Zappos Summer 2017 Intern Challenge

Name: Sanket Patel

Email: gets.sanket.patel@gmail.com

University of Texas at Dallas

Major: Business Analytics

Date: 6th Feb 2017

Contents

PART I - Brainstorm Factors

1. **First time visitor to the site:** has never shopped with Zappos before
    - Cache and cookies stored in the browser: if a customer comes back to visit the website in a shorter time, we can use the cookies stored in the browser to give the recommendation. It is a broad aspect but it will definitely works. We need to handle many things while taking cache and cookies into account.
    - Most trending items recently: We can use the most recent trending items to suggest to the customer. We can suggest this by just simply sorting the best product from each category.
    - Seasonality and festivals (Ex. Valentine's Day, Christmas, Thanksgiving): We can suggests the products based on the current season, weather, festivals, etc.
    - New releases: We can also market the new release products which is the recent fashion
2. **Returning customer:** made their first purchase three months ago, has repurchased once since
    - Recommend the better version of the newer one
    - Additional product that might be useful for them related to their previous purchase (If a customer has bought the mobile, it is better to suggest the backcover of that mobile, SD card, screen-guard etc.)
3. **Loyal customer:** has been shopping with Zappos for eight years and has ordered over two hundred products
    - Number of times different products have bought (Purchase history)
    - Major Interest: Finding the major interest depending on their purchases

PART II - Select Products

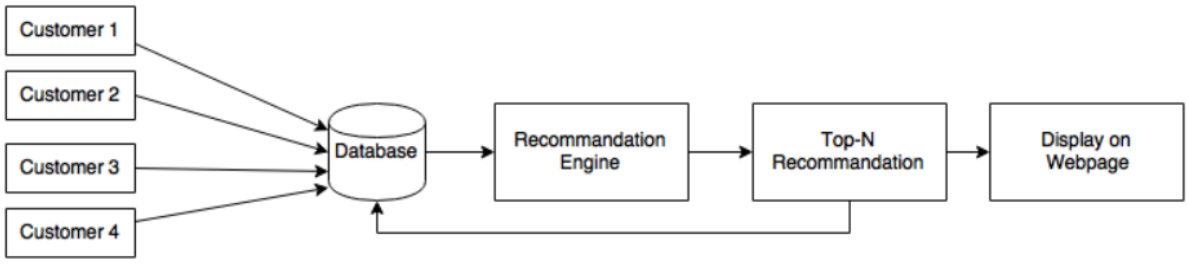
Major Assumptions

1. Record the visits or clicks: It is assumed that when the customer visits the website, we record every moment of his on the server. It includes, number of seconds it stays on the product page, number of visits it has made before purchasing.
2. The dataset that I have taken is meant for the different purpose but I have modified it so that I can use it to make a recommendation engine.
   File: u.data: The full u data set, 100000 visits by 943 users on 1682 items. Each user has visited at least 20 items. Users and items are numbered consecutively from 1. The data is randomly ordered. This is a tab separated list of user id | item id | visits | timestamp.
   User id: User id of the customer.
   Visits: Number of visits the customer has made on a product.
   The time stamps are unix seconds since 1/1/1970 UTC.

User Id	Product Id	Visits	Timestamp
1	314	4	1789520
2	426	3	1789860
…	…	…	…
…	…	…	…

System Design

The implementation has been done on python 2.7. The code and supported file are provided on the link shown below.



All the customer data are being stored by the server to the database. We fetch the necessary details required that is need to recommend the product to the customer. We use number of visits made by each user on a product. We assume that if a customer visits a product more frequently, he/she tend to buy that product. So, we run the algorithm on that data and find the best suitable product for that customer. We then display it to the webpage or store it to the database depending on the requirement.

Algorithms

I have used two widely-used distance calculation methods which are

1. **Item based collaborative filtering**

$$sim_{xy} = \frac{\sum_{u \in U_{xy}} (r_{ux} - \overline{r_u})(r_{uy} - \overline{r_u})}{\sqrt{\sum_{u \in U_{xy}} (r_{ux} - \overline{r_u})^2} \sqrt{\sum_{u \in U_{xy}} (r_{uy} - \overline{r_u})^2}}$$

Where

- U_{xy}: Set of users who have visited both items x and y
- r_{ux}: User u's visit for product x
- r_{uy}: User u's visit for product y
- $\overline{r_u}$: User u's average item visit

$$pred_{up} = \frac{\sum_{i \in I_u} sim_{ip} r_{ui}}{\sum_{i \in I_u} sim_{ip}}$$

Where

- I_u: Set of items visited by the user of interest which are nearest to the item of interest
- sim_{ip}: The similarity between the item of interest and the other item
- r_{ui}: The user of interest's visit for the item

2. **User based collaborative filtering**

$$sim_{ab} = \frac{\sum_{p \in I_{ab}} (r_{ap} - \overline{r_a})(r_{bp} - \overline{r_b})}{\sqrt{\sum_{p \in I_{ab}} (r_{ap} - \overline{r_a})^2} \sqrt{\sum_{p \in I_{ab}} (r_{bp} - \overline{r_b})^2}}$$

Where

- p: Product set $\{p_1 \dots p_m\}$
- I_{ab}: Set of products visited by both users a and b
- r_{ij}: User i's visit for product j
- $\overline{r_i}$: Average visit of user i

$$pred_{ap} = \overline{r_a} + \frac{\sum_{b \in N} sim_{ab} \times (r_{bp} - \overline{r_b})}{\sum_{b \in N} sim_{ab}}$$

Where

- $\overline{r_a}$: Average visit of the user of interest
- b: A user other than the user of interest
- N: Nearest neighbor set
- sim_{ab}: The similarity between the user of interest and the other user
- r_{bp}: The other user's visit for the product
- $\overline{r_b}$: the other user's average visit

Both these methods are similar to each other. These methods are nothing but the calculation of the distance between two products those are of user's interest. If two products are very similar in terms of other customer's interest, they will have the higher similarity.
By implementing this mathematical equation on the data, we can make the prediction on the customer that he/she will likely to make visit on the similar products. So, we suggest/recommend them to do so by showing them on the Home page.

Please review the Python code file **recommendation.py** to see the actual implementation.

Implementation

The implementation has been done on python 2.7. The code and supported file are provided on the link shown below.

Github: https://github.com/sanket9192/ZapposChallenge

The main objective of this project is to give the core algorithm of the whole recommendation system. There are things which can be done before and after this process of the implementation.

- It is assumed that the data are kept in appropriate format. Many times, we need to prepare the data to make it eligible to pass to the recommendation engine.
- It is also a good question how often we need to run the recommendation engine so that recommendations keep updated. It is a trade-off between resources, efficiency, and requirements.
- Prepare the whole automation system is also a bigger task that takes care of generating the recommendations time to time, storing it to database and showing it to the customer.
- Actual data is having much more complicated than this in terms of number of rows and columns. We might need to think of the big data computation as well while creating the system.

PART III - Improve your recommendations

Evaluation metrics for recommender systems can be divided into four major classes: 1) Predictive accuracy metrics, 2) Classification accuracy metrics, 3) Rank accuracy metrics and 4) Non-accuracy metrics.

We will talk about only Classification accuracy metrics.

	Relevant	Irrelevant	Total
Recommended	Tp	fp	tp+fp
Not Recommended	Fn	tn	fn+tn
total	tp+fn	fp+tn	

This is the contingency table or confusion matrix that accumulates the numbers of true/false positive/negative recommendations.

Classification accuracy metrics try to assess the successful decision making capacity (SDMC) of recommendation algorithms. They measure the amount of correct and incorrect classifications as relevant or irrelevant items that are made by the recommender system and are therefore useful for user tasks such as finding good items. SDMC metrics ignore the exact rating or ranking of items as only the correct or incorrect classification is measured. This type of metric is particularly suitable for applications in e-commerce that try to convince users of making certain decisions such as purchasing products or services. A comprehensive overview of classic, basic information retrieval metrics, such as recall and precision and further improved metrics, is given in the survey by Powers. The following short summary is based on his terminology and descriptions for these metrics.

The common basic information retrieval (IR) metrics are calculated from the number of items that are either relevant or irrelevant and either contained in the recommendation set of a user or not. These numbers can be clearly arranged in a contingency table that is sometimes also called the confusion matrix (see the above Table).

Precision or true positive accuracy (also confidence in data mining) is calculated as the ratio of recommended items that are relevant to the total number of recommended items:

$$\text{precision} = tpa = \frac{tp}{tp+fp}$$

This is the probability that a recommended item corresponds to the user's preferences. The behaviour of this (and the following) IR metrics can be observed in Figure 1 which shows a potential ranking of four relevant items in a set of ten items by a recommender.

Recall or true positive rate (also called sensitivity in psychology) is calculated as the ratio of recommended items that are relevant to the total number of relevant items:

$$\text{recall} = \text{tpr} = \frac{tp}{tp+fn}$$

This is the probability that a relevant item is recommended. The two measures, precision and recall, are inversely related which we notice if we vary the size of the set of recommendations. In most cases, increasing the size of the recommendation set will increase recall but decrease precision. Because of this mutual dependence, it makes sense

to consider precision and recall in conjunction with two other metrics that are called fallout and miss rate.

Fallout or false positive rate is calculated as the ratio of recommended items that are irrelevant to the total number of irrelevant items:

$$\text{fallout} = \text{fpr} = \frac{fp}{fp+tn}$$

This is the probability that an irrelevant item is recommended.

Miss rate or false negative rate is calculated as the ratio of items not recommended but actually relevant to the total number of relevant items:

$$\text{missRate} = \text{fnr} = \frac{fn}{tp+fn}$$

This is the probability that a relevant item is not recommended. Just as precision and recall describe the recommender's performance regarding the true positives, two similar metrics can be defined that measure how the algorithm behaves regarding true negatives. Since this corresponds to interchanging the true and false values of the underlying data for the precision and recall metric, they are called inverse precision and inverse recall.

Inverse precision or true negative accuracy is calculated as the ratio of items not recommended that are indeed irrelevant to the total number of not recommended items:

$$\text{inversePrecision} = \text{tna} = \frac{tn}{fn+tn}$$

This is the probability that an item which is not recommended is indeed irrelevant.

Inverse recall or true negative rate (also called specificity) is calculated as the ratio of items not recommended that are really irrelevant to the total number of irrelevant items:

$$\text{inverseRecall} = \text{tnr} = \frac{tn}{fp+tn} = 1 - \text{fpr}$$

This is the probability that an irrelevant item is indeed not recommended.

The testing accuracies are shown in the code and it shows if the recommended products have been visited or not. As we can see from the output, it has been very good. But it only covers the 1 stage of the recommendation system. It only shows that if the visitor has visited the product those are recommended or not.

There are many other ways through which we can improve the recommendations as well.