Y.U.M.I - Yelp User's Maturity Index

A way to emphasize relevant users in Yelp®

by Alessio Benedetti - October 27th, 2015

Introduction

Businesses pages on Yelp generally have a high number of reviews. Evaluating them, in order to choose or exclude a particular business, can be time consuming, especially on mobile devices.

The idea behind this paper is to create an index: *the "Y.U.M.I" (Yelp User's Maturity Index)*, which summarizes the "maturity" of each user as well as builds a model to predict its value. By "mature users" we intend users with a consistent contribution to Yelp over time: good business reviewing activity as well as a presence in the yelpers community.

Methods and Data

Premises

Yelp dataset includes five *json* files: `business`, `review`, `user`, `check-in` and `tips`. For a description of the structure of the fields, please refer to "Notes on the Dataset" section, on the Yelp Challenge page.

In our document we will only use two data files: `review` and `user`. We assume that both dataframes are available in RDS flatten format. The sample will be set dynamically without replacement, and will be taken only from the `review` dataframe.

For the full code please refer to the Github repository. This work is licensed under MIT License.

Selection of the fields

The selected/discarded fields are listed in the previous table, together with the relative motivation.

Field	Selected	Motivation
yelping_since	yes	Changeable by the account owner
review_count	yes	Changeable by the account owner
name	no	Not relevant for the index
user_id	yes	Metadata field
friends	yes	Changeable by the account owner
fans	no	Changeable by other users
average_stars	no	Not relevant for the index
type	no	Not relevant for the index
elite	no	Not relevant for the index
votes	no	Changeable by other users
compliments	no	Changeable by other users

(a) user dataframe

Field	Selected	Motivation
user_id	yes	Metadata field
review_id	yes	Metadata field
stars	no	Not relevant for the index
date	no	Not relevant for the index
text	yes	Changeable by the account owner
type	no	Not relevant for the index
business_id	no	Not relevant for the index
votes	no	Changeable by other users

(b) review dataframe

Apart the metadata fields needed to connect data, our criteria was to consider only the fields changeable by the owner.

For example, the `fans` field varies when other users decide to follow or unfollow you. On the other side, the `friends` field contains only the users identified by the owner as friends. The field `elite` is excluded since this value is attributed by Yelp to users who meet particular requirements. Here you can find more details about elite users. The fields `stars` and `average_stars`, are excluded because they already are expression of a rating made by Yelp.

Creation of the working dataset (WD)

Once the field selection is done we can start to build our working dataset.

From exploratory analysis we know that the `user` dataframe already contains the `review_count` data. Unfortunately we're unable to extrapolate the word count from it. The word count can be executed only on the `review` dataframe because is where we have the `text` field.

So initially we need to use the `review` data by counting the number of words for each row, hence for each review. Then we do the aggregation on the word and review counts by `user_id`.

Relation between words and reviews

Next goal is to combine the fields `words` and `reviews`. Apparently one way could be to compute the *simple ratio* (SR) `words`/`reviews`. However, this approach leads to an undesiderable situation where users with only one review jump on top when sorting from higher to lower SR.

An alternative way is to use the *weighted sort* (WS). This kind of sort essentially says this: *if the count column of the sort is very low, assume that the column of interest is roughly the average for the data in question.* Expressed in formula:

$$SR_i = \frac{w_i}{r_i} \qquad WS_i = \frac{w_i}{w_{max}} r_i + (1 - \frac{w_i}{w_{max}})\bar{r} \qquad where\ w = \texttt{words}\ and\ r = \texttt{reviews}$$

Here follow the comparison tables with users sorted by SR and WS, with the evidence that WS should be preferred.

user_id	words	reviews	SR	WS
vbs7oDoBdFJ4fRgZ0Z9SlQ	936.00	1.00	936.00	1.04
G_-On5_kHhR61qlkfvW1GQ	923.00	1.00	923.00	1.04
jFLB2kQLIJSEQ7xZMyl1sQ	923.00	1.00	923.00	1.04
H6fszVMVr0syml5qOmp6RQ	917.00	1.00	917.00	1.04

(a)

user_id	words	reviews	SR	WS
0bNXP9quoJEgyVZu9ipGgQ	1999.00	6.00	333.17	6.00
zTWH9b_ItSdLOK9ypeFOIw	1959.00	4.00	489.75	3.94
kGgAARL2UmvCcTRfiscjug	1020.00	6.00	170.00	3.59
wx12_24dFiL1Pc0H_PygLw	1596.00	4.00	399.00	3.41

(b)

For more details, see the following articles by Peter J. Meyers and Avinash Kaushik.

Need of a preliminary model

It's now important to observe that an immediate merge by `user_id` cannot be done. This is due to the fact that not all of the reviews made by a user, of which total is reported in the `review_count` field of the `user` dataframe, are contained in the `review` dataframe.

user_id	reviews count (as taken by `review`)	reviews count (as taken by `user`)
`--f43ruUt7LBeB3aU74z-w`	12	35
`--_L4WuJxAfQkRtC1e43hg`	5	185
`--OKsjlAThNWua2Pr4HStQ`	6	58

As shown in the previous table, it's clear that even with an initial sample made of the entire `review` dataframe we would have needed to fit a model anyway, due to our current inability to know the complete word count for `user` dataframe users.

Fit the word count model (Model A)

The word count model, called "model A", will be built from our working dataset (WD). The word count will be the outcome, while the review count and weighted sort will be the predictors. We choose a Random Forest algorithm for its performance.

With 10 trees and 5 folds we obtain an accuracy of 0.9619253 and estimated out-of-sample error of 0.0380747.

Creation of the full working dataset (FWD)

Once the "model A" is created, before we can apply it to the `user` dataframe to obtain the full words count for every user, we need to highlight a constraint and an hypothesis:

- constraint: `user_id` contained in the sample must exist also in the `user`
- hypothesis: the weighted sort of a user is constant

The constraint must be applied because we can possibly select a set of `user_id` not available in the `user`dataframe. The hypothesis is made because we're assuming that a user doesn't change his rating behaviour. In other terms he makes reviews on Yelp in a way that keeps his WS constant.

Before applying the word count model we merge the working dataset (WD) with the full `user` dataframe. We take the `WS` field from the first and the `review_count` from the second. The merge is done by `user_id`, and the dataframe we obtain is the full working dataset (FWD).

Predict full word count

In the following table we show the comparison between the WD word and review count, the FWD review count, and the predicted value of the FWD words.

user_id	words WD	words FWD	reviews WD	reviews FWD
3gIfcQq5KxAegwCPXc83cQ	1235	1134	4	1155
ia1nTRAQEaFWv0cwADeK7g	986	1134	4	1353
kGgAARL2UmvCcTRfiscjug	1020	1091	6	1971

Adding the remaining fields of interest

Now that we have our full working dataset we can add the other two fields of interest `friends` and `yelping_since`. The field `friends` is changed: from a list of `user_id`'s friends to a count. The field `yelping_since` is also changed: from a string with format `YYYY-MM` to a difference in days, between the actual sysdate and the date the user signed up in Yelp.

Building the metrics

Once the FWD is complete our next goal is to establish the metrics. With "metric" we intend a new field that expresses the user field value related to the whole dataset (for example a mean, a maximum value etc. . .).

We use again the weighted sort for word and review counts, but we add also the weighted sort of word count and yelping days.

$$WS_i^r = \frac{w_i}{w_{max}} r_i + (1 - \frac{w_i}{w_{max}})\bar{r} \qquad WS_i^y = \frac{w_i}{w_{max}} y_i + (1 - \frac{w_i}{w_{max}})\bar{y} \qquad where\ w = \texttt{words}\ and\ y = \texttt{yelping}$$

Our choice of the metric will be a normalization related to the maximum value of the field, in formula:

$$M_{1i} = \frac{WS_i^r}{WS_{max}^r} \qquad M_{2i} = \frac{WS_i^y}{WS_{max}^y} \qquad M_{3i} = \frac{friends_i}{friends_{max}}$$

With this choice we obtain indicators ranging from 0 to 1, that can be easily read as a percentage.

YUMI percent

Now is the moment to define the percent maturity index for the *i-eth* user as a weighted sum:

$$YUMI_i^p = \sum_{k=1}^{3} \alpha_k M_{ki} \qquad where \qquad \sum_{k=1}^{3} \alpha_k = 1 \; and \; 0 \leq \alpha_k \leq 1$$

With the weight vector, alpha vector, set to $\alpha\,(0.5, 0.4, 0.1)$, we obtain the top and bottom three records shown in the following table, sorted by `YUMI_P` field in descending order:

The alpha vector coefficients was choosen to exalt the WS fields.

user_id	words	reviews	friends	YUMI_P
kGgAARL2UmvCcTRfiscjug	1091.00	1971	2224	0.95
0bNXP9quoJEgyVZu9ipGgQ	1731.00	760	650	0.71
zTWH9b_ItSdLOK9ypeFOIw	1091.00	988	749	0.69

(a) Top YUMIs percent

user_id	words	reviews	friends	YUMI_P
M85UVtvhnvKfO1Bz-1xINw	658.00	1	0	0.20
G_-On5_kHhR61qlkfvW1GQ	906.00	1	0	0.19
vbs7oDoBdFJ4fRgZ0Z9SlQ	911.00	10	2	0.18

(b) Bottom YUMIs percent

YUMI predictive model (Model B)

Finally we can build the model to predict the YUMI percent value. We set the `words`, `reviews`, `friends` and `yelping` as predictors, and `YUMI_P` as outcome.

We choose again a Random Forest algorithm, and with 10 trees and 5 folds we obtain an accuracy of 0.9777299 and estimated out-of-sample error of 0.0222701.

Results

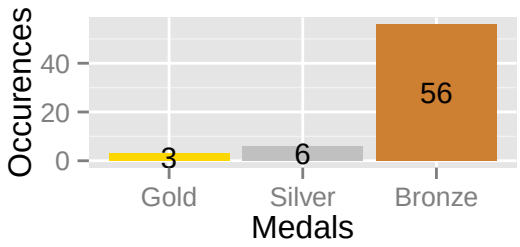
The second model "model B", let us rate individual Yelp users by feeding the predictors.

An immediate application would be by taking data from a yelp user web page. The only drawback is that we need to find automated scrapers to extract several reviews at the time from the web.

With the YUMI-p index, we can also establish a ranking. We can define three clusters (medals), gold, silver & bronze, based on the range of YUMI-p values. We call this new field YUMI-c, which stands for *YUMI class*. Its value is a percentage of a normalized distance `d`. The distance is the ratio between the number of FWD records whose `YUMI-p` value is under the index mean, and the total number of records. In formula:

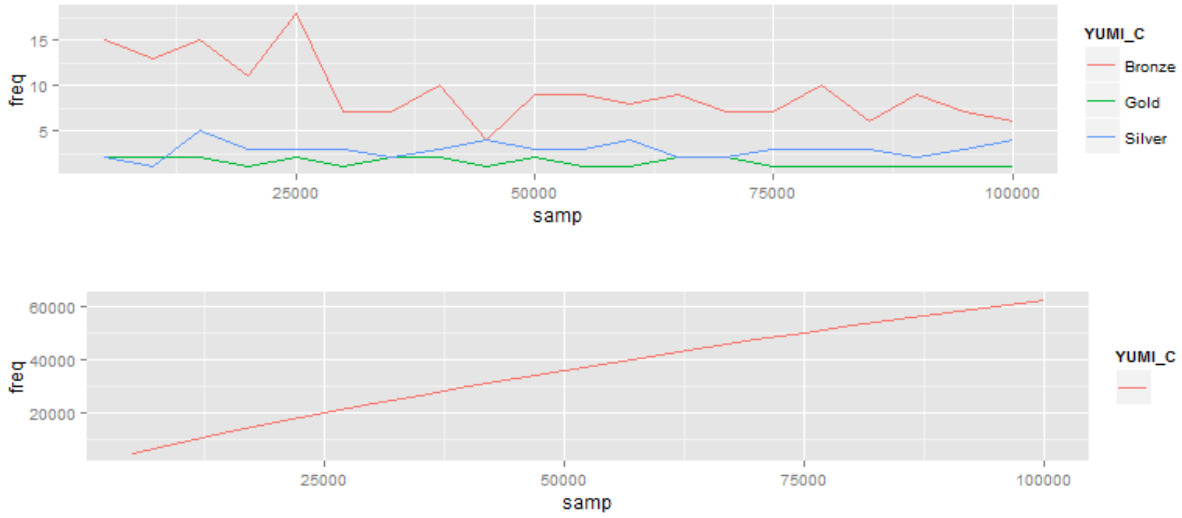
$$d = \frac{N_{rec}^{FWD}|_{YUMI_p < \overline{YUMI_p}}}{N_{rec}^{Tot}} \qquad \begin{cases} if \; YUMI_p \geq 0,9d \; then \; YUMI_c \rightarrow "Gold" \\ if \; 0,7d \leq YUMI_p < 0,9d \; then \; YUMI_c \rightarrow "Silver" \\ if \; 0,5d \leq YUMI_p < 0,7d \; then \; YUMI_c \rightarrow "Bronze" \end{cases}$$

With the `d` we intended to relate top users (e.g 90%) to users under the YUMI-p mean, and not to the entire interval of variation.



Discussion

In this section we make some conclusive considerations of the results obtained. By keeping the alpha vector fixed (every value equal to $\frac{1}{3}$), and by looping on the sample `size` parameter in the interval `[5000 to 100000]`, we can plot a distribution of YUMI-c.



We can observe that in the selected interval, the number of medals is not growing with the increase of the sample size. This mean that for a high population, we may have a low probability to encounter high rated YUMI users (users with medals).

In relation of the initial question, this may be an undesiderable effect because high rated YUMI users wouldn't emerge among others when evaluating businesses.

This effect may be countered by reducing the sample size, for example by taking only users from one country, or by assigning higher weight to the coefficients of the alpha vector that raises YUMI-p.