Christopher Wong
999234204

MAT 167 Programming Project **5/27/2016**

Introduction

Image and pattern recognition is an important topic in today's digitalized world. It's importance comes from how it can applied to automate trivial tasks that can be performed by people, but could be viewed as an inefficient use of their time such as digit or facial recognition. In this programming project, we compare the effectiveness and accuracy of two algorithms: a simplest algorithm and a SVD basis algorithm. We apply both algorithms to identify handwritten digits with the goal of being able to automate the task of identifying different digits given our data from the United States Postal Service (USPS). An advantage to the automation of this task is that while people are adept at recognizing digits, it is mundane and we are prone to make mistakes after performing the task for some length of time. A computer on the other hand, will not only be able to do this task (a lot) quicker, but with a predetermined accuracy. Thus, if we are able to find such an image recognition algorithm that performs this task fairly well, then we are able to do a lot more in a shorter span of time. We explore two such algorithms below.

Step 1: The Data

In this programming project, we are given a handwritten digit database "USPS.mat" which contains 4 arrays.

- train_patterns is of size 256 × 4649 and contains the patterns of our digits from our training set
  Contains a raster scan of the 16×16 gray level pixel intensities that have been normalized to lie within the range [-1,1]
- test_patterns is of size 256 × 4649 and contains the patterns of our digits from our test set
  Contains a raster scan of the 16×16 gray level pixel intensities that have been normalized to lie within the range [-1,1]
- train_labels is of size 10 × 4649 and contains the classification of our digits from our training set
  Contains the true information about the digit images
- test_labels is of size 10 × 4649 and contains the classification of our digits from our test set
  Contains the true information about the digit images

The difference between training and test data is that training data has the correct/expected output which is because it is often collected carefully and checked by humans. On the other hand, test data is the data we want to apply our model to after we have created the model using the training data. In general, we do not know the correct output of our test data, but in our case we do; it is stored in test_labels. But, taking a look at test_patterns we notice that these digits are a lot messier than the ones in train_patterns. Hence we are going to test our two different algorithms on these "messy" digits after training them on the "clean" digits. This is an example of *machine learning* where we first train our model/algorithm on nice, clean data before testing them on messier data. Below are the first 16 images from our training data:

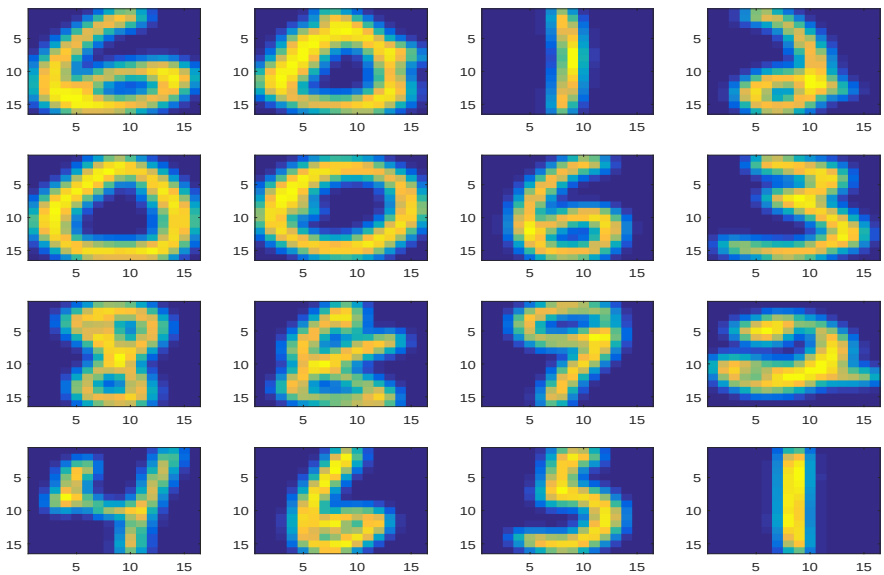

FIGURE 1. First 16 images in train_patterns

Step 2: Mean Digit Image

In this step, we find the mean digits from our training data. That is, for each digit from 0 to 9, we find all columns corresponding to each digit from our train_patterns matrix and take the mean of the row. Each row corresponds to the gray level pixel intensity of that specific pixel out of our 256 pixels. So, when we average over each row for the digit 1 for example, we are finding the average gray level pixel intensity of each pixel for each image of 1 in our training data. In essence, we are finding the gray level pixel intensities that correspond to what the mean digit 1 looks like from our training data (which is correctly labeled). We repeat this for the other digits and the results are below:

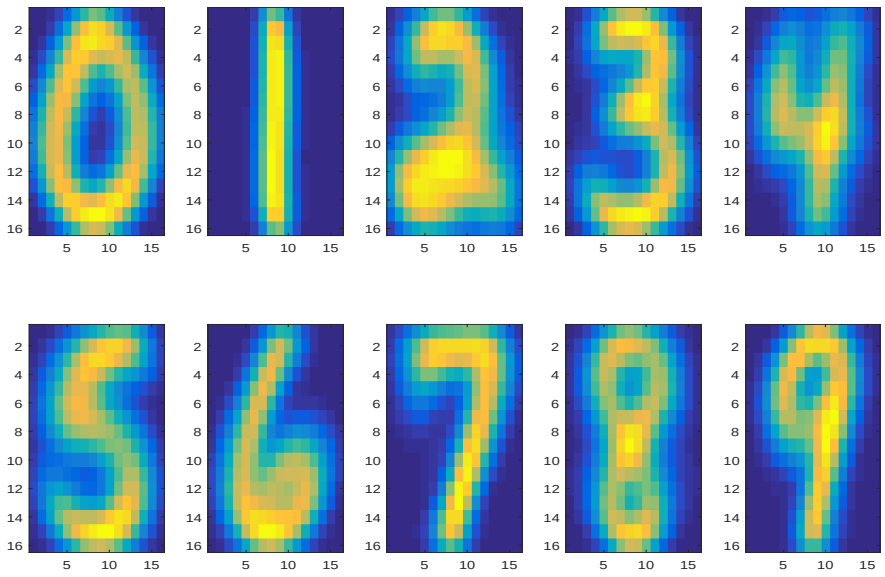

FIGURE 2. Mean Digit Images

Steps 3/4: Algorithm Effectiveness

At the end of steps 3 and 4, we have our two confusion matrices, one for each algorithm. We will begin with a few tables and then analyze our findings.

TABLE 1. Simplest Algorithm Confusion Matrix

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	656	1	3	4	10	19	73	2	17	1
1	0	644	0	1	0	0	1	0	1	0
2	14	4	362	13	25	5	4	9	18	0
3	1	3	4	368	1	17	0	3	14	7
4	3	16	6	0	363	1	8	1	5	40
5	13	3	3	20	14	271	9	0	16	6
6	23	11	13	0	9	3	354	0	1	0
7	0	5	1	0	7	1	0	351	3	34
8	9	19	5	12	6	6	0	1	253	20
9	1	15	0	1	39	2	0	24	3	314

Digit	0	1	2	3	4	5	6	7	8	9
Accuracy	0.9111	0.8932	0.9118	0.8783	0.7658	0.8338	0.7884	0.8977	0.7644	0.7441

TABLE 2. SVD basis Algorithm Confusion Matrix

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	772	2	1	3	1	1	2	1	3	0
1	0	646	0	0	0	0	0	0	0	1
2	3	6	431	6	0	3	1	2	2	0
3	1	1	4	401	0	7	0	0	4	0
4	2	8	1	0	424	1	1	5	0	1
5	2	0	0	5	2	335	7	1	1	2
6	6	4	0	0	2	3	399	0	0	0
7	0	2	0	0	2	0	0	387	0	11
8	2	9	1	5	1	1	0	0	309	3
9	0	5	0	1	0	0	0	4	1	388

Digit	0	1	2	3	4	5	6	7	8	9
Accuracy	0.9797	0.9458	0.9840	0.9525	0.9815	0.9544	0.9732	0.9675	0.9656	0.9557

We find that the digit 9 is the hardest and 2 is the easiest to identify using the simplest algorithm. This may be because if we take a look at our mean image for the digit 9, it can be seen that other digits such as 7 or 8 is also a close match to some degree. The simplest algorithm performs poorly because it does not take into account any variation within each class of digits, e.g. some of our 1's may have a tail at the top leading it to be misidentified as a 7. It is not surprising that 2 is the easiest to identify because there are no digits that look similar to 2.

On the other hand, for our SVD basis algorithm, we find that 1 is the hardest and 2 is the easiest to identify. This seems to be a result of how the SVD basis algorithm takes into account variation within each class of digits because not everybody writes the digit 1 the same way. Some people may put a tail at the top of the digit and the SVD basis algorithm will accommodate for this variation when creating the class of 1's. This leads to our SVD basis algorithm to sometimes misidentify a 7 as a 1 for example because the tails at the top are similar. It is not surprising that 2 is the easiest to identify because there

are no digits that look similar to 2 and variations between 2's is expected to be low since from personal experience everybody writes 2 the same way.

Conclusion

In conclusion, we find that our SVD basis algorithm performs significantly better than our simple algorithm. Comparing their accuracies, the lowest accuracy for the SVD basis algorithm is 94.58% which is higher than the best accuracy for our simple algorithm 91.18%. So, even at its worst, the SVD basis algorithm outperforms the simplest algorithm and is clearly superior to the simple algorithm beating it by over 20% accuracy on some digits such as 4.