Documentation for the KBS-assignment

Group number: G5
Group Members:

Projjwal Raj (12041160)
Mohit Kumar (1204____)
Chinmay Sahoo (12040480)

Table of contents

Problem Statement

Finding how much discount should a shopping website offer on a product on the basis of factors like demand, availability and date of manufacturing of the product and using fuzzy logic and *Takagi and Sugeno's Approach*, given that the maximum discount that can be offered is 60%.

Introduction

Shopping website like Myntra has algorithm to decide how much discount should it offer on a given product. The algorithm considers several factors, a few of them being

i) demand of the product in the market

higher the demand, lower will be the discount offered for the product and vice versa.

ii) stock available with the seller

If the stock available with the seller is high, the seller should offer higher discount and vice versa.

iii) span of time the product has been in the warehouse

Consider a product is new in very the market then the seller would not offer much discount to it even if its demand is quite low or its stock available with the seller is abundant. This is because, newer product may take some time to grasp the market and for the consumer to get to know about it.

Design of the solution

We are taking three inputs to the program, `I 1` , `I 2` and `I 3`, standing for demand, stock and time since its first sale respectively.

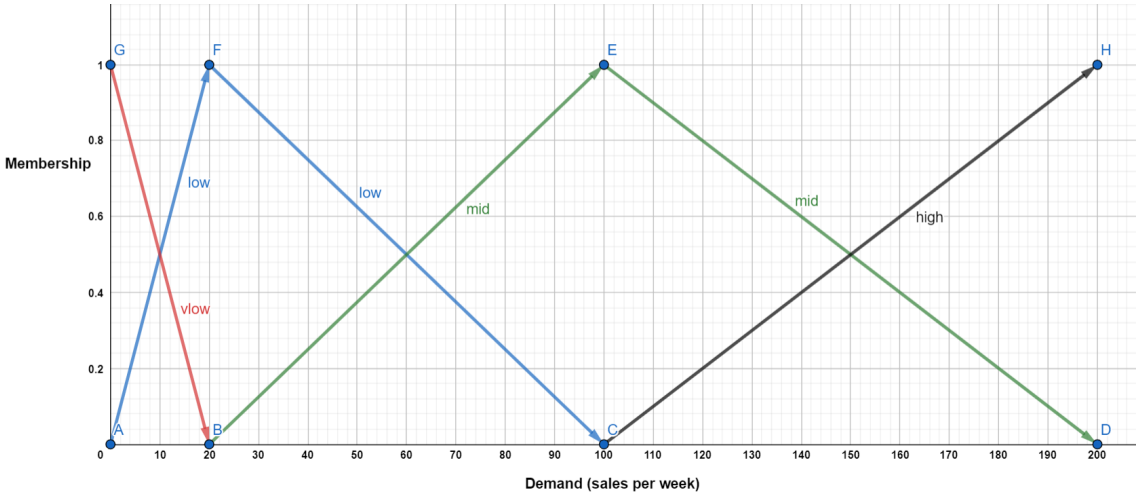
First Input, `I 1`

- Represents demand
- quantitatively, defined as number of products sold per week.
- Linguistic terms used and coefficient for each linguistic term, in `I 1`
  - for sales 0 to 20: `“VLOW”` ; $a_1=1$
  - for sales 0 to 100: `"LOW"`; $a_2=2$
  - for sales 20 to 200: `"MID"`; $a_3=3$
  - for sales 100 to 200: `"HIGH"`; $a_4=5$
- Note that any sale higher than 200 will be taken as `"HIGH"` with membership value 1.

- This input is different than other inputs since the output is inversely related (note that discount will be lowered as the sale increases and vice versa), unlike other inputs. And due to this reason we have normalized it as `I 1 =200-I 1` (after calculating membership value for it).

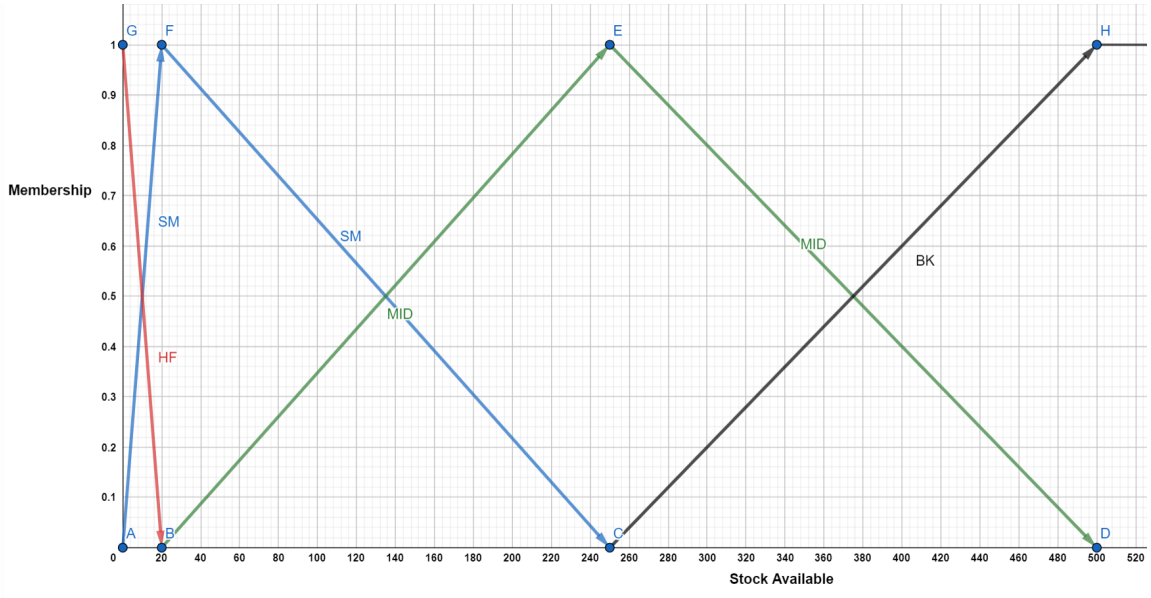
```
I1 = 200 - I1;  //normalizing I1
```

- Here is the graph for membership value vs number of sales for different linguistic terms



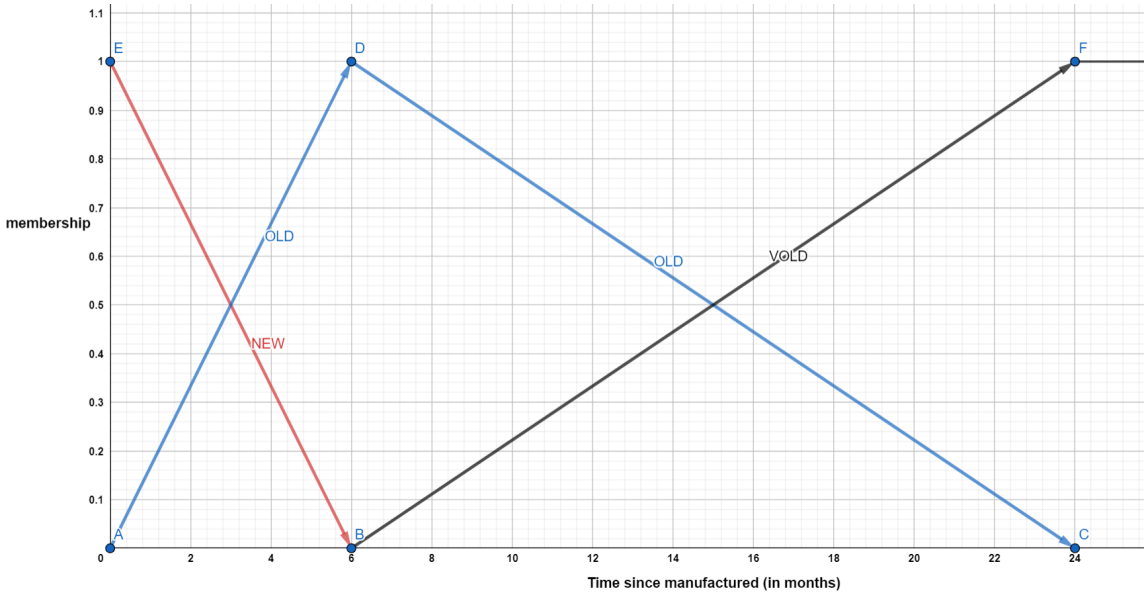
Second Input, `I 2`

- Represents stock available
- quantitatively, defined as quantity of product available with the seller.
- Linguistic terms used and coefficient for each linguistic term, in `I 2`
  - for stock 0 to 20: `"HF"` standing for **H**and**F**ul, b_1=1
  - for stock 0 to 250: `"SM"` standing for **SM**all, b_2=3
  - for stock 20 to 500: `"MID"`, b_3=5
  - for stock 250 to 500: `"BK"` standing for **B**ul**K**, b_4=5
- Note that stocks bigger than 500 will be taken as `"BK"` with membership value 1.
- Here is the graph for membership value vs no. of sales for different linguistic terms in `I 2`



Third Input, `I 3`

- Represents how old/new the product is.
- quantitatively, defined as number of months the product has been in the warehouse.
- Linguistic terms used and coefficient for each linguistic term, in `I 3`
  - for stock 0 to 6mo old: `"NEW"` , $c_1=1$
  - for stock 0 to 24mo old: `"OLD"` , $c_2=3$
  - for stock 6 to 24mo: `"VOLD"`, $c_3=5$
- Products older than 24 months will considered `"VOLD"` with membership value 1.
- Here is the graph for membership value vs no. of sales for different linguistic terms in `I 3`



Explaining the `<Code>`

The `main()` function

- takes inputs from the keyboard and limits the maximum value each input can take

```
//Taking Inputs
float I1, I2, I3;
cout << "What is the number of sales per week for the product? (0-200):"; cin >> I1;
cout << "How large is the stock of product? (0-500):"; cin >> I2;
cout << "How long has the product been in the warehouse? (0-24mo):"; cin >> I3;

//Normalizing Inputs
if (I1 > 200) I1 = 200; //sales greater than 200 is taken as 'high' sale with mu=1;
if (I2 > 500) I2 = 500; //stock greater than 500 is taken as 'bulk' with mu=1;
if (I3 > 24) I3 = 24;   //product older than 24 mon is taken as 'very old' with mu=1;
```

- variables `m1`, `m2` and `m3` of data type `map` stores membership value against each linguistic terms in first, second and third input respectively. For example, for input `I 1 =3`, `m_1` has data: `m1["VLOW"]=0.5`, `m2["LOW"]=0.5`, `m3["MID"]=0` and `m4["HIGH"]=0`. Calculation of membership function is done by the functions `update_m1`, `update_m2` and `update_m3`.

```
//map data structure to store membership value for each linguistic term
map<string, float> m1, m2, m3;
update_m1(m1, I1);
update_m2(m2, I2);
update_m3(m3, I3);
```

- To store `weights` of different combination of possible solution (e.g. a combination could be I_1 is `"VLOW"`, I_2 is `"BL"` and I_3 is `"OLD"`) we are using the the `weights`. Weight is defined as product of membership function for all combinations of input. Thus, we the `weights` variable will store $4 * 4 * 3 = 48$ values. Note that we are storing all elements even if it is zero.

```
//calculating weights for each input combination to produce different outputs
vector<float> weights; //to store weights
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        for (int k = 0; k < 3; k++) {
            weights.push_back(m1[lings1[i]] * m2[lings2[j]] * m3[lings3[k]]);
        }
    }
}
```

- as commented in the code, `coeff` variables store coefficients for each linguistic term in each variable. This makes it easier to change coefficient values without messing with the whole code.

```
//higher the coefficient higher the effect of that linguistic term in increasing output (discount)
float coeff1[] = { 1,2,3,5 };  //storing coefficients for each linguistic term of I1
float coeff2[] = { 1,3,5,7 };  //storing coefficients for each linguistic term of I2
float coeff3[] = { 1,3,5 };    //storing coefficients for each linguistic term of I3
```

- `y_value` variable stores `y_value` (which has been defined in the book). Technically, we should have calculated and stores only `2*2*2=8` `y_value`s, but we are calculating `4*4*3=48` `y_value`s. As we'll see later the excess data will automatically be evaluated to zero while calculating the final output.

```
//calculating y_value for each combination
vector<float> y_value;
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        for (int k = 0; k < 3; k++) {
            y_value.push_back(coeff1[i] * I1 + coeff2[j] * I2 + coeff3[k] * I3);
        }
    }
}
```

- Now the data stored so far in `y_value` and `weights` are sent to other functions to calculate the final output and to print the formatted answer.

```
//Now calculating the final discount % based on weights and and y-value's.
float discount = calc_discount(weights, y_value);

//prints the dicount value beautifully
display_discout(discount);
```

The `calc_discount()` function

Using the `weights` and the `y_value` data, we can calculate the final output, the discount%.

```
float result = 0;
float weights_sum = 0;
for (int i = 0; i < weights.size(); i++) {
    result += (weights[i] * y_value[i]);
    weights_sum += weights[i];
}
result = (result / weights_sum);
```

The above piece of code calculates, $\Sigma w_i * y\_value_i$ and divides, it the to sum of `weights`. Note that we stated earlier, that we are calculating excess value amount of `y_value`. These 'extra' `y_values` will have its `weight` zero, thus making the product `weights[i] * y_value[i]` zero. Thus, we have contribution from only the relevant `y_values`.

The value of `result` is not the final discount% we need. We need to process this data. Since $\Sigma w_i * y\_value_i$ is an empirical formula, we need to normalize its output such that for any input the output lies in the range 0 to 60.

The minimum value of `result` will be for the limiting case when $I_1=\infty$, $I_2=0$ and $I_3=0$, i.e. when the product is doing excellent in the market and the stock of the product is almost completely sold out and the product is also new. Then the discount offered should be zero.

The maximum value of `result` will be for the limiting case when $I_1=0$, $I_2=\infty$ and $I_3=\infty$. i.e. when the not a single piece is selling, the stock available with the seller is very large and

also the products are so old that it has become out-dated. In this case, the discount offered should be maximum i.e. 60%.

Now after getting the minimum (which we found to be 0) and the maximum (which we found to be 2970) value for `result`, normalize it by such that the range is 0-60 as shown.

$$result = \frac{result - result_{min}}{result_{max}} \times 60$$

$$result = \frac{result - 0}{2970} \times 60$$

```
result = (result - 0) * 60;
result = result / 2970;
return result;
```

The `find_mu()` function

- Finds the membership value using the properties of triangle.
- gets called by `update_m` function.

The `display_discount` function

Based on the value of discount, it judges whether the discount is LOW, MID, HIGH or HEAVY and prints it the screen in a formatted manner.