

# WTA PROJECT REPORT



## E-Commerce Web Application with Item-to-Item Collaborative Filtering

Submitted by

B S Suhas - 15IT110  
Rakshith G - 15IT134  
A Aditya - 15IT201  
M M Vikram - 15IT217

Submitted to

Ms. Chaitra Bhat  
Assistant Lecturer  
Department of Information Technology  
NITK Surathkal

## Table of Contents

<b>Table of Contents</b>	<b>1</b>
<b>Acknowledgement</b>	<b>2</b>
<b>Introduction</b>	<b>3</b>
<b>Specifications</b>	<b>4</b>
<b>Stages of Development - Methodology</b>	<b>4</b>
1. Web Scraping for product data	4
2. Creation of Database	4
3. Writing the Backend and testing with the frontend	4
4. The Finishing Touches	5
<b>Recommendation Engine</b>	<b>6</b>
Traditional Collaborative Filtering	6
Item-to-Item Collaborative Filtering	7
<b>Results and Analysis</b>	<b>9</b>
<b>References</b>	<b>11</b>

## Acknowledgement

This project consumed huge amount of work and dedication. The completion of this project gives us much pleasure. The implementation would not have been possible if we did not have the support of Ms. Chaitra Bhat.

Therefore we would like to extend our sincere gratitude to her for giving us a good guideline for the project throughout numerous consultations.

## Introduction

Recommendation algorithms are best known for their use on e-commerce Web sites, where they use input about a customer's interests to generate a list of recommended items. Many applications use only the items that customers purchase and explicitly rate to represent their interests, but they can also use other attributes, including items viewed, demographic data, subject interests, and favorite artists.

In this project, we have developed an ecommerce Website that implements an item to item based collaborative filtering as the recommendation algorithm.

Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization established as a 501(c)(3) non-profit.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Some well-known sites that use Django include the Public Broadcasting Service, Instagram, Mozilla, The Washington Times, Disqus, Bitbucket, and Nextdoor. It was used on Pinterest, but later the site moved to a framework built over Flask.

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. For proprietary use, several paid editions are available, and offer additional functionality.

MySQL is a central component of the LAMP open-source web application software stack (and other "AMP" stacks). LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python". Applications that use the MySQL database include: TYPO3, MODx, Joomla, WordPress, phpBB, MyBB, and Drupal. MySQL is also used in many high-profile, large-scale websites, including Google (though not for searches), Facebook, Twitter, Flickr, and YouTube.

## Specifications

For this project we have many technologies. The versions are as mentioned below:

- a. Django 1.11 on Python 2.7
- b. MySQL 5.7.20
- c. Bootstrap 4.0
- d. HTML5
- e. CSS3
- f. jQuery 3.2.1
- g. AJAX

The project was developed and tested on Google Chrome and Mozilla Firefox on a Linux machine.

## Stages of Development - Methodology

### 1. Web Scraping for product data

On observation, Flipkart was chosen for web scraping of products. The Python library, BeautifulSoup was used for this purpose. We scraped data for electronic items, furnitures and books, each having specific attributes. All the data was stored in csv files, so that the database could be populated easily.

### 2. Creation of Database

Django models were used to create the database, with the database engine being MySQL. After the creation, the scraped data was populated into the database.

The database was carefully designed, to accommodate multiple values and various relationships between entities.

### 3. Writing the Backend and testing with the frontend

The next step was to modularly develop each use case and parallelly develop the frontend so that the backend code could be immediately tested.

A test database was created with a few records from the actual database for this purpose.

- First, we worked on the index page display, that is querying from the database and rendering to the front end. Parallely, we also worked on beautifying the page using bootstrap.
- Next, we worked on the 'details' view, that is displaying the details of the product when clicked.
- The template design for this view was worked on with great detail, for displaying multiple images and the specifications for each product.
- Next, the authentication feature was added. Django comes with a handy and easy to use authentication system, which we made use of.
- A 'bootstrapped' template was written for this and also, the registration page was developed parallely.
- Later, the 'Add To Cart ' Feature was developed. We decided to use Ajax for this purpose, as it would make the UX (User Experience) better. We used the HTML Modal to display the cart.
- Next, the checkout feature (Buy Now) was developed, where an invoice was generated containing the products bought by the user. The user is also given an option to print the invoice.
- Later, the Search was feature was implemented. Products could be searched across all the categories, to make navigation easier.
- At the end, the recommendation engine was integrated with the Django backend. Again, AJAX was used to get the recommendations from the backend. This was done to make sure the recommendation engine does not slow down the template rendering. The recommended products, which were got through AJAX, were displayed to the frontend using JavaScript.

#### 4. The Finishing Touches

After the website was fully functional, the styling was fine tuned and also, the backend was optimised. All the features were tested and improvements were made, if any.

## Recommendation Engine

Most recommendation algorithms start by finding a set of customers whose purchased and rated items overlap the user's purchased and rated items. The algorithm aggregates items from these similar customers, eliminates items the user has already purchased or rated, and recommends the remaining items to the user. Two popular versions of these algorithms are collaborative filtering and cluster models. Other algorithms — including search-based methods and our own item-to-item collaborative filtering — focus on finding similar items, not similar customers. For each of the user's purchased and rated items, the algorithm attempts to find similar items. It then aggregates the similar items and recommends them.

### Traditional Collaborative Filtering


A traditional collaborative filtering algorithm represents a customer as an N-dimensional vector of items, where N is the number of distinct catalog items. The components of the vector are positive for purchased or positively rated items and negative for negatively rated items. To compensate for best-selling items, the algorithm typically multiplies the vector components by the inverse frequency (the inverse of the number of customers who have purchased or rated the item), making less well-known items much more relevant.

For almost all customers, this vector is extremely sparse. The algorithm generates recommendations based on a few customers who are most similar to the user. It can measure the similarity of two customers, A and B, in various ways; a common method is to measure the cosine of the angle between the two vectors:

$$\text{similarity}(\vec{A}, \vec{B}) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \bullet \vec{B}}{\|\vec{A}\| * \|\vec{B}\|}$$

The algorithm can select recommendations from the similar customers' items using various methods as well, a common technique is to rank each item according to how many similar customers purchased it. Using collaborative filtering to generate recommendations is computationally expensive. It is  $O(MN)$  in the worst case, where M is the number of customers and N is the number of product catalog items, since it examines M customers and up to N items for each customer.

However, because the average customer vector is extremely sparse, the algorithm's performance tends to be closer to  $O(M + N)$ . Scanning every customer is approximately



$O(M)$ , not  $O(MN)$ , because almost all customer vectors contain a small number of items, regardless of the size of the catalog.

But there are a few customers who have purchased or rated a significant percentage of the catalog, requiring  $O(N)$  processing time. Thus, the final performance of the algorithm is approximately  $O(M + N)$ . Even so, for very large data sets — such as 10 million or more customers and 1 million or more catalog items — the algorithm encounters severe performance and scaling issues.

It is possible to partially address these scaling issues by reducing the data size. We can reduce  $M$  by randomly sampling the customers or discarding customers with few purchases, and reduce  $N$  by discarding very popular or unpopular items. It is also possible to reduce the number of items examined by a small, constant factor by partitioning the item space based on product category or subject classification. Dimensionality reduction techniques such as clustering and principal component analysis can reduce  $M$  or  $N$  by a large factor.

Unfortunately, all these methods also reduce recommendation quality in several ways. First, if the algorithm examines only a small customer sample, the selected customers will be less similar to the user. Second, item-space partitioning restricts recommendations to a specific product or subject area. Third, if the algorithm discards the most popular or unpopular items, they will never appear as recommendations, and customers who have purchased only those items will not get recommendations.


Dimensionality reduction techniques applied to the item space tend to have the same effect by eliminating low-frequency items. Dimensionality reduction applied to the customer space effectively groups similar customers into clusters; as we now describe, such clustering can also degrade recommendation quality.

## Item-to-Item Collaborative Filtering

Rather than matching the user to similar customers, item-to-item collaborative filtering matches each of the user's purchased and rated items to similar items, then combines those similar items into a recommendation list. To determine the most-similar match for a given item, the algorithm builds a similar-items table by finding items that customers tend to purchase together. We could build a product-to-product matrix by iterating through all item pairs and computing a similarity metric for each pair. However, many product pairs have no common customers, and thus the approach is inefficient in terms of processing time and memory usage.

The following iterative algorithm provides a better approach by calculating the similarity between a single product and all related products:

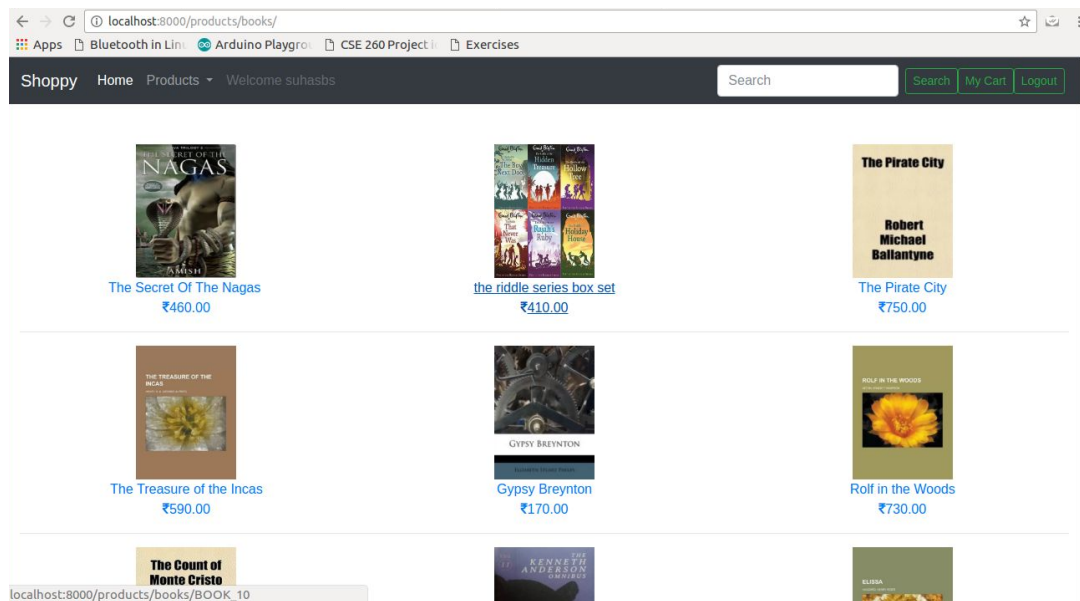




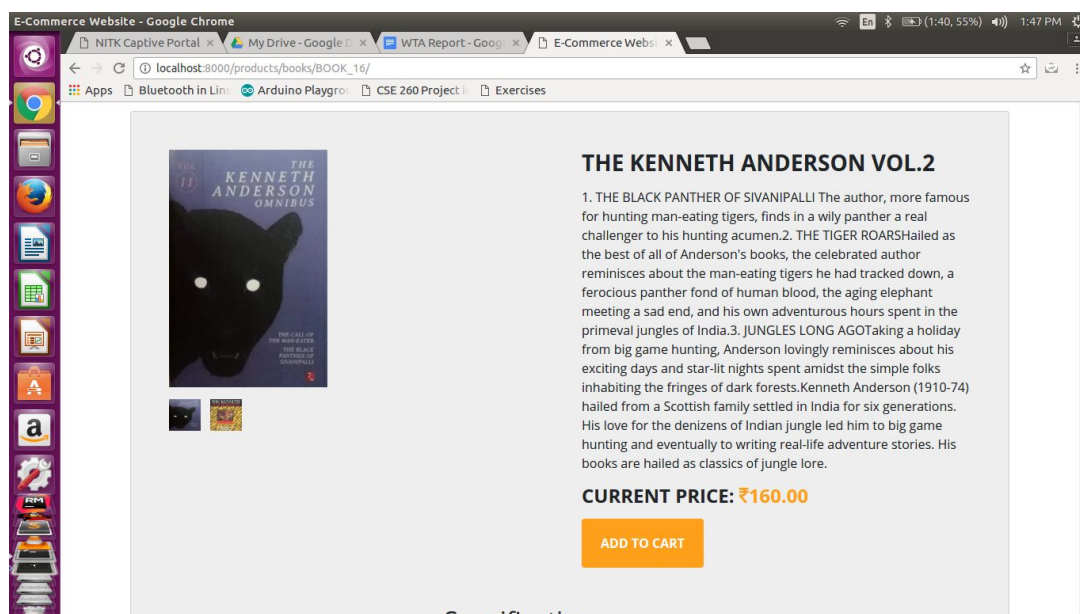
```
For each item in product catalog,  $I_1$ 
  For each customer C who purchased  $I_1$ 
    For each item  $I_2$  purchased by
      customer C
        Record that a customer purchased  $I_1$ 
          and  $I_2$ 
    For each item  $I_2$ 
      Compute the similarity between  $I_1$  and  $I_2$ 
```

It's possible to compute the similarity between two items in various ways, but a common method is to use the cosine measure we described earlier, in which each vector corresponds to an item rather than a customer, and the vector's  $M$  dimensions correspond to customers who have purchased that item. This offline computation of the similar-items table is extremely time intensive, with  $O(N^2M)$  as worst case. In practice, however, it's closer to  $O(NM)$ , as most customers have very few purchases. Sampling customers who purchase best-selling titles reduces runtime even further, with little reduction in quality. Given a similar-items table, the algorithm finds items similar to each of the user's purchases and ratings, aggregates those items, and then recommends the most popular or correlated items. This computation is very quick, depending only on the number of items the user purchased or rated.

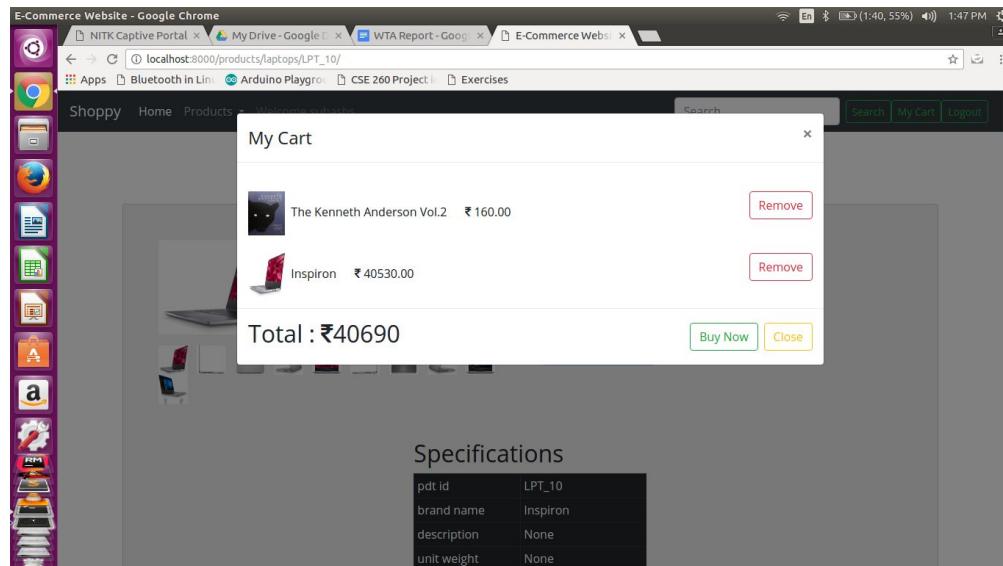
## Results and Analysis



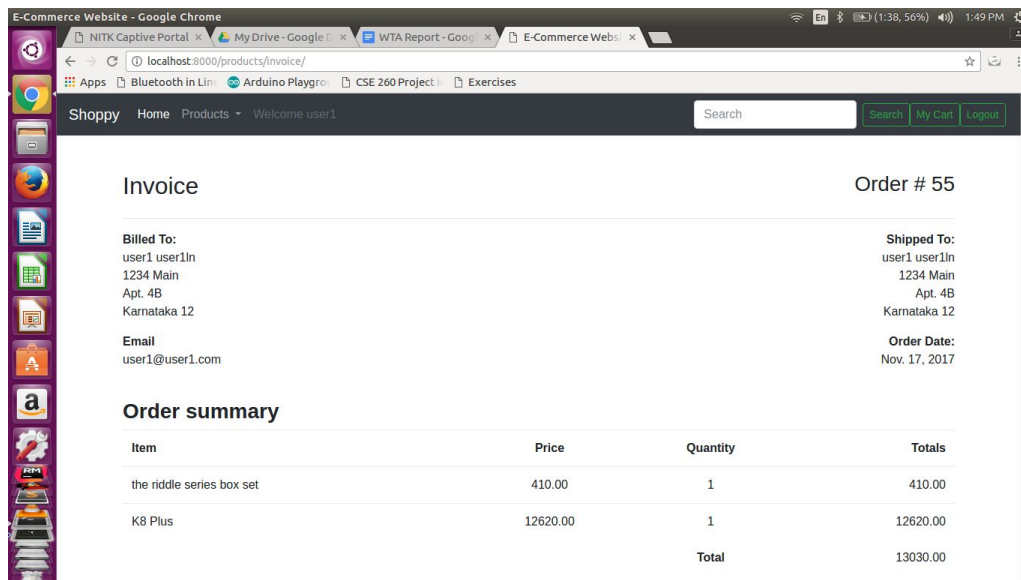
### Index View



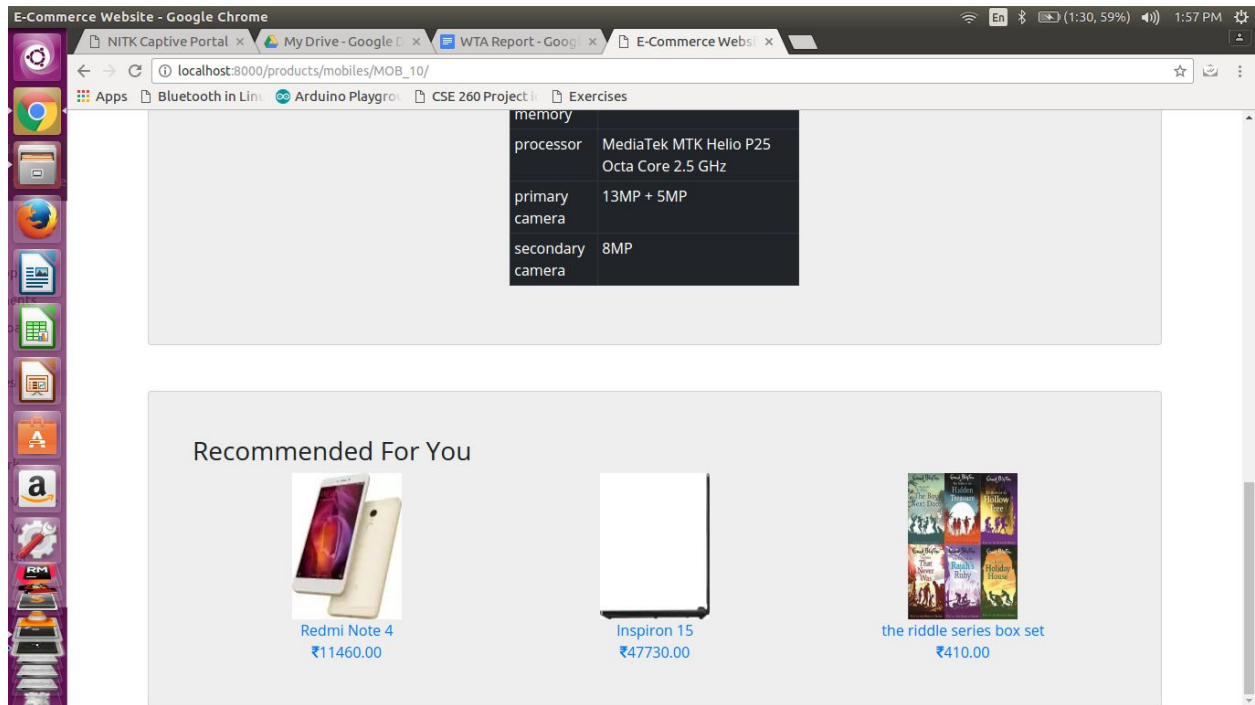
### Product Details View



**Cart Modal**



**Order Invoice**



***Recommendations Loaded through AJAX***

## References

- [1] <http://ieeexplore.ieee.org/document/1167344/?section=abstract>
- [2] <https://docs.djangoproject.com/en/1.11/>
- [3] <https://www.w3schools.com/>
- [4] [https://en.wikipedia.org/wiki/Collaborative\\_filtering](https://en.wikipedia.org/wiki/Collaborative_filtering)
- [5] <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>