# Content Based Recommendation System

How to build your first recommender from scratch?

Jamila Omrane
AI/ML Engineer at Talan Innovation Factory - Talan Tunisia

September 2019

## 1 First Things First: What's a recommendation System?

As Defined by the Recommender System Handbook, *Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read.* Recommendation systems fall down, mainly, into two major categories based on the approach of handling the subject:

- Collaborative Filtering: This type of recommender tend to propose, to an active user, items that other users with similar tastes liked in the past. This similarity is usually computed based on users' ratings of the items. This category of RS is out of the scope of this paper.
- Content Based: This type of recommender systems recommends an item to the user based on a description of the item and a profile of the user interests. We will be discussing this category of recommender in the rest of the paper.

## 2 Content Based Recommendation system

Content based engines, generally, consist of three leading components:

1. **The content analyzer:** It has the role of constructing the item profile. Here, we are trying to find a representation of the item and we are responding to the question: what attributes describe the most our item? This can be done, for example, by analyzing the description associated to the item, or maybe just by using the category or the brand.
2. **The profile learner:** In fact, our recommender tries to deduct a profile for the user that, kind of, encapsulates his preferences or even his personality traits. Here, many techniques can be used. As an example, we can analyse user behaviour during his browsing, the ratings he gave to some items, or even, his reviews and comments upon some items, etc.
3. **The filtering component:** It is responsible for suggesting relevant items to the user by matching the user profile against the item representation (or profile).

## 3 Our implementation

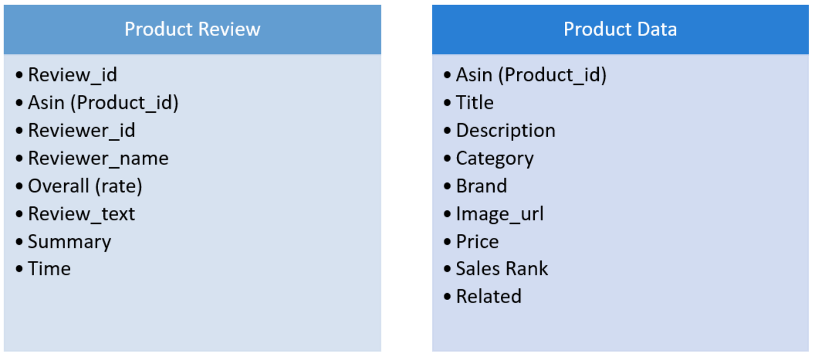
You can find the jupyter notebook of the recommender system implementation in the following link https://github.com/Talan-Innovation-Factory/Content-Based-Recommendation-System.

### 3.1 The Amazon dataset

"Good" data remains the secret for a good ML/DL model. That's why we started by looking for a good dataset, and good, in our case, basically means complete or sufficiently rich that we can make use of it in an exhaustive yet optimized way . After some research, we did land on a dataset provided by amazon.
Two main tables interest us in this dataset, which are "Product Data" and "Product Review". The reason why these two are important, is that they provide us with:

- Data about the product, so we can build an item profile,
- And data, about the reviews given by the customers to the different items, so we can conclude a user profile.

And, as we mentioned it before, these two bricks are crucial for such a recommendation system.
Hereafter, we give a representation of the two tables:



| Product Review |
|---|
| • Review_id |
| • Asin (Product_id) |
| • Reviewer_id |
| • Reviewer_name |
| • Overall (rate) |
| • Review_text |
| • Summary |
| • Time |

| Product Data |
|---|
| • Asin (Product_id) |
| • Title |
| • Description |
| • Category |
| • Brand |
| • Image_url |
| • Price |
| • Sales Rank |
| • Related |

## 3.2 Item Profiling

Obviously, in order to construct an item profile, we have to refer to the "Product Data" table, since it gives us many details about the products.
We think that the "Description" is the most important feature from which we can retrieve the most of information about an item. For that reason, we choose to use this field in addition to the "Category" field. There are some other fields that might seem interesting, namely the "Title" and "Brand", but we decide to not use them because they are generally mentioned in the "Description" field.

### Data pre-processing

Every ML/DL exercice starts with a pre-processing step, in which, we transform our data to a set which is ready for the training phase. So we proceed as following:

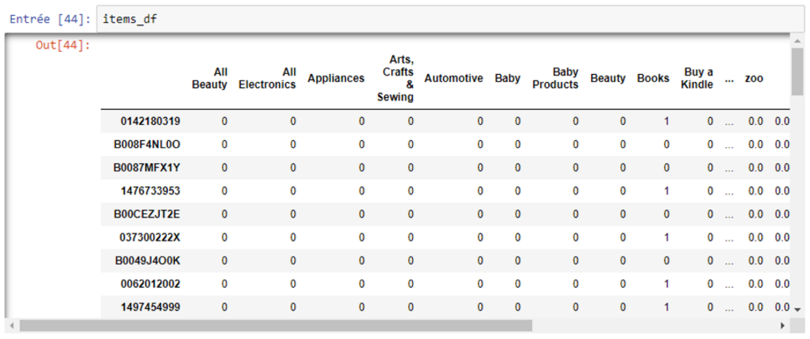
- First, we removed items with NaN values.
- Next, we applied a one hot encoder (OHE) on the "Category" field, which gave us 33 other fields representing the unique categories we have.

### Information Retrieval

- After that, we moved on to the "Description" field. As computers cannot handle textual data, we have to found out a solution to represent the text in a computer understandable way without losing its sense or its semantics. A common way to do this is "Text Embedding" . We choose to use TF-IDF as a Text Embedding technique.
  Before feeding our textual data to the TF-IDF vectorization, we need to prepare it, as well:
  - We started by building our corpus, which is an array of descriptions.
  - Then, we transformed it to lowercase and removed numbers, punctuation, and extra white spaces.
  - After that we passed to the tokenization, which means every description was splitted to a list of words.
  - Last but not least, we applied a stemmer, so every word is reduced to its stem.

  Next, we fit our corpus to the TF-IDF vectorizer, which resulted in a 15306 dimensions vector for each description.

- Finally, we concatenated the two matrices obtained by the OHE and the TF-IDF. The result was a dataframe of 10829 rows X 15339 columns (Obviously, here 10829 is the number of items ). The following figure shows the data we obtained:



```
Entrée [44]: items_df
Out[44]:
```

| | All Beauty | All Electronics | Appliances | Arts, Crafts & Sewing | Automotive | Baby | Baby Products | Beauty | Books | Buy a Kindle | ... | zoo | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0142180319 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ... | 0.0 | 0.0 |
| B008F4NL0O | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.0 | 0.0 |
| B0087MFX1Y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.0 | 0.0 |
| 1476733953 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ... | 0.0 | 0.0 |
| B00CEZJT2E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.0 | 0.0 |
| 037300222X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ... | 0.0 | 0.0 |
| B0049J4O0K | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0.0 | 0.0 |
| 0062012002 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ... | 0.0 | 0.0 |
| 1497454999 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ... | 0.0 | 0.0 |

10829 X 15339

At this level, we can provide users with recommendations similar to an item they are viewing per example. Since we have a vectorial representation of each item, we can evaluate similarities by using any measure of distance or similarity (cosine similarity is more adequate in this case. In fact, it is recommended when we are dealing, majorly, with textual data).

## 3.3 User Profiling

In order to build a profile representing the user, we have to rely on the review he gave to the items. We focused on the "overall" feature from the table "Product Review" which represents the rate that the user, identified by "reviewer_id" field, granted to the item.
So, having our (reviewers X items) matrix, we only have to multiply it by the items' profiles' matrix (items X attributes). The result here is (reviewers X attributes) matrix, where each row represents a user profile. And, evidently, we can deduce the similarity ( or difference) between two users' tastes, just by computing the similarity ( or difference) between their representative profiles. More than that, considering that the user profile and the item profile have the same attributes, we can measure how much an item can interest a user only by calculating the distance (or similarity) between them. And that's exactly what we did.

## 3.4 Recommendation

In fact, in order to recommend to the user the items that match to his personality, we measured the similarity between the user and every single item (based on their respective profiles). Then, we suggested the K most close items. in our

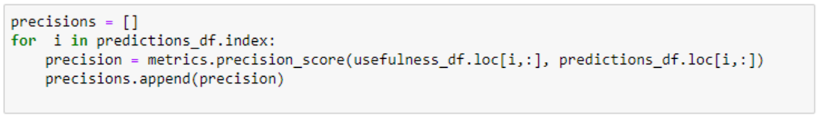
case K = 10.
To do so, we used a “sklearn.neighbors” algorithm that enables us to find the K nearest neighbor regarding a user profile.
**Recommendations Done! But how can we make sure if our recommender is doing well?**
We do think that evaluating an ML model is as complicated as creating it. The first and major challenge remains in choosing the evaluation metric.
The table below depicts the confusion matrix of a recommendation system:

| | **Recommended** | **Not Recommended** |
|---|---|---|
| **Used** | True Positive (TP) | False Negative (FN) |
| **Not Used** | False Positive (FP) | True Negative (TN) |

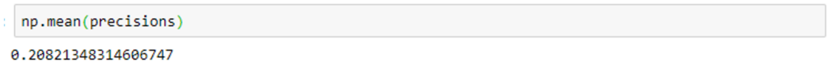
Logically, in recommendation systems, we don’t need to focus on the True Negative rate . We don’t need to pay a lot attention to the False Positives neither, because, in this case the recommendation action is not very costly (actually, we are not sending sms or emails). But, we do care about the True Positive rate, since it tells us how much does our recommender understands its customers.
Considering all these facts, we decided to use the “Precision” as an evaluation metric for our recommender. Precision is defined as below:

$$Precision = \frac{TP}{TP+FP}$$

While measuring the Precision of our recommender, we obtained this value:



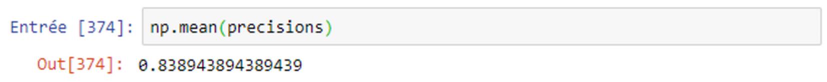
```
precisions = []
for  i in predictions_df.index:
    precision = metrics.precision_score(usefulness_df.loc[i,:], predictions_df.loc[i,:])
    precisions.append(precision)
```



```
np.mean(precisions)
```

```
0.20821348314606747
```

In order to find out the reason of this low value, we looked deeper in our data, and we found out that we are recommending 10 items to each user, while many users are only buying 2 or 3 items; this will increase the rate of False Positives and reduce that of True Positives, which will automatically decrease the precision. Thus, we have to find out a way capable of determining the real precision of our system. And the idea is quite simple: Measuring the precision at a certain value K, which means, if the client bought 10 items, we recommend 10 items and calculate the precision, and so on for different values of K, and our recommender precision will be the mean of the different precisions at K.
We tried to measure the precision at K = 10, and what we obtained is : 0.84 of precision.



```
Entrée [374]: np.mean(precisions)
   Out[374]: 0.838943894389439
```

Pretty good!

## 4 Retrospective

What we came up with, in the demarche we adopted, is that it has one main drawback: Evaluating the precision for different values of K is expensive because measuring the precision, here, means launching different K-recommendations and evaluating each one. This led us to think about a recommender which suggests to a user a number of items equivalent to the number of items he already bought. So we can facilitate the evaluation phase. However, changing a recommender just to facilitate the evaluation doesn't make sense, furthermore, one of the objectives of recommendation systems is to increase and diversify purchase, so, we have to encourage the customer to buy by providing him more suitable products.
Another point, we are considering, is giving more interest in the False Negative rate, which can be done through the "Recall" metric. Hence, we focused on the F-measure which is , kind of, summarizing precision and recall.
We think that recommendation, in itself, here, is costly. For each user, we are calculating the distance with every single item. The performance of this recommender has not been yet assessed in production.

## 5 Conclusion

The leading idea behind this work, was the understanding and the implementation of a recommender system. Beside the existence of predisposed libraries for building content based or collaborative filtering recommenders, such as Surprise or LightFM, we chose to implement it from scratch and see what problems can occur, aiming to contribute in the resolution of existing limitations.