Larry Liu, Suriya Kodeswaran, Hanjin Hu, Subhadeep Ghosh
lliu65, kodeswa2, hanjinh2, sghosh12
CS 411 - Spring 2017
May 24, 2017

Final Project Design Documentation

Project Description

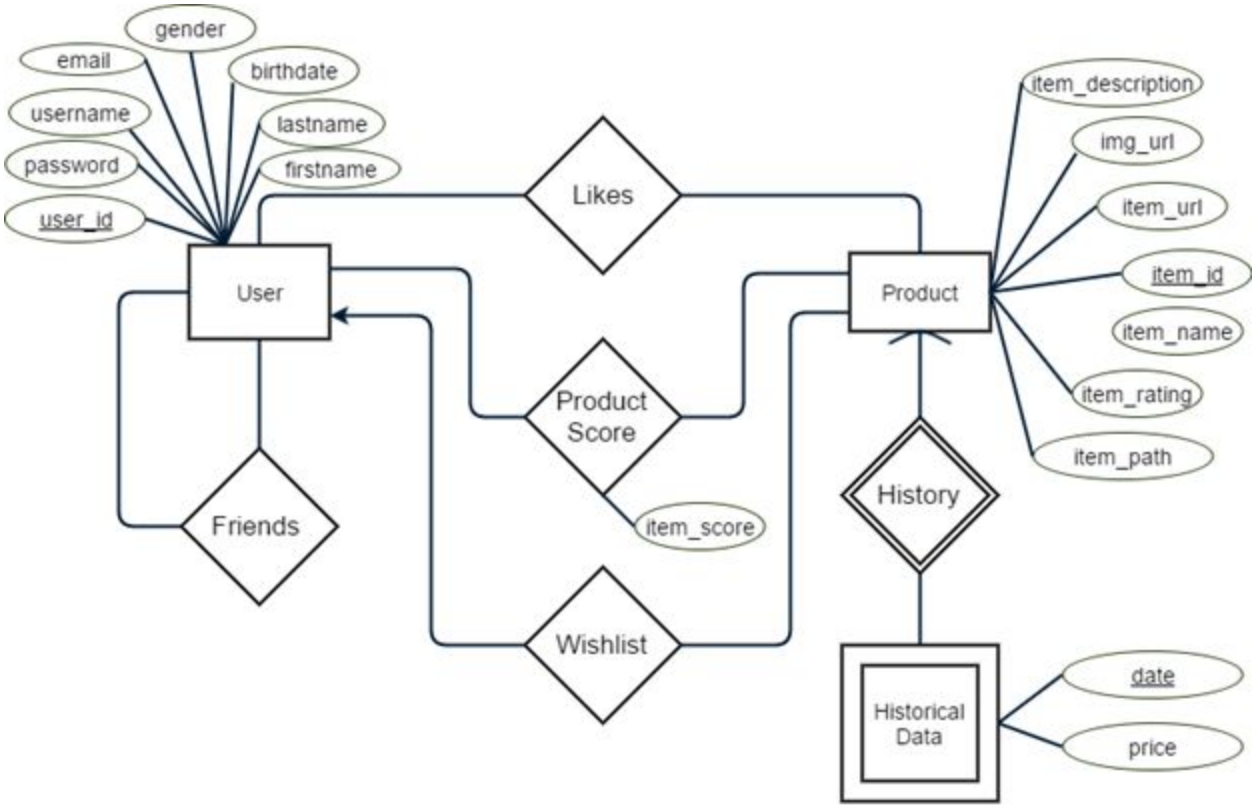
The purpose of this project was to develop a stable web application that can route client-side requests and results to a simple user interface. We communicated to the MySQL databases using PHP, a popular server-scripting language. The development plan started off from writing up an ER diagram, relational schemas, and figuring out the implementation of functionalities.

Our project focuses on collecting data from online retailers with large inventories. For the scope of this course, we focused on products sold in Walmart's electronics department. Our databases store over 3,000 records of product data from Walmart. When considering the user-side of the project, users are able to look-up any desired electronic filtered by various parameters. More importantly, the user can request recommendations on which video games to buy based off of ratings records. Additionally, a user can choose to display the price history for a product along with its predictive scoring. This is simply a general introduction to the advanced features that were implemented in our project, which will be thoroughly discussed in later sections.

Application of Project

Our project aims to provide users a more informative perspective on electronics through predictive analytics that are processed in real-time. A user can receive new recommendations dynamically; thus, enabling a smarter decision on choosing the right video games to buy if he or she were to make full use of the recommender system (for video games only). Similarly, a user can evaluate the most optimal time to buy a product in the electronics department based off of forecasted prices. Clearly, these features leverage all of the product data in our databases in order to generate effective analysis. Other aspects in our application such as filtering and social media also provide focus on creating consumer convenience. Evidently, these approaches aimed to help solve the problem of consumer indecisiveness.

ER Diagram and Schema



Entity-Relationship Diagram

Relation Schema

Shown below are the relational schemas for our database tables:
Friends (host_id, friend_id)
HistoricalData (item_id, date, price)
Likes (user_id, item_id)
Product (item_id, item_name, item_url, item_rating, img_url, item_description, item_path)
UserProductScore (user_id, item_id, item_score)
Users (user_id, email, password, username, gender, birthdate, firstname, lastname)
Wishlist (user_id, item_id)

One VIEW we created was:
GameNames (item_id, *item_name*)

Data Collection

The original intent for our project was to crawl data from large sites such as Walmart, Amazon, Target, etc. However, we found a Walmart Product Search API and were able to successfully register for it. Thus, this was the more practical source for acquiring mass amounts of data that were supplemented with additional product information such as product images, descriptions, customer reviews, descriptions, etc. Since we were able to route essentially unlimited data from Walmart; thus, we decided to focus only on Walmart data because of how descriptive it was compared to web crawlers. Since we can dynamically run a program to make API calls, our system can both function as real-time and capture historical data. Because there exists a constraint that we can not store unlimited data in our databases, we decided to only insert data from Walmart's electronics department (instead of every single item from all departments). This was the most practical approach given the limitation of resources.

List of Functionalities

Our web application provides the following functionalities:

- Sign-in & Sign-up form for login
- User profile changes e.g. Change username, email, password
- Search for any product in Walmart's electronic department
- Filter search based on various criteria such as: Most Popular (most liked products), Price Descending, Price Ascending, Minimum Price, Maximum Price
- Information about product description, image, customer ratings and link to product page
- Adding and Removing friends from a user's friends list
- Viewing a friend's wish list
- Adding/Removing items to wish list, Showing wishlist
- Liking/Unliking items
- Show product history
- Show forecasted product prices
- Recommender system based on user ratings for video games

One Basic Function Example

One basic function we had was filtering products by popularity. This popularity metric was based off of how many likes a product had received from registered users in our system. Although the functionality was basic, the SQL query ended up being very complex due to the number of tables involved as well as the logic that existed. We were able to successfully utilize only 1 query, eliminating the need for multiple. The query performed a `LEFT JOIN` between the `HistoricalData` table and the `Likes` table. This was necessary because we wanted to sort the item ids based off of their occurrences in the Likes table. Thus, this is executed by both the '`GROUP BY`' and '`ORDER BY`' latter portions of the query. What made this even more interesting was the fact that the Product table needed to be included in order to utilize the user's search query as part of finding items similar to the string of text that the user intended on searching for. Clearly, the complexity of this query contributed to the functionality of filtering by minimum and maximum prices, ordering results by descending or ascending, and filtering by product popularity. The SQL query is shown in the next section.

SQL Query Snippet

The SQL Query for the previously mentioned basic function is shown below:

```
SELECT h.price, h.item_id, COUNT(DISTINCT l.item_id) AS post_count
      FROM HistoricalData h
      LEFT JOIN Likes l ON h.item_id = l.item_id
      WHERE h.price >= '$min' AND h.price <= '$max'
            AND h.date = '2017-04-20' AND h.item_id
      IN (  SELECT item_id
            FROM Product
            WHERE item_name LIKE '$search_string'
           )
      GROUP BY h.item_id
      ORDER BY `post_count` DESC LIMIT 50";
```

Note:

`$search_string`: "%" + "string of text" + "%", where the string of text represents what the user had searched for.

`$min, $max`: These variables were input from the user's desired filtering parameters.

Dataflow

HTML/JS to PHP to HTML/JS:

The dataflow starts with the user searching for a product name by inputting a string of text. After clicking the "Search" button, the HTML page makes an POST request to the linked PHP file associated with that query. The PHP file establishes a connection with MySQL, acquires the user's unique identifier from the current SESSION, and performs the SQL query with the POST request parameter as a constraint in the query. These values are then returned back to the HTML file as a JSON encoded array. These values delivered are processed by a simple Javascript function and are dynamically placed into various portions of the page found by matching document element IDs.

HTML/JS to PHP to Python (to PHP) to HTML/JS:

When considering this datapath, the process is very similar with the steps discussed previously. We decided to pass information from a PHP file to Python through file I/O operations on files (instead of socket connections to pass them directly). More importantly, is the fact that executing our Python scripts involve storing the scripts in the /public_html/cgi-bin/virtualenv-15.1.0/myVE/bin/ directory because our Python programs used libraries that had to be installed using virtualenv (discussed in the Technical Challenge section later). Finally, when routing the results of our scripts back to either PHP or HTML/JS, we simply wrote the results into a file formatted either as JSON encoded values or the actual Javascript code to be run.

Screenshots Outlining Dataflow:

```
    <input id="predictive_search_param" class="form-control" placeholder="Product Name" name="product_name" type="text" onchange="script()" autofocus>
</div>
<input class="btn btn-lg btn-success " name="Search" type="submit" value="Search" onclick="script()" />
<script>
    function script(){
        var myNode = document.getElementById("predictive_search");
        while (myNode.firstChild) {
            myNode.removeChild(myNode.firstChild);
        }
        xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function(){
            if(this.readyState == 4){
                myObj = JSON.parse(this.responseText);
                if(myObj == "false"){
                    myObj = "No Items Found."
                }
                //document.getElementById("demo").innerHTML = myObj;
                else{
                    for(var key in myObj){
                        if(myObj.hasOwnProperty(key)){
                            createLink(myObj[key]['item_name'], "predictive_scoring.php?item_id=" + myObj[key]['item_id'] + "&item_name=" + myObj[key]['
                        }
                    }
                }
            }
        };
        xmlhttp.open("POST", "product_price_history.php", true);
        xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
        xmlhttp.send("product_name=" + document.getElementById("predictive_search_param").value);
    }
```

FIGURE 1. Passing data from HTML/JS to PHP

```
<?php
  $tablename = "HistoricalData";

  session_start();
    $link = mysql_connect('webhost.engr.illinois.edu', 'dbsystems_kodeswa2', 'cs411');
    if (!$link) {
        die('Could not connect: ' . mysql_error());
    }
    mysql_select_db('dbsystems_test');


 $results = array();
 $results[] = array('item_id','date','price');
 $item_name = $_GET['item_name'];
 $item_id = $_GET['item_id'];
 echo $item_name;
 echo " - ";
 echo $item_id;
 if ($item_name == ""){
     echo $results;
 }
 $id = $_SESSION['user_id'];
 $search_name = '%' . $product_name . '%';


 $sql0 = "SELECT date, price FROM HistoricalData WHERE item_id = $item_id ";
 $res0 = mysql_query($sql0);
 while ($row = mysql_fetch_array($res0)){
     $results[] = array($item_id,$row['date'],$row['price']
                        );
 }
 $fp = fopen('../cgi-bin/virtualenv-15.1.0/myVE/bin/HistoricalData.csv', 'w');
 foreach($results as $line){
     fputcsv($fp, $line);
 }
  fclose($fp);

$command = 'cd ../cgi-bin/virtualenv-15.1.0/myVE/bin && ./python predictive_scoring.py > log';
$last_line = system($command, $retval);
```

FIGURE 2. Passing data from PHP to Python

```
with open("../../../../data/data.js", "w") as f:
    f.write("var jsondata = [")
    i = 1
    for key, value in orig.items():
        if i == 1:
            i = i + 1
            if value[0] == None:
                f.write('{"date": ' + '"' + str(key) + '"' + ','+  '"forecasted price": ' + '"' + str(value[1]) + '"' + '}')
            elif value[1] == None:
                f.write('{"date": ' + '"' + str(key) + '"' + ','+ '"orig_price": ' + '"' + str(value[0]) + '"' + '}')
            else:
                f.write('{"date": ' + '"' + str(key) + '"' + ','+ '"orig_price": ' + '"' + str(value[0]) + '"' + ',' + '"forecasted price": ' + '"'+ str(value[1]) + '"'+ '}')
        else:
            f.write(",")
            if value[0] == None:
                f.write('{"date": ' + '"' + str(key) + '"' + ','+  '"forecasted price": ' + '"'+str(value[1]) +'"' + '}')
            elif value[1] == None:
                f.write('{"date": ' + '"' + str(key) + '"' + ','+ '"orig_price": ' + '"'+str(value[0]) +'"'+ '}')
            else:
                f.write('{"date": ' + '"' + str(key) + '"' + ','+ '"orig_price": ' + '"'+str(value[0]) + '"'+',' + '"forecasted price": ' + '"'+str(value[1]) +'"'+ '}')
    f.write('];')
    f.write("$(function(){Morris.Line({element: 'predict', data: jsondata, xkey: 'date', ykeys: ['orig_price', 'forecasted price'], labels: ['original price', 'forecasted price
```

FIGURE 3. Passing information from Python to HTML/JS

Advanced Feature 1: Recommender System

The video game recommender (VGR) used for our website is based on collaborative filtering via linear regression. Therefore, the VGR recommends video games to a user based on rating scores from all other users.

The algorithm of the VGR is as follows:

- Describe features/characteristics of a video game using a vector of decimal numbers, Rv, which shows how much the game is action, adventure, role-playing, etc.;
- Describe preferences of a user using a vector of decimal numbers, Lu, which shows how much the user likes action, adventure, role-playing, etc.;
- Calculate rating scores by R_v dot producing Lu;
- Sort the video games by rating scores; and
- Recommend games with the highest score.

Because not every movie has a rating score from every user, the VGR uses Stochastic Gradient Descent and mean normalization to predict scores for video games by optimizing the cost function, a convex in this case (see the figure below as an example). The optimization is through minimizing the sum of squared errors (absolute difference between predicted values and observed values). Multi-linear regression is utilized because the VGR is using two features: user preferences and game features/characteristics.

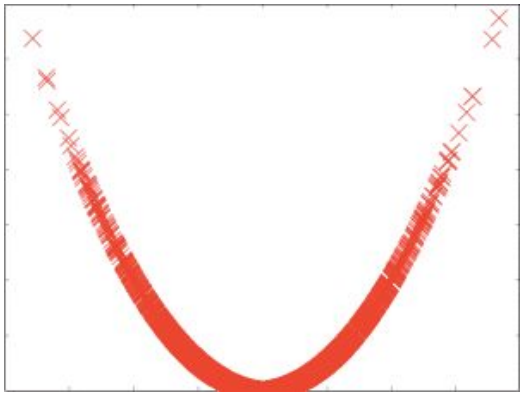


FIGURE 4. Gradient Descent for Convex

The VGR, however, is not accurate enough partially because dataset of the observed rating scores used to train algorithm is not large enough, and the algorithm used by VGR has some limitations. To improve the VGR, clustering algorithms such as k-means will be used in the future.

Advanced Feature 2: Predictive Scoring

For our second advanced feature requirement, we decided to leverage the mass amount of historical data for each product (over the course of 6 months) to generate real-time predictive scoring. We implemented a data predicting structure similar to the Autoregressive Moving Average Model. When a user searches for a product name keyword and selects a particular product, the client interface immediately sends a request to the back-end PHP code with the item's unique identifier. The PHP code then makes a call to our predictive scoring Python program and passes all historical price data that is available for the product (acquired from appropriate querying of MySQL databases). Our Python program reads the input data, makes the proper transformations on formatting it (into a Pandas DataFrame for convenient attribute accesses), and create a time series model for the available dataset). We then further pre-process the data with a Boxcox transformation that resolves skewness. Fortunately, all of our data points are simply positive numbers; thus, leading to a valid transformation since the Boxcox method contains a logarithmic equation in its time series analysis.

When considering the intent of the ARMA model, the autoregressive portion involves explaining values based on previously seen values. The moving average part of the model's definition involves iterating through all points in the dataset, and computing weighted averages from each section with respect to neighboring values. With this in mind, we used the SARIMAX method from the statsmodels Python library. With these calculations computed on the dataset in real-time, we dynamically generate enough statistical measures to then create predictions on future values. We decided to simply forecast the next week's worth of prices for a particular item because extending the latest date would dramatically increase the runtime of the program; thus, forcing the user to wait for much longer until results are visible. One idea we originally had was to pre-compute all of these forecasted prices beforehand; however, we do not have the disk space resources to hold predictive scoring data for all products prevalent in our database. The output of the predictive scoring program is then sent to the client side, where Javascript code properly

formats document elements as well as creates a user-interactable Morris chart graph plots both historic data points and forecasted values as seen in Figure 5. One attribute we believe would be even more useful and if we had the time to implement, would be to provide the user a confidence interval on whether he or she should buy the product at that time instant or in a future time. This confidence interval would be based off of another statistical measure of trends within the plot of our data. This particular feature would further extend upon the usefulness of our predictive scoring advanced feature.

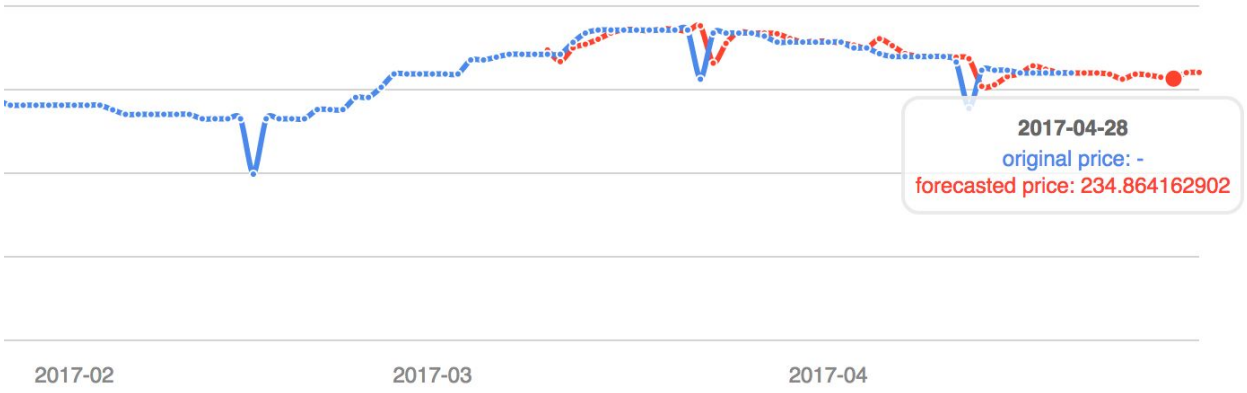


FIGURE 5. Morris chart depicting results from Predictive Scoring

Technical Challenge

One technical challenge we faced in this project was validating the data-flow between PHP to Python. This is mostly challenging since we initially had no direction in knowing how to execute Python scripts from PHP files. We first attempted to use the `system` method for invoking an execution on the Python program file path. However, after multiple failed variations of this attempt, we realized that we initially installed numerous Python libraries we required in our programs using virtualenv. From our understanding, it seems that we can use the default Python interpreter to execute our Python scripts since multiple modules have not been installed in the scope of the default interpreter. Thus, we decided to store our scripts in the directory where the required libraries are accessible: /public_html/cgi-bin/virtualenv-15.1.0/myVE/bin/. As a result, this resulted in further complexity in knowing exactly where both intermediary and output files used to store data are saved. We thought that this was an intriguing technical challenge, since it demonstrated the importance of verifying a system's data-flow for processing requests and recognizing the unexpected inconsistencies that may exist in between.

Project Development Evaluation

Most of our project plans were successful. We believe that the required task of scheduling out project development plans really helped us perceive the amount of work required week-to-week. Additionally, starting early and building our system from the ground-up with schemas helped us identify potential flaws in our system in the early stages of development. One specific plan that we originally had and did not go into our final project, was the utilization of web crawlers across multiple online retailers. This is definitely an additional feature we can very easily expand upon if we decide to improve our web application in the future. However, since we were still able to collect mass data from Walmart (having the same effect as web-crawling Walmart) and given the state of limited memory resources, we thought that this was not a major issue. Also, we initially were intending on building a social media platform as an advanced feature; however, only realized later that there was nothing "advanced" about that. Thus, we decided to implement 1) Recommender System for Video Games, and 2) Predictive Scoring as our advanced features because they required not just tremendous technical complexity, but also contributed to making our platform more meaningful. With these advanced features and other smaller features as part of the original development plan, they strongly define the purpose of our project.

Division of Labor

Larry - Worked mostly on back-end work (PHP scripting), Predictive Scoring advanced feature - Python program, added enhancements to front-end interface
Suriya - Worked on all Javascript functionalities, Advanced Feature 2, PHP files, created database tables
Hanjin - Worked on Recommender System - Python program
Subhadeep - Worked on Recommender System, Walmart API functionality