Video Game Sales

Felipe Urrego

08/03/2019

Summary

Introduction

- What is Machine Learning?

In simple words, we can say that it is just a problem of approximation of a function or a relationship with the purpose to obtain new information. It can be any function of any complexity. It can even do not exist, in this case, we assume that there is a kind of law of nature or a relationship that we want to approximate.

- Notations

We note this function $f(x)$ and its values $y = f(x)$ (or $y = f(x) + \epsilon$ where ϵ is random). Conventions for x and y depend on the field of science - Machine Learning or Statistical Learning - they are a bit different.

x / Features / Independent Variables (it's not the case for the raw data, but we should treat it in order to get only independent variables, for example, with help of PCA - Principal Component Analysis). y / Label / Dependent Variable.

x can be a vector (vector of features), its length is the number of independent variables. y is usually a scalar value.

Observation is one couple $\{x, y\}$ or just one x.

There are different types of Machine Learning problems, but we now consider Supervised Learning problem, it is when we have labels as described.

- Problem formulation

We have two sets of data, one is labelled (it has x and y and called 'train/training set') and another not (it has only x and called 'test set'). Train set is $\{x_i, y_i\}_{i=0,1..n}$, where n is the number of observations (they ideally should be independent).

The objective is to get y for the test set (we say to predict or to forecast). So, we choose a Machine Learning model g and train (or fit) it on the train set (calibrate its internal parameters) so that $g(x)$ looks similar to y for $\{x, y\}$ in the train set. We quantify "look similar" by introducing a function to minimise called Error Measure. Thus, g is our approximation of f and $g(x)$ for x in the test set (called Prediction) is our guess of y for the test set as asked.

- Error measure

Figure 1: k-Cross Validation (k=10)

There are different error measures (also called Loss/Cost/Objective Functions), but here we use the most common one - Root Mean Square Error (RMSE), for a model g:

$$RMSE(g) = \sqrt{\frac{1}{n}\sum_{i=1..n}(y_i - g(x_i))^2}$$

```
RMSE <- function(y, g)
  return(sqrt(mean( (y-g)**2 )))
```

- How to know which model is better? (Cross Validation and Overfitting)

There are many techniques for that, but the most popular is 'k-Cross Validation'. For a considered model, we randomly divide the train set into k parts and perform k actions - we take the first part as new test set and other nine as new train set, we train the model on the new train set and predict y for the new test set. Then we compare the predictions with the real y (we say we measure the error). Then, we take another part as new train set and other nine as new test set and go on until we have k error values. We take the average over these k errors and call "error of the k-cross validation" as on the Figure 1.

This average error is a reasonable way to compare models (a model that describes the nature better than others has the smallest error) and to avoid the overfitting. Overfitting is a situation of using of an overcomplicated model as on the Figure 2 (the model starts to explain the noise)

- Video Game Sales with Ratings

The database consists of a list of video games with its scores, ratings and sales. For each game we have a year of release, a platform, a genre, a publisher, a score given by critics (aggregated) and a score given by users, a number of critics and a number of users participated in the scoring. Then we have a rating (age rating assigned by ESRB) and a number of sales that we want to predict.

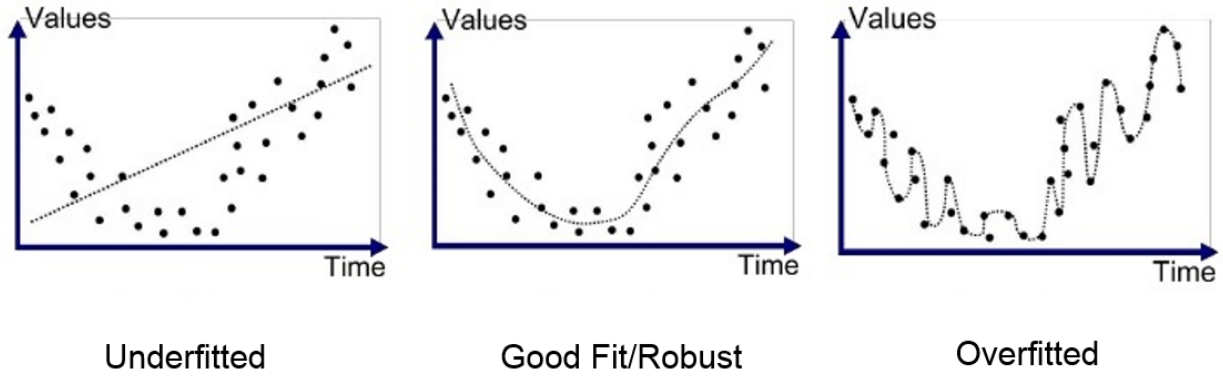


Figure 2: Overfitting

Names are duplicated because a game can be released for multiple platforms. In any case, there are too many of names, so that we are not able to use them all. We remove Name column.

Preprocessing

```
# List of packages we need
listOfPackages <- list("randomForest",
                       "kableExtra",
                       "tidyverse",
                       "corrplot",
                       "ggplot2",
                       "caret",
                       "knitr")

# Function that loads and installs if necessary indicated packages
UsePackages <- function(listOfPackages) {
  for (p in listOfPackages){
    # if (!is.element(p, installed.packages()[,1]))
    #   install.packages(p)
    require(p, character.only = TRUE)
  }
}
UsePackages(listOfPackages)

# precision
prec <- 4
```

Read and clean the data

```
# read the data
the_data <- read_csv("input/Video_Games_Sales_as_at_22_Dec_2016.csv",
```

```
                      col_types = "cciccddddddidicc")
# structure of the data
str(the_data, give.attr=FALSE)
```

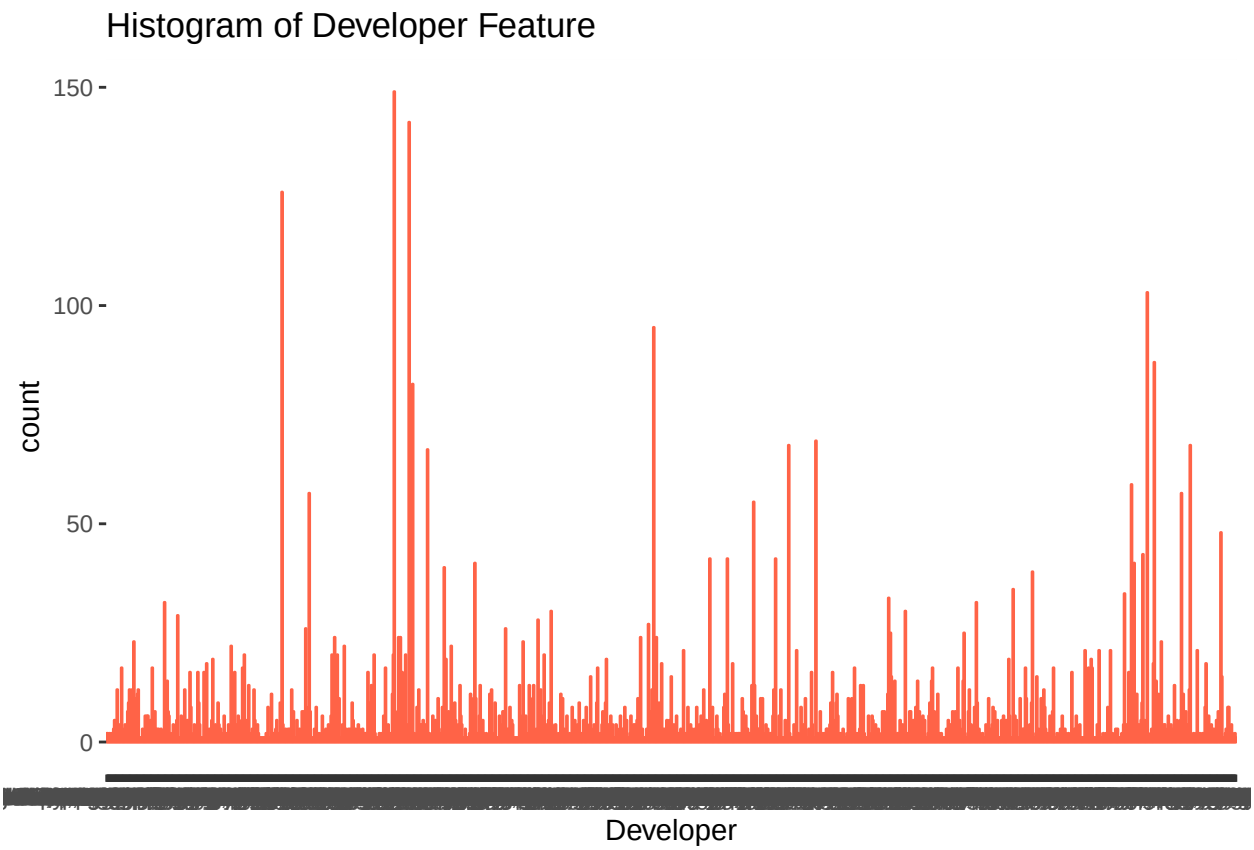
```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 16719 obs. of  16 variables:
##  $ Name           : chr  "Wii Sports" "Super Mario Bros." "Mario Kart Wii" "Wii Sports Resort" ...
##  $ Platform       : chr  "Wii" "NES" "Wii" "Wii" ...
##  $ Year_of_Release: int  2006 1985 2008 2009 1996 1989 2006 2006 2009 1984 ...
##  $ Genre          : chr  "Sports" "Platform" "Racing" "Sports" ...
##  $ Publisher      : chr  "Nintendo" "Nintendo" "Nintendo" "Nintendo" ...
##  $ NA_Sales       : num  41.4 29.1 15.7 15.6 11.3 ...
##  $ EU_Sales       : num  28.96 3.58 12.76 10.93 8.89 ...
##  $ JP_Sales       : num  3.77 6.81 3.79 3.28 10.22 ...
##  $ Other_Sales    : num  8.45 0.77 3.29 2.95 1 0.58 2.88 2.84 2.24 0.47 ...
##  $ Global_Sales   : num  82.5 40.2 35.5 32.8 31.4 ...
##  $ Critic_Score   : num  76 NA 82 80 NA NA 89 58 87 NA ...
##  $ Critic_Count   : int  51 NA 73 73 NA NA 65 41 80 NA ...
##  $ User_Score     : num  8 NA 8.3 8 NA NA 8.5 6.6 8.4 NA ...
##  $ User_Count     : int  322 NA 709 192 NA NA 431 129 594 NA ...
##  $ Developer      : chr  "Nintendo" NA "Nintendo" "Nintendo" ...
##  $ Rating         : chr  "E" NA "E" "E" ...
```

The dataset consists of 16719 observations and 16 variables. Three of the variables are sales in different regions of the world and their sum is equal to the global sales variable which is the label that we are going to predict, so we remove region sales.

```
# look at variables that have a big number of unique values (more than 1 %)
the_data %>% select_if(is.character) %>%
  summarise_all(function(x) length(unique(x))/n()) %>% select_if(function(x) x > 0.01)
```

```
## # A tibble: 1 x 3
##    Name Publisher Developer
##   <dbl>     <dbl>     <dbl>
## 1 0.692    0.0348     0.102
```

```
# there are too many unique values to be used

# plot histrogram of the Developer Feature
ggplot(the_data %>% filter(complete.cases(.))) +
  geom_bar(aes(Developer), col="tomato") + ggtitle("Histogram of Developer Feature")
```




```
# so we also remove Name and Developer features
the_data <- the_data %>%
  select(-c("NA_Sales", "EU_Sales", "JP_Sales", "Other_Sales",
            "Name", "Publisher", "Developer")) %>%
  filter(complete.cases(.))
# data dimension
the_data %>% dim
```

```
## [1] 6826    9
```

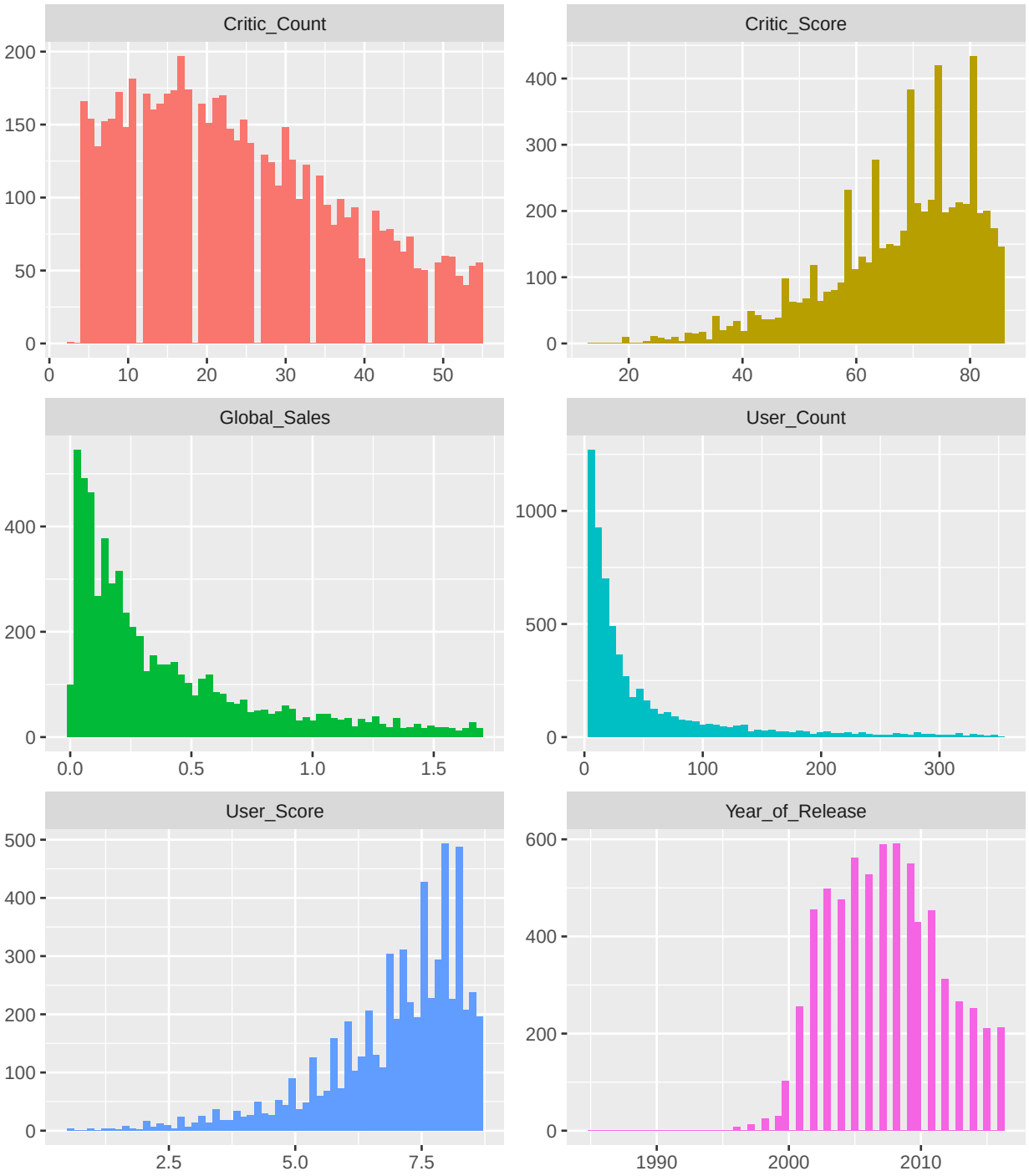
Summary and Data Visualisation

```
# summary
summary(the_data)
```

```
##    Platform         Year_of_Release    Genre            Global_Sales
##  Length:6826        Min.   :1985    Length:6826        Min.   : 0.0100
##  Class :character   1st Qu.:2004    Class :character   1st Qu.: 0.1100
##  Mode  :character   Median :2007    Mode  :character   Median : 0.2900
##                     Mean   :2007                       Mean   : 0.7775
##                     3rd Qu.:2011                       3rd Qu.: 0.7500
##                     Max.   :2016                       Max.   :82.5300
##   Critic_Score    Critic_Count      User_Score      User_Count
##  Min.   :13.00   Min.   :  3.00   Min.   :0.500   Min.   :   4.0
##  1st Qu.:62.00   1st Qu.: 14.00   1st Qu.:6.500   1st Qu.:  11.0
```

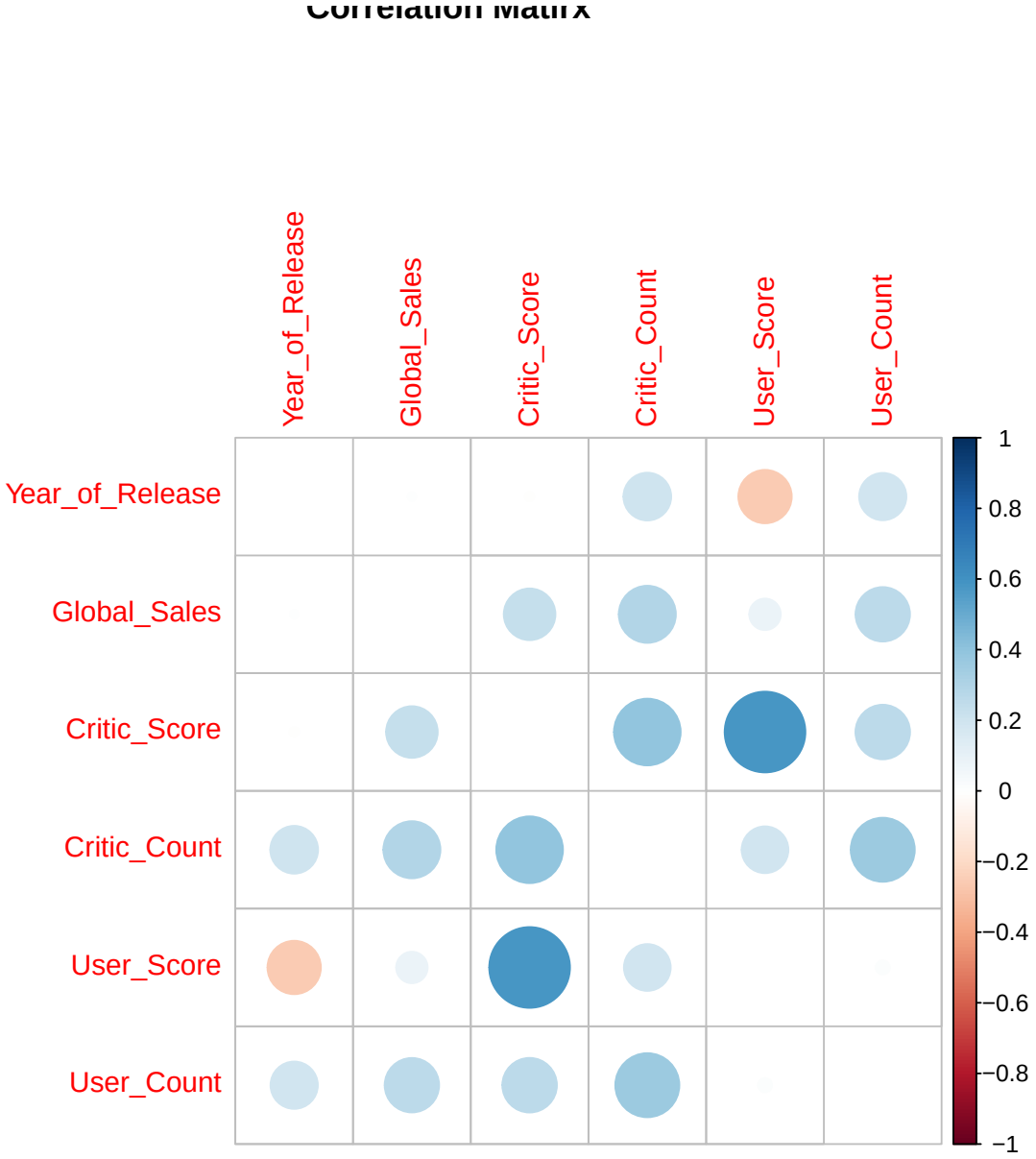
```
##  Median :72.00   Median : 25.00   Median :7.500   Median :   27.0
##  Mean   :70.27   Mean   : 28.93   Mean   :7.185   Mean   :  174.7
##  3rd Qu.:80.00   3rd Qu.: 39.00   3rd Qu.:8.200   3rd Qu.:   89.0
##  Max.   :98.00   Max.   :113.00   Max.   :9.600   Max.   :10665.0
##     Rating
##  Length:6826
##  Class :character
##  Mode  :character
##
##
##
```

```
# histrograms of numeric variables
df <- the_data %>% select_if(is.numeric) %>% gather
df2 <- df %>% group_by(key) %>% filter(key=="Year_of_Release" | value < quantile(value,0.9))
ggplot(df2, aes(value, fill=key)) +
  facet_wrap(~key, scales="free", ncol=2) +
  geom_histogram(bins=60) + theme(legend.position='none') + ylab(NULL) + xlab(NULL)
```




```
# histrograms of character variables
df <- the_data %>% select_if(is.character) %>% gather
ggplot(df, aes(value, fill=key)) +
  facet_wrap(~key, scales="free", ncol=1) +
  geom_bar() + theme(legend.position='none') + ylab(NULL) + xlab(NULL)
```


```
# plot correlations
corrplot(cor(the_data %>% select_if(is.numeric)), diag=FALSE, title="Correlation Matirx")
```



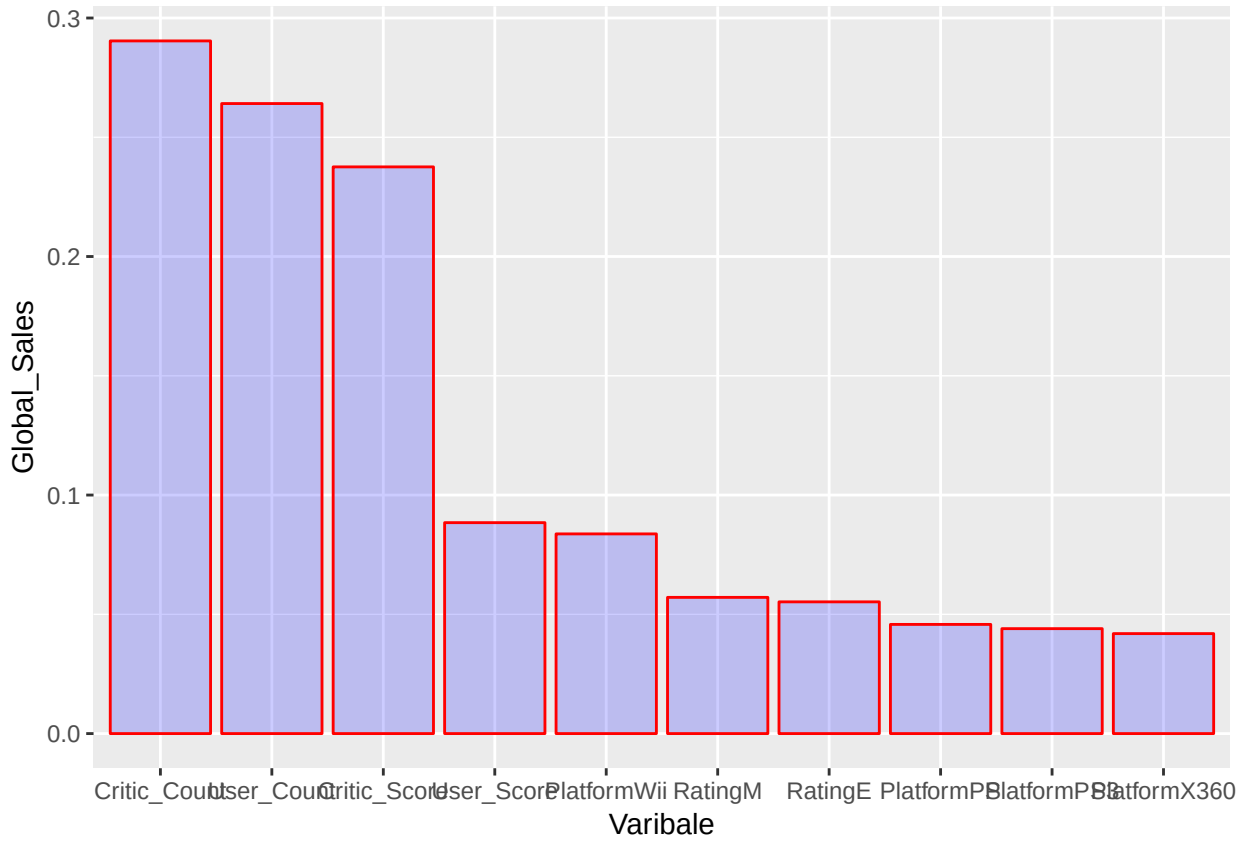

```
# liberate memory
remove(df, df2)
```

Feature Selection and Feature Engineering

Feature Selection means that we remove irrelevant variables that only add noise.

Feature Engineering means that we add new variables

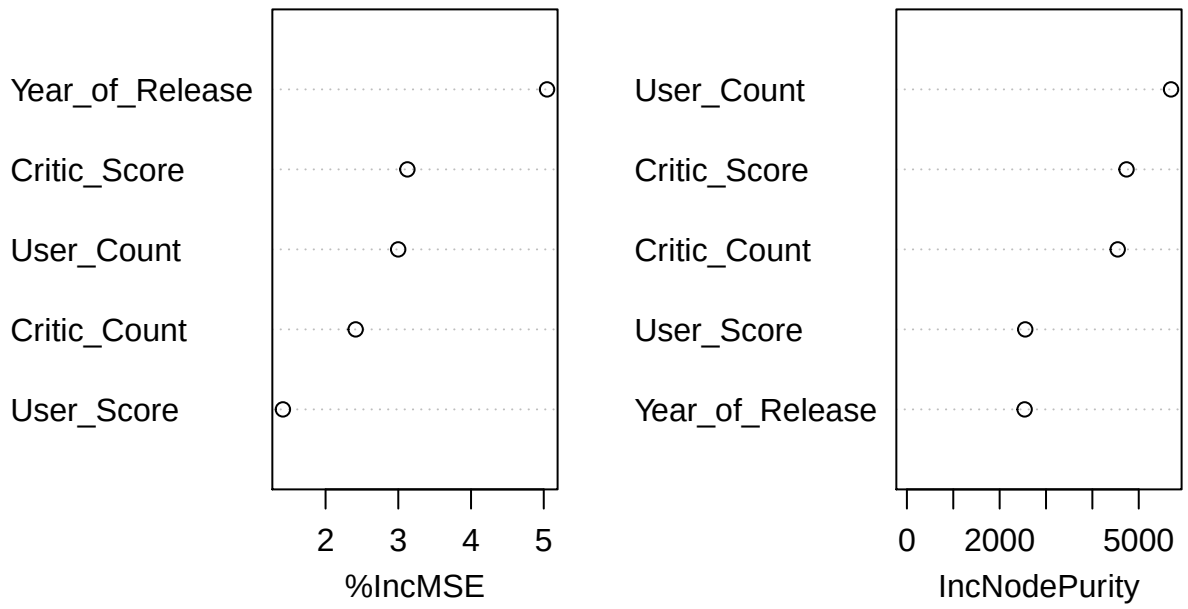
```
# categorical variable to dummy variables
dummy <- dummyVars(~ ., data = the_data, fullRank = TRUE)
dummy_data <- predict(dummy, the_data) %>% as_tibble
names(dummy_data) <- gsub("\\+|-", "", names(dummy_data))

# variables that are correlated to Global_Sales the most
as_tibble(cor(dummy_data)) %>% select("Global_Sales") %>%
  mutate(Varibale=names(dummy_data)) %>%
  filter(Global_Sales > 0.04 & Varibale!="Global_Sales") %>% arrange(desc(Global_Sales)) %>%
  mutate(Varibale=factor(Varibale, levels=Varibale)) %>%
  ggplot() + geom_col(aes(Varibale, Global_Sales), fill=I("blue"), col=I("red"), alpha=I(.2))
```



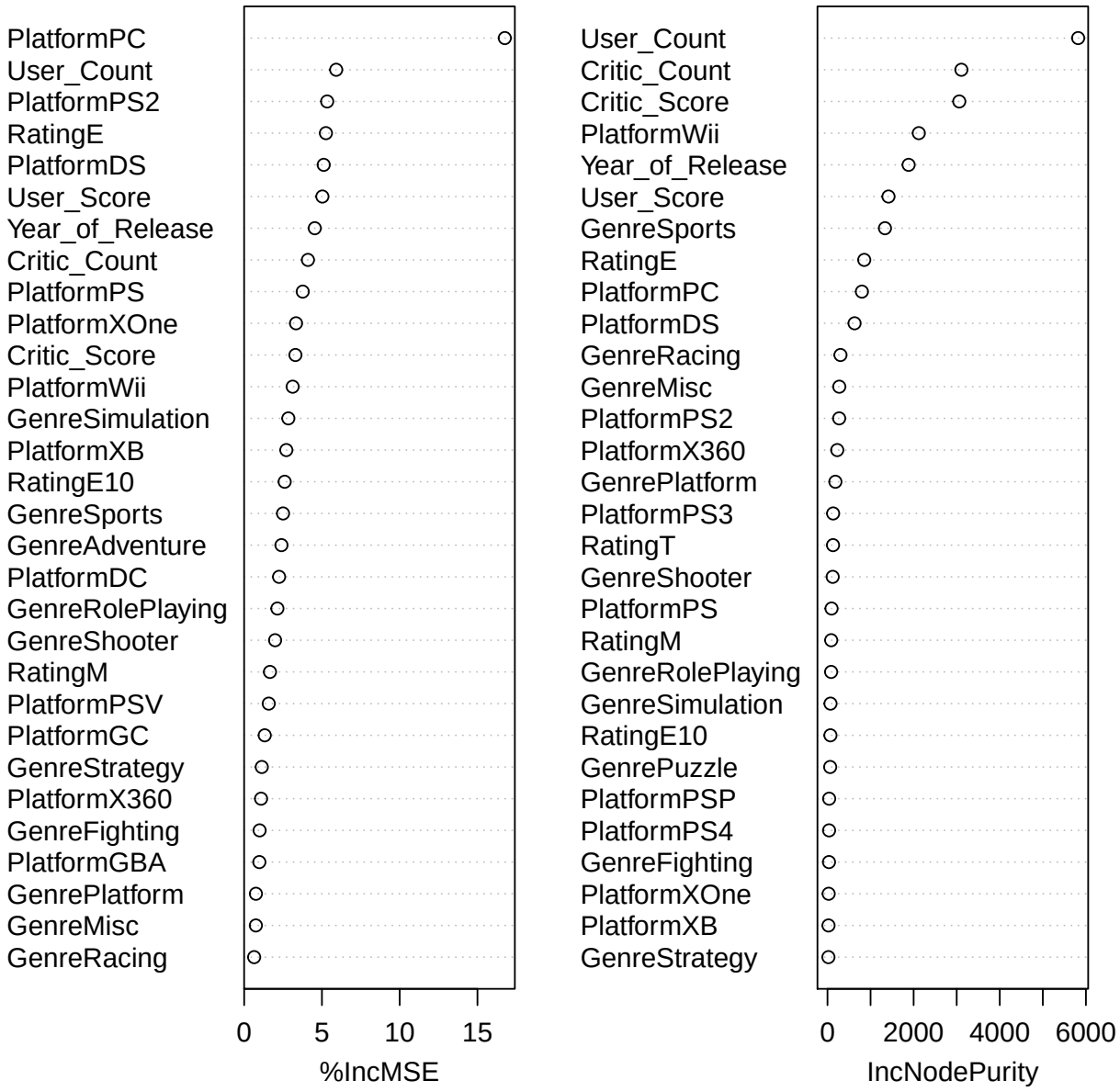
Feature Importances

It shows which variables play the in predicting the Global Sales as if we use Random Forest Model

```
# numeric features
model_RandomForest <- randomForest(Global_Sales ~ .,
                                   data  = the_data %>% select_if(is.numeric),
                                   ntree = 100, keep.forest = FALSE, importance = TRUE)
varImpPlot(model_RandomForest, main="Feature Importances of Numeric Features")
```




```r
set.seed(1)
# all features
model_RandomForest <- randomForest(Global_Sales~ .,
                                   data = dummy_data,
                                   ntree = 100, keep.forest = FALSE, importance = TRUE)
varImpPlot(model_RandomForest, main="Feature Importance of All Features")
```



After looking at the correlation matrix and the Feature Importances, we can say what features are little relevant to the Global Sales and probably only add noise.

```
# 15 the most important features accroding to two metrics
IncMSE_15 <- sort(model_RandomForest$importance[,1], decreasing=TRUE)[1:15] %>% names
IncNod_15 <- sort(model_RandomForest$importance[,2], decreasing=TRUE)[1:15] %>% names
# how many features are in both list?
sum(IncMSE_15 %in% IncNod_15)
```

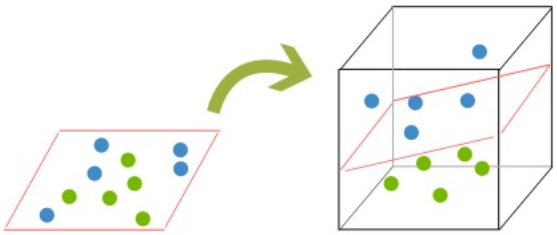


Figure 3: SVM for a Classification Problem

```
## [1] 13
```

```
# importance lists are almost the same
dummy_data_short <- dummy_data %>% select(Global_Sales, IncMSE_15)
```

Description of models

Little description of used model

Linear Regression Model - 'lm'

It is a statistical model where we suppose that the label of an observation can be expressed as follow

$$y_i = \sum_{j=1..m} \beta_j x_{ij} + \epsilon_i = \beta^t x_i + \epsilon_i = g_{lm}(x_i) + \epsilon_i$$

Where m is the number of features, in other words, the length of vector x_i. $\{e_i\}_{i=1..n}$ are iid of $N(0, \sigma_\epsilon)$

Generalized Linear Model - 'glmnet'

The model is defined as a function of the linear regression model

$$E(y_i) = g_{glm}(x_i) = f^{-1}\left(\beta^t x_i\right)$$

for a function f of a certain form. $E(y)$ is expected value of y

Support Vector Machine - 'svmRadial'

This model is similar to the linear one, but the difference is that it works by increasing the dimension by one. In the case of binary classification with two features, the linear model divides the space into two labels by a straight line, while the SVM divides it by a plane, as on the figure below.

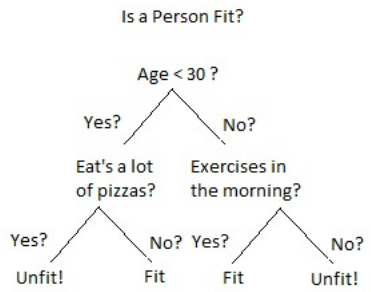


Figure 4: Decision Tree for a Classification Problem

Decision Tree - 'rpart'

Decision Tree is a method that looks at features one by one and tries to deduce the result by binary decisions as on the following figure.

Random Forest - 'rf'

Random Forest prediction is given as an average prediction (or majority vote for classification) of a number of Decision Trees. Each tree is created on random subsets of the initial data.

Extreme Gradient Boosting (XGBoost) - 'xgbTree'

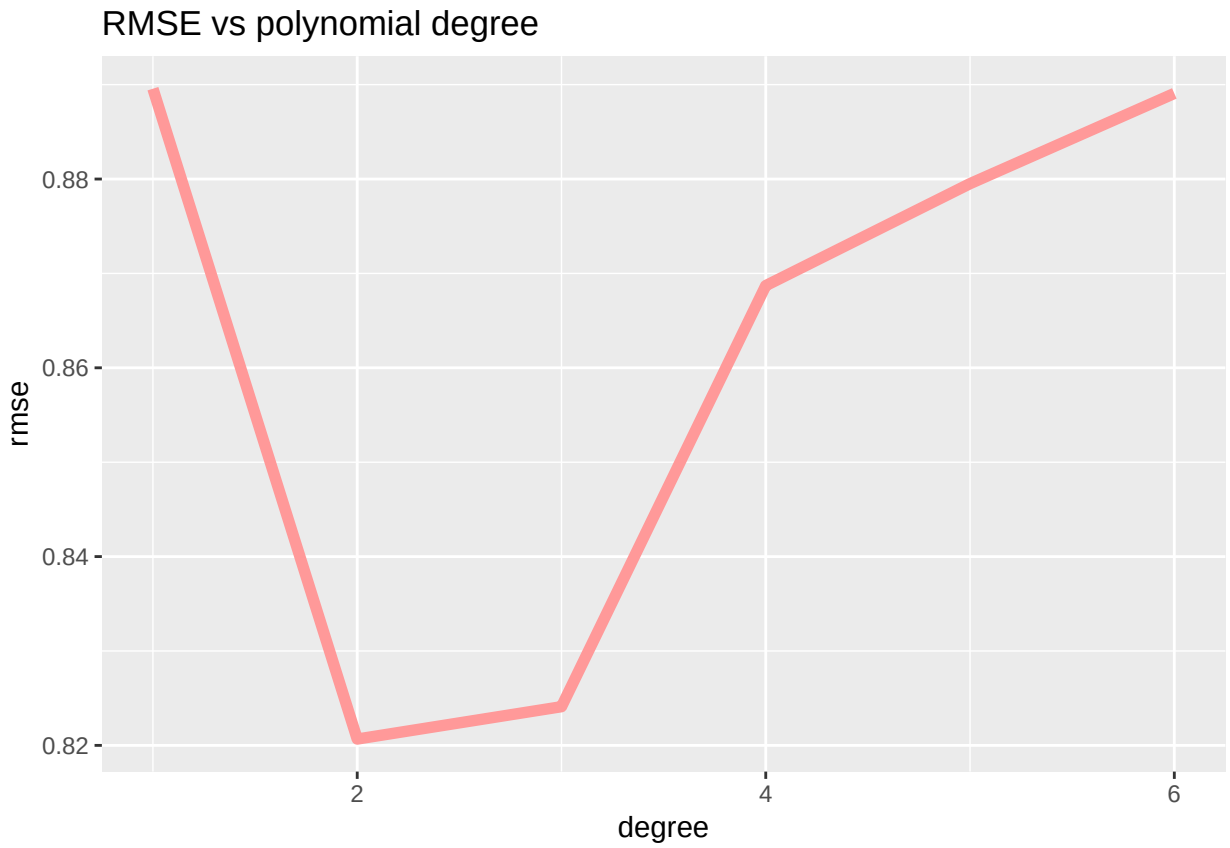
As Random Forest, XGBoost also consists of a set of Decision Trees. But trees are created one by one in a way to minimise the error of the aggregation of all already existing trees.

An example of overfitting

If we try to predict Global Sales by a polynomial function of the Critic Count, we can overfit the system if we choose too high polynomial degree.

```
# we choose a small part of data
N <- 200
set.seed(1)
df <- the_data[sample(1:nrow(the_data), N),]

# train and validation
valid_df <- df[1:floor(N/3),]
train_df <- df[ceiling(N/3):N,]
RMSEs <- tibble()
for (i in 1:6){
  suppressWarnings(
    my_lm <- lm(Global_Sales ~ poly(Critic_Count, i, raw=TRUE), train_df)
  )
  suppressWarnings(
    prediction <- predict(my_lm, valid_df)
  )
  RMSEs <- bind_rows(RMSEs,
```

```
                    tibble("degree" = i, "rmse" = RMSE(valid_df$Global_Sales, prediction)))
}
# plot RMSE as a function of degree of polynomial
ggplot(RMSEs, aes(degree, rmse)) + geom_line(col="#FF9999", size=2) +
  ggtitle("RMSE vs polynomial degree")
```



We see that the optimal degree of polynomial function is 2 and not higher.

Train and Prediction

A function that does the training, cross validation, parameter tuning and test on the validation set for a list of models.

Training, cross validation and parameter tuning are carried out by 'train' function from 'caret' package.

The cross validation is specified in 'train_control' variable.

The parameter tuning is carried out by default with a default grid of parameters (which depends on the model).

Although we can customise the grid if we feel that the problem is not a standard one

```
my_ML <- function(train_data, valid_data, models, rmse_path, custom_tuning=FALSE){

  # # set the random generator state
  # set.seed(1)
```

```
  # if custom_tuning=FALSE, we use a default grid of parameters for the tuning
  if(!custom_tuning)
    grids <- NULL

  # data frame to stock valdiation rmse of models
  df_rmse <- tibble()
  # list to stock trained models (its coefficients and train errors)
  list_models <- list()

  # loop by models
  for (model_name in models){
    # print(model_name)
    # time measure
    start.time <- Sys.time()

    # train the model on the train set
    suppressWarnings(
      my_model <- train(Global_Sales ~ .,
                        method       = model_name,
                        data         = train_data,
                        trControl    = train_control,
                        tuneGrid     = grids[[model_name]])
    )

    # predict on the validation set
    suppressWarnings(
      prediction <- predict(my_model, valid_data)
    )

    # calculate prediction RMSE
    df_rmse <- df_rmse %>% bind_rows(list("model"= model_name,
                                          "rmse" = RMSE(valid_data$Global_Sales, prediction),
                                          "time" = signif(Sys.time() - start.time, prec)))
    # add the model to the list of models
    list_models[[model_name]] <- my_model
  }
  # save errors on the disk
  df_rmse %>% write_csv(rmse_path)

  ## train error analysis
  dotplot(resamples(list_models))

  # return valdiation rmse
  return(df_rmse)
}
```

Create train and validation set (10% of the data)

```
set.seed(1)
# separate data into train and valdiation sets randomly, 90 and 10%
valid_index <- createDataPartition(y = the_data$Global_Sales, p = 0.1, list = FALSE)
train_data  <- dummy_data[-valid_index,]
valid_data  <- dummy_data[valid_index,]
# Short set of features
```

```
train_data_short  <- dummy_data_short[-valid_index,]
valid_data_short  <- dummy_data_short[valid_index,]
```

Settings

```
# list of models
models <- c("lm", "glmnet", "svmRadial", "rpart", "rf", "xgbTree")

# range of parameters for the customised parameters tuning
grids                 <- list()
grids[["lm"]]         <- NULL
grids[["glmnet"]]     <- expand.grid(alpha     = c(0, .5, 1),
                                     lambda    = c(.1, 1))
grids[["svmRadial"]] <- expand.grid(sigma     = c(.05, .045),
                                     C         = c(2.2))
grids[["rpart"]]      <- expand.grid(cp        = c(.05, .2))
grids[["rf"]]         <- expand.grid(.mtry     = c(1, 5, 15))
grids[["xgbTree"]]    <- expand.grid(nrounds   = c(10, 100),
                                     max_depth = c(5, 10, 15),
                                     eta       = c(.4, .1, .01),
                                     gamma     = c(0, 1, 3),
                                     subsample = c(.8, 1),
                                     colsample_bytree = 1,
                                     min_child_weight = 1)

# inialise the Cross-Valdiation (10-Cross-Section Validation)
train_control  <- trainControl(method = "cv", number = 10)
```

All features

Default Parameters Tuning

```
rmse_path <- "output/rmse.csv"
results <- my_ML(train_data, valid_data, models, rmse_path, custom_tuning=FALSE)
kable(results %>% mutate(rmse=signif(rmse,prec)),
      "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

model	rmse	time
lm	1.0340	1.626 secs
glmnet	1.0320	1.787 secs
svmRadial	0.9540	163.620 secs
rpart	1.1280	1.792 secs
rf	0.9187	1674.000 secs
xgbTree	1.0290	136.320 secs

Customised Parameters Tuning

```
rmse_path_t <- "output/rmse_tuning.csv"
results <- my_ML(train_data, valid_data, models, rmse_path_t, custom_tuning=TRUE)
```

```
kable(results %>% mutate(rmse=signif(rmse,prec)),
      "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

model	rmse	time
lm	1.0340	0.9111 secs
glmnet	1.0300	1.2380 secs
svmRadial	0.9344	109.6800 secs
rpart	1.0740	0.9930 secs
rf	0.9304	606.6000 secs
xgbTree	0.9113	968.4000 secs

Short set of features

Default Parameters Tuning

```
rmse_path_s <- "output/rmse_short.csv"
results <- my_ML(train_data_short, valid_data_short, models, rmse_path_s, custom_tuning=FALSE)
kable(results %>% mutate(rmse=signif(rmse,prec)),
      "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

model	rmse	time
lm	1.0460	0.6527 secs
glmnet	1.0420	0.9154 secs
svmRadial	0.9520	77.5200 secs
rpart	1.1280	1.0370 secs
rf	0.9166	759.6000 secs
xgbTree	0.9499	101.8200 secs

Customised Parameters Tuning

```
rmse_path_st <- "output/rmse_short_tuning.csv"
results <- my_ML(train_data_short, valid_data_short, models, rmse_path_st, custom_tuning=TRUE)
kable(results %>% mutate(rmse=signif(rmse,prec)),
      "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

model	rmse	time
lm	1.0460	0.6352 secs
glmnet	1.0410	0.9749 secs
svmRadial	0.9397	56.5200 secs
rpart	1.0740	0.7268 secs
rf	0.9262	639.0000 secs
xgbTree	0.8819	587.7000 secs

Results

```
results <- suppressMessages(
  read_csv("output/rmse.csv") %>%
    bind_rows(
      read_csv("output/rmse_tuning.csv") %>% mutate(model = paste0(model,"_t"))
    ) %>%
    bind_rows(
      read_csv("output/rmse_short.csv") %>% mutate(model = paste0(model,"_s"))
    ) %>%
    bind_rows(
      read_csv("output/rmse_short_tuning.csv") %>% mutate(model = paste0(model,"_st"))
    ) %>%
    arrange(rmse) %>%
    mutate("rmse" = signif(rmse,prec))
  )

# all rmse
kable(results, "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

model	rmse	time
xgbTree_st	0.8819	587.7000
xgbTree_t	0.9113	968.4000
rf_s	0.9166	759.6000
rf	0.9187	1674.0000
rf_st	0.9262	639.0000
rf_t	0.9304	606.6000
svmRadial_t	0.9344	109.6800
svmRadial_st	0.9397	56.5200
xgbTree_s	0.9499	101.8200
svmRadial_s	0.9520	77.5200
svmRadial	0.9540	163.6200
xgbTree	1.0290	136.3200
glmnet_t	1.0300	1.2380
glmnet	1.0320	1.7870
lm	1.0340	1.6260
lm_t	1.0340	0.9111
glmnet_st	1.0410	0.9749
glmnet_s	1.0420	0.9154
lm_s	1.0460	0.6527
lm_st	1.0460	0.6352
rpart_st	1.0740	0.7268
rpart_t	1.0740	0.9930
rpart_s	1.1280	1.0370
rpart	1.1280	1.7920

```
# the smallest error
kable(results %>% slice(which.min(rmse)),
      "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

model	rmse	time
xgbTree_st	0.8819	587.7

Conclusion

As we see, models with the lowest RMSE are Random Forest and XGBoost which are both Ensemble Learning Methods and have a high degree of freedom (so they are able to fit a complex data). Ensemble Learning Method is a model that being a strong learner consists of an aggregation of a big number of weak learners, in the case of Random Forest and XGBoost, Decision Trees. The difference in models is the way of aggregating.

```
# best models
kable(results %>% slice(1:5),
      "latex", booktabs = TRUE) %>% kable_styling(latex_options = "striped")
```

model	rmse	time
xgbTree_st	0.8819	587.7
xgbTree_t	0.9113	968.4
rf_s	0.9166	759.6
rf	0.9187	1674.0
rf_st	0.9262	639.0

The best model is XGBoost trained on reduced set of features with customised tuning grid. Reduced set of features allows to exclude the noises by taking into account only the pertinent information. Customising the parameter tuning permits to exploit all the capacities of the model. It worth mentioning that XGBoost is one of the most used models in Kaggle competitions.

As second model comes Random Forest on reduced set of features, it performs a bit badlier with customised tuning, but in any case better than XGBoost with default tuning. In terms of computational time, the both models are quite the same.