Review Union

Parag Jain Saurabh Sawant
University of Illinois, Urbana Champaign
{pjain11,ssawant2}@illinois.edu

1. Abstract

Customer reviews are a very important feature of eCommerce websites. Customers rely heavily on these to make buying decisions. Because of their importance, these are also used widely in research and industry for different applications. Within research community, Opinion Integration, Opinion Analysis and Opinion Summarization are active research areas that require updated product review data sets. Similarly in industry, several consumer applications are built on the top of product reviews relying on reliable and continuously updated review datasets. In this project we propose and develop ReviewUnion mobile application which fetches the the product information from Walmart and Amazon to help customers compare product information from different website and also have a quick reference to review and feedback of that product. We have used the opinion summarization algorithm called Opinosis [3] to provide the crisp reviews which can be very beneficial for decision making.

2. Introduction

Shopping is getting reviews driven. The feedback from the buyers provides the trustworthy and helpful source of information which can help subsequent buyers. If we want to check reviews of a product, we have to search a product in one of the top shopping sites like Walmart, Amazon or Target. These are the websites which hold lots of products and it provides a lot of redundant information if a user just wants to check the reviews. Searching a product in these sites is a cumbersome and time consuming task. In addition, we need to make sure we do not miss a review which can reveal a significant problem about the product.

We have crawled Walmart Shopping websites to store around 3.7 million product and reviews of each one of them. We are utilizing the review crawling implementation of shopadvisor study done under Professor Zhai. The product details from amazon website are fetched on the fly when the product is searched in the application. Using the barcode entity resolution mechanism this product is matched with already crawled products from walmart. And finally the summarized reviews for this products (using the Opinosis algorithm implementation [3]) and comparison information is displayed via the mobile application.

The study project has given insights into efficient product crawling and product entity resolution technique which can be used for extending studies on multiple sites and domain applications.

3. Related Work

To our best knowledge we were not able to find out exactly similar tool. However, there are tools or websites which concentrates on this problem partially. There is a website epinions.com[3] but it does not collect reviews from different sites, users have to post reviews of product on that site.

4. Problem Definition

Nowadays, even if we want to buy a light bulb, we check the reviews of that product to see what people have to say about it, just to make our decision concrete. If we want to check reviews of a product, we have to search a product in one of the top shopping sites like Walmart, Amazon or Target. These websites hold lots of products and it provides a lot of redundant information if a user just wants to compare the same product on different website and check the reviews.

The primary source of consuming information are becoming handheld devices like smartphones and tablets. This devices bring in their own challenges as the screen is small how can we present a big set of review information in a summarized form. So that its easy to digest such information and is helpful in quick decision making.

For ReviewUnion, we focus on three dimensions of the problem namely

1. Product information crawling from different websites
2. Entity resolution to match similar products from different websites
3. Review summarization

Some of the challenges in building this system includes a way to identify the products with different names. For example at one website it may be by the name “iphone 5s” while another may have “iphone 5s Gold” or “iPhone 5s Silver”. The scope of a project is itself a challenge. Every website like Walmart, Amazon or Target has different website structure. So, crawling logic of one website will not be applicable to another website.

5. Methods

Our system comprise mainly of four components - A walmart product crawler, review crawler, database to store product information and an Android application to show product comparison and reviews in summarized form.

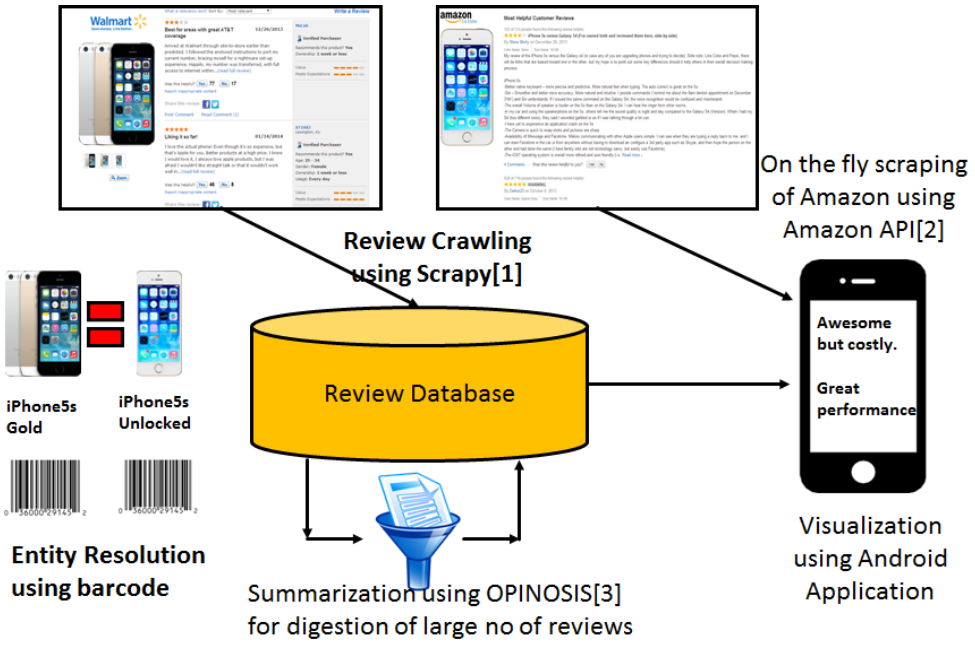

Figure 1: Flow diagram of Review Union

Now lets understand the different components one by one -

5.1 Walmart Product Details Crawler -

This section explores various strategies to crawl a huge product website like Walmart in an efficient and fast manner. We chose scrapy [1] framework written python for this purpose as it is highly customizable, well documented, fast and easy to work with. The list of all the product details that was needed to be scraped by this crawler are Product Id, Product Name, Product Walmart URL, Product Price, Product category and Product Image URL.

First, let us understand the structure of walmart site. There are broadly three patterns of links that on walmart site that are interesting for our study –

Type 1. These type of links are the product page links. They contain all the product information that we need to scrape. These links have pattern "***/ip/****". These links are of form `http://walmart.com/<product-name>/<product_id>`

Type 2. These type of links are home page for a particular department (e.g. Television) or contains links to products of a particular category. These links have pattern "****/cp/****". These type of pages contains links of **Type 3.** for same category.

Type 3. These type of links contains product listing obtained after applying a filter condition. They have pattern "****/browse/*****". These pages contains a list of products and links of **Type 1.** to actual product pages

For our case (can be used in general case also) we have identified three basic requirements that the crawling strategy should satisfy in order to be useful. These requirements are -

1. **Speed (Time to convergence) -** An optimal speed need to be chosen which does not puts load on server along with being fast. It should not have worst case response time in order of several weeks or months since there are around 4 million products.
2. **Efficiency -** The crawling strategy should be efficient in terms of memory. It should not cause memory overflow issues.
3. **Server Load -** The crawling strategy should put less load on the target server i.e. walmart website in our case, so that it can prevent itself from getting blocked.

Now, lets explore the various crawling strategies and discuss their pros and cons -

5.1.1 First Strategy

The first strategy that we tried is the very basic strategy of using a Breadth First Search (BFS) to explore all the different product categories and ultimately the product pages starting from an index page such as http://www.walmart.com/cp/All-Departments/121828. This strategy was not used because of various reasons -

1. Keeping all the visited links in memory can be expensive as there are millions of products. It can potentially cause memory overflow problem.

2. The product list page displays only a subset of all products at one go – 32 or 60 products per page. Therefore, to crawl all the products pagination should be dealt with which causes spider to slow down.
3. Since many products can be present in more than one category this strategy will try to crawl the same product more than once, so it puts extra load on the server.

5.1.2 Second Strategy

This strategy is to start with a page like www.walmart.com/browse/0/0 whicc list all of the products present on the walmart website in pagination form. The spider will go through all the pages one by one and crawl all the products present on each page. A fully functional spider for this strategy is implemented in scrapy but it was not chosen because of following problems -

5.1.2.1 Problems with this strategy

1. While crawling the website using this strategy a lot of time-outs were happening. The response time of server was eventually becoming slow after going through many pages.
2. Another problem with this strategy is that we are crawling more pages than necessary. By default the product listing webpage presents 32 product links at a time. So, for crawling each 32 products we are crawling one more extra page. Thus, if we extrapolate it to 4 million product pages effectively we are crawling around 125K more pages.

It should have taken a month to complete and also puts load on the server.

5.1.3 Third Strategy

The third strategy is to use sitemap [7] to crawl all the products. A sitemap contains all the pages accessible to users or web crawlers. Therefore, if we can obtain a list of all the product pages it would be very easy and fast to crawl all the products as we are not navigating through index pages instead we are directly downloading an XML file which contains product links. On walmart the sitemap XML for all the product pages is `http://www.walmart.com/sitemap_ip.xml`. This XML file has following structure-

```
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <sitemap>
    <loc>http://www.walmart.com/sitemap_ip1.xml.gz</loc>
    <lastmod>2014-05-10</lastmod>
  </sitemap>
  ....
  ....
  <sitemap>
    <loc>http://www.walmart.com/sitemap_ip75.xml.gz</loc>
    <lastmod>2014-05-10</lastmod>
  </sitemap>
</sitemapindex>
```

As the there are around 4 million products, all the product links are divided in 75 separate compressed XML files. Now each of this XML file has the following structure-

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.walmart.com/ip/product-name/product_id</loc>
    <lastmod>2005-01-01</lastmod>
    ...
  </url>
  ...
</urlset>
```

As can be seen each XML file has several <*url*> tags and within this tag we have the required product page link which we need to extract and parse to scrape the required product information.

5.1.3.1 Sitemap Crawler implementation

Scrapy has inbuilt spider class for crawling sitemap called Sitemap spider [8]. We extended it and implemented a custom parse function for product page links in the sitemap files. In this function we use Xpath expressions to extract the desired information. This spider takes URL of form `http://www.walmart.com/sitemap_ipN.xml.gz` where N ranges from 1 to 75 in our case as the start_url parameter.

Now, lets look at some of the major implementation challenges -

1. **Different Page Structure -** We use Xpath [9] expressions to scrape the required information from the web page. However, we observed that in some of the cases like "Price" the element/node in which the information is present is not the same. To handle this we used exception handling whenever an exception is thrown for wrong Xpath, we try alternate Xpath expression to reach the required node in the except block. If that also fails we populate that field with a default value. An example nested try-except will look like -

```
...
try:
        item['field'] = selector.select('xpath1')[0].extract()
except Exception as e:
        try:
                item['field'] = selector.select('xpath2')[0].extract()
        except Exception as e:
                item['field'] = DEFAULT_VALUE
...
```

2. **Internal Server Errors -** Many times the server was throwing 500 Internal server error which was causing the spider to stop. We realized that the problem was because of broken links. To handle this issue the crawler retries the link 3 times and even after retries if the request fails then the program handles it by checking the response code in the callback parse function. If the HTTP response code is among any one the following – 400, 403, 500, 404 etc. then to prevent this product from being skipped we parse the product page url directly to extract product id and product name only as the urls are of form `walmart.com/ip/<product-name>/<product-id>` and store this information in the database.

Finally this strategy was used for crawling. The advantages of using this strategy are -

1. It is very fast as we are getting the product links directly from the sitemap that way we are skipping the navigation through walmart index or main page.
2. It helps in deciding when to crawl the sitemap again as our spider will run only when the sitemaps are updated. This information can be known using the <*lastmod*> tag in the sitemap.
3. It helps in incremental crawling as new products are added only to the last sitemap file so if any other information like price etc. does not need to be updated then we can only crawl the last sitemap to get new products. This saves a lot of time and computing resources.
4. It is fault tolerant as if spider dies at any point we can restart the spider from the current sitemap instead of restarting it from start.
5. It puts less load on the server because of all the above mentioned points and also because it does not tries to visit any link more than once.

Thus, this crawling strategy satisfies all the three requirement that we mentioned earlier at the starting of the section. Using this spider we were able to crawl around 3.7 million products at an average crawling rate of 450 items/minute in a week's time which is very less as compared to a month's time for first and second strategy.

Table 1: Summary of all crawling strategy

Requirements / Strategy	Speed	Efficiency	Server Load
First	Not fast enough	No	Considerable
Second	Faster than first	Yes	Minimizes to some extent
Third	Fastest	Yes	Minimum

5.2 Database Design

Database design remains a critical aspect of any system for efficient retrieval or access of data as well as making the data processing an easy task. The availability of information in an accurate form makes the data manipulation or analysis logic simpler.
The main objective of the Database design in the study project of ReviewUnion was to keep all the information regarding the products across multiple sources like Walmart, amazon or Target etc. under one roof.

The following major data storage was to be considered.

1. **Product Information:** Information regarding the name of a product, price, a source (web site) from which the product has been considered its rating etc. In short all the primary information about the different products.
2. **Products Reviews:** The reviews of a product is the important domain of the application. There can be several number of reviews of a product and all will be critical from the

functionality of a ReviewUnion project study. For every product we are expecting multiple reviews and may will be beneficial to store multiple review properties like rating(how useful the review was) or date of posting etc. *(This review related data was not stored as it was not considered to be essential for summarizing product reviews)*

3. **Summarized Reviews:** Summarized reviews is the data expected after summarizing the product reviews using opinion summarization algorithm. Here also we expect multiple reviews to be present for a product.

The Database role can be explained in the following component diagram figure 1 and the detailed information and schema of the database has been discussed in the following subsection.

5.2.1 Database Schema

The general schema of the Database consists of two tables mainly PRODUCTS to store product related primary information and PRODUCT_CATEGORY to store the category hierarchy of the product.

1. **Product Information:** The following information is expected in the corresponding fields.

- PRODUCT_ID (BIGINT):Every product in the shopping website like Walmart or Amazon or Target has a unique identifier which is assigned to the product. It is also reflected in the URL when a product is searched. It is useful to identify the product.
- PRODUCT_NAME: (VARCHAR(255)):It is the description name which appears on the product page on a shopping web site.
- MAP_ID: Clubbing of similar products is achieved by assigning similar products a similar ID called as MAP_ID.
- BARCODE (BIGINT): It is 12 or 13 digit unique identification number which is universally used as a reference for product identification. It is also called as UPC (Universal Products Code) or EAN (European Article Number now known as International Article Number which is UPC 12 digit + 1 checksum digit). Every shopping website keeps track of a barcode number for product identification using electronic devices like barcode scanner which are available in mobile phone applications. The only issue is he barcode is not present for all products in same format.
- SOURCE (VARCHAR(255)): Source is a text field which keeps track of source or name of the shopping website from which the product has been considered. If a product is considered from Walmart, then this field will contain “Walmart” or if it is from Amazon or Target, then it will contain either “Amazon” or “Target”.
- SOURCE_LINK (VARCHAR(255)): It is a URL for the product which can be used to crawl the data or reviews for the product.
- RATING: (DECIMAL(3,2)): It is a decimal number which is average of all the ratings given by customers for a product on a 5 star scale. It will be very useful for the customers to decide the product buying based on rating as it reflects the liking of that product among people.

- PRICE (DECIMAL(15,2)): It is a price of a product on the source from which it has been considered. In case of similar products, users can decide purchasing a product from a website which has lowest price.
- IMAGE_ID (VARCHAR(255)): This field contains the image URL from the source website of the product for a quick view of a product. In case of multiple images, the URLS can be "," or ";" or any other delimiter separated.
- LAST_REVIEW_DATE (DATE): This field contains the date at which the reviews are crawled for a product. In case the new reviews get posted for that particular product after this date, then incremental review crawling products logic uses this date and crawls all the reviews which have been posted after this date.
- CATEGORY (VARCHAR(255)): Category of a product contains the ":" delimited category values. The Category of a product is very useful for the advanced search as well as consideration of entity resolution as per category reducing the possible mismatches possible.

2. Products reviews:

Product reviews are stored in a text file and the path for the text file is stored in the corresponding product table. But rather than storing such excessive data, the directory can be fixed and review file name can be named after the product_id like P1.txt

3. Summarized reviews:

Similar to the product reviews, summarized reviews can be stored in a similar way like in a text file with named after a product id.

5.2.2 Barcode Crawling:

Barcode is the unique identification number assigned by a manufacturer for a product universally or within a specific territory. There are different barcode types which are used but the one found on Amazon, Walmart and Target are UPC (Universal Product Code) or EAN13 (European Article Number now known as International Article Number) Walmart uses UPC a 12 digit barcode 5].

Barcode can be used for searching on a shopping website which has a barcode support. So for entity resolution barcode can be very helpful. The barcode can be fetched from a website by finding out the XML tags which hold such information. For example in the case of' Axe Sports Blast' product the XML structure of the page has barcode, we can see that UPC_CODE tag in amazon web source page has barcode information.

Figure 2: UPC Code for Walmart

To get the barcode from Walmart automatically there are two options -

1. Download the source web page for a product using product id and then scrape it.
2. Use Walmart API to get the response and extract the barcode form the returned response [http://i2.walmartimages.com/catalog/getItem.do?item_id=XXXXXXXX]

For the implementation of the ReviewUnion, second approach has been used. As it will not be optimal to download the entire source web page for each product and then extract the specific tag information, it is faster and beneficial to just use API and extract the barcode information. Process flow of the barcode readings:

Approach:
Step 1: Fetch the PRODUCT_ID: Fetch the PRODUCT_IDs for which the barcode has not been fetched during the crawling.

Step 2: Create the URL: For each PRODUCT_ID, form a URL for Walmart API by appending the product id at the id argument place as follows.

Step 3: If PRODUCT_ID = 21187518, then API call for this product id will be will be http://i2.walmartimages.com/catalog/getItem.do?item_id=21187518 and the response will be something like shown in figure 3. We will use the ImagePath node to get the barcode which is higlighted in red box in figure 3.

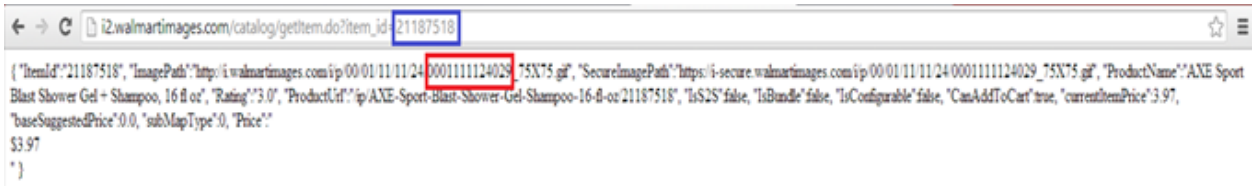

Figure 3: Response of Walmart API

Step 4: Find the barcode using regular expression: The required barcode is the text in red box and we can get it using a regular expressions. Regular expression used for matching the barcode is ".+/(\d+)_.+".

Problem Faced:
Not all products have integer UPC on Walmart making it difficult to use further For example I66NLNVcQ0003 is the UPC a product [13]. This number is not useful as it is not used in the other shopping websites as any reference number. This number seems to be for internal usage. Therefore, we plan to explore more ways for entity resolution in the future work.

5.2.3 Entity resolution:

The entity resolution task is being explored by numerous communities in areas like statistics (probabilistic record linkage), databases and artificial intelligence. In general entity resolution problem is defined as the deduplication, or merge–purge problem in which records determined to represent the same entity are successively located and merged making it robust to spelling mistakes or incomplete names. For example the following records r1 and r2 may represent the same entity as more matching probability due to similar surname.

ID	Name	Phone	E-mail
r1	JohnDoe	819-6190	jdoe@yaho.com
r2	J.Doe	819-6199	
r3	JohnD	819-6199	jdoe@yaho.com

Figure 4 : Records for Entity Matching [12]

The ER problem in ReviewUnion study can be formulated as matching and merging the similar products crawled from different shopping websites. This can be very useful for the review reference as reviews from different web sites can be merged to give the better shopping guidance. As explained in the database section the “iPhone5s” from Walmart and “iPhone5s Unlocked” from Amazon can be merged together.

Entity Resolution Using Barcode:

Barcode serves as a handy unique identifier for matching products across different shopping web sites. We found that barcode serves as a best possible matching technique when the problem involves matching different products from multiple sources. This technique eliminates the product name or price comparison and promises the best matching results removing uncertainty. We match the barcode on the fly as will be explained in the next section.

5.3 Product Review Crawler

We are utilizing review crawling implementation of Shopadvisor study done under Professor Zhai.

5.4 Android Application (Amazon scraping and Entity Resolution)

We implemented our solution as an android application instead of a web application because mobile devices are becoming a main source of information consumption. They also help in consuming information on the fly which is very helpful in shopping scenario. For the product browsing interface we chose to use amazon mobile website because of following reasons -

1. We didn’t want to reinvent the browser interface which is already well done by amazon.
2. Users are familiar with amazon interface so its easy for them to adapt to the application.

Using our product detail and review crawler we have already crawled all the product information and corresponding reviews into a database. For the scope of this project we will deal with a small subset of walmart products to demonstrate our proof concept as we have not setup a web server which can listen to request from apps and serve the walmart products information on the go. We have also preprocessed the summarized reviews for this subset of products and stored all this information along with the app locally.

However, since we are giving amazon interface for browsing therefore we perform scraping of amazon product on the fly when the user selects a product. We are using amazon product advertising api for this purpose. So, when a user selects a product our application extracts the amazon specific product id from the url of that product. Amazon product urls of this form - www.amazon.com/gp/aw/d/<product-id>/... After extracting the product id out application sends a request to amazon server using the advertising api and asks for the product details. A response is received in XML form which we parse to get the required information like Product name, Price, description etc.

However, the Universal product code (upc) is not obtained directly but there is an attribute called 'ItemAttributes' which is returned by api. This attribute contains the upc as a subsequence in the value string for this attribute. This subsequence is repeated twice consecutively in the string and the upc has length of 12 or 13 (in case the website used EAN13 [4] coding) digits. Thus, we used a regular expression to find this repeating subsequence of length 24 and 26 and check whether the first 12 or 13 digits are equal to the last 12 or 13 digits. Among the 13 digit or 12 digit code we use the one which matches as they are almost the same because EAN 13 is equivalent to UPC 12 digit code plus 1 digit of checksum [4]

At the same time when we scrape amazon information we try to perform entity resolution using Universal product code (upc) of the product with our walmart dataset. If we find a matching product we save the walmart product information and summarized reviews along with the amazon information. All the summarized reviews for the product are preprocessed as their are lot of prerequisite stages like POS tagging etc. that needs to be carried on the raw reviews to get the summarized reviews. We thought its not feasible to do it on the fly so we store the preprocessed reviews along with the product information. Finally we display all the information in a tabbed interface to user.

6. Evaluation/Sample Results

Results for crawling the product information - Using this crawler we were able to crawl around 3.7 million products at average crawling rate of 450 items/minute in a week's time.
Results for crawling the reviews - We crawled all the reviews for products in our database.
Results for the android application - We have created a simple and easy to use application to browse products, compare them and see their summarized reviews with minimalistic design. Lets go through the screenshots of the application to understand what we have achieved -

1. Home Page (fig 1.) - User clicks on the select product button or add on top right of the screen-

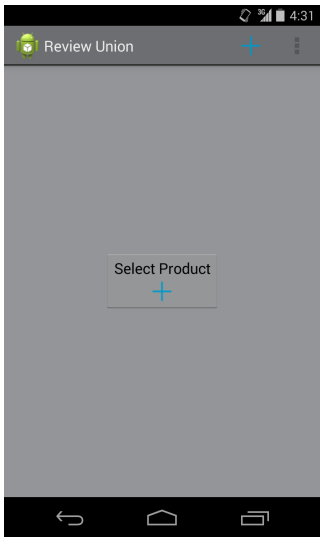

Figure 1. Home page of app

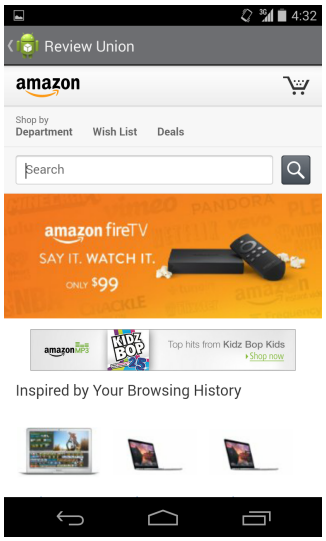

Figure 2. Amazon product browser

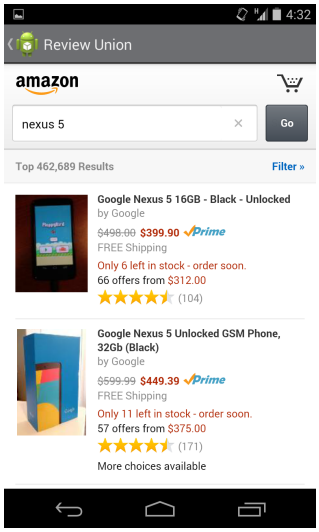

Figure 3. Search Results

2. Amazon product browser page opens up (Figure 2.) and say user searches for “Nexus 5” (Figure 3.) and taps on the first result (Figure 4.)

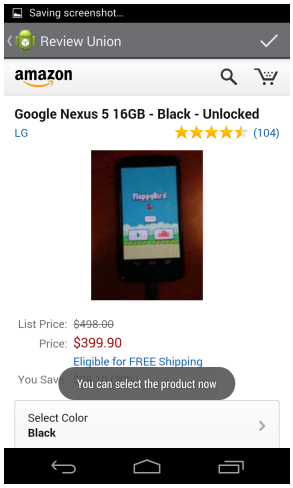

Figure 4. User can select product

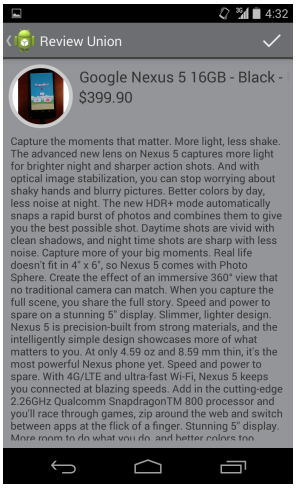

Figure 5. Scraped data

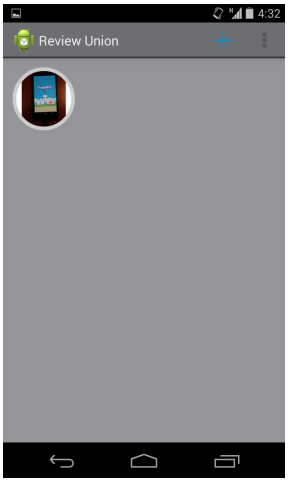

Figure 6. Product on user’s home screen

3. User selects the product by tapping on checkbox on top right (Figure 4.). On the fly scraping is done and scraped data is shown to user (Figure 5.)

4. When the users confirm the product by tapping on the checkbox on top right of the screen (Figure 5.) entity resolution is done and matching walmart product information and summarized reviews are stored along with the scraped information from amazon. The product shows up on the user’s home screen (Figure 6.)

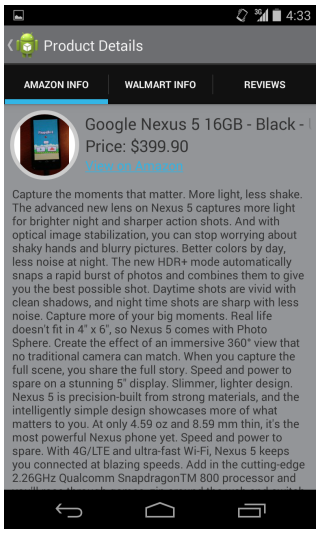

Figure 7. Amazon product Info

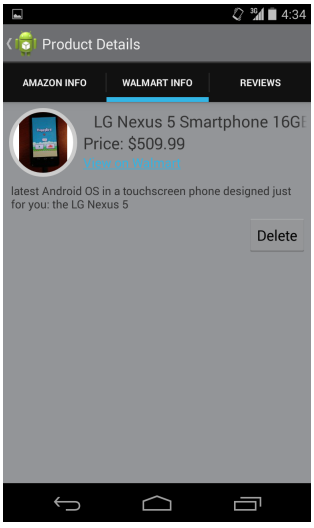

Figure 8. Walmart product info

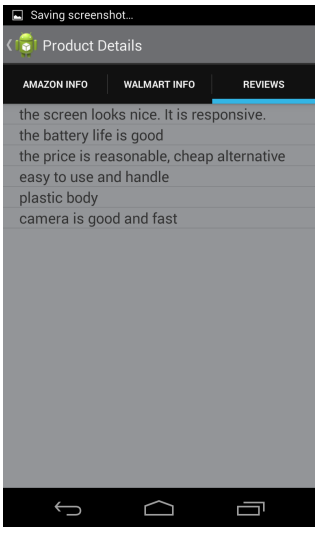

Figure 9. Summarized reviews

5. When the user taps on the product on the home screen (Figure 6.). A tabbed interface will open where user can the amazon info in first tab (Figure 7.), walmart info on second tab (Figure 8.) and summarized reviews on third tab (Figure 9.). Thus, the user can quickly compare product information like Price from different websites and also see summarized reviews for

making quick decisions. If user want to explore more the amazon and walmart product page links are also provided and a delete button is provided if user want to remove this product from its home page.

7. Conclusion & Future Work

The ReviewUnion course project study has been realized in the form of an mobile application which compares the product information across the two shopping websites namely walmart and Amazon. The Application provides the information about description, price from each website and summarized reviews. The Application has a potential to scale across multiple sites as the product entity resolution can work best in case the barcodes are supported by web sites
For future work, we plan to incorporate products from more websites like Target. We also plan to use the string and price matching techniques for entity resolution for products which do not have barcode and can not be considered for the entity resolution using barcode. The use of API which can provide the prices across multiple shopping sites can also be a good incorporation making application more useful.

8. Appendix: Individual Contributions

Parag Jain - Product Crawling, Amazon scraping and Android App
Saurabh Sawant - Database Design, Entity Resolution and Android App.

9. References

[1] Scrapy, scrapy.org
[2] Amazon Product Advertising API,
https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html
[3] Ganesan, Kavita A., Zhai ChengXiang, and Han Jiawei, Opinosis: A Graph Based Approach to Abstractive Summarization of Highly Redundant Opinions, Proceedings of the 23rd International Conference on Computational Linguistics (COLING '10), (2010)
[4] Wikipedia, http://en.wikipedia.org/wiki/International_Article_Number_(EAN)
[5] Wikipedia, http://en.wikipedia.org/wiki/Universal_Product_Code
[6] Wikipedia, http://en.wikipedia.org/wiki/Service-level_agreement
[7] Sitemap, http://www.sitemaps.org/
[8] Scrapy, http://doc.scrapy.org/en/latest/topics/spiders.html#sitemapspider
[9] W3, http://www.w3.org/TR/xpath/
[10]Stackoverflow,http://stackoverflow.com/questions/23327460/500-internal-server-error-scrapy
[11] Epinions, http://www.epinions.com/
[12] Benjelloun, Omar, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. "Swoosh: a generic approach to entity resolution." The VLDB Journal—The International Journal on Very Large Data Bases 18, no. 1 (2009): 255-276.
[13] Walmart, http://www.walmart.com/ip/33093458