SoftMax Weights for a Weighted Least Squares Linear Model

SCR Written Exam 2

R-team

January 08, 2019

Exam Instructions

The exam consists of 4 tasks for a total of 100 points. Behind each (sub)task you will see the number of points that can be earned. The grading model can always be modified as long as the modification(s) will turn out to be advantagesous to all students.

During this exams, only use functions from the 1) libraries that automatically load with `RStudio`; 2) the libraries in `tidyverse`, and 3) the library `AmesHousing`. Furthermore, load the variables that are stored in `0_data/Exam_answer_variables.RData`, you will need them.

Your style of coding can affect your final exam grade. Even though a correct answer is preferred above beautiful code, it may cost you points when very complicated code is provided as an answer at the place where simple `R` functions would suffice. Moreover, adhere to a consistent and neat programming style, e.g. as those in

1. https://google.github.io/styleguide/Rguide.xml,
2. or http://style.tidyverse.org.

Write down your exam answers in a neat `.Rmd` script file **that does not contain the text of the (sub)tasks themselves**.

We should be able to knit your `.Rmd` script into a nice and readabile `.pdf` report on our computer. Note that it will cost you points when, during the knitting process, we run into any errors that can be traced back to your .Rmd source file. Assume that your `.Rmd` file is knitted in the exact same directory as the one in which you can find the `.Rmd` file of this exam. E.g. code like `load("0_data/Exam_answer_variables.RData")` evaluates correctly on our computer.

Only upload your `Lastname_ULCN.Rmd` file to Blackboard, and do so **before** 17.05 hours.

- Go to Statistical Computing with R –> Exams & Assignments –> Exam January 08, 2019
- Each extra minute counted from 17:05 hours will cost you 10 out of 100 points: when you submit your exam at 17:10 hours spot on, it means you already lost 50 points. Exams submitted after 17:10 hours will not be graded.

Last, you are allowed to consult the internet and all files on your computer, or your physical prepared written notes. However, keep in mind that any form of communication with others is not allowed, and is considered **FRAUD**.

Success!

the R-team.

1. The data (20 points)

Install and load the `AmesHousing` library with the code:

```
install.packages("AmesHousing")
library("AmesHousing")
```

Having loaded the `AmesHousing` library, you can acces the `ames_raw` data set which we will need in this exam. Information about the data can be found via:

- the code `help(package = "AmesHousing")`;
- https://cran.r-project.org/web/packages/AmesHousing/AmesHousing.pdf;
- the original study from De Cock (2011), see http://jse.amstat.org/v19n3/decock.pdf.

In this exam we will use the linear model defined by the following formula:

```
price_model <- formula(
  SalePrice ~ `Lot Area` + `Total Bsmt SF` +
    `Gr Liv Area` + `Garage Cars` + `Fireplaces`
)
```

Here, the selling price of a house (in dollars) is regressed on the lot size (in square feet); the total square feet of the basement area; the above ground living area in square feet; size of the garage expressed in car capacity; and, last, the number of fireplaces.

1.1. Preparing The Data (12 points)

In this subtask we ask you to load and prepare the data set that we will be using in this exam.

Write a chunk of code with which you create a filtered data set that is similar to `ames_prep` data set. The `ames_prep` data set is stored in `Exam_answer_variables.RData`, and this `.Rdata` file is located in the `0_data` folder.

Create with your own code the `ames_prep` data set from the `ames_raw` data set. This latter data set becomes available when loading the `AmesHousing` package. To create `ames_prep`,

1. Remove the houses of more than 4000 square feet (`Gr Liv Area`) from the data set;
2. Only keep the columns that are used in the linear model `price_model`;
3. Remove the houses that have missing values on at least one of the variables in the linear model, i.e. the variable `price_model`.

Answer:

```
load("0_data/Exam_answer_variables.RData")
ames_prep <- ames_raw %>%
  filter(`Gr Liv Area` < 4000) %>%
  select(c(
    "SalePrice", "Lot Area", "Total Bsmt SF",
    "Gr Liv Area", "Garage Cars", "Fireplaces"
  )) %>%
  na.omit()
```

Or using base `R`:

```
index <-ames_raw$`Gr Liv Area` < 4000
ames_prep_2 <- ames_raw[index,c(
```

```
    "SalePrice", "Lot Area", "Total Bsmt SF",
    "Gr Liv Area", "Garage Cars", "Fireplaces")]
ames_prep_2 <- na.omit(ames_prep_2)

all.equal(ames_prep_2,ames_prep)
```

```
## [1] TRUE
```

1.2. A bit of Exploration (4 points)

In this exam we will later be using selling prices (`SalePrice`) as the exponents of the natural number e. However, some tricks may be needed in `R` to get decent results.

When we use the selling prices as exponents of e, does R output any finite values ($< \infty$)?

Answer:

```
any(exp(ames_prep$SalePrice) < Inf)
```

```
## [1] FALSE
```

Nope...

1.3. Maximum and Average Prices (4 points)

What are the maximum and the average prices for which a house got sold in the prepared data?

Answer:

```
summary(ames_prep$SalePrice)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   12789  129500  160000  180456  213500  625000
```

2. From Exponential Weights to a Soft Maximum (20)

In subtask 1.3 you have computed a 'hard' maximum. There is also a way to compute a 'soft' maximum which is nowadays frequently used in statistical / machine learning practices.

Let us denote the selling price of a house i by y_i. Let $v(\lambda)_i$ be a weight function taking on values in the interval $[0, 1]$, and define it as

$$v(\lambda)_i = \frac{e^{y_i/\lambda}}{\sum_{i'=1}^{N} e^{y_{i'}/\lambda}}, \tag{1}$$

where $0 < \lambda < \infty$. We refer to these weights as the exponential weights.

Then, when $\lambda \to 0$ we can use the exponential weights to find the maximum, since

$$\lim_{\lambda \to 0} \sum_{i=1}^{N} v(\lambda)_i y_i = \max_i(y_i).$$

In a comparable way we can find the mean of y when $\lambda \to \infty$:

$$\lim_{\lambda \to infty} \sum_{i=1}^{N} v(\lambda)_i y_i = \bar{y}.$$

2.1 (5 points)

On Wikipedia some `R` code is given to compute the exponential weights for $\lambda = 1$, i.e.

```
v <- function(y) {
  exp(y) / sum(exp(y))
}
```

Modify this function by including the argument λ such that it represents equation 1, and also give your coded function another name.

When applying your modified function on the prices (`SalePrice`) in the data set, show that for $\lambda = 1$ you will obtain `NaN`'s only, and show that you obtain informative values for $\lambda = 1,000$.

Answer:

```
BadlyObtainV <- function(y, lambda) {
  exp(y / lambda) / sum(exp(y / lambda))
}
```

```
y <- ames_prep$SalePrice
v_1e0 <- BadlyObtainV(y, 1)
all(is.nan(v_1e0))
```

```
## [1] TRUE
```

```
v_1e3 <- BadlyObtainV(y, 1e3)
summary(v_1e3)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000000 0.0000000 0.0000000 0.0003421 0.0000000 0.9999527
```

2.2 (10 points)

With a different programming strategy, it is possible with `R` to obtain informative values for the eponential weights of the selling prices for values of λ in the interval $[0, 100]$.

Note that, $v(\lambda)_i$ can also be expressed as

$$v(\lambda)_i = \frac{e^{y_i/\lambda - C}}{\sum_{i'=1}^{N} e^{y_{i'}/\lambda - C}}, \tag{2}$$

where C is a constant ($C \in \mathbb{R}$).

Modify your function for the exponential weights again, but now use equation (2), and define C as:

$$C = \max_i \left(y_i/\lambda \right).$$

Based on this modification of your function of the exponential weights, give a summary of the exponential weights of the selling prices (`SalePrice`) for $\lambda = 1$.

Answer:

The function to obtain the weights:

```
GetWeights <- function(y, lmbd = 100, na.rm = TRUE) {
 if (na.rm == TRUE) na.omit(y)
 mx_num <- max(y / lmbd)
 num <- exp(y / lmbd - mx_num)
 v <- num / sum(num)
 return(v)
}
```

```
v_1e0 <- GetWeights(y, lmbd = 1)
summary(v_1e0)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000000 0.0000000 0.0000000 0.0003421 0.0000000 1.0000000
```

The function to obtain a soft maximum

```
FindSoftMax <- function(lmbd, y, na.rm = TRUE) {
  v <- GetWeights(y, lmbd, na.rm)
  if (na.rm == TRUE) { y <- na.omit(y) }
  return(sum(v * y))
}
```

2.3 (5 points)

For this exercise, use the variable `v_from_0_to_1e6` stored in `Exam_answer_variables.RData`. The variable `v_from_0_to_1e6` contains in each column the exponential weights of the selling prices of the houses ('SalePrice'). The value for λ that is used for the exponential weights is stored in the name of the corresponding column.

Create a visualization (line plot) for

$$SoftMax(\lambda) = \sum_{i=1}^{N} v(\lambda)_i y_i, \tag{3}$$

against $\lambda \in [0, 10^6]$ on the horizontal axis. Then, add two horizontal lines to the plot:
1. a red dashed line to indicate the maximum value of the housing prices.
2. a blue dashed line to indicate the mean value of the housing prices.

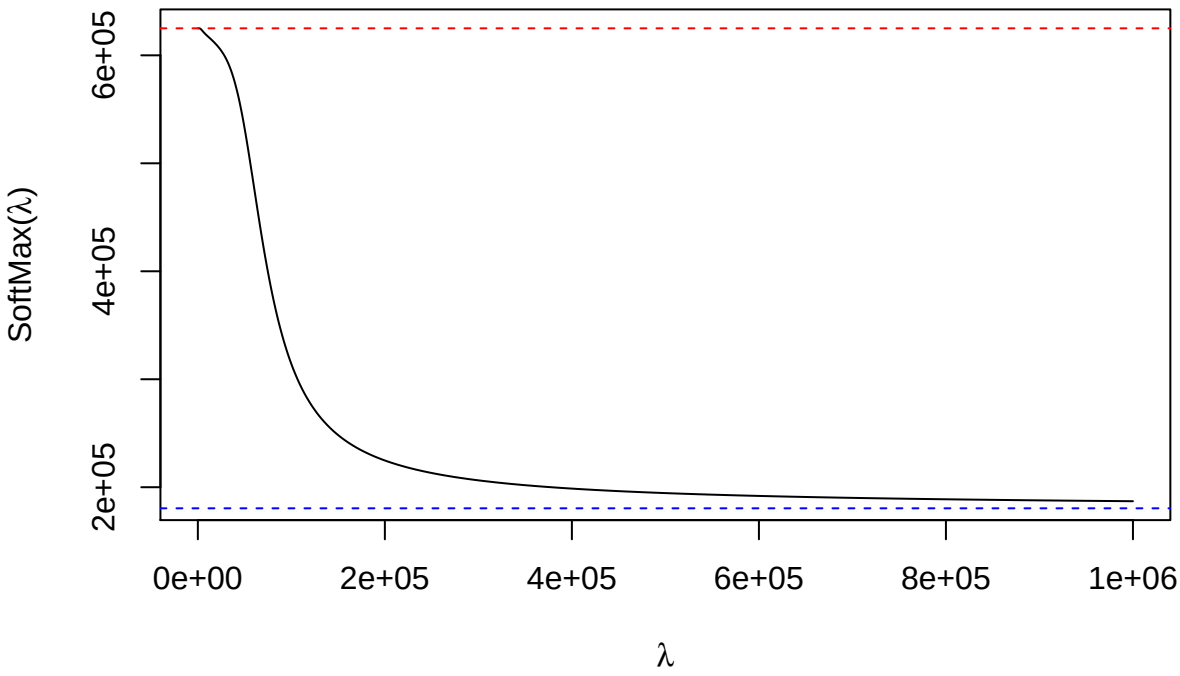
Answer:

Obtain the softmax values:

```
lmbds <- as.numeric(names(v_from_0_to_1e6))
SoftMxs <- numeric(length(lmbds))
for (i in 1:length(lmbds)) {
  SoftMxs[i] <- FindSoftMax(lmbd = lmbds[i], y = y)
}
```

The visualization:

```
plot(y = SoftMxs, x = lmbds,
     xlab = expression(lambda),
     ylab = bquote("SoftMax(" * lambda * ")"),
     type = "l"#", pch = 20
)
abline(h = max(y), col = 'red', lty = 2)
abline(h = mean(y), col = 'blue', lty = 2)
```



The higher λ, the softer the maximum..

3. Weighted Least Squares Linear Regression (30 points)

A linear model for the housing prices (`SalePrice`) could be fitted with linear regression, using `lm()` and the formula:

```
price_model <- formula(
  SalePrice ~ `Lot Area` + `Total Bsmt SF` +
    `Gr Liv Area` + `Garage Cars` + Fireplaces
)
```

This specific linear regression for our data can be conducted based on ordinary least squares (OLS) or weighted least squares (WLS). The results are shown in Figure 1

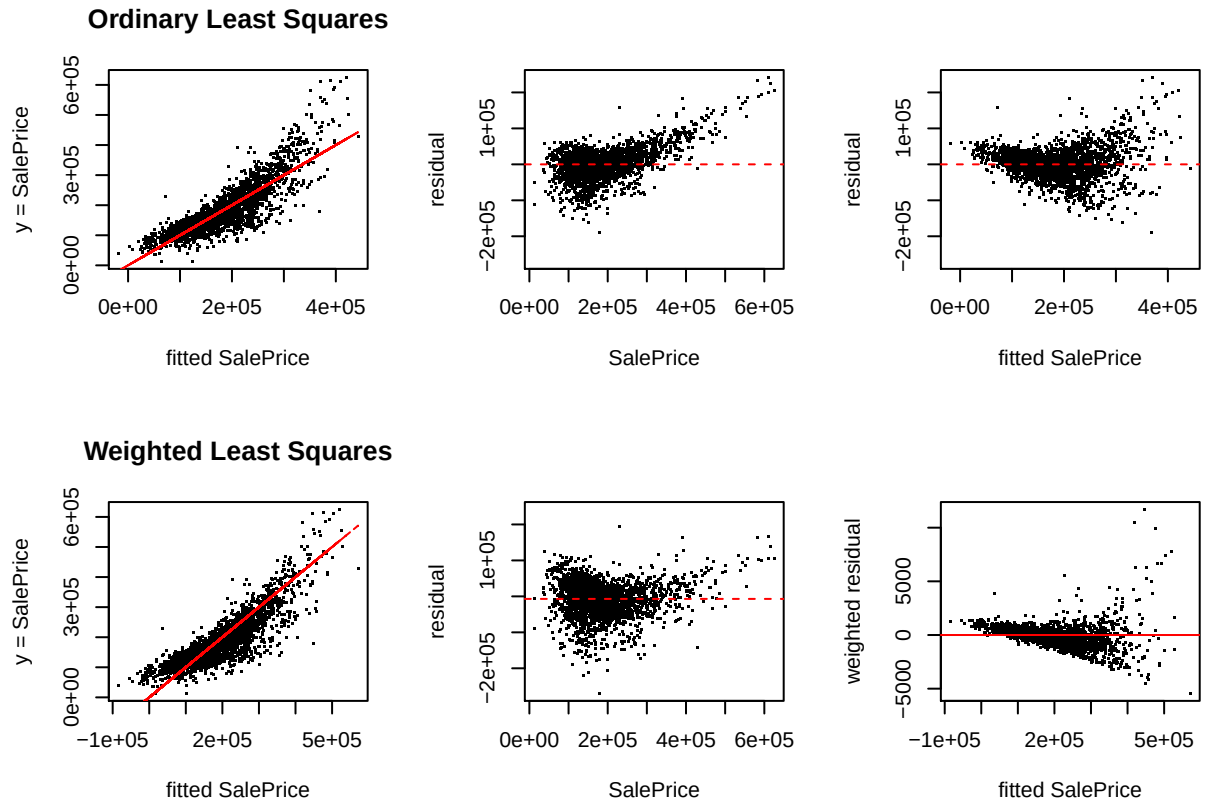


Figure 1: The top row panels represent the results for OLS linear regression, and the bottom row panels represent the results for WLS linear regression. The solid red represent fitted values ($= \widehat{y}$) of the housing prices (y = 'SalePrice'). The dashed red lines represent the average of the (weighted) residuals ($= y - \widehat{y}$).

In the case of the results for the OLS linear model, we see that the residuals (= SalePrice - Fitted SalePrice) seem to become higher when the prices (`SalePrice`) become higher. A too strong correlation between the response (`SalePrice`) and the residuals is undesirable, and is difficult to get rid off due to regulations provided by a protocol of a certain mortgage provider. In this particular protocol it states that

1. no transformation of the outcome variable is allowed;
2. nor a change in the formula of the linear model; and
3. a simple linear regression analysis has to be used (e.g. using `lm()` only).

Suppose a correct prediction of the higher prices is more lucrative (as compared to the lower prices), then a solution is needed for this specific linear model based on OLS. A solution could be to fit the linear model based on weighted least squares (LM-WLS). With this specific LM-WLS analysis you may find that you can assure that the residuals are not (linearly) correlated with the actual housing prices, see the middle panel on

the bottom row of Figure 1.

Note that the linear model based on weighted least squares can still be fitted with the function `lm()`. The only difference now is that we use the argument `weights` in the function `lm()`. Thus, we still comply with the protocol of the certain mortgage provider.

3.1 (5 points)

Fit and store the linear model with the formula `price_model` based on the ordinary least squares loss function on the prepared Ames data.

Show that your object is very close or equal to linear model we used: `lm_ols` (see `Exam_answer_variables.RData`).

Answer:

The linear model based on ordinary least squares

```
lm_ols <- lm(
  SalePrice ~ `Lot Area` + `Total Bsmt SF` + `Gr Liv Area` + `Garage Cars` + Fireplaces,
  data = ames_prep
)
summary(lm_ols)
```

```
##
## Call:
## lm(formula = SalePrice ~ `Lot Area` + `Total Bsmt SF` + `Gr Liv Area` +
##     `Garage Cars` + Fireplaces, data = ames_prep)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -188186  -19716     759   19882  241833
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -4.143e+04  2.718e+03 -15.240   <2e-16 ***
## `Lot Area`       2.060e-01  1.004e-01   2.052   0.0403 *
## `Total Bsmt SF`  6.410e+01  2.055e+00  31.200   <2e-16 ***
## `Gr Liv Area`    6.508e+01  1.919e+00  33.923   <2e-16 ***
## `Garage Cars`    2.759e+04  1.185e+03  23.279   <2e-16 ***
## Fireplaces       1.131e+04  1.322e+03   8.556   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 40100 on 2917 degrees of freedom
## Multiple R-squared:  0.7399, Adjusted R-squared:  0.7394
## F-statistic:  1659 on 5 and 2917 DF,  p-value: < 2.2e-16
```

3.2 Recover the value for λ of the WLS model (15 points)

The linear model that we fitted based on WLS is stored in `lm_wls` (see `Exam_answer_variables.RData`). It was fitted in the same way as the linear model based on OLS, but for the `weights` argument we used the exponential weights of `SalePrice` computed with a function based on equation (2). Could you show with `R` code the value of λ that we used?

The value for λ should be any of the values in

```
as.numeric(names(v_from_0_to_1e6))
```

Answer:

We could have found $\lambda = 150000$ using the following code:

```
v_weights <- weights(lm_wls)
ids_v <- sapply(
  X = v_from_0_to_1e6,
  FUN = function(x) {
    all(v_weights == x)
  }
)
cat("lambda = ", names(v_from_0_to_1e6)[which(ids_v)])
```

```
## lambda =  150000
```

3.3 (10 points)

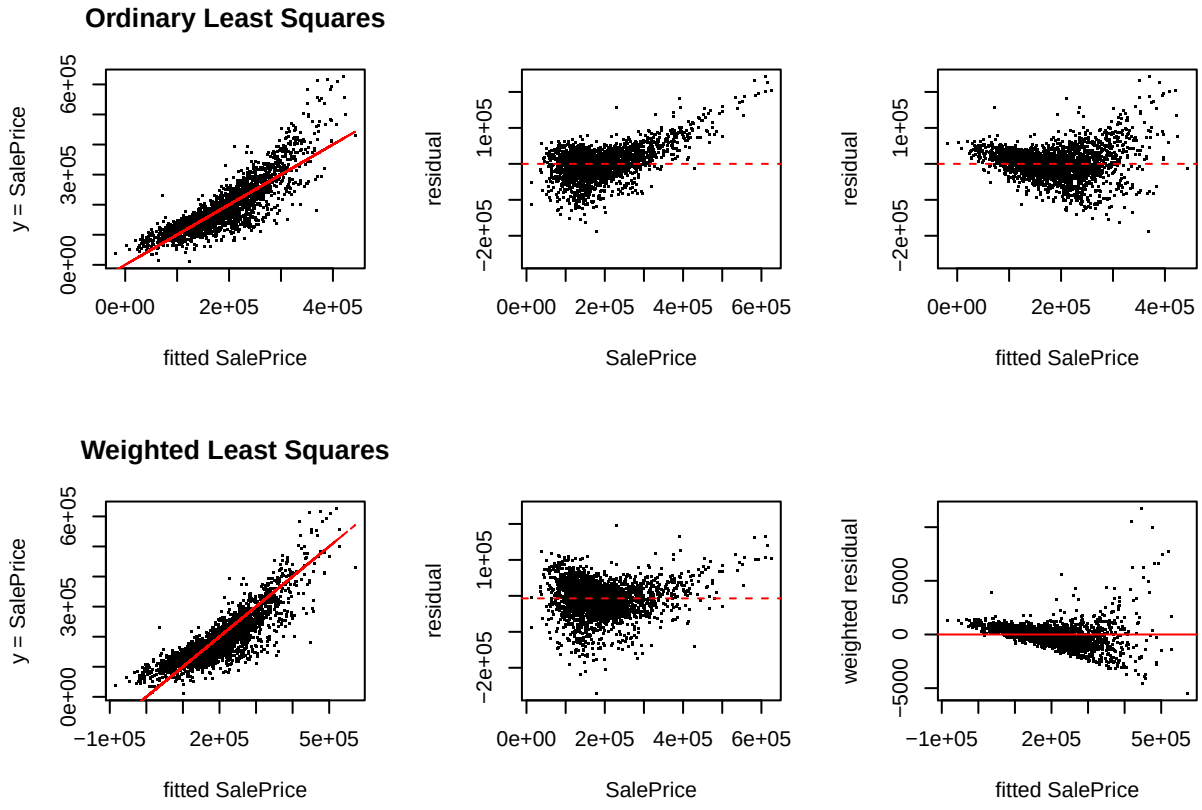
Replicate Figure 1 by using either the results stored in the variables `lm_ols` and `lm_wls`, or use your own answers from subtasks 3.1 and 3.2.

Answer:

The figure:

```
par(mfrow = c(2,3))#, pty = "s")

lims_resid <- range(c(residuals(lm_ols), residuals(lm_wls)))

# Ordinary Least Squares
plot(
  y = y, x = fitted(lm_ols),
  ylab = "y = SalePrice", xlab = "fitted SalePrice",
  main = "Ordinary Least Squares",
  pch = "."
)
lines(fitted(lm_ols), fitted(lm_ols), col = 'red')

plot(
  x = y, y = residuals(lm_ols),
  ylab = 'residual', xlab = "SalePrice",
  ylim = lims_resid, pch = "."
)
abline(h = mean(residuals(lm_ols)), col = 'red', lty = 2)

plot(
  x = fitted(lm_ols), y = residuals(lm_ols),
  ylab = 'residual', xlab = "fitted SalePrice",
  pch = ".", ylim = lims_resid
)
abline(h = 0, col = 'red', lty = 2)

# Weighted Leas Squares
plot(
  y = y, x = fitted(lm_wls),
```

```
  ylab = "y = SalePrice", xlab = "fitted SalePrice",
  main = "Weighted Least Squares",
  pch = "."
)
lines(fitted(lm_wls), fitted(lm_wls), col = 'red', lty = 2)

plot(
  x = y, y = residuals(lm_wls),
  ylab = 'residual', xlab = "SalePrice",
  pch = ".", ylim = lims_resid
)
abline(h = mean(residuals(lm_wls)), col = 'red', lty = 2)

plot(
  x = fitted(lm_wls), y = sqrt(v_weights) * residuals(lm_wls),
  ylab = 'weighted residual', xlab = "fitted SalePrice",
  pch = "."#, ylim = lims_resid
)
abline(h = 0, col = 'red')
```



4. About Bootstrapping for Validation (35 points).

This exam's particular linear regression analysis based on WLS comes with two disadvantages when compared to the OLS version: 1. it has a higher Mean Squares Error; 2. it has a higher prediction error expressed as $1 - R^2$. Here, this prediction error can be expressed as

$$Err = 1 - COR(y, \widehat{y})^2, \tag{4}$$

i.e. where the squared Pearson correlation between the predicted outcome $\widehat{y}$ and the outcome variable y is substracted from 1. Notet hat we have used this prediction error in one of the course weeks.

In this last task of the exam you will need to show that there is only a very small increase in prediction error when using the WLS model, as compared to the OLS model.

To do so, you will be programming a validation procedure that is proposed as an improvement on cross-validation: The .632+ Bootstrap Method (Efron & Tibshirani, 1997).

4.1. A probability of .632 (5 points)

It is a fact that bootstrap samples are supported on approximately $.632N$ of the original data points. When N is large enough, we have

$$\lim_{N \to \infty} P(i \in X^b) = 1 - e^{-1} \approx 0.632,$$

where $X^{(b)}$ denotes a bootstrap sample and i an observation that is present in the original sample X.

Create a function that computes the exact probability that i ends up (at least once) in the bootstrap sample $X^{(b)}$, i.e. $P(i \in X^b)$. Then, show in a visualization that this probability already becomes $\approx 1 - e^{-1}$ for $N = 50$.

Hint: Compute $1 - P(i \in X \in i \in X^b)$. *The probability that an observation* i *is NOT the first observation in the bootstrap sample is* $1 - 1/N$, *and the sampling process is identical and independent for all other observations in the boostrap sample.*

Answer:

The probability is

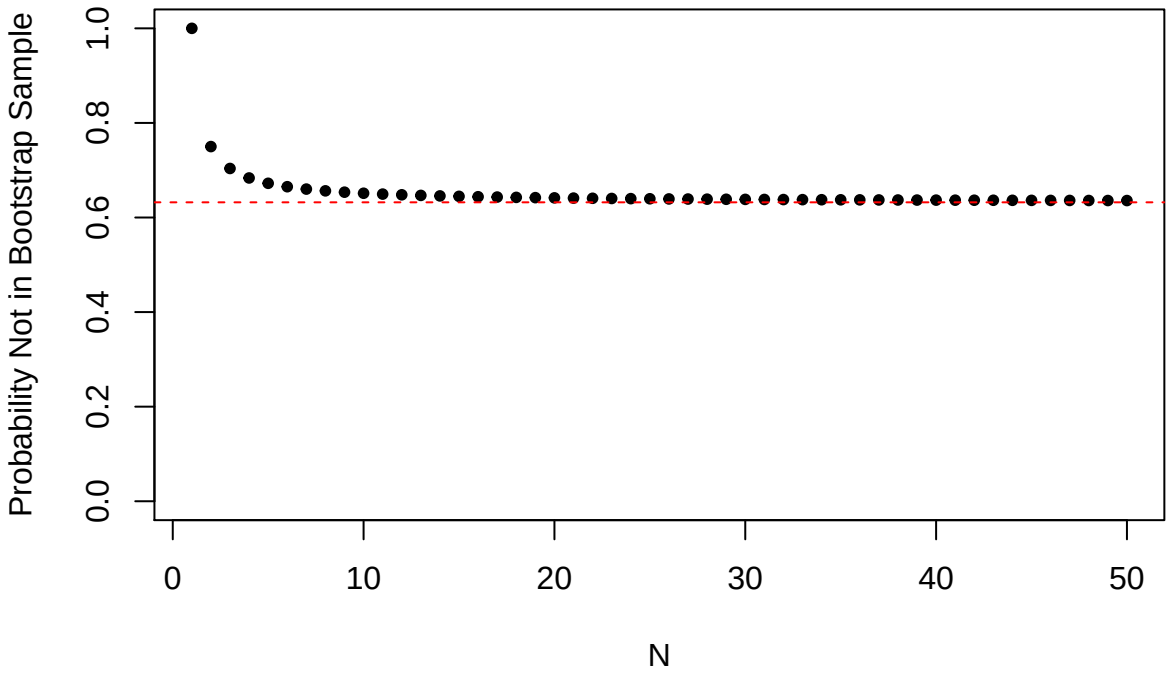
$$\lim_{\lambda \to \infty} \left(1 - \frac{1}{N}\right)^N = \frac{1}{e}$$

Coded in `R` this can be:

```
InSampleProb <- function(N) {
  1 - (1 - 1 / N )^(N)
}
```

And the requested visualization...

```
par(mfrow = c(1,1))
plot(
  x = 1:50,
  y = InSampleProb(1:50),
  pch = 20, xlab = "N", ylab = "Probability Not in Bootstrap Sample",
  ylim = c(0,1)
)
abline(h = 1 - 1 / exp(1), lty = 2, col = 'red')
```



4.2 A Smoothed Version of Cross-Validation: The .632+ Bootstrap Method (25 points)

For this subtask, write down the code for the resampling procedure that is used inside the bootstrap method such that you can obtain an estimate of the increase in prediction error. The algorithm can be programmed as follows:

The Algorithm

Set $B = 100$. Then, for each $b \in \{1, \ldots, B\}$ bootstrap samples of the data set `ames_prep`, do the following:

1. Train the models.

Let the bootstrap sample X^b be a training data set on which we fit the linear model with OLS and the WLS linear regression model (as was asked to you in subtaks 3.1 and 3.2.)

2. Training error.

Denote $\widehat{y}_{OLS}^{(b)}$ and $\widehat{y}_{WLS}^{(b)}$ as the fitted values for `SalePrice` of the OLS and WLS models, respectively. Define the increase in prediction error for your training data set as

$$\overline{\text{err}}_b = cor(y^{(b)}, \widehat{y}_{OLS}^{(b)})^2 - cor(y^{(b)}, \widehat{y}_{WLS}^{(b)})^2, \tag{5}$$

and store this specific value.

3. Obtain predictions for test data.

Use the observations that were NOT selected in your bootstrap sample X^b as the test data: $X^{(-b)}$. Based on your trained OLS and WLS models, obtain the predictions for the housing prices of the remain objects denoted as $\widehat{y}_{OLS}^{(-b)}$ and $\widehat{y}_{WLS}^{(-b)}$ respectively.

4. Test error.

Store the test error that can be computed as

$$\widehat{\text{Err}}_b^{(1)} = cor(y^{(-b)}, \widehat{y}_{OLS}^{(-b)})^2 - cor(y^{(-b)}, \widehat{y}_{WLS}^{(-b)})^2, \tag{6}$$

After completing these for steps for each bootstrap sample, compute the final increased training error as

$$\overline{\text{err}} = B^{-1} \sum_{b=1}^{B} \overline{\text{err}}_b \tag{7}$$

and the final increased test error as

$$\widehat{\text{Err}}^{(1)} = B^{-1} \sum_{b=1}^{B} \widehat{\text{Err}}_b^{(1)} \tag{8}$$

Hint 1: Decide for yourself how to deal with the exponential weights in the WLS-model. Although not completely correct, when you also bootstrap the exponential weights you can still obtain a perfect score for this task.

Hint 2: When you did not succeed in subtasks 3.1 and 3.2, then either create this resampling procedure for a model and a prediction error definition of your own choice, or use the ***trained_vars*** *variable that is a list that contains of B lists for each bootstrap sample. Each list of a bootstrap sample is again a list that contains the in-sample indices of the observations, the OLS model results, and the WLS model results.*

Answer:

Set the parameters:

```
X <- ames_prep
lambda <- 15e4
v_weights <- GetWeights(X$`SalePrice`, lmbd = lambda)
N <- nrow(X)
set.seed(20190108)
B <- 100
trn_errs <- numeric(B)
tst_errs <- numeric(B)
trained_vars <- vector(B, mode = "list")
```

The Resampling Procedure..

```
for (b in 1:B) {
  # b <- 1
  # Training Data:
  train_i <- sample(N, replace = TRUE)
  Xb <- X[train_i,]
```

```
  # v_b <- v_weights[train_i,]
  y_trn <- Xb$SalePrice

  # Train Models and Obtain Test Error
  v_b <- GetWeights(Xb$`SalePrice`, lmbd = lambda)
  lm_ols_b <- lm(price_model, data = Xb)
  lm_wls_b <- lm(price_model, data = Xb, weights = v_b)

  trained_vars[[b]] <- list(train_i, lm_ols_b, lm_wls_b)

  R2_ols_trn <- cor(y_trn, fitted(lm_ols_b))^2
  R2_wls_trn <- cor(y_trn, fitted(lm_wls_b))^2
  trn_errs[b] <- R2_ols_trn - R2_wls_trn

  # Test Data
  X_tst <- X[-train_i,]
  # v_tst <- v_weights[-train_i,]
  y_tst <- X_tst$SalePrice

  # Test Error
  ypred_ols <- predict(lm_ols_b, X_tst)
  ypred_wls <- predict(lm_wls_b, X_tst)
  R2_ols_tst <- cor(y_tst, ypred_ols)^2
  R2_wls_tst <- cor(y_tst, ypred_wls)^2
  tst_errs[b] <-  R2_ols_tst - R2_wls_tst
}
err_bar <- mean(trn_errs)
Err1 <- mean(tst_errs)
```

4.3

To obtain a 'good' estimate of the increase in prediction error, denoted by $\widehat{\text{Err}}^{(.632+)}$, Efron & Tibshirani (1997) advise to combine $\overline{\text{err}}$ and $\widehat{\text{Err}}^{(1)}$ as follows:

$$\widehat{\text{Err}}^{(.632+)} = (1 - \widehat{w})\overline{\text{err}} + \widehat{w}\widehat{\text{Err}}^{(1)}, \tag{9}$$

where

$$\widehat{w} = \frac{0.632}{1 - 0.368\widehat{\text{ROR}}},$$

and $0 \leq \widehat{\text{ROR}} \leq 1$ stands for the relative overfitting rate. When $\widehat{\text{ROR}} = 0$, then there is basically no overfitting in the training error, and when $\widehat{\text{ROR}} = 1$ then there is a maximal amount of overfitting.

To avoid that this exam becomes too long, let us assume that there is no overfitting involved in the training error, i.e. $\widehat{\text{ROR}} = 0$.

What is the value of $\widehat{\text{Err}}^{(.632+)}$?

Hint: You can use `err_bar` *and* `Err1` *that came with the answer variables in* `Exam_answer_variables.RData`.

Answer:

```r
w_hat <- 0.632
Err_hat_0.632 <- (1-w_hat)*err_bar+w_hat*Err1
Err_hat_0.632
```

```
## [1] 0.004480271
```