Capstone Report - Product Classification by Product Name

Kai Notté

25.11.2019

Introduction

This capstone project is about the classification of products based on their product name. As a product manager of a distributor for electronic components one of my recurring tasks is to classify products. Beside of mercantile classifications like stocking or pricing, it is the classification for the product category in our web shop. This can be a highly time-consuming tasks, depending on the product information provided by the manufacturer. Indeed, according to some internal tests, our accuracy is only about 80% by manual work, which will take about 1h to classify 1,000 products.

The goal of this project is to develop a machine learning algorithm to somehow automatic this process by using giving product descriptions or names and improve the achieved accuracy and reduce time consumption.

As a project data set I choose the LEGO - Database data set from Kaggle, provided by Rachael Tatman. This data set was originally compiled by Rebrickable.Com, which is a website to help to identify original LEGO sets that can be built with already owned bricks.

This data set includes several files which will be analysed in the following sections. The file *parts.csv* includes a product ID, the product name and the corresponding category. This file will be the core of this project. Additional files will be introduced in the section **Data Analysis**

This Report is separated into three sections: The next section **Data Analysis** will load, analyse and prepare the data set. Afterwards, the section **Results** presents the final model which will be used in the script. The outcome as well as its limitation and future work will be summarized in the **Conclusion** section.

Data Analysis

This section is about the data preparation, the analysis and the development of the final model. It is seperated into four subsections: The first subsection introduces all used libraries and load them into the workspace. If necessary, these packages will be installed automatically from the *CRAN* repository. Thereafter all required data sets from the *LEGO Database* data set will be downloaded and prepared for the data analysis. In the third subsection, the data set will be split into a train and a validation data set. Finally, in the fourth subsection, the train data set will be analysed and evaluate for the final model.

Load Libraries

This subsection briefly introduces packages, used in this project. Furthermore, if the package is not loaded yet, it will be installed from the *CRAN* repository.

caret

The *caret* package was already introduced in the course. It is designed to support the creation of predictive models.

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

tidytext

The *tidyverse* package is a supportive package to *tidyverse*. It includes several functions to transform text into tidy data set.

```r
if(!require(tidytext)) install.packages("tidytext", repos = "http://cran.us.r-project.org")
```

tidyverse

The *tidyverse* package contains several packages useful for data science. Throughout the previous course these packages were used.

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

tm

The *tm* package is a collection of useful function for text mining application.

```r
if(!require(tm)) install.packages("tm", repos = "http://cran.us.r-project.org")
```

Load Data sets

The data set was originally downloaded as the data set LEGO - Database from Kaggle. Unfortunately, Kaggle does not allow an automatic download. Therefore I copied all required data into my GitHub repository. If the data files are not already provided in the project subfolder *data*, it will be downloaded into these from GitHub.

The original data set includes 9 csv-files:

File	Content
color.csv	Information about LEGO colors, including a unique color ID, the corresponding name and its approximate RGB value.
inventories.csv	Information on inventories. This file is not used in the project.
inventory__parts.csv	Information on parts of an inventory. This file is not used in the project.
inventory__sets.csv	Relation of inventories to each set. This file is not used in the project.
part__categories.csv	Information on all part categories, including a unique ID and its name.
parts.csv	Information on every part of the data set, including a unique ID, the corresponding name of the part and its related category ID.
sets.csv	Information on the LEGO sets. This file is not used in the project.
themes.csv	Information on LEGO themes. This file is not used in the project.

In this model, three out of these 9 files will be used: *parts.csv*, *part_categories.csv* and *colors.csv*.

parts.csv

The file *parts.csv* contains the part id (`part_num`), the name (`name`) and the related category ID (`part_cat_id`) for each part in this data set.

```r
filePath_parts <- "./data/parts.csv"

if(!file.exists(filePath_parts)){
  download.file("https://github.com/kai-notte/edx-PH125.9x-productclassification/raw/master/data/parts.c
  destfile="./data/parts.csv",
  method="auto")
}
parts <- read_csv(filePath_parts, col_names = TRUE)
head(parts)
```

```
## # A tibble: 6 x 3
##   part_num name                                                 part_cat_id
##   <chr>    <chr>                                                      <dbl>
## 1 0687b1   Set 0687 Activity Booklet 1                                   17
## 2 0901     Baseplate 16 x 30 with Set 080 Yellow House Print              1
## 3 0902     Baseplate 16 x 24 with Set 080 Small White House Pr~           1
## 4 0903     Baseplate 16 x 24 with Set 080 Red House Print                 1
## 5 0904     Baseplate 16 x 24 with Set 080 Large White House Pr~           1
## 6 1        Homemaker Bookcase 2 x 4 x 4                                   7
```

```r
parts %>% unique() %>% count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1 25993
```

As shown above, it contains 25993 different part numbers, each linked to a descriptive name and a category id.

part_categories.csv

The file *part_categories.csv* describes the product categories by adding the name to an unique ID.

```r
filePath_categories <- "./data/part_categories.csv"
if(!file.exists(filePath_parts)){
  download.file("https://github.com/kai-notte/edx-PH125.9x-productclassification/raw/master/data/part_ca
  destfile="./data/part_categories.csv",
  method="auto")
}
categories <- read_csv(filePath_categories, col_names = TRUE)
categories
```

```
## # A tibble: 57 x 2
##       id name
##    <dbl> <chr>
##  1     1 Baseplates
##  2     2 Bricks Printed
##  3     3 Bricks Sloped
##  4     4 Duplo, Quatro and Primo
##  5     5 Bricks Special
##  6     6 Bricks Wedged
##  7     7 Containers
##  8     8 Technic Bricks
##  9     9 Plates Special
## 10    10 Tiles Printed
## # ... with 47 more rows
```

The data set contains 57 different categories represented in this file. To make the data set more readable, the categorie names will be add to `parts`.

```
parts <- inner_join(parts, categories, by = c("part_cat_id" = "id")) %>% select(part_num, name.x, name.
rm(categories)
```

colors.csv

The last file *colors.csv* contains all colors used for LEGO bricks. This information will be used to improve the data set by removing unimportant color information from the product name.

```
filePath_colors <- "./data/colors.csv"
if(!file.exists(filePath_parts)){
  download.file("https://github.com/kai-notte/edx-PH125.9x-productclassification/raw/master/data/colors
  destfile="./data/colors.csv",
  method="auto")
}
# Adjust to lower cases to normalize the spelling
colors <- read_csv(filePath_colors, col_names = TRUE) %>% mutate(lcName = tolower(name))
```

Separate train and validation data set

The original data set 'parts' will be split into the data sets `train` and `validation`. The `train` data set will be used to analyse the data set as well as to train and evaluate the model in progress. The evaluation of the final model will be done by using the `validation` data set.

```
# Validation data set will be 10% of original data set
set.seed(1, sample.kind="Rounding")
index <- createDataPartition(y = parts$cname, times = 1, p = 0.1, list = FALSE)
temp1 <- parts[-index,]
temp2 <- parts[index,]

# Make sure every category name `cname` is in the train data set

validation <- temp2 %>% semi_join(temp1, by = "cname")
```

```
# Add rows removed from validation set back into train set

removed <- anti_join(temp2, validation)
train <- rbind(temp1, removed)

# Remove unused dfs
rm(removed, temp1, temp2, index, parts)
```

Data Analysis

To start with the development of the model, some general information about the `train` data set:

```
dim(train)
```

```
## [1] 23371     3
```

```
head(train)
```

```
## # A tibble: 6 x 3
##   pid    pname                                            cname
##   <chr>  <chr>                                            <chr>
## 1 0687b1 Set 0687 Activity Booklet 1                      Non-LEGO
## 2 0901   Baseplate 16 x 30 with Set 080 Yellow House Print      Baseplates
## 3 0902   Baseplate 16 x 24 with Set 080 Small White House Print Baseplates
## 4 0903   Baseplate 16 x 24 with Set 080 Red House Print         Baseplates
## 5 1      Homemaker Bookcase 2 x 4 x 4                     Containers
## 6 10     Baseplate 24 x 32                                Baseplates
```

As you can see, the train data set includes 23371 observations with 3 different variable: `pid` as the product ID, `pname` as the product name and `cname` as the category name.

The value `pid` is unique:

```
# Number of rows:
train %>% nrow()
```

```
## [1] 23371
```

```
# Number of unique pids:
train$pid %>% unique() %>% length()
```

```
## [1] 23371
```

`pid` will be used as primary key in the data set, but not analysed regarding its usage for the final algorithm.

As mentioned earlier, there are 57 different categories in the overall data set. The value `cname` should contain all of them:

```
# Number of categories:
train$cname %>% unique() %>% length()
```

```
## [1] 57
```

```r
# List of categories:
train$cname %>% unique()
```

```
##  [1] "Non-LEGO"
##  [2] "Baseplates"
##  [3] "Containers"
##  [4] "Minifigs"
##  [5] "Minifig Accessories"
##  [6] "Duplo, Quatro and Primo"
##  [7] "Non-Buildable Figures (Duplo, Fabuland, etc)"
##  [8] "Other"
##  [9] "Rock"
## [10] "Technic Connectors"
## [11] "Plates Special"
## [12] "Tiles"
## [13] "Windscreens and Fuselage"
## [14] "Bricks Curved"
## [15] "Bars, Ladders and Fences"
## [16] "Plants and Animals"
## [17] "Flags, Signs, Plastics and Cloth"
## [18] "Technic Steering, Suspension and Engine"
## [19] "Power Functions, Mindstorms and Electric"
## [20] "Technic Gears"
## [21] "Wheels and Tyres"
## [22] "Transportation - Land"
## [23] "Tools"
## [24] "Bricks Printed"
## [25] "Bionicle, Hero Factory and Constraction"
## [26] "Tiles Special"
## [27] "Tiles Printed"
## [28] "Bricks Special"
## [29] "Plates"
## [30] "Plates Round and Dishes"
## [31] "Windows and Doors"
## [32] "Bricks Wedged"
## [33] "Technic Beams"
## [34] "Technic Panels"
## [35] "Technic Special"
## [36] "Transportation - Sea and Air"
## [37] "Bricks Round and Cones"
## [38] "Mechanical"
## [39] "Hinges, Arms and Turntables"
## [40] "Plates Angled"
## [41] "String, Bands and Reels"
## [42] "Tubes and Hoses"
## [43] "Bricks"
## [44] "Panels"
## [45] "Bricks Sloped"
## [46] "Technic Axles"
## [47] "Magnets and Holders"
## [48] "Technic Pins"
## [49] "Technic Beams Special"
## [50] "Pneumatics"
## [51] "Belville, Scala and Fabuland"
## [52] "Supports, Girders and Cranes"
## [53] "HO Scale"
## [54] "Technic Bricks"
## [55] "Technic Bushes"
## [56] "Znap"
## [57] "Clikits"
```

`cname` will be used to name the classification.

The value `pname` will be used as input to the machine-learning algorithm. Therefore some further preprocessing is required.

The idea of the final algorithm is the following: `pname` will be split into its tokens, means single words. The result will be a tidy text data frame which will be transformed into a Document-Term-Matrix. This Matrix will be the input for the train-function of the `caret` package.

To create tokens of `pname` the following code will be used:

```
# Create token without preprocessing as a tidy text data frame
train_token <- train %>% unnest_tokens(output = word, input = pname)
dim(train_token)
```

```
## [1] 232348      3
```

```
head(train_token)
```

```
## # A tibble: 6 x 3
##   pid    cname      word
##   <chr>  <chr>      <chr>
## 1 0687b1 Non-LEGO   set
## 2 0687b1 Non-LEGO   0687
## 3 0687b1 Non-LEGO   activity
## 4 0687b1 Non-LEGO   booklet
## 5 0687b1 Non-LEGO   1
## 6 0901   Baseplates baseplate
```

```
# Number of unique tokens:
train_token %>% select(word) %>% unique() %>% count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1 11389
```

The first try to create tokens of the value `pname` generates a big data frame with a lot of tokens. For an efficient algorithm it is important to minimize the input as much as possible. This means, further analysis of `pname` should be done.

As you can see from the examples, the value `pname` contains information of each single brick. Some of these are written like 'number x number' or 'number x number x number'. These patterns describe the dimension of each brick, which means it is a single information. But the first try to create tokens separates these patters, which means that the information of their relation is lost. As a simple trick, the spaces in these patterns will be removed (e.g.: "2 x 2" -> "2x2")

```
# Remove spaces in pattern: " x " to "x"
train <- as_tibble(lapply(train, function(x) {
  gsub(" x ", "x", x)
}))

# Create tidy text data frame without further preprocessing
train_token <- train %>% unnest_tokens(output = word, input = pname)
dim(train_token)
```

```
## [1] 214302      3
```

```
head(train_token)
```

```
## # A tibble: 6 x 3
##   pid    cname      word
##   <chr>  <chr>      <chr>
## 1 0687b1 Non-LEGO   set
## 2 0687b1 Non-LEGO   0687
## 3 0687b1 Non-LEGO   activity
## 4 0687b1 Non-LEGO   booklet
## 5 0687b1 Non-LEGO   1
## 6 0901   Baseplates baseplate
```

The results shows a big impact on the data frame which becomes smaller with the optimized patterns.

Similar to any other text `pname` contains stop words which do not transport any information for the algorithm. Furthermore, numbers will be remove as well due to the same reason. The following code will remove these.

```
# Remove stop words
train_token <- train_token %>% anti_join(stop_words, by=c("word" = "word"))
dim(train_token)
```

```
## [1] 178439      3
```

```
# Remove numbers
train_token <- train_token %>% filter(!str_detect(word, "^[0-9]*$"))
dim(train_token)
```

```
## [1] 167330      3
```

```
head(train_token)
```

```
## # A tibble: 6 x 3
##   pid    cname      word
##   <chr>  <chr>      <chr>
## 1 0687b1 Non-LEGO   set
## 2 0687b1 Non-LEGO   activity
## 3 0687b1 Non-LEGO   booklet
## 4 0901   Baseplates baseplate
## 5 0901   Baseplates 16x30
## 6 0901   Baseplates set
```

Again, these preprocessings show some impact by removing tokens from the tidy text data frame.

As mentioned in the previous section, `pname` includes color information of each brick. No category is matching the color of the brick which makes them unimportant for the algorithm. Therefore, the following code will remove them:

```r
# Remove color information
train_token <- train_token %>% anti_join(colors, by=c("word" = "lcName"))
dim(train_token)
```

```
## [1] 152369      3
```

```r
head(train_token)
```

```
## # A tibble: 6 x 3
##   pid    cname      word
##   <chr>  <chr>      <chr>
## 1 0687b1 Non-LEGO   set
## 2 0687b1 Non-LEGO   activity
## 3 0687b1 Non-LEGO   booklet
## 4 0901   Baseplates baseplate
## 5 0901   Baseplates 16x30
## 6 0901   Baseplates set
```

Instead of each single step, the whole preprocessing can be summarized:

```r
# Remove spaces in pattern: " x " to "x"
train <- as_tibble(lapply(train, function(x) {
  gsub(" x ", "x", x)
}))

# Create tidy text data frame
train_token <- train %>%
  unnest_tokens(output = word, input = pname) %>%
  # Remove stop words
  anti_join(stop_words, by=c("word" = "word")) %>%
  # Remove color information
  anti_join(colors, by=c("word" = "lcName")) %>%
  # Remove numbers
  filter(!str_detect(word, "^[0-9]*$"))
```

The final data frame is in a tidy text format but not useful as an input for the function `train` of the *caret* package. The function `train` requires a matrix, each value representing a factor and each line representing an observation.

One opportunity to create this matrix is to cast the tidy text data frame as a Document-Term-Matrix. A Document-Term-Matrix contains the documents as observation in each row. In this project the document is the product, represented by the value `pid`. Each column of the Document-Term-Matrix represents a term, which means here the single token from the value `pname`. The value of each document-term relation contains the information, how often this document contains this term. The following code will the preparation and the casting. The result is a sparse-matrix.

```r
train_dtm <- train_token %>%
  # Count each word (term) for each product
  count(pid, word) %>%
  # Cast a document-term matrix
  cast_dtm(document = pid, term = word, value = n)
inspect(train_dtm)
```

```
## <<DocumentTermMatrix (documents: 23365, terms: 7908)>>
## Non-/sparse entries: 149799/184620621
## Sparsity           : 100%
## Maximal term length: 24
## Weighting          : term frequency (tf)
## Sample             :
##               Terms
## Docs            arms brick dark hands head minifig print stud tile torso
##   3626cpr1885     0     0    0     0    1       1     1    1    0     0
##   3626cpr2080     0     0    0     0    1       1     1    2    0     0
##   973pdb          0     0    0     0    0       1     1    0    0     1
##   973pn6          0     0    0     0    0       1     1    0    0     1
##   973pr2664c01    0     0    2     0    0       0     1    0    0     1
##   973pr3015c01    0     0    5     0    0       0     1    0    0     1
##   973pr3235       0     0    1     1    0       1     1    0    0     1
##   973pr3264c02    1     0    0     1    0       0     2    0    0     1
##   973pr3309c01    1     0    1     1    0       0     0    0    0     1
##   973pr3552c01    1     0    1     1    0       1     1    0    0     1
```

As shown above my using the `inspect()` function from the *tm* package, the Document-Term-Matrix is quite big. It includes several terms which are used in only 1 product. This high sparsity cause a very long time to compute each model. In fact, it my system never calculates one model in less than 6h. To reduce this time, the following time will decrease the sparsity be removing terms which are used very seldom.

```r
train_dtm <- removeSparseTerms(train_dtm, sparse = .99)
inspect(train_dtm)
```

```
## <<DocumentTermMatrix (documents: 23365, terms: 99)>>
## Non-/sparse entries: 77116/2236019
## Sparsity           : 97%
## Maximal term length: 8
## Weighting          : term frequency (tf)
## Sample             :
##               Terms
## Docs            arms brick dark hands head minifig print stud tile torso
##   3626bpr1538     0     0    1     0    1       1     1    1    0     0
##   3626cpr1538     0     0    1     0    1       1     1    1    0     0
##   3626cpr1569     0     0    2     0    1       1     1    1    0     0
##   970c00pr0657    0     0    1     0    0       0     1    0    0     0
##   973pr1581c01    1     0    1     1    0       0     1    0    0     1
##   973pr2571c01    1     0    1     1    0       0     1    0    0     1
##   973pr2664c01    0     0    2     0    0       0     1    0    0     1
##   973pr3015c01    0     0    5     0    0       0     1    0    0     1
##   973pr3235       0     0    1     1    0       1     1    0    0     1
##   973pr3552c01    1     0    1     1    0       1     1    0    0     1
```

As the `inspect()` function shows, the impact is quite massive and the models become computable.

Finally, the target variable has to be prepared by adding the category name to each row of the Document-Term-Matrix. For easy use, the target variable will be named `train_y`:

```r
train_y <- as.data.frame(train_dtm$dimnames$Docs) %>%
  left_join(train, by = c("train_dtm$dimnames$Docs" = "pid")) %>%
  select("cname")
```

To summarize earlier shown steps, the data set `train` was prepared to be used in the machine learning algorithm. Therefore, the input matrix `train_dtm` as a Document-Term-Matrix and the target vector `train_y` were created.

The next challenge is to determine the final model. To train the model, the *caret* package will be used which was introduced in previous courses.

According to a previous task, the following code evaluates different models from the *caret* package. The target is to identify the model which performs best with standard options. The selection was overtaken from a previous course and adjusted to the requirements of this project, e.g. by removing models which requires different data sets. To get results quickly, cross-validation will be performed only twice in the evaluation.

```r
# CAUTION: This evaluation will take several minutes up to hours!

# List of models from the caret package
models <- c("naive_bayes",  "svmLinear",
            "rf", "ranger",  "wsrf", "Rborist",
            "avNNet", "mlp", "monmlp")

# Configuration of trControl
control <- trainControl(method = "cv",
                        number = 2,
                        p = .9,
                        savePrediction = "final")

# Train each model on the train data set
set.seed(1, sample.kind = "Rounding")
model_fits <- lapply(models, function(model){
  train(x = as.matrix(train_dtm),
        y = factor(train_y$cname),
        method = model,
        trControl = control)
})
```

```
## Growing trees.. Progress: 99%. Estimated remaining time: 0 seconds.
## Growing trees.. Progress: 98%. Estimated remaining time: 0 seconds.
## Growing trees.. Progress: 69%. Estimated remaining time: 14 seconds.
## Fitting Repeat 1
##
## # weights:  214
## initial  value 170868.165585
## final  value 11686.000000
## converged
## Fitting Repeat 2
##
## # weights:  214
## initial  value 165880.136995
## final  value 11686.000000
## converged
## Fitting Repeat 3
##
## # weights:  214
## initial  value 159172.605593
## final  value 11686.000000
## converged
## Fitting Repeat 4
##
## # weights:  214
## initial  value 177229.244022
## final  value 11686.000000
## converged
## Fitting Repeat 5
##
## # weights:  214
## initial  value 164697.915370
## final  value 11686.000000
## converged
## Fitting Repeat 1
##
## # weights:  528
## initial  value 172029.565977
## final  value 11686.000000
## converged
## Fitting Repeat 2
##
## # weights:  528
## initial  value 167590.496427
## final  value 11686.000000
## converged
## Fitting Repeat 3
##
## # weights:  528
## initial  value 170902.926748
## final  value 11686.000000
## converged
## Fitting Repeat 4
##
## # weights:  528
## initial  value 178319.978253
## final  value 11686.000000
## converged
## Fitting Repeat 5
##
## # weights:  528
## initial  value 186783.321748
## final  value 11686.000000
## converged
## Fitting Repeat 1
##
## # weights:  842
## initial  value 183369.274899
## final  value 11686.000000
## converged
## Fitting Repeat 2
##
## # weights:  842
## initial  value 179592.247685
## final  value 11686.000000
## converged
## Fitting Repeat 3
##
## # weights:  842
## initial  value 179143.870530
## final  value 11686.000000
## converged
## Fitting Repeat 4
##
## # weights:  842
## initial  value 185723.568592
## final  value 11686.000000
## converged
## Fitting Repeat 5
##
## # weights:  842
## initial  value 197772.830408
## final  value 11686.000000
## converged
## Fitting Repeat 1
##
## # weights:  214
## initial  value 163867.871525
## iter  10 value 15105.297134
## iter  20 value 10259.033314
## iter  30 value 9169.729503
## iter  40 value 8400.419869
## iter  50 value 8111.302100
## iter  60 value 7941.089894
## iter  70 value 7791.573286
## iter  80 value 7657.558833
## iter  90 value 7578.495828
## iter 100 value 7497.566831
## final  value 7497.566831
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  214
## initial  value 172524.841257
## iter  10 value 15495.632978
## iter  20 value 10817.583180
## iter  30 value 10191.974928
## iter  40 value 9727.883650
## iter  50 value 8992.158573
## iter  60 value 8710.496702
## iter  70 value 8397.090627
## iter  80 value 8231.244726
## iter  90 value 8114.617345
## iter 100 value 8007.291796
## final  value 8007.291796
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  214
## initial  value 161529.223735
## iter  10 value 19199.328851
## iter  20 value 15803.793051
## iter  30 value 11935.553081
## iter  40 value 11764.669697
## iter  50 value 10577.698306
## iter  60 value 8292.381870
## iter  70 value 8085.487870
## iter  80 value 7927.243280
## iter  90 value 7748.050243
## iter 100 value 7614.179748
## final  value 7614.179748
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  214
## initial  value 168732.385133
## iter  10 value 15319.747753
## iter  20 value 12305.414200
## iter  30 value 11219.305664
## iter  40 value 9263.859967
## iter  50 value 8471.721959
## iter  60 value 8166.046955
## iter  70 value 7938.746511
## iter  80 value 7852.515227
## iter  90 value 7801.705063
## iter 100 value 7713.669949
## final  value 7713.669949
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  214
## initial  value 185026.413707
## iter  10 value 15780.997076
## iter  20 value 10975.759476
## iter  30 value 9435.023375
## iter  40 value 8701.176821
## iter  50 value 8436.034980
## iter  60 value 8048.602388
## iter  70 value 7918.741815
## iter  80 value 7760.299239
## iter  90 value 7684.058326
## iter 100 value 7598.756434
## final  value 7598.756434
## stopped after 100 iterations
## Fitting Repeat 1
```

```
##
## # weights:  528
## initial  value 174175.593153
## iter  10 value 18836.308556
## iter  20 value 10599.759604
## iter  30 value 8488.009844
## iter  40 value 8124.547857
## iter  50 value 7753.696993
## iter  60 value 7220.677071
## iter  70 value 6712.718015
## iter  80 value 6446.269352
## iter  90 value 6222.587480
## iter 100 value 5984.936296
## final  value 5984.936296
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  528
## initial  value 176815.206536
## iter  10 value 17385.694416
## iter  20 value 11703.703335
## iter  30 value 10126.034689
## iter  40 value 9348.419564
## iter  50 value 8049.359342
## iter  60 value 7377.208521
## iter  70 value 6918.295723
## iter  80 value 6357.410008
## iter  90 value 5983.675957
## iter 100 value 5831.308436
## final  value 5831.308436
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  528
## initial  value 169989.500526
## iter  10 value 17103.781864
## iter  20 value 11385.272069
## iter  30 value 9411.850458
## iter  40 value 8249.744702
## iter  50 value 7764.477382
## iter  60 value 7234.119972
## iter  70 value 6628.325745
## iter  80 value 6159.625833
## iter  90 value 5918.830935
## iter 100 value 5808.262618
## final  value 5808.262618
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  528
## initial  value 178517.076762
## iter  10 value 15738.227153
## iter  20 value 11224.078514
## iter  30 value 8786.591783
## iter  40 value 8544.628987
## iter  50 value 8187.273684
## iter  60 value 7813.354097
## iter  70 value 7343.747428
## iter  80 value 7023.500925
## iter  90 value 6767.103167
## iter 100 value 6641.351167
## final  value 6641.351167
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  528
## initial  value 187379.497378
## iter  10 value 19079.976428
## iter  20 value 10808.759578
## iter  30 value 8933.561347
## iter  40 value 8414.452418
## iter  50 value 7752.196496
## iter  60 value 7287.043057
## iter  70 value 6437.737331
## iter  80 value 5996.961023
## iter  90 value 5720.870044
## iter 100 value 5577.222363
## final  value 5577.222363
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  842
## initial  value 202768.229487
## iter  10 value 19701.291216
## iter  20 value 12591.985174
## iter  30 value 11293.831395
## iter  40 value 8731.783513
## iter  50 value 7878.047702
## iter  60 value 7315.832876
## iter  70 value 7036.799739
## iter  80 value 6765.835907
## iter  90 value 6549.989281
## iter 100 value 6356.002093
## final  value 6356.002093
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  842
## initial  value 181506.706503
## iter  10 value 18449.658810
## iter  20 value 12200.877243
## iter  30 value 11660.147106
## iter  40 value 11094.574346
## iter  50 value 10599.614093
## iter  60 value 9898.014091
## iter  70 value 9539.788933
## iter  80 value 9226.496932
## iter  90 value 9006.237190
## iter 100 value 8734.844402
## final  value 8734.844402
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  842
## initial  value 201517.212286
## iter  10 value 18426.044571
## iter  20 value 12507.821980
## iter  30 value 11614.440195
## iter  40 value 10971.429997
## iter  50 value 10497.211813
## iter  60 value 9983.571290
## iter  70 value 9466.793546
## iter  80 value 9126.374009
## iter  90 value 8886.230678
## iter 100 value 8640.071549
## final  value 8640.071549
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  842
## initial  value 166051.919959
## iter  10 value 20463.773778
## iter  20 value 11642.897794
## iter  30 value 8724.546213
## iter  40 value 7879.212123
## iter  50 value 7291.329109
## iter  60 value 6830.396629
## iter  70 value 6322.939083
## iter  80 value 6066.856036
## iter  90 value 5839.783704
## iter 100 value 5596.479825
## final  value 5596.479825
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  842
## initial  value 182897.066748
## iter  10 value 16758.882221
## iter  20 value 11679.166078
## iter  30 value 9698.028714
## iter  40 value 8125.463737
## iter  50 value 7033.010681
## iter  60 value 6432.058854
## iter  70 value 5937.970217
## iter  80 value 5457.221903
## iter  90 value 5200.064162
## iter 100 value 5024.812462
## final  value 5024.812462
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  214
## initial  value 183393.739807
## iter  10 value 14255.717048
## iter  20 value 11715.626820
## iter  30 value 11631.888695
## iter  40 value 11613.621078
## iter  50 value 11547.717215
## iter  60 value 10278.669601
## iter  70 value 10239.022525
## iter  80 value 10237.075352
## iter  90 value 10234.204693
## iter 100 value 10227.305260
## final  value 10227.305260
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  214
## initial  value 164653.190738
## iter  10 value 13958.340770
## iter  20 value 11712.198305
## iter  30 value 10517.109302
## iter  40 value 10396.436569
## iter  50 value 8988.783689
## iter  60 value 8480.340938
## iter  70 value 8437.522703
## iter  80 value 8299.006038
## iter  90 value 8282.345882
## iter 100 value 8138.575157
## final  value 8138.575157
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  214
## initial  value 160310.739879
## iter  10 value 14182.615535
## iter  20 value 11710.737761
## iter  30 value 10444.449885
## iter  40 value 10140.811339
## iter  50 value 8478.553900
## iter  60 value 8314.337088
## iter  70 value 8242.047855
## iter  80 value 8206.429750
## iter  90 value 8200.057590
## iter 100 value 8198.498480
## final  value 8198.498480
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  214
## initial  value 170151.987787
## iter  10 value 13994.828559
## iter  20 value 11558.708254
## iter  30 value 10234.080444
## iter  40 value 8254.458951
## iter  50 value 8187.321001
## iter  60 value 8170.023695
## iter  70 value 8165.967157
## iter  80 value 8160.586104
## iter  90 value 8157.252193
## iter 100 value 8156.410985
## final  value 8156.410985
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  214
## initial  value 166673.372292
## iter  10 value 14043.812742
## iter  20 value 11713.178989
## iter  30 value 10428.214236
## iter  40 value 10346.125401
## iter  50 value 8444.481231
## iter  60 value 8181.035241
## iter  70 value 8088.118570
## iter  80 value 8050.497377
## iter  90 value 8014.253532
## iter 100 value 7975.448018
## final  value 7975.448018
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  528
## initial  value 176869.066179
## iter  10 value 11832.827134
## iter  20 value 11685.361880
## iter  30 value 11680.917388
## iter  40 value 11679.809973
## iter  50 value 11671.448655
## iter  60 value 11614.520375
## iter  70 value 11601.128384
## iter  80 value 10619.586016
## iter  90 value 10383.676378
## iter 100 value 10207.799651
## final  value 10207.799651
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  528
## initial  value 174567.162488
## iter  10 value 15120.706591
## iter  20 value 11725.599471
## iter  30 value 10416.744747
## iter  40 value 8985.525996
## iter  50 value 8970.271825
## iter  60 value 8961.956886
## iter  70 value 8958.108621
## iter  80 value 8955.207317
## iter  90 value 8928.136728
## iter 100 value 8924.923440
## final  value 8924.923440
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  528
## initial  value 177143.832103
## iter  10 value 11833.572646
## iter  20 value 11652.472230
## iter  30 value 11641.955064
## iter  40 value 11481.071954
## iter  50 value 11264.782727
## iter  60 value 11211.507863
## iter  70 value 11203.583142
## iter  80 value 11201.822202
## iter  90 value 11198.842320
## iter 100 value 11198.405224
## final  value 11198.405224
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  528
## initial  value 196658.931317
## iter  10 value 14989.905241
## iter  20 value 11724.091434
## iter  30 value 11677.856510
## iter  40 value 8686.120182
## iter  50 value 8383.606728
## iter  60 value 8331.372922
## iter  70 value 8313.159951
## iter  80 value 8307.714072
## iter  90 value 8306.926463
## iter 100 value 8306.261876
## final  value 8306.261876
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  528
## initial  value 179811.640613
## iter  10 value 15820.809934
## iter  20 value 11738.034756
## iter  30 value 11690.654600
## iter  40 value 11650.713495
## iter  50 value 11588.275502
## iter  60 value 11410.867026
## iter  70 value 11374.599221
## iter  80 value 11167.898261
## iter  90 value 11124.742027
## iter 100 value 11050.082505
## final  value 11050.082505
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  842
## initial  value 174228.278025
## iter  10 value 11842.587740
## iter  20 value 11676.808746
## iter  30 value 9989.887268
## iter  40 value 7992.841091
## iter  50 value 7806.737408
## iter  60 value 7713.508257
## iter  70 value 7689.832982
## iter  80 value 7676.353068
## iter  90 value 7520.442352
## iter 100 value 7355.000278
## final  value 7355.000278
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  842
## initial  value 193233.214189
## iter  10 value 11883.917633
## iter  20 value 11686.526868
## iter  30 value 11649.324885
## iter  40 value 11392.438962
## iter  50 value 11292.597806
## iter  60 value 11289.178509
## iter  70 value 11262.945475
## iter  80 value 11256.258550
## iter  90 value 11254.825936
## iter 100 value 11196.817818
## final  value 11196.817818
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  842
## initial  value 186952.636667
## iter  10 value 11876.108373
## iter  20 value 11655.989800
## iter  30 value 11140.611742
## iter  40 value 7840.452675
## iter  50 value 7087.807921
## iter  60 value 6959.544897
## iter  70 value 6883.602312
## iter  80 value 6862.409494
## iter  90 value 6843.891157
## iter 100 value 6812.162182
## final  value 6812.162182
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  842
## initial  value 172817.289319
## iter  10 value 11844.946284
## iter  20 value 11657.383083
## iter  30 value 10979.361141
## iter  40 value 10975.367399
## iter  50 value 10975.172533
## iter  60 value 10974.513742
## iter  70 value 10974.325723
## iter  80 value 10973.375423
## iter  90 value 10971.409531
## iter 100 value 10926.664827
## final  value 10926.664827
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  842
## initial  value 159597.182613
## iter  10 value 11827.117683
## iter  20 value 11688.728499
## iter  30 value 11374.688996
## iter  40 value 10943.644899
## iter  50 value 10772.601081
## iter  60 value 10656.229944
## iter  70 value 10623.002712
## iter  80 value 10611.069000
## iter  90 value 10602.417717
## iter 100 value 10599.149878
## final  value 10599.149878
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  214
## initial  value 169579.467884
## final  value 11679.000000
## converged
## Fitting Repeat 2
##
## # weights:  214
## initial  value 171970.402703
## final  value 11679.000000
## converged
## Fitting Repeat 3
##
## # weights:  214
## initial  value 149978.805683
## final  value 11679.000000
## converged
## Fitting Repeat 4
##
## # weights:  214
## initial  value 152118.920805
## final  value 11679.000000
## converged
## Fitting Repeat 5
##
## # weights:  214
## initial  value 173281.094650
## final  value 11679.000000
## converged
## Fitting Repeat 1
##
## # weights:  528
## initial  value 187075.551078
## final  value 11679.000000
## converged
## Fitting Repeat 2
##
## # weights:  528
## initial  value 179151.441170
## final  value 11679.000000
## converged
## Fitting Repeat 3
##
## # weights:  528
## initial  value 175227.797828
## final  value 11679.000000
## converged
## Fitting Repeat 4
##
## # weights:  528
## initial  value 180873.474620
## final  value 11679.000000
## converged
## Fitting Repeat 5
##
## # weights:  528
## initial  value 182934.701894
## final  value 11679.000000
## converged
## Fitting Repeat 1
##
## # weights:  842
## initial  value 179967.383565
## final  value 11679.000000
## converged
## Fitting Repeat 2
##
## # weights:  842
## initial  value 188535.160179
## final  value 11679.000000
## converged
## Fitting Repeat 3
##
## # weights:  842
## initial  value 182674.439621
## final  value 11679.000000
## converged
## Fitting Repeat 4
##
## # weights:  842
## initial  value 194700.452562
## final  value 11679.000000
## converged
## Fitting Repeat 5
##
## # weights:  842
## initial  value 174664.621195
## final  value 11679.000000
## converged
## Fitting Repeat 1
##
## # weights:  214
## initial  value 179549.764532
## iter  10 value 15733.305998
## iter  20 value 11176.486481
## iter  30 value 10109.941735
## iter  40 value 8865.609579
## iter  50 value 8316.997789
## iter  60 value 8207.009192
## iter  70 value 7795.518364
## iter  80 value 7616.997295
## iter  90 value 7523.900269
## iter 100 value 7497.609961
## final  value 7497.609961
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  214
## initial  value 184694.148723
## iter  10 value 18811.891493
## iter  20 value 16277.627629
## iter  30 value 15677.760265
## iter  40 value 14564.179776
## iter  50 value 10020.799190
## iter  60 value 9032.708954
## iter  70 value 8441.746624
## iter  80 value 8231.273277
## iter  90 value 7936.534959
## iter 100 value 7829.187225
## final  value 7829.187225
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  214
## initial  value 172565.356668
## iter  10 value 19600.571455
## iter  20 value 16021.626413
## iter  30 value 15707.489330
## iter  40 value 15340.185657
## iter  50 value 15010.999476
## iter  60 value 12714.850510
## iter  70 value 11656.070930
## iter  80 value 10213.161021
## iter  90 value 9067.310511
## iter 100 value 8191.131433
## final  value 8191.131433
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  214
## initial  value 169290.022416
## iter  10 value 19588.092437
## iter  20 value 13299.831913
## iter  30 value 11848.807132
## iter  40 value 11511.371860
## iter  50 value 10447.871646
## iter  60 value 9602.791420
## iter  70 value 9023.809157
## iter  80 value 8535.340392
## iter  90 value 8392.264621
## iter 100 value 8333.134434
## final  value 8333.134434
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  214
## initial  value 171325.269346
## iter  10 value 15625.681885
## iter  20 value 10667.643363
## iter  30 value 9589.088081
## iter  40 value 8290.852741
## iter  50 value 8093.445134
## iter  60 value 7978.811799
## iter  70 value 7852.831511
## iter  80 value 7707.001295
## iter  90 value 7662.355664
## iter 100 value 7563.387116
## final  value 7563.387116
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  528
## initial  value 174697.655516
## iter  10 value 18227.302529
## iter  20 value 9249.485769
## iter  30 value 8721.159044
## iter  40 value 8162.680800
## iter  50 value 7455.706565
## iter  60 value 6415.231600
## iter  70 value 5892.699885
## iter  80 value 5665.636065
## iter  90 value 5525.626429
## iter 100 value 5436.752136
## final  value 5436.752136
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  528
## initial  value 188660.365442
## iter  10 value 17734.784222
## iter  20 value 11225.516514
## iter  30 value 9163.205696
## iter  40 value 8717.333576
## iter  50 value 8116.891704
## iter  60 value 7413.978118
## iter  70 value 7021.974054
## iter  80 value 6837.992961
## iter  90 value 6774.322837
## iter 100 value 6643.569552
## final  value 6643.569552
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  528
## initial  value 167394.490013
## iter  10 value 18848.121486
## iter  20 value 10301.116803
## iter  30 value 8912.343645
## iter  40 value 8480.200210
## iter  50 value 8088.313524
## iter  60 value 7625.025556
## iter  70 value 7044.533767
## iter  80 value 6636.156492
## iter  90 value 6292.638311
## iter 100 value 6027.093877
## final  value 6027.093877
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  528
## initial  value 164259.808180
## iter  10 value 17413.849060
## iter  20 value 12453.660507
## iter  30 value 8386.927756
## iter  40 value 7737.048376
## iter  50 value 7301.458531
## iter  60 value 6935.944510
## iter  70 value 6444.621173
## iter  80 value 6214.189315
## iter  90 value 6084.703839
## iter 100 value 5907.720685
## final  value 5907.720685
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  528
## initial  value 177574.010571
## iter  10 value 20754.305132
## iter  20 value 16340.554392
## iter  30 value 15944.915189
## iter  40 value 12734.948986
## iter  50 value 10281.263323
## iter  60 value 9137.139065
## iter  70 value 7868.687919
## iter  80 value 7331.923658
## iter  90 value 6744.358178
## iter 100 value 6517.998588
## final  value 6517.998588
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  842
## initial  value 174774.032583
## iter  10 value 17153.731586
## iter  20 value 12150.662616
## iter  30 value 11303.790143
## iter  40 value 10643.862862
## iter  50 value 10209.314433
## iter  60 value 9726.660117
## iter  70 value 9186.601705
## iter  80 value 6200.386871
## iter  90 value 5826.177245
## iter 100 value 5452.010842
## final  value 5452.010842
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  842
## initial  value 162352.300013
## iter  10 value 16287.195985
## iter  20 value 10993.608475
## iter  30 value 9161.942074
## iter  40 value 7844.504779
## iter  50 value 7069.267686
## iter  60 value 6288.978435
## iter  70 value 5797.979516
## iter  80 value 5445.412834
## iter  90 value 5294.723800
## iter 100 value 5172.180649
## final  value 5172.180649
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  842
## initial  value 176755.821914
## iter  10 value 15847.707086
## iter  20 value 13463.954994
## iter  30 value 11838.722557
## iter  40 value 11087.501531
## iter  50 value 9506.340678
## iter  60 value 8297.636524
## iter  70 value 7758.093192
## iter  80 value 7405.992145
## iter  90 value 7138.178869
## iter 100 value 6813.146109
## final  value 6813.146109
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  842
## initial  value 189750.456549
## iter  10 value 20062.498281
## iter  20 value 10219.122298
## iter  30 value 8233.574685
## iter  40 value 7687.053988
## iter  50 value 7212.560130
## iter  60 value 6695.074393
## iter  70 value 6301.238314
## iter  80 value 5762.076509
## iter  90 value 5390.577135
## iter 100 value 5145.042876
## final  value 5145.042876
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  842
## initial  value 155819.768634
## iter  10 value 15807.302127
## iter  20 value 10316.751726
## iter  30 value 9114.808404
## iter  40 value 8370.491128
## iter  50 value 7724.988389
## iter  60 value 7113.014324
## iter  70 value 6057.492203
## iter  80 value 5583.825027
## iter  90 value 5257.948778
## iter 100 value 5144.485649
## final  value 5144.485649
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  214
## initial  value 179342.090743
## iter  10 value 14477.053842
## iter  20 value 11711.259364
## iter  30 value 9715.426870
## iter  40 value 8741.861490
## iter  50 value 8260.315360
## iter  60 value 8125.063636
## iter  70 value 8019.178484
## iter  80 value 7966.109743
## iter  90 value 7851.996110
## iter 100 value 7716.913545
## final  value 7716.913545
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  214
## initial  value 187161.652806
## iter  10 value 14211.178567
## iter  20 value 11708.033710
## iter  30 value 10428.117569
## iter  40 value 10353.813778
## iter  50 value 10340.452839
## iter  60 value 10338.688829
## iter  70 value 10267.243242
## iter  80 value 10257.822393
## iter  90 value 10233.077437
## iter 100 value 10213.191929
## final  value 10213.191929
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  214
## initial  value 175292.379732
## iter  10 value 14313.922392
## iter  20 value 11709.378587
## iter  30 value 10370.872411
## iter  40 value 10342.136650
## iter  50 value 10273.590987
## iter  60 value 10271.742614
## iter  70 value 10269.738656
## iter  80 value 10269.500125
## iter  90 value 10267.868491
## iter 100 value 10266.322803
## final  value 10266.322803
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  214
## initial  value 179512.901297
## iter  10 value 14111.317676
## iter  20 value 11706.650782
## iter  30 value 10421.137608
## iter  40 value 10349.770531
## iter  50 value 10337.237291
## iter  60 value 9410.879960
## iter  70 value 8383.813016
## iter  80 value 8184.855848
## iter  90 value 8173.338220
## iter 100 value 8162.998241
## final  value 8162.998241
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  214
## initial  value 179612.821338
## iter  10 value 15684.082414
## iter  20 value 15582.762393
## iter  30 value 14931.330467
## iter  40 value 14924.538656
## iter  50 value 10363.542200
## iter  60 value 10207.405175
## iter  70 value 9573.858651
## iter  80 value 9389.957980
## iter  90 value 9296.047990
## iter 100 value 9269.332599
## final  value 9269.332599
## stopped after 100 iterations
## Fitting Repeat 1
```

```
##
## # weights:  528
## initial  value 172964.278414
## iter  10 value 14966.731440
## iter  20 value 11716.904963
## iter  30 value 11679.043465
## iter  40 value 11678.643189
## iter  50 value 11536.651960
## iter  60 value 8355.512330
## iter  70 value 8232.699439
## iter  80 value 8199.227108
## iter  90 value 8193.088195
## iter 100 value 8188.690168
## final  value 8188.690168
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  528
## initial  value 191736.434330
## iter  10 value 15284.340864
## iter  20 value 11720.560521
## iter  30 value 8311.590288
## iter  40 value 8200.184377
## iter  50 value 7947.009284
## iter  60 value 7677.207277
## iter  70 value 7614.718978
## iter  80 value 7577.820968
## iter  90 value 7562.090895
## iter 100 value 7547.807677
## final  value 7547.807677
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  528
## initial  value 176882.831898
## iter  10 value 15158.560697
## iter  20 value 11719.116604
## iter  30 value 9797.008182
## iter  40 value 8798.392689
## iter  50 value 8790.389618
## iter  60 value 8779.618998
## iter  70 value 8683.223019
## iter  80 value 8678.393195
## iter  90 value 8676.721551
## iter 100 value 8658.425887
## final  value 8658.425887
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  528
## initial  value 187243.710123
## iter  10 value 15034.263341
## iter  20 value 11717.683553
## iter  30 value 9915.424824
## iter  40 value 8398.474614
## iter  50 value 8312.137103
## iter  60 value 8305.579954
## iter  70 value 8290.650243
## iter  80 value 8277.017231
## iter  90 value 8269.604932
## iter 100 value 8265.935083
## final  value 8265.935083
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  528
## initial  value 173790.147573
## iter  10 value 11822.829317
## iter  20 value 11681.550595
## iter  30 value 11609.391468
## iter  40 value 11501.067495
## iter  50 value 11444.102501
## iter  60 value 11221.469224
## iter  70 value 11144.679018
## iter  80 value 11133.378991
## iter  90 value 11117.138710
## iter 100 value 11100.595378
## final  value 11100.595378
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  842
## initial  value 177322.016567
## iter  10 value 15837.818017
## iter  20 value 11680.266253
## iter  30 value 11570.404239
## iter  40 value 11561.802046
## iter  50 value 11489.009468
## iter  60 value 11091.441828
## iter  70 value 11037.437939
## iter  80 value 10919.096293
## iter  90 value 10865.788701
## iter 100 value 10805.577350
## final  value 10805.577350
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  842
## initial  value 176643.254218
## iter  10 value 11837.040621
## iter  20 value 11682.240284
## iter  30 value 10860.900017
## iter  40 value 9840.909058
## iter  50 value 9353.258045
## iter  60 value 8529.773730
## iter  70 value 8504.408750
## iter  80 value 8399.083855
## iter  90 value 8395.374015
## iter 100 value 8393.838972
## final  value 8393.838972
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  842
## initial  value 200557.780677
## iter  10 value 11847.903291
## iter  20 value 11681.318630
## iter  30 value 11664.019204
## iter  40 value 10364.947580
## iter  50 value 8569.554984
## iter  60 value 8447.617362
## iter  70 value 8416.654435
## iter  80 value 8398.262647
## iter  90 value 8318.675712
## iter 100 value 8246.995913
## final  value 8246.995913
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  842
## initial  value 183896.159201
## iter  10 value 11845.575474
## iter  20 value 11681.744619
## iter  30 value 10472.917634
## iter  40 value 9103.624864
## iter  50 value 8954.599370
## iter  60 value 8918.802895
## iter  70 value 8912.626879
## iter  80 value 8907.231045
## iter  90 value 8906.681848
## iter 100 value 8904.590413
## final  value 8904.590413
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  842
## initial  value 160670.172055
## iter  10 value 11819.171909
## iter  20 value 11662.689134
## iter  30 value 11447.297096
## iter  40 value 11419.835018
## iter  50 value 11053.078895
## iter  60 value 10979.015893
## iter  70 value 10976.196113
## iter  80 value 10974.903438
## iter  90 value 10973.383639
## iter 100 value 10960.265316
## final  value 10960.265316
## stopped after 100 iterations
## Fitting Repeat 1
##
## # weights:  842
## initial  value 391279.590286
## iter  10 value 30253.015368
## iter  20 value 23452.195880
## iter  30 value 19643.891942
## iter  40 value 18683.211805
## iter  50 value 17128.348864
## iter  60 value 16360.268639
## iter  70 value 14786.041075
## iter  80 value 14004.054238
## iter  90 value 12962.597558
## iter 100 value 12134.374763
## final  value 12134.374763
## stopped after 100 iterations
## Fitting Repeat 2
##
## # weights:  842
## initial  value 305333.499651
## iter  10 value 33720.355608
## iter  20 value 25789.692156
## iter  30 value 23330.309358
## iter  40 value 22728.632016
## iter  50 value 21979.615812
## iter  60 value 16796.694143
## iter  70 value 15661.230745
## iter  80 value 15340.687947
## iter  90 value 15124.827257
## iter 100 value 14646.978557
## final  value 14646.978557
## stopped after 100 iterations
## Fitting Repeat 3
##
## # weights:  842
## initial  value 339398.529484
## iter  10 value 42549.624034
## iter  20 value 29021.814176
## iter  30 value 23066.524425
## iter  40 value 19174.754856
## iter  50 value 17425.972043
## iter  60 value 15902.285491
## iter  70 value 14505.052077
## iter  80 value 13673.570686
## iter  90 value 13202.067346
## iter 100 value 12877.542313
## final  value 12877.542313
## stopped after 100 iterations
## Fitting Repeat 4
##
## # weights:  842
## initial  value 354893.779050
## iter  10 value 39926.800822
## iter  20 value 27065.907283
## iter  30 value 24556.250624
## iter  40 value 21976.311754
## iter  50 value 17897.985476
## iter  60 value 15667.856083
## iter  70 value 14466.854026
## iter  80 value 13881.730217
## iter  90 value 12842.423238
## iter 100 value 11725.779538
## final  value 11725.779538
## stopped after 100 iterations
## Fitting Repeat 5
##
## # weights:  842
## initial  value 359940.663570
## iter  10 value 42253.253618
## iter  20 value 35068.269086
## iter  30 value 27234.144153
## iter  40 value 22576.660435
## iter  50 value 18494.658074
## iter  60 value 17987.324317
## iter  70 value 17503.705585
## iter  80 value 16941.559138
## iter  90 value 15865.471953
## iter 100 value 15005.810016
## final  value 15005.810016
## stopped after 100 iterations
## ** Ensemble 1
## 0.9793421
## ** 0.9793421
##
## ** Ensemble 1
## 0.9452198
## ** 0.9452198
##
## ** Ensemble 1
## 0.9143009
## ** 0.9143009
##
## ** Ensemble 1
## 0.9793775
## ** 0.9793775
##
## ** Ensemble 1
## 0.9447521
## ** 0.9447521
##
## ** Ensemble 1
## 0.914346
## ** 0.914346
##
## ** Ensemble 1
## 0.9142849
## ** 0.9142849
```

```r
# Summarize results
## Add benchmark of 80% accuracy by manual work
model_results <- tibble(Method = "Benchmark (Manual)", Accuracy = 0.8)
```

```r
## Add each model's accuracy to result table
for(i in seq(1:length(models))){
  model_results <- bind_rows(model_results, tibble(Method = model_fits[[i]]$method,
                                                   Accuracy = max(model_fits[[i]]$results$Accuracy)))
}

## Show results
model_results %>% arrange(desc(Accuracy))
```

```
## # A tibble: 10 x 2
##    Method             Accuracy
##    <chr>                 <dbl>
##  1 Benchmark (Manual)    0.8
##  2 rf                    0.788
##  3 ranger                0.786
##  4 wsrf                  0.784
##  5 svmLinear             0.774
##  6 mlp                   0.704
##  7 avNNet                0.677
##  8 monmlp                0.512
##  9 naive_bayes           0.329
## 10 Rborist               0.329
```

```r
# Add model information
names(model_fits) <- models
```

```r
model_fits$rf
```

```
## Random Forest
##
## 23365 samples
##    99 predictor
##    57 classes: 'Bars, Ladders and Fences', 'Baseplates', 'Belville, Scala and Fabuland', 'Bionicle, 
##
## No pre-processing
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 11687, 11678
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.5767520  0.4369065
##   50    0.7882736  0.7522212
##   99    0.7851491  0.7485576
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 50.
```

As the results show, the model *rf* performs best with the current data set but still worse than the given benchmark of 80%. To exceed this benchmark, the rf model will be tuned by adjustig its parameter.

To tune the model, the *caret* packages offers limited options. One option is of course to adjust the parameter on how many turns are computed in cross validation. Compute more turns results often in better insights of the model, especially reducing the overfitting to the training data. For performance reasons, this project will stay with 2 turns for the cross validation, which already take a lot of time.

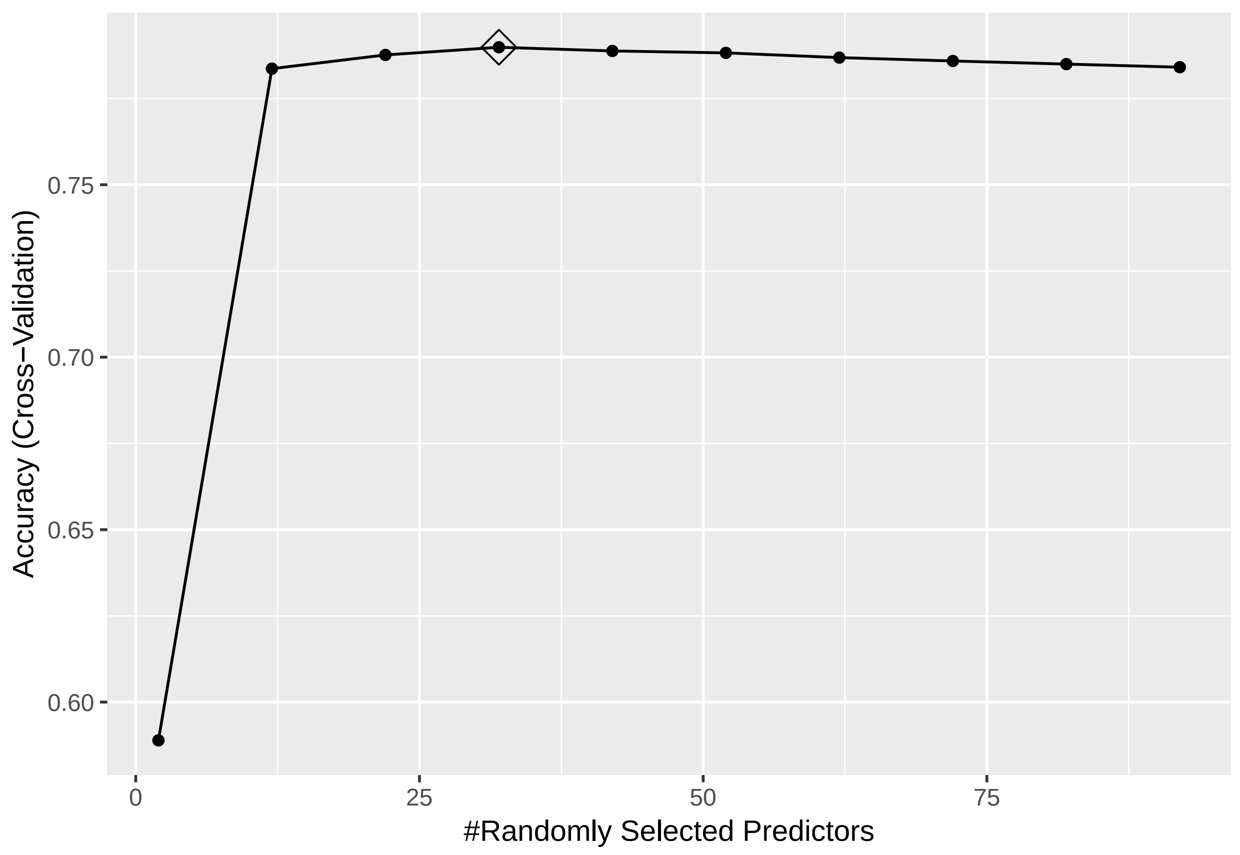
Indeed, to train the rf-model, the *caret* package only allows to tune the parameter `mtry` which represents the number of variables randomly sampled as candidates at each split. There is no further parameter for tuning available. In the previous calculation the model was trained with 3 tuning parameters, automatically set by the *caret* package: 2, 50, 99. The extend the range, the tuning grid will be set to every thenth values of `mtry` between 2 and 100.

```
# Configuration of trControl
control <- trainControl(method = "cv",
                        number = 2,
                        p = .9,
                        savePrediction = "final")

# Define tuneGrid for mtry
tunegrid <- data.frame(mtry = seq(2, 100, 10))

# Train rf model on the train data set
set.seed(1, sample.kind = "Rounding")
model_fit <- train(x = as.matrix(train_dtm),
                   y = factor(train_y$cname),
                   method = "rf",
                   trControl = control,
                   tuneGrid = tunegrid)

# Accuracy of the traing data set
ggplot(model_fit, highlight = TRUE)
```



```r
# Set mtry_tune to the best mtry to speed calculations
mtry_tune <- model_fit$bestTune$mtry
```

As the results show, there is just a little improvement by tuning the parameter `mtry`. The reason for this is, that the function `train` of the *caret* package allready tuned the model with standard parameters.

Because the model *rf* with above identified tuning parameter `mtry_tune` is the model with the highest accucary, the final model will use these.

Results

The data analysis and evaluation of the different models in the previous section showed, that the model *rf* performed best by providing the highest accuracy for the `train` data set. In a second step, this model was tuned by adjusting its parameter and using 10 instead of 2 cross-validation turns.

To discuss the model performance and compare it to the benachmarking of 80% accuracy, the evaluation toward the `validation` data set is necessary. The following code prepares the `validation` data set, computes the prediction and evaluates the output of the algorithm.

```r
# Prepare validation data set
## Remove spaces in pattern: " x " to "x"
validation <- as_tibble(lapply(validation, function(x) {
  gsub(" x ", "x", x)
}))
```

```r
# Create tidy text data frame
validation_token <- validation %>%
  unnest_tokens(output = word, input = pname) %>%
  # Remove stop words
  anti_join(stop_words, by=c("word" = "word")) %>%
  # Remove color information
  anti_join(colors, by=c("word" = "lcName")) %>%
  # Remove numbers
  filter(!str_detect(word, "^[0-9]*$"))

## Cast DTM
validation_dtm <- validation_token %>%
  # Count each word (term) for each product
  count(pid, word) %>%
  # Cast a document-term matrix
  cast_dtm(document = pid, term = word, value = n)

## Generate validation target vector
validation_y <- as.data.frame(validation_dtm$dimnames$Docs) %>%
  left_join(validation, by = c("validation_dtm$dimnames$Docs" = "pid")) %>%
  select("cname")

# Compute prediction by validaion data set
prediction <- predict(model_fit, validation_dtm)

# Evaluate prediction
mean(prediction == validation_y$cname)
```

```
## [1] 0.7810831
```

The optimized model generates a prediction with an accuracy of 78.10% on the validation data set. The result is lower than the current benchmark of manual work.

Conclusion

The target of this project was to develop an algorithm which supports my recurring task to classify products into our web shop categories. As a benchmark, internal tests state an accuracy of 80% by doing this manually. In additional, controlling calculates with about 1h work for about 1,000 products.

To develop this algorithm, the *LEGO Database* data set from Kaggle was used. This data set provides information on LEGO bricks, including their unique product IDs, their product name and the corresponding product category. As a best performing model, this project identifies the *rf* model by using the *caret* package. Finally, this model achieved an accuracy of about 78.10% which is lower than the benchmarking.

Nonetheless, it does not mean that this project fails, due to several reasons: 1. No information about the accuracy of the *LEGO Database* is provided. This means, that the algorithm could be more accurate than calculated because of fault classification in the original data set. 2. The benchmark of 80% is more a guessed value than a counted fact. This number was calculated by statistical methods. Furthermore, the circumstances were to show how good the classification is, not about its potential to be improved. 3. For electronic components, we could use short descriptions with about 80 characters and long descriptions with about 1024 characters. Furthermore, technical parameters for each product are provided by the manufacturer. This information is available for the manual process but not in the used data set. 4. Probably the strongest argument: While manual work takes about 1h to classify 1,000 products, it takes seconds to do the same with the algorithm if the model is computed. But it is work which can be processed without manual controll and therefore is no human resource.

To conclude, this project already provides some good insights of the possible algorithm for future usage. The next step would be to check it with data of my real business. Unfortunately, I am not allowed to share these data and the results.