**Master Project: Recommendation Engines**

Adrien Lequiller

Master Student in Data Analytics & AI at EDHEC Business School

Github: https://github.com/adrienlequiller/Recommender\_Engines

Date: June 2019



Motivations

This document is my personal final MSc end-of-year project. I chose to focus on recommender engines that represent real leverages to personalize the use of the internet. They are still ongoing issues for data scientists and are subject to Artificial Intelligence to make them ever-more profile-based. Moreover, and considering the courses we attended in this MSc – Statistical Learning, Data Mining, Machine and Deep Learning among others – this subject is, according to me, really complete and encompass lots of topics we have been discussing. Therefore, it is both a pretty hot topic that is worth investigating and a cross-cutting theme from a business & analytics academic point of view.

Despite the fact that I have always been amazed and wondering how streaming websites were able to recommend our next favorite song or the series we will be watching with enthusiasm for the next couple of weeks, I have a more personal reason. At the very beginning of the academic year, during a web-dev bootcamp, we implemented with some friends a platform that enable to manage job opportunities using a chrome extension. We thought about implementing recommendations to help users in their job searches. However, we didn’t have the tools yet to properly deal with such an issue. Thus, I also chose this topic to give a kind of answer to our start-of-the-year questioning and concerns.

**Table of contents**

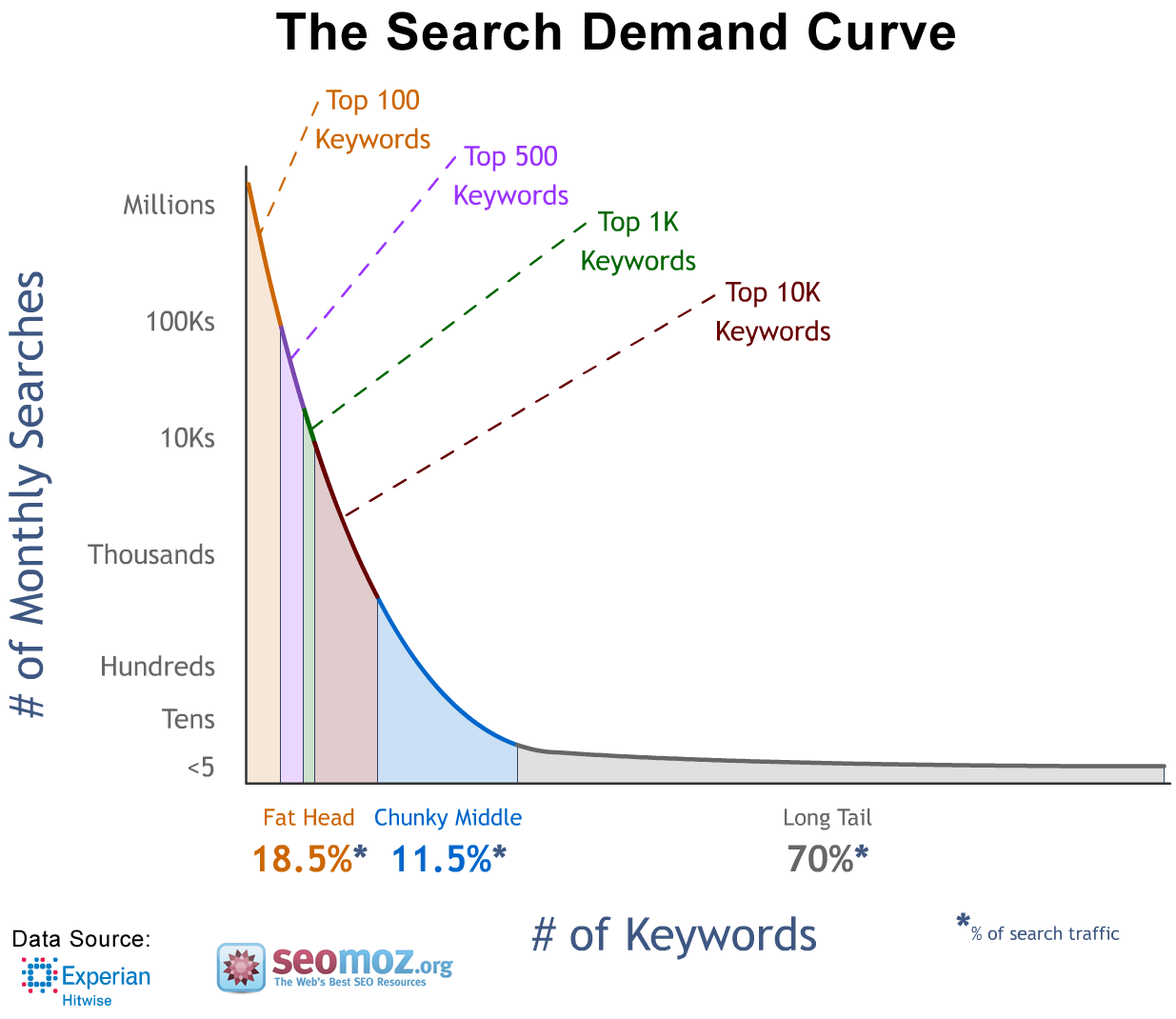
1. **Introduction**
2. Recommendation systems: objectives, techniques and evaluation
   1. *Definition and objectives*
   2. *Techniques and principles*
   3. *How to evaluate the quality of recommendation systems?*
3. Mathematical framework and models behind recommendation techniques
   1. *Introduction*
   2. *Content-Based Filtering*
   3. *Collaborative Filtering*
4. Application to a Movie Dataset (MovieLens)
   1. *Introduction*
   2. *Content-Based movie recommender*
   3. *Collaborative movie recommender*
5. Conclusion
6. References & Appendix

0. Introduction

35% of Amazon's revenue is generated by its recommendation engines, it is 75% of watching items for Netflix. [1]

“We have discovered through the years that there is tremendous value to our subscribers in incorporating recommendations to personalize as much of Netflix as possible,” say by Xavier Amatriain and Justin Basilico (Personalization Science and Engineering) in their Netflix Tech Blog. [2]

Information retrieval systems (SEO companies such as Google, Altavista,...) have been using recommendation algorithms for nearly more than two decades now. Indeed, it tackles the issue of 'giving' to the users what they are looking for but it means backend algorithms that manage to classify contents in order for the search engine to respond as relevantly as possible to the user query. These companies partially solved the problems of interest for the user. As a matter of fact, first algorithms were based on the simple statement that the majority of contents of interest represent a minority of all available contents. We often refer it to the Long Tail problem. However, what they didn't really deal with were the issues of personalization and prioritization (user's interests and preferences).



Basically, recommender systems' (RS) objetive is to filter the overwhelming content that an entity has to propose to the user but based on the user profile this time. Indeed, comparing to search engine optimization (at least at the early stages), every content can have equal interest independently but one (or more of course) may have much more interest for a specific user than the others. That's why entities that need/see an interest in it try to resolve through recommendation systems.

"Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of dynamically generated information according to user’s preferences, interest, or observed behavior about item. Recommender system has the ability to predict whether a particular user would prefer an item or not based on the user’s profile, " according to the definition of The Egyptian Informatics Journal. [3]

Even if the underlying interests for the business environement are obvious, there also have lots of interests for non-profitable organizations because they improve decision-making, process and quality. The positive effect is both for the entity and the user (at least in general). Furthermore, such recommendation systems are increasingly used by digital companies to learn more about their customers and propose them products or services more personalized in order to drive them into continous engagement.

From e-commerce businesses (Amazon, Zalando, Decathlon,...) to entertainement companies (Netflix, Spotify,...) and social networks (Instagram, Shapr,...) but also other B2B/B2C marketplaces, they all have been developing such algorithms to predict what a customer could be interested in and try to catch its attention by proposing him new but targeted contents and enhance the global user experience.

It has lots of benefits, as users feel more acknowledged by companies: shopping online becomes today simliar to have a personal adviser. The company can leverage its different offers to best fit the customer needs and from a financial perspective it may significantly increase their revenue and return on investments (ROI).

Thus, these recommendation systems are all around us and seem to govern our shopping, listening and watching patterns nowadays. However, they represent huge amounts of work and continuous improvements to become ever more performing.

1. Recommendation systems: objectives, techniques and evaluation

1.0 Definition and objectives

**What is a recommendation engine?**

*"Recommender system is defined as a decision making strategy for users under complex information environments", The Egyptian Informatics Journal.*

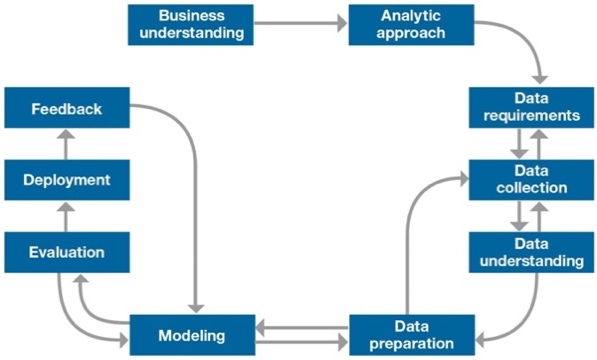
A more personal definition… A recommendation engine is a filtering system that seeks to predict a content, a product or a service that a user might be interested in regarding his profile and behaviour trrough data collected onto the internet.

In other words, it aims at feeding him with something relevant regarding his personality instead of randomly proposing him something he probably doesn't care about. It's a kind of personal advisor using its personal experience to recommend the items a company may propose.



**How does it work?**

A Recommendation engine is about statistics, data-mining and machine/deep learning and follows a 3-step process: **the collection of data, the learning phase and the prediction/recommendation phase.** It is about data science, there are usual major steps to be well implemented and it is important to note that it is also an **iterative process**. Feedbacks and arrangements are very important to improve such engines.

[4]

The collection step requires to gather data, that can be either explicit (user ratings for instance) or implicit (click through rates, pageviews,...). The more data you have, the better will be your recommendation engine as it will get ever more elements to train and predict. It's really about a 'smart' effect, because it will be able to respond more effectively to specific situations. That's why any recommendation engine quickly turn into big data projects. Every company can track differently its users or potential ones by using methods offered by others or with their own tools (from source code to online polls,…). The (open) internet has lots of advantages today because data is much easier to collect than ever. Storing is also easier thanks to the cloud and datacenters. Furthermore, new tools have made analytics easier such as Hadoop, Spark,… but also dedicated machine learnings tools that not only improve the accuracy of your algorithm by using very performing methods but also make real-time adjustments. The **learning phase** is merely about using data at hand to construct an algorithm than then will be able to do predictions (**prediction/recommendation phase**).

**When to implement one?**

Implementing such engines is worth when - among the company's objectives (and especially for internet ones) - there is a will to leverage the power of data to create better user experiences.

It involves from a company's team to be able to understand the data and play with it. It requires first and foremost, to intelligently capture the data you think could contribute to reaching your objectives. Data can be implicit or explicit. Implicit is something you can measure (like bounce rate, click, search,...), while explicit data is something you asked the user to do (react, give a grade,...).

**Which benefits?**

When set-up and configured properly, benefits are numerous for companies. It can be revenue, customer satisfaction, more adapted marketing strategies, user patterns understanding,…

On one hand, using such algorithms require less time and resources than traditional market researches but it also significantly contributes to increasing conversion rates. On the other hand, thanks to personalization and ‘genius recommendations’, it triggers more engagement and satisfaction from the customer bringing about more loyalty and retention down the road.

When intelligently implemented, meaning if a company manages to communicate with its customer as an individual, it becomes a real competitive advantage.

1.1 Techniques and principles

**Type of Recommender Systems**

There are plenty of recommendation engines - each of them using different methods to best fit a particular issue. But basically, the three main important ones are : Content-based Filtering, Collaborative Filtering and Hybrid Recommendation systems.

1.1.1 Content-based Filtering

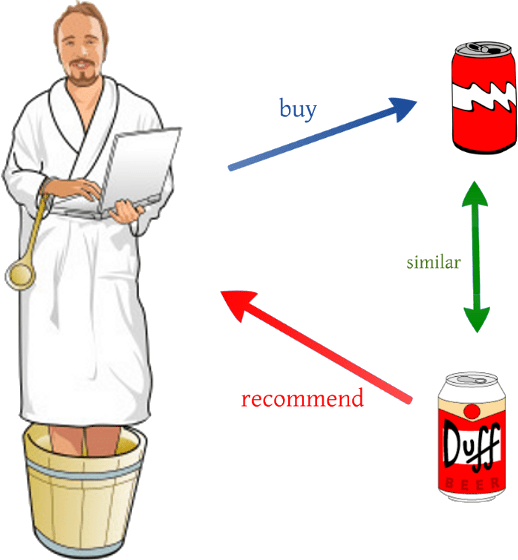
Content-based systems leverage product information for its recommendations. Basicallty, it consits in looking at the set of products and use product information to find similar ones.

Let's assume you listened to 100 Blues & Soul songs and 10 of rock music since you are part of Spotify community, then you are likely to see appear in your recommendations (if you haven't been listening to it yet) the Pain in My Heart Album of Otis Redding... It uses the similarity on the basis of the core features/inherent characteristics of items between them. Thus, when a user opens a link to a specific product, the algo recommends the products most similar to the viewed one.

There are many ways to build such systems, but the limit is usually set by the number of characteristics (features) we have about the product. Indeed, it requires to have tags/metadata about the items in order for the algorithm to be able to make relationships between them.

For example, regarding a movie, features could be the date and country of production, the genre (Documentary, Action,...), the realisator (Clint Eastwood, James Cameron,...) and many others. Once, features vectors are defined for all products, we can measure the relationship between each pair of them.

The advantage of this method is that we don't really need a lot of transactions to construct the model as it's based on the features of elements at hand. And it has the advantage to be transparent, and the user may understand why some items are proposed to him. The drawback - linked to the advantage first mentionned – is that we can not improve it by adding more transactions as it's feature-centric. Furthermore, and more importantly, it can lead to over-specialization. Indeed, a product that has similar tags can be efficient if it's about newpapers because the user may have interests (economy,science,...) but becomes useless when it comes to recommend coca because the user already added sprite to his basket and both are soda. It would have been smarter to look at usual items other users may buy with sprite : pizza for example...

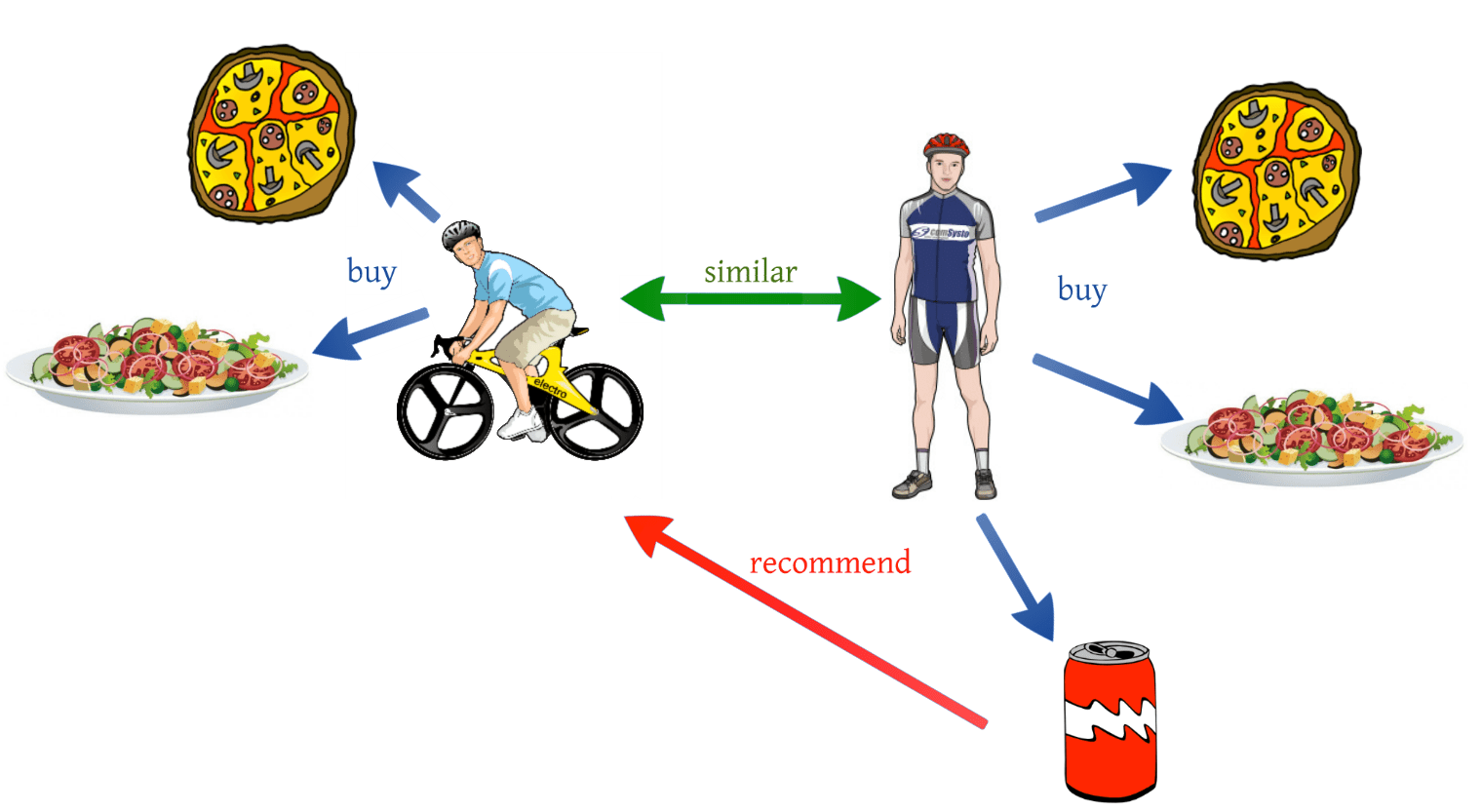
That’s the principle of collaborative filtering (see below).

1.1.2 Collaborative Filtering

This type of recommendation leverages product transactions, ratings,… to give recommendations. It is based on the collection of users' behaviorals, activities or preferences. It's about predicting what they are likely to do (depending on the business proposals and objectives) using the principles of 'similarity' between the,. This approach is looking for lookalike customers and predicting what a customer would be interested in based on its neighbours.

If we take the example of a simple market basket analysis…Let’s assume that a person likes Fajitas and Pizza and another one likes Pizza, then the last is likely to like Fajitas.

From a more technical approach, it consits in looking at the history of actions of customers. For a specific customer, we try to find customers with similar history of actions and recommend the top products that these similar customers have interacted with (note that proposals to our active user are only for unobserved items). Therefore, we’re taking into a ‘collaboration’ concept among users.



To be more precise, there are two underlying approaches for this filtering system:

- The user-user approach

In this approach, items are recommended to a user based on an evaluation of items by other users in the same neighborhood, with whom he/she shares common preferences. If the article was positively rated by the sub-community, it will be recommended to the user. In the user-based approach, articles which are already rated by a user, play an important role in searching for a group that shares appreciations with him/her. In other words, it looks at the similarity between people like the common preferences you can share with your best friends.

- The item-item approach

This one use the actions' history of other users (what they bought at the same time for example, or within a specific period of time) to propose you products that could have an interest regarding your purchase history or current basket.

For example, Amazon uses a lot this type of algorithm to add products that are often bought with the item you have just added to your basket: example a video game when you buy a play console.

The advantage here is that it is improving over time by adding information to the database but also because it can recommend complex items (such as movies) without having to 'understand' them (few features to enter/recognize). But, it requires time and huge computational power when it comes to deal with large amount of transactions (scalability issue). Like the content-based filtering, the upside becomes a downside and vice versa.

Collaborative filtering also relies on past preferences or rating correlation between users. However, this technique can lead to bad prediction if the article is unpopular and very few users have given feedback about them, called the cold-start issue.

So basically, if metadata is all a company has available, it can start with content-based approaches. If it has a large number of user interactions, it can experiment with more powerful collaborative filtering.

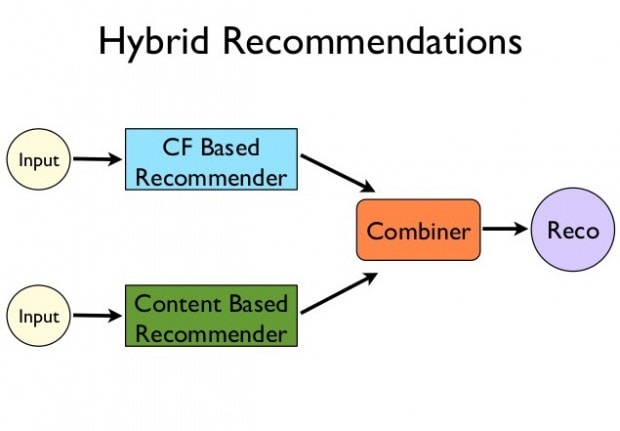
To summarise…

Item-Item: *“Users who liked this item also liked…”*

Item-Item: *“Users who are similar to you also liked…”*

And what about mixing both of them ? Maybe it would have been relevant regarding our previous issue. Indeed, it can be relevant for certain purposes to overcome (at least mitigate) the difficulties we have just seen - for each method taken independantly - by combining both ... This is the job of Hybrid recommendation systems.

1.1.3 Hybrid Filtering

There are many ways we can mix and match collaborative filtering models (transaction-based data) and content-based models(meta/tags-based data). But as we mentionned in the problem above it may have lots of benefits for entities because you can avoid some limitations and problems of pure recommendation systems we saw above. Indeed, the other algorithm in the hybrid system can overcome the issues of the first one considered, but also because it may be more relevant to take both methods in account for some problems. 

Hybrid means that we can construct an algorithm using both techniques but it can also only consists in combining results of both methods. There are many ways to use hybridization such as :

* Weighted hyrbidization: it consists in combining results using a linear combination. Weights are adjusted in the learning process through prediction accuracy.
* Switching hybridization: this one uses the method that provides the best prediction accuracy. In other words, it switches to one from another regarding the accuracy result regarding a particular issue we want to solve.
* Mixed hyridization: recommendations from different recommenders are presented together to give the recommendation (there are as much type of recommendations as there are recommenders)
* Cascade hybridization: it’s an iterative refinement process in which it uses one of both techniques as the main one and apply the other after to refine the prediction.
* Feature combination: the features of one recommendation technique is merge to another so that it doesn’t only take in account it’s own data at hand but every data you think could be relevant.
* Meta-Level: the outputs generated by one algorithm is used as input for the other algorithm
* …

A few words to conclude

To summarise, there are **methods for every specific issue**. But the advantage of using hybrid (and the different way to apply hyrbidization) offer larger possibilites to get more accurate results. Every problem has one or more solutions and that can even be adapted and reviewed in time. Like every machine learning project, there are as many ways to deal with the problem at stake as there are existing problems and the only good advice is too **experiment** as much as possible and adapt an **iterative approach**.

Finally, in order to embrace with such an iterative approach, **evaluation metrics** must be implemented…

*Note that we have talked about only three major techniques here, but there exists other methods like utility-based, knowledged-based, demographic-based,...*

1.2 How to evaluate the quality of recommandation systems?

****

**Quality** is a very subjective word and indeed there are different ways to measure this quality. However, there are two major ways of testing recommendation engines.

1.2.1 With offline methods

This first ones are **offline methods.** They are ideal for experimental purposes. The data is split into a training set and a test set (simple split or cross-validations). The final objective is generally to minimize prediction errors using measures such in order to evaluate the accuracy of the model that have been used for the modelisation. But when using such a method there may be discrepancies because there may be factors in the results that can't be adequatly represented or taken in account by the model (especially when implicit data do play an important role in the reality).

There are several way to assess the **prediction accuracy** in order to measure their performance. **Metrics depends on the problem and it’s good use to mix them** (when it’s possible).

Usual machine learning metrics can be used such as MAE (mean absolute error), (R)MSE ((root) mean squared error), Classification Accuracy, Logarithmic Loss, Confusion Matrix, AUC (Area under the curve), …

But, there are also specific techniques to recommendation engines. Let’s enumerate some of them below *[7]*:

* **Coverage**: this is the percent of items in the training data the model is able to recommend on a test
* **MAPK and MARK**: for Mean Average Precision @ K and Mean Average Recall @ K. MAPK gives insights into how relevant the list of recommended items are. MARK measures the recall at the k-th recommendations, so how well the recommender is able to recall all the items the user has rated positively in the test set. MAPK and MARK are ideal for evaluating an ordered list of recommendations therefore
* **Personalization rate**: it is the dissimilarity between user’s lists of recommendations (1 – cosine-similarity). A high personalization score indicates that user’s recommendations are different. In other words, that the model is offering a personalized experience to each user
* **Intra-list similarity**:it’s the average cosine similarity of all items in a list of recommendations. It uses the features of the recommended items (such as article thema) to calculate the similarity. It can be calculated for each user and averaged over all users in the test set to get an estimate of intra-list similarity for the model

1.2.2 With online methods

The second ones are online methods, called **A/B testing**. It is today the most prominent approach because it allows to rapidly adapt and redesign models. Basically, it's a live experiment that compares two versions of a thing to find out which ones work better. Version A for example is the banner (in the dev environment) you usually used on your website, version B is the new banner your designer has just finished to create. 50% will still see version A on your website and the other 50% will see appear the new banner. The idea is to measure which one is preferred by users using specific measures: it could be bounce rates, click through rates, conversion rates, activities per session,... in our banner case. Finally, the banner which gets the best scores (in average) should be taken into account from now on for every users.

There are advanced methods of A/B testing because, as you can see, there is importance regarding the split you do at the beginning. For instance, if you give coca to half of your population and pepsi to the other half (of course they don't know which one is coca or pepsi) and you measure whether there are willing to buy another one after, with a result of 20% in the first sample and 5% in the other, you could misinterpret those results. Indeed, if the first half of your population gathers 70% of big consumers of soda, the other half 30%, then it could lead to bad interpretation.

That's why there exist several variants of A/B testing. For example, it could be proposing to each user banner A for a certain page and banner B for another webpage of your website,... so that you can measure results for each user individually.

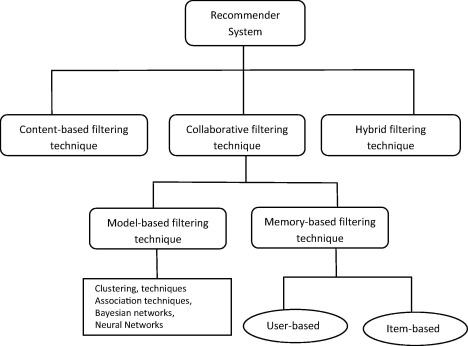
Finally, there are also possibilities to mix both of them which could be relevant in some cases and if an entity has enough resources to do it.

*Note that in the application, we will only do evaluation with offline methods as we don’t have the possibility to do A/B testing.*

2. Mathematical framework and models behind recommendation techniques

2.0 Introduction

Before moving to the real mathematical aspects behind content-based, collaborative and hybrid filtering, let’s detail a bit more about the global structure of the general framework of recommender systems.



As you can see, it’s a cascade architecture (especially for the collaborative filtering technique). Indeed, we didn’t mention it previously, but we will go through – in this short introduction – about a more detailed approach of collaborative filtering.

**Collaborative Filtering** is broadly divided into 2-types of techniques, the model-basedone and the memory-based one.

**Memory-based** first loads the entire or a sample of the user database into the system memory and make predictions for recommendation based on such in-line memory-based – which can become a problem with huge amount of data and takes a lot of time down the road. However, creation of model and explainability are easier than in model-based because it’s only about **arithmetical calculations**.

**Model-based** uses data mining, machine learning and/or even deep learning models to do recommendations. There are basically two main types of models: the ones based on the neighbors and the ones that perform matrix factorization - using latent factors – we will see later in this paper. These techniques try to predict which behavior would a user make based on his past preferences. Thus, it’s about calculating a grade for example for an item, or predicting its future behavior like buying or not an item, responding to a poll or not, click on your ad,… In contrast with memory-based and content-based, we don’t only look at available data but we try to predict its behavior (it’s about assumptions) and we suggest him items depending on these predictions (0/1, or grades).

These techniques may be faster to recommend contents to users and more flexible since we can change lots of parameters and adapt them. While content-based and memory-based depends on the data available. That’s why they are quite limited.

To come back to the memory-based approach, it can be divided into two main sections: **user-user** and **item-item** filtering that we already went through in the section 1.2.2.

So now that we have provided with more explanations about the global architecture let’s have a look at the mathematical framework behind each technique.

2.1 Content-based Filtering

As we previously said, content-based filtering relies on meta-data representing the features/characteristics of an item and what a specific user were interested in. So, it does take in account the behavior of one user but can only recommend items that share common features with the ones you already interacted with. If no historical data about the user, there is lots of possibilities to recommend items. It could be for example, feeding the user with the top items in his region (using his IP address). However, we won’t do it here as it’s a special case.

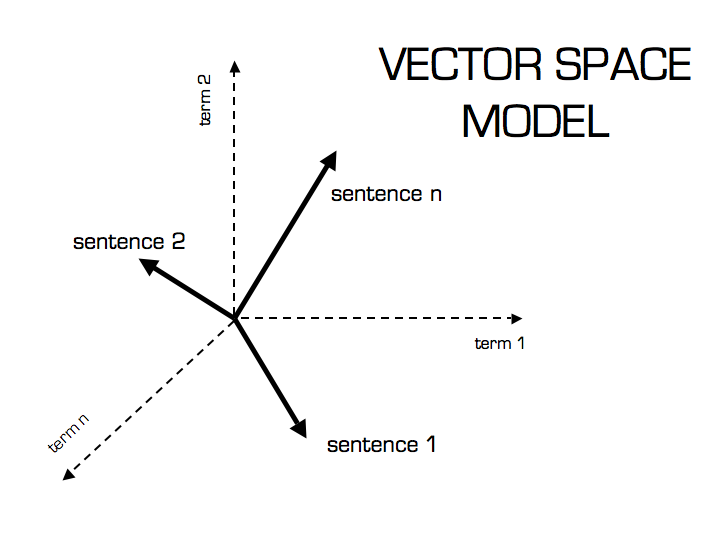
The general idea behind content-based is to measure the **similarity** between user’s preferences and an item defined by its features.

Before moving forward to this measure of proximity between vectors, we must introduce the concept of **TF-IDF** used in Text Feature Extraction. Indeed, thisconcept (Term Frequency, Inverse Document Frequency) is used to determine the importance of words relationships between documents. This method enables to give more importance to features that describe the core identity of an item and removes features that are somewhat quite general (appears in lots of documents). TF stands for how many times a word appears in a document. Basically, it’s only about counting the number of apparitions of a given word. While IDF tells us how rare it is for a document to have this term. The calculation is made by taking the inverse of how many documents have this word divided by the total number of documents. This calculation prevents from taking words (like: ‘the’) that appear in a lot of documents. The more it appears in documents, the lower is this IDF value. Multiplying TF-IDF gives then a weight per word, promoting core terms and demoting common terms – more generally, promoting very descriptive features of item and demoting general ones.

After having modified every features for each item, we have to look at the similarity between vectors considered. In our case, we won’t have two vectors of keywords to compare but a vector of weights for our user and another vector containing information about the item.

Note that the coefficients (per feature existing in the database) of the user may be improved when it completes more and more actions, that’s why some websites request him to give grades (or like/unlike items) as much as possible. It has critical importance because it influences the similarity measurement below.

Indeed, in order to measure the similarity between user’s preferences and an item, we use the **cosine similarity** (a vector space model calculation) which is a measure of distance/orientation between our user’s preferences and an item. The usual cosine similarity is merely a normalized dot-product of the two vectors at hand that will measure how closely the two vectors align.



The dot product between two vectors a (the vector of user’s preferences per feature) and b (values explaining how much this item a given feature) is:



And the cosine similarity is defined as below:



Therefore, we have:



There are many ways to sharpen this method depending again on the problem at hand, but we remain here on a general overview.

Conclusion: This vector-based model technique is probably the most intuitive one. This is only the arithmetic part of the recommender engine.

2.2 Collaborative Filtering

As we previously saw there are different approaches regarding collaborative filtering: memory-based and model-based. Memory-based only use arithmetic operations while model-based use learning parameters – data mining, machine learning and/or deep learning tools. Indeed, memory-based use as many data as possible (whole data or sample depending on the size of the dataset) and calculate distances to recommend the most relevant items. In the model-based method, it’s about using unsupervised or supervised learning models depending on the issue.

As you can see, Collaborative Filtering is much more flexible than content-based and offers various possibilities.

2.2.1 Memory-Based

As a reminder also, in this approach there are two manners to recommend items to a user. First, using the user-user method and second by using the item-item method. But both user have predefined vector-space of attributes – that’s why we call them memory-based (or knowledge-Based).

On one hand, the user-user method is about looking at closest users between them depending on their history. In this case, considering a specific/active user, we will only select the other users that are in its closest neighborhood to recommend what this neighborhood has also interacted with.

On the other hand, the item-item approach is about looking at what other users (generally all users) have interacted with after having also interacted with the item that the active user is considering/has considered.

**User-User approach**

In this approach, therefore, the issue is about finding similar users considering one active/specific user. As for the content-based approach, this is only about arithmetic. Indeed, we will look at the distance between two users (plays the role of the document for the example we take above) regarding items (plays the role of features, or words in our example). The values taken by items are numerical and represents the likings. We could use the **cosine similarity** or the **Pearson correlation** depending on the problem at hand.

Pearson correlation:



Where (x,y) are the users defined by their interactions with items (binary, categorical or linear depending on the issue) and are the averages of each vector. It can be useful when the values in each feature need to be standardized (ratings for example).

The goal is then to predict what our active user is able to like regarding items of its closest neighbors he hasn’t observed yet.

**Item-Item Approach**

In this case, we start from an item (let’s call it The Item) that the active user has interacted with. Then we must find which items other users have also interacted with having also The Item in their history. Therefore, we will take a subset of the user database. It’s an **intersection**. The difference with content-based is that we don’t look at the core features/attributes of items, but we look at the other users’ behaviors given that they interact with The Item. In other words, how people treat two items in terms of like/dislike, bought/not-bought.

Note that this method is much stable than the user-user one as the average item has (probably) been the object of lots of interactions by user, while in the user-user one, some user may have small history which could lead to biased predictions.

To calculate this similarity between two items, we have to look into the set of items the target user has interacted with and computes how similar they are to The Item considered and take the top ones (by arranging them beforehand). Similarity between two items is calculated by taking the ratings (let’s assume that if a user bought an item it corresponds to 1 and if not to 0 – it’s a sort of rating therefore) of the users for both items.

The calculation common-used is also the cosine similarity. In our case, vector a represents the grades given by other users for The Item and vector b represents the grades given by the same users for an unobserved item of our user considered.

Once we have the similarity between The Item and others, the prediction is computed by taking a weighted average of the target user’s ratings on these similar items. And of course, we will take the one that has the best cosine value, let’s call it The Best.

However, as our user doesn’t have only one item in its database (in general) we must take a **weighted average** of The Bests Items regarding his grades per viewed items to classify the most interesting one for him. Indeed, items in his database don’t have the same interest for him. He may have preferred one to another (ex: grade 4 for Item 1 and grade 1 for Item 2) and we must take it into account when proposing him new items. That’s why we do a weighted average between The Best ones.

2.2.2 Model-Based

In the model-based filtering, as we previously said it’s about learning from the data and create models for prediction. There are supervised and unsupervised methods depending on the problem (data) at hand. Sometimes we don’t know the real results. Therefore, we use unsupervised methods and when we know it (such as rating for example) we can use supervised learning methods. However, both can be used again, it’s a matter of prediction accuracy.

Let’s enumerate therefore some techniques common-used.

**Supervised Learning**

Classification

Sometimes, for recommender systems we know that users and items belong to a certain category. Therefore, we use classification algorithms such as Logistic Regression, Bayes Classifier (Linear Discrimination Analysis), KNN (K-Nearest Neighbors), MLP (Multi-Layer Perceptron),…

**Unsupervised learning**

Clustering

Clustering is based on the assumption that users in the same group have the same interests. Therefore, users are partitioned into groups (i.e clusters) which are defined as sets of similar users. Here, we want to discover the inherent groupings in the data. Usual methods are K-Means or Gaussian Mixture Models.

Association Rules

Association rules are particularly interesting for market basket analysis. It identifies sets of items that frequently occur together. One famous Approach is the Apriori algorithm.

Latent Variable Models

They are commonly used for data preprocessing, such as reducing the number of features (dimensionality reduction). Indeed, sometimes some features may be highly correlated together and these models allow to gain time and effort. Principal Components Analysis (PCA) is often used, Non-Negative Matrix Factorization (NMF) that automatically extracts sparse and meaningful features from a set of nonnegative vectors (which is the case of ratings generally – therefore we won’t standardize them before but we can normalize them however).

Moreover, Simon Funk developed, for the Netflix Data Prize, a method using the Singular Value Decomposition principle – **SVD** – which is a matrix factorization technique for user-item interactions. The idea behind is to factor the matrix containing our users’ ratings – by taking eigenvalues - in order to lower the dimensionality. The latent factors are designed such that they explain a large portion of the variance (but generally we don’t understand what they explicitly mean). For example, and to use a simple description case, it could be:



where…the above is a mapping of items based upon two related latent factors. The factor on the horizontal axis groups movies on the left as horror / lowbrow comedies while movies on the right contain movies such as comedies with serious tones and dramas. On the vertical axis, the factor is characterized as “independent, critically acclaimed, quirky films on the top, and on the bottom, mainstream formulaic films” *[9].*

*Note: between all these Latent Variables methods, the advantage of Non-Negative is that extracted features are more interpretable than for other methods.*

A few words to conclude

To summarise: *“Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a single technique. Consequently, our solution is an ensemble of many methods”* argued the winning Team, BellKor’s Pragmatic Chaos of the Netflix Prize.

The mix of content-based and collaborative-based filtering seems (intuitively) obvious. That’s why hybrid filtering gives better result than a single approach. However, we must also note that inside of both methods there are different ways to do and best algorithms combined the different techniques we saw above.

Note that there are other types of recommender systems like Multi-Criteria, Risk-Aware, Mobile,… we won’t detail here but that may be used independently or combined with others ones (hybrid).

3. Application to a Movie Dataset (MovieLens)

3.0 Introduction

**The context**

Netflix posted more than a decade ago an open competition on Kaggle: the Netflix Data Prize. In a few words, this prize sought to substantially improve the accuracy of predictions about how much someone is going to enjoy a movie based on its movie preferences.

This challenge paved the way to more **personalized** recommendation engines, or at least opened new ways of building them using a user-centric approach. Indeed, the difference is that companies like Netflix were able now to collect (through their streaming platform) data about their users’ tastes and collaborative filtering have become then crucial to feed people with what they could be interested in. It’s a bit different from items sold on a marketplace because the impact on users is higher when it comes to entertainment and having good time on a Sunday night alone or with your friends/family. According to Netflix, it’s even a way to ‘connect’ people so that they can share about their impressions at the coffee break in the Monday morning…

The winner team, BellKor’s Pragmatic Chaos – made up of statisticians, machine learning experts and computer engineers – enlightened new approaches and especially regarding the collaborative filtering part of their algorithm.

However, we will use here datasets from MovieLens [10] that also contains tags for movies (which we don’t have by taking the datasets from the Netflix Prize).

Now that we have given some information about the context let’s move toward the description of the datasets we use.

**Datasets Description**

Content-Based Filtering Algorithm

For this section, we will use the ml-latest-small.zip which contains 100,000 ratings, 3,600 tags applied to 9,000 movies by 600 users (last updated in September 2018). We could have taken a larger one, but it would have led to longer time processing.

Our first dataset – *tags* – is composed of tags representing the features of a movie. It contains the id of the movie, the id of a user, a tag (string format) associated by the user for this movie and the timestamp.

We also have a dataset called *ratings*, that contains the user ratings per movie. The variables are as followed: userid, movieid, rating (integer between 1 and 5) and the timestamp.

We will finally use a third dataset called *movies* that contains the movieid, the title (with year of publication) and a column containing the genre(s) (string format – a movie may contain several genres).

*movies*



*tags*



*ratings*



Collaborative Filtering Algorithms

We will focus on this paper on the model-based approach as memory-based is (quite) similar to content-based filetring in the way we build the algorithms (see more explanations *part 3.2)*. Therefore, in order to provide the most accurate predictions, we will use larger datasets than the ones used in the content-based approach. MovieLens provides with a dataset called ml-100k with 100,000 ratings, from 1000 users on 1700 movies.

*Note: the library surprise we will use (see part 3.2) already includes the dataset by calling the Dataset.load\_builtin method to ‘ml-100k’. It contains movieId, userId and ratings columns.*

3.1 Content-based movie recommender

Our dataset contains tags that will be used to recommend movies to users, as they represent the core features of our items (i.e words for document frequency). The purpose therefore is to build a recommender engine using the content-based method by using tags previously entered by a specific user for the movies he watched. The most similar movies (through the relevancy of tags of unobserved movies) will be recommended to the user.

Basically, the first step – excluding the loading data part – is to combine each tag for the movie. We did this in our code in the ‘Let’s have a look at Tags’ part, to finally get this.

Note that we added columns: mean & median of ratings and number of tags per movie.



Now that our data is grouped by movie, we can switch to the creation of TF-IDF vectors, containing therefore balanced values for tags per movie using the concept mentionned in the content-based part (*2.1*).

Scikit provides a method so that we don’t have to do it manually called *TfidVetcorizer()*. We have to call it using *fit\_transform* on the column containing the tags of each movie.

This done, we have to calculate the cosine similarity between each pair of films, resulting in a matrix composed of the cosine values between two movies.



In the Notebook for presentation purpose, we selected as example, the first movie (movieId=1) that corresponds to ‘Toy Story’. Thanks to the cosine similarity, we can then see which movies are most similar to ‘Toy Story’ from an arithmetic point of view (and given the data we have – that may be improved by adding data). We get:



However, the final purpose of our engine is to recommend a movie using these cosine values for **a certain user**. In the notebook, we decided to take user 1. This user rated 232 movies and among these 232 movies, 114 are present in our dataset of tags.

We must take into account his preferences in order to give him accurate personalized recommendations. Therefore, we need his ratings that are the numerical format of its preferences.

Instead of using weights from scratch (i.e ratings) we standardize them to promote films that he really enjoyed and demoting films he didn’t enjoy.

For each movie he watched, we will calculate the cosine similarity with unobersed movies (in the notebook, we do it for every films but we remove films that are already present in the user database).

Finally, we sort them by cosine values to get the most relevant ones, corresponding therefore to our final recommendations:



3.2 Collaborative movie recommender

Memory-Based

In this paper, we won’t go through the code of memory-based algorithms (user-user & item-item) as the build-up is quite similar to the content-based algorithm.

However, let’s give some information about the main steps we would have to go through.

*User-User*: we would have contstruct a matrix containing pearson correlations between each user. Why Pearson ? Because, a user may give in average a grade of 2,5 out of 5 and another one may give in average a grade of 4 out of 5. Therefore, we have to take this variation of behaviour between user before comparing similar ones. We must note also that there may be a problem with this approach when the number of common ratings is small. Indeed, it might be possible that two users have only two ratings in common but the value of correlation is very high (close to one). To remove this, we may have weighted the cosine milarity beforehand.

*Item-Item:* we would have build a method that takes a user and construct a dataframe containing every movie he saw and take for each movie the best cosine similarity (by calculating it before thanks to others users’ ratings) with unobersved movies. Then, we would have weighted each row (therefore containing a movie and its closest neighbour) by the grade given by the user for this movie (which is a sign of preference) so tht we can finally suggest him the most potential best movies for him.

Model-Based

For this part, we will use the **Surprise library** (that *stands for : Simple Python Recommendation System Engine*). This library is an easy-to-use Python scikit for recommendation systems *[11].*

As we mentionned above (*part 2.2.2*) there are data mining/machine learning/deep learning algorithms we can use to do predictions and recommendations down the road using collaborative filtering.

In the notebook, we will code different algorithms and comprare their predictions accuracy between them using a train/test split of the data. Note, that we will use cross- validation to prevent from overfitting or selection bias.

Then we will chose the best model and use it/them to do recommendations.

Here is the list of the alogirhtms we are going to implement:

* SVD, performed by the winning team of the Netflix Prize
* KNN-Basic, simple KNN algorithm
* KNNwithMeans, a KNN algorithm that takes into account the mean ratings of each user
* SVD++, an extension of SVD that takes into account implicit ratings
* Co-Clustering\*
* NMF, an algorithm based on Non-Negative Matrix Factorization

\*We didn’t go through the principle of co-clustering yet, but we said that sometimes it would be interesting to use an algorithm that both takes into account our users and and items. In contrast with classical clustering that groups our users or items into ‘similar’ categories (principle of object-similarity but as you can understand it depends on the data we have), co-clustering is bimodal. In other words, it’s a joint interaction between two type of entities. For us: users’ ratings and movies. It’s a simulatenous clustering of rows and and columns of a matrix (instead of doing it on rows only - or coulumns). It enables to look for similarity in the pairwise interactions between objects we have, which is therefore interesting in our case.

*Note that in general, we didn’t look at which parameter leads to the best prediction accuracy, because we don’t need it here (we could have used the GridSearchCV).*

For each method, we used the MSE to measure the level of accuracy and our best model is the KNN-Basic (in our case: using a train-test split of 0.75/0.25 and a 5 cross-validation split).

*With KNN-Basic:*



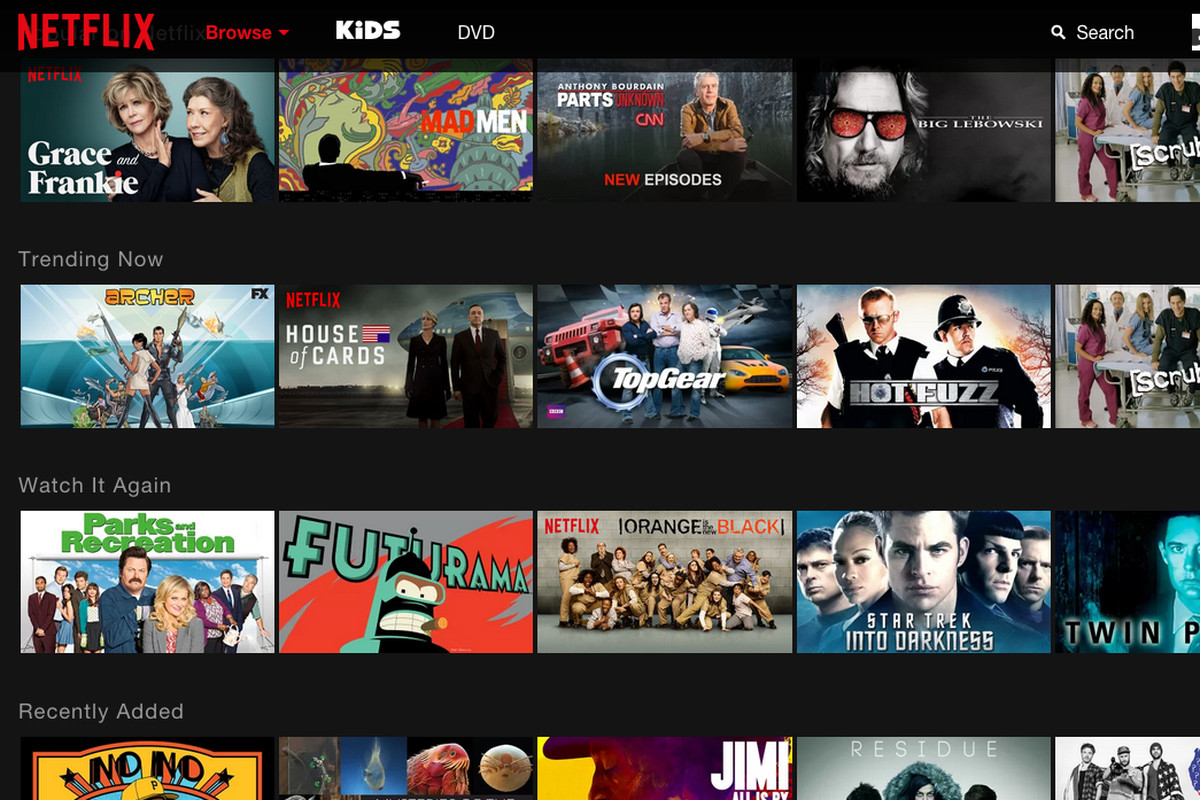
We predict the rating of user 1, with movie 10 and we get an estimation of 4,08 for the grade he would give to this movie.



**A few words to conclude:**

Here we went through the content-based and model-based techniques. As we can see, there are muplitples ways to do (especially with the model-based technique as you can chose different models and parameters,…).

As we have said earlier, the memory-based method building is quite similar to the content-based.   
Mixing algorithms using the hybrid concept would have allow to give probably very accurate recommendations. However, it’s also true that hybrid have more sense according to me, when you can suggest to the user different ‘type’ of recommendations. For example, Netflix is using different categories (‘Best success’, ‘Current Trends’, ‘Because you watched,…’, ‘Similar to’…) of recommendations which may raise the user engagement. This way of doing is also great to **test and compare algorithms online** and see which ones may lead to better conversion rates or ratings.



4. Conclusion

Recommender engines are very powerful tools to make the use of the internet more personalized. As we have seen, it has lots of benefits when it comes to enhance user engagement.

There are severals ways of building them but also improve them depending on specific purposes. Basically the most common approaches are content-based filtering, collaborative filtering and a mix of both called hybrid filtering. In this regard, we must emphasize the fact that combining algorithms gives more accurate results because one method is not self-sufficient.

Furthermore, it is through experiment and ongoing evaluations & feedbacks that we can highly improve them. It is an iterative process that can also be intensely improved using AI when one has lots of data at hand.

However, we didn’t go trough the fact that they may arouse some problems. When badly implemented, they may give irrelevant recommendations and decrease users’ satisfcation down the road. More generally and from an ethic point of view, they may promote personnal data collection which is a sensitive topic. Thus, there require specific code of conducts for companies to keep a balance beween relevant content for users and data privacy/protection.

5. References & Appendix

References

1. <https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers>
2. <https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429>
3. <https://www.journals.elsevier.com/egyptian-informatics-journal>
4. <https://www.ibmbigdatahub.com/blog/why-we-need-methodology-data-science>
5. <https://www.multisensorproject.eu/wp-content/uploads/2015/12/D5.3_DecisionSupportTechniques.pdf>
6. <https://towardsdatascience.com/evaluation-metrics-for-recommender-systems-df56c6611093>
7. <https://github.com/statisticianinstilettos/recmetrics>
8. <https://www.researchgate.net/publication/321753015_Model-based_approach_for_Collaborative_Filtering>
9. <http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-gower-netflix-SVD.pdf>
10. <https://grouplens.org/datasets/movielens/>
11. <https://surprise.readthedocs.io/en/stable/>

Code

The code below has a text format. Please go to my github to look at the notebooks for better presentation using this link:

<https://github.com/adrienlequiller/Recommender_Engines>

You can also download for the content-based part the data directly in my github or by using the MovieLens Website (we used ml-latest-small for the content-based):

<https://grouplens.org/datasets/movielens/>

Content-Based

import pandas as pd

import numpy as np

import math

import io

from sklearn import preprocessing

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

# %pylab inline

"""## \*\*Data Loading\*\*"""

from google.colab import files

uploaded = files.upload()

tags = pd.read\_csv(io.BytesIO(uploaded['tags.csv']),encoding="ISO-8859-1")

tags.head()

# links are : Identifiers that can be used to link to other sources of movie data are contained in the file `links.csv`. Each line of this file after the header row represents one movie, and has the following format:

links = pd.read\_csv(io.BytesIO(uploaded['links.csv']),encoding="ISO-8859-1")

links.head()

movies = pd.read\_csv(io.BytesIO(uploaded['movies.csv']),encoding="ISO-8859-1")

movies.head()

ratings = pd.read\_csv(io.BytesIO(uploaded['ratings.csv']),encoding="ISO-8859-1")

ratings.head()

"""## \*\*Let's have a look at Tags\*\*

Let's first select a movie among our db of movies...

"""

movies[movies.movieId == 1]

"""... and look at tags associated with this movie"""

tags[tags['movieId'] == 1]

"""As you can see in the tags dataset, we have a tag per user and movie. We will create a function that concatenates every tag from a movie in a single line. See below."""

def \_concatenate\_tags\_of\_movie(tags):

tags\_as\_str = ' '.join(set(tags))

return tags\_as\_str

tags\_per\_movie = tags.groupby('movieId')['tag'].agg(\_concatenate\_tags\_of\_movie)

tags\_per\_movie.name = 'movie\_tags'

tags\_per\_movie = tags\_per\_movie.reset\_index()

tags\_per\_movie.head()

"""Let's now select the one for our first movie : Toy Story"""

tags\_per\_movie[tags\_per\_movie['movieId'] == 1]

avg\_ratings = ratings.groupby('movieId')['rating'].agg(['mean', 'median', 'size'])

avg\_ratings.columns = ['rating\_mean', 'rating\_median', 'num\_tags']

avg\_ratings = avg\_ratings.reset\_index()

avg\_ratings.head()

"""Now let's concatenate all of our information in a single dataframe. We will remove movies without tags (if any to avoid NAs)."""

movies\_with\_ratings = pd.merge(movies, avg\_ratings, how='left', on='movieId')

my\_data = pd.merge(movies\_with\_ratings, tags\_per\_movie, how='left', on='movieId')

my\_data = my\_data[~my\_data.movie\_tags.isnull()].reset\_index(drop=True)

my\_data.head()

"""## \*\*Algo Building\*\*

As we have explained in the paper, content-based filtering algorithm is similar to feature document analysis. We use the concept of TF-IDF to rmove tags that may appear in lots of movies (which therefore don't describe it with a lots of acuracy) to promote tags that may define each movie more relevantly.

Having doing so, we switch to the cosine similarity calculation (as explained also in the paper) using again scikit learn that contains a cosine\_similarity method.

### TF-IDF

Note that scikit-learn provides with a method to do this called TfidfVectorizer().

"""

tf\_idf = TfidfVectorizer()

movies\_tf\_idf = tf\_idf.fit\_transform(my\_data.movie\_tags)

movies\_tf\_idf.shape

"""### \*\*Cosine Similarity\*\*"""

cosine = cosine\_similarity(movies\_tf\_idf)

movies\_cosine = pd.DataFrame(cosine\_similarity(movies\_tf\_idf))

indices = my\_data.movieId

movies\_cosine.columns = [str(indices[int(col)]) for col in movies\_cosine.columns]

movies\_cosine.index = [indices[idx] for idx in movies\_cosine.index]

movies\_cosine.head()

"""## \*\*Most Similar Movies to Toy Story\*\*"""

movies\_cosine.iloc[0].sort\_values(ascending=False)[:10]

top\_3 = [2355,122918,3114]

my\_top\_3 = my\_data[(my\_data.movieId).isin(top\_3)]

my\_top\_3[['title','genres','movie\_tags']]

"""## \*\*Recommendation for a User\*\*

We have to first select a user from our db, let's take the user 1.

"""

user\_ratings = ratings[ratings.userId == 1]

print(user\_ratings.shape)

user\_data = my\_data.reset\_index().merge(user\_ratings, on='movieId')

print(user\_data.shape)

user\_data.head()

#it's very good because he rated lots of movies (232 actually but only 114 of them are in the tag db) which will lead to better prediction accuracy

"""Let's compute standardized weights we have explained into the paper."""

user\_data['weight'] = preprocessing.scale(user\_data['rating'])

user\_data.head()

user\_profile = np.dot(movies\_tf\_idf[user\_data['index'].values].toarray().T, user\_data['weight'].values)

C = cosine\_similarity(atleast\_2d(user\_profile), movies\_tf\_idf)

R = argsort(C)[:, ::-1]

recommendations = [i for i in R[0] if i not in user\_data['index'].values]

my\_data['title'][recommendations].head(10)

Collaborative, Model-Based

!pip install scikit-surprise

import pandas as pd

import numpy as np

"""## \*\*Data Loading\*\*"""

from surprise import Dataset

# Load the movielens-100k dataset (download it if needed),

movie\_data = Dataset.load\_builtin('ml-100k')

"""We then split it with a training and a test set."""

from surprise.model\_selection import train\_test\_split

trainset, testset = train\_test\_split(movie\_data, test\_size=.25)

"""## \*\*Algo Building\*\* : With Surprise

### With SVD

"""

from surprise import SVD

from surprise import Dataset

from surprise import accuracy

from surprise.model\_selection import KFold

# define a cross-validation iterator

kf = KFold(n\_splits=5)

algo = SVD()

for trainset, testset in kf.split(data):

# train and test algorithm.

algo.fit(trainset)

predictions = algo.test(testset)

# Compute and print Root Mean Squared Error

accuracy.rmse(predictions, verbose=True)

"""### With KNN-Basic"""

from surprise import KNNBasic

kf = KFold(n\_splits=5)

algo = KNNBasic()

for trainset, testset in kf.split(data):

algo.fit(trainset)

predictions = algo.test(testset)

accuracy.rmse(predictions, verbose=True)

"""### With KNN-with-means"""

from surprise import KNNWithMeans

kf = KFold(n\_splits=5)

algo = KNNWithMeans()

for trainset, testset in kf.split(data):

algo.fit(trainset)

predictions = algo.test(testset)

accuracy.rmse(predictions, verbose=True)

"""### With SVD++"""

from surprise import SVDpp

kf = KFold(n\_splits=5)

algo = SVDpp()

for trainset, testset in kf.split(data):

algo.fit(trainset)

predictions = algo.test(testset)

accuracy.rmse(predictions, verbose=True)

"""### With Co-Clustering"""

from surprise import CoClustering

kf = KFold(n\_splits=5)

algo = CoClustering()

for trainset, testset in kf.split(data):

algo.fit(trainset)

predictions = algo.test(testset)

accuracy.rmse(predictions, verbose=True)

"""### With NMF"""

from surprise import NMF

kf = KFold(n\_splits=5)

algo = NMF()

for trainset, testset in kf.split(data):

algo.fit(trainset)

predictions = algo.test(testset)

accuracy.rmse(predictions, verbose=True)

"""### Predictions with KNNBasic

KNNBasic gave the best results here.

Let's do a prediction for a particular couple of user, movie. Let's take user 10 and movie 100.

"""

user\_id = str(1)

movie\_id = str(10)

prediction = algo.predict(user\_id,movie\_id)

prediction