Topics in Privacy & Security

Trust-Enhanced Reputation Metrics

Y1481702

March 14, 2018

Contents

i Description

The tool can be run from either the `run.php` or `attack.php` scripts in order to calculate reviews or simulate attacks, respectively. For example, from the command-line, `php run.php` can be called. The program will prompt the user for input of the ratings filename and values of `MAX_RATE` and `ALPHA`.
N.B. An SQL database must be setup and defined in the `setup.php` file.

In my implementation of the tool, a number of steps have been taken to help the tool scale. The tool utilises relational databases to quickly access existing product rating data. The `CUSTOMERS` and `PRODUCTS` arrays, as mentioned in the pseudocode, are tables in the database. This means that the tool does not need to store all data in memory, which is usually the most limited resource. If the tool is needed to scale further, SQL provides `LIMIT` and `SKIP` keywords to ensure that memory usage is kept under control.

As is visible from the pseudocode of the tool, I have also attempted to cut down on superfluous computation wherever possible.

```
1  MAX_RATE, ALPHA, FILE = take input from user
2  CUSTOMERS, PRODUCTS = initialise empty array: []
3
4  for each RATING of PRODUCT_J by CUSTOMER_I in FILE:
5      if RATING made by a new customer:
6          CUSTOMERS.append(New CUSTOMER_I with Default Trust Level of 0.5)
7
8      if RATING made of a new PRODUCT:
9          PRODUCTS.append(New PRODUCT_J with RATING)
10         continue to next RATING
11     else:
12         Update the Product Rating for PRODUCT_J
13
14     for each CUSTOMER of PRODUCT_J in CUSTOMERS:
15         Update the Trust Level of CUSTOMER
16
17     for each PRODUCT bought by CUSTOMERS of PRODUCT_J:
18         Update the Product Rating of PRODUCT
```

On Line 6 of the above pseudocode, Equation 3 (from the brief) always returns 0.5 when run with an empty set of products. There is no need to run this calculation each time. Not doing so will save us a small amount of compute time. On Line 10, in the case that this rating is for a new product, the algorithm skips the updating of related customers and products as this will have little effect. This is because, if a product only has one customer, its overall rating is the same as that customer's rating.

The loops to update customer trust levels and product ratings on Lines 14 and 17 respectively, are kept to a minimum by filtering down to only updating trust levels of customers that bought the newly rated product. This is important as these nested loops are the part of the program with the worst worst-case complexity. For efficiency, the final implementation need to take care to ensure that each of customer and product is updated only once. If required, further steps to reduce runtime that have not been taken in my implementation, could include only updating Product Ratings (Line 18) if the customer trust levels have changed significantly and running trust level (Line 15) and product rating (Line 18) updates in different, parallel threads.

As systems scale, they are more likely to become the target of a form of cyber attack. Prepared statements have been used to sanitise inputs whenever input data from outside the program's control

is entered into the database. This should successfully protect against SQL injection attacks.

ii Analysis

A tool has been created to provide overall product ratings from an input file. Screenshots of the tool output for the text-file `Q2Ratings.txt` are shown in Fig. 2. Plots of the ratings for products with IDs 4, 7 and 29 are shown in Fig. 1. For comparison these plots also show the arithmetic mean- the likely rating of the product if a trust-based reputation system is not used.

The products here (Figs. 1a & 1c) show the trust-based reputation system working well. Product #4's ratings are all at the bottom end of the scale, with each rating being either 1, 2 or 3. Product #29's ratings are also all similar, but at the top end of the scale, with each rating being 4 or 5. In these cases the reputation system works well, keeping the overall rating close to the arithmetic mean as the ratings roughly agree how the product should be rated.

Product #7's ratings are much more hit and miss, with every rating being either the minimum rating of 1 or the maximum rating of 5. Here (Fig. 1b) we can see the reputation system working to compensate for users' disagreement. When using the arithmetic mean, each rating is treated with equal weight, so the overall rating falls significantly with each subsequent poor rating. When using the trust-based system, the first rating it receives (5 from Customer #16), is treated as the most trustworthy, so it is harder for the subsequent lower ratings to affect overall ratings, when they differ from the overall rating. The alpha value acts as the tolerance for difference in ratings, lower values of alpha only accept reviews as trusted if they very close to the overall rating.

Using a high-value for alpha, $\alpha \geqslant MAX\_RATE$, generally results in a rating that is very similar to the arithmetic mean. The only difference between this trust-based reputation system with $\alpha = 5$ and using the arithmetic mean is that new customers are not fully trusted initially. As customers leave more ratings, their trust level is mindlessly increased whether or not they fit with what is normal within the context of the rest of the population. While newcomers should always be distrusted[1], it is also important to assess the actions of known customers before upgrading the level of trust they are assigned. As such, it is likely we will want to be slightly more discriminatory against users' actions than this.

At the other end of the spectrum, particularly low values of α may not be tolerant enough to account for natural variation of opinion. Clearly some users will be harder to impress than others so allowing reviews to differ by at least some small amount should be expected. If the system is not tolerant enough to variation it becomes too risky for trustworthy users to post new reviews[2] as the system is more likely to significantly lower their trust, especially if their opinions go against the status quo.

It's difficult to fully assess how successful the reputation system is at it's aim of predicting the future behaviour of the customers without further data.

iii Simulated Attacks

Self-promoting attacks of varying sizes have been simulated on product #4, this is shown by Fig. 3a. Slander attacks of varying sizes have been simulated on product #29, this is shown by Fig. 3b.

iv Results

In Section ii, the advantages and disadvantages of different values of alpha were discussed with regards to normal functioning of the reputation system. Here they will be discussed with regards to the simulated attacks as described in Section iii. The plots of each attack show that the system is least susceptible to attack when an α value of 2 is used. This is the case for both self-promotion and slander attacks of *all* sizes in the test-set. However, it's worth analysing these results in detail before jumping to the conclusion that $\alpha = 2$ is the best value for the system.

Many good reputation systems ensure that newcomers must behave correctly for a given amount of time in order to gain a good reputation. It is important to balance this resilience to attacks with the ease of admittance for new customers.[3] As discussed in Section ii, values of α that are close to `MAX_RATE` do nothing to ensure that users behave correctly. While this will disincentivise the act of creating new accounts for attacks, they allow attackers to perform multiple attacks across a variety of products using the same account in order to increase their trust level. Obviously a compromise is needed.

As well as reducing the ease of admittance for new customers, values of alpha that are too low ($\alpha = 1$) don't perform as well as slightly higher values at protecting against attacks. This can be seen clearly in the plots for each attack (Fig. 3). These lower values tend not to trust several of the genuine reviews as they vary slightly too much from the overall rating. As such, when attack occurs, the overall rating will begin to deviate from its initial value much more quickly.

As this reputation system is fairly primitive, self-promote and slander attacks still have significant effects on the overall rating, with even the most effective value of alpha allowing the attack to change the rating by at least 1. As larger and larger attack sizes are used, the attacker is able to have a significant effect on the product's overall rating. Furthermore, if the attacker has knowledge of the system, they would know that a more effective attack vector could be to submit reviews that are equal to α added to, or subtracted from, from the overall rating as the rating is slowly lowered. A significantly better system is needed to combat these forms of attack. In particular, the ability to find proof of successful transactions, as well as preventing the hacker from obtaining multiple identities, as has been the case in both of these attacks, will be significantly more effective in mitigating both self-promoting and slander attacks.[3]

In conclusion, it seems that $\alpha = 2$ is the most effective value for the system after all. It provides a compromise by ignoring as few genuine reviews as possible while still defending well against different types of attack. Further work should also be done in order to assess the impact that other forms of attack could have on this reputation system. For example, as the system uses the current trust level of the reviewer to calculate overall product reviews, it could be particularly susceptible to a whitewashing attack. This is where the reviewer posts malicious or self-promoting reviews and then attempts to repair their reputation afterwards[3].

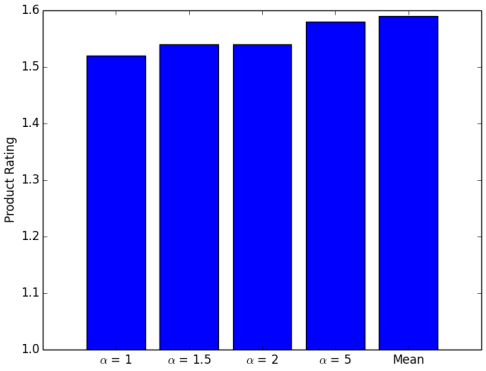
v References

[1] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, "Reputation systems," *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, Dec. 2000. [Online]. Available: http://doi.acm.org/10.1145/355112.355122

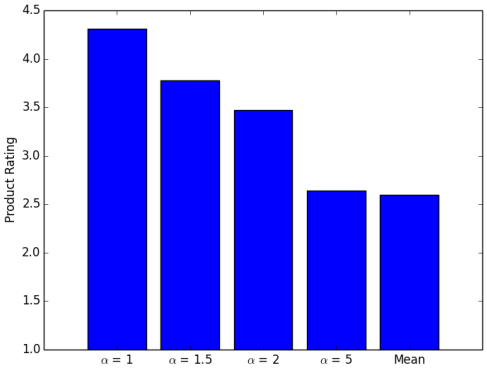
[2] F. Hendrikx, K. Bubendorfer, and R. Chard, "Reputation systems: A survey and taxonomy," *Journal of Parallel and Distributed Computing*, vol. 75, pp. 184 – 197, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001464

[3] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Comput. Surv.*, vol. 42, no. 1, pp. 1:1–1:31, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1592451.1592452

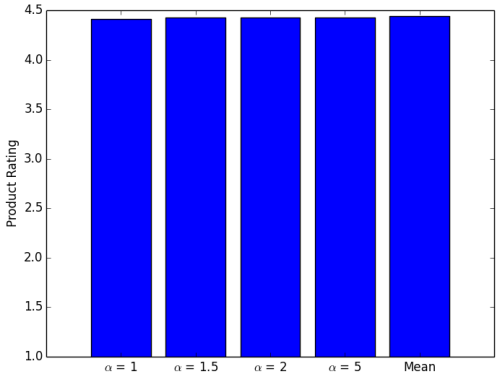
vi Appendix



(a) Product 4



(b) Product 7



(c) Product 29

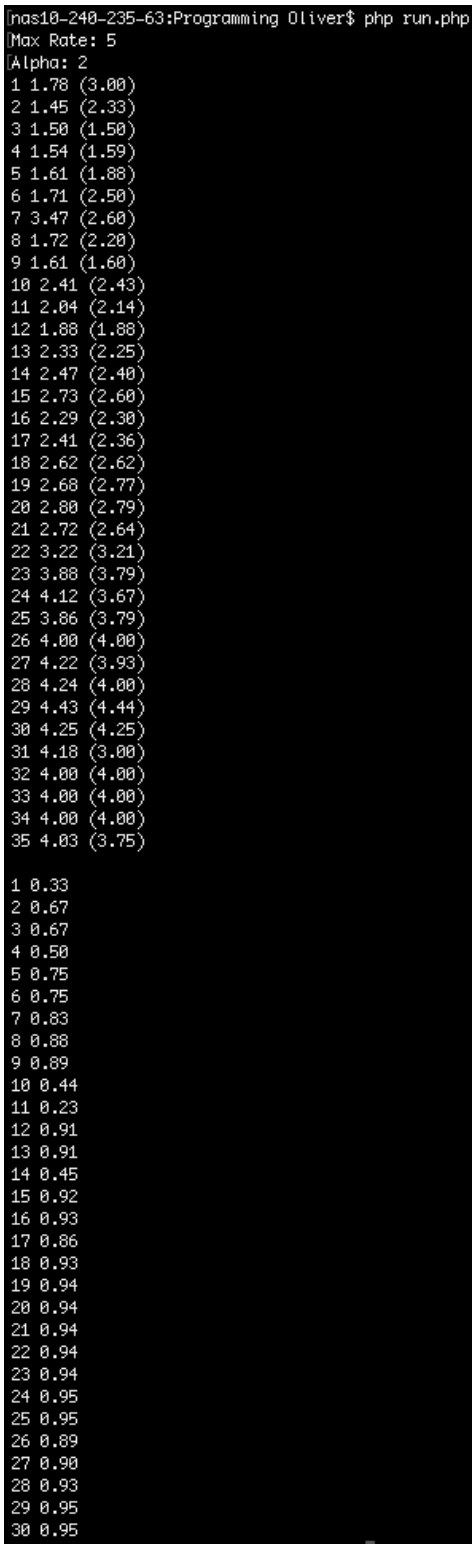
Figure 1: Rating change for varying α

```
nas10-240-235-63:Programming Oliver$ php run.php
Max Rate: 5
Alpha: 1
1 1.69 (3.00)
2 1.40 (2.33)
3 1.44 (1.50)
4 1.52 (1.59)
5 1.51 (1.88)
6 1.47 (2.50)
7 4.31 (2.60)
8 1.62 (2.20)
9 1.57 (1.60)
10 2.45 (2.43)
11 1.85 (2.14)
12 1.87 (1.88)
13 2.44 (2.25)
14 2.56 (2.40)
15 2.71 (2.60)
16 2.32 (2.30)
17 2.54 (2.36)
18 2.61 (2.62)
19 2.61 (2.77)
20 2.78 (2.79)
21 2.77 (2.64)
22 3.22 (3.21)
23 4.04 (3.79)
24 4.08 (3.67)
25 3.88 (3.79)
26 4.00 (4.00)
27 4.33 (3.93)
28 4.45 (4.00)
29 4.41 (4.44)
30 4.26 (4.25)
31 4.34 (3.00)
32 4.00 (4.00)
33 4.00 (4.00)
34 3.93 (4.00)
35 4.13 (3.75)

1 0.33
2 0.67
3 0.67
4 0.50
5 0.75
6 0.75
7 0.83
8 0.62
9 0.44
10 0.11
11 0.15
12 0.73
13 0.82
14 0.09
15 0.83
16 0.87
17 0.86
18 0.93
19 0.81
20 0.81
21 0.76
22 0.82
23 0.75
24 0.79
25 0.74
26 0.78
27 0.85
28 0.67
29 0.70
30 0.79
```

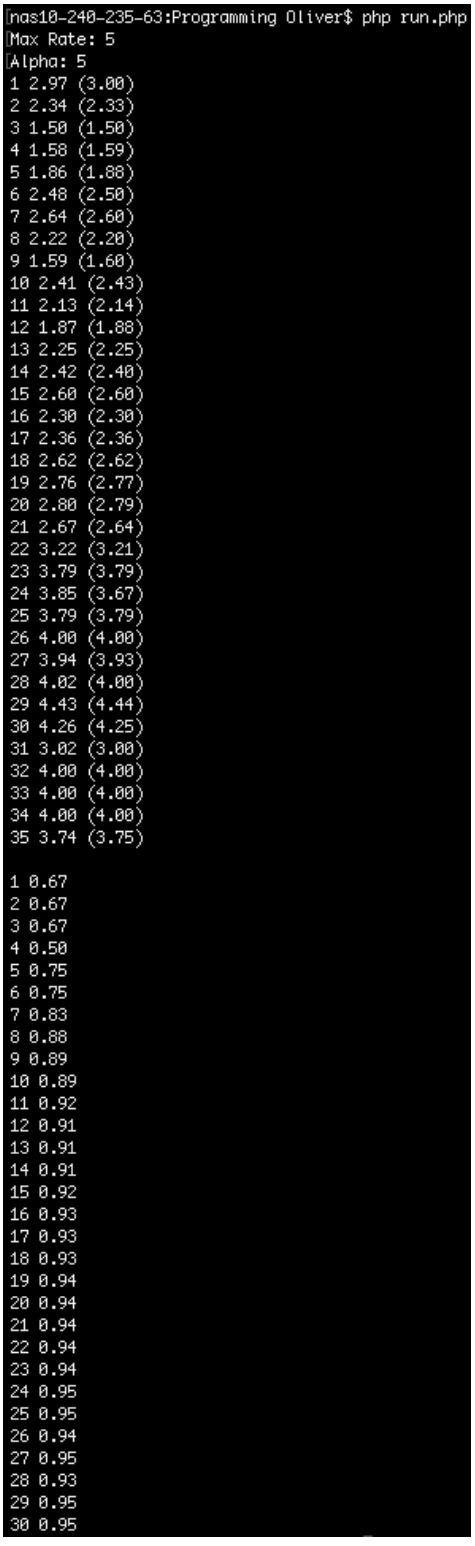
(a) $\alpha = 1$

```
nas10-240-235-63:Programming Oliver$ php run.php
Max Rate: 5
Alpha: 1.5
1 1.59 (3.00)
2 1.31 (2.33)
3 1.49 (1.50)
4 1.54 (1.59)
5 1.58 (1.88)
6 1.62 (2.50)
7 3.78 (2.60)
8 1.65 (2.20)
9 1.62 (1.60)
10 2.39 (2.43)
11 1.90 (2.14)
12 1.87 (1.88)
13 2.29 (2.25)
14 2.52 (2.40)
15 2.75 (2.60)
16 2.29 (2.30)
17 2.43 (2.36)
18 2.64 (2.62)
19 2.68 (2.77)
20 2.80 (2.79)
21 2.75 (2.64)
22 3.21 (3.21)
23 3.94 (3.79)
24 4.10 (3.67)
25 3.90 (3.79)
26 4.00 (4.00)
27 4.32 (3.93)
28 4.30 (4.00)
29 4.43 (4.44)
30 4.26 (4.25)
31 4.41 (3.00)
32 4.00 (4.00)
33 4.00 (4.00)
34 4.00 (4.00)
35 4.07 (3.75)

1 0.33
2 0.67
3 0.67
4 0.50
5 0.75
6 0.75
7 0.83
8 0.75
9 0.89
10 0.44
11 0.15
12 0.91
13 0.82
14 0.18
15 0.92
16 0.93
17 0.86
18 0.93
19 0.94
20 0.88
21 0.94
22 0.88
23 0.94
24 0.89
25 0.89
26 0.89
27 0.85
28 0.93
29 0.95
30 0.89
```

(b) $\alpha = 1.5$

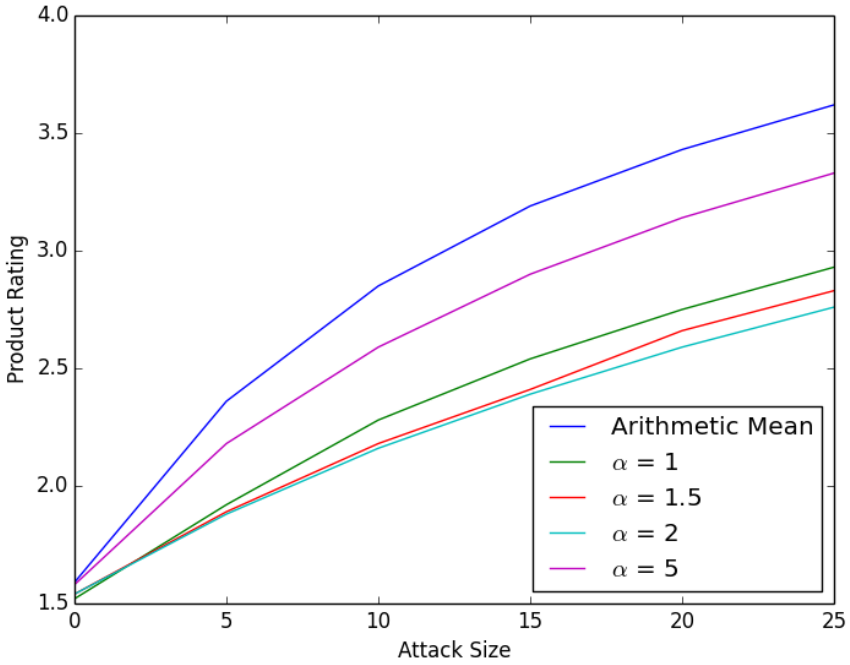


(c) $\alpha = 2$

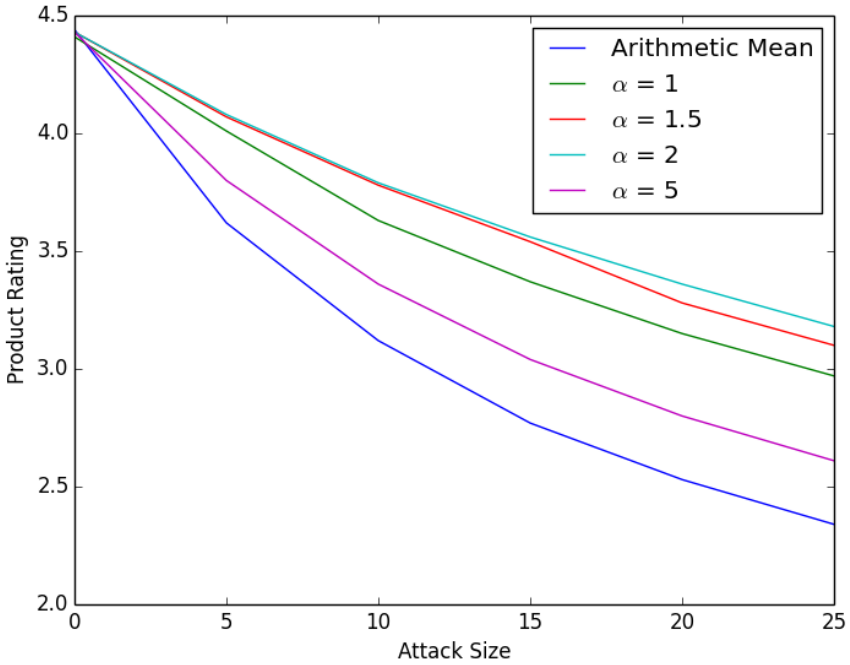


(d) $\alpha = 5$

Figure 2: Tool Output for Varying Alpha



(a) Effect of Self-Promotion Attacks of Varying Sizes on Product 4



(b) Effect of Slander Attacks of Varying Sizes on Product 29

Figure 3: Attacks on the Online Store Rating System

vi.1 run.php

```php
<?php
require_once("setup.php");
require_once("process.php");

while(!feof($myfile)){
    //FOR EACH RATING.. update the database:
    $data = explode(",", fgets($myfile));
    if(count($data) != 3){break;}
    $customer_id = $data[0];
    $product_id = $data[1];
    $rating = $data[2];

    log_new_rating($db, $data[0], $data[1], $data[2]);
}
fclose($myfile);

if (basename(__FILE__) == basename($_SERVER["SCRIPT_FILENAME"])) {
    //Only run output if file was run DIRECTLY from console,
    //NOT included in another file: i.e. attack.php
    $base_output = "output/Alpha_" . strval(ALPHA) . "_";
    require_once("output.php");
}
```

vi.2 attack.php

```php
<?php

//$attack_type = readline("Attack Type (slander/promote): ");
$attack_type = "slander";
require_once("run.php");

for($j = 0; $j < 5; $j++){
    for($i = 0; $i < 5; $i++){
        //Rating is 0 if slander, MAX_RATE if self-promoting
        $rating = MAX_RATE;
        $product_id = 4;
        if($attack_type == "slander"){
            //Lowest rating is 1 NOT 0
            $rating = 1;
            $product_id = 29;
        }

        //Null customer rating- creates new customer id
        //product id = 29, as stated in question
        log_new_rating($db, null, $product_id, $rating);
    }

    $base_output = "output/" . ucfirst($attack_type) . "_" .
        strval(5 * ($j + 1)) . "_Alpha_" . strval(ALPHA) . "_";
    require("output.php");
}
```

vi.3 setup.php

```
<?php
$filename = readline("Input File: ");
$myfile = fopen($filename, "r");

define("MAX_RATE", intval(readline("Max Rate: ")));
define("ALPHA", floatval(readline("Alpha: ")));


$db = new mysqli("localhost", "psec", "password");
$db->query("DROP DATABASE psec_assessment;");
$table_setup = "
   CREATE DATABASE psec_assessment;
   USE psec_assessment;
   CREATE TABLE ratings(
      id INT AUTO_INCREMENT PRIMARY KEY,
      customer_id INT,
      product_id INT,
      rating INT
   );
   CREATE TABLE customers(
      id INT AUTO_INCREMENT PRIMARY KEY,
      trust_level FLOAT
   );
   CREATE TABLE products(
      id INT AUTO_INCREMENT PRIMARY KEY,
      rating FLOAT
   );
";
$db->multi_query($table_setup);
while($db->more_results()){
    $res = $db->next_result();
}
```

vi.4 process.php

```
<?php
function log_new_rating($db, $customer_id, $product_id, $rating){
    $trust = 0.5; //Equation 3 returns 0.5 when given the EMPTY SET
    //Check if this is a new user:
    if($customer_id == null){
        //This is a simulated attack:
        //completely new customer ID must be created:
        $stmt = $db->prepare(
            "INSERT INTO customers (trust_level) VALUES(?);"
        );
        $stmt->bind_param("s", $trust);
        $stmt->execute();

        $customer_id = $db->insert_id;
    }else{
        $stmt = $db->prepare(
```

```
17              "SELECT COUNT(*) FROM customers where id=?;"
18          );
19          $stmt->bind_param("s", $customer_id);
20          $stmt->execute();
21          if($stmt->get_result()->fetch_assoc()["COUNT(*)"] == 0){
22              //initialise trust level if new customer: This is 0.5
23              if($stmt =
24                  $db->prepare(
25                      "INSERT INTO customers VALUES (?, ?);"
26                  )){
27                  $stmt->bind_param("ss", $customer_id, $trust);
28                  $stmt->execute();
29              }
30          }
31      }
32
33
34      //LOG THE NEW RATING:
35      $stmt = $db->prepare(
36          "INSERT INTO ratings (customer_id, product_id, rating)
37  ------VALUES(?, ?, ?);"
38      );
39      $stmt->bind_param("sss", $customer_id, $product_id, $rating);
40      $stmt->execute();
41
42      //Calculate overall product rating
43      $stmt = $db->prepare("SELECT COUNT(*) FROM products where id=?;");
44      $stmt->bind_param("s", $product_id);
45      $stmt->execute();
46      //If NEW product
47      if($stmt->get_result()->fetch_assoc()["COUNT(*)"] == 0){
48          //initialise rating with the rating of the NEW customer:
49          if($stmt = $db->prepare("INSERT INTO products VALUES (?, ?);")){
50              $stmt->bind_param("ss", $product_id, floatval($rating));
51              $stmt->execute();
52          }
53          //NEW PRODUCT, nothing left to update?
54          return;
55      }//IF EXISTING product:
56
57      //Update trust levels of all customers who bought this product
58      $stmt = $db->prepare(
59          "SELECT customer_id FROM ratings WHERE product_id=?;"
60      );
61      $stmt->bind_param("s", $product_id);
62      $stmt->execute();
63      $result = $stmt->get_result();
64      while($row = $result->fetch_assoc()){
65          update_trust($db, $row["customer_id"]);
66
67      }
68
```

```
69      //Recalculate all products other than project j
70      //LIMIT THIS TO PRODUCTS THAT HAVE BEEN AFFECTED!:
71      $stmt = $db->prepare(
72          "SELECT DISTINCT product_id FROM ratings
73  WHERE customer_id IN
74  (SELECT customer_id FROM ratings WHERE product_id = ?)"
75      );
76      $stmt->bind_param("s", $product_id);
77      $stmt->execute();
78      $result = $stmt->get_result();
79      while($row = $result->fetch_assoc()){
80          update_rating($db, $row["product_id"]);
81      }
82  }
83
84  function update_rating($db, $product_id){
85      $stmt = $db->prepare(
86          "SELECT rating, trust_level FROM ratings, customers
87  WHERE customer_id = customers.id AND product_id = ?;"
88      );
89      $stmt->bind_param("s", $product_id);
90      $stmt->execute();
91      $result = $stmt->get_result();
92
93      //Equation 2 (Brief.pdf):
94      $numerator = 0;
95      $denominator = 0;
96
97      foreach($result as $rating){
98          $numerator += $rating["rating"] * $rating["trust_level"];
99          $denominator += $rating["trust_level"];
100     }
101
102     $stmt = $db->prepare("UPDATE products SET rating = ? WHERE id = ?;");
103     $rating = $numerator / $denominator;
104     $stmt->bind_param("ss", $rating, $product_id);
105     $stmt->execute();
106 }
107
108
109 function update_trust($db, $customer_id){
110     $fetch_products = "SELECT
111 ratings.rating AS customer_rating,
112 products.rating AS overall_rating
113 FROM ratings, products
114 WHERE product_id=products.id AND customer_id=?;";
115
116     $stmt = $db->prepare($fetch_products);
117     $stmt->bind_param("s", $customer_id);
118     $stmt->execute();
119     $result = $stmt->get_result();
120
```

```
121     $stmt = $db->prepare(
122         "UPDATE customers SET trust_level = ? WHERE id = ?;"
123     );
124     $tl = trust_index($result);
125     $stmt->bind_param("ss", $tl, $customer_id);
126     $stmt->execute();
127 }
128
129 //Equation 3 (Brief.pdf):
130 function trust_index($products){
131     $numerator = 1;
132     $denominator = 2;
133
134     foreach($products as $product){
135         $numerator += is_trusted(
136             $product["overall_rating"],
137             $product["customer_rating"]
138         );
139
140         $denominator++;
141     }
142
143     return $numerator / $denominator;
144 }
145
146 //Equation 4 (Brief.pdf):
147 function is_trusted($overall_rating, $customer_rating){
148     if(abs($overall_rating - $customer_rating) <= ALPHA){
149         return 1;
150     }
151     return 0;
152 }
```

vi.5 output.php

```
1  <?php
2  //Output to file or console:
3  $customers = fopen($base_output . "Customers.txt", "w");
4  $products = fopen($base_output . "Products.txt", "w");
5
6  $rep_based = $db->query("SELECT * FROM products;");
7  $average = $db->query(
8      "SELECT AVG(rating) rating FROM ratings
9  GROUP BY product_id ORDER BY product_id;"
10 );
11 foreach($rep_based as $product){
12     $avg = $average->fetch_assoc();
13
14     $out_str = sprintf("%u %0.2f (%0.2f)\n",
15         $product["id"],
16         $product["rating"],
17         $avg["rating"]
18     );
```

```
19
20      echo $out_str;
21      fwrite($products, $out_str);
22  }
23  echo "\n";
24  $result = $db->query("SELECT * FROM customers;");
25  foreach($result as $customer){
26      $out_str = sprintf("%u %0.2f\n",
27          $customer["id"],
28          $customer["trust_level"]
29      );
30
31      echo $out_str;
32      fwrite($customers, $out_str);
33  }
```