Linnéuniversitetet
Kalmar Växjö

Report

# Assignment 1

*K-Nearest neighbors*

*Author*:
Findlay Forsblom ff22ey
*Termin*: VT19
*Ämne*: Machine Learning
*Kurskod*: 2DV516

# Tables of contents

**Tables of contents** **2**

**Task 3 (VG-Task)** **3**

Results 3

# Task 3 (VG-Task)

In this task we were asked to use our own k-NN classifier to classifier handwritten digits using the MNIST dataset. The goal of the classifier was to be as good as possible on recognizing handwritten digits.

Since each image consist of 28 x 28 pixels, we would like to get it as a column vector instead in order to be able to build our X and y arrays. After some googling and forum reading i found out that the technique used to be able get an image which is represented as an 28 x 28 matrix to a single vector is through row or column flattening by going column by column or row by row and putting the values in a vector. I later also found a library that did that for me, and also helped converted the files in a numpy X, y array as i wanted it. It was the *mlxtend.data* library and from it the *loadlocal_mnist* module.

After some googling and reading different forums i chose to use the *mlxtend.data* library and it i chose to use the *loadlocal_mnist* to be able to read the file files in a numpy array.

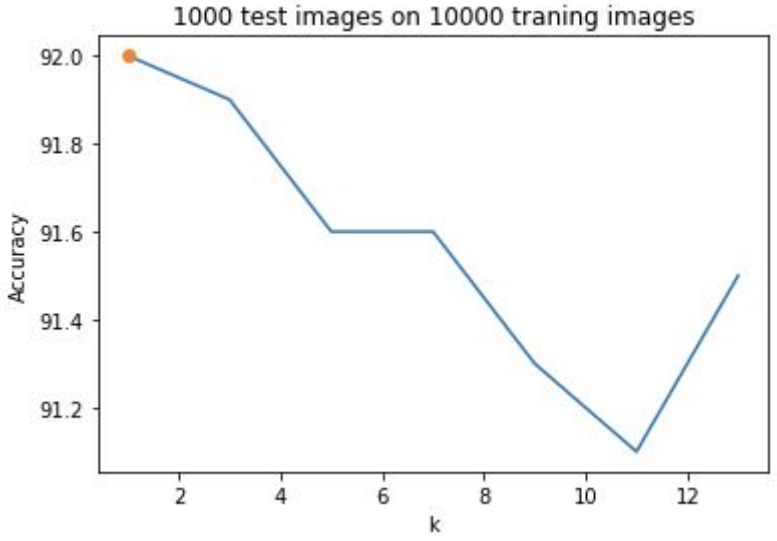
Since there are 784 pixels, the trick here is to compute the square sum of each pixel of a test image to each pixel of all the images training set and takes the image that is it's k's nearest neighbor. Down below are the results that i got

## Results

To begin with i tested 1000 test images on a training set of 10000 using the following number distributions

```
number of 0 in the training set 1001
number of 1 in the training set 1127
number of 2 in the training set 991
number of 3 in the training set 1032
number of 4 in the training set 980
number of 5 in the training set 863
number of 6 in the training set 1014
number of 7 in the training set 1070
number of 8 in the training set 944
number of 9 in the training set 978
---------------------------------
number of 0 in the test set 85
number of 1 in the test set 126
number of 2 in the test set 116
number of 3 in the test set 107
number of 4 in the test set 110
number of 5 in the test set 87
number of 6 in the test set 87
number of 7 in the test set 99
number of 8 in the test set 89
number of 9 in the test set 94
```

i got the following results




```
best k is 1 with 92.0% accuracy
```

when testing with different ks from 1 - 15 and using only odd numbers, for some strange reason k = 1 gave the best accuracy. This took around 2.35 minutes to complete in total, so around 22,5 seconds for each k (at least on my computer).

I then tried testing 1000 test images on the full training set, meaning all 60000 training images using with the following number distribution in the test set.

```
number of 0 in the test set 85
number of 1 in the test set 126
number of 2 in the test set 116
number of 3 in the test set 107
number of 4 in the test set 110
number of 5 in the test set 87
number of 6 in the test set 87
number of 7 in the test set 99
number of 8 in the test set 89
number of 9 in the test set 94
```

This were the results that i got


As can be seen from the pictures above, the best k here as well was when k was 1, 3 and 7 which all gave an accuracy of 96.2%. It also seems like the accuracy increases as the number of training data increases. This test took around 5 minutes to complete for each k.

Finally i tried testing all 10,000 test images on the full training set with the same amount of ks as the previous test but it took way too long. I then stopped it and tried testing a single k when k was 5 and just that alone took 47,5 minutes with an accuracy of 96.9%. Assuming that the time is linear it would have taken 5.5 hours to complete all 7 other k's(1, 3, 7, 9,11,13,15)