Base Pricing Analysis and Price Elasticity Estimation

Günter J. Hitsch

January 12, 2017

Goal

The goal is to conduct a base pricing analysis. We estimate brand-level demand using scanner data, and then we make profitability predictions corresponding to specific base price changes. We estimate log-linear demand models that use (log) prices and promotions as inputs, and predict log quantities, `log(1+Q)`. We focus on model specifications that include a focal brand for which we predict demand, and we control for (log) prices and promotions of three competitors. Obviously, this approach generalizes to an arbitrarily large number of competing products as long as the sample size is large enough.

Compare the estimation results and profit predictions for the following three brands:

1. **Tide**, the top brand in the liquid laundry detergent category
2. **Tropicana**, the top brand in the refrigerated orange juice category
3. **ReaLemon**, the second brand according to total revenue (after private label) in the lemon/lime juice category

Packages

Make sure to install two packages we have not used before: lfe and stargazer.

```
library(bit64)
library(data.table)
library(lfe)
library(stargazer)
library(ggplot2)
```

Data overview

The data source is the Nielsen RMS retail scanner data set. The data set captures weekly price and quantity data for all products (UPC's) sold in the stores of a large number of U.S. retail chains. The Kilts data do not include all retailers (for example, Walmart is not part of the data), and the identity of the retailers is not revealed. However, we know if a store belongs to the same retail chain.

Brand data

The data.table `brands` in `Brands.RData` includes brand information for the top five brands in four categories (product modules):

```
1036   FRUIT JUICE - LEMON/LIME
1040   FRUIT JUICE - ORANGE - OTHER CONTAINER
7012   DETERGENTS - HEAVY DUTY - LIQUID
```

The data include the brand code and brand description and total revenue calculated across all observations. The top five brands were selected based on total brand revenue.

Store data

See `stores` in the file `Stores.RData`. `store_code_uc` identifies each retail stores. For some (but not all) stores we know the corresponding `retailer_code` that identifies the chain (banner) that the store belongs to. The data include the Scantrack (SMM) market code and the Scantrack market description. Scantrack markets correspond to large metropolitan market areas such as *Chicago* or *Raleigh-Durham* (see the data manual for a map of the Scantrack markets). The three-digit ZIP code of each store is also included.

Movement data

The movement data (`move`) are in the files of the form `brand_move_<module code>.RData`. The data are at the brand/store/week level and include prices and quantities (`units`). The data are aggregates of all UPC's that share the same brand name. Brand prices are measured as the weighted average over all store/week UPC prices in equivalent units, and quantities represent total product volume measured in equivalent units such as ounces. For example, in the orange juice category (module 1040), prices represent dollars per ounce and units are total product volume in ounces per store/week. The aggregation weights are based on total store-level UPC revenue across all weeks, and hence the aggregation weights are constant within each store. The movement data also include a promotion indicator (`promo_dummy`), a logical `TRUE/FALSE` variable.

The `week_end` variable date is the last day of a Nielsen week, which always starts on a Sunday and ends on a Saturday. Note that prices may change during the period, and hence even the UPC-level price may be an average over more than one posted price. The sample includes data for the 2010-2013 period.

Both the full data, and 25 and 50 percent sub-samples are available. The sub-samples were obtained by sub-sampling stores, hence for each included store the whole time series of quantities, prices, and promotions is available.

Please consult the official Kilts Center Retail Scanner Dataset Manual for all details.

Prepare the data for the demand analysis

We first load the brand and store meta data.

```
load("./Data/Brands.RData")
load("./Data/Stores.RData")
```

Select the category and brands

Choose a product category (module code) and select the corresponding brand-level meta data from the data.table `brands`. Then sort (order) the brand data corresponding to total brand revenue, and select the top four brands (ranked by revenue).

```
selected_module = 7012
model_name      = "Detergents-Tide"
```

Let's assign each brand a new name using a new variable, `brand_name`. Call the top-ranked (according to total revenue) brand `own`, and the top three other "competitor" brands `comp_1`, `comp_2`, and `comp_3`.

The point of this is to make the code **reusable**. From now on, after renaming the brands, the code can be run for any other product category without having to manually change the brand names. Of course, the naming scheme can also be trivially adapted to estimate demand for any other brand, not just the top-ranked brand. For example, if you wanted to estimate demand for the top two brands, according to revenue:

Note that we also specified a `model_name` string variable. We can use this variable later to save the output from the analysis in a named file. Here, we indicate the product category and the focal brand, *Tide*.

Prepare the movement data

Load the movement data, and—for better readability—change the variable names from `units` to `quantity` and from `promo_dummy` to `promotion`. Change the data type of the `promotion` variable from `logical` to `numeric` using the `as.numeric` function. Finally, merge the new `brand_name` variable with the movement table.

Remove outliers

Most data contain some "flaws" or outliers. Here is an easy way of removing such outliers:

First, we create a function that flags all observations in a vector `x`, for example a price series, as outliers if the ratio between a value and the median value among all `x` observations is below or above a threshold.

```
isOutlier <- function(x, threshold_bottom, threshold_top) {
   is_outlier = rep(FALSE, times = length(x))
   median_x   = median(x, na.rm = TRUE)
   is_outlier[x/median_x < threshold_bottom | x/median_x > threshold_top] = TRUE
   return(is_outlier)
}
```

Now run this function on the price data, separately for each brand and store. Then tabulate the number of outliers, and remove the corresponding observations from the data set.

I recommend to use a lower threshold (`threshold_bottom`) value of 0.35 and an upper threshold (`threshold_top`) of 2.5.

Reshape the movement data from long to wide format

To prepare the data for the regression analysis, we need to **reshape the data from long to wide format** using `dcast`.

All the details on casting and the reverse operation, melting from wide to long format using `melt`, are explained in the data.table html vignettes: https://rawgit.com/wiki/Rdatatable/data.table/vignettes/datatable-reshape.html.

Let's be specific about the structure of the data that we need to use to estimate a demand model. We would like to obtain a table with observations, characterized by a combination of store id (`store_code_uc`) and week (`week_end`) in rows, and information on quantities, prices, and promotions in columns. Quantities, prices, and promotions are brand-specific. Hence, the structure of the wide-format data.table that we want to create is

`store_code_uc + week_end ~ brand_name`,

and the brand-specific variables that we would like have in the columns are

`value.var = c("quantity", "price", "promotion")`.

Let's use `dcast` to obtain the corresponding data:

```
move = dcast(move, store_code_uc + week_end ~ brand_name,
             value.var = c("quantity", "price", "promotion"))
head(move)
```

Merge store information with the movement data

Now merge the movement data with the store meta data, in particular with the retailer code, the Scantrack (SMM) market code, and the Scantrack market description. But only with the

store meta data where we have a valid retailer code. Hence, we need to remove store data if the retailer code is missing (`NA`). Use the `is.na` function to check if `retailer_code` is `NA` or not.

Check how many stores there are in the data.

Create time variables or trends

Extract the year and month from the week (`week_end`) variable in the movement data (use the functions `year` and `month`) and create a time trend (1, 2, 3, ...) such that 1 corresponds to a week in the first month in the data, 13 corresponds to a week in the 13th month in the data, etc.

Remove missing values

Finally, **retain only complete cases**, i.e. rows without missing values:

```
move = move[complete.cases(move)]
```

Data inspection

Before running the demand models we would like to understand the degree of price variation in the data. Comment on why this is important for a regression analysis such as demand estimation!

Create histograms of own prices, and the ratios of own prices relative to the price of (separately) competitor 1, 2, and 3.

Estimation

Now we are ready to estimate demand models for the "own" brand.

We want to estimate a sequence of models with an increasing number of controls and compare the stability of the key results across these models. In all models the output is `log(1+quantity_own)`.

Let's start with the following models:

1. log of own price as only input
2. Add store fixed effects
3. Add a time trend—maybe linear, or a polynomial with higher-order terms
4. Instead of a time trend add fixed effects for each month (more precisely: for each year/month combination)

Estimate the models using the `felm` function from the lfe package (consult the corresponding notes on Canvas). Store the outputs in some appropriately named variables (objects).

Hint: Recall that it is perfectly legitimate to write model formulas such as

```
log(1+quantity_own) ~ log(price_own)
```

Hence, there is no need to create new variables such as the logarithm of own price, etc., before estimating a demand model.

You can display the regression coefficients using the `summary` function, which is standard. As a much more elegant solution, however, I recommend to use the stargazer package. Here's an easy example of how to use stargazer (note that the `fit` objects are the regression outputs, adjust the names if necessary):

```
stargazer(fit_base, fit_store_FE, fit_trend, fit_month_FE,
          type  = "text",
          column.labels  = c("Base", "Store FE", "Trend", "Store + year/month FE"),
          dep.var.labels.include = FALSE)
```

As you can see, the stargazer automatically displays the estimates for the same variable in the same row. This vastly improves readability and comparability of the estimates across models.

To learn more, consult the stargazer options or http://jakeruss.com/cheatsheets/stargazer.html.

Comment on the results. Does controlling for store or time fixed effects make a difference?

Before moving on, you may want to remove the regression output objects that are no longer used, because they take up much space in memory:

```
rm(fit_base, fit_store_FE, fit_trend)
```

We keep `fit_month_FE` as the current preferred model for comparison.

Controlling for competitor prices

Now add the competitor prices, for competitors 1, 2, and 3. Compare the results and comment on the cross-price elasticities.

Controlling for promotions

Now add the promotions dummies, first just for the own brand, then for all brands. Compare the results. Did controlling for promotions change the own price elasticity estimate in an expected manner?

We will use the final model including all variables (I called it `fit_promo_comp`) as our preferred model. Also, we **save the model output object in a named file in the folder `Results`**. Make sure to create this folder in the same directory where the source R Markdown is located.

```
save(fit_promo_comp, file = paste0("./Results/fit_", model_name, ".RData"))
```

Warning: The estimation output object is typically large, because R includes all the original data inside. To use stargazer, the whole estimation output object is needed. Other summaries can be obtained using only the result of the `summary` function:

```
summary_promo_comp = summary(fit_promo_comp)
```

This summary object is of considerably smaller size than the original output object.

```
print(object.size(fit_promo_comp), units = "Mb")
print(object.size(summary_promo_comp), units = "Mb")
```

Profitability analysis

Now we use our preferred demand estimates and predict the profitability impact for a range of price changes.

To predict profits you need to predict demand using the regression output. However, `felm` does not have an associated predict function, and hence I created my own (see my notes on the lfe package). Make sure to download and source the `predict.felm.R` script.

```
source("./predict.felm.R")
```

I recommend to first create a profit function that takes the following variables as inputs:

- Regression output/object
- Data including the inputs for the demand prediction
- A separate price vector indicating the price of the brand for which you make a prediction
- Unit production cost
- Retail margin

Here is an implementation:

```
predictProfit <- function(fit, move_DT, price, cost, retail_margin) {
   Y      = predict.felm(fit, move_DT)                    # Y is the prediction of log(1+Q)
   profit = (exp(Y)-1)*(price*(1-retail_margin) - cost)  # Important to subtract 1 from exp(Q) = 1+Q!
   return(sum(profit))
}
```

To make the profit predictions, **let's only retain data for one year, 2013**.

Although we have excellent demand data, we do not know the production costs of the brands (this is confidential information). We can infer the cost making an informed assumption on retail margins and the gross margin of the brand.

```
gross_margin  = 0.38
retail_margin = 0.18

cost = (1-gross_margin)*(1-retail_margin)*mean(move$price_own)
```

Now create a vector indicating a percentage price change, say over the range -0.1, -0.08, . . . , 0.06, 0.08, 0.1, corresponding to price changes that are at most 10 percent.

```
percentage_delta = seq(-0.1, 0.1, 0.02)
```

Then **calculate the predicted profits and combine them in a table, then save the table to a named file for later use.**

Plot profits in levels and in ratios relative to the base profit at current price levels.

Summary of analysis

Compare and discuss the estimation results for all brands. Are the results as expected?

Discuss the profitability predictions and the reasons for the differences in the predictions across the brands.