MATH-36032 PSBC Project 3: Purchasing Data Analysis

This project seeks to analyse a dataset collected over a four-year period, for some of the active customers of a large online retailer. The data is provided in a comma separated values (CSV) format. The following information is available in the dataset.

- `Date:` Online transaction date from 1st January 2015 to 31st December 2018.
- `Customer_ID:` Anonymised customer ID.
- `Product_Category:` Category of the product from 5 categories tagged `A-E`.
- `Product_Value:` The total amount of the order in British pounds (£).
- `Rating:` Customer rating from `1-5`. Any sales for which a rating was not provided have the value `0`.
- `Return:` `Y` (Yes) or `N` (No) indicating if the product was returned for a refund.

The first few entries in the dataset can be seen below.

```
   Date,      Customer_ID,  Product_Category,  Product_Value,  Rating,  Return;
01-Jan-2015,    1010408,           B,              33.5,         5,       N;
01-Jan-2015,    1014220,           C,              18.4,         4,       N;
01-Jan-2015,    1016167,           E,              23.2,         4,       N;
01-Jan-2015,    1019094,           D,               46,          0,       Y;
```

This project tackles the following three problems.

1. Logistic Regression

Problem

Majority of the customers who returned the product gave a low rating. Let $P(r)$ be the probability that a product is likely to be returned. We seek to find a relationship between $P(r)$ and the rating r, given by the customer in the following *logistic function* with two parameters α and β.

$$P(r) = \frac{1}{1 + \exp(-\alpha r - \beta)}. \tag{1.1}$$

1.1 Approach

The two parameters α and β can be obtained by minimising a loss function. A loss function is a measure of how well a prediction model performs in predicting the expected outcome, i.e. a smaller value of the loss function implies a better performance. From the several loss functions available, we minimise the 'Sum of Squares' loss function which is defined as follows:

$$\sum_i (P_i - P(r_i))^2. \tag{1.2}$$

Suppose we have a MATLAB function `sumsquareloss`, which implements the loss function defined in (1.2).

```
function S = sumsquareloss(a,r,p)
% Sum of squares loss function.
% a = [alpha, beta]
S = 0;
for i = 1:length(r)
    % sum the values (P_i - P(r_i))^2
    S = S + (p(i) - 1/(1 + exp(-a(1)*r(i) - a(2))))^2;
end
```

Listing 1: Implementation of Sum of Squares loss function defined in (1.2).

We can use the `fminsearch` function in MATLAB to minimise `sumsquareloss` as a function of α and β. However, it is necessary to perform some pre-processing on the given dataset before we can optimise the logistic regression function.

1.1.1 Pre-processing

In order to reduce the overall computational complexity, we only consider ratings given by those customers who have returned an order at least once. To extract this data from the dataset, we first gather the `Customer_ID`'s of all the orders that were returned. Using this result, we can obtain all the orders belonging to those customers only. Once we have this sub-table, we can remove any orders that were not rated by the customer, i.e. `Rating` $= 0$.

Finally, we can extract the array `ratings`, and the corresponding array `probs`, containing logical values 1 or 0 which represents whether the product was returned or not. These can be used as the input parameters `r` and `p` for the `sumsquareloss(a,r,p)` function.

1.1.2 Implementation

Upon the pre-processing of the data, we make use of the `fminsearch` and `sumsquareloss` functions to minimise the loss in the logistic regression model and find the parameters α and β. The proposed function, `LogisticFunction()` serves the purpose.

```
function params = LogisticFunction()
% Returns the optimised parameters alpha and beta in P(r).

data = readtable('purchasing_order.csv');

% Pre-processing stage to extract relevant data.
cust_ret = data(ismember(data.Return, {'Y'}), :).Customer_ID;
all_orders = data(ismember(data.Customer_ID, cust_ret), :);
orders_w_rating = all_orders(all_orders.Rating > 0, :);
ratings = orders_w_rating.Rating;
probs = strcmp(orders_w_rating.Return, 'Y');

% Optimise the parameters by minimising the sum of squares loss function.
params = fminsearch(@(a) sumsquareloss(a, ratings, probs), [0 0]);
end
```

Listing 2: `LogisticFunction()` returns the parameters α and β in $P(r)$.

1.2 Result

The `LogisticFunction()` returns the two parameters α and β. In particular, the optimised parameters for the model are calculated as the following.

```
>> LogisticFunction()

ans = -17.542411276527133   17.418797309784775
```

Output of `LogisticFunction()`.

The output from the function yields the values $\alpha = -17.5424$ (4dp) and $\beta = 17.4188$ (4dp). Figure 1.1 depicts a plot of the logistic regression function calculated above. It is evident that a low rating of 1 has a high probability, $P(r)$, which was what was expected.

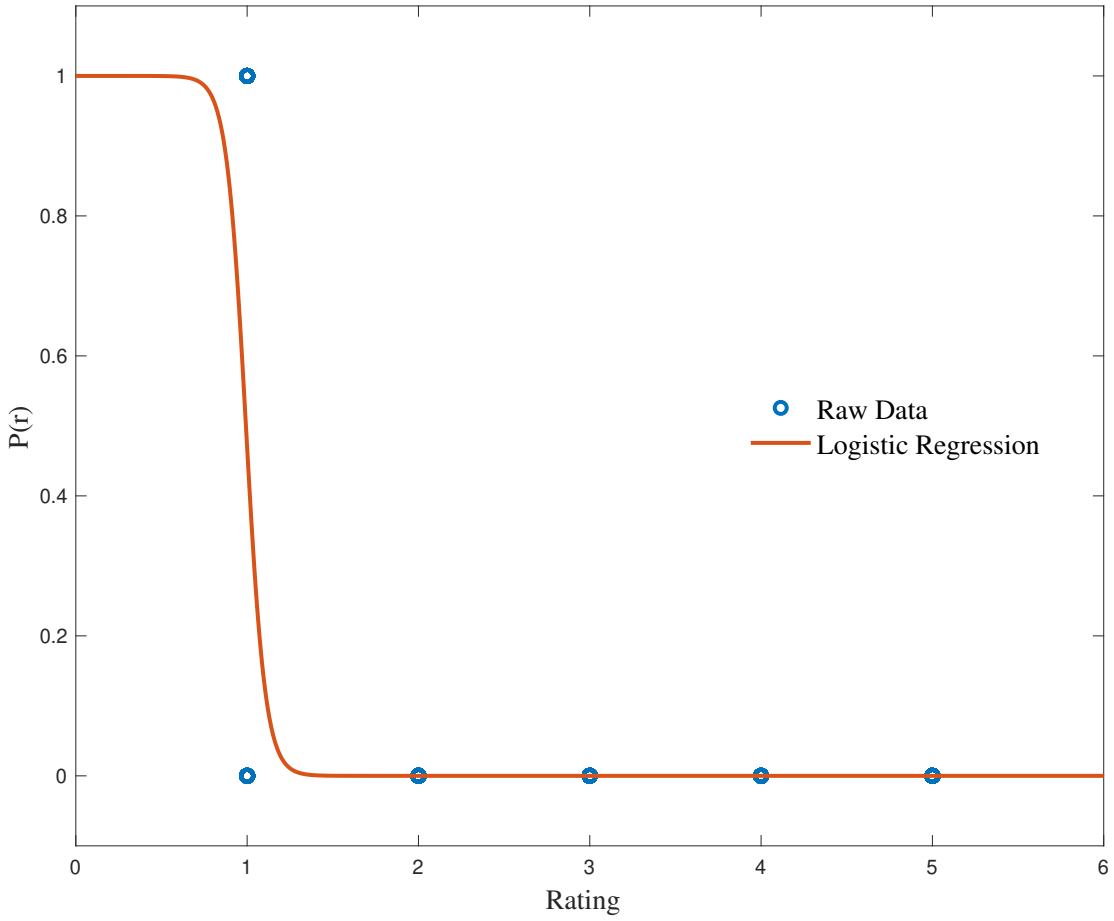

Figure 1.1: Plot of the Logistic Regression function.

2. Returning Customers

Problem

It is intuitive to believe that if a customer returns a product, they are less likely to make future orders from the company. The aim of this task is to investigate if this is actually the case by defining a suitable likelihood criteria.

2.1 Approach

First of all, we define the criteria for the likelihood that a customer will make another purchase after returning a product. We define this such that the likelihood of a customer making future orders is measured by the percentage of total amount spent after the return, over the total amount spent over the four year period, excluding the refunded orders.

2.1.1 Implementation

Suppose we have the following function `ReturningLikelihood()` which returns an overall likelihood that customers who made one return, placed another order in the future. This is showcased below.

```
1  function return_prob = ReturningLikelihood()
2  % Returns the average likelihood of customers placing future orders after
3  % refunding an order.
4
5  data = readtable('purchasing_order.csv');
6
7  returned_transactions = data(ismember(data.Return, {'Y'}), :);
8  returned_customers = unique(returned_transactions.Customer_ID);
9  likelihood = zeros([length(returned_customers), 1]);
10
11 for i = 1:length(returned_customers)
12     all_orders = data(data.Customer_ID == returned_customers(i), :);
13     orders_no_return = all_orders(ismember(all_orders.Return, {'N'}), :);
14     total_spent = sum(orders_no_return.Product_Value);
15     return_date = all_orders(ismember(all_orders.Return, {'Y'}), :).Date(1);
16     spent_after_return = sum(all_orders(ismember(all_orders.Return, {'N'}) ...
17         & all_orders.Date > return_date, :).Product_Value);
18     likelihood(i) = spent_after_return / total_spent;
19 end
20 likelihood = likelihood*100;
21 return_prob = mean(likelihood);
22 end
```

Listing 3: `ReturningLikelihood()` returns an overall likelihood (%) of customers returning after a refunded order.

2.1.2 Correctedness

The function first extracts all the transactions that were returned (line 7). From this, all the unique `Customer_ID`'s of such customers are obtained in an array (line 8). The function then loops over this list of `Customer_ID`'s. In each iteration, i.e. for each customer, all orders placed by that customer are retrieved (line 12). This is further refined to orders that were not returned and the total amount spent

by the customer (`total_spent`) is calculated (line 13-14). The date of the first order returned by the customer is obtained and the total amount spent after that date (`spent_after_return`), excluding refunded orders, is calculated (line 15-17). Finally, the `likelihood` array that corresponds to the customer ID's is populated with the ratio, `spent_after_return/total_spent` (line 18).

Once the likelihood for each customer is calculated, it is converted to a percentage (line 20) and then a mean of these likelihoods is returned by the function (line 21). Hence, we successfully calculate the desired likelihood. □

2.2 Result

The output of the `ReturningLikelihood()` function is an overall percentage representing the likelihood of a customer making another purchase after they have returned an order.

```
1 >> ReturningLikelihood()
2 ans = 50.421477089777326
```

Output of `ReturningLikelihood()`.

The average likelihood is approximately 50%, which is not the best indication whether the customer is less likely to come back after returning an order. Therefore, we plot a histogram to analyse the distribution of likelihoods for all such customers. Figure 2.1 depicts the histogram.

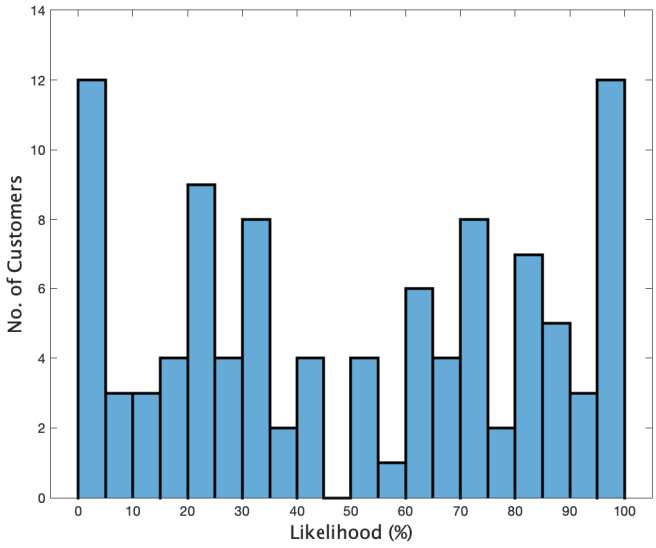

Figure 2.1: Distribution of likelihoods.

We consider the original intuition that customers are less likely to make another purchase after a refunded order. If this was the case, the histogram would be a triangular shape with the peak closer to a likelihood of 0% ($x = 0$). However, this is not the case as illustrated by Figure 2.1. Hence, we can conclude that the customers are not necessarily less likely to make a purchase after having a product returned.

NB: It is important to note that the `ReturningLikelihood()` function is implemented to only consider the date of the first order that is returned. However, a customer could have multiple returns. Therefore, it will be a better indicator to consider the average of the amounts spent in all the segments between these return dates. This was not implemented in this function because in the given dataset, none of the customers made more than one refund.

3. Loyalty Reward

Problem

The company has decided to send 100 coupons with 30% off for clothing in product category C, to build a long-term customer loyalty and to manage inventory more effectively. The customers who have placed orders with large values in category C, and have given higher ratings are the targets for these coupons. The aim of the task is to rank the customers based on a weighted average of following two criteria:

(I) Average amount spent per order in Category `'C'`.

(II) The average rating given for such orders.

The weights must be chosen in such a way such that the customers ranked at the top according to either (I) or (II) independently are still likely to be at the top with both (I) and (II) combined.

3.1 Approach

Since the rankings are based on two separate variables, we must find a way to normalise these into a weighting where they can be used together. The approach we take is to normalise the average amount spent and the ratings given, so that they are both values between 0 and 1. Once we have values between 0 and 1 for the rating and the amount spent for each customer, we can add the two normalised values together and rank the customers according to the highest sum. The customers with the highest sum will be ranked at the top. This approach allows us to compare both the `Rating` and the `Product_Value` equally because they are both values between 0 and 1. Hence, adding those together would give a reasonable indicator of rank.

The average `Product_Value` for each customer is normalised by dividing by the most expensive order value. On the other hand, `Rating` is divided by 5 (highest possible rating).

3.1.1 Implementation

Suppose we have a function `RankCustomers()`, which returns the ranked list of customers that have placed orders in product category C. To implement such a function, we first obtain all the orders placed in category C. We can use this to retrieve all the unique `Customer_ID`'s and make use of the `accumarray` function in MATLAB to get a mean of the total value spent and the mean ratings given for each customer. These averages can then be normalised as discussed above and generate a table with `Customer_ID`'s and their corresponding normalised average product value (`norm_value`), average rating (`norm_rating`) and the sum (`norm_sum`). The function is implemented as follows.

```
function rankings = RankCustomers()
% Returns a ranking of the 'most loyal' customers who buy products from
% Category 'C'.

data = readtable('purchasing_order.csv');

% Filter all orders that were for Category C.
orders_C = data(ismember(data.Product_Category, {'C'}) & ismember(data.Return, {'N'}), :);

% Generate a table with unique Customer_IDs and their average value spent
```

```
% and ranking given for Category C orders.
[unique_cust, ~, idx] = unique(orders_C.Customer_ID);
avg_value = accumarray(idx, orders_C.Product_Value, [], @mean);
avg_rating = accumarray(idx, orders_C.Rating, [], @mean);

% The highest amount spent on an order (for normalising Product_Value).
max_value = max(avg_value);

% Normalise the average product values and rankings and find the normalised sum.
norm_value = avg_value / max_value;
norm_rating = avg_rating / 5;
norm_sum = norm_value + norm_rating;

% Create a table containing the Customer_IDs, the normalised Rating,
% Product_Value and their sum.
customers_C = table(unique_cust, norm_value, norm_rating, norm_sum);

% Sort in descending order by the normalised sum to obtain the rankings.
rankings = sortrows(customers_C, {'norm_sum'}, {'descend'});
rankings.Properties.VariableNames([1]) = {'customer_id'};
end
```

Listing 4: `RankCustomer()` returns the customers ranked in order.

3.2 Result

The `RankCustomers()` function outputs a table in order of rankings for all the customers who made a purchase in product category C. The top 10 and the customers ranked from 91-100 are outputted below as a result of `RankCustomers()`.

```
>> rankings = RankCustomers();

% Top 10 Customers
>> rankings(1:10,:)
ans =
    customer_id       norm_value        norm_rating        norm_sum
    -----------    -----------------    -----------    ----------------
      1014288      0.998784933171324          1        1.99878493317132
      1011981      0.992709599027947          1        1.99270959902795
      1016309      0.875455650060753          1        1.87545565006075
      1014429      0.838396111786148          1        1.83839611178615
      1012195      0.814702308626975          1        1.81470230862697
      1016443      0.806804374240583          1        1.80680437424058
      1015864      0.800121506682868          1        1.80012150668287
      1014953                      1        0.8                     1.8
      1014887      0.792223572296476          1        1.79222357229648
      1011918      0.772782503037667          1        1.77278250303767

% Rank 91-100 Customers
>> rankings(91:100,:)
ans =
    customer_id       norm_value        norm_rating        norm_sum
    -----------    -----------------    -----------    ----------------
      1010573      0.572296476306197          0.9         1.4722964763062
      1015326      0.621506682867558         0.85        1.47150668286756
      1013142      0.471445929526124            1        1.47144592952612
      1012489      0.468408262454435            1        1.46840826245444
      1013800      0.515188335358445         0.95        1.46518833535844
      1011041       0.46415552855407            1        1.46415552855407
```

```
30      1011068      0.462940461725395          1        1.46294046172539
31      1017300       0.86269744835966        0.6        1.46269744835966
32      1011436      0.461725394896719          1        1.46172539489672
33      1013998      0.461117861482382          1        1.46111786148238
```

Rankings of customers calculated by `RankCustomers()`.

We can see in the top 10 result above that both the `norm_value` and `norm_rating` values are very close to 1. One of the requirements of the rankings was that the rankings must be similar to the cases where criteria (I) and (II) are considered independently. To validate this, we plot the normalised average product value against the rating, highlighting the points that were picked as the top 100 customers. A good weighting of both criteria would result in selecting the points towards the top right-hand triangular region of the graph, as this region indicates that the rankings will be similar if considered independently. Figure 3.1 depicts the result of all such customers. It is clear to see that the top 100 chosen customers are in the top-right region, indicating that the rankings are somewhat optimal.

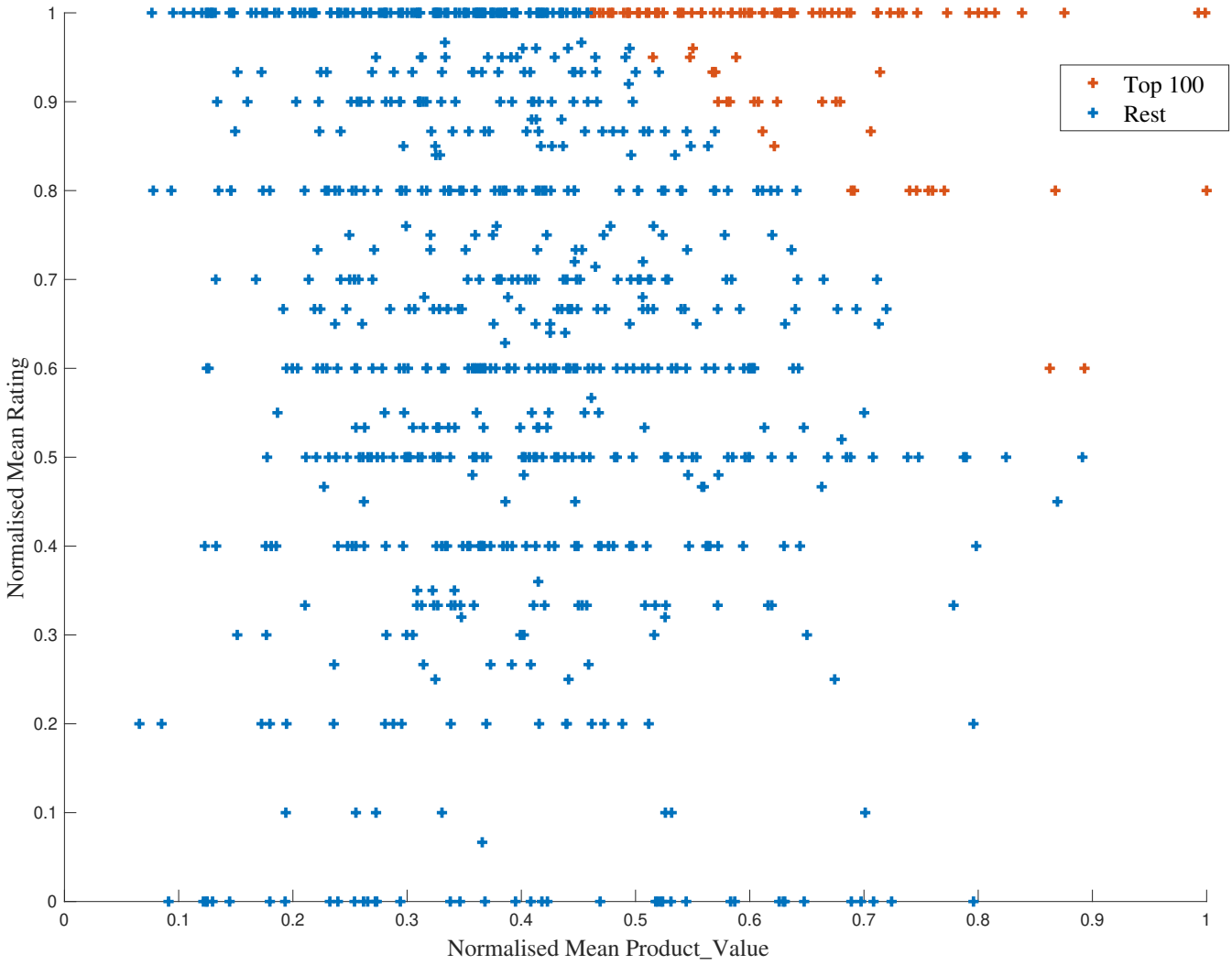

Figure 3.1: Normalised Product Value (`norm_value`) vs. Rating (`norm_rating`), highlighting the top 100 customers.