Exercise_1_KNN

1.4 Performing K-nearest neighbor (Report 1) Group 4

In this report, we perform the k-nearest neighbour algorithm to classify handwritten digits data. We analyse the performance of the algorithm changing the sample size, subject dependencies and most importantly the parameter K - minimal number of closest neighbours needed to properly classify the sample.

1.4.1 K-Nearest Neighbour:

Using the methods learned in the Chapter 3 in "Machine Learning WithR", KNN can now be performed on our own generated dataset. First we will test on a single person. Remember to split between training and test set (split in two equally sized parts). Document the results. Can you explain the performance (computation time and test accuracy) on the training and test-set?

```
 accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}

load("data/id100.Rda")
set.seed(423)

k_error_avg_vec <- c()
timing_now_avg_vec <- c()
ks <- c( 19)

for (k_now in ks ){
  #print(paste0("k_now = ", k_now))
  k_error <- 0
  k_error_avg <- 0

  timing_now_avg <- 0
  for (i in 1:10) {
    # shuffle dataset
    shuffled_df <- id[sample(nrow(id)),]

    # split 10/90 for training and testing data
    test_df <- shuffled_df[1:200,]  # shuffled_df[1:360,]
    train_df <- shuffled_df[201:400,] # shuffled_df[361:400,]

    # get the labels - supervision part
    id_train_labels <- train_df[,1]
    id_test_labels <- test_df[,1]

    # check the time
    t_start <- Sys.time()
    # get the prediction
    numbers_test_pred <- knn(train = train_df[,c(2:325)], test = test_df[,c(2:325)], cl = id_train_label
    timing_now <- (Sys.time() - t_start ) * 1000 #ms

    k_error <- k_error +mean(id_test_labels != numbers_test_pred)
```

```
    timing_now_avg <- timing_now_avg + timing_now

  }
  # get summary of one k
  k_error_avg <- k_error / 10
  k_error_avg_vec <- c(k_error_avg_vec, k_error_avg)

  timing_now_avg <- timing_now_avg / 10
  timing_now_avg_vec <- c(timing_now_avg_vec, timing_now_avg)

  print(paste0("k_now = ", k_now, " k_error_avg = ", k_error_avg, " timing_now_avg=", timing_now_avg ))
}
```

```
## [1] "k_now = 19 k_error_avg = 0.061 timing_now_avg=20.5644845962524"
```

```
#plot(ks, k_error_avg_vec, type="o", ylab="misclassification error")
#plot(ks, timing_now_avg_vec, type="o", ylab="average execution time [ms]")
```

As we can see, we achieved the predicted labels from the supervised learning of the KNN algorithm. Since the prediction was calculated with a single k = 19, we ran the algorithm 10 times in order to be able to get average results. From this, we can see that on average classifying all the unknown data took 22 ms with an average error of 0.061.

But to actually compare the method's performance we should investigate how the performance changes with different parameters - in the case of a KNN algorithm we will be changing K's.

1.4.2 Performance of varying K:

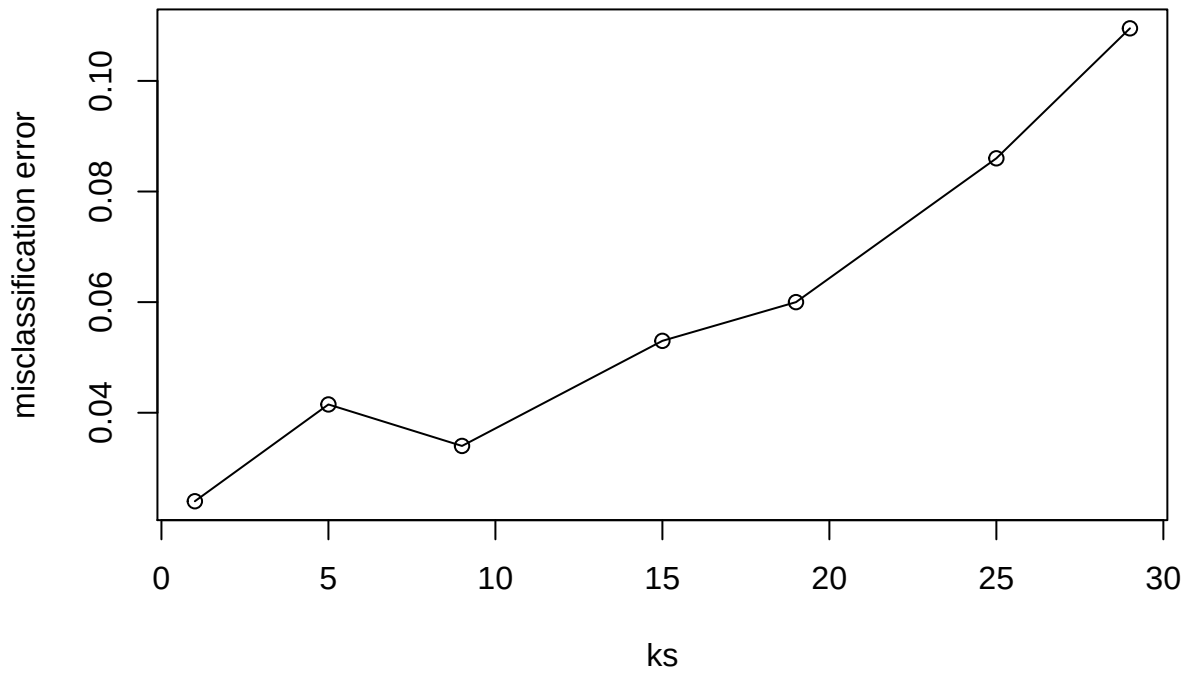
Analyse performance with varying K.

```
 accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}

load("data/id100.Rda")
set.seed(423)

k_error_avg_vec <- c()
timing_now_avg_vec <- c()
ks <- c(1, 5, 9, 15, 19, 25, 29)
# ks <- 1:40

for (k_now in ks ){
  #print(paste0("k_now = ", k_now))
  k_error <- 0
  k_error_avg <- 0

  timing_now_avg <- 0
  for (i in 1:10) {
    # shuffle dataset
    shuffled_df <- id[sample(nrow(id)),]

    # split 10/90 for training and testing data
    test_df <- shuffled_df[1:200,]  # shuffled_df[1:360,]
    train_df <- shuffled_df[201:400,] # shuffled_df[361:400,]

    # get the labels - supervision part
    id_train_labels <- train_df[,1]
```

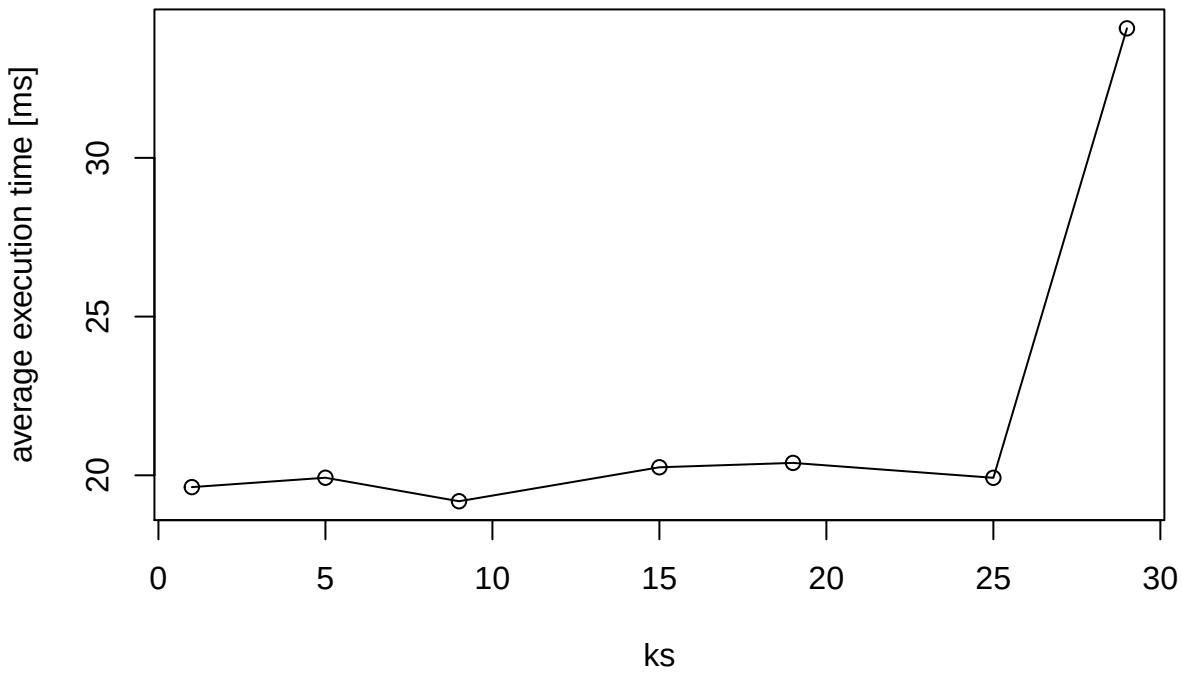
```
    id_test_labels <- test_df[,1]

    # check the time
    t_start <- Sys.time()
    # get the prediction
    numbers_test_pred <- knn(train = train_df[,c(2:325)], test = test_df[,c(2:325)], cl = id_train_label
    timing_now <- (Sys.time() - t_start ) * 1000 #ms

    k_error <- k_error +mean(id_test_labels != numbers_test_pred)
    timing_now_avg <- timing_now_avg + timing_now

  }
  # get summary of one k
  k_error_avg <- k_error / 10
  k_error_avg_vec <- c(k_error_avg_vec, k_error_avg)

  timing_now_avg <- timing_now_avg / 10
  timing_now_avg_vec <- c(timing_now_avg_vec, timing_now_avg)

  print(paste0("k_now = ", k_now, " k_error_avg = ", k_error_avg, " timing_now_avg=", timing_now_avg ))
}
```

```
## [1] "k_now = 1 k_error_avg = 0.024 timing_now_avg=19.6284532546997"
## [1] "k_now = 5 k_error_avg = 0.0415 timing_now_avg=19.9242115020752"
## [1] "k_now = 9 k_error_avg = 0.034 timing_now_avg=19.1832542419434"
## [1] "k_now = 15 k_error_avg = 0.053 timing_now_avg=20.2534675598145"
## [1] "k_now = 19 k_error_avg = 0.06 timing_now_avg=20.3906536102295"
## [1] "k_now = 25 k_error_avg = 0.086 timing_now_avg=19.9213743209839"
## [1] "k_now = 29 k_error_avg = 0.1095 timing_now_avg=34.0799808502197"
```

```
plot(ks, k_error_avg_vec, type="o", ylab="misclassification error")
```




```
plot(ks, timing_now_avg_vec, type="o", ylab="average execution time [ms]")
```



In order to ensure that the model will accurately predict and generalize the future data, we must choose the appropriate K. To do so, we tested several (odd) values of k on the dataset, through trial and errors.

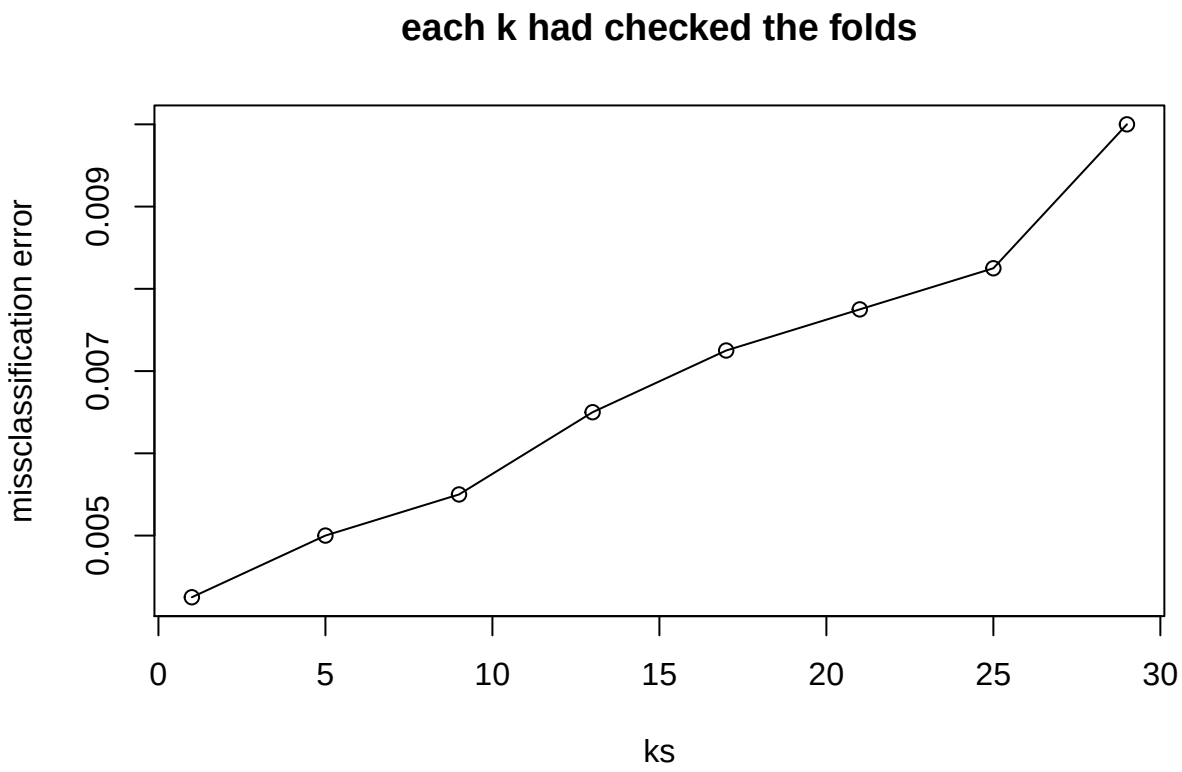
Following the same steps as before, we computed the misclassification error and average execution time for each K and plotted them. As we can see, the larger the value of K, the more the misclassification error rate grows exponentially, with the lowest misclassification error at k = 9, being 0.0075 . And as for the average computational time, values of K between 5 and 25 seem to take more time than values larger than 30.

1.4.3 Cross validation:

Perform a cross validation with a 90% / 10% split with 10 runs. Report mean and standard deviation of the performance.

```
library(class)
library(caret)
load("data/id100.Rda")

# shuffle dataset
set.seed(423)
shuffled_df <- id[sample(nrow(id)),]

# for every fold check all the ks
ks <- c(1,5,9,13,17,21,25,29)
mse_k_all_folds_avg <- rep(0, length(ks)) # vector for storing error of each k

# do the folds
folds <- createFolds(shuffled_df$X1, k = 10)
```

```r
fold_no <- 0 # just for printing the progress

#current fold - to ignore in training data but to use as a validation dataset: fold -> test!
for (fold in folds){
  fold_no <- fold_no + 1
  #print(paste0("fold_no =", fold_no))

  fold <- sample(fold) # shuffle id numbers of fold

  # use not-fold data as training and fold data as test
  not_f_df <- shuffled_df[ -fold, ]
  f_df <- shuffled_df[ fold, ]

  # get labels
  f_train_labels <- not_f_df[,1]
  f_test_labels <- f_df[,1]

  # predict with knn - test all k's for each fold

  k_no <- 0 # just to keep track
  k_error <- 0
  for (k in ks){
    k_no <- k_no + 1
    f_test_pred <- knn(train = not_f_df[,c(2:325)], test = f_df[,c(2:325)], cl = f_train_labels, k=k)

    k_error <- mean(f_test_labels != f_test_pred) # error of current k in this fold
    #print(paste0("      k =", k, " k_error =", k_error))
    #print( mse_k_all_folds_avg) # Error vector that we want to plot

    # we just want to plot the results so we came up with idea of saving the errors in vector
    # for each k in the vector all the folds are summed up together, and then we get the average
    mse_k_all_folds_avg[k_no] <- mse_k_all_folds_avg[k_no] + k_error
  }
}
mse_k_all_folds_avg <- mse_k_all_folds_avg / length(folds)

#print(paste0("mse_k_all_folds_avg = ", mse_k_all_folds_avg))
plot(ks, mse_k_all_folds_avg, type="o", ylab="missclassification error", main="each k had checked the f
```



Cross-validation refers to a set of methods for measuring the performance of a given predictive model on new test data sets. We have a training data set, that is a set of observations with different features, and each observation is associated with a label. Our goal here is to predict the label of any new samples by learning patterns from the training data. With this method we solve the training and test data problem, while still using all the data for testing the accuracy.

To do so, we split our data into a number of folds. If we have n folds, then the first step will be to train the model using n-1 of the folds and test the accuracy on the one left. We repeat this process n times until each fold has been used as in the test set.

In this exercise, we performed a cross-validation with a 90/10 with 10 runs. This was executed with two loops. An outer loop which splits the data into 10 folds then trains the model on 9 folds at a time, testing the accuracy on the fold left. And an inner loop, in which we test each fold for varying values of Ks as we did in the exercise above. Finally, we computed the misclassification error for each k for all the folds from which we obtained the average.

While the misclassification error rate continues to show a tendency to grow exponentially with larger values of K's, we notice a slight difference from the previous exercise: notably, that k = 5 has now the lowest misclassification error of all the K's we've tried — 0.005.

1.4.4 Person independent KNN:

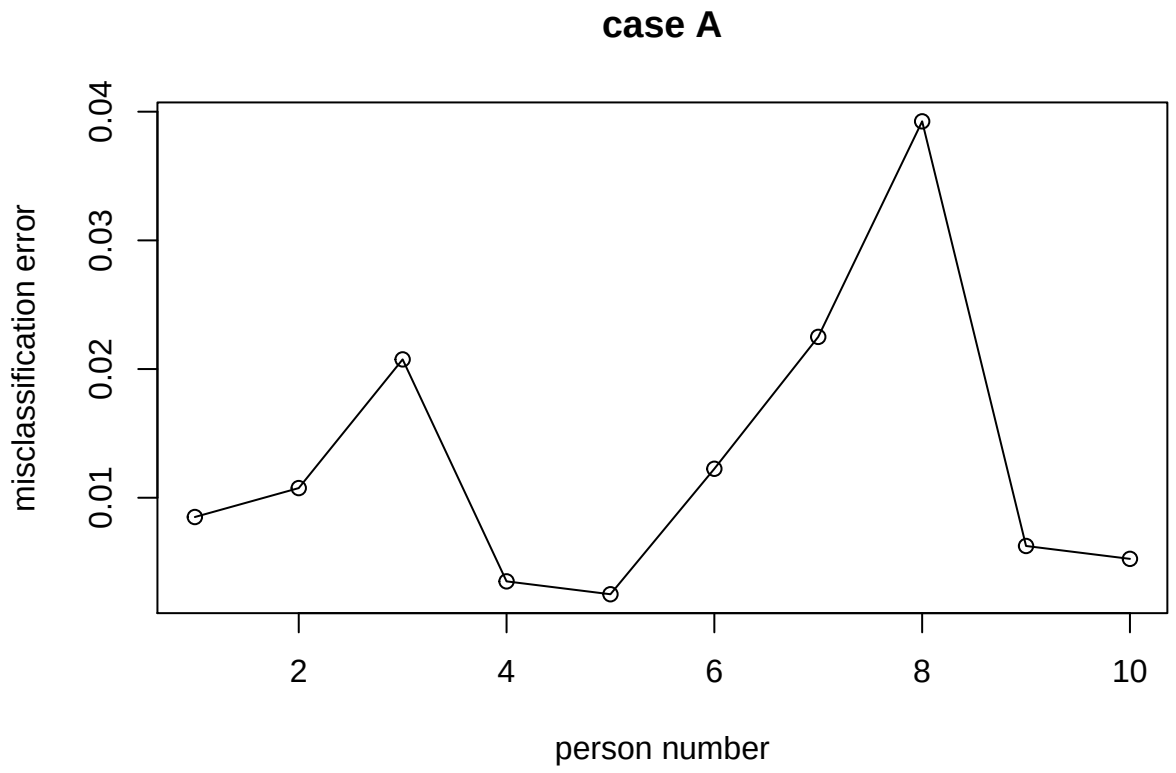
Now try to apply k-nearest neighbor classification to the complete data set from all students attending the course. Distinguish two cases: Having data from all individuals in the training set and splitting the data according to individuals. Generate and explain the results.

```
# CASE A
load("data/idList-co-100.Rdata")
```

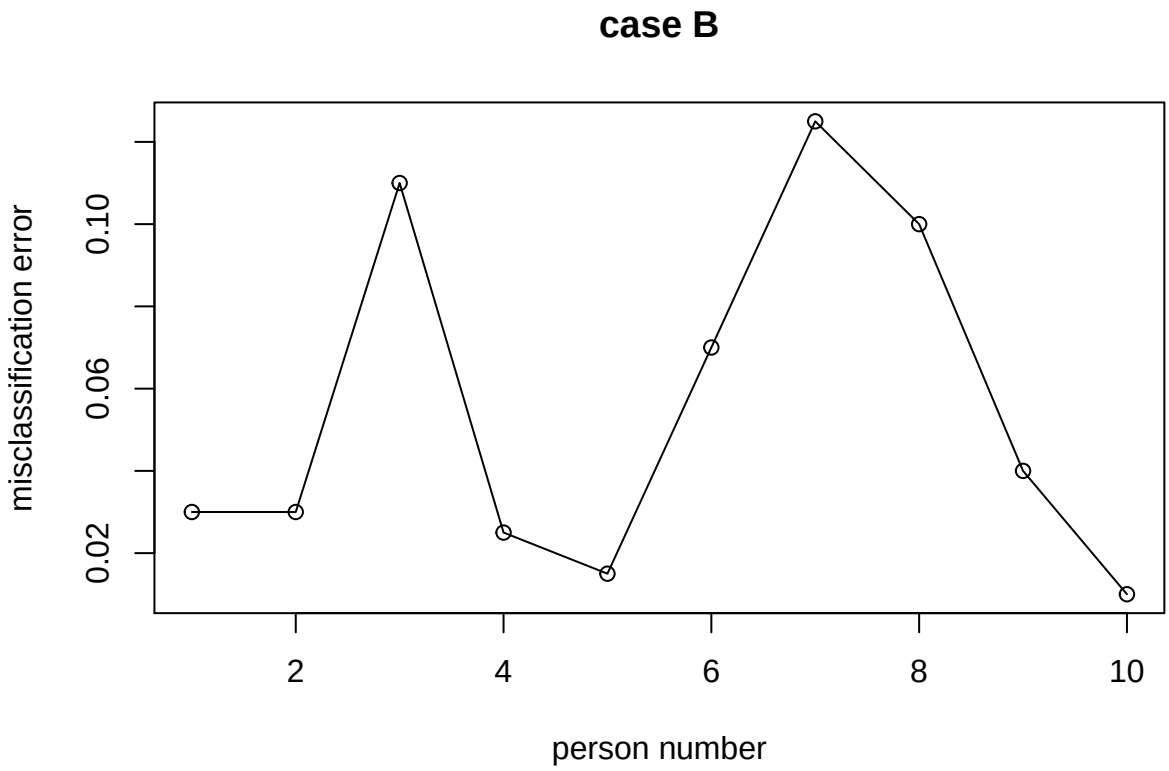
```r
id <- do.call(rbind, idList[1:10])
id <- as.data.frame(id)
id$V1 <- factor(id$V1)

# id - huge dataframe with everyone as TRAINING
train_df <- id
id_train_labels <- train_df[,1]

person_error_v <- c()
# individually test sets for each person
for (person_df in idList){
  print("new_person")
  test_df <- person_df
  id_test_labels <- person_df[,1]

  # get the prediction
  numbers_test_pred <- knn(train = train_df[,c(2:325)], test = test_df[,c(2:325)], cl = id_train_labels

  person_error <- mean(id_test_labels != numbers_test_pred)
  person_error_v <- c(person_error_v, person_error)
}
```

```
## [1] "new_person"
## [1] "new_person"
## [1] "new_person"
## [1] "new_person"
## [1] "new_person"
## [1] "new_person"
## [1] "new_person"
## [1] "new_person"
## [1] "new_person"
## [1] "new_person"
```

```r
#print(paste0("dim(person_error_v) = ", dim(person_error_v)))
plot(1:length(idList), person_error_v, type="o", ylab="misclassification error", xlab="person number", m
```




```
# CASE B
person_error_v <- c()
for (person_df in idList){
  # use individually data for training and testing
  shuffled_df <- person_df[sample(nrow(person_df)),]

  test_df <- shuffled_df[1:200,]
  train_df <- shuffled_df[201: 400,]

  id_test_labels <- test_df[,1]
  id_train_labels <- train_df[,1]

  # get the prediction
  numbers_test_pred <- knn(train = train_df[,c(2:325)], test = test_df[,c(2:325)], cl = id_train_labels

  person_error <- mean(id_test_labels != numbers_test_pred)
  person_error_v <- c(person_error_v, person_error)

}

plot(1:length(idList), person_error_v, type="o", ylab="misclassification error", xlab="person number", r
```



We could assume that predicting the data would have high accuracy when both the training and test data come from a single person with the same handwriting style. To have a better insight of the KNN classification performance, we conducted person-independent tests with the data obtained from 10 people. That means the handwriting style in the training data and the test data will vary, which means we expect slightly less accurate predictions. (Case A)

To understand how using data of different people for prediction is influencing the performance of classification we compare results of the individual and person-independent digit recognition. (Case B)

Comparing both cases using k=5, we can see that in both cases few people had illegible handwriting - for those people in general KNN had worse accuracy.

What is more interesting is that overall, case B (person-dependent) turned out to have lower accuracy than case A (person-independent). That means that using data from multiple people (person-independent) had a positive outcome. We could understand it in the following way: since every person in the dataset has a similar way to write digits in general, using a diverse training dataset leads to a more general classifying method —which translates into higher error rates in case B rather than case A.

1.3.5 Performance of sample size:

Lastly report computational time of the prediction step for varying 'k' and using a small and large datasets. You don't have to test every 'k' simply give anoverview. Discuss how the accuracy changes with different sizes of the dataset, is 'k' dependent on the dataset size?

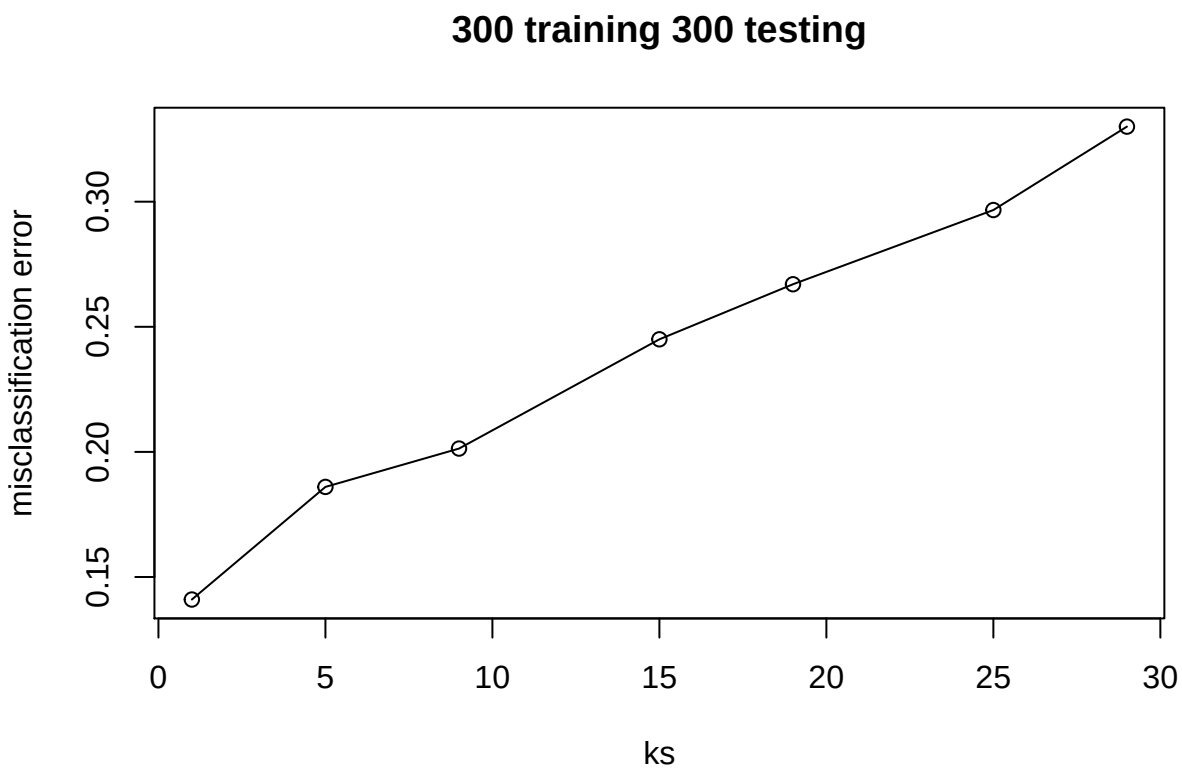
```
load("data/idList-co-100.Rdata")
id <- do.call(rbind, idList[1:10])
id <- as.data.frame(id)
id$V1 <- factor(id$V1)
set.seed(423)
```

```r
k_error_avg_vec <- c()
timing_now_avg_vec <- c()
ks <- c(1, 5, 9, 15, 19, 25, 29)
# ks <- 1:40

# Case A - smaller dataset
for (k_now in ks ){
  #print(paste0("k_now = ", k_now))
  k_error <- 0
  k_error_avg <- 0

  timing_now_avg <- 0
  for (i in 1:10) {
    # shuffle dataset
    shuffled_df <- id[sample(nrow(id)),]

    # split for training and testing data
    #test_df <- shuffled_df[1:20000,]  # shuffled_df[1:360,]
    #train_df <- shuffled_df[20001:40000,] # shuffled_df[361:400,]

    # try to decrease the sample size so it can be run
    test_df <- shuffled_df[1:300,]  # shuffled_df[1:360,]
    train_df <- shuffled_df[301:600,] # shuffled_df[361:400,]

    # get the labels - supervision part
    id_train_labels <- train_df[,1]
    id_test_labels <- test_df[,1]

    # check the time
    t_start <- Sys.time()
    # get the prediction
    numbers_test_pred <- knn(train = train_df[,c(2:325)], test = test_df[,c(2:325)], cl = id_train_label
    timing_now <- (Sys.time() - t_start ) * 1000 #ms

    k_error <- k_error +mean(id_test_labels != numbers_test_pred)
    timing_now_avg <- timing_now_avg + timing_now

  }
  # get summary of one k
  k_error_avg <- k_error / 10
  k_error_avg_vec <- c(k_error_avg_vec, k_error_avg)

  timing_now_avg <- timing_now_avg / 10
  timing_now_avg_vec <- c(timing_now_avg_vec, timing_now_avg)

  print(paste0("k_now = ", k_now, " k_error_avg = ", k_error_avg, " timing_now_avg=", timing_now_avg ))
}
```

```
## [1] "k_now = 1 k_error_avg = 0.141 timing_now_avg=38.5498762130737"
## [1] "k_now = 5 k_error_avg = 0.186 timing_now_avg=38.324499130249"
## [1] "k_now = 9 k_error_avg = 0.201333333333333 timing_now_avg=38.4593963623047"
## [1] "k_now = 15 k_error_avg = 0.245 timing_now_avg=39.2290830612183"
## [1] "k_now = 19 k_error_avg = 0.267 timing_now_avg=39.1318321228027"
```

```
## [1] "k_now = 25 k_error_avg = 0.296666666666667 timing_now_avg=39.8173570632935"
## [1] "k_now = 29 k_error_avg = 0.33 timing_now_avg=40.3743743896484"
```

```
plot(ks, k_error_avg_vec, type="o", ylab="misclassification error", main="300 training 300 testing")
```



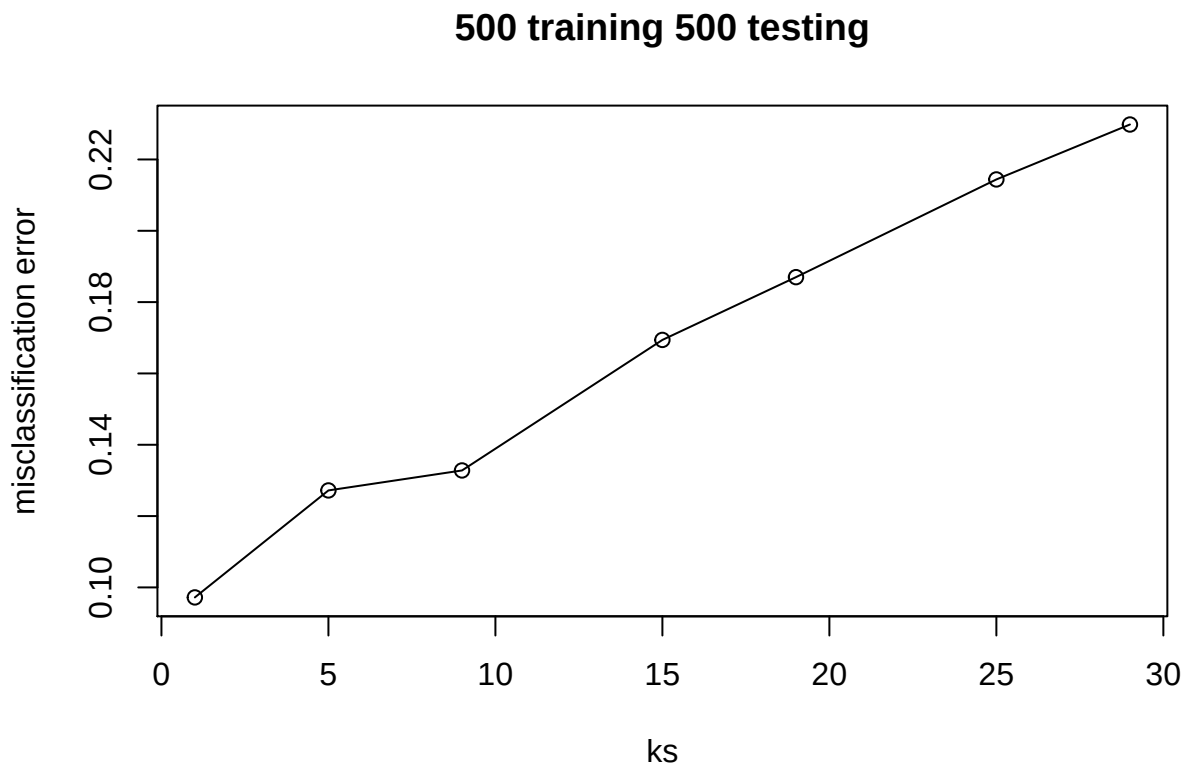

```
plot(ks, timing_now_avg_vec, type="o", ylab="average execution time [ms]", main="300 training 300 testi
```


```
# Case B - bigger dataset
k_error_avg_vec_big <- c()
timing_now_avg_vec_big <- c()
for (k_now in ks ){
  #print(paste0("k_now = ", k_now))
  k_error <- 0
  k_error_avg <- 0

  timing_now_avg <- 0
  for (i in 1:10) {
    # shuffle dataset
    shuffled_df <- id[sample(nrow(id)),]

    # split for training and testing data
    test_df <- shuffled_df[1:500,]  # shuffled_df[1:360,]
    train_df <- shuffled_df[501:1000,] # shuffled_df[361:400,]

    # get the labels - supervision part
    id_train_labels <- train_df[,1]
    id_test_labels <- test_df[,1]

    # check the time
    t_start <- Sys.time()
    # get the prediction
    numbers_test_pred <- knn(train = train_df[,c(2:325)], test = test_df[,c(2:325)], cl = id_train_labe
    timing_now <- (Sys.time() - t_start ) * 1000 #ms
```

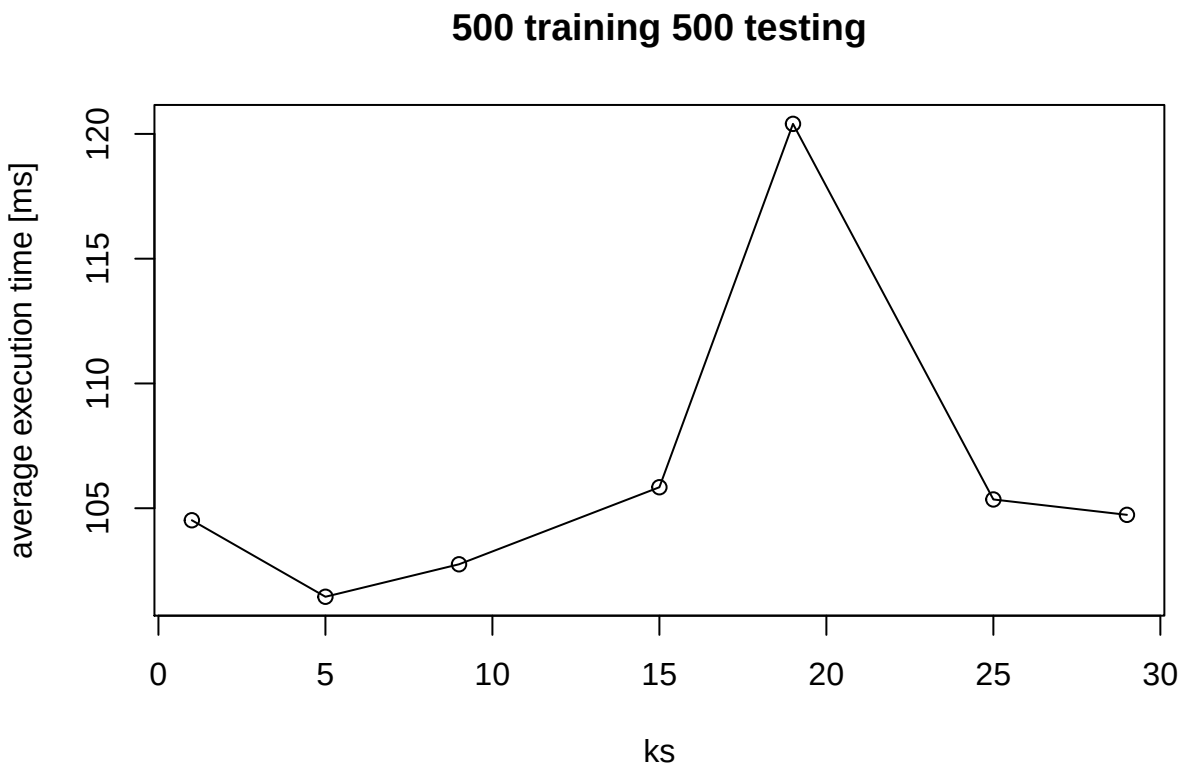
```
    k_error <- k_error +mean(id_test_labels != numbers_test_pred)
    timing_now_avg <- timing_now_avg + timing_now

  }
  # get summary of one k
  k_error_avg <- k_error / 10
  k_error_avg_vec_big <- c(k_error_avg_vec_big, k_error_avg)

  timing_now_avg <- timing_now_avg / 10
  timing_now_avg_vec_big <- c(timing_now_avg_vec_big, timing_now_avg)

  print(paste0("k_now = ", k_now, " k_error_avg = ", k_error_avg, " timing_now_avg=", timing_now_avg ))
}
```

```
## [1] "k_now = 1 k_error_avg = 0.0972 timing_now_avg=104.518532752991"
## [1] "k_now = 5 k_error_avg = 0.1272 timing_now_avg=101.453113555908"
## [1] "k_now = 9 k_error_avg = 0.1328 timing_now_avg=102.754235267639"
## [1] "k_now = 15 k_error_avg = 0.1694 timing_now_avg=105.841684341431"
## [1] "k_now = 19 k_error_avg = 0.187 timing_now_avg=120.4021692276"
## [1] "k_now = 25 k_error_avg = 0.2144 timing_now_avg=105.352139472961"
## [1] "k_now = 29 k_error_avg = 0.2298 timing_now_avg=104.734992980957"
```

```
plot(ks, k_error_avg_vec_big, type="o", ylab="misclassification error", main="500 training 500 testing")
```



```
plot(ks, timing_now_avg_vec_big, type="o", ylab="average execution time [ms]", main="500 training 500 t
```



During the execution of the KNN algorithm for the person-independent dataset (case A), we could already see that the computational time increases with the increase of the sample size. Here we have compared the average computational time and error for two datasets — a small one with 600 samples and a bigger one with 1000 samples. For each dataset, we divided the training and testing data 50/50 and obtained average results from 10 runs.

As we can see with a larger dataset the KNN algorithm takes significantly more time to classify. The least time it took to classify the small dataset was 38.32526 ms for k=5 and the fastest it went for the big dataset was for k=5, taking 99.8517ms.

Despite the computational time, the average accuracy also changes with the data size. This time with larger sample of data we can observe a decrease of the average error, meaning that with larger datasets, the accuracy increases.