SYDDANSK UNIVERSITET

K-nearest neighbor

Bjarki Sigurdsson
Abdulrahman Abdulrahim
Miguel de la Colina

February 20, 2017

ABSTRACT

THE PURPOSE OF THIS REPORT IS TO USE THE K-NEAREST NEIGHBOR ALGORITHM ON A DATA-SET MADE OF CIPHERS. WE WILL BE DIVIDING THE DATA INTO TWO SETS, ONE FOR TRAINING AND ONE FOR TESTING. WE WILL ANALYZE THE RESULTS FOR DIFFERENT VALUES OF K AND DPI TO SEE HOW THIS ALTERS OUR RESULTS. ALSO WE WILL BE APPLYING THE GAUSSIAN SMOOTHING WITH VARIOUS SIGMAS IN ORDER TO SEE HOW DOES THIS ALTER THE RESULTS.

1 K-NEAREST NEIGHBOR

We will be using R which is a statistical language in order to test the k-nearest neighbor algorithm, firstly we will have to generate our data-sets which are taken from scanned ciphers and loaded through the function `loadSinglePersonsData`, once the data is loaded we shuffle the data with a seed for reproducible results. With this done we split the data into test and train so that we are able to test the data after we have trained and be able to use different data from the one that was trained.

For this test we will be varying the number of k to see how the result actually varies when we begin to change it's value we will check how the speed and test recognition are affected by this change. This is to see how important really is to select the correct k and to see if having selected the wrong one could affect your results substantially.

We will also be doing cross validation of the results in order to see if the results of the trained model will fit for other hypothetical, set of data this will be done by running 10 times a 90%/10% split of the data-set. The pseudocode for the cross-validation is shown below.

```
M_xval <- list()
for (i in 1:10) {
  # Split matrix into 10 parts
  M_xval[[i]] <- M_shuffled[((i-1)*nrow(M)/10+1):(i*nrow(M)/10),]
}
for (i in 1:10) {
  # Recombine 9 parts for training and keep 1 for testing
  M_xval_test <- M_xval[[i]]
  M_xval_train <- do.call(rbind, M_xval[-i])
  true_class_xval <- M_xval_train[,1]
  class_xval = knn(M_xval_train, M_xval_test, true_class_xval, k_it)
  true_class_xval <- factor(true_class_xval, levels(class_xval))
  success_xval <- sum(true_class_xval == class_xval)/length(class_xval)
  cat("Result", i, ":", success_xval, "\n")
}
```

Finally after testing it with the smoothing implementation that was in `loadImage.r` we have to implement the smoothing using a different method. We used the Gaussian smoothing with various sigmas, this is implemented in the EBImage package as `gblur()` which receives as parameter the image and the sigma.

RESULTS

RESULTS WITH DPI=100

DPI=100			
K	Training Set acc.	Test Set acc.	Time
1	1	0.9995	3.853
10	0.9945	0.9945	3.334
25	0.991	0.986	3.386
50	0.984	0.986	3.395
100	0.9865	0.989	3.807

RESULTS WITH DPI=200

DPI=200			
K	Training Set acc.	Test Set acc.	Time
1	1	0.9995	3.419
10	0.9945	0.9945	4.170
25	0.991	0.986	3.393
50	0.984	0.986	3.419
100	0.9865	0.989	3.58

RESULTS WITH DPI=300

DPI=300			
K	Training Set acc.	Test Set acc.	Time
1	1	0.9995	5.011
10	0.9945	0.9945	3.484
25	0.991	0.9875	4.956
50	0.984	0.986	3.779
100	0.9865	0.989	4.087

We see that the accuracy does vary with k. Notably, k=1 results in a perfect fit for the training set but this will often, in theory, result in overfitting of the data. Thus we use the rule of thumb of taking the square root of the sample size as the k and approximate it to k=50 for further testing.

Though the timing measurements presented in the table are quite noisy, we see that the computation time seems to increase with DPI. This is logical as the number of pixels in the images has a square relationship to the DPI. Below we will present more accurate timing measurements and discuss their dependence on k.

For more accurate timing measurements, we ran the `knn()` function repeatedly for k=1 and k=100 and compared the results. for DPI=300, we obtained mean computation times of 3.98s and 4.75s for k=1 and k=100, respectively. This is to be expected as for larger k, each point needs to be checked against more neighbours, resulting in more computations.

We performed cross-validation on the data with our chosen k for varying DPI values. In general, we found the accuracy of the kNN algorithm to be independent on the image quality, at least for the DPI values tested. Thus only results for DPI=100 are shown below. We obtained a mean accuracy of 0.880 with a standard deviation of 0.056.

CROSS-VALIDATION K=50 DPI=100

DPI=100 k=50	
Number	Result
Result 1	0.8675
Result 2	0.9225
Result 3	0.8525
Result 4	0.9525
Result 5	0.915
Result 6	0.8275
Result 7	0.82
Result 8	0.7975
Result 9	0.9525
Result 10	0.8875

For our tests with image preprocessing we obtained the results shown below for different sigmas in the gaussian low-pass smoothing filter. We see that the smoothing seems to increase the accuracy of the algorithm to some degree.

RESULTS FOR GAUSSIAN SMOOTHING

DPI=100			
sigma	Training Set	Test Set	Time
.1	0.6405	0.6235	66.464
.5	0.7235	0.702	66.900
1	0.8215	0.828	66.404
2	0.932	0.941	66.231

Finally we ran the first test on the entire data set. Cross-validation as well as accurate timing was not performed for this part due to a lack of computation power. This was also the reason we only tested with k=50. k=200 may have been more accurate as the data set was 18 times larger than in previous parts.

We obtained accuracy results of 0.9954 on the training set and 0.9949 on the test set. We timed the execution time of the `knn()` function and found it to be 20 minutes and 35 seconds. From these results, in comparison to the previous, one can argue that the benefit of having a larger data set is greater than the variance introduced by different handwritings. Moreover, the variance in handwriting in the data may even be beneficial to the accuracy of the algorithm.