## Algorithms

### Recommender Engine

#### Introduction

As a car rental solution, CRP has to be able to introduce the most suitable vehicles to customers. This will helps improving not only the customers’ but also the providers’ satisfaction, and ultimately gaining more transactions, as well as interest for our application. Consequently, a recommender engine is necessary.

#### Common approaches

Two common approaches on designing recommender solution are *Collaborative filtering* and *Content-based filtering*.

Collaborative filtering methods are based on collecting and analyzing a large amount of information on users’ behaviors, activities or preferences and predicting what users will like based on their similarity to other users. [[1]](#_References)

Content-based filtering methods are based on a description of the item and a profile of the user’s preference. In a content-based recommender system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). [[2]](#_References)

Collaborative approach does not require analyzing the content of item that it recommend since it is based entirely on the user’s information. However, exactly because it depends solely on the data generated by users, it suffers the three common problem of computing, namely *cold* *start* (Lack of user’s interaction in the beginning), *scalability* (Scale badly when the number of user and their interaction increase), and *sparsity* (The number of item is too large compared to the number of user). Considering the low-number-of-transaction nature of car rental service when compares to other kind of services, scalability and sparsity issues can be evaded.

Content-based approach works well even under scarce user’s interaction environment since its recommendations are based on the items’ description. Its issue of not being able to recommend items with different content type (For instance, car and phone) is also not a problem, since CRP only has one type of item.

#### Solution’s approaches

CRP’s recommender design take a hybrid approach between content-based and collaborative.

Since our system’s recommended targets, namely vehicles, have many attributes that we can take advantage of (Brand, number of seat, color…), we initially tackles the problem using content-based methods. These includes *Vector space model* and *tf-idf.*

##### Represent items using vector space model

We first abstract the vehicles and their attributes by applying *Vector space model* [[3]](#_References), an item presentation algorithm.

Each vehicle is modeled as a vector (Refer to as *master vector* from now) in a multi-dimension space (Refer to as *vector space* from now), with each dimension corresponds to an attribute. If a vehicle has an attribute, the component vector (Refer to as *attribute vector* from now) corresponding to that attribute will has non-zero length.

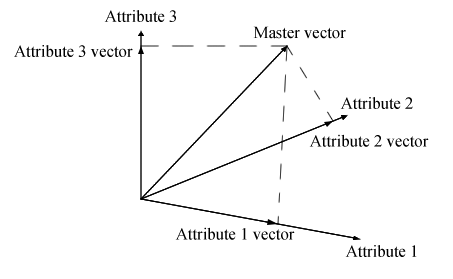


Figure 1 – Example for 3-dimensional vector space

There are several weighting scheme that can be used to calculate the attribute vector’s length. Amongst them, the most popular one is *tf-idf* (Term frequency – Inverse document frequency)[[4]](#_References), which. In our solution, tf-idf is applied with *binary scheme* (Further explanation will be given in *solution’s design* section).

##### Customer profile

A customer profile is another vector in the *vector space* which indicate a particular customer’s interest in vehicle, like which color or which fuel type that he has more affinity with.

In vector space model, the angle between 2 vectors determine the similarity between them. This means the smaller the angle between a vehicle’s *master vector* and a customer profile, the more similar that vehicle is to the customer’s reference.

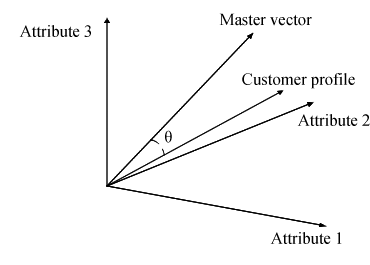


Figure 2 – Example for customer profile vector

It is common to use the cosine of this angle to represent their similarity, since their values are between 1 and -1. This form allows us to tell at how many percent a customer will like a vehicle (Positive value), or dislike it (Negative value). Our recommender engine will find the cosine between each vehicle’s *master vector* and a customer profile, then recommend vehicles with the highest cosine value to that customer.

##### Mix in collaborative element

To further improve the diversity of our solution’s recommendation, we apply collaborative methods into our content-based solution. Our collaborative approach work under the assumption that other customers that has booked the same vehicle with this customer (Refer to as neighbor from now) will have similar vehicle reference; and the more *neighbors* a vehicle has, the more similar it is to this customer’s reference.

Our collaborative approach introduces new attributes into the *vector space*. Each of these new attribute represent a neighbor and whether the item has been booked by this neighbor before.

|  | Neighbor 1 | Neighbor 2 | Neighbor 3 | Neighbor 4 |
| --- | --- | --- | --- | --- |
| Vehicle 1 | Yes | Yes | No | No |
| Vehicle 2 | No | Yes | No | No |
| Vehicle 3 | Yes | Yes | Yes | No |

Table 1 – Example of vehicle with 4 neighbor attributes

#### Solution’s design

##### How to calculate cosine of angle between 2 vectors

Cosine of the angle θ between 2 vectors and in n-dimension *vector space* can be calculated as follow:

where is dot product of 2 vectors, and is the norm of each vector.

The dot product can be calculated as follow:

The norm of a vector can be calculated as follow:

##### Apply tf-idf weighting scheme

Applying tf-idf, length, or ‘weight’ of each *attribute vector* of can be calculated as follow:

idf (Inverse document frequency) can be calculated as follow:

with being the total number of item and being the total number of item that has attribute i.

The scheme to calculate tf will be discussed in the next section.

##### Binary representation of attributes

In our approach, raw attributes can only either appear or does not appear in an item. This leads to representing them as binary values. Under this form, we can apply *binary tf weighting scheme* of *tf-idf*, where tf weight equals the raw binary value.

|  | 4-seat | 7-seat | Gasoline | Diesel | Neighbor 1 | Neighbor 2 |
| --- | --- | --- | --- | --- | --- | --- |
| Vehicle 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| Vehicle 2 | 1 | 0 | 0 | 1 | 0 | 1 |
| Vehicle 3 | 0 | 1 | 1 | 0 | 1 | 1 |
| Vehicle 4 | 0 | 1 | 1 | 0 | 1 | 0 |
| Vehicle 5 | 1 | 0 | 0 | 1 | 0 | 0 |

Table 2 – Example tf values of vehicles with 6 binary attributes

With this approach, there is also no need to apply normalization to eliminate item’s size bias (For document-like item, the total number of ‘word’ in them varies, meaning bigger document will more likely to have more ‘hit’(tf) for each search term/attribute. However, with binary attributes, an item either ‘has’ or ‘does not have’ an attribute. In other words, tf is limited to 1 and 0, and therefore has no such bias).

| **attribute** | | 4-seat | 7-seat | Gasoline | Diesel | Neighbor 1 | Neighbor 2 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **tf** | Vehicle 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| Vehicle 2 | 1 | 0 | 0 | 1 | 0 | 1 |
| Vehicle 3 | 0 | 1 | 1 | 0 | 1 | 1 |
| Vehicle 4 | 0 | 1 | 1 | 0 | 1 | 0 |
| Vehicle 5 | 1 | 0 | 0 | 1 | 0 | 0 |
| **df** | | 3 | 2 | 3 | 2 | 3 | 3 |
| **idf (With D = 5)** | | 0.097 | 0.222 | 0.097 | 0.222 | 0.097 | 0.097 |
| **weight** | Vehicle 1 | 0.097 | 0 | 0.097 | 0 | 0.097 | 0.097 |
| Vehicle 2 | 0.097 | 0 | 0 | 0.222 | 0 | 0.097 |
| Vehicle 3 | 0 | 0.222 | 0.097 | 0 | 0.097 | 0.097 |
| Vehicle 4 | 0 | 0.222 | 0.097 | 0 | 0.097 | 0 |
| Vehicle 5 | 0.097 | 0 | 0 | 0.222 | 0 | 0 |

Table 3 – Example weight of 5 vehicles with 6 binary attributes

##### Build the customer profile

We can build a user profile by using that user’s booking history as reference. Every booking has all the necessary data to construct an item vector similar to vehicle vector.

| # | Vehicle | 4-seat | 7-seat | Neighbor 1 | Neighbor 2 | Star |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Vehicle 1 | 1 | 0 | 1 | 1 | 2 |
| 2 | Vehicle 5 | 1 | 0 | 0 | 0 | 1 |
| 3 | Vehicle 3 | 0 | 1 | 1 | 1 | 4 |
| 4 | Vehicle 2 | 1 | 0 | 0 | 1 | 5 |
| 5 | Vehicle 3 | 0 | 1 | 1 | 1 | - |

Table 4 – Example of 5 bookings with 4 attributes

The bookings also have star-rating, which we can utilize to determine whether the customer liked or dislike the booking. In our approach, we assume that a rating lower than 3-star indicates *dislike* (-1), higher than 3-star indicates *like* (1) and equals 3 or empty rating means *neutral* (0).

| # | Vehicle | 4-seat | 7-seat | Neighbor 1 | Neighbor 2 | Star | Like |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Vehicle 1 | -1 | 0 | -1 | -1 | 2 | -1 |
| 2 | Vehicle 5 | 1 | 0 | 0 | 0 | 1 | -1 |
| 3 | Vehicle 3 | 0 | 1 | 1 | 1 | 4 | 1 |
| 4 | Vehicle 2 | 1 | 0 | 0 | 1 | 5 | 1 |
| 5 | Vehicle 3 | 0 | 0 | 0 | 0 | - | 0 |

Table 5 – Example tf of 5 bookings with 4 attributes applying like-dislike scheme

tf of each of the customer profile’s *attribute vector* can be calculated as tf of sum of tf of every booking’s attribute vector of the same dimension. Since these sum are no longer binary, we calculate customer profile’s of dimension i using *log normalization* scheme instead:

In this equation, m is the total of booking. This equation has been modified from the original scheme to accomplice negative value. The customer profile’s vector can be calculated using our default tf-idf weighting scheme.

| **Attribute** | # | Vehicle | 4-seat | 7-seat | Neighbor 1 | Neighbor 2 |
| --- | --- | --- | --- | --- | --- | --- |
| **Booking’s tf** | 1 | Vehicle 1 | -1 | 0 | -1 | -1 |
| 2 | Vehicle 5 | -1 | 0 | 0 | 0 |
| 3 | Vehicle 3 | 0 | 1 | 1 | 1 |
| 4 | Vehicle 2 | 1 | 0 | 0 | 1 |
| 5 | Vehicle 3 | 0 | 0 | 0 | 0 |
| **df** | | | 3 | 1 | 2 | 3 |
| **idf (D=5)** | | | 0.097 | 0.398 | 0.222 | 0.097 |
| **Customer profile** | **tf** | | -1 | 1 | 0 | 1 |
|  | **weight** | | -0.097 | 0.398 | 0 | 0.097 |

Table 6 – Example weight for attribute vectors of a customer profile

##### Calculate vehicle’s score

As mentioned, the score used to recommend vehicle will be the cosine between the vehicle master vector and the customer profile.

| Attribute | | | 4-seat | 7-seat | Neighbor 1 | Neighbor 2 |
| --- | --- | --- | --- | --- | --- | --- |
| **Customer profile** | | | -0.097 | 0.398 | 0 | 0.097 |
| **Vehicle** | **tf** | Vehicle 1 | 1 | 0 | 1 | 1 |
| Vehicle 2 | 1 | 0 | 0 | 1 |
| Vehicle 3 | 0 | 1 | 1 | 1 |
| Vehicle 4 | 0 | 1 | 1 | 0 |
| Vehicle 5 | 1 | 0 | 0 | 0 |
| **df** | | 3 | 2 | 3 | 3 |
| **idf (D=5)** | | 0.097 | 0.222 | 0.097 | 0.097 |
| **weight** | Vehicle 1 | 0.097 | 0 | 0.097 | 0.097 |
| Vehicle 2 | 0.097 | 0 | 0 | 0.097 |
| Vehicle 3 | 0 | 0.222 | 0.097 | 0.097 |
| Vehicle 4 | 0 | 0.222 | 0.097 | 0 |
| Vehicle 5 | 0.097 | 0 | 0 | 0 |
| **Score** | Vehicle 1 | 0 | | | |
| Vehicle 2 | 0 | | | |
| Vehicle 3 | 0.889907 | | | |
| Vehicle 4 | 0.866331 | | | |
| Vehicle 5 | -0.23042 | | | |

Table 7 – Example score for 5 vehicles with 4 attributes

A positive value shows us the probability the customer will like the vehicle, while a negative one shows us the probability the customer will dislike the vehicle.

#### Algorithm’s time complexity

Consider a *vector space* with n attributes, k vehicles, a customer with m bookings in her booking history, and Math.Log10(double) and Math.Sqrt(double) has O(1) time complexity, we can calculate the time complexity of this recommender algorithm.

##### Build customer profile

| **Step** | **Complexity** |
| --- | --- |
| Calculate each attribute vector of the profile | O(m) |
| * Calculate | O(m) |
| * Calculate | O(m) |
| * Calculate | O(1) |
| * Calculate | O(m) |
| * Calculate | O(m) |
| * Calculate | O(1) |
| * Calculate | O(1) |
| **Total** | **O(nm)** |

Table 8 – Time complexity of **Build customer profile** step

##### Build vehicle vectors

| **Step** | **Complexity** |
| --- | --- |
| Calculate each vehicle vector | O(nk) |
| * Calculate each attribute vector of | O(k) |
| * Calculate | O(k) |
| Calculate | O(k) |
| Calculate | O(1) |
| * Calculate | O(1) |
| **Total** | **O(nk2)** |

Table 9 – Time complexity of **Build vehicle vectors** step

##### Score vehicles

| **Step** | **Complexity** |
| --- | --- |
| Calculate norm | O(n) |
| Calculate each vehicle’s score | O(n) |
| * Calculate norm | O(n) |
| * Calculate dot product | O(n) |
| * Calculate vehicle’s score | O(1) |
| **Total** | **O(nk)** |

Table 10 – Time complexity of **Score vehicles** step

##### Overall time complexity

The overall time complexity of our algorithm is **O(n(k2 + k + m))**. This complexity has 2 pain points. The first being k2, which can somehow be reduced by applying normal filtering first to lessen the number of vehicle in vector space. The second pain point is the number of collaborative attributes in n.

n is the sum between the content-based attributes presented naturally on every vehicle and the collaborative attributes which are the neighbors a vehicle has. The more booking the customer makes, the more neighbors he may have, and the more n will increase.

Conclusively, this algorithm’s scalability is quite poor, but still stay in the typical zone of collaborative recommendation solution. In the future that the system will increase in size, there will be a need for either an upgrade of hardware and software to increase computing power, or the development a new solution.

# References

[1] [https://en.wikipedia.org/wiki/Recommender\_system#Collaborative\_filtering](https://en.wikipedia.org/wiki/Recommender_system%23Collaborative_filtering)

[2] [https://en.wikipedia.org/wiki/Recommender\_system#Content-based\_filtering](https://en.wikipedia.org/wiki/Recommender_system%23Content-based_filtering)

[3] <https://en.wikipedia.org/wiki/Vector_space_model>

[4] <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>