Testing the performance of our simple Amazon EB model

Thomas Ma

1/8/2022

At the end of our prior-fitting process, we found the the weights for the 5 parameters of our Dirichlet prior were given by $\hat{\alpha} = (18.41, 4.78, 7.57, 21.55, 309.98)$. We now want to see how well our Bayesian model performs in assessing the rating distribution of products.

Reasoning behind evaluation procedure

Let's say that we have some initial data for a product p with a 10 1-star ratings, 2 2-star ratings, 3 3-star ratings, 20 4-star ratings, and 80 5-star ratings. Given this data and the fitted Dirichlet prior from above, we seek some measure of the rating distribution of the product, in order to assess how "good" this product is.

Let $r_p = (10, 2, 3, 20, 80)$. Feeding this into our fitted Dirichlet prior, we obtain a posterior Dirichlet distribution with parameters (28, 6.78, 10.57, 41.55, 389.98). Of course, this posterior distribution does not give us exactly what we want; in essence, the posterior distribution gives us a distribution over possible categorical distributions that the product could have.

Recall that the output of a Dirichlet distribution with parameters $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ is some 5-tuple $(X_1, X_2, X_3, X_4, X_5)$ where $\sum_i x_i = 1$. In our problem, this 5-tuple represents the weights for some categorical distribution with 5 categories. Using this information, I elected to estimate the proportion of i-star ratings that a product p with Dirichlet posterior D_p would have as $E_{X \sim D_p}[X_i]$. This, at the time, seemed to me like the best way to account for the spread of outcomes suggested by the posterior distribution.

For example, using the numbers from our imaginary product above, we would estimate the expected proportion of 1-star ratings the product gets as $\frac{28}{28+6.78+10.57+41.55+398.89} \approx 0.576$.

Experiment design and outcomes

For each product held out to test time, the following procedure was executed:

1. Ratings for the product were sorted by timestamp and split 70/30 into two groups: One to fit the posterior, and one for testing.
2. After a posterior with parameters $(\beta_1 ... \beta_5)$ for the product was constructed, $E[X_i]$ was calculated as $\frac{\beta_i}{\sum_i \beta_i}$.
3. To test our performance, the squared difference between $E[X_i]$ and the proportion of i-star ratings in the test set was added to our loss $\mathcal{L}$.
4. I computed the MSE as $\mathcal{L}/(5 \cdot k)$, where k is the total number of products in our test set.

As a benchmark, I also created a "frequentist" baseline model for the 70/30 split that simply tried to predict the distribution of the 30% by copying the distribution of the 70%. This method is equivalent to MLE for a categorical distribution after seeing the 70% training data.

Actual testing code

```
library(data.table)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::between()   masks data.table::between()
## x dplyr::filter()    masks stats::filter()
## x dplyr::first()     masks data.table::first()
## x dplyr::lag()       masks stats::lag()
## x dplyr::last()      masks data.table::last()
## x purrr::transpose() masks data.table::transpose()
```

```
# Load the data

raw_df = read.csv("../data/Gift_Cards.csv", header = FALSE, col.names=c("item", "user", "rating", "time
head(raw_df)
```

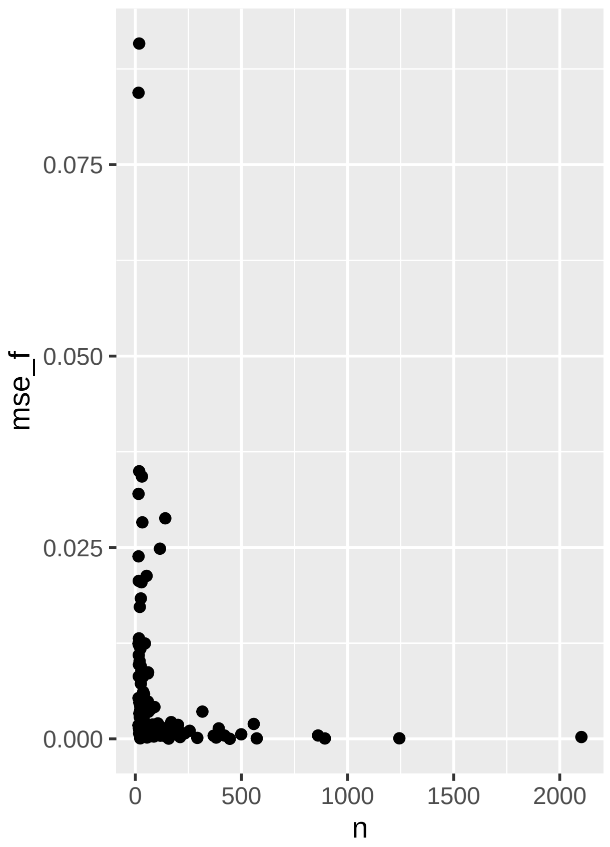
```
##         item           user rating  timestamp
## 1 B001GXRQW0  APV13CM0919JD      1 1229644800
## 2 B001GXRQW0 A3G8U1G1V082SN      5 1229472000
## 3 B001GXRQW0  A11T2Q0EVTUWP      5 1229472000
## 4 B001GXRQW0  A9YKGBH3SV22C      5 1229472000
## 5 B001GXRQW0 A34WZIHVF3OKOL      1 1229472000
## 6 B001GXRQW0 A221J8EC5HNPY6      3 1229385600
```

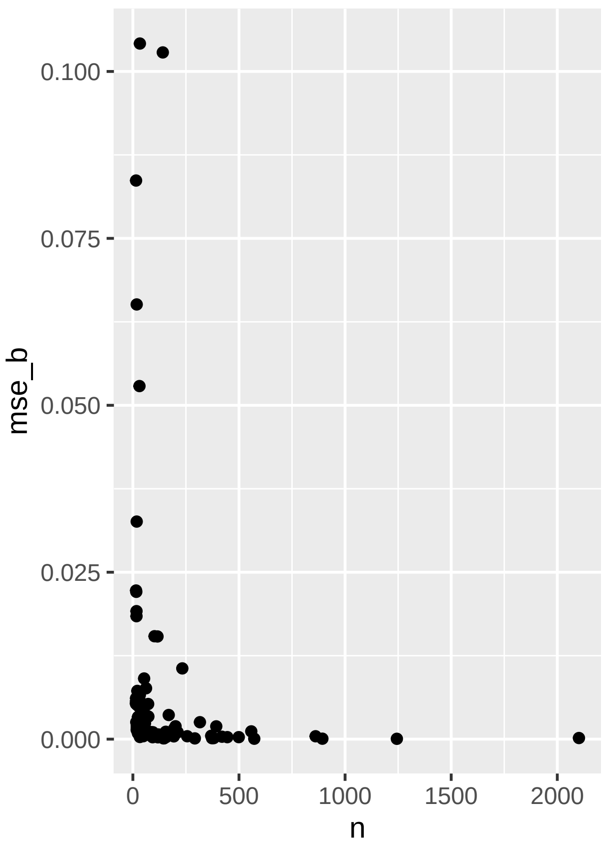
```
df <- raw_df %>%
  select(c("item", "rating"))

set.seed(1729)
# train-test split

counts <- count(df, item) %>%
  filter(n > 20)

df = merge(counts, df, by="item")
in.train = sample(unique(df$item), size = 0.6*length(unique(df$item)))
df.train = filter(df, item %in% in.train)
df.rest = filter(df, !item %in% in.train)
in.val = sample(unique(df.rest$item), size=0.5*length(unique(df.rest$item)))
df.val = filter(df.rest, item %in% in.val)
df.test = filter(df.rest, !item %in% in.val)

# alphas taken from train set (see amazon_experiment.R for how to generate this)
alphas = c(18.41, 4.78, 7.57, 21.55, 309.98)

# testing step 1: order results by timestamp

df.test <- filter(raw_df, item %in% df.test$item) %>%
  group_by(item) %>%
  arrange(timestamp, .by_group = TRUE)

# get 70th percentile value for each product
quant_df <- group_by(df.test, item) %>%
```

```r
  summarize(q70 = quantile(timestamp, .7))

df.test <- inner_join(df.test, quant_df, by="item")

#split test set by percentile
df.test_learn = filter(df.test, timestamp <= q70)
df.test_eval = filter(df.test, timestamp > q70)

df.test_learn <- dcast(setDT(df.test_learn), item ~ rating, fun.aggregate = length) %>% arrange(item)
df.test_eval <- dcast(setDT(df.test_eval), item ~ rating, fun.aggregate = length) %>% arrange(item)

colnames(df.test_learn) <- c("item", "n1", "n2", "n3", "n4", "n5")
colnames(df.test_eval) <- c("item", "n1", "n2", "n3", "n4", "n5")

df.test_learn$n = df.test_learn$"n1" + df.test_learn$"n2" + df.test_learn$"n3" + df.test_learn$"n4" + d
df.test_eval$n = df.test_eval$"n1" + df.test_eval$"n2" + df.test_eval$"n3" + df.test_eval$"n4" + df.tes

df.test_learn <- mutate(df.test_learn, b1 = n1 + alphas[1], b2 = n2 + alphas[2], b3 = n3 + alphas[3],
                        b4 = n4 + alphas[4], b5 = n5 + alphas[5], b = n + sum(alphas))

#frequentist estimate
df.test_learn_frequentist <- transmute(df.test_learn, n1 = n1 / n, n2 = n2 / n, n3 = n3 / n, n4 = n4 / 
#bayesian estimate
df.test_learn_eb <- transmute(df.test_learn, b1 = b1/b, b2 = b2/b, b3 = b3/b, b4 = b4/b, b5 = b5/b)

# actual distributions of 30% held-out data
df.test_eval <- transmute(df.test_eval, n1 = n1 / n, n2 = n2 / n, n3 = n3 / n, n4 = n4 / n, n5 = n5 / n

# better metric to look at? This basically says for prediction of future values, freq and eb are same
mse_freq = sum((df.test_learn_frequentist - df.test_eval)^2) / (nrow(df.test_eval)*ncol(df.test_eval))
mse_eb = sum((df.test_learn_eb - df.test_eval)^2) / (nrow(df.test_eval)*ncol(df.test_eval))

# let's plot a scatter plot of number of datapoints and mse

df.test_learn$mse_f = rowSums((df.test_learn_frequentist - df.test_eval)^2)/5
df.test_learn$mse_b = rowSums((df.test_learn_eb - df.test_eval)^2)/5

a <- ggplot(df.test_learn, aes(x=n, y=mse_f)) + geom_point()
b <- ggplot(df.test_learn, aes(x=n,y = mse_b)) + geom_point()

library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
grid.arrange(a, b, ncol=2)
```





## Next steps

- How do we deal with the “missing reviews” problems in this model? i.e. if, when training the prior, one product doesn’t have some any reviews of one rating, what should we do?
- Is this preliminary approach of calculating MSE really the best way to evaluate the performance of this model? Ultimately, we want something that approximates an overall “goodness” of a product, and we’re currently trying to express that goodness as 5 weights over each rating category.