Linear Regression

Art Tay

Gradient Derivation

a)

$$\begin{aligned}\frac{\partial}{\partial \beta_0}\left[\frac{1}{2m}\sum_{i=0}^{m}(\beta_0+\beta_1 x^{(i)}-y^{(i)})^2\right] &= \frac{\partial}{\partial \beta_0}\left[\frac{1}{2m}\sum_{i=0}^{m}(\beta_0+\beta_1 x^{(i)}-y^{(i)})^2\right] \\ &= \frac{1}{2m}\sum_{i=0}^{m}\left[2(\beta_0+\beta_1 x^{(i)}-y^{(i)})\cdot 1\right] \\ &= \frac{1}{m}\sum_{i=0}^{m}\beta_0+\beta_1\frac{1}{m}\sum_{i=0}^{m}x^{(i)}-\frac{1}{m}\sum_{i=0}^{m}y^{(i)} \\ &= \beta_0+\beta_1\bar{x}-\bar{y}\end{aligned}$$

b)

$$\begin{aligned}\frac{\partial}{\partial \beta_1}\left[\frac{1}{2m}\sum_{i=0}^{m}(\beta_0+\beta_1 x^{(i)}-y^{(i)})^2\right] &= \frac{1}{2m}\sum_{i=0}^{m}\frac{\partial}{\partial \beta_1}\left[(\beta_0+\beta_1 x^{(i)}-y^{(i)})^2\right] \\ &= \frac{1}{2m}\sum_{i=0}^{m}\left[2x^{(i)}(\beta_0+\beta_1 x^{(i)}-y^{(i)})\right] \\ &= \frac{1}{m}\sum_{i=0}^{m}\left[\beta_0 x^{(i)}+\beta_1 {x^{(i)}}^2-x^{(i)}y^{(i)}\right] \\ &= \beta_0\frac{1}{m}\sum_{i=0}^{m}x^{(i)}+\beta_1\frac{1}{m}\sum_{i=0}^{m}{x^{(i)}}^2-\frac{1}{m}\sum_{i=0}^{m}x^{(i)}y^{(i)} \\ &= \beta_0\bar{x}+\beta_1\bar{x^2}-\bar{xy}\end{aligned}$$

Linear Regression by Gradient Decent

```
# Generates linear data with normal residuals
set.seed(123)
x <- rnorm(n = 30)

epsilon <- rnorm(n = 30)

y <- 5*x + 1 + epsilon
```

```
summary(lm(y ~ x))
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6085 -0.5056 -0.2152  0.6932  2.0118
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.1720     0.1534   7.639 2.54e-08 ***
## x             4.8660     0.1589  30.629  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8393 on 28 degrees of freedom
## Multiple R-squared:  0.971,  Adjusted R-squared:   0.97
## F-statistic: 938.1 on 1 and 28 DF,  p-value: < 2.2e-16
```

```r
# Calculates the mean squared error for a simple linear regression model.
# @param x - a vector of the explainatory variable.
# @param y - a vector of the response variable.
# @param beta_0 - the intercept value for the current SLR model.
# @param beta_1 - the slope value for the current SLR model.
# @return - sum total mean squared error (y_hat - y)^2
slr_mse <- function(x, y, beta_0, beta_1){
    cost <- ((beta_1 * x + beta_0) - y)^2
    return(sum(cost))
}
```

```r
# Calculates the slope and intercept values for SLR
# or simple linear regression.
# @param x - a vector of the explanatory variable.
# @param y - a vector of the response variable.
# @param alpha - the learning rate.
# @return betas - a vector containing the calculated betas.
slr_gradient_desc <- function(x, y, alpha){

    # Summary statistic calculations.
    # Helps to calculate the gradient faster.
    x_bar <- mean(x)
    y_bar <- mean(y)
    xy_bar <- mean(x*y)
    x_sqbar <- mean(x^2)

    # initial guess for beta_0 and beta_1.
    beta_0 <- y_bar
    beta_1 <- 0

    # A counter to determine is the error is unchanging.
    # This is the Loop-Control-Variable (LCV).
    count_same <- 0

    # Iterate 100 times or until the cost remains unchanged for 10 iterations.
```

```
    for(i in 1:1000){

        # Stop the loop if the LCV >= 10.
        if(count_same >= 10){
            break
        }

        # Cost prior to beta adjustment.
        cost_start <- slr_mse(x, y, beta_0, beta_1)

        # Calculate gradient values.
        g_0 <- beta_0 + (beta_1 * x_bar) - y_bar
        g_1 <- (beta_0 * x_bar) + (beta_1 * x_sqbar) - xy_bar

        # Update betas.
        beta_0 <- beta_0 - (alpha * g_0)
        beta_1 <- beta_1 - (alpha * g_1)

        # If the cost is unchanged add 1 to the LCV.
        if(cost_start == slr_mse(x, y, beta_0, beta_1)){
            count_same <- count_same + 1
        }
    }

    return(c(beta_0 = round(beta_0, 4),
             beta_1 = round(beta_1, 4),
             iterations = i))
}
```

```
slr_gradient_desc(x, y, alpha = 0.1)
```

```
##     beta_0     beta_1 iterations
##      1.172      4.866    256.000
```

Linear Model on Airbnb Data

```
# Libraries
library(tidyverse)
library(tidymodels)
```

Modeling Bulding

```
# Import raw data.
data_raw <- read.csv(file = "AB_NYC_2019.csv", stringsAsFactors = T,
                     header = T)
```

```
# 70/30 train/test data split
# Data was split prior to cleaning to prevent data leakage.
```

```r
set.seed(123)
data_split <- initial_split(data_raw, prop = 0.7)

train_data <- training(data_split)
test_data <- testing(data_split)
```

```r
# Define a cleaning recipe from the data.
# Assigns price the role of response.
# Recipe is used to create a pipline that can works with new raw data
# assuming that the format matches the training data.
cleaning_recipe <- recipe(price ~ ., data = train_data)
```

```r
cleaning_recipe <- cleaning_recipe %>% step_rm(id, host_id, name, host_name)
```

```r
cleaning_recipe <- cleaning_recipe %>%
    step_mutate(
        # numeric recodes
        latitude = ifelse(latitude == 0, NA, latitude),
        longitude = ifelse(longitude == 0, NA, longitude),
        minimum_nights = ifelse(minimum_nights == 0, NA, minimum_nights),

        #Factor recodes
        # It was done using replace to maintain factor coding
        room_type = replace(room_type,
            room_type == "" | room_type == " ", NA),
        neighbourhood_group = replace(neighbourhood_group,
            neighbourhood_group == "" | neighbourhood_group == " ",  NA),
        neighbourhood = replace(neighbourhood,
            neighbourhood == "" | neighbourhood == " ", NA),
        last_review = replace(last_review,
            last_review == "" | room_type == " ", NA)
    )
```

```r
# Skip = T tell the fit to ignore the step if it can't be done.
# Necessary because we assume the test_data does not have the response.
cleaning_recipe <- cleaning_recipe %>%
                   step_mutate(price = ifelse(price == 0, NA, price), skip = T)
```

```r
# Geodist (?)
cleaning_recipe <- cleaning_recipe %>%
    step_mutate(

            # Change NA reviews per month to be 0.
            reviews_per_month = ifelse(is.na(reviews_per_month),
                0, reviews_per_month),

            # Create a new variable last_review_year to reduce dimensionality.
            last_review_year = substring(last_review, 1, 4),

            # Modify NA last_review to be a new level "none".
            last_review_year = replace(last_review_year,
            is.na(last_review_year), "none"),
```

```
            # Cast to be factor.
            last_review_year = as.factor(last_review_year),

            # Remove old last review variable
            last_review = NULL
    )
```

```
plot_skew <- train_data %>%
             select(where(is.numeric)) %>%
             select(-c(id, host_id)) %>%
             pivot_longer(cols = everything()) %>%
             ggplot(aes(value)) +
             geom_histogram() +
             facet_wrap(~name, scales = "free_x") +
             theme_bw() +
             labs(x = "", y = "Frequency")
```

```
cleaning_recipe <- cleaning_recipe %>%
                   step_YeoJohnson(all_numeric_predictors()) %>%
                   # price was logged instead of YeoJohnson to maintain
                   # reversibility and interpretability.
                   step_log(price, skip = T)
```

```
cleaning_recipe <- cleaning_recipe %>%
                   step_normalize(all_numeric_predictors()) %>%
                   step_normalize(price, skip = T) %>%
                   step_dummy(all_nominal_predictors()) %>%
                   step_impute_knn(everything(), neighbors = 10,
                                   impute_with = imp_vars(everything()),
                                   skip = T)
```

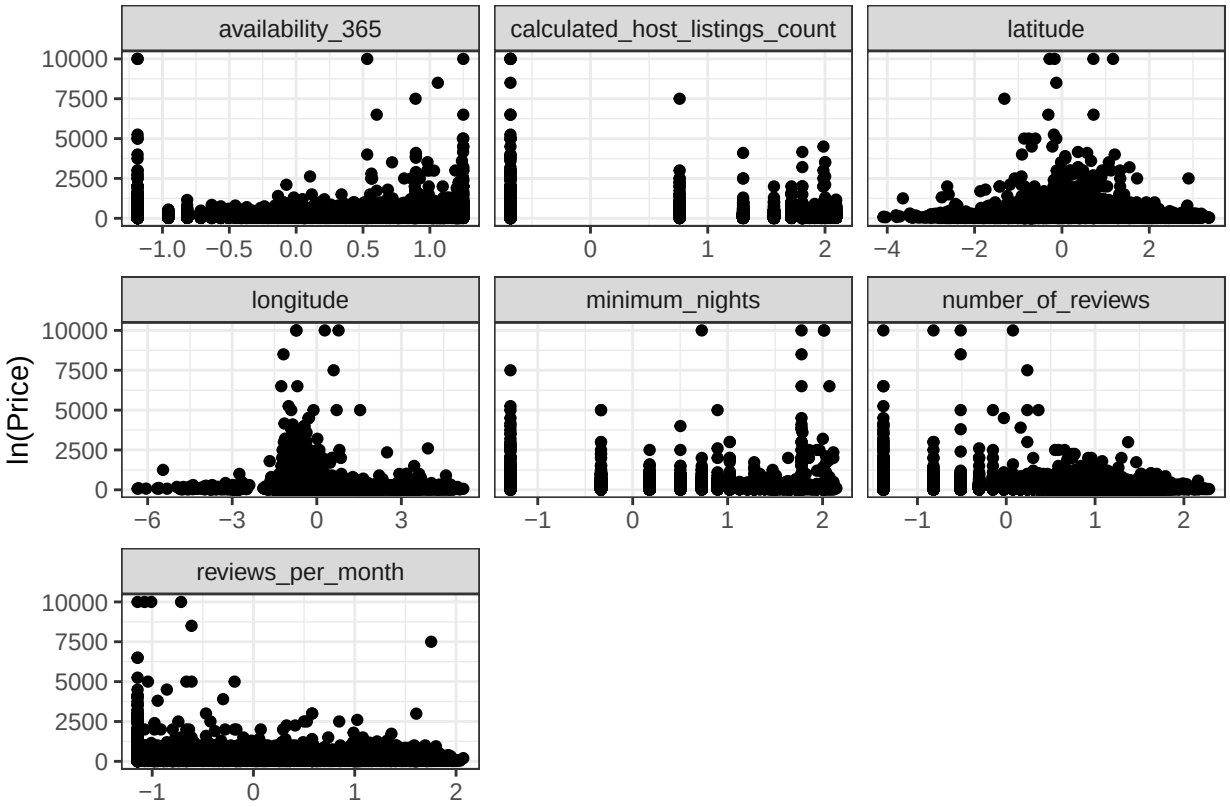
```
cleaning_recipe <- cleaning_recipe %>%
                   # removes predictor with less than 10% unique values and
                   # a greater than 95/5 ratio of most prevalent to next most
                   step_nzv(all_predictors()) %>%
                   step_corr(all_numeric_predictors())
```

```
cleaning_recipe <- cleaning_recipe %>% prep(retain = T, verbose = T)
```

```
## oper 1 step rm [training]
## oper 2 step mutate [training]
## oper 3 step mutate [training]
## oper 4 step mutate [training]
## oper 5 step YeoJohnson [training]
## oper 6 step log [training]
## oper 7 step normalize [training]
## oper 8 step normalize [training]
## oper 9 step dummy [training]
## oper 10 step impute knn [training]
## oper 11 step nzv [training]
## oper 12 step corr [training]
## The retained training set is ~ 5.5 Mb  in memory.
```

```
train_scatter <- bake(cleaning_recipe, new_data = train_data)
```

```
# Matrix Scatterplot
plot_scatter <- train_scatter %>%
                select(-starts_with(c("neighbourhood",
                                      "room_type", "last_review"))) %>%
                pivot_longer(cols = -price) %>%
                ggplot(aes(x = value, y = price)) +
                geom_point() +
                facet_wrap(~name, scales = "free_x") +
                theme_bw() +
                labs(x = "", y = "ln(Price)")

plot_scatter
```



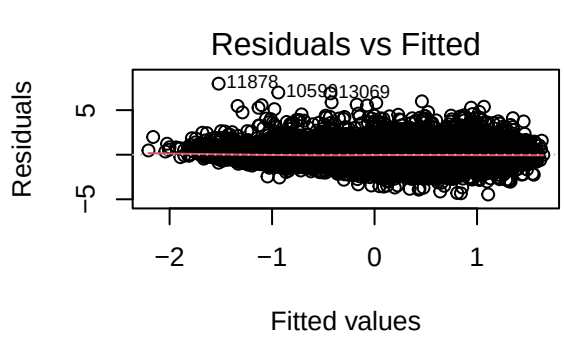

```
cleaning_recipe <- cleaning_recipe %>%
                   # makes sure not to accidentally spline your response variable
                   step_ns(-starts_with(c("neighbourhood",
                                "room_type", "last_review", "price")),
                           deg_free = 3)
```

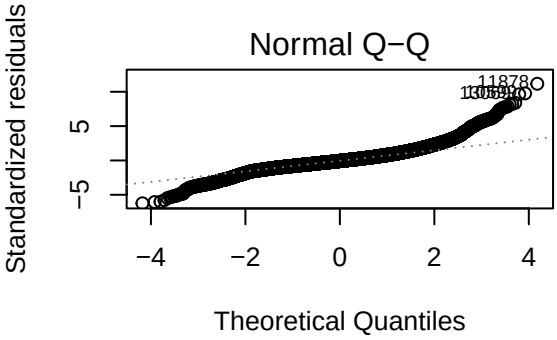
```
mlr_mod <- linear_reg() %>% set_engine("lm")

mlr_wflow <- workflow() %>%
```

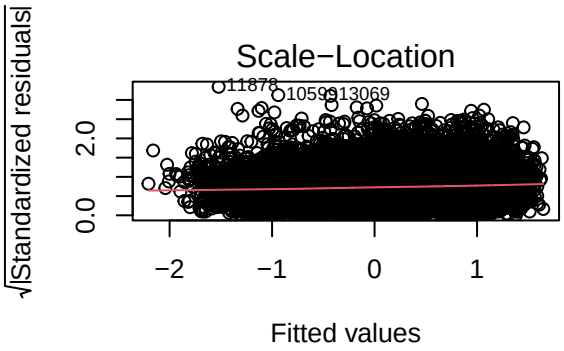
```
            add_model(mlr_mod) %>%
            add_recipe(cleaning_recipe)

mlr_wflow
```

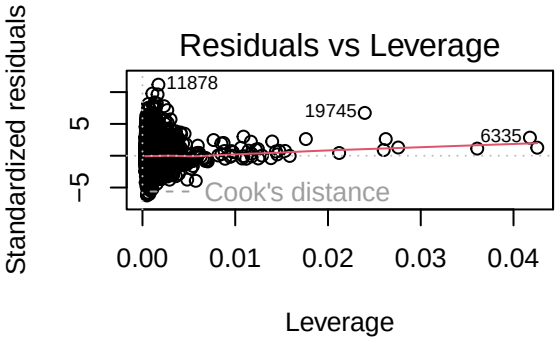
```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 13 Recipe Steps
##
## * step_rm()
## * step_mutate()
## * step_mutate()
## * step_mutate()
## * step_YeoJohnson()
## * step_log()
## * step_normalize()
## * step_normalize()
## * step_dummy()
## * step_impute_knn()
## * ...
## * and 3 more steps.
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

```
mlr_fit <- mlr_wflow %>%
           fit(data = train_data)

#mlr_fit %>% extract_fit_parsnip() %>% tidy() %>% print(n = 35)

mlr_fit_lm <- mlr_fit %>% extract_fit_engine()

par(mfrow = c(2,2))
plot(mlr_fit_lm)
```










```r
# Make predictions from MLR
mlr_pred <- predict(mlr_fit, test_data)
```