KNN Project

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```
# Packages needed to run this R notebook:
#    - "class" for KNN algorithm
#    - "e1071" for SVM algorithm
#    - "rpart" for decision trees algorithm
#    - "rpart.plot" for plotting decision tree
#    - "randomForest" for random forest algorithm
```

```
 # Run this to install and load the necessary packages

ipak <- function(pkg){
    new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
    if (length(new.pkg))
        install.packages(new.pkg, dependencies = TRUE)
    sapply(pkg, require, character.only = TRUE)
}

packages <- c("class", "e1071", "rpart", "rpart.plot", "randomForest", "stringr")
ipak(packages)
```

```
## Loading required package: class
```

```
## Loading required package: e1071
```

```
## Loading required package: rpart
```

```
## Loading required package: rpart.plot
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
## Loading required package: stringr
```

```
##        class        e1071        rpart   rpart.plot randomForest
##         TRUE         TRUE         TRUE         TRUE         TRUE
##      stringr
##         TRUE
```

```
## If you would like to calculate the time it takes to execute a chunk of code, pleas
e use the code below
sleep_for_a_minute <- function() { Sys.sleep(0) }
start_time <- Sys.time()
sleep_for_a_minute()
# Insert code you would like to measure
end_time <- Sys.time()
time <- end_time - start_time
time  # Duration of the execution in seconds
```

```
## Time difference of 0.004997969 secs
```

```
# The first three chunks are to read the observations
# TRAINING SET
# First, we get the list of all files in the directory

list_files <- list.files(path = "digits/trainingDigits",all.files = TRUE) # We get th
e list of all files
paste0("There are ",length(list_files)," files in the directory")
```

```
## [1] "There are 1936 files in the directory"
```

```
list_files <- list_files[3:1936] # We remove the first two (they indicate the folders
)

list_files_path <- list()  # We create a list with the relative filepath of files
for (item in list_files){
  name <- paste("digits/trainingDigits/",item,sep = "") #This is the relative path. D
epending on your operating system, you might need to enter the full path
  list_files_path[item] <- name
}
length(list_files_path)==length(list_files)
```

```
## [1] TRUE
```

```r
# TRAINING CLASS (Y)
# Here we create a list with the class of each object (the first number of each file
represents the class)

train_Y <- c()

for (f in list_files){
  first_char <- substr(f, 1, 1)
  train_Y[f] <- first_char
}
train_Y <- Reduce('rbind', train_Y)
train_Y <- c(train_Y)
train_Y <- as.data.frame(train_Y)

train_Y = train_Y[,1]
```

```r
# OBSERVATIONS (X)
# Now, we will read through the list of training files and put them on a list. Number
of observations: 1934

train_num_list <- list()

for (i in list_files_path){
  read_txt <- scan(i,what="character", sep=NULL) #We read the txt as characters, not
as numbers
  oneline <- as.numeric(stringr::str_split(paste(read_txt, collapse = ""), "")[[1]])
# We convert 32x32 to 1x1024
  train_num_list[[i]] <- oneline
}
print(length(train_num_list) == length(train_Y)) # We make sure that the number of ob
servations and the length of the classes match
```

```
## [1] TRUE
```

```r
# TEST SET
# First, we get the list of all files in the directory
test_list_files <- list.files(path = "digits/testDigits",all.files = TRUE)
length(test_list_files)
```

```
## [1] 948
```

```
test_list_files <- test_list_files[3:length(test_list_files)]

test_list_files_path <- list()  # We create a list with the relative filepath of file
s
for (item in test_list_files){
  name <- paste("digits/testDigits/",item,sep = "") #This is the relative path. Depen
ding on your operating system, you might need to enter the full path
  test_list_files_path[item] <- name
}

#   name <- paste("digits/testDigits/",i,sep = "")

#  print(test_list_files_path[946])
```

```
# TEST CLASS (test_y)
# Here we create a list with the class of each object (the number each file represent
s)

test_y <- list()

for (f in test_list_files){
  first_char <- substr(f, 1, 1)
  test_y[f] <- first_char
}

test_y <- Reduce('rbind', test_y)
test_y <- c(test_y)
test_y <- as.data.frame(test_y)

test_y = as.data.frame(test_y[,1])

length(test_y) == length(test_list_files_path)
```

```
## [1] FALSE
```

```
# OBSERVATIONS (test_x)
# Now, we will read through the list of training files and put them on a list. Number
of observations: 1934

test_num_list <- list()

for (i in test_list_files_path){
  read_test_txt <- scan(i,what="character", sep=NULL) #We read the txt as characters,
not as numbers
  oneline <- as.numeric(stringr::str_split(paste(read_test_txt, collapse = ""), "")[[
1]]) # We convert 32x32 to 1x1024
  test_num_list[[i]] <- oneline
}

print(length(test_num_list) == length(test_y)) # We make sure that the number of obse
rvations and the length of the classes match
```

```
## [1] FALSE
```

```
# We get the X matrix for our training data as a data frame

X_train <- as.data.frame(do.call(rbind, unname(train_num_list)))
```

```
# We get the X of our test data that we would like to predict, as a data frame

X_test <- as.data.frame(do.call(rbind, unname(test_num_list)))
```

```
# KNN algorithm and we convert to a list with predictions

library(class)
knn1 <- knn(train=X_train, test=X_test, cl=train_Y, k=3)
knn_df <- as.data.frame(knn1)
```

```
# We have to compare knn outcome with test_y, so we know how precise the knn is

results <- cbind.data.frame(knn_df[,1], test_y[,1])

count_right <- 0

# We run through all the predicted values and
# compare them with the real values. If correct, we add 1 to count_right

for (row in 1:nrow(results)){
  knn_pred <- results[row,1]
  test_obs <- results[row,2]
  if (knn_pred == test_obs){
    count_right <- count_right + 1
  }
    }
count_right
```

```
## [1] 932
```

```
knn_accuracy <- count_right/nrow(results)
knn_accuracy
```

```
## [1] 0.9852008
```

```
# We create a confusion matrix with the predicted values and real values.
confMat <- table(data=results[,1], reference=results[,2])
confMat
```

```
##     reference
## data   0   1   2   3   4   5   6   7   8   9
##    0  87   0   0   0   0   0   0   0   0   0
##    1   0  96   0   0   0   0   0   0   3   1
##    2   0   0  92   0   0   0   0   0   0   0
##    3   0   0   0  84   0   1   0   0   1   1
##    4   0   0   0   0 114   0   0   0   0   0
##    5   0   0   0   0   0 105   0   0   0   1
##    6   0   0   0   0   0   2  87   0   1   0
##    7   0   1   0   0   0   0   0  96   0   1
##    8   0   0   0   0   0   0   0   0  86   0
##    9   0   0   0   1   0   0   0   0   0  85
```

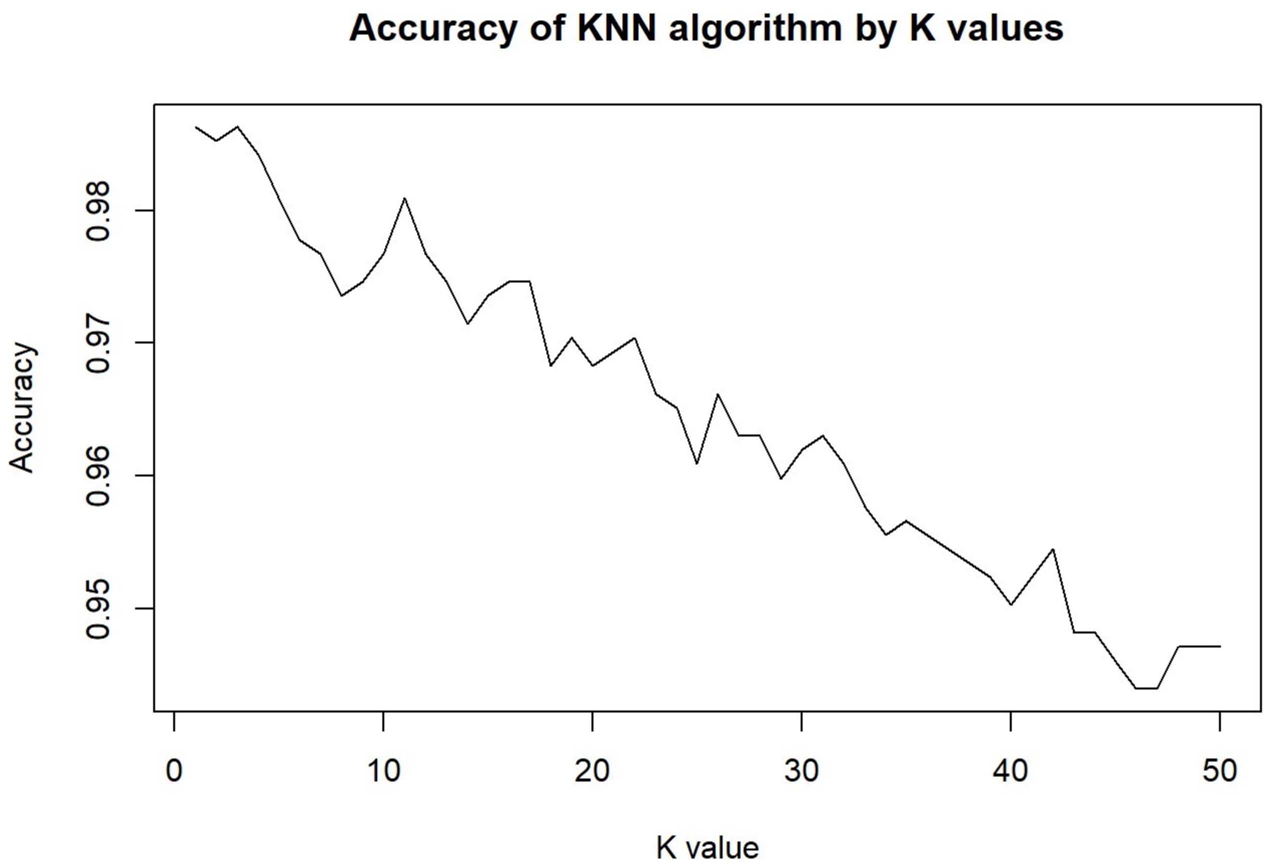
```
# We compare the accuracy of the different values of k. It will take several minutes
to compute

library(class)

accuracy_list <- c()

for (k in 1:50){
  knn2 <- knn(train=X_train, test=X_test, cl=train_Y, k, use.all=FALSE)
  knn_df2 <- as.data.frame(knn2)
  results2 <- cbind.data.frame(knn_df2[,1],test_y[,1])
  count_right2 <- 0
  for (row in 1:nrow(results2)){
    knn_pred1 <- results2[row,1]
    test_obs1 <- results2[row,2]
    if (knn_pred1 == test_obs1){
      count_right2 <- count_right2 +1
    }
  }
  accuracy <- count_right2/nrow(results2)
  accuracy_list[k] <- accuracy
}

print(accuracy_list)
```

```
##  [1] 0.9862579 0.9852008 0.9862579 0.9841438 0.9809725 0.9778013 0.9767442
##  [8] 0.9735729 0.9746300 0.9767442 0.9809725 0.9767442 0.9746300 0.9714588
## [15] 0.9735729 0.9746300 0.9746300 0.9682875 0.9704017 0.9682875 0.9693446
## [22] 0.9704017 0.9661734 0.9651163 0.9608879 0.9661734 0.9630021 0.9630021
## [29] 0.9598309 0.9619450 0.9630021 0.9608879 0.9577167 0.9556025 0.9566596
## [36] 0.9556025 0.9545455 0.9534884 0.9524313 0.9503171 0.9524313 0.9545455
## [43] 0.9482030 0.9482030 0.9460888 0.9439746 0.9439746 0.9471459 0.9471459
## [50] 0.9471459
```

```
print(summary(accuracy_list))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9440  0.9545  0.9641  0.9644  0.9746  0.9863
```

```
# We see above that the optimal K value is 1 which gives us an accuracy of nearly 99%
# We plot the accuracy of the different k values

plot(accuracy_list[1:50], ylab="Accuracy", xlab="K value", type="l", main="Accuracy o
f KNN algorithm by K values")
```




```
# Now we will apply the Support Vector Machine (SVR) for classification and compare t
he results
```

```
library(e1071)
svm_model <- svm(train_Y ~ ., data=X_train, scale=FALSE)
summary(svm_model)
```

```
##
## Call:
## svm(formula = train_Y ~ ., data = X_train, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.0009765625
##
## Number of Support Vectors:  1139
##
##  ( 78 130 101 124 109 122 87 93 135 160 )
##
##
## Number of Classes:  10
##
## Levels:
##  0 1 2 3 4 5 6 7 8 9
```

```
# We get the predicted values for the test data
pred_svm <- predict(svm_model, X_test)
summary(pred_svm)
```

```
##   0   1   2   3   4   5   6   7   8   9
##  88  99  87  80 111 108  87  97  96  93
```

```
# We calculate the accuracy of the algorithm
rightguess <- 0

pred <- as.data.frame(pred_svm)

results_svm <- cbind.data.frame(pred,test_y)

results_svm <- as.data.frame(results_svm)

for (row in (1:nrow(results_svm))){
  svm_pred <- results_svm[row,1]
  svm_test <- results_svm[row,2]
  if (svm_pred==svm_test){
    rightguess <- rightguess+1
  }
}

accuracy_svm <- rightguess/946
accuracy_svm
```

```
## [1] 0.9735729
```

```
confMat <- table(data=results_svm[,1], reference=results_svm[,2])
confMat
```

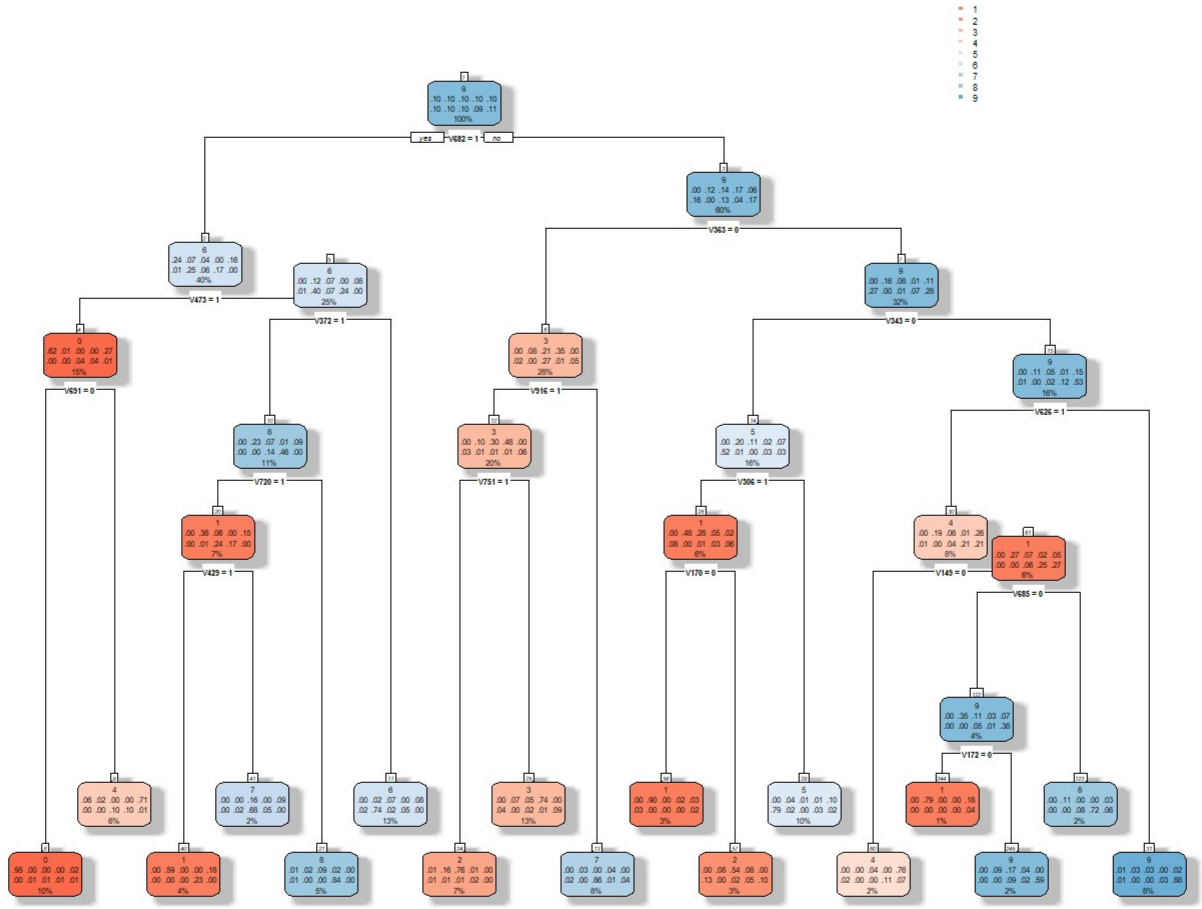
```
##     reference
## data   0   1   2   3   4   5   6   7   8   9
##    0  87   0   0   0   1   0   0   0   0   0
##    1   0  96   0   0   1   0   0   0   1   1
##    2   0   0  87   0   0   0   0   0   0   0
##    3   0   0   0  79   0   0   0   0   0   1
##    4   0   0   0   0 109   0   1   1   0   0
##    5   0   0   0   1   0 107   0   0   0   0
##    6   0   0   0   0   0   0  86   0   1   0
##    7   0   1   0   0   0   0   0  95   0   1
##    8   0   0   4   1   2   0   0   0  89   0
##    9   0   0   1   4   1   1   0   0   0  86
```

```
summary(confMat)
```

```
## Number of cases in table: 946
## Number of factors: 2
## Test for independence of all factors:
##  Chisq = 8027, df = 81, p-value = 0
```

```
# We try now the Decission Tree algorithm
```

```
library(rpart)
library(rpart.plot)
tree_model <- rpart(train_Y ~ ., method="class", data=X_train)
plot <- rpart.plot(tree_model, box.palette="RdBu", shadow.col="gray", nn=TRUE)
```



```
# We calculate the accuracy of the Decission Tree predictions

prediction_dt <- predict(object=tree_model,newdata=X_test,type="class")

results_dt <- cbind.data.frame(prediction_dt,test_y)
rightcount_dt <- 0
for (item in 1:nrow(results_dt)){
  dt_pred <- results_dt[item,1]
  dt_true <- results_dt[item,2]
  if (dt_pred==dt_true){
    rightcount_dt = rightcount_dt+1
  }
}

accuracy_dt <- rightcount_dt/nrow(results_dt)
accuracy_dt
```

```
## [1] 0.7547569
```

```
confMat <- table(data=results_dt[,1], reference=results_dt[,2])
confMat
```

```
##     reference
## data  0  1  2  3  4  5  6  7  8  9
##    0 82  0  0  0  1  0  0  2  0  0
##    1  0 47  0  0 14  2  0  0 19  2
##    2  0 25 60  4  0  4  0  0  6  2
##    3  0 10  4 74  1  2  0  1  1  6
##    4  1  1  0  0 70  0  1  5  2  1
##    5  0  2  3  1  9 89  2  0  0  1
##    6  2  2  5  0 14  1 84  0  8  0
##    7  0  3  9  2  2  7  0 86  1  4
##    8  2  6  6  0  0  1  0  1 49  0
##    9  0  1  5  4  3  2  0  1  5 73
```

```
# We train the Random Forest algorithm
library(randomForest)
number_of_trees <- 200  # I tried with 200, change this value to try different tree n
umbers
rf_model <- randomForest(data=X_train, y=train_Y, x=X_train, ntree=number_of_trees)
prediction_rf <- predict(object=rf_model, newdata=X_test, type="class")
```

```
# We calculate the accuracy of Random Forest
results_rf <- cbind.data.frame(prediction_rf,test_y)

rightcount_rf <- 0
for (item in 1:nrow(results_rf)){
  rf_pred <- results_rf[item,1]
  rf_true <- results_rf[item,2]
  if (rf_pred==rf_true){
    rightcount_rf = rightcount_rf+1
  }
}

accuracy_rf <- rightcount_rf/nrow(results_rf)
accuracy_rf
```

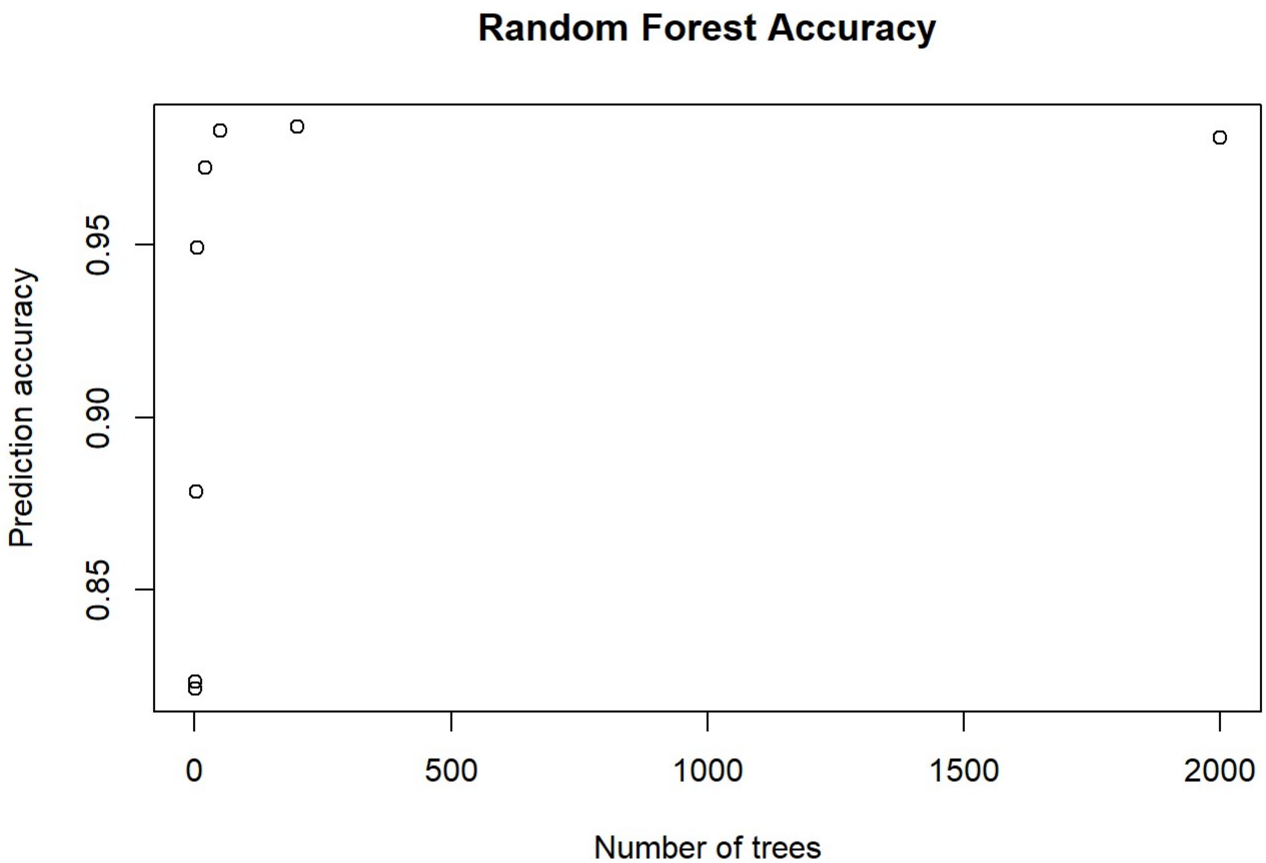
```
## [1] 0.9809725
```

```
# Accuracy of random forest for different number of trees
accuracy_rf_list <- c()

x <- c(1,2,3,5,20,50,200,2000)

for (i in 1:length(x)){
  rf_model <- randomForest(data=X_train, y=train_Y, x=X_train, ntree=x[i])
  prediction_rf <- predict(object=rf_model, newdata=X_test, type="class")

  results_rf <- cbind.data.frame(prediction_rf,test_y)
  rightcount_rf <- 0
  for (item in 1:nrow(results_rf)){
    rf_pred <- results_rf[item,1]
    rf_true <- results_rf[item,2]
    if (rf_pred==rf_true){
      rightcount_rf = rightcount_rf+1
    }
  }
  accuracy_rf <- rightcount_rf/nrow(results_rf)
  accuracy_rf_list[i] = accuracy_rf

}
```

```
accuracy_rf_list
```

```
## [1] 0.8213531 0.8234672 0.8784355 0.9492600 0.9725159 0.9830867 0.9841438
## [8] 0.9809725
```

```
plot(y=accuracy_rf_list, x=x, xlab="Number of trees", ylab="Prediction accuracy", mai
n="Random Forest Accuracy")
```




```
# Naive Bayes algorithm

nb_model <- naiveBayes(x=X_train, y=train_Y, laplace = 0.01)
```

```
# Predictions of new data and we calculate accuracy

prediction_nb <- predict(object=nb_model, newdata=X_test,
  type = c("class"))
prediction_rf <- predict(object=rf_model, newdata=X_test, type="class")

results_nb <- cbind.data.frame(prediction_nb,test_y)
rightcount_nb <- 0
for (item in 1:nrow(results_nb)){
  nb_pred <- results_nb[item,1]
  nb_true <- results_nb[item,2]
  if (nb_pred==nb_true){
    rightcount_nb = rightcount_nb+1
  }
}

accuracy_nb <- rightcount_nb/nrow(results_nb)
accuracy_nb
```

```
## [1] 0.8657505
```

```
confMat <- table(data=results_nb[,1], reference=results_nb[,2])
confMat
```

```
##     reference
## data  0  1  2  3  4  5  6  7  8  9
##    0 86  0  0  0  1  0  0  0  0  0
##    1  0 72  0  0  1  0  0  0  0  1
##    2  0  2 81  0  0  0  0  0  0  0
##    3  0  0  1 73  0  4  0  0  0 14
##    4  0  0  0  0 92  0  0  0  0  1
##    5  0  0  0  0  0 93  0  0  0  0
##    6  0  1  0  0  4  1 86  0  1  0
##    7  0  0  0  1  1  0  0 93  0  2
##    8  1 20 10  9 13  7  1  3 90 18
##    9  0  2  0  2  2  3  0  0  0 53
```