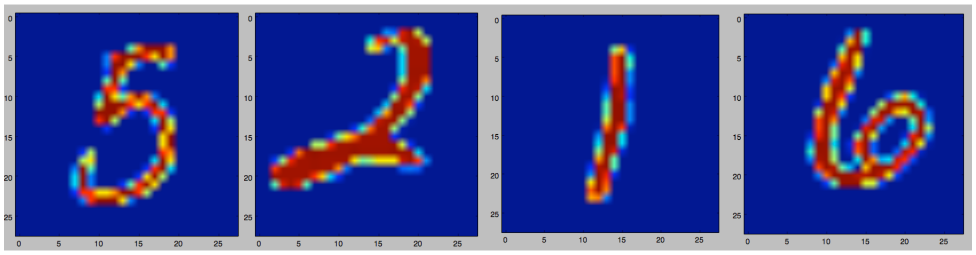
Bit manipulation KNN on mnist

Xiangyue Zheng G42416206

Dataset samples



(5) (2) (1) (6)

We have total 42000 data items in the data set. Since there is no training procedure. Let’s divide the dataset into two parts: “the model data”(90%) itself and “test data”(10%)

Data preprocessing

Quantize data

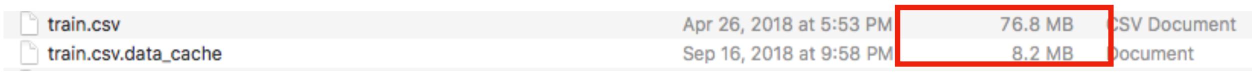
As for this specific problem, an intuition to process the data is to normalize each pixel. We really don’t care much “how bold or how light” we are writing this digit at a specific position. (e.g. In general the edge of the digit has small pixel value) Rather we may only concern 、if a pixel has ink on it.

So I **normalize each pixel to 0 if the original pixel value is less than 100 and 1 otherwise.**

It’s good to have only 0 and 1 in the image pixel array because the distance of the two arrays can be **calculated by count how many numbers are different, which could be done with bit manipulation.**

[28 * 28] matrix => 784 bit integer

The benefits are obvious, we could save a mount of space and time to store/access data.

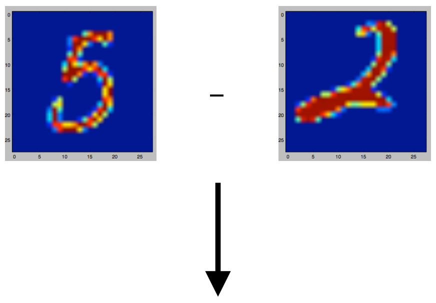


train.csv	Apr 26, 2018 at 5:53 PM	76.8 MB	CSV Document
train.csv.data_cache	Sep 16, 2018 at 9:58 PM	8.2 MB	Document

Algorithm Description

Since image data is stored in one bit stream of length 784, the responding KNN algorithm might be slightly different.

We notice that each pixel(bit)'s difference could be only 1 or 0. The distance of two bitstreams is the number of digit 1 in the binary number of the "exclusive or" result

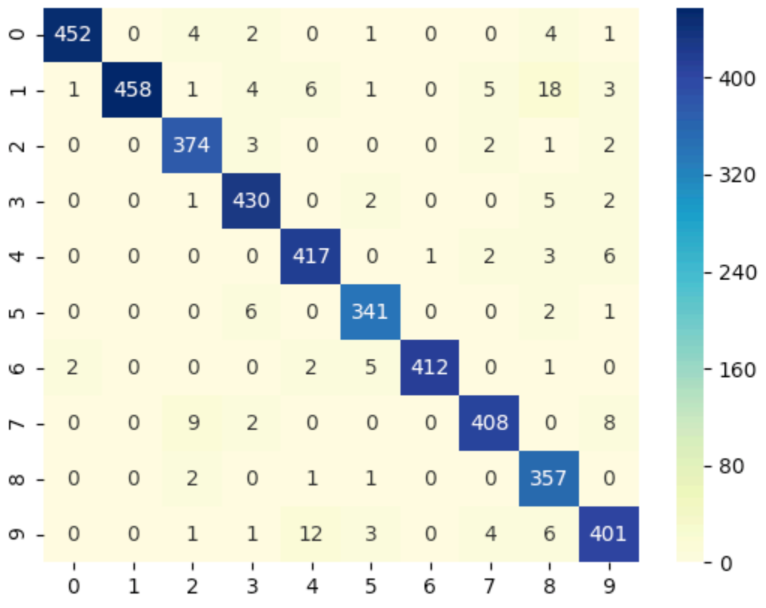


numOfDigit1(00000..1010101..10101000 xor 00..1010010..101010000)

Algorithm Results

Confusion matrix when k = 5

Predict label



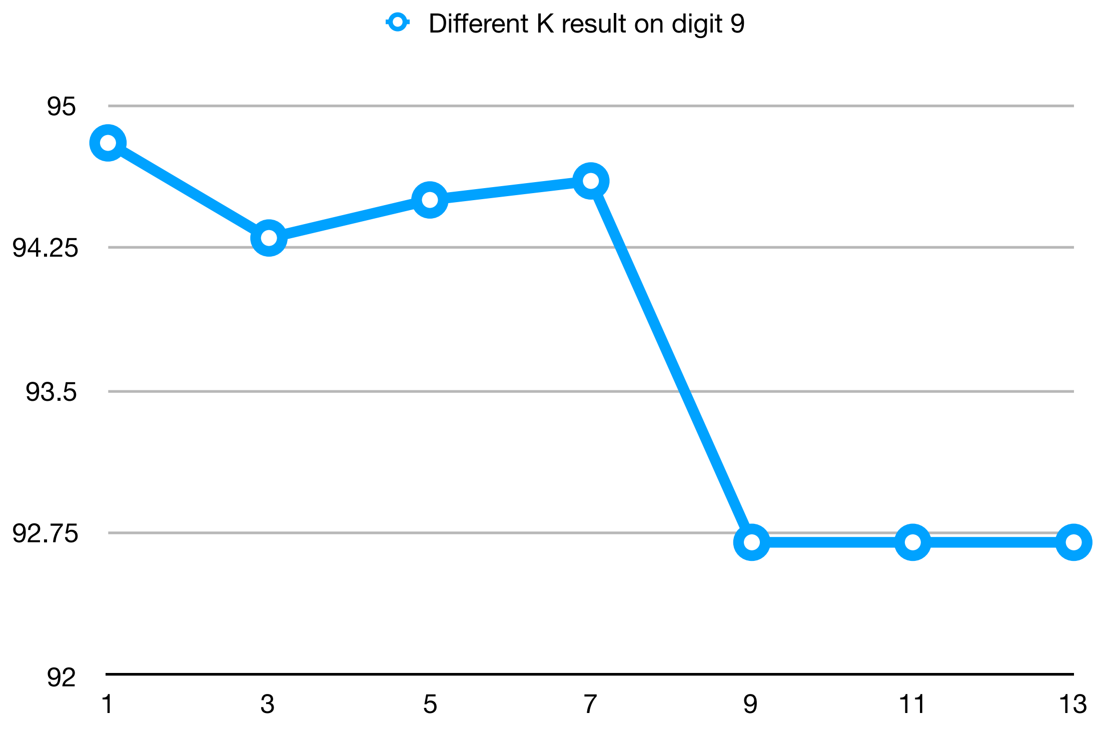
	0	1	2	3	4	5	6	7	8	9
0	452	0	4	2	0	1	0	0	4	1
1	1	458	1	4	6	1	0	5	18	3
2	0	0	374	3	0	0	0	2	1	2
3	0	0	1	430	0	2	0	0	5	2
4	0	0	0	0	417	0	1	2	3	6
5	0	0	0	6	0	341	0	0	2	1
6	2	0	0	0	2	5	412	0	1	0
7	0	0	9	2	0	0	0	408	0	8
8	0	0	2	0	1	1	0	0	357	0
9	0	0	1	1	12	3	0	4	6	401

Real test label

As shown above, the accuracy of the 4200 test data is 96.6%

The calculation takes 835 seconds.

Now let's consider the effectiveness of “K”. If we consider only digit 9 in the test set, the relation of accuracy and k is shown bellow.



I also choose different K value on other digits and it shows that when K <=7, the result is relatively good.

And for this specific problem, each feature dimension is equally contribute to the result causing that K=1 have a good result.

Summarizing

In this experiment, I came up with a fast computing method using bit calculating. Though the process seems a little different from traditional KNN method, the thought behind these are the same.

Another thing is I would love to dig out how some of the KNN library works and see how they compute so fast..

In general, the result is satisfying for such a simple method.