Analyzing the Amazon Popular Books Dataset

Jonathan Fung

June 21, 2022

Contents

Introduction

- From Bright Data
- `https://github.com/luminati-io/Amazon-popular-books-dataset`

Libraries

```
library(dplyr)
library(tidyr)

library(ggplot2)
library(gridExtra)
library(ggfortify)

library(tidyjson)

library(MASS)
```

Utils

Author: J. Peter Marquardt, GPL. Deconstruct a formula object into strings of its components. Predictors are split by '+', so interaction terms will be returned as a single string.

```r
deconstruct_formula <- function(formula) {
    # deparsing formula into string with no spaces and newlines
    form_string <- gsub(" ", "",
                        gsub("\n", "", deparse1(formula, collapse = "")))

    # extracting components
    if (substr(form_string, 1, 5) == "Surv(") { # Survival formula
        # Extracting the Surv() part of it
        surv_string <-
            strsplit(form_string, ")")[[1]][1]
        # extracting everything inside the Surv()
        surv_params <-
            strsplit(substr(surv_string, 6,
                            nchar(surv_string)), ",")
        outcome <- surv_params[[1]][1] # assigning time variable name
        censor_event <- surv_params[[1]][2] # assigning cens variable name
    } else { # ordninary formula
        outcome <- strsplit(form_string, "~")[[1]][1]
        censor_event <- NULL
    }
    predictors <- strsplit(strsplit(form_string, "~")
                           [[1]][2], split = "+",
                           fixed = TRUE)[[1]] # same for all

    # assembling output list
    component_list <- list(
        "outcome" = outcome,
        "predictors" = predictors
    )
    if (!is.null(censor_event)) {
        component_list$`censor_event` <- censor_event
    }

    return(component_list)
}
```

Sometimes `images_count` is not a string, need to convert to number.

```r
parse_rating <- function(rting) {
    as.double(sub(" .*$", "\\1", rting))
}
```

```r
parse_category <- function(cat_string, n) {
    subcats <- tail(strsplit(cat_string, "/")[[1]], -1)
    subcats <- trimws(subcats)
    if (n > length(subcats)) {
        n <- length(subcats)
    }
    res <- subcats[1:n]
    res <- paste(res, collapse = "/")
    return(res)
}
```

```r
gg_regress <- function(data, model, method = "glm", ...) {
    formula <- getElement(model, "call") %>%
        getElement("formula") %>%
        deconstruct_formula()
    family <- getElement(model, "call") %>%
        getElement("family")
    if (is.null(family)) {
        family <- "gaussian" # default for glm
    }

    indep <- getElement(formula, "outcome")
    deps <- getElement(formula, "predictors")
    p <- data %>%
        ggplot(aes_string(x = deps, y = indep)) +
        geom_smooth(method = method, method.args = list(family = family)) +
        geom_point(...)
    return(p)
}
```

Parse Data

```r
system("jq '.[1:2269] | .[].images_count |= tonumber | map(del(.description, .format,
↪  .video_count))' Amazon_popular_books_dataset.json > truncated.json")

df <- tidyjson::read_json("truncated.json") %>%
    gather_array() %>%
    spread_all() %>%
    mutate(rating = parse_rating(rating)) %>%
    # remove columns with one constant value
    dplyr::select(where(~ n_distinct(.) != 1))

head(df)
```

```
# A tbl_json: 6 x 30 tibble with a "JSON" attribute
  ..JSON            array.index asin  ISBN10 answered_questi... availability brand
  <chr>                   <int> <chr> <chr>               <dbl> <chr>        <chr>
1 "{\"asin\":\"000...         1 0007... 97800...              0 In Stock.    Drew...
2 "{\"asin\":\"000...         2 0008... 00081...              0 <NA>         Bern...
3 "{\"asin\":\"000...         3 0008... 00083...              0 In Stock.    Davi...
4 "{\"asin\":\"000...         4 0008... 00083...              0 In Stock.    Caro...
5 "{\"asin\":\"000...         5 0008... 00083...              0 Only 13 lef... J. R...
6 "{\"asin\":\"000...         6 0008... 00084...              0 Usually shi... J. R...
# ... with 23 more variables: buybox_seller <chr>, date_first_available <chr>,
#   discount <dbl>, final_price <dbl>, image_url <chr>, images_count <dbl>,
#   initial_price <dbl>, item_weight <chr>, manufacturer <chr>,
#   model_number <chr>, plus_content <lgl>, product_dimensions <chr>,
#   rating <dbl>, reviews_count <dbl>, root_bs_rank <dbl>, seller_id <chr>,
#   seller_name <chr>, timestamp <chr>, title <chr>, url <chr>, video <lgl>,
#   image <chr>, number_of_sellers <dbl>
```

All NA: `date_first_available`, `manufacurer`, `department`, `model_number`, `upc`.

Some NA: `product_dimensions`, `root_bs_rank`, `buybox_seller`, `final_price`, `initial_price`, `seller_id`, `availability`, `discount`, `item_weight`.

Of the JSON arrays: `best_sellers_rank`, `categories` are interesting. `colors`, `delivery`, and `features` are not very relevant, as they are either very verbose or empty.

Each book has 3 categories, one "Books", and two other. `categories` are always in the `best_sellers_rank` categories.

Process Data

```
df_arrays <- df %>%
    gather_object %>%
    filter(is_json_array(.)) %>%
    gather_array()
```

Select some key features, then drop all NA rows:

```
df_array_na_rows <- df_arrays %>%
    dplyr::select(asin, final_price, initial_price,
                  reviews_count, rating, availability, discount,
                  plus_content, images_count) %>%  drop_na() %>% distinct
head(df_array_na_rows %>% as.data.frame)
```

```
        asin final_price initial_price reviews_count rating
1 0008387753       41.12         59.99         20453    4.8
2 0060244887       53.99        120.00         11222    4.8
3 0060254920       13.20         19.95         27536    4.9
4 0060256656        9.09         17.99         23158    4.9
5 0060555661       14.29         24.99         28414    4.7
6 0060652888       22.49         24.99         10958    4.8
                         availability discount plus_content images_count
1 Only 13 left in stock - order soon.    18.87        FALSE            1
2 Only 12 left in stock - order soon.    66.01        FALSE            2
3                           In Stock.     6.75        FALSE            6
4                           In Stock.     8.90        FALSE            3
5                           In Stock.    10.70        FALSE            3
6                           In Stock.     2.50        FALSE            5
```

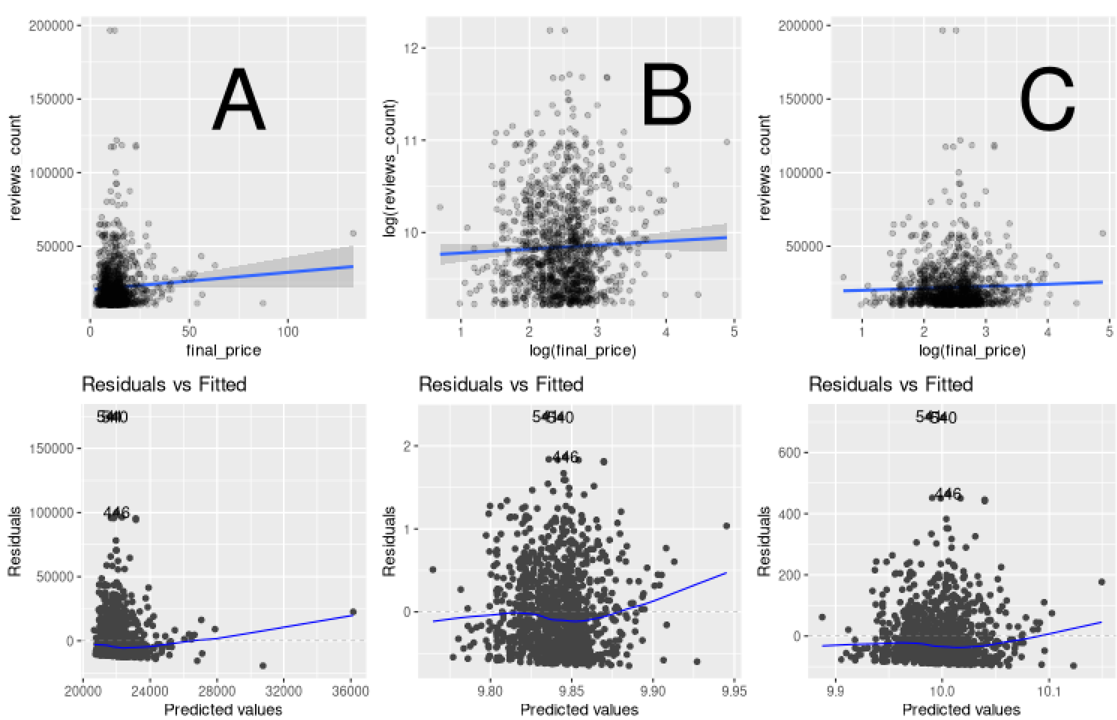
Compare Regression Models on reviews count ∼ final price

```r
df_rows_lm_linear <- glm(reviews_count ~ final_price,
                         data = df_array_na_rows)
df_rows_lm <- glm(log(reviews_count) ~ log(final_price),
                  data = df_array_na_rows)
df_rows_glm <- glm(reviews_count ~ log(final_price),
                   data = df_array_na_rows, family = poisson("log"))

df_rows_lm_linear_resid <- autoplot(df_rows_lm_linear, which = 1, ncol = 1)
df_rows_lm_resid <- autoplot(df_rows_lm, which = 1, ncol = 1)
df_rows_glm_resid <- autoplot(df_rows_glm, which = 1, ncol = 1)

df_rows_lm_linear_plot <- df_array_na_rows %>%
                          gg_regress(df_rows_lm_linear, alpha = 1 / 5) +
                          annotate("text", x = 75, y = 150000, label = "A", size = 20)
df_rows_lm_plot <- df_array_na_rows %>%
                   gg_regress(df_rows_lm, alpha = 1 / 5) +
                   annotate("text", x = 4, y = 11.5, label = "B", size = 20)
df_rows_glm_plot <- df_array_na_rows %>%
                    gg_regress(df_rows_glm, alpha = 1 / 5) +
                    annotate("text", x = 4, y = 150000, label = "C", size = 20)
```

```r
grid.arrange(df_rows_lm_linear_plot, df_rows_lm_plot, df_rows_glm_plot,
             attr(df_rows_lm_linear_resid, "plots")[[1]],
             attr(df_rows_lm_resid, "plots")[[1]],
             attr(df_rows_glm_resid, "plots")[[1]], nrow = 2)
```



A: reviews count $\sim$ final price	Gaussian Family
B: $\log$ (reviews count) $\sim$ $\log$ (final price)	Gaussian Family
C: reviews count $\sim$ $\log$ (final price)	Poisson Family

Here, we compare three different regression models with `reviews_count` $\sim$ `final_price`. A and B are in the Gaussian family, with C being a Poisson regression with log link. Model A and B only differ in B's log transformation of the dependent variable. Concerning assumptions of regression, all are roughly homoscedastic, and B's residuals being more *normal* than C's. Poisson regression makes the most sense with count data, so C should be the most appropriate model.

A generalization of the Poisson distribution is the negative binomial, which is used for more overdispersed cases. Comparing the two, we see that the Poisson model is more appropriate over the Negative Binomial model.

```
df_rows_glm_nb <- glm.nb(reviews_count ~ log(final_price), data = df_array_na_rows)
pchisq(2 * (logLik(df_rows_glm) - logLik(df_rows_glm_nb)),
       df = 1, lower.tail = FALSE)
```

```
`geom_smooth()` using formula 'y ~ x'
`geom_smooth()` using formula 'y ~ x'
`geom_smooth()` using formula 'y ~ x'
png
  2
'log Lik.' 1 (df=2)
```

The coefficient on `log(final_price)` is `0.062357` (p-value: `0`), so, there is pretty much no effect of final price on `reviews_count`.

Compare Ordinal Regression Models on reviews count $\sim$ final price

rating takes on values from 3.9 to 4.9 in units of starts out of 5.0. Since this is a discrete variable, with implicit ordering, ordinal regression should be applied here.

```
clean_df_array_na_rows <- df_array_na_rows %>%
                         dplyr::select(-availability, -asin) %>%
              ## Need to scale reviews_count so SVD in regression can converge
                         mutate(reviews_count = reviews_count / 10) %>%
                         mutate(discount_rel = discount / initial_price) %>%
                         distinct

(ord_model2 <- polr(ordered(rating) ~
                  discount_rel + plus_content + images_count,
                  Hess = TRUE, data = clean_df_array_na_rows))
```

```
log(final_price)
      "0.062357"
[1] 0
Call:
polr(formula = ordered(rating) ~ discount_rel + plus_content +
    images_count, data = clean_df_array_na_rows, Hess = TRUE)

Coefficients:
    discount_rel plus_contentTRUE     images_count
      -0.2466810        0.2339840        0.1296111

Intercepts:
    3.6|3.9       3.9|4       4|4.1     4.1|4.2     4.2|4.3     4.3|4.4
-6.65807760 -5.04496078 -4.08190638 -3.59451219 -2.67649658 -2.10438745
    4.4|4.5     4.5|4.6     4.6|4.7     4.7|4.8     4.8|4.9
-1.48469678 -0.72192278  0.08974073  1.18404473  2.73695440

Residual Deviance: 4375.213
AIC: 4403.213
```

For a one point increase in relative discount, the odds of the rating being a tenth of a star higher is `0.78139` times the previous, holding all equal. Books with `plus_content` have `1.2636` times the probability of being a higher rating than books without, holding all equal. For a one point increase in a book's images count, the odds of the next rating is `1.1384` times the previous.

Resources

- UCLA DAE: Negative Binomial Regression
- UCLA DAE: Ordinal Logistic Regression