Problem Solving By Computer - Project 3

Introduction

In the e-commerce world, there is an importance in being able to predict the behaviour of customers to survive in the market. In particular, is there a relation between a product being returned and the rating that was left by the customer? Moreover, are customers less likely to make future orders after they return a product for a refund? Finally, how can we determine the most valuable customers by deriving a ranking system which takes multiple attributes into account?

In this project, we will analyse online transactions that were collected over four years for a given large online retailer. The data set is provided as a CSV (comma-separated values) file, and the first few lines of the data file are as follows.

```
   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
01–Jan–2015     1019535           D               25.5          0       N
```

The schema of the data file is as follows.

```
Date: online transaction date (from 01 Jan 2015 to 31 Dec 2018)
Customer_ID: anonymised customer ID
Product_Category: one of 'A', 'B', 'C', 'D' or 'E'
Product_Value: the total amount of the order, in £GBP
Rating: the customer rating as a 1–5 star rating system, and 0 if no rating
Return: whether the product was returned for refund, N or Y
```

We will start by reading the CSV file into a table. MATLAB uses type inference to determine the appropriate data types for the values in each column. In order to see the data types inferred by MATLAB, we can use the function `summary(AllOrders)`.

```
AllOrders = readtable('purchasing_order.csv');
summary(AllOrders)
```

1 The return rate on low rating items

Since the majority of the customers who returned an order gave a low rating, we will start by finding a relation between the probability $P(r)$ that a product is likely to be returned and the rating r the customer gave, using the following logistic function with two parameters α and β.

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

That is, if the rating is low, then the customer is more likely to return the product. To reduce the overall computation complexity, we will only consider the ratings of those customers who have returned a product for a refund.

1.1 Preprocessing the data

As we wish only to consider the ratings of those customers who have returned at least one product for a refund, we first need to filter out orders from the data set that do not satisfy this requirement. First, we can obtain the list of customer IDs for those customers who have returned a product for a refund by filtering for orders that were returned, followed by MATLAB's `unique()` function to remove any duplicates.

```
ReturnedOrders = AllOrders(strcmp(AllOrders.Return, 'Y'), :);
UsersWithReturns = unique(ReturnedOrders.Customer_ID);
```

Next, we need to obtain the orders that were placed by customers that have returned at least one order and have a rating left by the customer. That is, we need to obtain the set of orders whose `Customer_ID` belongs in the `UsersWithReturns` list, and have `Rating > 0`. We can obtain such orders by using logical indexing as follows.

```
Orders = AllOrders(ismember(AllOrders.Customer_ID, UsersWithReturns), :);
OrdersWithRatings = Orders(Orders.Rating > 0, :);
```

That is, `OrdersWithRating` contains orders from `AllOrders` that have a rating, and the customers who placed the orders have returned at least one product for a refund.

1.2 Finding the parameters α and β

Suppose we have a function $f(\alpha, \beta)$ which tells us the lack of fit of the logistic function $P(r, \alpha, \beta)$, then we can minimise the lack of fit by using MATLAB's `fminsearch()` function, which finds the minimum of a multi-variable function. That is, we can create an cost function $f(\alpha, \beta)$ which tells us the lack of fit of the logisitic function $P(r)$ with parameters α and β, then use `fminsearch()` to find the values of α and β which correspond to the minimum cost.

1.2.1 Models for the cost function

We will use the non-linear least squares model for our cost function. As the name suggests, the error is given by the sum of the square of the errors. The non-linear least squares error function is as follows.

$$f(\alpha, \beta) = \sum_i \Big(p_i - P(r_i)\Big)^2 = \sum_i \left(p_i - \frac{1}{1 + \exp(-\alpha r_i - \beta)}\right)^2$$

where (r_i, p_i) is an order with rating $r_i \in [1, ..., 5]$ and a boolean flag $p_i \in \{0, 1\}$ which indicates whether the order was refunded, with 1 indicating that the order was refunded. The implementation of the non-linear least squares cost function is as follows.

```
function S = costFunction(a, r, p)
  S = 0;
  for k = 1 : length(r)
    S = S + (p(k) - 1 / (1 + exp(-a(1) * r(k) - a(2))))^2;
  end
end
```

Finally, we use `fminsearch` to find α and β by minimising the error function, as follows.

```
orderRating = OrdersWithRatings.Rating;
isReturned = strcmp(OrdersWithRatings.Return, 'Y');
lrParams = fminsearch(@(a) costFunction(a, orderRating, isReturned), [0 0]);
fprintf('[alpha, beta] = [%.5f, %.5f]\n', lrParams);
```

1.3 Results

We conclude that the relation between the probability $P(r)$ that a product is returned and the rating r the customer gave, is governed by the following logistic function.

$$P(r) = \frac{1}{1 + \exp(17.542411r - 17.418797)}$$

That is, we conclude that the parameters are $\alpha = -17.542411$ and $\beta = 17.418797$. Further, we can plot the logistic function against the data points as follows.

```
hx = linspace(0, 6, 1001);
plot(orderRating, isReturned, 'o', ...
     hx, 1 ./ (1 + exp(-lrParams(1) * hx - lrParams(2))));
lg = legend('Raw Data', 'Logistic Regression');
set(lg, 'Location', 'east');
xlabel('Rating'); xticks([1 : 5]);
ylabel('Order Returned');
yticks([0, 1]); yticklabels({'No', 'Yes'});
axis([0.5, 5.5, -0.1, 1.1]);
```

2 Making future orders after a refund

It is believed that customers are less likely to make future orders from a company after they have returned a product for a refund. In order to check the validity of this claim, we will calculate the percentage of total values spent after the first return over the total value spent during the four years, excluding any returned and hence refunded orders.

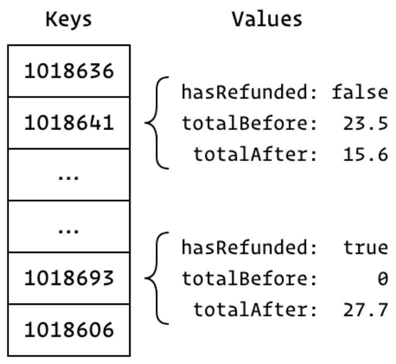
2.1 Approach

We will keep track of three values for each customer, namely the total spent before the refund, the total spent after the refund, and a boolean flag `hasReturned` to indicate whether a customer has returned a product yet. It follows that we can enumerate over the data set, and increment either the total spent before or total spent after value, depending on the value of the `hasReturned` flag. The pseudocode for the algorithm is as follows.

```
Enumerate over each order in the data set
  If the order was refunded
    Set the hasReturned flag on the customer to true
  Else
    If the customer has refunded an item already
      Add the order value to the customers total after value
    Else
      Add the order value to the customers total before value
```

2.2 Implementation

From our algorithm, we see that for each customer that has made a refund, we wish to store three properties, namely `totalBefore`, `totalAfter` and a `hasReturned` flag. In order to store such properties for a single customer, we will create a state `struct` which contains the required properties. In order to have a state `struct` for each customer, we will use a `Map` whose key will be the customer ID, with the value being the state structure.



We can make use of `repmat()` to setup the map such that every customer is mapped to the same initial state, namely `{hasRefunded = false, totalBefore = 0, totalAfter = 0}`.

```
initialState = struct('hasRefunded', false, 'totalBefore', 0, 'totalAfter', 0);
initialStates = repmat({initialState}, length(UsersWithReturns), 1);
m = containers.Map(UsersWithReturns, initialStates);
```

It follows that we can enumerate over the orders and update the customers state as follows.

```
for row = 1 : height(Orders)
  order = Orders(row, :);
  customer = m(order.Customer_ID);

  if strcmp(order.Return, 'Y')
    customer.hasRefunded = true;
  else
    if customer.hasRefunded
      customer.totalAfter = customer.totalAfter + order.Product_Value;
    else
      customer.totalBefore = customer.totalBefore + order.Product_Value;
    end
  end

  % Update the customer object in map
  m(order.Customer_ID) = customer;
end
```

All that remains to do is to calculate the percentage of total values spent after the first return for each customer. Instead of iterating over the map of customers, we can make use of vector operations in MATLAB to compute a vector of percentages of total values spent after the first return. First, we destructure the cell array of map values, then destructure the fields of the resulting structure array into vectors. The implementation is as follows.

```
mapValues = m.values;
customerData = [mapValues{:}];
totalSpending = [customerData.totalBefore] + [customerData.totalAfter];
percentSpentAfterRefund = [customerData.totalAfter] ./ totalSpending * 100;
```

That is, `percentSpentAfterRefund` is a vector whose values are the percentages of total values spent after the first return, where each entry corresponds to one customer. Finally, we can get the mean percentage of funds that were spent after the first refund as follows.

```
averageSpentAfter = mean(percentSpentAfterRefund);
fprintf('The average total order value spent after first refund is %.2f%%\n', ...
        averageSpentAfter);
```

2.3 Results

We conclude that the average percentage of customers total order value after the first refund is 50.42%. That is, on average, customers spend just over half of their total order value after the first refund, and so the data does not support the claim that customers are less likely to make future orders after their first returned order.

We can plot the distribution of total value spent after the first refund as follows.

```
histogram(percentSpentAfterRefund, 20);
grid on;
xlabel('Percentage of total value spent after the first refund');
ylabel('Number of customers');
```

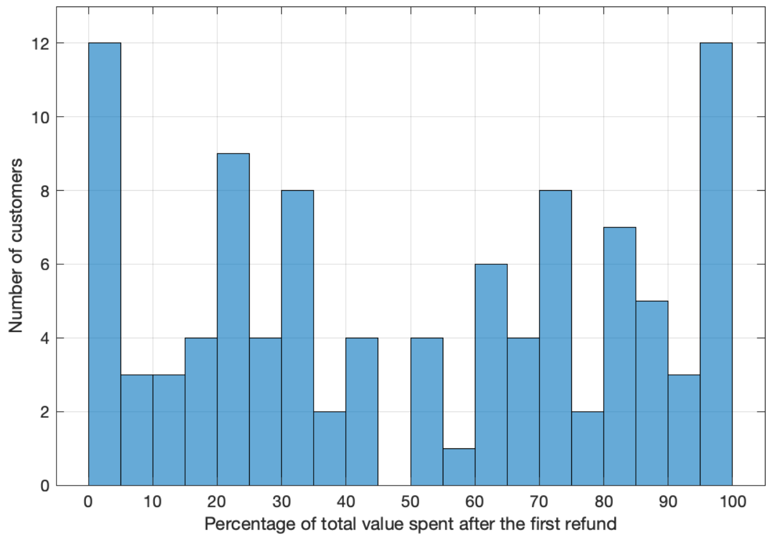


Figure 1: Distribution of total value spent after a customers first refund

Figure 1 shows that customers are not less likely to make future orders after their first returned ordered as the distribution of total value spent after the first refund does not appear to be in a decreasing manner. That is, if it were the case that customers are less likely to place future orders after their first refund, we would expect Figure 1 to resemble a right-angle triangle, where the number of customers decreases as the percentage of total value spent increases.

3 Determining the most valuable customers

In e-commerce, the customer lifetime value (CLV) is the value a customer contributes to your business over their lifetime at your company. For example, we could say that customers who purchase many products from category C are valuable, as products from this category have a high profit margin. Similarly, we could say that customers that leave high ratings on our products are valuable, as they are more likely to recommend our products through word of mouth.

For simplicity, all customers will be ranked according to the following criteria:

(1) average product value per order in category C

(2) the average rating left on orders in category C

Moreover, we wish to design our ranking system in such a way that customers ranked top according to one criterion are still likely to be on top in the ranking with both criteria. That is, the two given criteria are of equal importance and should be weighted equally. For example, a customer with a high average product value per order in Category C that leaves a low average rating on orders should still be towards the top in our ranking system.

3.1 Calculating the average order value and rating

In order to get the average product value per order in Category C by a customer, we start by filtering for orders with product category C, then use `groupsummary()` to group records by customer ID and take the mean of the product values. The implementation is as follows.

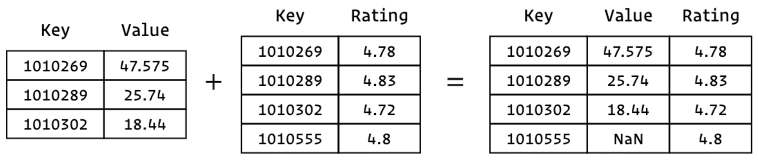
```
OrdersFromC = Orders(strcmp(Orders.Product_Category, 'C') ...
                     & strcmp(Orders.Return, 'N'), :);
CatCValue = groupsummary(OrdersFromC, {'Customer_ID'}, 'mean', 'Product_Value');
```

Similarly, we can get the average rating for each customer by using the `groupsummary()` method on the subset of orders which have a rating. Given that the a rating of 0 implies that a particular product was not rated by the customer, we can get the subset of orders which have a rating by simply checking whether the rating is non-zero. Further, this check will also filter out any orders which have a **`NaN`** rating, and so we have also dealt with case when the data set contains missing data.

```
OrdersWithRatings = OrdersFromC(OrdersFromC.Rating > 0, :);
Ratings = groupsummary(OrdersWithRatings, {'Customer_ID'}, 'mean', 'Rating');
```

3.2 Creating a customer table using outer join

Now that we have an average order value and an average rating for each customer, we wish to create a table such that each row contains a customer ID, their average order value, and their average rating. In fact, we can create such table by using an outer join on the two separate tables we have created. The implementation of creating such table is as follows.



Key	Value
1010269	47.575
1010289	25.74
1010302	18.44

+

Key	Rating
1010269	4.78
1010289	4.83
1010302	4.72
1010555	4.8

=

Key	Value	Rating
1010269	47.575	4.78
1010289	25.74	4.83
1010302	18.44	4.72
1010555	NaN	4.8

```
TLeft = table(CatCValue.Customer_ID, CatCValue.mean_Product_Value, ...
              'VariableNames', {'Customer_ID', 'Mean_Value'});
TRight = table(Ratings.Customer_ID, Ratings.mean_Rating, ...
               'VariableNames', {'Customer_ID', 'Mean_Rating'});
T = outerjoin(TLeft, TRight, 'MergeKeys', true);
```

Observe that customers that have either not placed any orders for products in category C or not left any ratings on orders, but not both, will have a **`NaN`** value in the table `T`. Further, any customer who has never placed an order for a product in category C and has never left a rating on a order will not appear in `T`. However, that is the expected behaviour, as we cannot rank customers who have no orders that satisfy the ranking criteria. We will replace any potential **`NaN`** values with 0 using the `fillmissing()` function.

```
T = fillmissing(T, 'constant', 0);
```

All that remains is to derive a ranking based on the values `Mean_Value` and `Mean_Rating`.

3.3 How do we rank the customers?

Observe that we have constructed a table `T` whose rows consists of customer IDs along with their average order value for products in category C and the average rating they left across all orders. The first few rows of the table `T` are as follows.

```
Customer_ID      Mean_Value        Mean_Rating
___________    _____________     _____________
  1010269             47.575     4.77777777778
  1010289              25.74     4.83333333333
  1010302              18.44     4.71428571429
  1010314              40.72               4.8
  1010320        33.566666667              4.8
```

We can gain insights regarding the range of the data in our table by calling `summary(T)`.

```
Mean_Value: 913x1 double              Mean_Rating: 913x1 double
  Values:                               Values:
    Min              5.4                  Min                0
    Median          33.5                  Median           4.5
    Max             82.3                  Max                5
```

Observe that the range of values for the average order value is much higher than the range of values for the average rating. If we were to take the mean of the two values, the ranking criteria would favour a higher average rating. In order to ensure that the two attributes have equal weighting, we will standardise the data by converting them into standard scores. The standard score is the number of standard deviations by which the value of a data point is above or below the mean value. For example, a customer who leaves an average rating of 4.6 will have a positive standard score, whereas a customer who leaves an average of 4.2 will have a negative standard score, as the mean rating across all orders is 4.3636.

A data point x is converted into a standard score Z by

$$Z = \frac{x - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of the population, respectively. In particular, we can make use of the `zscore()` function to calculate the standard scores for each of the two columns. Finally, we will calculate a ranking score for each customer by summing the two standard scores.

```
T.Mean_Value = zscore(T.Mean_Value);
T.Mean_Rating = zscore(T.Mean_Rating);
Ranking = table(T.Mean_Value + T.Mean_Rating, 'VariableNames', {'Ranking'});
T = [T Ranking];
```

3.4 Results

In order to get the most valuable customers, we need to sort the table `T` based on the `Ranking` column, in descending order. That is, a customer with a higher ranking is a more valuable customer, and perhaps we may wish to reward the most valuable customers with a coupon for their next purchase as a sign of gratitude. We can sort the table based on the derived ranking and display the top 10 and the last 10 of the top 100 as follows.

```
T = sortrows(T, 'Ranking', 'Descend');
disp(T(1:10, {'Customer_ID', 'Ranking'}));
disp(T(91:100, {'Customer_ID', 'Ranking'}));
```

The first and last few customers ID's of the top 100 customers are listed below with their respective ranking score.

```
 Rank     Customer_ID     Ranking             Rank     Customer_ID     Ranking
_____     ___________    _________           _____     ___________    _________
   1        1014288       4.369582              91       1014680       1.626465
   2        1011981       4.331798              92       1011938       1.600016
   3        1016326       3.700806              93       1016515       1.578864
   4        1014953       3.622792              94       1015724       1.577346
   5        1016309       3.602567              95       1010546       1.573568
   6        1014429       3.372085              96       1012660       1.559120
   7        1012195       3.224727              97       1016303       1.539376
   8        1016438       3.187610              98       1014434       1.536450
   9        1016443       3.175608              99       1016848       1.535784
  10        1015864       3.134046             100       1015439       1.533931
```

4 Appendix

The data set `purchasing_order.csv` can be found on the following GitHub gist.
https://gist.github.com/jTanG0506/eb2f7f7f3a98d95f0da420f0a7e438ae

Listing 1: refund_probability.m

```
AllOrders = readtable('purchasing_order.csv');

% Get all the users who have made at least one refund.
ReturnedOrders = AllOrders(strcmp(AllOrders.Return, 'Y'), :);
UsersWithReturns = unique(ReturnedOrders.Customer_ID);

% Get the orders that are placed by users that have made a refund.
Orders = AllOrders(ismember(AllOrders.Customer_ID, UsersWithReturns), :);
OrdersWithRatings = Orders(Orders.Rating > 0, :);

% Find parameters by minimising the cost function.
orderRating = OrdersWithRatings.Rating;
isReturned = strcmp(OrdersWithRatings.Return, 'Y');
lrParams = fminsearch(@(a) costFunction(a, orderRating, isReturned), [0 0]);
fprintf('[alpha, beta] = [%.5f, %.5f]\n', lrParams);

% Plot the linear regression graph.
hx = linspace(0, 6, 1001);
plot(orderRating, isReturned, 'o', ...
     hx, 1 ./ (1 + exp(-lrParams(1) * hx - lrParams(2))));
lg = legend('Raw Data', 'Logistic Regression');
set(lg, 'Location', 'east');
xlabel('Rating'); xticks([1 : 5]);
ylabel('Order Returned'); yticks([0, 1]); yticklabels({'No', 'Yes'})
axis([0.5, 5.5, -0.2, 1.2]);
```

Listing 2: costFunction.m

```
function S = costFunction(a, r, p)
  S = 0;
  for k = 1 : length(r)
    S = S + (p(k) - 1 / (1 + exp(-a(1) * r(k) - a(2))))^2;
  end
end
```

Listing 3: returning_customers.m

```
AllOrders = readtable('purchasing_order.csv');

% Get all the users who have made at least one refund.
ReturnedOrders = AllOrders(strcmp(AllOrders.Return, 'Y'), :);
UsersWithReturns = unique(ReturnedOrders.Customer_ID);

% Get the orders that are placed by users that have made a refund.
Orders = AllOrders(ismember(AllOrders.Customer_ID, UsersWithReturns), :);

initialState = struct('hasRefunded', false, 'totalBefore', 0, ...
                      'totalAfter', 0);
initialStates = repmat({initialState}, length(UsersWithReturns), 1);
m = containers.Map(UsersWithReturns, initialStates);

% Enumerate over all orders placed by users with at least one refund.
for row = 1 : height(Orders)
  order = Orders(row, :);
  customer = m(order.Customer_ID);

  % If the order was returned, set the customer's hasRefunded flag to true.
  if strcmp(order.Return, 'Y')
    customer.hasRefunded = true;
  else
    % Update the total value based on is customer's hasRefunded flag.
    if customer.hasRefunded
      customer.totalAfter = customer.totalAfter + order.Product_Value;
    else
      customer.totalBefore = customer.totalBefore + order.Product_Value;
    end
  end

  m(order.Customer_ID) = customer;
end

mapValues = m.values;
customerData = [mapValues{:}];
totalSpending = [customerData.totalBefore] + [customerData.totalAfter];
percentSpentAfterRefund = [customerData.totalAfter] ./ totalSpending * 100;

% Average of a customers total order value after the first returned order.
averageSpentAfter = mean(percentSpentAfterRefund);
fprintf('The average of a customers total order value after the first refund
     is %.2f%%\n', averageSpentAfter);

histogram(percentSpentAfterRefund, 20);
grid on;
xlabel('Percentage of total value spent after the first refund');
ylabel('Number of customers'); ylim([0 13]);
```

Listing 4: valuable_customers.m

```
Orders = readtable('purchasing_order.csv');
OrdersFromC = Orders(strcmp(Orders.Product_Category, 'C'), :);

% Mean order value for Category C products, by Customer ID.
NonReturnedOrders = OrdersFromC(strcmp(OrdersFromC.Return, 'N'), :);
CatCValue = groupsummary(OrdersFromC, {'Customer_ID'}, ...
                         'mean', 'Product_Value');

% Mean rating, by Customer ID.
OrdersWithRatings = OrdersFromC(OrdersFromC.Rating > 0, :);
Ratings = groupsummary(OrdersWithRatings, {'Customer_ID'}, ...
                       'mean', 'Rating');

% Create a table of the form [Customer ID, Mean Value, Mean Rating].
TLeft = table(CatCValue.Customer_ID, CatCValue.mean_Product_Value, ...
              'VariableNames', {'Customer_ID', 'Mean_Value'});
TRight = table(Ratings.Customer_ID, Ratings.mean_Rating, ...
               'VariableNames', {'Customer_ID', 'Mean_Rating'});
T = outerjoin(TLeft, TRight, 'MergeKeys', true);

% Replace any potential NaN values with 0.
T = fillmissing(T, 'constant', 0);

% Standardize attributes and calculate ranking by summing the z-scores.
T.Mean_Value = zscore(T.Mean_Value);
T.Mean_Rating = zscore(T.Mean_Rating);
Ranking = table(T.Mean_Value + T.Mean_Rating, 'VariableNames', {'Ranking'});
T = [T Ranking];

% Sort customers based on ranking and output top 10 customers.
T = sortrows(T, 'Ranking', 'Descend');
disp(T(1:10, {'Customer_ID', 'Ranking'}));
```