Machine Learning:

Karsten Standnes - STNKAR012

October 2017

1 Introduction

In the world today there are many methods that fall under the category of "Machine Learning", some being very similar and some very different. All of them share in common that they use data to produce a model that can be used on unseen data. When successful this is very appealing in today's society where we got lots of data on situations where there is likely to be a underlying pattern, another requirement for learning. There is broad agreement that Machine Learning is a good way to make prediction and classification models, and often the only computational feasible way to do so. This makes it a task to decide which method in machine learning to chose for a given problem. The answer is not always the same and several methods can be good, but for different reasons. In this task I will show several different machine learning algorithms and how they perform on classifying pictures of handwritten numbers into even and odd numbers.
COMMENT on data snooping

Contents

2 Methods

As mentioned in the different methods in Machine Learning will be used to try to build a model that manage to correctly classify if pictures of handwritten numbers are odd or even. This would be simple if the method should only classify previously seen data, the challenge arise when asked to make a model that is highly accurately in classifying new data. The goal is to make a a model that trains without overfitting on the in-sample data in order to well estimate the out-of-sample data. In the methods below this is done by using different techniques like cross-validation, validation-set and regularization. In all of the methods the data is classified into the numbers $0-9$. Another approach would be to classify the number's directly into either a odd or even category. The reason for using all ten numbers is mainly because this makes the extension of the use purpose a lot smoother, f.ex. to also classify if the number is a prime number or not.

2.1 Tree based methods

2.1.1 Classification tree

Classification tree is maybe the method in Machine Learning which is easiest to interpret due to it's intuitive construction and clear visualization. The method uses a greedy approach using recursive binary splitting to structure a tree that can classify input based on it's variables. The goal of the classification tree is to classify a set of data as best as possible while have a low complexity to avoid overfitting. Below we see that one classification tree is not enough to make a great model for the problem, but combining many of them gives us a "Random Forest" which is discussed in subsection 2.1.2. Classification trees also gives a nice visual image of the classification.

2.1.2 Random Forest

Random Forest is as mentioned (subsection 2.1.1) constructed of many classification tree. After such a construction by using a set of training data, new data can be ran through the "forest". The new data is classified with the label the majority of trees labelled it. The trees in other words "vote" for the best classification for the data. As in real life, voting makes little difference if all the votes are the same. To avoid making a forest out of n ($n \in \mathbb{Z}_{>0}$) identical trees two things are changed in the construction of the tree from the classification tree. One is that a the training data for each tree is a sample of size N from the whole set of N data sampled with replacement. The other is that m randomly selected variables are used for each tree, where $m << M$ and M is the whole set of variables in the problem. This gives a rich variety of trees where with enough trees a predictive model can be created.

2.2 Neural Networks

2.2.1 Artificial?

2.2.2 Convolutional Neural Network

For Neural Networks I have decided to use a convolutional neural network because of it great performance in problems that can be represented as an "image". This because it convolutional neural networks(CNN's) works by recognizing paterns in images or represented as a matrix or tensor(multidimensional matrix) in the computer. It does so by using convolutional, pooling and voting layers. Multiple of these can be stacked to create a very a precise recognising images.

Figure 1: Error measures E_{out} and E_{val} are plotted for different sizes of training and validation sets.

2.3 Support vector machines

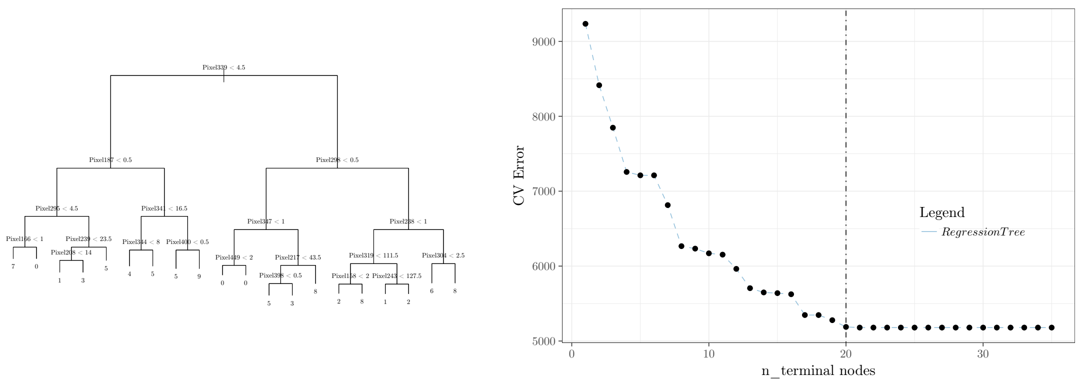
2.4 K-Nearest Neighbours

K-Nearest Neighbours(KNN) is a easy-to interpret and maybe one of the simplest methods in Machine Learning. The method does exactly as the name suggest; find the k nearest neighbours, where k is a number and labels each data point based on those neighbours. The method is attractive because of it's easy interpretability, fast execution time and often precise classification and regression power. There are many variants of the method where the distance measure and way to handle draws differ. KNN's with $k = 1$ have a special property that the hyperplane of the data points will be partitioned into an equal amount of partitions as there are data points. Each partition consist of all point in the plane closer a specific data point than any other. The amount of neighbours k can have a great impact on the accuracy of the classification. In some cases a low k will be preferable with very separated data, in other case it can lead to overfitting due to higher influence from noise and outliers. when k = 1, Voroni partition[look up], negative complexity scaling

3 Results

3.1 Tree based methods

3.1.1 Classification tree



(a) E_{CV} for different amounts of leaf nodes.

(b) Regression tree after pruning

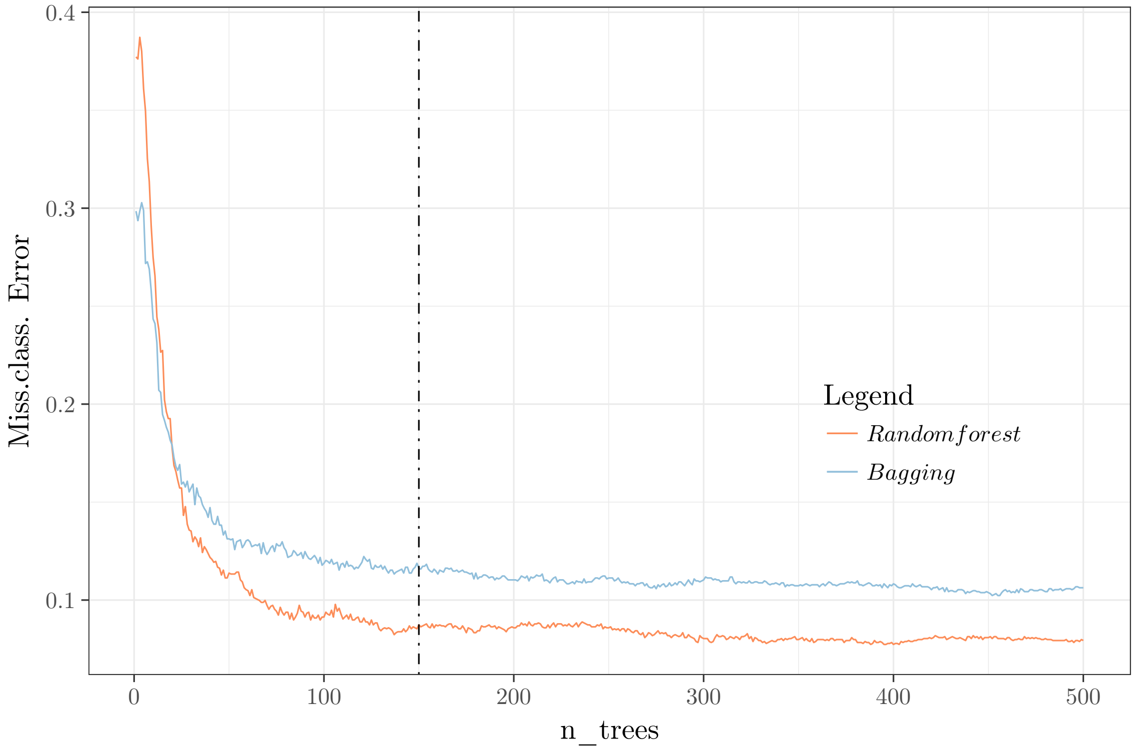
Figure 2: A figure with two subfigures

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	31	0	5	1	1	3	4	1	0	1	0.340	16/47
1	0	44	1	2	0	2	3	0	1	0	0.170	9/53
2	0	3	32	2	1	0	0	0	2	0	0.200	8/40
3	0	1	1	26	1	0	1	1	0	3	0.235	8/34
4	0	0	0	0	26	0	10	0	1	1	0.316	12/38
5	4	2	2	12	9	31	6	10	3	7	0.640	55/86
6	0	1	3	1	2	1	23	0	0	2	0.303	10/33
7	1	0	1	2	0	0	2	34	0	5	0.244	11/45
8	0	2	5	3	6	5	6	0	39	3	0.435	30/69
9	3	0	2	2	7	3	2	1	6	27	0.491	26/53
Total	0	0	0	0	0	0	0	0	0	0	0.371	185/498

3.1.2 Random Forest

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	46	0	0	1	0	0	0	0	0	0	0.021	1/47
1	0	42	0	0	0	1	0	1	0	0	0.045	2/44
2	0	2	48	0	0	0	0	1	0	0	0.059	3/51
3	0	0	0	41	0	1	0	0	3	0	0.089	4/45
4	0	1	0	0	52	1	1	4	0	1	0.133	8/60
5	0	0	0	4	0	42	1	0	1	0	0.125	6/48
6	0	0	0	0	1	0	47	0	2	0	0.060	3/50
7	0	1	0	0	0	0	0	48	0	2	0.059	3/51
8	1	2	0	1	0	0	0	0	42	1	0.106	5/47
9	0	0	0	1	2	1	0	0	1	48	0.094	5/53
Total	0	0	0	0	0	0	0	0	0	0	0.081	40/496

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	47	0	0	2	0	3	0	0	0	0	0.096	5/52
1	0	42	0	0	0	0	0	0	0	0	0.000	0/42
2	0	2	46	0	0	0	0	1	0	0	0.061	3/49
3	0	1	0	41	0	2	0	1	2	0	0.128	6/47
4	0	1	0	0	49	1	1	2	0	2	0.125	7/56
5	0	0	0	3	0	38	1	0	1	0	0.116	5/43
6	0	0	1	0	1	1	47	0	2	1	0.113	6/53
7	0	0	0	0	0	0	0	47	0	3	0.060	3/50
8	0	2	1	1	1	0	0	2	43	1	0.157	8/51
9	0	0	0	1	4	1	0	1	1	45	0.151	8/53
Total	0	0	0	0	0	0	0	0	0	0	0.103	51/496



0.4
0.3
0.2
0.1
Miss.class. Error
0
100
200
300
400
500
n_trees
Legend
Randomforest
Bagging

Figure 3

3.2 Neural Networks

3.2.1 Artificial Neural Network

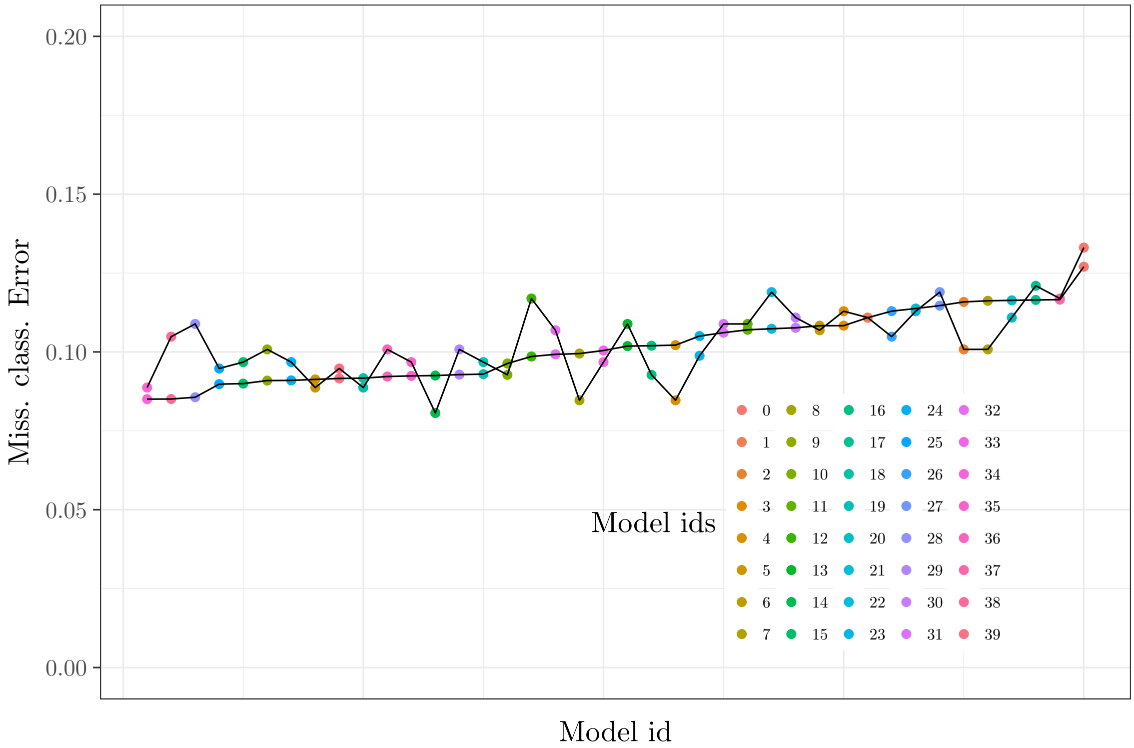
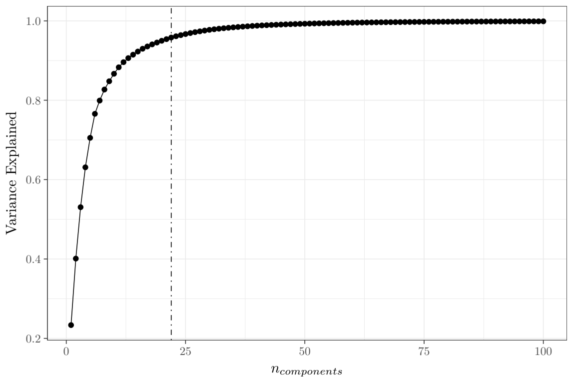


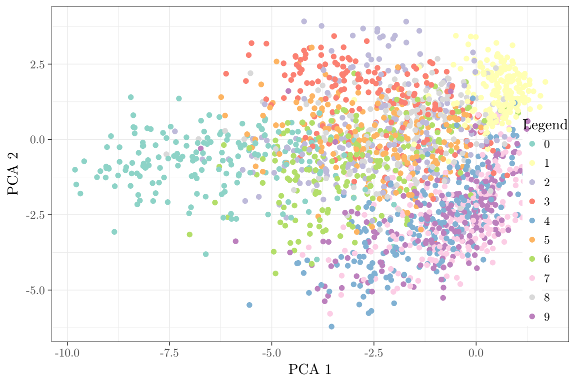
Figure 4

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	42	0	0	0	0	2	0	0	0	0	0.045	2/44
1	0	47	0	0	1	0	0	0	4	0	0.096	5/52
2	2	1	46	2	0	0	1	0	0	0	0.115	6/52
3	0	0	0	42	1	1	0	1	0	0	0.067	3/45
4	0	0	0	0	50	0	0	2	0	2	0.074	4/54
5	1	0	0	2	1	43	0	1	6	1	0.218	12/55
6	1	0	0	1	1	0	48	0	0	0	0.059	3/51
7	1	0	2	0	1	0	0	49	1	3	0.140	8/57
8	0	0	0	1	0	0	0	0	38	0	0.026	1/39
9	0	0	0	0	0	0	0	1	0	46	0.021	1/47
Total	0	0	0	0	0	0	0	0	0	0	0.091	45/496

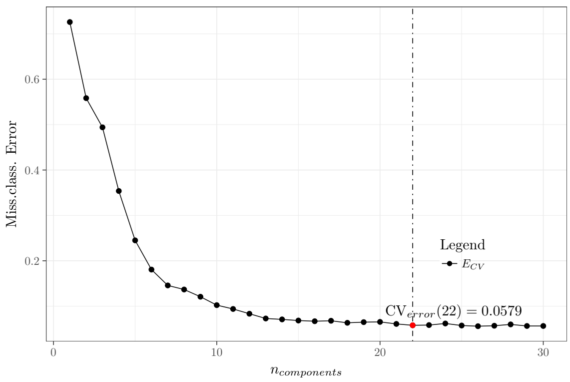
3.3 Support vector machines



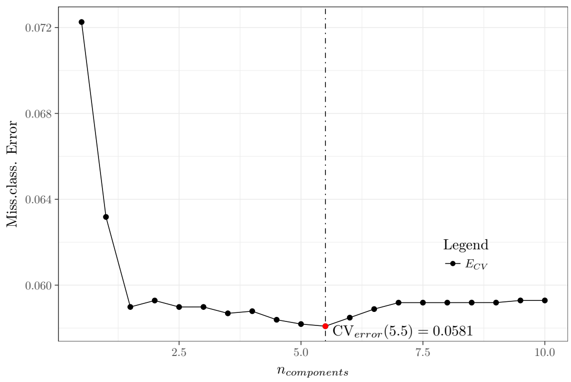
(a)



(b)



(a)



(b)

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	47	0	0	0	0	0	0	0	0	0	0.000	0/47
1	0	48	0	0	0	0	1	0	0	0	0.020	1/49
2	0	0	46	0	0	0	0	0	1	1	0.042	2/48
3	0	0	1	41	0	0	0	1	2	0	0.089	4/45
4	0	0	0	0	53	0	0	2	0	1	0.054	3/56
5	0	0	0	4	0	46	0	0	0	0	0.080	4/50
6	0	0	0	0	1	0	48	0	1	0	0.040	2/50
7	0	0	0	0	0	0	0	49	0	4	0.075	4/53
8	0	0	1	1	0	0	0	1	45	1	0.082	4/49
9	0	0	0	2	1	0	0	1	0	45	0.082	4/49
Total	0	0	0	0	0	0	0	0	0	0	0.056	28/496

3.4 K-Nearest Neighbors

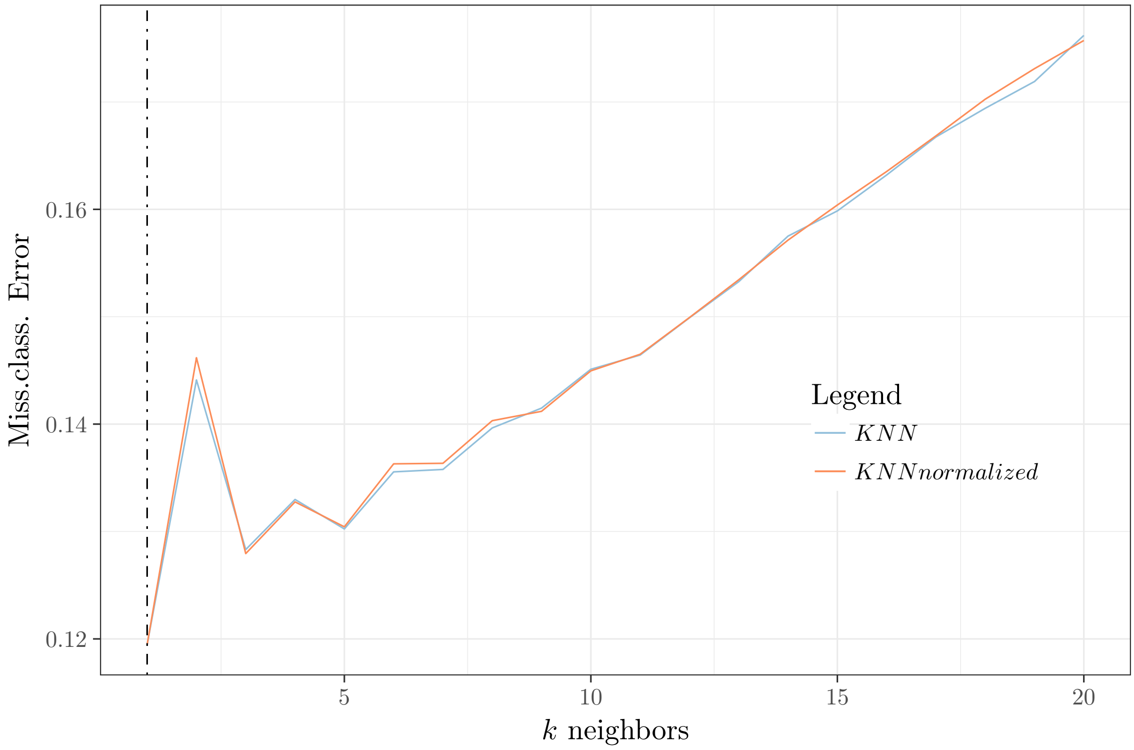


Figure 7

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	45	0	0	1	0	1	0	0	1	0	0.062	3/48
1	0	46	3	0	0	2	0	2	1	0	0.148	8/54
2	0	0	45	0	0	0	0	2	1	0	0.062	3/48
3	0	0	0	40	0	1	0	0	6	0	0.149	7/47
4	0	1	0	0	52	0	0	3	1	4	0.148	9/61
5	1	0	0	4	0	41	1	0	1	0	0.146	7/48
6	1	0	0	0	1	0	48	0	0	0	0.040	2/50
7	0	1	0	0	0	0	0	45	0	1	0.043	2/47
8	0	0	0	3	0	0	0	1	37	0	0.098	4/41
9	0	0	0	0	2	1	0	1	1	47	0.096	5/52
Total	0	0	0	0	0	0	0	0	0	0	0.101	50/496

4 Discussion

5 Acknowledgments

6 Appendices

6.1 R-Code: Help Functions

```r
require(caret)           # useful library to split up data set
library(tikzDevice)      # library to export plots to .tex files
library(xtable)          # library to export data frames to tables in .tex files

options(tikzMetricPackages = c("\\usepackage[utf8]{inputenc}", "\\usepackage[T1]{fontenc}",
                               "\\usetikzlibrary{calc}", "\\usepackage{amssymb}"))

ggplot_to_latex <- function(
    ggplot,
    destination_path,
    width,
    height
){
    tikz(file = paste0(destination_path, ".tex"), width = 6, height = 4)
    print(ggplot)
    dev.off()
}

create_confusion_matrix <- function(
    predicted_value,
    true_value,
    destination_path
){
    conf <- confusionMatrix(predicted_value, true_value)
    conf_df <- as.data.frame.matrix(conf$table) # extract confusion matrix
    # add row for total error
    conf_df <- rbind(conf_df, Total = rep(0, ncol(conf_df)))

    rows_in_df <- nrow(conf_df)

    classification_frac <- rep("", rows_in_df)
    classification_float <- rep(0, rows_in_df)

    total_wrong <- 0
    total_classified <- 0

    # make columns that shows accuracy
    for(i in 1:(rows_in_df - 1)){
        correct_classified <- conf_df[i, i]
        amount_classified <- sum(conf_df[i, ])
        missclassified <- amount_classified - correct_classified

        classification_frac[i] <- paste0(missclassified, "/", amount_classified)
        classification_float[i] <- missclassified / amount_classified

        total_wrong <- total_wrong + missclassified
        total_classified <- total_classified + amount_classified
    }

    classification_frac[rows_in_df] <- paste0(total_wrong, "/", total_classified)
    classification_float[rows_in_df] <- total_wrong / total_classified

    conf_df <- cbind(temp = row.names(conf_df),          # added extra column
```

```
                     conf_df,                                    # to get predicted classes
                     Error = classification_float,
                     Rate = classification_frac)
    names(conf_df) <- c("", names(conf_df)[-1]) # remove name of predicted classes

    write.csv(x = conf_df, file = paste0(destination_path, "_Confusion_Matrix.csv"))
    print(xtable(conf_df, display = c("s", rep("d", 11), "f", "s"),
                 digits = c(rep(0, 12), 3, 0)),
          #table.placement = "H",
          only.contents = TRUE,
          file = paste0(destination_path ,"_Confusion_Matrix.tex"),
          include.rownames = FALSE)

}

create_cv_indexes <- function(N, n_folds){
    indexes_per_fold <- floor(N/n_folds)
    index_matrix <- matrix(0L, nrow = n_folds, ncol = indexes_per_fold)
    index_available <- 1:N
    for(i in 1:n_folds){
        selected_indexes <- sample(index_available, indexes_per_fold)
        index_available <- index_available[! index_available %in% selected_indexes]

        index_matrix[i, ] <- selected_indexes
    }
    return(index_matrix)
}
```

../R_scripts/Help_Scripts/to_latex_functions.R

6.2 R-Code: Regression Tree

```
## Libraries and seed
rm(list = ls())
library(caret)
library(readr)
library(tree)
library(randomForest)
library(gbm)
library(tikzDevice)     # library to export plots to .tex files

options(tikzMetricPackages = c("\\usepackage[utf8]{inputenc}", "\\usepackage[T1]{fontenc}",
                               "\\usetikzlibrary{calc}", "\\usepackage{amssymb}"))

set.seed(420)

if(!exists("create_confusion_matrix", mode = "function")){
    source("Help_Scripts/to_latex_functions.R")
}

#-------------------#

## Data
path_data <- paste0(getwd(), "/data")
```

```r
path_to_here <- paste0(getwd(), "/Tree_Based_Methods")   # getwd give path to project
# which is one folder over

train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"), header = TRUE)
unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv"), header = TRUE)

# Remove unnessesary varibles which have a low variance
split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list = FALSE)
test_data <- train_data[-split_train_test, ]
train_data <- train_data[split_train_test, ]

# Remove variable with low variance which are near zero. Doing it after
# splitting in train/test set to avoid contaminating the data.
near_zero_variables <- nearZeroVar(train_data[,-1], saveMetrics = T, freqCut = 10000/1,uniqueCut = 1/7)
cut_variables <- rownames(near_zero_variables[near_zero_variables$nzv == TRUE,])
variables <- setdiff(names(train_data), cut_variables)
train_data <- train_data[, variables]
test_data <- test_data[, variables]

train_data[, 1] <- as.factor(train_data[, 1])
test_data[, 1] <- as.factor(test_data[, 1])

#train_data[,1] <- as.factor(train_data[,1])

unclassified_data[,1] <- as.factor(unclassified_data[,1])

sum(near_zero_variables$nzv)

#-------------------#

## REGRESSION - tree

regression <- function(
    minimum_development,
    train_data,
    test_data
    ){
    # Chanage name of pixel columns to work with tikz library

    print(getClass(class(train_data)))
    colnames(train_data)[ 2:length(train_data[1,])] <- c(paste0("Pixel", 1:(length(train_data[1,]) - 1)))
    colnames(test_data)[ 2:length(train_data[1, ])] <- c(paste0("Pixel", 1:(length(train_data[1,]) - 1)))
    minimum_development <- 0.005
    tree_model <- tree(Digit ~ ., data = train_data, mindev = minimum_development)
    plot(tree_model)
    text(tree_model, cex = .5)
    print(summary(tree_model))

    cross_validation <- cv.tree(tree_model, K = 10)
    cross_validation$k[1] <- 0
```

```
    alpha <- round <- round(cross_validation$k)

    plot(cross_validation$size, cross_validation$dev, type = "b",
         xlab = "Number of terminal nodes", ylab = "CV error")

    ggplot_df <- data.frame(size = cross_validation$size, dev = cross_
        validation$dev)

    destination_path <- paste0(path_to_here, "/Results_TBM/Regression_Tree")

    ggplot1 <- ggplot(data = ggplot_df, aes(x = size, y = dev)) +
                geom_line(aes(colour = "$RegressionTree$"), linetype = "
                    dashed") +
                geom_point() +
                geom_vline(xintercept = 20, color = "black", linetype = "
                    dotdash") +
                xlab("n\\_{terminal nodes}") +
                ylab("CV Error") +
                scale_colour_manual("Legend",
                                    breaks = c("$RegressionTree$"),
                                    values = c("#91bfdb"),
                                    guide = guide_legend(override.aes = list(
                                        linetype = c("solid"),
                                        shape = c(16)
                                    ))) +
                theme_bw() +
                theme(legend.position = c(0.8, 0.355),
                   legend.background = element_rect(fill=alpha('white', 0)))
    ggsave(paste0(destination_path, ".png"))

    ggplot_to_latex(ggplot1, destination_path, width = 5, height = 5)
    tree_prune <- prune.tree(tree_model, best = 20)
    summary(tree_prune)

    tikz(file = paste0(destination_path, "_Tree.tex"), width = 6, height =
        4)
    plot(tree_prune)
    text(tree_prune, cex = .5)
    dev.off()

    predicted <- predict(tree_prune, test_data, type = "class")
    create_confusion_matrix(predicted, test_data[,1], destination_path)
    }

regression(0.05, train_data, test_data)
#-------------------#

## RANDOM FORREST -randomForest



#-------------------#

## BOOSTING - gbm

#-------------------#
```

../R_scripts/Tree_Based_Methods/Regression_Tree.R

6.3 R-Code: Random Forest

```r
## Libraries and seed
rm(list = ls())
library(randomForest)   # library giving a easy-to-use random forest method
library(caret)          # useful library to split up data set
library(tikzDevice)     # library to export plots to .tex files
library(xtable)         # library to export data frames to tables in .tex files
set.seed(420)           # seed to replicate results and get consistent test and training set

# Load help script with functions to export the results to latex
# These functions gathered to avoid duplicate code
if(!exists("create_confusion_matrix", mode = "function")){
    source("Help_Scripts/to_latex_functions.R")
}

#-------------------#

## Data
path_data <- paste0(getwd(), "/data")
path_to_here <- paste0(getwd(), "/Tree_Based_Methods")   # getwd give path to project
                                                         # which is one folder over

train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"), header = TRUE)
unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv"), header = TRUE)

train_data[,1] <- as.factor(train_data[, 1])

# split training set into training and test set

split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list = FALSE)
test_data <- train_data[-split_train_test, ]
train_data <- train_data[split_train_test, ]

#-------------------#

## Random forest
# Train forest
train_random_forest <- function(
    data,
    n_trees,
    minimum_development = 0.01
    ){
        random_forest <- randomForest(Digit ~ .,
                                      data = data,
                                      ntree = n_trees,
                                      #mindev = minimum_development,
                                      importance = TRUE,
                                      na.action = na.exclude)
        return(random_forest)
    }

```

```r
# Plot error as the number of trees increase

plot_error_development <- function(
    random_forest_data,
    destination_path
    ){
        error_data <- data.frame(n_trees = 1:nrow(random_forest_data$err.rate),
                                 error <- random_forest_data$err.rate[,"OOB"])

        write.csv(error_data, file = paste0(destination_path, ".csv"))

        ggplot1 <- ggplot(data = error_data, aes(x = n_trees)) +
            geom_line(aes(y = error, colour = "$Random forest")) +
            xlab("$n\\_{trees}$") +
            ylab("Miss. class. Error") +
            scale_colour_manual("Legend",
                                breaks = c("$Random forest$"),
                                values = c("black"),
                                guide = guide_legend(override.aes = list(
                                    linetype = c("solid"),
                                    shape = c( 16)
                                ))) +
            theme(legend.position = c(0.9, 0.2))
        ggsave(paste0(destination_path, ".png"))

        ggplot_to_latex(ggplot1, destination_path, width = 6, height = 4)
}

main <- function(){
    n_trees = 50
    random_forest <- train_random_forest(train_data, n_trees)
    plot_error_development(random_forest, paste0(path_to_here, "/Results_TBM/Random_Forest_",
                                                 n_trees, "trees_Error_plot"))

    prediction <- predict(random_forest, newdata = test_data)
    create_confusion_matrix(predicted_value = prediction, true_value = test_data$Digit,
                            paste0(path_to_here, "/Results_TBM/Random_Forest_",
                                                              n_trees, "trees"))
}

main()
```

../R_scripts/Tree_Based_Methods/Random_Forest.R

6.4 R-Code: Bagging

```r
## Libraries and seed
rm(list = ls())
library(randomForest)   # library giving a easy-to-use random forest method
library(caret)          # useful library to split up data set
library(tikzDevice)     # library to export plots to .tex files
library(xtable)         # library to export data frames to tables in .tex files
set.seed(420)           # seed to replicate results and get consistent test and training set

# Load help script with functions to export the results to latex
# These functions gathered to avoid duplicate code
if(!exists("create_confusion_matrix", mode = "function")){
    source("Help_Scripts/to_latex_functions.R")
}

#-------------------#

## Data

path_data <- paste0(getwd(), "/data")
path_to_here <- paste0(getwd(), "/Tree_Based_Methods")   # getwd give path to project
# which is one folder over

train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"), header = TRUE)
unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv"), header = TRUE)

train_data[,1] <- as.factor(train_data[, 1])

# split training set into training and test set

split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list = FALSE)
test_data <- train_data[-split_train_test, ]
train_data <- train_data[split_train_test, ]

#-------------------#

## Random forest
# Train forest
train_bagging <- function(
    data,
    n_trees,
    minimum_development = 0.01
){
    n_features <- ncol(data) - 1
    bagging <- randomForest(Digit ~ .,
                                  data = data,
                                  ntree = n_trees,
                                  #mindev = minimum_development,
                                  mtry = n_features,
                                  importance = TRUE,
                                  na.action = na.exclude)
    return(bagging)
```

```r
}

# Plot error as the number of trees increase

plot_error_development <- function(
    random_forest_data,
    destination_path
){
    error_data <- data.frame(n_trees = 1:nrow(random_forest_data$err.rate),
                             error <- random_forest_data$err.rate[,"OOB"])

    write.csv(error_data, file = paste0(destination_path ,".csv"))
    ggplot1 <- ggplot(data = error_data, aes(x = n_trees)) +
        geom_line(aes(y = error, colour = "$Bagging$")) +
        xlab("n\\_{trees}") +
        ylab("Miss.class. Error") +
        scale_colour_manual("Legend",
                            breaks = c("$Bagging$"),
                            values = c("black"),
                            guide = guide_legend(override.aes = list(
                                linetype = c("solid"),
                                shape = c( 16)
                            ))) +
        theme(legend.position = c(0.9, 0.2))
    ggsave(paste0(destination_path, ".png"))

    ggplot_to_latex(ggplot1, destination_path, width = 6, height = 4)
}

main <- function(){
    n_trees <- 50
    bagging <- train_bagging(train_data, n_trees)
    plot_error_development(bagging, paste0(path_to_here, "/Results_TBM/Bagging_",
                                           n_trees, "trees_Error_plot"))

    prediction <- predict(bagging, newdata = test_data)

    create_confusion_matrix(predicted_value = prediction, true_value = test_data$Digit,
                            paste0(path_to_here, "/Results_TBM/Bagging_",
                                   n_trees, "trees"))
}

main()
```
../R_scripts/Tree_Based_Methods/Bagging.R

6.5 R-Code: Boosting

```
## Libraries and seed
rm(list = ls())
library(caret)            # useful library to split up data set
library(tikzDevice)       # library to export plots to .tex files
library(gbm)              # library with powerful boosting method
library(xtable)           # library to export data frames to tables in .tex files
set.seed(420)             # seed to replicate results and get consistent test and training set

# Load help script with functions to export the results to latex
# These functions gathered to avoid duplicate code
if(!exists("create_confusion_matrix", mode = "function")){
    source("Help_Scripts/to_latex_functions.R")
}

#--------------------#

## Data

path_data <- paste0(getwd(), "/data")
path_to_here <- paste0(getwd(), "/Tree_Based_Methods")   # getwd give path to project

train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"), header = TRUE)
unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv"), header = TRUE)

train_data[,1] <- as.factor(train_data[, 1])

# split training set into training and test set

split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list = FALSE)
test_data <- train_data[-split_train_test, ]
train_data <- train_data[split_train_test, ]

#--------------------#

## Boosting
# Train booster
boosting <- function(
    data,
    n_trees,
    minimum_development = 0.01,
    interaction_depth = 2,
    shrinkage = 0.001
){
    # boosting <- gbm(Digit ~ .,
    #                 data = data,
    #                 distribution = "multinomial",
    #                 n.trees = n_trees,
    #                 interaction.depth = interaction_depth,
    #                 shrinkage = shrinkage,
    #
    #                 bag.fraction = 1,
```

```
    #                  cv.folds = 10,
    #                  n.cores = 4)


    tune_control <- trainControl(method = "cv",
                                 number = 5,
                                 repeats = 1)
    training_grid <- expand.grid(n.trees = c(n_trees),
                                 interaction.depth = c(interaction_depth),
                                 shrinkage = c(shrinkage),
                                 n.minobsinnode = c(10))
    print(training_grid)
    boosting <- train(Digit ~ ., data = data, method = "gbm",
                      trControl = tune_control,
                      tuneGrid = training_grid)
    return(boosting)
}


# Plot error as the number of trees increase

plot_error_development <- function(
    boosting_data,
    destination_path
){
    error_data <- data.frame(n_trees = 1:length(boosting_data$cv.error),
                             error <- boosting_data$cv.error)
    write.csv(error_data, file = paste0(destination_path ,".csv"))

    ggplot1 <- ggplot(data = error_data, aes(x = n_trees)) +
        geom_line(aes(y = error, colour = "$Boosting$")) +
        xlab("$n_{trees}$") +
        ylab("Miss.class. Error") +
        scale_colour_manual("Legend",
                            breaks = c("$Boosting$"),
                            values = c("black"),
                            guide = guide_legend(override.aes = list(
                                linetype = c("solid"),
                                shape = c( 16)
                            ))) +
        theme(legend.position = c(0.9, 0.2)) +
        theme_bw() +
        theme(legend.position = c(0.8, 0.355),
              legend.background = element_rect(fill=alpha('white', 0)))
    ggsave(paste0(destination_path, ".png"))

    ggplot_to_latex(ggplot1, destination_path, width = 6, height = 4)
}

main <- function(){
    n_trees = 10
    boosting_train <- boosting(train_data,n_trees)
    plot_error_development(boosting_train, paste0(path_to_here,
                                                  "/Results_TBM/Boosting_",
                                                  n_trees,
                                                  "trees_Error_plot"))
    #predicted <- predict(boosting_train, test_data)
    #create_confusion_matrix(predicted, test_data$Digit, paste0(path_to_here
        ,
```

```
    #                                                                   "/Results_
        TBM/Boosting_",
    #                                                                   n_trees))
}

main()
```

../R_scripts/Tree_Based_Methods/Boosting.R

6.6 R-Code: Plot Random Forest w/ Bagging

```r
rm(list = ls())
library(ggplot2)
library(tikzDevice)      # library to export plots to .tex files

path_data <- paste0(getwd(), "/data")
path_to_here <- paste0(getwd(), "/Tree_Based_Methods")   # getwd give path to project
                                                         # which is one folder over

plot_random_forest_bagging <- function(
    n_trees,
    path,
    destination_path
    ){
        rf_path <- paste0(path,
                          "Random_Forest_",
                          n_trees,
                          "trees_Error_plot_",
                          n_trees,
                          "trees.csv")
        bagging_path <- paste0(path,
                          "Bagging_",
                          n_trees,
                          "trees_Error_plot_",
                          n_trees,
                          "trees.csv")

        random_forest_error <- read.csv(rf_path)
        bagging_error <- read.csv(bagging_path)

        ggplot_df <- data.frame(n_trees = 1:nrow(bagging_error),
                                rf = random_forest_error[3],
                                bag = bagging_error[3])
        names(ggplot_df) <- c("n_trees", "rf", "bag")
        print(str(ggplot_df))

        tikz(file = paste0(destination_path, ".tex"), width = 6, height = 4)
        ggplot1 <- ggplot(data = ggplot_df, aes(x = n_trees)) +
            geom_line(aes(y = rf, colour = "$Random forest$")) +
            geom_line(aes(y = bag, colour = "$Bagging$")) +
            geom_vline(xintercept = 150, color = "black", linetype = "dotdash") +
            xlab("n\\_{trees}") +
            ylab("Miss.class. Error") +
            scale_colour_manual("Legend",
                                breaks = c("$Random forest$", "$Bagging$"),
                                values = c("#91bfdb", "#fc8d59"),
                                guide = guide_legend(override.aes = list(
                                    linetype = c("solid", "solid"),
                                    shape = c( 16, 16)
                                ))) +
            theme_bw() +
            theme(legend.position = c(0.8, 0.355),
                  legend.background = element_rect(fill=alpha('white', 0)))
        ggsave(paste0(destination_path, ".png"))
        print(ggplot1)
```

```
        dev.off()
    }

main <- function(

    ){
        n_trees = 500
        path = paste0(path_to_here, "/Results_TBM/")
        destination_path = paste0(path, "/Random_Forest_Bagging_",
                                  n_trees, "trees")

        plot_random_forest_bagging(n_trees, path, destination_path)

    }

main()
```

../R_scripts/Tree_Based_Methods/Plot_Random_Forest_Bagging.R

6.7 R-Code: Neural Network

```r
## Libraries and seed
rm(list = ls())
library(h2o)
library(caret)
library(reshape2)

set.seed(420)

# Load help script with functions to export the results to latex
# These functions gathered to avoid duplicate code
if(!exists("create_confusion_matrix", mode = "function")){
    source("Help_Scripts/to_latex_functions.R")
}
#-------------------#

## Data

path_data <- getwd()
path_to_here <- paste0(getwd(), "/Neural_Networks")

train_data <- read.csv(paste0(path_data, "/data/Train_Digits_20171108.csv"))
unclassified_data <- read.csv(paste0(path_data, "/data/Test_Digits_20171108.
    csv"))

local.h2o <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,max_
    mem_size = "7G", nthreads = -1)

train_data[,1] <- as.factor(train_data[, 1])
split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
    FALSE)
test_data <- train_data[-split_train_test, ]
train_data <- train_data[split_train_test, ]

train_data <- as.h2o(train_data)
unclassified_data <- as.h2o(unclassified_data)
test_data <- as.h2o(test_data)

#-------------------#

## Getting useful data from grid run of neural networkss

get_data_in_df <- function(
    data
)
{
    n <- length(data@model_ids)
    mse_errors <- rep(0,n)
    mean_per_class_errors <- rep(0,n)
    hidden <- rep("", n)
    str(hidden)
    rate <- rep(0,n)
    l1 <- rep(0,n)
    epochs <- rep(0,n)
    model_numbers <- rep(0,n)
    train_error <- rep(0,n)
    train_mse <- rep(0,n)
    test_error <- rep(0,n)
```

```r
    test_mse <- rep(0,n)
    activation <- rep("",n)
    input_dropout_ratio <- rep(0,n)
    nesterov_accelerated_gradient <- rep("", n)

    model_df <- data.frame(model_numbers = mse_errors, hidden, rate, l1, epochs,
                           train_error, test_error, train_mse, test_mse, activation, input_dropout_ratio, stringsAsFactors = FALSE)
    str(model_df)

    for(i in 1:n){
        model <- h2o.getModel(data@model_ids[[i]])
        model_df$mse_errors[i] <- h2o.mse(model)
        #model_df$mean_per_class_error[i] <- model@model$cross_validation_metrics@metrics$mean_per_class_error
        model_df$mean_per_class_error[i] <- h2o.performance(model, xval = T)@metrics$mean_per_class_error

        model_paramaters <- model@allparameters
        model_name <- model@model_id
        model_number <- sub(".*model_(.*)$", "\\1", model_name)
        model_df$model_numbers[i] <- as.integer(model_number)
        model_df$hidden[i] <- paste(as.character(model_paramaters$hidden), sep = " ", collapse = ", ")
        model_df$rate[i] <- model_paramaters$rate
        model_df$l1[i] <- model_paramaters$l1
        model_df$epochs[i] <- model_paramaters$epochs
        model_df$activation[i] <- model_paramaters$activation
        model_df$input_dropout_ratio[i] <- model_paramaters$input_dropout_ratio
        model_df$nesterov_accelerated_gradient[i] <- model_paramaters$nesterov_accelerated_gradient

        #print(model)
        train_performance <- h2o.performance(model, train_data)@metrics
        train_performance_error <- train_performance$mean_per_class_error
        train_performance_mse <- train_performance$MSE

        model_df$train_error[i] <- train_performance_error
        model_df$train_mse[i] <- train_performance_mse

        test_performance <- h2o.performance(model, test_data)@metrics
        test_predictions <- h2o.predict(model, test_data)
        test_accuracy <- test_predictions$predict == test_data$Digit
        test_performance_error <- 1 - mean(test_accuracy)
        #test_performance_error <- test_performance$mean_per_class_error
        test_performance_mse <- test_performance$MSE

        model_df$test_error[i] <- test_performance_error
        model_df$test_mse[i] <- test_performance_mse

    }
    model_df <- model_df[with(model_df, order(model_numbers)),]
    model_df
}

activation <- list("Rectifier", "RectifierWithDropOut")# "Tanh")
```

```r
hidden <- list(c(100,100), c(150, 150),  c(540, 320), c(100, 100, 100), c(540, 320, 100))
input_dropout_ratio <- list(0, 0.2)
nesterov_accelerated_gradient <- list( TRUE)
epochs <- list(20)
l1 = list(0,1.4e-5)
hyper_params <- list(activation = activation, hidden = hidden, input_dropout_ratio = input_dropout_ratio, nesterov_accelerated_gradient = nesterov_accelerated_gradient, epochs = epochs, l1 = l1)

grid_deep_learning <- h2o.grid(algorithm = "deeplearning",
                               x = 2:785,
                               y = 1,
                               training_frame = train_data,
                               nfolds = 10,
                               stopping_metric = "MSE",
                               stopping_tolerance = 0.0025,
                               hyper_params = hyper_params)
    save_results <- function(results){
    write.csv(results, file = paste0(path_to_here, "/Neural_Networks/results_NN/grid_run_20.csv"))
}

df <- get_data_in_df(grid_deep_learning)
save_results(df)

#results_df <- df

results_df <- read.csv(paste0(path_to_here, "/Neural_Networks/results_NN/grid_run_40.csv"))

results_df <- results_df[with(results_df, order(mean_per_class_error)),]
results_df$row_names <- 1:length(results_df[,1])

melt_datas <- melt(results_df[c("test_error","mean_per_class_error", "row_names",
                                "model_numbers")], id = c("row_names", "model_numbers"))
plot_list  <- list()
# Plot classification error
plot_list[[1]] <- ggplot(data=melt_datas,
                         aes(x=row_names, y=value)) +
    geom_point(aes(colour = as.factor(model_numbers), group = as.factor(model_numbers)), size = 1.25) +
    geom_line(aes(group = variable)) +
    xlab("Model id") +
    ylab("Miss. class. Error") +
    scale_y_continuous(limits = c(0, 0.2)) +
    theme_bw() +
    theme(legend.position = c(0.675, 0.255),
          legend.background = element_rect(fill=alpha('white', 0)),
          legend.direction = "horizontal",
          legend.text = element_text(size=6),
          legend.key = element_rect(size = 3),
          legend.key.size = unit(1.0, 'lines'),
          axis.text.x=element_blank(),
          axis.ticks.x=element_blank()) +
    scale_colour_discrete(name = "Model ids") +
    guides(fill = guide_legend(title = "Legend"))
```

```r

ggplot_to_latex(plot_list[[1]],
                paste0(path_to_here, "/results_NN/per_class_error"), width = 6, height = 4)
ggsave(paste0(path_to_here,"/results_NN/per_class_error3.png"))

#                              deep_learning_predicting <- h2o.predict(object = deep_learning_results, newdata = test_data)
#deep_learning_performance <- h2o.performance(model = deep_learning_results3, newdata = test_data)
#deep_learning_performance
#deep_learning_predicting_data_frame <- as.data.frame((deep_learning_predicting))


#deep_learning_results2 <- h2o.deeplearning(x = 2:785,
#                                            y = 1,
#                                           training_frame = train_data,
#                                            activation = "Tanh",
#                                            hidden = c(160, 160, 160, 160, 160),
#                                            nfolds = 10,
#                                            keep_cross_validation_predictions = TRUE,
#                                            epochs = 40)

deep_learning_results3<- h2o.deeplearning(x = 2:785,
                                          y = 1,
                                          training_frame = train_data,
                                          #activation = "RectifierWithDropout",
                                          activation = "Rectifier",
                                          input_dropout_ratio = 0.2,
                                          #hidden_dropout_ratios = c(0.2, 0.2, 0.2),
                                          nfolds = 10,
                                          balance_classes = TRUE,
                                          hidden = c(540, 320),
                                          l1 = 1.4e-5,
                                          #momentum_stable = 0.99,
                                          stopping_metric = "MSE",
                                          stopping_tolerance = 0.0025,
                                          nesterov_accelerated_gradient = TRUE,
                                          epochs = 20)

h2o.performance(deep_learning_results3, test_data)


predicted <- predict(deep_learning_results3, test_data, type = "response")

predicted_confusion_matrix <- as.factor(as.vector(predicted$predict))
test_data_confusion_matrix <- as.data.frame(test_data)
create_confusion_matrix(predicted_confusion_matrix,
                        test_data_confusion_matrix[, "Digit"],
                        paste0(path_to_here, "/results_NN/540_320_neural_net"))

```

```
h2o.shutdown()
```

../R_scripts/Neural_Networks/uneural_network.R

6.8 R-Code: Convolutional Neural Network

```
## Libraries and seed
rm(list = ls())
library(mxnet)
library(caret)
set.seed(420)

#-------------------#

## Data

path_to_here <- getwd()

train_data <- read.csv(paste0(path_to_here, "/data/Train_Digits_20171108.csv
    "), header = TRUE)
unclassified_data <- read.csv(paste0(path_to_here, "/data/Test_Digits_
    20171108.csv"), header = TRUE)

train_data[,1] <- as.factor(train_data[, 1])

# split training set into training and test set

split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
    FALSE)
test_data <- train_data[-split_train_test, ]
train_data <- train_data[split_train_test, ]

# convert to matrix, required by "mxnet"

train <- data.matrix(train_data)
test <- data.matrix(test_data)

train_x <- t(train[, -1]/255)
train_y <- train[, 1]

train_array <- train_x
dim(train_array) <- c(28, 28, 1, ncol(train_x))

#train_x <- t(train_x)#/255)

test_x <- test[, -1]
test_y <- test[, 1]

test_x <- t(test_x/255)

# transpose and normalize to more

#-------------------#

## Setting up Convolutional Neural Network(CNN)

data <- mx.symbol.Variable("data")
fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")
fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64)
act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu")
fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=10)
softmax <- mx.symbol.SoftmaxOutput(fc3, name="sm")
```

```
devices <- mx.cpu()

mx.set.seed(0)

model <- mx.model.FeedForward.create(softmax, X=train_x, y=train_y,
                                     ctx=devices, num.round=10, array.batch.size=100,
                                     learning.rate=0.07, momentum=0.9, eval.metric=mx.metric.accuracy,
                                     initializer=mx.init.uniform(0.07),
                                     epoch.end.callback=mx.callback.log.train.metric(100))

preds <- predict(model, test_x)
```

../R_scripts/Neural_Networks/convolutional_neural_network.R

6.9 R-Code: K-nearest Neighbours

```r
## Libraries and seed
rm(list = ls())

library(foreach)        # library for running loop in parallel
library(doParallel)     # library for running loop in parallel
library(caret)          # useful library to split up data set
library(tikzDevice)     # library to export plots to .tex files
library(xtable)         # library to export data frames to tables in .tex files
library(kknn)
library(class)
set.seed(420)           # seed to replicate results and get consistent test and training set

# Load help script with functions to export the results to latex
# These functions gathered to avoid duplicate code
if(!exists("create_confusion_matrix", mode = "function")){
    source("Help_Scripts/to_latex_functions.R")
}

#-------------------#

## Data
path_data <- paste0(getwd(), "/data")
path_to_here <- paste0(getwd(), "/Tree_Based_Methods")   # getwd give path to project
# which is one folder over


train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"), header = TRUE)
unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv"), header = TRUE)

train_data[,1] <- as.factor(train_data[, 1] )

# split training set into training and test set

split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list = FALSE)
test_data <- train_data[-split_train_test, ]
train_data <- train_data[split_train_test, ]


k_nearest_neighbors <- function(
    train_data,
    train_data_norm,
    k_folds,
    k_neighbors = 20
){
    # First find number of k(neighbors) using crossvalidation
    cv_indexes <- create_cv_indexes(nrow(train_data), k_folds)

    cores=detectCores()
    cl <- makeCluster(cores[1]-1) #not to overload your computer
    registerDoParallel(cl)

```

```r
    cv_error <- c()
    indexes <- 1:nrow(train_data)

    final_df <- foreach(i = 1:k_neighbors,
                        .combine = "rbind",
                        .packages = "class") %dopar%{
            error_kfolds <- 0
            error_kfolds_norm <- 0
            for(k in 1:k_folds){
                train_val <- train_data[!indexes %in% cv_indexes[k, ], ]
                validation_val <- train_data[cv_indexes[k, ],]

                train_val_norm <- train_data_norm[!indexes %in% cv_indexes[k, ], ]
                validation_val_norm <- train_data_norm[cv_indexes[k, ],]

                # without normalization
                knn_pred_class <- knn(train_val[-1],
                                      validation_val[-1],
                                      train_val[, 1],
                                      k = i)
                # with normalization
                knn_pred_class_norm <- knn(train_val_norm[-1],
                                           validation_val_norm[-1],
                                           train_val_norm[, 1],
                                           k = i)

                error <- 1 - mean(validation_val[, 1] == knn_pred_class)
                error_norm <- 1 - mean(validation_val_norm[, 1] == knn_pred_class_norm)
                error_kfolds <- error_kfolds + error
                error_kfolds_norm <- error_kfolds_norm + error_norm
            }
            temp_df <- data.frame(k_neighbors = i,
                                  error_kfolds = error_kfolds,
                                  error_kfolds_norm = error_kfolds_norm)
            temp_df
    }
    final_df[, 2:3] <- final_df[, 2:3] / k_folds
    stopCluster(cl)
    return(final_df)
}

average_cv_error <- function(
    train_data,
    test_data,
    train_data_norm,
    test_data_norm,
    n_avg
){
    k_neighbors <- 20

    # set up error data frame
    error_df <- data.frame(k_neighbors = 1:k_neighbors,
                           error_kfolds = rep(0, k_neighbors),
                           error_kfolds_norm = rep(0, k_neighbors))

    # progress bar
    pb <- txtProgressBar(min = 0, max = n_avg, style = 3)
```

```

    # average cv-error over several runs to get more precise measure
    for(i in 1:n_avg){
        setTxtProgressBar(pb, i)
        error_df[-1] <- error_df[-1] + k_nearest_neighbors(train_data,
                            train_data_norm,
                            2,
                            k_neighbors)[-1]
    }
    close(pb)
    error_df[-1] <- error_df[-1]/n_avg
    ggplot1 <- ggplot(data = error_df, aes(x = k_neighbors)) +
        geom_line(aes(y = error_kfolds, colour = "$KNN$")) +
        geom_line(aes(y = error_kfolds_norm, colour = "$KNN normalized$")) +
        geom_vline(xintercept = 1, color = "black", linetype = "dotdash") +
        xlab("$k$ neighbors") +
        ylab("Miss.class. Error") +
        scale_colour_manual("Legend",
                            breaks = c("$KNN$", "$KNN normalized$"),
                            values = c("#91bfdb", "#fc8d59"),
                            guide = guide_legend(override.aes = list(
                                linetype = c("solid", "solid"),
                                shape = c( 16, 16)
                            ))) +
        theme_bw() +
        theme(legend.position = c(0.8, 0.355),
              legend.background = element_rect(fill=alpha('white', 0)))
    ggplot_to_latex(ggplot1,
                    paste0("K_Nearest_Neighbors/Results_KNN/knn_error_compare",n_avg),
                    width = 5, height = 5)
    return(error_df)
}

predict_on_test <- function(
    best_k,
    train_data,
    test_data
){
    knn_pred_class <- knn(train_data[-1],
                               test_data[-1],
                               train_data[, 1],
                               k = best_k)
    create_confusion_matrix(predicted_value = knn_pred_class,
                            true_value = test_data[, 1],
                            destination_path = "K_Nearest_Neighbors/Results_KNN/")
}
#cv_final <- k_nearest_neighbors(train_data, 10)
train_data_norm <- train_data
train_data_norm[-1] <- train_data_norm[-1]/255
test_data_norm <- test_data
test_data_norm[-1] <- test_data_norm[-1]/255
#cv_norm <- k_nearest_neighbors(train_data, train_data_norm, 10)

# average_error <- average_cv_error(train_data = train_data,
#                   test_data = test_data,
#                   train_data_norm = train_data_norm,
#                   test_data_norm = test_data_norm,
```

```r
#                    n_avg = 30)

predict_on_test(1, train_data_norm, test_data_norm)

#knn <- kknn(Digit ~ ., train = train_val, test = validation_val[-1], k = i)
#knn_prediction <- fitted.values(knn)
```

../R_scripts/K_Nearest_Neighbors/kknn.R

6.10 R-Code: Support Vector Machines

```
## Libraries and seed
rm(list = ls())
library(e1071)          # library to make margins for svm
library(caret)          # useful library to split up data set
library(readr)

set.seed(420)           # seed to replicate results and get consistent test and training set

# Load help script with functions to export the results to latex
# These functions gathered to avoid duplicate code
if(!exists("create_confusion_matrix", mode = "function")){
    source("Help_Scripts/to_latex_functions.R")
}

#-------------------#

## Data
path_data <- paste0(getwd(), "/data")
path_to_here <- paste0(getwd(), "/Support_Vector_Machines")   # getwd give path to project
# which is one folder over

train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"), header = TRUE)
unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv"), header = TRUE)

train_data[,1] <- as.factor(train_data[, 1])

nzr <- nearZeroVar(train_data[,-1], saveMetrics = TRUE, freqCut = 10000/1, uniqueCut = 1/7)
sum(nzr$zeroVar)

sum(nzr$nzv)

cut_variables <- rownames(nzr[nzr$nzv == TRUE, ])
variable <- setdiff(names(train_data), cut_variables)
train_data <- train_data[, variable]

# split data into test and training set
split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list = FALSE)
test_data <- train_data[-split_train_test, ]
train_data <- train_data[split_train_test, ]

# remove label temporarely by storeing as it's own vector
digit <- train_data[1]
train_data$Digit <- NULL
train_data <- train_data/255
cov_train <- cov(train_data)

digit_test <- test_data[1]
test_data$Digit <- NULL
test_data <- test_data/255

# use prcomp to do PCA on the covariance matrix
```

```r
train_pc <- prcomp(cov_train)

# get information of about the variance
var_explained <- train_pc$sdev^2/sum(train_pc$sdev^2)
var_explained_cumsum <- cumsum(var_explained)
var_explained_df <- data.frame(number = 1:length(train_pc$sdev),
                               variance_explained = var_explained,
                               cumsum = var_explained_cumsum)
# see plot in end of script


# find optimal number of components
find_number_of_components <- function(
    components_range,
    increase_by
){
    components <- seq(from = components_range[1],
                      to = components_range[2],
                      by = increase_by)
    missclassification_error <- rep(0, length(components))
    cv_error <- rep(0, length(components))

    components_error_df <- data.frame(components,
                                      missclassification_error,
                                      cv_error)

    for(i in 1:length(components)){

        # traing the svm for the components given
        train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:components[i]]
        train_data_temp <- cbind(Digit = digit, as.data.frame(train_score))

        tune_svm <- tune.svm(Digit ~ ., data = train_data_temp, cost = 1:10, kernal = "radial")
        fit_svm <- tune_svm$best.model

        # review the performance

        predicted <- predict(fit_svm, train_score, type = "response")
        missclassification_error <- 1 - mean(train_data_temp[, "Digit"] == predicted)

        components_error_df[i, "cv_error"] <- tune_svm$best.performance
        components_error_df[i, "missclassification_error"] <- missclassification_error
    }
    return(components_error_df)
}

# make ggplot of cv error for different amount of components
# also marks selected number of components manually (22 in this task)
plot_components_error <- function(
    components_error_df
){
    components_chosen <- components_error_df[22, ]

    ggplot1 <- ggplot(data = components_error_df, aes(x = components)) +
        geom_line(aes(y = cv_error, colour = "$E_{CV}$")) +
```

```
        geom_point(aes(y = cv_error, colour = "$E_{CV}$")) +
        geom_vline(xintercept = 22, color = "black", linetype = "dotdash") +
        xlab("$n_{components}$") +
        ylab("Miss.class. Error") +
        geom_point(data = components_chosen,
                   aes(x = components, y = cv_error),
                   color = "red") +
        geom_text(data = components_chosen,
                  aes(y = cv_error,
                      label = paste0("$\\mathrm{CV}_{error}(22) = ",
                                     round(cv_error,4), "$")),
                  hjust = 0.2, vjust = -1.0) +
        scale_colour_manual("Legend",
                            breaks = c("$E_{CV}$"),
                            values = c("black"),
                            guide = guide_legend(override.aes = list(
                                linetype = c("solid"),
                                shape = c( 16)
                            ))) +
        theme_bw() +
        theme(legend.position = c(0.8, 0.255),
              legend.background = element_rect(fill=alpha('white', 0)))
    ggplot_to_latex(ggplot1,
                    paste0(path_to_here,
                           "/Results_SVM/number_of_components_cv_error"),
                    width = 5, height = 5)
}

# Calculate best cost parameter for svm
find_optimal_cost_for_components <- function(
    selected_components,
    cost_range,
    cost_increase,
    n_avg
){
    train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:selected_
       components]
    train_data_temp <- cbind(Digit = digit, as.data.frame(train_score))

    cost <- seq(from = cost_range[1], to = cost_range[2], by = cost_increase
       )

    cost_error <- data.frame(cost = cost, cv_error = rep(0, length(cost)))
    for(i in 1:n_avg){
        tune_svm <- tune.svm(Digit ~ ., data = train_data_temp, cost = cost,
             kernal = "radial")

        cv_error <- tune_svm$performances$error
        cost_error[, "cv_error"] <- cost_error[, "cv_error"] + cv_error

    }
    cost_error[, "cv_error"] <- cost_error[, "cv_error"]/n_avg

    best_cost <- cost_error[which(cost_error[, "cv_error"] == min(cost_error
       [, "cv_error"])),]

    ggplot1 <- ggplot(data = cost_error, aes(x = cost)) +
        geom_line(aes(y = cv_error, colour = "$E_{CV}$")) +
        geom_point(aes(y = cv_error, colour = "$E_{CV}$")) +
```

```r
        geom_vline(xintercept = best_cost[, "cost"],
                   color = "black", linetype = "dotdash") +
        xlab("$n_{components}$") +
        ylab("Miss.class. Error") +
        geom_point(data = best_cost,
                   aes(x = cost, y = cv_error),
                   color = "red") +
        geom_text(data = best_cost,
                  aes(y = cv_error,
                      label = paste0("$\\mathrm{CV}_{error}(",
                                     cost,
                                     ") = ",
                                     round(cv_error,4), "$")),
                  hjust = -0.05, vjust = 0.9) +
        scale_colour_manual("Legend",
                            breaks = c("$E_{CV}$"),
                            values = c("black"),
                            guide = guide_legend(override.aes = list(
                                linetype = c("solid"),
                                shape = c( 16)
                            ))) +
        theme_bw() +
        theme(legend.position = c(0.8, 0.255),
              legend.background = element_rect(fill=alpha('white', 0)))
    ggplot_to_latex(ggplot1,
                    paste0(path_to_here,
                           "/Results_SVM/cost_cv_error"),
                    width = 5, height = 5)
    return(best_cost)
}

# Make prediction on the test set and make confusion matrix
predict_on_test <- function(
    selected_components,
    cost
){
    train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:selected_
        components]
    train_data_temp <- cbind(Digit = digit, as.data.frame(train_score))

    test_score <- as.matrix(test_data) %*% train_pc$rotation[,1:selected_
        components]
    test_data_temp <- cbind(Digit = digit_test, as.data.frame(test_score))

    svm_predict <- svm(Digit ~ ., data = train_data_temp, cost = cost,
        kernal = "radial")

    predicted <- predict(svm_predict, test_data_temp)

    create_confusion_matrix(predicted_value = predicted,
                            true_value = test_data_temp[, "Digit"],
                            destination_path = paste0(path_to_here,
                                                      "/Results_SVM/
                                                          confusion_matrix")
                                                          )

}

# some help-plots for pca
```

```r
plot_variance_explained <- function(
    var_explained_df,
    chosen_number_components
){
    # plot amount of varibles and how much of the variance this describe
    ggplot1 <- ggplot(data = var_explained_df[1:100,], aes(x = number, y = cumsum)) +
        geom_line() +
        geom_point() +
        geom_vline(xintercept = chosen_number_components,
                   color = "black", linetype = "dotdash") +
        xlab("$n_{components}$") +
        ylab("Variance Explained") +
        theme_bw() +
        theme(legend.position = c(0.8, 0.255),
              legend.background = element_rect(fill=alpha('white', 0)))

    # plot the data as described by the two first principle components

    train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:chosen_number_components]
    train_data_temp <- cbind(Digit = digit, as.data.frame(train_score))

    ggplot2 <- ggplot(data = train_data_temp, aes(x = PC1, y = PC2)) +
        geom_point(aes(colour = Digit)) +
        xlab("PCA 1") +
        ylab("PCA 2") +
        scale_colour_manual("Legend",
                            values = c('#8dd3c7','#ffffb3','#bebada','#fb8072',
                                       '#80b1d3','#fdb462','#b3de69','#fccde5',
                                       '#d9d9d9','#bc80bd')
                            ) +
        theme_bw() +
        theme(legend.position = c(0.9545, 0.355),
              legend.background = element_rect(fill=alpha('white', 0)))

    ggplot_to_latex(ggplot1,
                    paste0(path_to_here,
                           "/Results_SVM/variance_explained_pca"),
                    width = 5, height = 5)

    ggplot_to_latex(ggplot2,
                    paste0(path_to_here,
                           "/Results_SVM/map_pca1_pca2"),
                    width = 5, height = 5)
}

main <- function(){
    # Find best number of components to use with svm
    components_range <- c(1, 30)
    increase_by <- 1
    components_error_df <- find_number_of_components(components_range = components_range,
                                                     increase_by = increase_by)
    # Plot best cv error over number of components
    plot_components_error(components_error_df)
```

```


    selected_components <- 22

    # Find optimal cost parameter for the svm function
    best_cost <- find_optimal_cost_for_components(cost_range = c(0.5, 10),
                                                  cost_increase = 0.5,
                                                  selected_components =
                                                     selected_compnents,
                                                  n_avgs = 5)
    # Plot pca variance and to principle components
    plot_variance_explained(var_explained_df, selected_components)
}

main()
```

../R_scripts/Support_Vector_Machines/support_with_pca.R