Machine Learning: Assignment 2

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 simular 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 sucessful this is very appeling in todays society when 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. 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.
COMMENT on data snooping

2 Methods

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

For Neural Networks I have decided to use a convulutional 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(multidimension 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 recongising images.

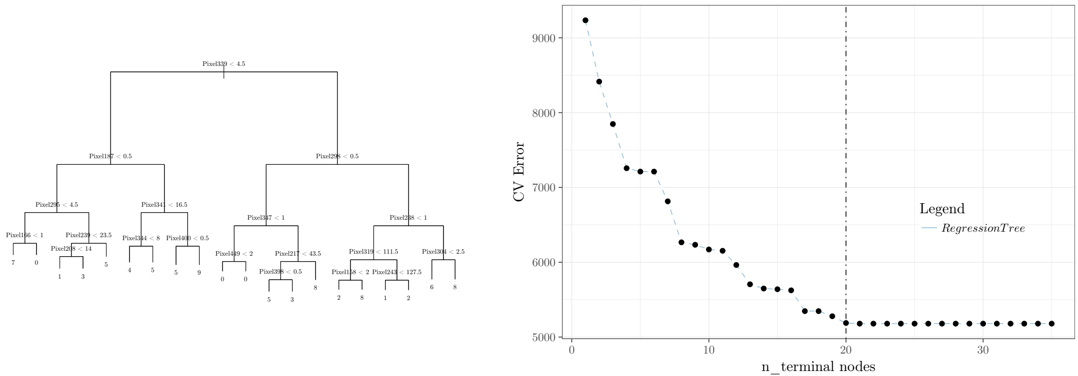
Figure 1: Error measures E_{out} and E_{val} are plotted for different sizes of training and validation sets.

2.3 Support vector machines

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

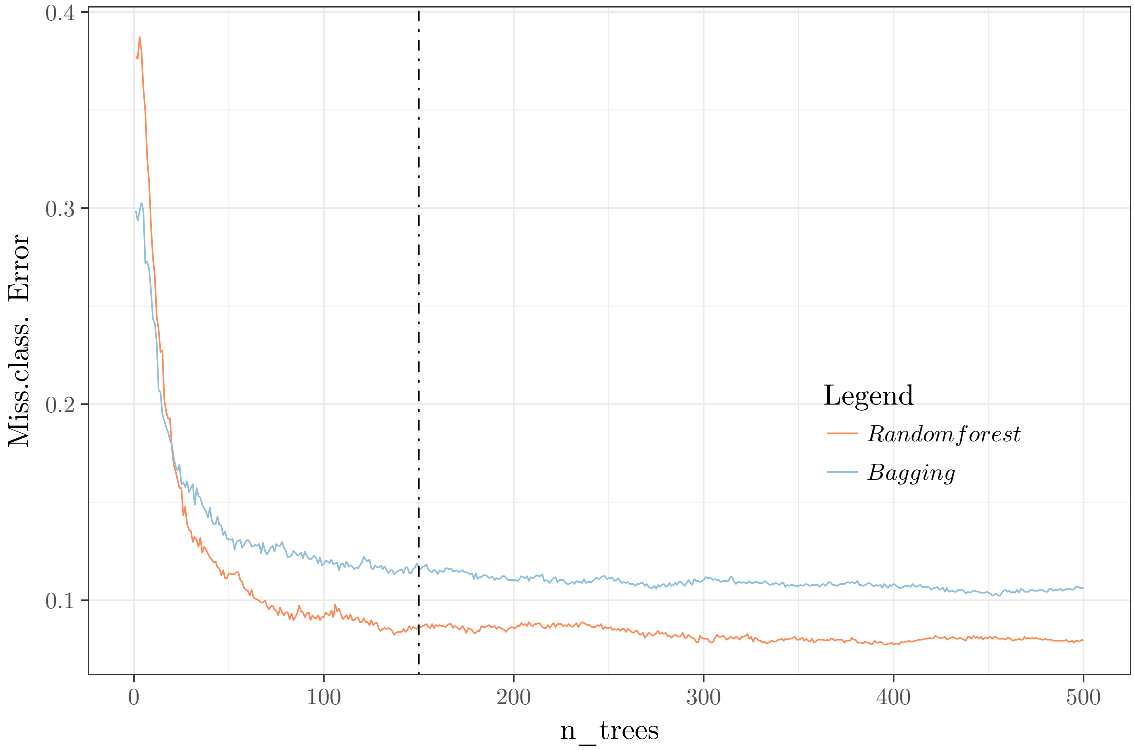


Figure 3: Error measures E_{out} and E_{val} are plotted for different sizes of training and validation sets.

3.2 Neural Networks

3.3 Support vector machines

4 Discussion

5 Acknowledgments

6 Appendices

6.1 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")
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)))
```

```r
    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

6.2 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

6.3 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)
```

```
}

# 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

6.4 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

6.5 R-Code: Neural Network

```r
## Libraries and seed
library(h2o)
library(caret)
library(reshape2)

set.seed(420)

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

## Data

path_to_here <- getwd()

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

local.h2o <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,
    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)
    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")
hidden <- list(c(100,100), c(150, 150), c(100, 100, 100)) #c(100, 100, 100))
    #, c(150, 150, 150))
input_dropout_ratio <- list(0, 0.2)
nesterov_accelerated_gradient <- list( TRUE, FALSE)
epochs <- list(20)#, 20)
l1 = list(1.4e-5)
```

```r
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_evenodd2.csv"))
}

df <- get_data_in_df(grid_deep_learning)
save_results(df)

results_df <- df

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 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 = 3) +
    geom_line(aes(group = variable)) +
    labs(y = "Missclassification error in range 0 to 1",
         x = "Models",
         title = "Missclassification error for training model and test set",
         caption = "Top - Training model, Bottom - Test set",
         colour = "Model id") +
    scale_y_continuous(limits = c(0, 0.2))
ggsave(paste0(path_to_here,"/Neural_Networks/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(150, 150, 150),
                                           momentum_stable = 0.99,
                                           nesterov_accelerated_gradient =
                                              TRUE,
                                           epochs = 15)

h2o.performance(deep_learning_results3, test_data)
```
../R_scripts/Neural_Networks/uneural_network.R

6.6 R-Code: Convolutional Neural Network

```r
## 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

6.7 R-Code: K-nearest Neighbours

```r
## 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(xtable)          # library to export data frames to tables in .tex files
library(kknn)
set.seed(420)            # seed to replicate results and get consistent test and training set

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

## 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, ]

knn_pred <- kknn(Digit ~ ., train = train_data[], test = test_data[-1], k=1)
k_pred <- fitted.values(knn_pred)
confusionMatrix(k_pred, test_data[,1])
```

../R_scripts/K_Nearest_Neighbors/kknn.R

6.8 R-Code: Support Vector Machines

```
## Libraries and seed
rm(list = ls())
library(e1071)
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(readr)

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

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

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

## 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_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, ]

label <- train_data[1]
train_data$Digit <- NULL
train_data <- train_data/255
cov_train <- cov(train_data)

train_pc <- prcomp(cov_train)
varex <- train_pc$sdev^2/sum(train_pc$sdev^2)
varcum <- cumsum(varex)
result <- data.frame(num = 1:length(train_pc$sdev),
                     ex = varex,
```

```r
                        cum = varcum)

plot(result$num, result$cum, type = "b", xlim = c(0,100))
abline(v=25, lty=2)


train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:25]
train_data <- cbind(label, as.data.frame(train_score))
colors <- rainbow(length(unique(train_data$Digit)))
names(colors) <- unique(train_data$label)

plot(train_data$PC1, train_data$PC2, type = "n", main = "First two Principal
     Components")
text(train_data$PC1, train_data$PC2, label = train_data$Digit, col = colors[
    train_data$Digit])

svm_model <- svm(Digit ~ ., data = train_data, cost = 8, kernel = "radial")

test_data2 <- test_data[-1]/255
test_data2 <- as.matrix(test_data2) %*% train_pc$rotation[,1:25]
test_data2 <- as.data.frame(test_data2)

predicted <- predict(svm_model, test_data2)

confusion_matrix <- confusionMatrix(predicted, test_data$Digit)
```

../R_scripts/Support_Vector_Machines/support_with_pca.R