CS 391 HW 1: Eigendigits

Joel Walsh

February 7, 2020

1 Introduction

Prior to this class I had only a cursory knowledge of principal component analysis. I had certainly never hardcoded it from scratch, which did give me a more nuanced appreciation for the matrix operations and complexity evident with PCA. So many times in graduate school we implement models in a sort of ”blac-box” manner. I now understand why PCA can take so long. This was especially apprent when I realized that we had to multiply dense matrices and find eigenvectors, two operations with expense.

2 Methods

PCA: I wanted to get a sense of how changes in the number of images used effected computation time. I took a bottum up approach, first attempting to get the process working, and then later iterating through different numbers of images used. I was sure to save the elapsed time, and use pyplot to plot it against the number of training images used. (See Figure 1)

As far as using additional training images, I also compared digits using different numbers of eigenvectors. If I could do it again I would start at much lower increments, because starting at 100 was probably too high. (See Figure 2)

KNN: For K Nearest neighbors I used the scikitlearn KNN package. I first made data sets of five, ten, fifteen, and twenty thousand training images. In the interest of time, my first bit of experimentation was with changing the values of k. I began with iterating through

$$k = 5, 6, 7, 8, 9, 10$$

. (See figure 3)

The next step was to see how adding additional training images affected accuracy and computation times. I made training sets of five, ten, fifteen, and twenty thousand training images and compared reports for computation time and accuracy.

3 Results

PCA Results:

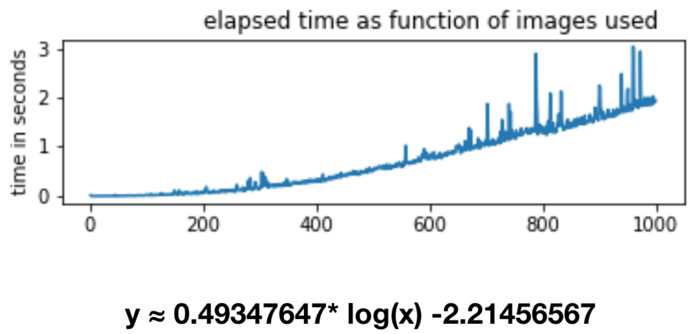

Figure 1: Elapsed time as a function of the number of training images

Using numpy's polyfit and alogx * b, I attained the parameters that you see in Figure 1. This would suggest that using all 60,000 training images would take about minutes, although I suspect that throwing out some outliers might give me a better prediction.

KNN results : I didn't notice a significant gain in precision, or decline in variance per trial as k increased for 5,000 images. (See Figure 3)

The CPU time increased linearly as I added more data. Accuracy did improve a few percentage points, which is fairly significant for most applications. (See Figures 4 and 5)

4 Summary

In terms of PCA, the "trick " to use low rank matrices did pay off quite a bit in terms of being able to use lower rank matrices. For KNN, it was unclear whether or not higher values of k did much of anything, training data held constant. In the future I might do a grid search for values of k and training sets to find best practices for selecting both parameters. For KNN, the computation time was essentially linearly increasing as we added more data. The accuracy payoffs were only a few percent; but if that is the difference between winning or not winning a kaggle challenge, or better detecting cancer cells, that tradeoff is worth it.

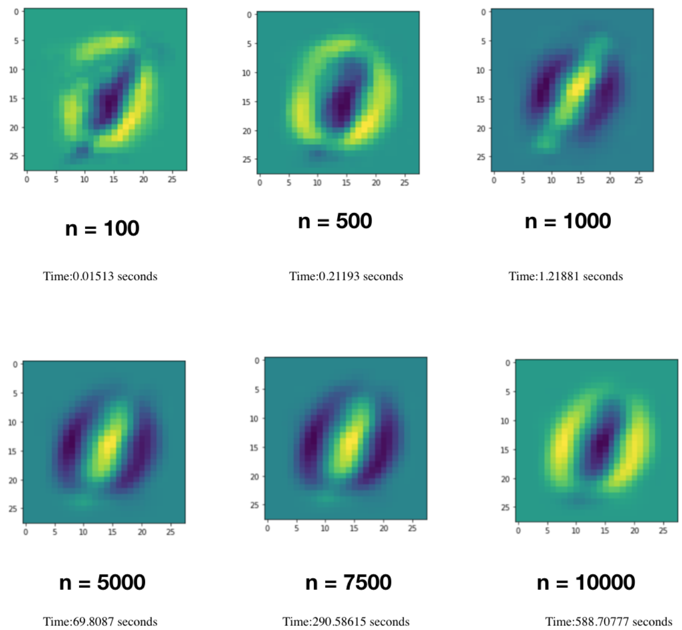

Figure 2: Using different numbers of eigenvectors

	precision	recall	f1-score	support
0	0.94	0.99	0.96	957
1	0.89	0.99	0.93	1116
2	0.98	0.90	0.94	1003
3	0.90	0.94	0.92	995
4	0.96	0.93	0.94	994
5	0.94	0.89	0.92	899
6	0.95	0.97	0.96	997
7	0.92	0.95	0.93	1069
8	0.98	0.85	0.91	989
9	0.91	0.91	0.91	981
accuracy			0.93	10000
macro avg	0.94	0.93	0.93	10000
weighted avg	0.94	0.93	0.93	10000

k = 5

	precision	recall	f1-score	support
0	0.94	0.99	0.96	957
1	0.86	0.99	0.92	1116
2	0.98	0.88	0.93	1003
3	0.89	0.94	0.92	995
4	0.96	0.92	0.94	994
5	0.94	0.89	0.91	899
6	0.95	0.97	0.96	997
7	0.93	0.95	0.94	1069
8	0.98	0.84	0.90	989
9	0.90	0.92	0.91	981
accuracy			0.93	10000
macro avg	0.93	0.93	0.93	10000
weighted avg	0.93	0.93	0.93	10000

k = 10

Figure 3: Using different numbers values of K in K Nearest Neighbors

	precision	recall	f1-score	support
0	0.94	0.99	0.96	957
1	0.89	0.99	0.93	1116
2	0.98	0.90	0.94	1003
3	0.90	0.94	0.92	995
4	0.96	0.93	0.94	994
5	0.94	0.89	0.92	899
6	0.95	0.97	0.96	997
7	0.92	0.95	0.93	1069
8	0.98	0.85	0.91	989
9	0.91	0.91	0.91	981
accuracy			0.93	10000
macro avg	0.94	0.93	0.93	10000
weighted avg	0.94	0.93	0.93	10000

CPU times: user 1min, sys: 372 ms, total: 1min
Wall time: 1min 1s

5000 images

	precision	recall	f1-score	support
0	0.96	0.99	0.97	957
1	0.91	0.99	0.95	1116
2	0.98	0.93	0.95	1003
3	0.93	0.96	0.94	995
4	0.97	0.95	0.96	994
5	0.96	0.94	0.95	899
6	0.96	0.97	0.96	997
7	0.94	0.95	0.95	1069
8	0.98	0.88	0.93	989
9	0.93	0.94	0.93	981
accuracy			0.95	10000
macro avg	0.95	0.95	0.95	10000
weighted avg	0.95	0.95	0.95	10000

CPU times: user 2min 2s, sys: 759 ms, total: 2min 3s
Wall time: 2min 4s

10000 images

Figure 4: Using different numbers of training data

	precision	recall	f1-score	support
0	0.97	0.99	0.98	997
1	0.94	0.99	0.97	1115
2	0.97	0.95	0.96	1026
3	0.96	0.94	0.95	1030
4	0.97	0.95	0.96	1021
5	0.95	0.95	0.95	919
6	0.97	0.99	0.98	993
7	0.95	0.96	0.95	998
8	0.98	0.90	0.94	954
9	0.91	0.95	0.93	947
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

CPU times: user 2min 53s, sys: 1.57 s, total: 2min 54s
Wall time: 3min 7s

15000 images

	precision	recall	f1-score	support
0	0.98	0.99	0.98	997
1	0.94	0.99	0.97	1115
2	0.98	0.95	0.97	1026
3	0.96	0.94	0.95	1030
4	0.98	0.96	0.97	1021
5	0.95	0.96	0.95	919
6	0.97	0.99	0.98	993
7	0.95	0.96	0.96	998
8	0.98	0.92	0.95	954
9	0.93	0.95	0.94	947
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

CPU times: user 4min 8s, sys: 2.33 s, total: 4min 10s
Wall time: 4min 19s

20000 images

Figure 5: Using different numbers of training data