

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

1	Introduction	1
2	Methods	3
2.1	Tree based methods	3
2.1.1	Classification tree	3
2.1.2	Random Forest	3
2.2	Neural Networks	3
2.2.1	Artificial?	3
2.2.2	Convolutional Neural Network	3
2.3	Support vector machines	4
2.4	K-Nearest Neighbours	4
3	Results	4
3.1	Tree based methods	4
3.1.1	Classification tree	4
3.1.2	Random Forest	5
3.2	Neural Networks	7
3.2.1	Artificial Neural Network	7
3.3	Support vector machines	8
3.4	K-Nearest Neighbors	9
4	Discussion	9
5	Acknowledgments	9

6	Appendices	9
6.1	R-Code: Help Functions	9
6.2	R-Code: Regression Tree	11
6.3	R-Code: Random Forest	14
6.4	R-Code: Bagging	16
6.5	R-Code: Boosting	18
6.6	R-Code: Plot Random Forest w/ Bagging	21
6.7	R-Code: Neural Network	23
6.8	R-Code: Convolutional Neural Network	28
6.9	R-Code: K-nearest Neighbours	30
6.10	R-Code: Support Vector Machines	34

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 $m \ll M$ 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

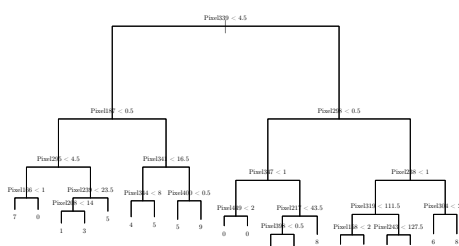
2.4 K-Nearest Neighbours

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

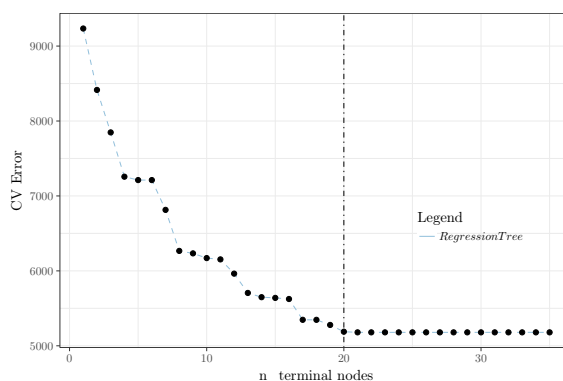
3 Results

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

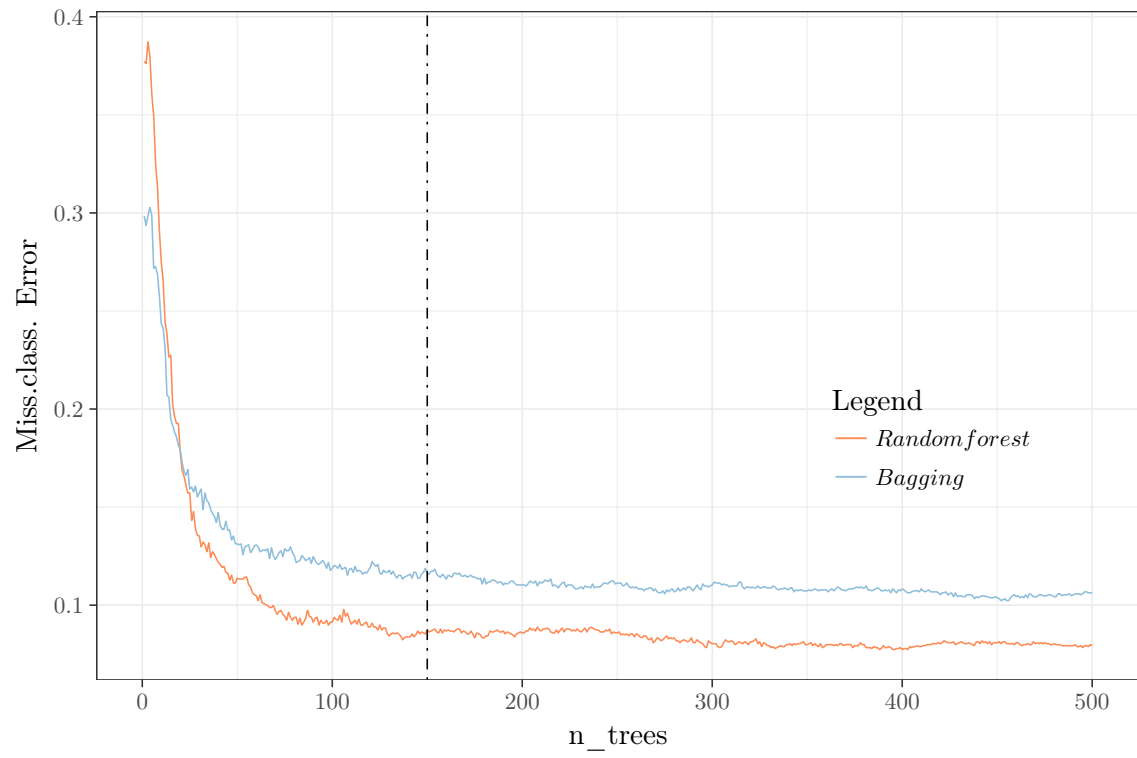


Figure 3

3.2 Neural Networks

3.2.1 Artificial Neural Network

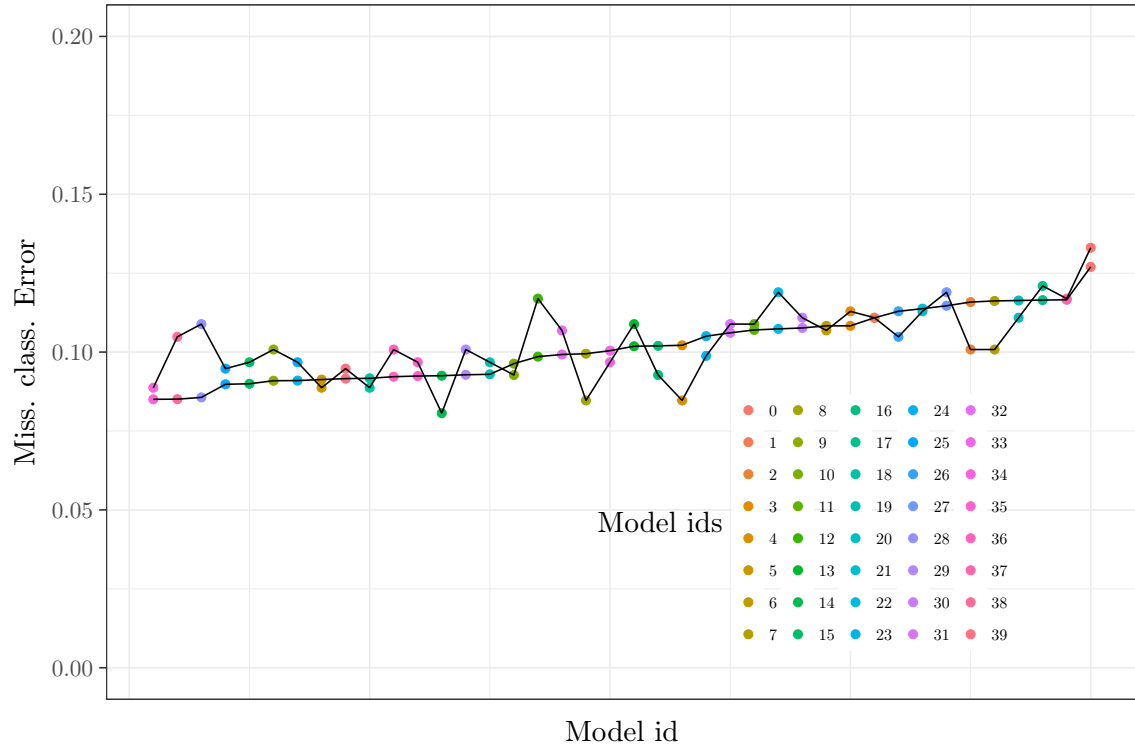
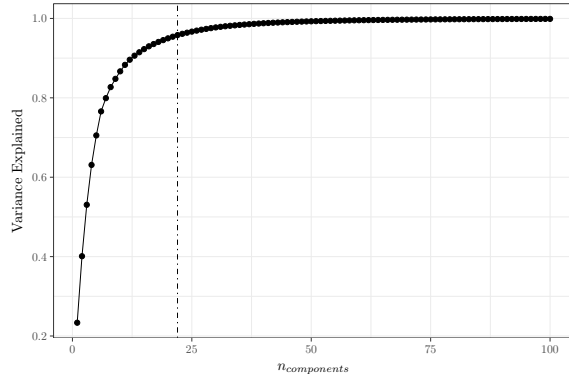


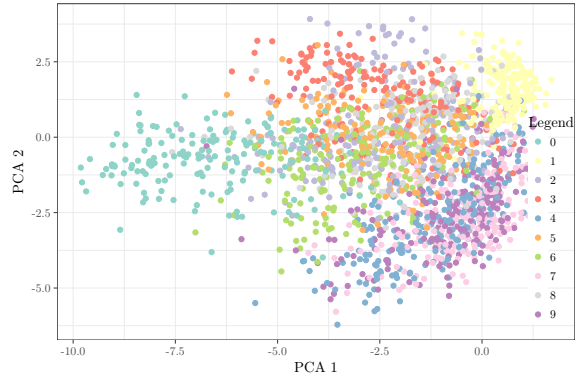
Figure 4

	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

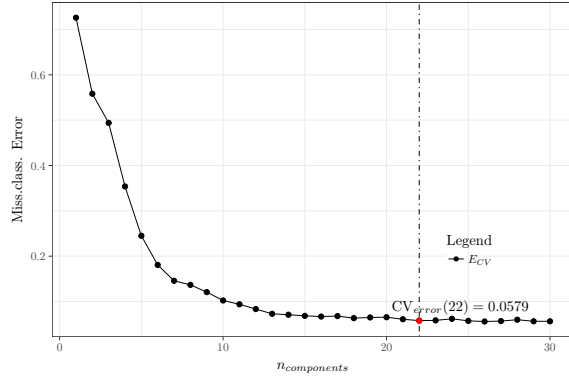
3.3 Support vector machines



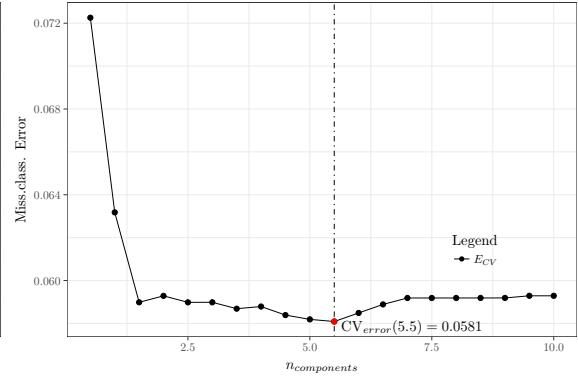
(a)



(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

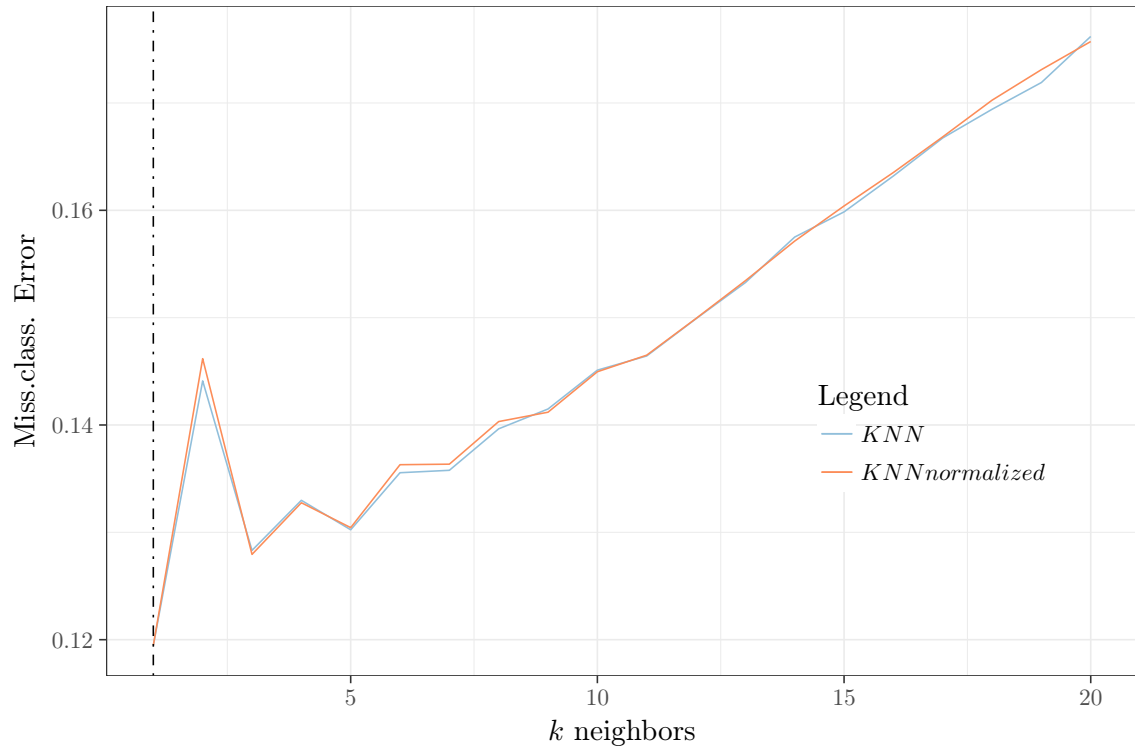


Figure 7

	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

```

1 require(caret)           # useful library to split up data set
2 library(tikzDevice)      # library to export plots to .tex files
3 library(xtable)          # library to export data frames to tables in .tex
   files
4
5 options(tikzMetricPackages = c("\\usepackage[utf8]{inputenc}", "\\usepackage
   [T1]{fontenc}",
6                                   "\\usetikzlibrary{calc}", "\\usepackage{
   amssymb}"))
7
8 ggplot_to_latex <- function(
9   ggplot,
10  destination_path,
11  width,
12  height
13 ){
14   tikz(file = paste0(destination_path, ".tex"), width = 6, height = 4)
15   print(ggplot)
16   dev.off()
17 }
18
19 create_confusion_matrix <- function(
20   predicted_value,
21   true_value,
22   destination_path
23 ){
24   conf <- confusionMatrix(predicted_value, true_value)
25   conf_df <- as.data.frame.matrix(conf$table) # extract confusion matrix
26   # add row for total error
27   conf_df <- rbind(conf_df, Total = rep(0, ncol(conf_df)))
28
29   rows_in_df <- nrow(conf_df)
30
31   classification_frac <- rep("", rows_in_df)
32   classification_float <- rep(0, rows_in_df)
33
34   total_wrong <- 0
35   total_classified <- 0
36
37   # make columns that shows accuracy
38   for(i in 1:(rows_in_df - 1)){
39     correct_classified <- conf_df[i, i]
40     amount_classified <- sum(conf_df[i, ])
41     missclassified <- amount_classified - correct_classified
42
43     classification_frac[i] <- paste0(missclassified, "/", amount_
       classified)
44     classification_float[i] <- missclassified / amount_classified
45
46     total_wrong <- total_wrong + missclassified
47     total_classified <- total_classified + amount_classified
48   }
49
50   classification_frac[rows_in_df] <- paste0(total_wrong, "/", total_
     classified)
51   classification_float[rows_in_df] <- total_wrong / total_classified
52
53   conf_df <- cbind(temp = row.names(conf_df),           # added extra
     column

```

```

54         conf_df,                                     # to get predicted
55         classes
56         Error = classification_float,
57         Rate = classification_frac)
58 names(conf_df) <- c("", names(conf_df)[-1]) # remove name of predicted
59         classes
60
61 write.csv(x = conf_df, file = paste0(destination_path, "_Confusion_
62 Matrix.csv"))
63 print(xtable(conf_df, display = c("s", rep("d", 11), "f", "s"),
64         digits = c(rep(0, 12), 3, 0)),
65         #table.placement = "H",
66         only.contents = TRUE,
67         file = paste0(destination_path, "_Confusion_Matrix.tex"),
68         include.rownames = FALSE)
69 }
70
71 create_cv_indexes <- function(N, n_folds){
72     indexes_per_fold <- floor(N/n_folds)
73     index_matrix <- matrix(0L, nrow = n_folds, ncol = indexes_per_fold)
74     index_available <- 1:N
75     for(i in 1:n_folds){
76         selected_indexes <- sample(index_available, indexes_per_fold)
77         index_available <- index_available[! index_available %in% selected_
78             indexes]
79
80         index_matrix[i, ] <- selected_indexes
81     }
82     return(index_matrix)
83 }

```

../R_scripts/Help_Scripts/to_latex_functions.R

6.2 R-Code: Regression Tree

```

1 ## Libraries and seed
2 rm(list = ls())
3 library(caret)
4 library(readr)
5 library(tree)
6 library(randomForest)
7 library(gbm)
8 library(tikzDevice) # library to export plots to .tex files
9
10 options(tikzMetricPackages = c("\\usepackage[utf8]{inputenc}", "\\usepackage
11     [T1]{fontenc}",
12     "\\usetikzlibrary{calc}", "\\usepackage{
13     amssymb}"))
14
15 set.seed(420)
16
17 if(!exists("create_confusion_matrix", mode = "function")){
18     source("Help_Scripts/to_latex_functions.R")
19 }
20
21 #-----#
22 ## Data
23 path_data <- paste0(getwd(), "/data")

```

```

23 path_to_here <- paste0(getwd(), "/Tree_Based_Methods") # getwd give path
    to project
24 # which is one folder over
25
26 train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"),
    header = TRUE)
27 unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv")
    , header = TRUE)
28
29 # Remove unnessesary variables which have a low variance
30 split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
    FALSE)
31 test_data <- train_data[-split_train_test, ]
32 train_data <- train_data[split_train_test, ]
33
34 # Remove variable with low variance which are near zero. Doing it after
35 # splitting in train/test set to avoid contaminating the data.
36 near_zero_variables <- nearZeroVar(train_data[, -1], saveMetrics = T, freqCut
    = 10000/1, uniqueCut = 1/7)
37 cut_variables <- rownames(near_zero_variables[near_zero_variables$nzv ==
    TRUE, ])
38 variables <- setdiff(names(train_data), cut_variables)
39 train_data <- train_data[, variables]
40 test_data <- test_data[, variables]
41
42 train_data[, 1] <- as.factor(train_data[, 1])
43 test_data[, 1] <- as.factor(test_data[, 1])
44
45 #train_data[,1] <- as.factor(train_data[,1])
46
47 unclassified_data[,1] <- as.factor(unclassified_data[,1])
48
49 sum(near_zero_variables$nzv)
50
51 #-----#
52
53 ## REGRESSION - tree
54
55 regression <- function(
56   minimum_development,
57   train_data,
58   test_data
59 ){
60   # Change name of pixel columns to work with tikz library
61
62   print(getClass(class(train_data)))
63   colnames(train_data)[ 2:length(train_data[1,])] <- c(paste0("Pixel", 1:(
64     length(train_data[1,]) - 1)))
65   colnames(test_data)[ 2:length(train_data[1,])] <- c(paste0("Pixel", 1:(
66     length(train_data[1,]) - 1)))
67   minimum_development <- 0.005
68   tree_model <- tree(Digit ~ ., data = train_data, mindev = minimum_
69     development)
70   plot(tree_model)
71   text(tree_model, cex = .5)
72   print(summary(tree_model))
73
74   cross_validation <- cv.tree(tree_model, K = 10)
75   cross_validation$k[1] <- 0

```

```

73 alpha <- round <- round(cross_validation$k)
74
75 plot(cross_validation$size, cross_validation$dev, type = "b",
76       xlab = "Number of terminal nodes", ylab = "CV error")
77
78 ggplot_df <- data.frame(size = cross_validation$size, dev = cross_
79                          validation$dev)
80
81 destination_path <- paste0(path_to_here, "/Results_TBM/Regression_Tree")
82
83 ggplot1 <- ggplot(data = ggplot_df, aes(x = size, y = dev)) +
84   geom_line(aes(colour = "$RegressionTree$"), linetype = "
85             dashed") +
86   geom_point() +
87   geom_vline(xintercept = 20, color = "black", linetype = "
88             dotdash") +
89   xlab("n\\_{terminal nodes}") +
90   ylab("CV Error") +
91   scale_colour_manual("Legend",
92                       breaks = c("$RegressionTree$"),
93                       values = c("#91bafb"),
94                       guide = guide_legend(override.aes = list(
95                         linetype = c("solid"),
96                         shape = c(16)
97                       ))) +
98   theme_bw() +
99   theme(legend.position = c(0.8, 0.355),
100         legend.background = element_rect(fill=alpha('white', 0)))
101 ggsave(paste0(destination_path, ".png"))
102
103 ggplot_to_latex(ggplot1, destination_path, width = 5, height = 5)
104 tree_prune <- prune.tree(tree_model, best = 20)
105 summary(tree_prune)
106
107 tikz(file = paste0(destination_path, "_Tree.tex"), width = 6, height =
108      4)
109 plot(tree_prune)
110 text(tree_prune, cex = .5)
111 dev.off()
112
113 predicted <- predict(tree_prune, test_data, type = "class")
114 create_confusion_matrix(predicted, test_data[,1], destination_path)
115 }
116
117 regression(0.05, train_data, test_data)
118 #-----#
119
120 ## RANDOM FORREST -randomForest
121
122
123 #-----#
124
125 ## BOOSTING - gbm
126
127 #-----#

```

../R_scripts/Tree_Based_Methods/Regression_Tree.R

6.3 R-Code: Random Forest

```
1 ## Libraries and seed
2 rm(list = ls())
3 library(randomForest) # library giving a easy-to-use random forest method
4 library(caret)        # useful library to split up data set
5 library(tikzDevice)   # library to export plots to .tex files
6 library(xtable)       # library to export data frames to tables in .tex
7                         # files
8 set.seed(420)          # seed to replicate results and get consistent test
9                         # and training set
10
11 # Load help script with functions to export the results to latex
12 # These functions gathered to avoid duplicate code
13 if(!exists("create_confusion_matrix", mode = "function")){
14   source("Help_Scripts/to_latex_functions.R")
15 }
16
17 #-----#
18 ## Data
19 path_data <- paste0(getwd(), "/data")
20 path_to_here <- paste0(getwd(), "/Tree_Based_Methods") # getwd give path
21                                                         # which is one
22                                                         # folder over
23
24 train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"),
25   header = TRUE)
26 unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv")
27   , header = TRUE)
28
29 train_data[,1] <- as.factor(train_data[, 1])
30
31 # split training set into training and test set
32
33 split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
34   FALSE)
35 test_data <- train_data[-split_train_test, ]
36 train_data <- train_data[split_train_test, ]
37
38 #-----#
39 ## Random forest
40 # Train forest
41 train_random_forest <- function(
42   data,
43   n_trees,
44   minimum_development = 0.01
45 ){
46   random_forest <- randomForest(Digit ~ .,
47     data = data,
48     ntree = n_trees,
49     #mindev = minimum_development,
50     importance = TRUE,
51     na.action = na.exclude)
52   return(random_forest)
53 }
```

```

51 # Plot error as the number of trees increase
52
53 plot_error_development <- function(
54   random_forest_data,
55   destination_path
56 ){
57   error_data <- data.frame(n_trees = 1:nrow(random_forest_data$err.
58     rate),
59     error <- random_forest_data$err.rate[, "OOB"
60     ])
61
62   write.csv(error_data, file = paste0(destination_path, ".csv"))
63
64   ggplot1 <- ggplot(data = error_data, aes(x = n_trees)) +
65     geom_line(aes(y = error, colour = "$Random forest")) +
66     xlab("$n\\_{trees}$") +
67     ylab("Miss. class. Error") +
68     scale_colour_manual("Legend",
69       breaks = c("$Random forest$"),
70       values = c("black"),
71       guide = guide_legend(override.aes = list(
72         linetype = c("solid"),
73         shape = c( 16)
74       ))) +
75     theme(legend.position = c(0.9, 0.2))
76   ggsave(paste0(destination_path, ".png"))
77
78   ggplot_to_latex(ggplot1, destination_path, width = 6, height = 4)
79 }
80
81 main <- function(){
82   n_trees = 50
83   random_forest <- train_random_forest(train_data, n_trees)
84   plot_error_development(random_forest, paste0(path_to_here, "/Results_TBM
85     /Random_Forest-",
86     n_trees, "trees_Error_plot"
87     ))
88
89   prediction <- predict(random_forest, newdata = test_data)
90   create_confusion_matrix(predicted_value = prediction, true_value = test_
91     data$Digit,
92     paste0(path_to_here, "/Results_TBM/Random_Forest
93     _",
94     n_trees, "
95     trees"))
96 }
97
98 main()

```

../R_scripts/Tree_Based_Methods/Random_Forest.R

6.4 R-Code: Bagging

```
1 ## Libraries and seed
2 rm(list = ls())
3 library(randomForest) # library giving a easy-to-use random forest method
4 library(caret)        # useful library to split up data set
5 library(tikzDevice)   # library to export plots to .tex files
6 library(xtable)       # library to export data frames to tables in .tex
7                         # files
8 set.seed(420)          # seed to replicate results and get consistent test
9                         # and training set
10
11 # Load help script with functions to export the results to latex
12 # These functions gathered to avoid duplicate code
13 if(!exists("create_confusion_matrix", mode = "function")){
14   source("Help_Scripts/to_latex_functions.R")
15 }
16
17 #-----#
18 ## Data
19
20 path_data <- paste0(getwd(), "/data")
21 path_to_here <- paste0(getwd(), "/Tree_Based_Methods") # getwd give path
22 # to project
23 # which is one folder over
24
25 train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"),
26   header = TRUE)
27 unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv")
28   , header = TRUE)
29
30 train_data[,1] <- as.factor(train_data[, 1])
31
32 # split training set into training and test set
33
34 split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
35   FALSE)
36 test_data <- train_data[-split_train_test, ]
37 train_data <- train_data[split_train_test, ]
38
39 #-----#
40 ## Random forest
41 # Train forest
42 train_bagging <- function(
43   data,
44   n_trees,
45   minimum_development = 0.01
46 ){
47   n_features <- ncol(data) - 1
48   bagging <- randomForest(Digit ~ .,
49     data = data,
50     ntree = n_trees,
51     #mindev = minimum_development,
52     mtry = n_features,
53     importance = TRUE,
54     na.action = na.exclude)
55   return(bagging)
```



```

52 }
53
54 # Plot error as the number of trees increase
55
56 plot_error_development <- function(
57   random_forest_data,
58   destination_path
59 ){
60   error_data <- data.frame(n_trees = 1:nrow(random_forest_data$err.rate),
61                             error <- random_forest_data$err.rate[, "OOB"])
62
63   write.csv(error_data, file = paste0(destination_path, ".csv"))
64   ggplot1 <- ggplot(data = error_data, aes(x = n_trees)) +
65     geom_line(aes(y = error, colour = "$Bagging$")) +
66     xlab("n\\_{trees}") +
67     ylab("Miss.class. Error") +
68     scale_colour_manual("Legend",
69                         breaks = c("$Bagging$"),
70                         values = c("black"),
71                         guide = guide_legend(override.aes = list(
72                           linetype = c("solid"),
73                           shape = c(16)
74                         ))) +
75     theme(legend.position = c(0.9, 0.2))
76   ggsave(paste0(destination_path, ".png"))
77
78   ggplot_to_latex(ggplot1, destination_path, width = 6, height = 4)
79 }
80
81 main <- function(){
82   n_trees <- 50
83   bagging <- train_bagging(train_data, n_trees)
84   plot_error_development(bagging, paste0(path_to_here, "/Results_TBM/
85     Bagging-",
86                                     n_trees, "trees_Error_plot"))
87
88   prediction <- predict(bagging, newdata = test_data)
89
90   create_confusion_matrix(predicted_value = prediction, true_value = test_
91     data$Digit,
92     paste0(path_to_here, "/Results_TBM/Bagging-",
93           n_trees, "trees"))
94 }
95
96 main()

```

../R_scripts/Tree_Based_Methods/Bagging.R

6.5 R-Code: Boosting

```
1 ## Libraries and seed
2 rm(list = ls())
3 library(caret)           # useful library to split up data set
4 library(tikzDevice)      # library to export plots to .tex files
5 library(gbm)             # library with powerful boosting method
6 library(xtable)          # library to export data frames to tables in .tex
   files
7 set.seed(420)            # seed to replicate results and get consistent test
   and training set
8
9 # Load help script with functions to export the results to latex
10 # These functions gathered to avoid duplicate code
11 if(!exists("create_confusion_matrix", mode = "function")){
12   source("Help_Scripts/to_latex_functions.R")
13 }
14
15 #-----#
16
17 ## Data
18
19 path_data <- paste0(getwd(), "/data")
20 path_to_here <- paste0(getwd(), "/Tree_Based_Methods") # getwd give path
   to project
21
22 train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"),
   header = TRUE)
23 unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv")
   , header = TRUE)
24
25 train_data[,1] <- as.factor(train_data[, 1])
26
27 # split training set into training and test set
28
29 split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
   FALSE)
30 test_data <- train_data[-split_train_test, ]
31 train_data <- train_data[split_train_test, ]
32
33 #-----#
34
35 ## Boosting
36 # Train booster
37 boosting <- function(
38   data,
39   n_trees,
40   minimum_development = 0.01,
41   interaction_depth = 2,
42   shrinkage = 0.001
43 ){
44   # boosting <- gbm(Digit ~ .,
45   #                 data = data,
46   #                 distribution = "multinomial",
47   #                 n.trees = n_trees,
48   #                 interaction.depth = interaction_depth,
49   #                 shrinkage = shrinkage,
50   #
51   #                 bag.fraction = 1,
```

```

52     #             cv.folds = 10,
53     #             n.cores = 4)
54
55
56     tune_control <- trainControl(method = "cv",
57                                   number = 5,
58                                   repeats = 1)
59     training_grid <- expand.grid(n.trees = c(n_trees),
60                                   interaction.depth = c(interaction_depth),
61                                   shrinkage = c(shrinkage),
62                                   n.minobsinnode = c(10))
63     print(training_grid)
64     boosting <- train(Digit ~ ., data = data, method = "gbm",
65                       trControl = tune_control,
66                       tuneGrid = training_grid)
67     return(boosting)
68 }
69
70
71 # Plot error as the number of trees increase
72
73 plot_error_development <- function(
74   boosting_data,
75   destination_path
76 ){
77   error_data <- data.frame(n_trees = 1:length(boosting_data$cv.error),
78                             error <- boosting_data$cv.error)
79   write.csv(error_data, file = paste0(destination_path, ".csv"))
80
81   ggplot1 <- ggplot(data = error_data, aes(x = n_trees)) +
82     geom_line(aes(y = error, colour = "$Boosting$")) +
83     xlab("$n_{trees}$") +
84     ylab("Miss.class. Error") +
85     scale_colour_manual("Legend",
86                         breaks = c("$Boosting$"),
87                         values = c("black"),
88                         guide = guide_legend(override.aes = list(
89                           linetype = c("solid"),
90                           shape = c(16)
91                         ))) +
92     theme(legend.position = c(0.9, 0.2)) +
93     theme_bw() +
94     theme(legend.position = c(0.8, 0.355),
95           legend.background = element_rect(fill=alpha('white', 0)))
96   ggsave(paste0(destination_path, ".png"))
97
98   ggplot_to_latex(ggplot1, destination_path, width = 6, height = 4)
99 }
100
101 main <- function(){
102   n_trees = 10
103   boosting_train <- boosting(train_data, n_trees)
104   plot_error_development(boosting_train, paste0(path_to_here,
105                                                  "/Results_TBM/Boosting-",
106                                                  n_trees,
107                                                  "trees_Error_plot"))
108   #predicted <- predict(boosting_train, test_data)
109   #create_confusion_matrix(predicted, test_data$Digit, paste0(path_to_here

```

```

110      #                               "/Results_
111      # TBM/Boosting_",               n_trees))
112  }
113
114  main()

../R_scripts/Tree_Based_Methods/Boosting.R

```

6.6 R-Code: Plot Random Forest w/ Bagging

```
1 rm(list = ls())
2 library(ggplot2)
3 library(tikzDevice)      # library to export plots to .tex files
4
5 path_data <- paste0(getwd(), "/data")
6 path_to_here <- paste0(getwd(), "/Tree_Based_Methods") # getwd give path
   to project
7
   # which is one
   folder over
8
9 plot_random_forest_bagging <- function(
10   n_trees,
11   path,
12   destination_path
13 ){
14   rf_path <- paste0(path,
15                     "Random_Forest_",
16                     n_trees,
17                     "trees_Error_plot_",
18                     n_trees,
19                     "trees.csv")
20   bagging_path <- paste0(path,
21                          "Bagging_",
22                          n_trees,
23                          "trees_Error_plot_",
24                          n_trees,
25                          "trees.csv")
26
27   random_forest_error <- read.csv(rf_path)
28   bagging_error <- read.csv(bagging_path)
29
30   ggplot_df <- data.frame(n_trees = 1:nrow(bagging_error),
31                          rf = random_forest_error[3],
32                          bag = bagging_error[3])
33   names(ggplot_df) <- c("n_trees", "rf", "bag")
34   print(str(ggplot_df))
35
36   tikz(file = paste0(destination_path, ".tex"), width = 6, height = 4)
37   ggplot1 <- ggplot(data = ggplot_df, aes(x = n_trees)) +
38     geom_line(aes(y = rf, colour = "$Random forest$")) +
39     geom_line(aes(y = bag, colour = "$Bagging$")) +
40     geom_vline(xintercept = 150, color = "black", linetype = "
       dotdash") +
41     xlab("n\\_{trees}") +
42     ylab("Miss.class. Error") +
43     scale_colour_manual("Legend",
44                         breaks = c("$Random forest$", "$Bagging$"),
45                         values = c("#91b9db", "#fc8d59"),
46                         guide = guide_legend(override.aes = list(
47                           linetype = c("solid", "solid"),
48                           shape = c(16, 16)
49                         ))) +
50     theme_bw() +
51     theme(legend.position = c(0.8, 0.355),
52           legend.background = element_rect(fill=alpha('white', 0)))
53   ggsave(paste0(destination_path, ".png"))
54   print(ggplot1)
```

```

55         dev.off()
56     }
57
58 main <- function(
59
60 ){
61     n_trees = 500
62     path = paste0(path_to_here, "/Results_TBM/")
63     destination_path = paste0(path, "/Random_Forest_Bagging-",
64                               n_trees, "trees")
65
66     plot_random_forest_bagging(n_trees, path, destination_path)
67
68 }
69
70 main()

```

../R_scripts/Tree_Based_Methods/Plot_Random_Forest_Bagging.R

6.7 R-Code: Neural Network

```
1 ## Libraries and seed
2 rm(list = ls())
3 library(h2o)
4 library(caret)
5 library(reshape2)
6
7 set.seed(420)
8
9 # Load help script with functions to export the results to latex
10 # These functions gathered to avoid duplicate code
11 if(!exists("create_confusion_matrix", mode = "function")){
12   source("Help_Scripts/to_latex_functions.R")
13 }
14 #-----#
15
16 ## Data
17
18 path_data <- getwd()
19 path_to_here <- paste0(getwd(), "/Neural_Networks")
20
21 train_data <- read.csv(paste0(path_data, "/data/Train_Digits_20171108.csv"))
22 unclassified_data <- read.csv(paste0(path_data, "/data/Test_Digits_20171108.
   csv"))
23
24 local.h2o <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, max_
   mem_size = "7G", nthreads = -1)
25
26 train_data[,1] <- as.factor(train_data[, 1])
27 split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
   FALSE)
28 test_data <- train_data[-split_train_test, ]
29 train_data <- train_data[split_train_test, ]
30
31 train_data <- as.h2o(train_data)
32 unclassified_data <- as.h2o(unclassified_data)
33 test_data <- as.h2o(test_data)
34
35 #-----#
36
37 ## Getting useful data from grid run of neural networkss
38
39 get_data_in_df <- function(
40   data
41 )
42 {
43   n <- length(data@model_ids)
44   mse_errors <- rep(0,n)
45   mean_per_class_errors <- rep(0,n)
46   hidden <- rep("", n)
47   str(hidden)
48   rate <- rep(0,n)
49   l1 <- rep(0,n)
50   epochs <- rep(0,n)
51   model_numbers <- rep(0,n)
52   train_error <- rep(0,n)
53   train_mse <- rep(0,n)
54   test_error <- rep(0,n)
```

```

55 test_mse <- rep(0,n)
56 activation <- rep("",n)
57 input_dropout_ratio <- rep(0,n)
58 nesterov_accelerated_gradient <- rep("", n)
59
60 model_df <- data.frame(model_numbers = mse_errors, hidden, rate, l1,
61 epochs,
62 train_error, test_error, train_mse, test_mse,
63 activation, input_dropout_ratio,
64 stringsAsFactors = FALSE)
65
66 str(model_df)
67
68 for(i in 1:n){
69   model <- h2o.getModel(data@model_ids[[i]])
70   model_df$mse_errors[i] <- h2o.mse(model)
71   #model_df$mean_per_class_error[i] <- model@model$cross_validation_
72   metrics@metrics$mean_per_class_error
73   model_df$mean_per_class_error[i] <- h2o.performance(model, xval = T)
74   @metrics$mean_per_class_error
75
76   model_paramaters <- model@allparameters
77   model_name <- model@model_id
78   model_number <- sub(".*model_(.*)$", "\\1", model_name)
79   model_df$model_numbers[i] <- as.integer(model_number)
80   model_df$hidden[i] <- paste(as.character(model_paramaters$hidden),
81     sep = " ", collapse = " ")
82   model_df$rate[i] <- model_paramaters$rate
83   model_df$l1[i] <- model_paramaters$l1
84   model_df$epochs[i] <- model_paramaters$epochs
85   model_df$activation[i] <- model_paramaters$activation
86   model_df$input_dropout_ratio[i] <- model_paramaters$input_dropout_
87   ratio
88   model_df$nesterov_accelerated_gradient[i] <- model_paramaters$
89   nesterov_accelerated_gradient
90
91   #print(model)
92   train_performance <- h2o.performance(model, train_data)@metrics
93   train_performance_error <- train_performance$mean_per_class_error
94   train_performance_mse <- train_performance$MSE
95
96   model_df$train_error[i] <- train_performance_error
97   model_df$train_mse[i] <- train_performance_mse
98
99   test_performance <- h2o.performance(model, test_data)@metrics
100   test_predictions <- h2o.predict(model, test_data)
101   test_accuracy <- test_predictions$predict == test_data$Digit
102   test_performance_error <- 1 - mean(test_accuracy)
103   #test_performance_error <- test_performance$mean_per_class_error
104   test_performance_mse <- test_performance$MSE
105
106   model_df$test_error[i] <- test_performance_error
107   model_df$test_mse[i] <- test_performance_mse
108
109 }
110 model_df <- model_df[with(model_df, order(model_numbers)),]
111 model_df
112 }
113
114 activation <- list("Rectifier", "RectifierWithDropOut")# "Tanh")

```



```

106 hidden <- list(c(100,100), c(150, 150), c(540, 320), c(100, 100, 100), c
      (540, 320, 100))
107 input_dropout_ratio <- list(0, 0.2)
108 nesterov_accelerated_gradient <- list( TRUE)
109 epochs <- list(20)
110 l1 = list(0,1.4e-5)
111 hyper_params <- list(activation = activation, hidden = hidden, input_dropout
      _ratio = input_dropout_ratio, nesterov_accelerated_gradient = nesterov_
      accelerated_gradient, epochs = epochs, l1 = l1)
112
113 grid_deep_learning <- h2o.grid(algorithm = "deeplearning",
114                               x = 2:785,
115                               y = 1,
116                               training_frame = train_data,
117                               nfolds = 10,
118                               stopping_metric = "MSE",
119                               stopping_tolerance = 0.0025,
120                               hyper_params = hyper_params)
121 save_results <- function(results){
122   write.csv(results, file = paste0(path_to_here, "/Neural_Networks/results
      _NN/grid_run_20.csv"))
123 }
124
125 df <- get_data_in_df(grid_deep_learning)
126 save_results(df)
127
128 #results_df <- df
129
130 results_df <- read.csv(paste0(path_to_here, "/Neural_Networks/results_NN/
      grid_run_40.csv"))
131
132 results_df <- results_df[with(results_df, order(mean_per_class_error)),]
133 results_df$row_names <- 1:length(results_df[,1])
134
135 melt_datas <- melt(results_df[c("test_error","mean_per_class_error", "row_
      names",
136                               "model_numbers")], id = c("row_names", "
      model_numbers"))
137
138 plot_list <- list()
139 # Plot classification error
140 plot_list[[1]] <- ggplot(data=melt_datas,
141                          aes(x=row_names, y=value)) +
142   geom_point(aes(colour = as.factor(model_numbers), group = as.factor(
      model_numbers)), size = 1.25) +
143   geom_line(aes(group = variable)) +
144   xlab("Model id") +
145   ylab("Miss. class. Error") +
146   scale_y_continuous(limits = c(0, 0.2)) +
147   theme_bw() +
148   theme(legend.position = c(0.675, 0.255),
149         legend.background = element_rect(fill=alpha('white', 0)),
150         legend.direction = "horizontal",
151         legend.text = element_text(size=6),
152         legend.key = element_rect(size = 3),
153         legend.key.size = unit(1.0, 'lines'),
154         axis.text.x=element_blank(),
155         axis.ticks.x=element_blank()) +
156   scale_colour_discrete(name = "Model ids") +
157   guides(fill = guide_legend(title = "Legend"))

```

```

157
158 ggplot_to_latex(plot_list[[1]],
159                 paste0(path_to_here, "/results_NN/per_class_error"), width =
160                     6, height = 4)
161 ggsave(paste0(path_to_here, "/results_NN/per_class_error3.png"))
162 #
163     deep_learning_predicting <- h2o.predict(
164         object = deep_learning_results, newdata = test_data)
165 #deep_learning_performance <- h2o.performance(model = deep_learning_results3
166     , newdata = test_data)
167 #deep_learning_performance
168 #deep_learning_predicting_data_frame <- as.data.frame((deep_learning_
169     predicting))
170
171 #deep_learning_results2 <- h2o.deeplearning(x = 2:785,
172 #
173     y = 1,
174 #
175     training_frame = train_data,
176 #
177     activation = "Tanh",
178 #
179     hidden = c(160, 160, 160, 160,
180     160),
181 #
182     nfold = 10,
183 #
184     keep_cross_validation_
185     predictions = TRUE,
186 #
187     epochs = 40)
188
189 deep_learning_results3<- h2o.deeplearning(x = 2:785,
190     y = 1,
191     training_frame = train_data,
192     #activation = "RectifierWithDropout
193     ",
194     activation = "Rectifier",
195     input_dropout_ratio = 0.2,
196     #hidden_dropout_ratios = c(0.2,
197     0.2, 0.2),
198     nfold = 10,
199     balance_classes = TRUE,
200     hidden = c(540, 320),
201     l1 = 1.4e-5,
202     #momentum_stable = 0.99,
203     stopping_metric = "MSE",
204     stopping_tolerance = 0.0025,
205     nesterov_accelerated_gradient =
206     TRUE,
207     epochs = 20)
208
209 h2o.performance(deep_learning_results3, test_data)
210
211 predicted <- predict(deep_learning_results3, test_data, type = "response")
212
213 predicted_confusion_matrix <- as.factor(as.vector(predicted$predict))
214 test_data_confusion_matrix <- as.data.frame(test_data)
215 create_confusion_matrix(predicted_confusion_matrix,
216     test_data_confusion_matrix[, "Digit"],
217     paste0(path_to_here, "/results_NN/540_320_neural_net
218     "))
219

```

```
205 | h2o.shutdown()
```

```
../R_scripts/Neural_Networks/uneural_network.R
```

6.8 R-Code: Convolutional Neural Network

```
1 ## Libraries and seed
2 rm(list = ls())
3 library(mxnet)
4 library(caret)
5 set.seed(420)
6
7 #-----#
8
9 ## Data
10
11 path_to_here <- getwd()
12
13 train_data <- read.csv(paste0(path_to_here, "/data/Train_Digits_20171108.csv"), header = TRUE)
14 unclassified_data <- read.csv(paste0(path_to_here, "/data/Test_Digits_20171108.csv"), header = TRUE)
15
16 train_data[,1] <- as.factor(train_data[, 1])
17
18 # split training set into training and test set
19
20 split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list = FALSE)
21 test_data <- train_data[-split_train_test, ]
22 train_data <- train_data[split_train_test, ]
23
24 # convert to matrix, required by "mxnet"
25
26 train <- data.matrix(train_data)
27 test <- data.matrix(test_data)
28
29 train_x <- t(train[, -1])/255)
30 train_y <- train[, 1]
31
32 train_array <- train_x
33 dim(train_array) <- c(28, 28, 1, ncol(train_x))
34
35 #train_x <- t(train_x)/255)
36
37 test_x <- test[, -1]
38 test_y <- test[, 1]
39
40 test_x <- t(test_x/255)
41
42 # transpose and normalize to more
43
44 #-----#
45
46 ## Setting up Convolutional Neural Network(CNN)
47
48 data <- mx.symbol.Variable("data")
49 fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
50 act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")
51 fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64)
52 act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu")
53 fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=10)
54 softmax <- mx.symbol.SoftmaxOutput(fc3, name="sm")
```

```

55 |
56 | devices <- mx.cpu()
57 |
58 | mx.set.seed(0)
59 |
60 | model <- mx.model.FeedForward.create(softmax, X=train_x, y=train_y,
61 |                                     ctx=devices, num.round=10, array.batch.
62 |                                     size=100,
63 |                                     learning.rate=0.07, momentum=0.9, eval
64 |                                     .metric=mx.metric.accuracy,
65 |                                     initializer=mx.init.uniform(0.07),
66 |                                     epoch.end.callback=mx.callback.log.
67 |                                     train.metric(100))
68 |
69 | preds <- predict(model, test_x)
70 |
71 | ..R_scripts/Neural_Networks/convolutional_neural_network.R

```

6.9 R-Code: K-nearest Neighbours

```
1 ## Libraries and seed
2 rm(list = ls())
3
4 library(foreach)          # library for running loop in parallel
5 library(doParallel)       # library for running loop in parallel
6 library(caret)            # useful library to split up data set
7 library(tikzDevice)       # library to export plots to .tex files
8 library(xtable)           # library to export data frames to tables in .tex
9                             files
10 library(kknn)
11 library(class)
12
13 set.seed(420)              # seed to replicate results and get consistent test
14                             and training set
15
16 # Load help script with functions to export the results to latex
17 # These functions gathered to avoid duplicate code
18 if(!exists("create_confusion_matrix", mode = "function")){
19   source("Help_Scripts/to_latex_functions.R")
20 }
21
22 #-----#
23
24 ## Data
25 path_data <- paste0(getwd(), "/data")
26 path_to_here <- paste0(getwd(), "/Tree_Based_Methods") # getwd give path
27                             to project
28 # which is one folder over
29
30 train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"),
31                       header = TRUE)
32 unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv")
33                             , header = TRUE)
34
35 train_data[,1] <- as.factor(train_data[, 1] )
36
37 # split training set into training and test set
38
39 split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
40   FALSE)
41 test_data <- train_data[-split_train_test, ]
42 train_data <- train_data[split_train_test, ]
43
44
45 k_nearest_neighbors <- function(
46   train_data,
47   train_data_norm,
48   k_folds,
49   k_neighbors = 20
50 ){
51   # First find number of k(neighbors) using crossvalidation
52   cv_indexes <- create_cv_indexes(nrow(train_data), k_folds)
53
54   cores=detectCores()
55   cl <- makeCluster(cores[1]-1) #not to overload your computer
56   registerDoParallel(cl)
```

```

52     cv_error <- c()
53     indexes <- 1:nrow(train_data)
54
55     final_df <- foreach(i = 1:k_neighbors,
56                         .combine = "rbind",
57                         .packages = "class") %dopar%{
58         error_kfolds <- 0
59         error_kfolds_norm <- 0
60         for(k in 1:k_folds){
61             train_val <- train_data[!indexes %in% cv_indexes[k, ], ]
62             validation_val <- train_data[cv_indexes[k, ], ]
63
64             train_val_norm <- train_data_norm[!indexes %in% cv_indexes[k
65             , ], ]
66             validation_val_norm <- train_data_norm[cv_indexes[k, ], ]
67
68             # without normalization
69             knn_pred_class <- knn(train_val[-1],
70                                 validation_val[-1],
71                                 train_val[, 1],
72                                 k = i)
73
74             # with normalization
75             knn_pred_class_norm <- knn(train_val_norm[-1],
76                                       validation_val_norm[-1],
77                                       train_val_norm[, 1],
78                                       k = i)
79
80             error <- 1 - mean(validation_val[, 1] == knn_pred_class)
81             error_norm <- 1 - mean(validation_val_norm[, 1] == knn_pred_
82             class_norm)
83             error_kfolds <- error_kfolds + error
84             error_kfolds_norm <- error_kfolds_norm + error_norm
85         }
86         temp_df <- data.frame(k_neighbors = i,
87                             error_kfolds = error_kfolds,
88                             error_kfolds_norm = error_kfolds_norm)
89     }
90     final_df[, 2:3] <- final_df[, 2:3] / k_folds
91     stopCluster(cl)
92     return(final_df)
93 }
94
95 average_cv_error <- function(
96     train_data,
97     test_data,
98     train_data_norm,
99     test_data_norm,
100     n_avg
101 ){
102     k_neighbors <- 20
103
104     # set up error data frame
105     error_df <- data.frame(k_neighbors = 1:k_neighbors,
106                           error_kfolds = rep(0, k_neighbors),
107                           error_kfolds_norm = rep(0, k_neighbors))
108
109     # progress bar
110     pb <- txtProgressBar(min = 0, max = n_avg, style = 3)

```

```

109
110 # average cv-error over several runs to get more precise measure
111 for(i in 1:n_avg){
112     setTxtProgressBar(pb, i)
113     error_df[-1] <- error_df[-1] + k_nearest_neighbors(train_data,
114                                                         train_data_norm,
115                                                         2,
116                                                         k_neighbors)[-1]
117 }
118 close(pb)
119 error_df[-1] <- error_df[-1]/n_avg
120 ggplot1 <- ggplot(data = error_df, aes(x = k_neighbors)) +
121   geom_line(aes(y = error_kfolds, colour = "$KNN$")) +
122   geom_line(aes(y = error_kfolds_norm, colour = "$KNN normalized$")) +
123   geom_vline(xintercept = 1, color = "black", linetype = "dotted") +
124   xlab("$k$ neighbors") +
125   ylab("Miss.class. Error") +
126   scale_colour_manual("Legend",
127                       breaks = c("$KNN$", "$KNN normalized$"),
128                       values = c("#91b9db", "#fc8d59"),
129                       guide = guide_legend(override.aes = list(
130                         linetype = c("solid", "solid"),
131                         shape = c(16, 16)
132                       ))) +
133   theme_bw() +
134   theme(legend.position = c(0.8, 0.355),
135         legend.background = element_rect(fill=alpha('white', 0)))
136 ggplot_to_latex(ggplot1,
137                 paste0("K_Nearest_Neighbors/Results_KNN/knn_error_
138                       compare",n_avg),
139                 width = 5, height = 5)
140 return(error_df)
141 }
142
143 predict_on_test <- function(
144   best_k,
145   train_data,
146   test_data
147 ){
148   knn_pred_class <- knn(train_data[-1],
149                         test_data[-1],
150                         train_data[, 1],
151                         k = best_k)
152   create_confusion_matrix(predicted_value = knn_pred_class,
153                           true_value = test_data[, 1],
154                           destination_path = "K_Nearest_Neighbors/Results_
155                           KNN/")
156 }
157 #cv_final <- k_nearest_neighbors(train_data, 10)
158 train_data_norm <- train_data
159 train_data_norm[-1] <- train_data_norm[-1]/255
160 test_data_norm <- test_data
161 test_data_norm[-1] <- test_data_norm[-1]/255
162 #cv_norm <- k_nearest_neighbors(train_data, train_data_norm, 10)
163
164 # average_error <- average_cv_error(train_data = train_data,
165 #                                     test_data = test_data,
166 #                                     train_data_norm = train_data_norm,
167 #                                     test_data_norm = test_data_norm,

```



```
166 | #                               n_avg = 30)
167 |
168 | predict_on_test(1, train_data_norm, test_data_norm)
169 |
170 | #knn <- kknn(Digit ~ ., train = train_val, test = validation_val[-1], k = i)
171 | #knn_prediction <- fitted.values(knn)
    |
    | ..../R_scripts/K_Nearest_Neighbors/kknn.R
```

6.10 R-Code: Support Vector Machines

```
1 ## Libraries and seed
2 rm(list = ls())
3 library(e1071)           # library to make margins for svm
4 library(caret)           # useful library to split up data set
5 library(readr)
6
7 set.seed(420)            # seed to replicate results and get consistent test
                        and training set
8
9 # Load help script with functions to export the results to latex
10 # These functions gathered to avoid duplicate code
11 if(!exists("create_confusion_matrix", mode = "function")){
12   source("Help_Scripts/to_latex_functions.R")
13 }
14
15 #-----#
16
17 ## Data
18 path_data <- paste0(getwd(), "/data")
19 path_to_here <- paste0(getwd(), "/Support_Vector_Machines") # getwd give
                        path to project
20 # which is one folder over
21
22 train_data <- read.csv(paste0(path_data, "/Train_Digits_20171108.csv"),
                        header = TRUE)
23 unclassified_data <- read.csv(paste0(path_data, "/Test_Digits_20171108.csv")
                        , header = TRUE)
24
25 train_data[,1] <- as.factor(train_data[, 1])
26
27 nzs <- nearZeroVar(train_data[, -1], saveMetrics = TRUE, freqCut = 10000/1,
                        uniqueCut = 1/7)
28 sum(nzs$zeroVar)
29
30 sum(nzs$nzv)
31
32 cut_variables <- rownames(nzs[nzs$nzv == TRUE, ])
33 variable <- setdiff(names(train_data), cut_variables)
34 train_data <- train_data[, variable]
35
36 # split data into test and training set
37 split_train_test <- createDataPartition(train_data$Digit, p = 0.8, list =
                        FALSE)
38 test_data <- train_data[-split_train_test, ]
39 train_data <- train_data[split_train_test, ]
40
41 # remove label temporarily by storeing as it's own vector
42 digit <- train_data[1]
43 train_data$Digit <- NULL
44 train_data <- train_data/255
45 cov_train <- cov(train_data)
46
47 digit_test <- test_data[1]
48 test_data$Digit <- NULL
49 test_data <- test_data/255
50
51 # use prcomp to do PCA on the covariance matrix
```

```

52 train_pc <- prcomp(cov_train)
53
54 # get information of about the variance
55 var_explained <- train_pc$sdev^2/sum(train_pc$sdev^2)
56 var_explained_cumsum <- cumsum(var_explained)
57 var_explained_df <- data.frame(number = 1:length(train_pc$sdev),
58                               variance_explained = var_explained,
59                               cumsum = var_explained_cumsum)
60 # see plot in end of script
61
62
63 # find optimal number of components
64 find_number_of_components <- function(
65   components_range,
66   increase_by
67 ){
68   components <- seq(from = components_range[1],
69                     to = components_range[2],
70                     by = increase_by)
71   missclassification_error <- rep(0, length(components))
72   cv_error <- rep(0, length(components))
73
74   components_error_df <- data.frame(components,
75                                     missclassification_error,
76                                     cv_error)
77
78   for(i in 1:length(components)){
79
80     # traing the svm for the components given
81     train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:
82       components[i]]
83     train_data_temp <- cbind(Digit = digit, as.data.frame(train_score))
84
85     tune_svm <- tune.svm(Digit ~ ., data = train_data_temp, cost = 1:10,
86       kernal = "radial")
87     fit_svm <- tune_svm$best.model
88
89     # review the performance
90
91     predicted <- predict(fit_svm, train_score, type = "response")
92     missclassification_error <- 1 - mean(train_data_temp[, "Digit"] ==
93       predicted)
94
95     components_error_df[i, "cv_error"] <- tune_svm$best.performance
96     components_error_df[i, "missclassification_error"] <-
97       missclassification_error
98   }
99   return(components_error_df)
100 }
101
102 # make ggplot of cv error for different amount of components
103 # also marks selected number of components manually (22 in this task)
104 plot_components_error <- function(
105   components_error_df
106 ){
107   components_chosen <- components_error_df[22, ]
108
109   ggplot1 <- ggplot(data = components_error_df, aes(x = components)) +
110     geom_line(aes(y = cv_error, colour = "$E_{CV}$")) +

```

```

107     geom_point(aes(y = cv_error, colour = "$E_{CV}$")) +
108     geom_vline(xintercept = 22, color = "black", linetype = "dotdash") +
109     xlab("$n_{components}$") +
110     ylab("Miss.class. Error") +
111     geom_point(data = components_chosen,
112               aes(x = components, y = cv_error),
113               color = "red") +
114     geom_text(data = components_chosen,
115             aes(y = cv_error,
116               label = paste0("$\\mathrm{CV}_{error}(22) = ",
117                             round(cv_error,4), "$")),
118             hjust = 0.2, vjust = -1.0) +
119     scale_colour_manual("Legend",
120                       breaks = c("$E_{CV}$"),
121                       values = c("black"),
122                       guide = guide_legend(override.aes = list(
123                         linetype = c("solid"),
124                         shape = c( 16)
125                       ))) +
126     theme_bw() +
127     theme(legend.position = c(0.8, 0.255),
128           legend.background = element_rect(fill=alpha('white', 0)))
129   ggplot_to_latex(ggplot1,
130                 paste0(path_to_here,
131                       "/Results_SVM/number_of_components_cv_error"),
132                 width = 5, height = 5)
133 }
134
135 # Calculate best cost parameter for svm
136 find_optimal_cost_for_components <- function(
137   selected_components,
138   cost_range,
139   cost_increase,
140   n_avg
141 ){
142   train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:selected_
143     components]
144   train_data_temp <- cbind(Digit = digit, as.data.frame(train_score))
145
146   cost <- seq(from = cost_range[1], to = cost_range[2], by = cost_increase
147     )
148
149   cost_error <- data.frame(cost = cost, cv_error = rep(0, length(cost)))
150   for(i in 1:n_avg){
151     tune_svm <- tune.svm(Digit ~ ., data = train_data_temp, cost = cost,
152                       kernal = "radial")
153
154     cv_error <- tune_svm$performances$error
155     cost_error[, "cv_error"] <- cost_error[, "cv_error"] + cv_error
156   }
157   cost_error[, "cv_error"] <- cost_error[, "cv_error"]/n_avg
158
159   best_cost <- cost_error[which(cost_error[, "cv_error"] == min(cost_error
160     [, "cv_error"])),]
161
162   ggplot1 <- ggplot(data = cost_error, aes(x = cost)) +
163     geom_line(aes(y = cv_error, colour = "$E_{CV}$")) +
164     geom_point(aes(y = cv_error, colour = "$E_{CV}$")) +

```

```

162     geom_vline(xintercept = best_cost[, "cost"],
163               color = "black", linetype = "dotted") +
164     xlab("$n_{components}$") +
165     ylab("Miss.class. Error") +
166     geom_point(data = best_cost,
167               aes(x = cost, y = cv_error),
168               color = "red") +
169     geom_text(data = best_cost,
170              aes(y = cv_error,
171                 label = paste0("$\\mathrm{CV}_{error}(",
172                                cost,
173                                ") = ",
174                                round(cv_error,4), "$")),
175              hjust = -0.05, vjust = 0.9) +
176     scale_colour_manual("Legend",
177                         breaks = c("$E_{CV}$"),
178                         values = c("black"),
179                         guide = guide_legend(override.aes = list(
180                           linetype = c("solid"),
181                           shape = c( 16)
182                         ))) +
183     theme_bw() +
184     theme(legend.position = c(0.8, 0.255),
185           legend.background = element_rect(fill=alpha('white', 0)))
186   ggplot_to_latex(ggplot1,
187                 paste0(path_to_here,
188                       "/Results_SVM/cost_cv_error"),
189                 width = 5, height = 5)
190   return(best_cost)
191 }
192
193 # Make prediction on the test set and make confusion matrix
194 predict_on_test <- function(
195   selected_components,
196   cost
197 ){
198   train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:selected_
199     components]
200   train_data_temp <- cbind(Digit = digit, as.data.frame(train_score))
201   test_score <- as.matrix(test_data) %*% train_pc$rotation[,1:selected_
202     components]
203   test_data_temp <- cbind(Digit = digit_test, as.data.frame(test_score))
204   svm_predict <- svm(Digit ~ ., data = train_data_temp, cost = cost,
205                     kernal = "radial")
206   predicted <- predict(svm_predict, test_data_temp)
207
208   create_confusion_matrix(predicted_value = predicted,
209                           true_value = test_data_temp[, "Digit"],
210                           destination_path = paste0(path_to_here,
211                                                       "/Results_SVM/
212                                                       confusion_matrix")
213 )
214 }
215 # some help-plots for pca

```

```

216 plot_variance_explained <- function(
217   var_explained_df,
218   chosen_number_components
219 ){
220   # plot amount of variables and how much of the variance this describe
221   ggplot1 <- ggplot(data = var_explained_df[1:100,], aes(x = number, y =
      cumsum)) +
222     geom_line() +
223     geom_point() +
224     geom_vline(xintercept = chosen_number_components,
225               color = "black", linetype = "dotted") +
226     xlab("$n_{components}$") +
227     ylab("Variance Explained") +
228     theme_bw() +
229     theme(legend.position = c(0.8, 0.255),
230           legend.background = element_rect(fill=alpha('white', 0)))
231
232   # plot the data as described by the two first principle components
233
234   train_score <- as.matrix(train_data) %*% train_pc$rotation[,1:chosen_
      number_components]
235   train_data_temp <- cbind(Digit = digit, as.data.frame(train_score))
236
237   ggplot2 <- ggplot(data = train_data_temp, aes(x = PC1, y = PC2)) +
238     geom_point(aes(colour = Digit)) +
239     xlab("PCA 1") +
240     ylab("PCA 2") +
241     scale_colour_manual("Legend",
242                         values = c('#8dd3c7', '#ffffb3', '#bebada', '#
      fb8072',
243                                   '#80b1d3', '#fdb462', '#b3de69', '#
      fccde5',
244                                   '#d9d9d9', '#bc80bd')
245     ) +
246     theme_bw() +
247     theme(legend.position = c(0.9545, 0.355),
248           legend.background = element_rect(fill=alpha('white', 0)))
249
250   ggplot_to_latex(ggplot1,
251                   paste0(path_to_here,
252                           "/Results_SVM/variance_explained_pca"),
253                   width = 5, height = 5)
254
255   ggplot_to_latex(ggplot2,
256                   paste0(path_to_here,
257                           "/Results_SVM/map_pca1_pca2"),
258                   width = 5, height = 5)
259 }
260
261 main <- function(){
262   # Find best number of components to use with svm
263   components_range <- c(1, 30)
264   increase_by <- 1
265   components_error_df <- find_number_of_components(components_range =
      components_range,
266                                                     increase_by = increase_
      by)
267
268   # Plot best cv error over number of components
269   plot_components_error(components_error_df)

```

```

269
270
271     selected_components <- 22
272
273     # Find optimal cost parameter for the svm function
274     best_cost <- find_optimal_cost_for_components(cost_range = c(0.5, 10),
275                                                    cost_increase = 0.5,
276                                                    selected_components =
277                                                         selected_components,
278                                                         n_avgs = 5)
279
280     # Plot pca variance and to principle components
281     plot_variance_explained(var_explained_df, selected_components)
282 }
283 main()

```

../R_scripts/Support_Vector_Machines/support_with_pca.R