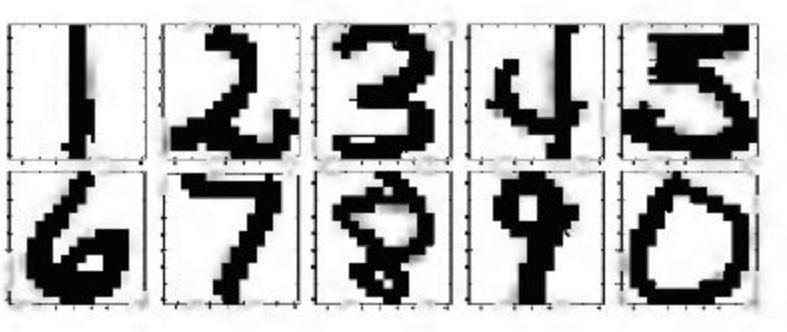
Mandoob Alanazi
Nate Johnson
Math 561

SVD Analysis of Handwritten Digits

Our project was on computer recognition of handwritten digits. This is a classic problem and has many solutions. We used a technique from linear algebra called singular value decomposition (SVD), which is a method for factoring a matrix. It has many different applications in addition to pattern recognition, such as signal processing, weather prediction, and recommender engines. It is also known as principal component analysis, or PCA. In our case, we are using it to reduce the dimensionality of the data to make our digit recognition program run in a reasonable amount of time.

Description of Data

The data we are working with comes from the U.S. Postal Service database of handwritten digits.



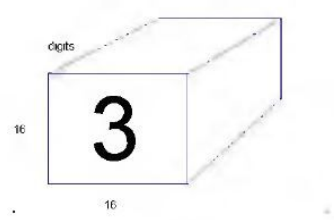
Each number can be thought of as a 16 by 16 array of floats. The floats range from -1 (solid black) to 1 (white). It is actually stored as a 1D array or list of 256 values. For example, here are the first 64 values of a digit (which I split into four rows of 16):

-1. , -1. , -1. , -0.567, -0.064, 0.979, -0.009, -0.167, -0.999, -1. , -1. , -1. , -1. , -1. , -1. , -1. ,
-1. , -1. , -1. , 0.405, 1. , 1. , 1. , 1. , 0.482, -0.701, -1. , -1. , -1. , -1. , -1. , -1. ,
-1. , -1. , -1. , -0.242, 1. , 1. , 1. , 1. , 1. , 0.809, 0.351, -0.764, -1. , -1. , -1. , -1. ,
-1. , -1. , -1. , -0.091, 1. , 1. , 1. , 1. , 0.848, 1. , 1. , 0.748, -0.536, -1. , -1. , -1. ,

These lists of 256 are actually columns in a 256 row table. Each column corresponds to one digit. The training set had 1707 digits, so our initial array was a 256 x 1707 array. A separate array that was 1 x 1707 was an index which had the digit , so we could know what each column was supposed to represent. For example, if position 3 in the 1 x1707 array had a zero, we knew that column 3 of other array represented a zero.

The first step in processing the data was to group like digits into their own array. For example, there were 319 zeros, so we created an array 256 x 319 that contained all the zeros (M0). Next we put all the ones together (M1), etc. We can visualize this by imagining all of the

grids for one digit, stacked up on one another. we Ifhad 319 of one digit, then the algorithm can analyze 319 examples of each square.



At this point, we are ready for SVD.

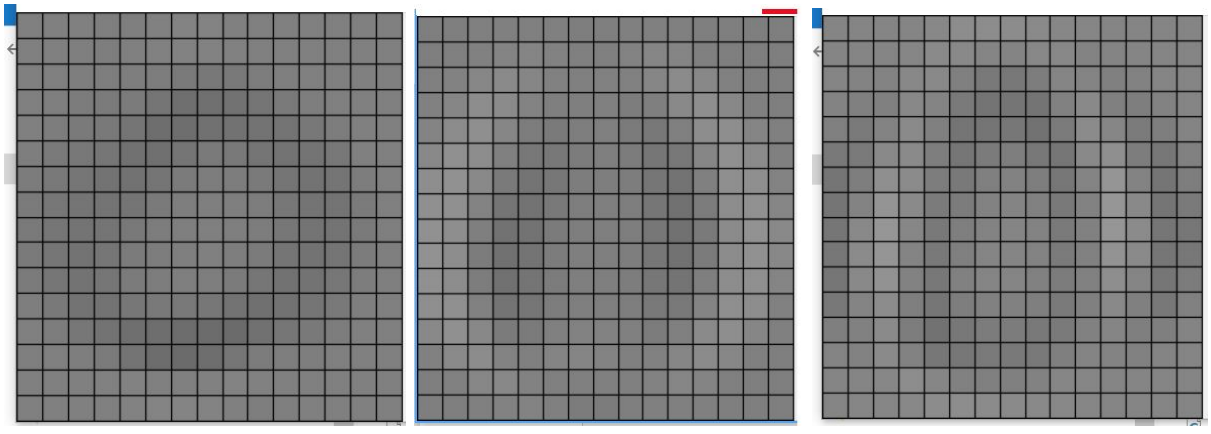
Methodology

Here is an example of the code that measures the distance from an unknown digit to our training data.

```
1 U0 , S ,VT = np.linalg.svd(M0)
2 k = 3
3 smallU0 = U0[:,0:k]
4 d =azip[:,53]
5 I = np.eye(256)
6 v =np.matmul ((I - np.matmul (smallU0 , smallU0.T)) , d)
7 result= np.linalg.norm(v)
```

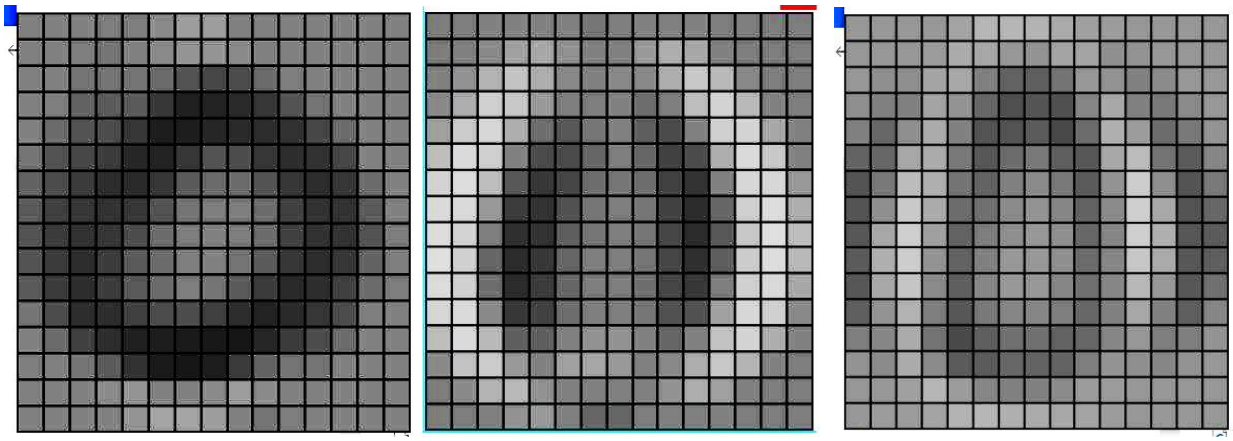
Line 1 uses numpy to perform the SVD, creating three matrices. We are interested in the first matrix, U0. U stands for unitary, 0 stands for the digit zero. U0 is a 256 x 256 matrix. The first column represents the strongest common signal coming from all the matrices put together, the second, the second strongest signal, etc. Below we can see the image generated by just using one column, the second column, and the third column.

First column **Second Column** **Third Column**



After increasing the contrast in an image editor:

First column **Second Column** **Third Column**



So U0 contains our reference values (or classification matrix) from the training data. The next step in line 3 is to take a subset of U0. In this case, we will only use the first three columns of our 256 column matrix. Next we take the norm of the least squares distance between our "unknown" matrix and our small U0. The sum of these distances gives us a number rating how good a match our number is to zero. In this case it was 7.702. Now we repeat the process with matrices for all the other digits. The matrix that produces the lowest score is our number.

Experiments with best k value

The algorithm worked well. The next question was how should we decide how many dimensions of k to use?

Here are some results using different numbers of dimensions on a single image, image 53 from the training set (k=number of dimensions) .

k=1

Matrix	0	1	2	3	4	5	6	7	8	9
Score	12.995	10.594	10.234	11.382	10.086	10.744	10.184	10.666	**9.503**	9.938

k=6

Matrix	0	1	2	3	4	5	6	7	8	9
Score	**7.702**	7.832	9.339	8.77	8.358	8.729	8.338	9.264	8.273	9.108

k=255

Matrix	0	1	2	3	4	5	6	7	8	9
Score	**0.000***	0.000*	0.056	0.051	0.003	0.099	0.025	0.037	0.001	0.058

* These weren't distinguishable until we go out 14 decimal places. Zero was 0.00000000000002 and one was 0.00000000000004.

Results

Note that with k=1, the algorithm would choose nine instead of zero. In general however, k=1 is astonishingly accurate. Here is a list which represents how accuracy improves as k increases.

[1464, 131, 45, 23, 9, 10, 7, 2, 1, 2, 1, 3, 1, 1, 0, 2, 2, 0, 1, 0]

The first number, 1464, tells us that using k=1, we can correctly classify 1464 out of 1707 digits. K=2 gains us an additional 131 correct classifications. K=3 gains us an additional 45 correct classifications, and so. Obviously accuracy improves as k increases, but, it is a trade off with the time it takes to complete. The question is, what is the minimum k we can use that provides an acceptable level of accuracy.

We explored this question by timing how long it took the algorithm to run on our test set data (2007 digits). As the table below shows, we did not see a relationship between k and time.

Accuracy vs. Time

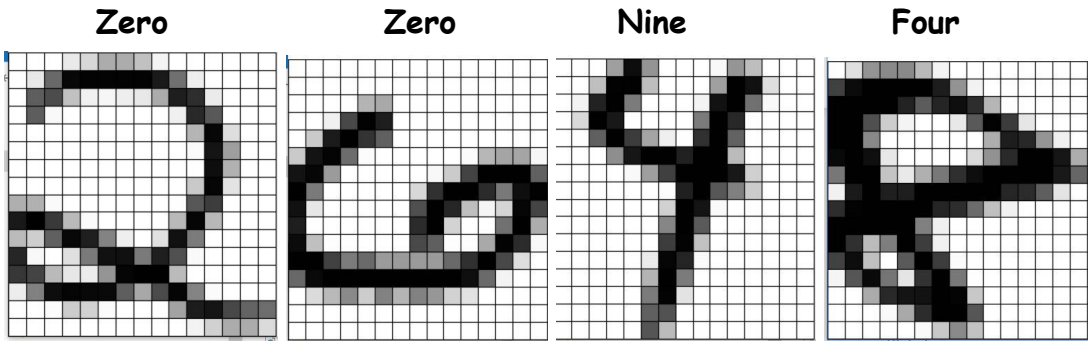
k	1	2	3	4	5	6	7	8	9	10
Accuracy	.773	.840	.859	.887	.875	.875	.895	.895	.914	.912
Time(sec.)	0.49	0.50	0.51	0.75	0.44	0.67	0.89	0.89	0.50	0.46

Apparently this is due to variability from tasks running in the background of the OS. In any case, for larger datasets, or a real-time digit recognition system, common sense tells time would increase k increases.

Conclusion

Our accuracy on the test set using k=7, was 89.5%. This seems good for this challenging data set, although, from the presentations we saw, not as accurate as neural nets. The runtime is also good,. After storing our U matrices, the time to classify all digits in the test set dropped from around 28 seconds to about 0.5 seconds. It is interesting that there is no quantitative way to decide what the value of k is. It was pretty obvious for us though. Looking at the data, we just stopped before adding another dimension of k that only gave us one or two more correct matches. A worthwhile extension of this project would be to automate this process.

Appendix: Examples of mis-classified digits.



Sources

1. Elden, Lars. *Numerical Linear Algebra and Applications in Data Mining,* Preliminary Version, 2005. (unpublished.) The published version is now available on Amazon under a slightly different title: *Matrix Methods in Data Mining and Pattern Recognition (Fundamentals of Algorithms),* Society for Industrial and Applied Mathematics (April 9, 2007). ISBN 978-0898716269.
2. Wikipedia contributors. Singular value decomposition. Wikipedia, The Free Encyclopedia. December 10, 2018, 21:36 UTC. Available at: https://en.wikipedia.org/w/index.php?title=Singular_value_decomposition&oldid=873057667. Accessed December 13, 2018.