USPS Handwritten Digits Analysis

I. Loading the USPS handwritten digits dataset. (256x1100x10)

II. Studying the role of the split of the training vs the testing data for classification.

a. *The selection algorithm*. To start-off, I have decided to split my data 1000:100 Training:Testing. Each digit class will be divided 1000 to 100, where the first 1000 instances/examples will be stored in the training set and the rest 100 will be stored in the testing set. The reason for such split is because I would like to have as much data as possible for training purposes, and since, N ranges from 100 to 1000, I have decided to go for max, i.e., N_1=1000.

b. *Computational cost for this project*.

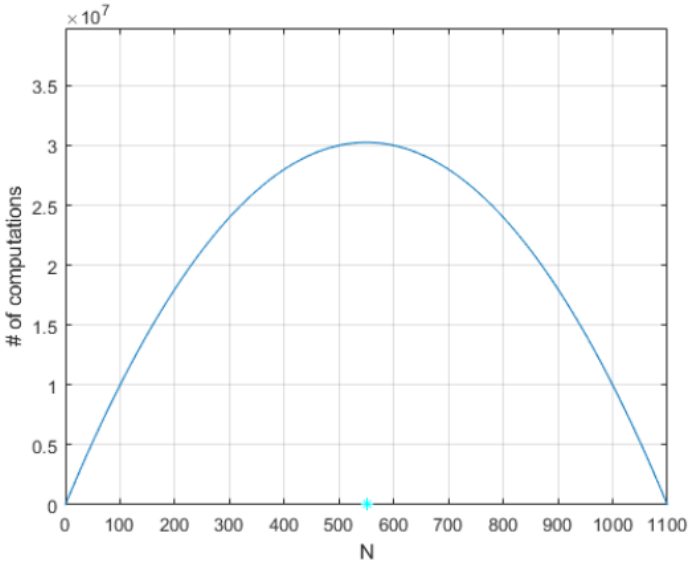
i. Note that both training and test sets are three-dimensional matrices, of size 256x1000x10 and 256x100x10, respectively. To simply my 'distance' computations, I converted both the matrices from 3D to 2D, hence the new sizes are, 256x10000 and 256x1000, respectively, where each column in the training set represents a digit instance/example (1-1000:digit1; 1001-2000:digit2…9001-10000:digit0), and each column in the testing set is an observation (or a query point) that needs to be classified based on kNN.

ii. Hence, the total number of distance computations needed will be $f(N_1) = f(1000) = 10000x1000 = 10^7$ computations (distances) to find the k=20 Nearest Neighbors for each query point. Thus, the formula for total computations is $f(N) = 10(N)*10(1100-N)$, where N is the size of the training set. In order for us to minimize the total computations, we would have to find an N that produces least number of computations. Using 1st derivative test, we see:

Table 2.1

	$-\infty < x < 550$	$x = 550$	$550 < x < \infty$
Sign	$f'(x) > 0$	$f'(x) = 0$	$f'(x) < 0$
Behavior	Increasing	Maximum	Decreasing

Fig. 2.1



f(N) is maximum when N=550, and then f(N) decreases as N increases.

iii. Thus, the training set must be either as big as possible or as small as possible for f(N) to be low. Considering that and the fact that I would like my training set to be bigger than the test set, I choose my next value of N to be 1050, for which $f(N_2) = 5.25 \times 10^6$ computations. Hence, I have N_1=1000 (less optimal) and N_2=1050 (more optimal w.r.t cost of computations, discussed further in part-d in terms of total time it took to find the nearest neighbors).

c. *Global success rates for both N_1 and N_2.*

i. Global success rate for N_1=1000 → 86.84%

a. 20 nearest neighbors were found for each of the 100 query points for each of the 10 digits (total of q=1000, and 20000 nearest neighbors). For these 20000 nearest neighbors, model assigned a total of 2632 incorrect digits as nearest neighbors, hence, error_rate = 13.16%

ii. Global success rate for N_2=1050 → 85.80%

a. Following the similar process as N_1, we found 10000 nearest neighbors for q=500 query points (50 for each of

the 10 digits). This time, model assigned 1420 incorrect digits as nearest neighbors, hence, error_rate = 14.20%

d. *Impact of training-set size*.

i. As discussed in part b, the function f(N) gives us the total number of computations needed for a given training set of size N for each digit. After minimizing f(N), we concluded that the bigger our N is, the lesser number of computations we will need to do. Hence, I chose N_2=1050, for which, *as expected*, it took just 2.1 seconds to find the 20 nearest neighbors for our test set, whereas, for N_1=1000, it took 4.6 seconds to carry out the same process. On average, I have noticed about 41%-45% decrease in computation time. Thus, we can confirm that smaller testing-set and larger training-set means much faster computations.

ii. Although, the global success rate dropped by 1% as our training set size increased from 1000 to 1050 (for each of the digits). Hence, to choose an optimal size of the training set, there is going to be a time_complexity–global_success_rate tradeoff.

iii. Since the drop in computation time was significantly large than the minimal dip in global success rate, I choose N_2=1050 as the optimal training size.

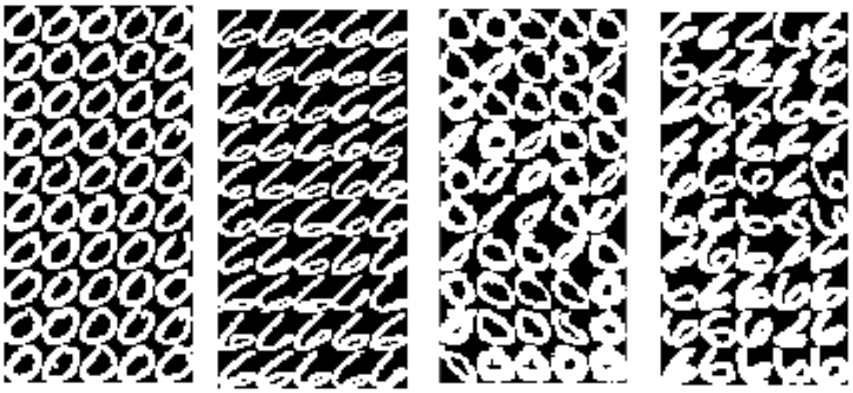
III. Studying the role of the structure of the split of the training vs testing data for classification.

a. *A new selection algorithm*. Considering the optimal value of N (=1050) for my new method of selecting the training set, I have decided to choose the 1050 nearest neighbors from the mean of each digit as the training data, and hence, the farthest 50 instances/examples from the mean vector of each digit will be included in the test data. Such a split is a great way to check how the model would perform in the worst-case scenario since the best examples of each digit will be used to train and the model will be tested on the worst handwritten digits. This is explained visually with examples in part-b of this section.

b. *Splitting the dataset into training and testing set using the new algorithm*.

i. The dataset is split with N=1050. Thus, the size of the training set is 256x1050x10, and the test set is 256x50x10. Below in the figure 3.1 are the examples of the how training-set and test-set look like for the digits 0 and 6. First pair of pictures is of 50 instances (out of 1050) of digit-0 and digit-6 that are in the training-set, whereas the second pair consists of all 50 instances of digit-0 and digit-6 that will be tested and classified using kNN.

Fig 3.1



c. *Global success rate with the new selection algorithm*.

i. Global success rate for optimal N → 73.46%

a. The success rate coincides with what I expected. 10000 nearest neighbors were found for q=500 query points (50 *worst* examples for each of the 10 digits). Model assigned 2654 incorrect digits as nearest neighbors, hence, error rate = 26.54%

d. *Impact of the structure of the train-test split using new algorithm*.

i. The structure of the split made sure that the best 1050 examples of each digit are used as the training data and the farthest examples from the mean vector (poorly written digits) were classified. Hence, based on the structure I chose, I was not expecting a better global success rate than what we had in section II, instead I was curious know how the model

would perform in the worst case when poorly written digits are fed to the model, if deployed into production environment.

ii. Comparing the structure of the split from this section to the previous section (where the first 1050 instances of each digit were picked for the training-set and the rest 50 for the test-set), we noticed how global success rate dropped from 85% to 73%. The train-test split from the previous section remains the best way to split the data. Thus, for the following sections, I will continue using the train-test split structure from section II.

IV. Analyzing the role of the metric in the classification process using the optimal value of N and the best training set selection method.

a. *Euclidean distance(L_2) vs Manhattan distance(L_1)*.

i. For the optimal N=1050, where the best structure of the split was simply choosing the first 1050 examples/instances for each of the digits as the training set and the rest 50 as the test set, we have our results in the following table:

Table 4.1

Metric	Euclidean (L_2)	Manhattan (L_1)
Incorrect Neighbors (globally)	1420	1594
Global Success Rate	85.80%	84.06%
Global Error Rate	14.20%	15.94%

ii. Both global success rates are relatively close to each other, yet L_1 metric 'incorrectly' assigned 174 query points more than the L_2 metric.

iii. Alongside Euclidean distance, one of the reasons Manhattan distance works quite well with this dataset is due to the curse of dimensionality. Since the datapoints are in a form of a grid.

iv. Note that, in $||.||_p$, the higher the value of p, the greater the emphasis on large values. $||.||_2$ ignores the small values. Hence, it is more sensitive to outliers than $||.||_1$. Since, for my model, Euclidean distance

metric results in a slightly better success rate, it might indicate that the outliers in my dataset are almost non-existent.

V. Analyzing the success percentage differences between individual digits in each case and drawing conclusions.

Digits	Selection Algorithm I (L_2)	Selection Algorithm I (L_2)	Selection Algorithm II (L_2)	Selection Algorithm I (L_1)	Mean %
Value of N	1000	1050 (optimal)	1050	1050	
1	99.70%	99.50%	97.50%	**99.80%**	99.13%
2	81.15%	77.50%	73.90%	75.90%	77.11%
3	91.40%	93.20%	70.70%	91.80%	86.78%
4	77.25%	76.30%	71.40%	75.60%	75.14%
5	88.50%	87.90%	64.20%	88.00%	82.15%
6	90.75%	87.70%	83.10%	86.50%	87.01%
7	92.45%	93.80%	71.60%	91.00%	87.21%
8	72.45%	69.30%	**49.00%**	63.50%	63.56%
9	79.65%	79.20%	76.50%	75.10%	77.16%
0	95.10%	93.60%	76.70%	93.40%	89.70%
Global Success Rates	86.84% Euclidean dist.	85.80% 45% faster calc.	73.46% worst-case scenario	84.06% Manhattan dist.	

Table 5.1

The major takeaway from the table is, regardless of the chosen N or the structure of the split, digit-1 had the best individual success-rate compared to others, where using a combination of the selection algorithm from Section I and $\|.\|_1$, success-rate for digit-1 reached with 99.80%. On the other hand, digit-8's success-rate was poor for all combinations, where the worst-case scenario test produced a success-rate of just 49%, meaning a total of 510 wrong nearest neighbors out of 1000 were assigned to 50 instances of digit-8. Considering the numbers discussed, we can say that digit-1 is the most uniformly well-written digit in the USPS dataset whereas digit-8 is the most poorly written digit.