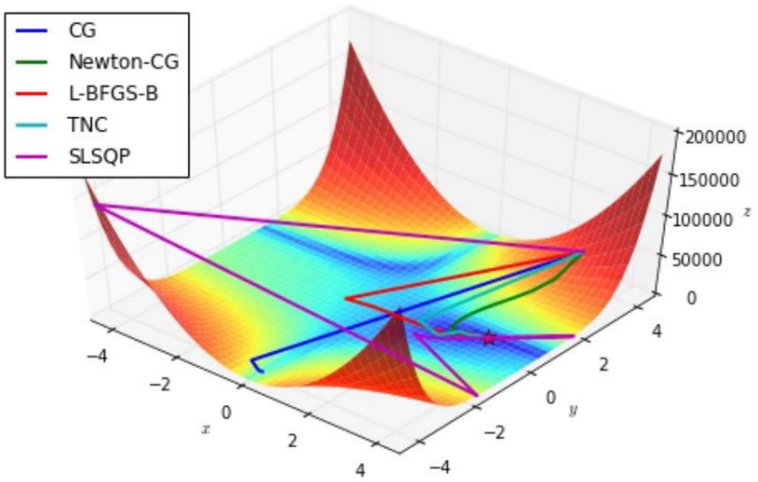
		Accuracy	False	True
0	Base 1	0.06	1078	69
1	Base 2	0.99	8	1139
2	Base 3	0.99	10	1137

If we take 1 basis
If we take 2 bases
If we take 3 bases

As shown above, the accuracy values are different from each other. As mentioned before, the tolerance and the number of bases that were used are opposite to each other. If we take 3 bases, the model will be more strict for some samples and it is going to predict wrong. Also if we decrease the number of bases, our model will not have enough information to predict, so it again gives the wrong predict. It is very important to optimize the accuracy regarding the number of bases. In this case, when 2 bases were used, we get the highest accuracy. Here is the bar plotting to see the relation between the reconstruction error and the accuracy. It is obvious to recognize that they are opposite to each other. When accuracy is low, the reconstruction error (When the bases multiplied with some coefficients to reconstruction the image) is very high which means that the prediction image did not reconstruct well. Also, the accuracies and the reconstruction errors are very close to each other for rest cases. So, it is not possible to say that the reconstruction error and the accuracy have direct relation. To get the lowest reconstruction errors and the highest accuracy, the number of bases should be found by trying.



There is another factor that affects the accuracy of our model, it is the way that you minimize the error. In this project, 'fmin' method was used to find the fittest coefficient. However, this method does not use any derivation, it just uses the function values. There are more modern optimization methods to get higher accuracy. As an example, scipy.optimize.minimize could be used for the project. It uses different algorithms to find or to get closer to the global minimum error value. The picture below shows how their different algorithms work to find the global minimum error. Instead of using "fmin" method which does not use the first or the second derivative, we could have used modern optimization techniques.



Sources

- How Are Principal Component Analysis and Singular Value Decomposition Related? (n.d.). Retrieved from **https://intoli.com/blog/pca-and-svd/**
- MIT. (n.d.). Color Images. Retrieved from https://www.commonlounge.com/discussion/244616b76d3d40f88e8f12103a22743d
- Hadrienj. (2018, March 26). Deep Learning Book Series · 2.8 Singular Value Decomposition. Retrieved from https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.8-Singular-Value-Decomposition/
- Moler, C. (2012, December 04). 1976 Matrix Singular Value Decomposition Film. Retrieved from https://www.youtube.com/watch?v=R9UoFyqJca8
- Gaudard, O. (2017, September 06). #8 Add confidence interval on barplot. Retrieved from https://python-graph-gallery.com/8-add-confidence-interval-on-barplot/
- Scipy.optimize.minimize¶. (n.d.). Retrieved from https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html
- Scipy.optimize.fmin¶. (n.d.). Retrieved from https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin.html
- Borne, K. (2018, September 05). Visualizing and Animating #Optimization #Algorithms with Matplotlib: Https://t.co/lZpSaWbqGD #Python #MachineLearning #DataScience #Coding #DataViz #DataScientists pic.twitter.com/izCdw3Eijy. Retrieved from https://twitter.com/kirkdborne/status/1037279295696191489