HAAGA-HELIA
University of Applied Sciences

Personalized web based application for movie recommendations

Killian Duay

Bachelor's Thesis
Degree program in Business Information Technology
2019

HAAGA-HELIA
University of Applied Sciences

Abstract

24th April 2019

Degree programme

Author(s)
Killian Duay

Degree programme
Bachelor of Science, Business Information Technology

Report/thesis title
Personalized web based application for movie recommendations

Number of pages and appendix pages
75 + 0

Since a few years, the Machine Learning becomes more and more important. It is used everywhere, especially in recommender systems and personnalized marketing. Big companies of movies streaming build those systems in order to recommend some content to their customers and increase their profit.

The main issues with those companies is that they are focusing on their own content. They obviously only recommend the content they have and they are not taking into consideration the whole offer of existing movies. The user can not get global and objective recommendations because he is in the middle of several recommender systems that are not complete and not connected together since they are in competition.

The aim of this thesis is first to study the Machine Learning and the ways it can be used in the case of a movie recommender system. Then the aim is to implement one of those solutions in a project of a web based application for movie recommendations. The goal of the project is to develop a fully functionnal web application that allows users to get recommendations on all existing movies in the world. The application will be tested by real users.

The application will have a ReactJS frontend (web based and responsive design), a Python backend and a Firebase Database and Authentication. The movies and their information will be fetched on the TMDB API that provides data on all existing movies.

The deliverables are the fully functionnal project and this document. This document presents the background study, the design and implementation of the project, the results and the discussion about them and the possible further developments of the project.

Keywords
Movie recommendations, Machine Learning, Web application, Recommender systems

Table of contents

Terms and abbreviations

AI	Artificial Intelligence
ML	Machine Learning
KNN	K-Nearest Neighbors
NLP	Natural Language Processing
TF-IDF	Term Frequency-Inverse Document Frequency
UB-CF	User-Based Collaborative Filtering
IB-CF	Item-Based Collaborative Filtering
UI	User Interface
API	Application Programming Interface
SDK	Software Development Kit
JSON	JavaScript Object Notation
DRF	Django Rest Framework
ID	Identifier Number
CSV	Comma-Separated Values
URL	Uniform Resource Locator

1 Introduction

Since a few years, the world is mad about Machine Learning. The computers became powerful enough to run Machine Learning algorithms everywhere and the scope of the possibilities is broad and limitless. The Machine Learning can be used everywhere to improve everything. Healthcare, financial trading, marketing, search engines, banking ; everybody can take advantage of it (Marr, 2016).

The big companies that offer some content or sell items seized the Machine Learning to build systems that make recommendations to their customers in order to increase their sales or to increase the consumption of content. The recommender engines are everywhere : YouTube, Netflix, Amazon, …

The thesis author thinks that the movie recommender systems are very interesting and attractive since there is a real benefit for the customers and this is not just about selling more items. The customers can discover new movies and find out new horizons. It can be viewed as much as a creative tool as a business tool. The main issue of movie recommender systems offered by big streaming companies like Netflix or Amazon is that they are centred on their own content. They operate in a vacuum since they aim to promote their own content and not their competitors content. The creative aspect and the discovering aspect is therefore limited for the users and customers.

The users are subscribed to several streaming operators that offer a lot of various and different contents but they all have their own recommender system and there is no bridge between all of them for the users. It would be good if the users could get some recommendations regarding the whole movies offer and without any commercial aspect.

The aim of this thesis is to study how the Machine Learning works and how it can be used and implemented in a movie recommender system. After that study part, the aim is to implement one of the found solution in a project that supports the whole architecture of a web based application of movie recommendations that offer all the existing movies in the world.

2 Research question and methodology

2.1 Objectives of the project and research question

The objective of this project is to build a personalized web application for movie recommendations. The web application should allow users to search for all existing movies in a third party database and to add the ones they liked to their profile. The application would then recommend them movies to watch based on that profile. It should also be a user-friendly and good-looking interface in order to be easily used by the users.

The second objective deriving from the first one is the research question : to find out how the Machine Learning can be used to build a good movie recommender system.

2.2 Scope of the project

The Machine Learning is a wide field that can be used in many various situations. The scope of this project is to study and introduce the basics of Machine Learning and the different ways it can be used for movie recommendations. One of the best ways (according to the performances, the accuracy of the results and the constraint of the time) would be selected to be implemented and tested in the project.

Regarding to the building of the web application, the scope of this project is to build the whole architecture that allows users to log in to the application, search for movies, add them to their profile et get recommendations. The whole architecture should contain a backend, a frontend, a database and several Application Programming Interfaces (APIs). Each part of this architecture could be implemented in different ways and the selected technologies should be presented.

2.3 Methodology

Since we do not have any knowledge about Machine Learning at the beginning of this thesis, the methodology is to perform firstly a general research about the Machine Learning and its goals, workings, applications and methods . Then it is to perform a specific state of the art about Machine Learning in the field of movie recommender systems.

The research and the state of the art should be done through multiple and various sources and the final aim is to cross them all in this document.

3 Background study

3.1 Machine Learning overview

Before going deeper in the working and the workflow of Machine Learning (ML), let's just vulgarize its concept. The goal of ML is to create a system that answers on its own to questions. This system is called a *model* and this model is built by a process called *training*. This is the learning phase. This phase is done by feeding the model with data. The more data we have, the better the model will answer to questions. The model can be continuously improved by feeding it with more data. (G, 2017; Rouse & Burns, 2018)

Machine Learning is a part of Artificial Intelligence (AI) (Tagliaferri, 2017). The main goal of ML is to build algorithms that can get input data and use those data to predict an output while updating outputs as new data arrive (Rouse & Burns, 2018). Machine Learning learns from data to build models that give accurate predictions, or to recognize patterns (Castrounis, 2019).

Machine Learning differs from traditional computing as it does not require to be explicitly programmed. In traditional computing we solve the problems by explicitly telling the algorithms what the problem is and by coding how to solve it. In Machine Learning we train models by feeding them with big quantity of data and then ask them to give specific outputs for given specific inputs, based on that learning. (Tagliaferri, 2017)

ML algorithms are like children learning from themselves (Kurama, 2017).

3.2 Machine Learning methods

Machine Learning algorithms are categorized by two main methods : supervised learning and unsupervised learning. Those two categories differ on how the learning is received and how the output is given to the system. (Tagliaferri, 2017)

3.2.1 Supervised learning

In supervised learning, the machine is trained with inputs of data that are labelled with the desired outputs. The machine will learn what is expected as an output for a given input. It will therefore know what output is expected for a new input by comparing it with all the data used for the training. (Tagliaferri, 2017)

There are two main types of supervised learning : classification and regression. A classification problem is when the output variable is a category, such as *red* or *blue*, *cat* or *dog*. A regression problem is when the output variable is a real or continuous value, such as *salary*, *weight*. (Shukla, 2019)

Example of classification problem

For example, we want to know for a given image if it represents a shark or an ocean. We need to feed a model with a big quantity of images representing sharks or oceans. Each image is labelled as a *shark* or as an *ocean*. The system will then learn how a shark looks like and how an ocean looks like. At the end we can give the machine a new image without any label and ask it if it is a shark or an ocean. (Tagliaferri, 2017)

Example of regression problem

For example, we want to know the price of a house. We need to feed a model with a big quantity of data about houses. Each house item within the dataset has two labels that are explicitly mentioned : *price* and *size*. The system will then learn how much a house costs according to its size. At the end we can give the machine a new house with its size but without its price and ask how much does it cost. (Shukla, 2019)

3.2.2 Unsupervised learning

In unsupervised learning, the machine is trained with data that do not have any label. The system is left to itself to find similitudes inside the dataset. The goal is that the machine finds hidden patterns within the dataset. (Tagliaferri, 2017)

There are two main types of unsupervised learning : clustering and association. A clustering algorithm tries to split the dataset into groups according to similarity, such as *different species of plants*. An association algorithm tries to split the dataset into groups of items that frequently occur together, such as “people who buy X tend to buy Y”. (Kurama, 2017; DataRobot, 2019; Priy, 2019)

Example of association algorithm

For example, we want to know what items can a customer be interested for. We need to feed a model with a big quantity of data about items purchased by customers. We tell then the system to find hidden patterns within the data or a way to represent and classify the customers based on their purchases. At the end we can ask the machine what items

could interest a specific customer and the machine will be able to answer based on the customer's list of purchased items and on comparisons with others customers. (Tagliaferri, 2017)

Example of clustering algorithm

For example, we want to classify different flowers according to their species. We need to feed a model with a big quantity of data about flowers and their features (sepal length, sepal width, petal length, petal width ,…). We tell then the system to cluster the data into groups by finding a pattern within the dataset. At the end we can ask the machine what are the different groups and which flower belongs to which group. (Kurama, 2018)

3.3 Concrete algorithms and models of Machine Learning

In this section, we are going through three of the most used ML algorithms and models that belong to the methods and types introduced in the previous section. The purpose of this section is to have a general understanding of the techniques used. We are not going to see all the models and algorithms existing since this is not the aim of this thesis.

3.3.1 K-Nearest Neighbors model

The K-Nearest Neighbors (KNN) model is one of the most used algorithm in Machine Learning. It is useful for both classification and regression problems when we have labelled data. (Navlani, 2018)

The goal of the KNN model is to classify new data into classes according to their features (Navlani, 2018; Bronshtein, 2017).

How does it work ?

KNN model receives a dataset (containing data, labelled by the class to which they belong, and their features) and automatically represents the data in a X-Axis Y-Axis space according to their features (X is a feature, Y is another feature). (Kurama, 2018)

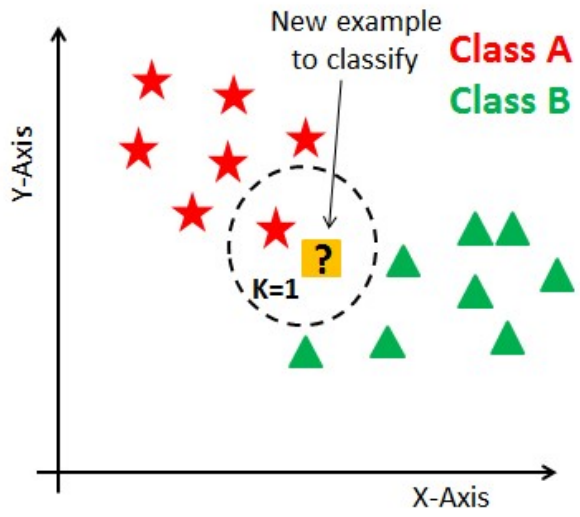

Figure 1 : KNN model representation (Navlani, 2018)

The figure above shows the data (labelled as *class A* or *Class B*) represented in the space by the feature X on the X-Axis and by the feature Y on the Y-Axis. Suppose now we have to classify a new point (the yellow *?* in the figure above), the model will calculate the distance (similarity) of the K-nearest points (where K is a parameter to choose) and labelled the new point as a member of the class having the biggest number of nearest neighbors. In the figure above, the new point will be classified as a member of the *Class A* because we set the K parameter to 1 (it means that we want to classify the point according to the first nearest neighbor). (Navlani, 2018)

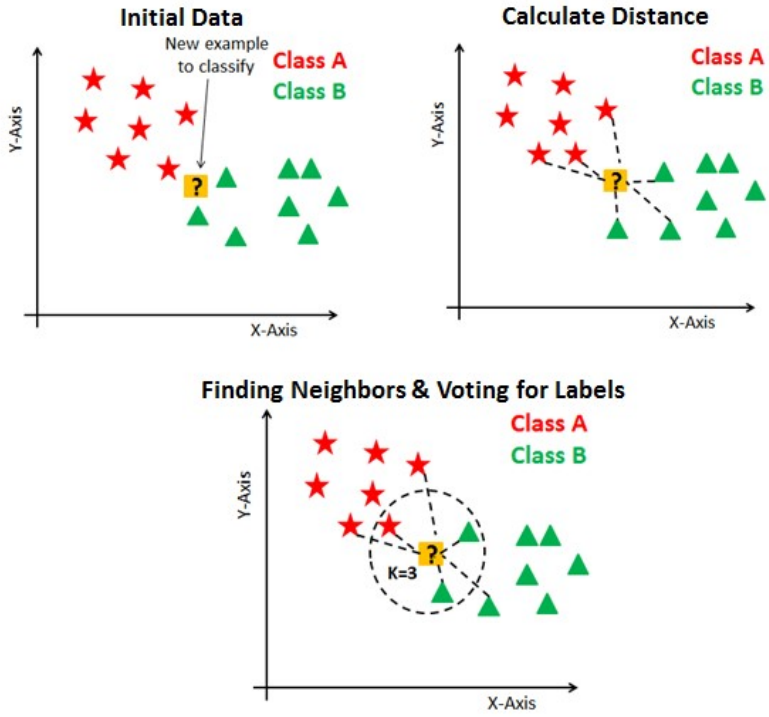

Figure 2 : algorithm for K=3 (Navlani, 2018)

In the figure above, we set the parameter K to 3. The yellow point will then ben classified as a member of the *Class B*.

There are different ways to calculate the distances between points, such as Euclidean distance, Hamming distance, Manhattan distance, cosine similarity and Minkowski distance (Navlani, 2018).

The choice of the K parameter is important and it is made at the time of model building. There is not any optimal number of neighbors (K). It depends on each dataset and we have to experiment different K parameters to choose the good one according to the accuracy of the results. (Navlani, 2018)

3.3.2 Linear regression model

Linar regression models are useful in regression problems when we have labelled data (Mishra, 2018).

The goal of linear regression models is to predict an output variable (Y) for a new input variable (X). It assumes a linear relationship between X and Y. (Mishra, 2018)

How does it work ?

Linear regression model receives labelled data (X and Y variables) and will represent them in a X-Axis Y-Axis space (Mishra, 2018).

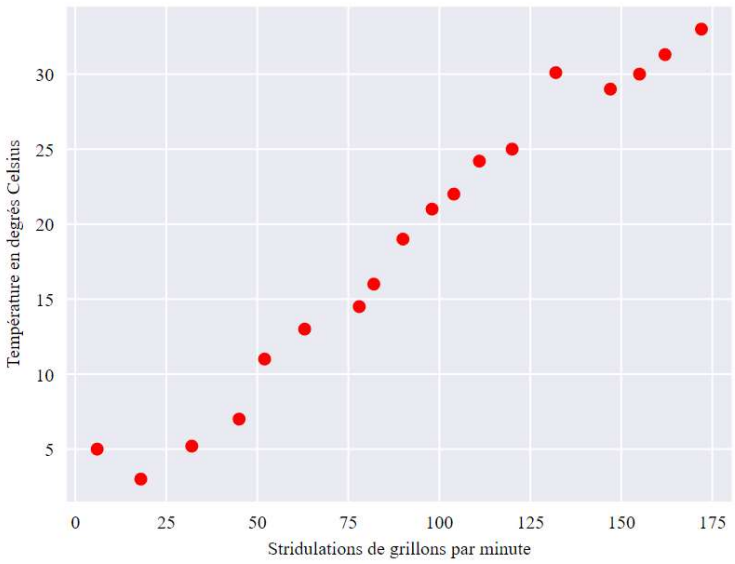

Figure 3 : linear regression model representation (Google, 2019)

The figure above shows the data represented by the noise made by crickets by minute on the X-Axis and the temperature Celsius on the Y-Axis. It will then try to approximate the relationship between X and Y by calculating a linear line between X and Y. The linear line is a function y = aX + b. (Google, 2019)

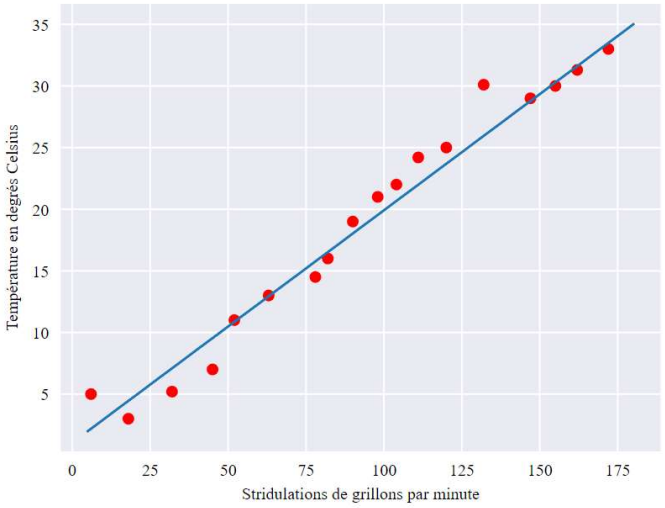

Figure 4 : linear line representing the relationship between X and Y (Google, 2019)

The figure above shows the linear line (function) calculated by the model. That function can then be used to predict the output Y (temperature Celsius) according to a new input X (noise made by the crickets by minute) (Google, 2019).

3.3.3 K-Means model

The K-Means models are useful in clustering problems when we have data without any label (Trevino, 2016).

The goal of K-Means models is to find groups within the dataset, to know to which group the data of the dataset belong and to know to which group new data belong (based on their features) (Trevino, 2016).

How does it work ?

The K-Means model receives a dataset (containing unlabelled data and their features) and a parameter *K* that represents the number of clusters that we want the model to find (we arbitrarily choose this parameter). It will automatically represent those data in a X-Axis Y-Axis space based on their features (X is a feature, Y is another). (Trevino, 2016)

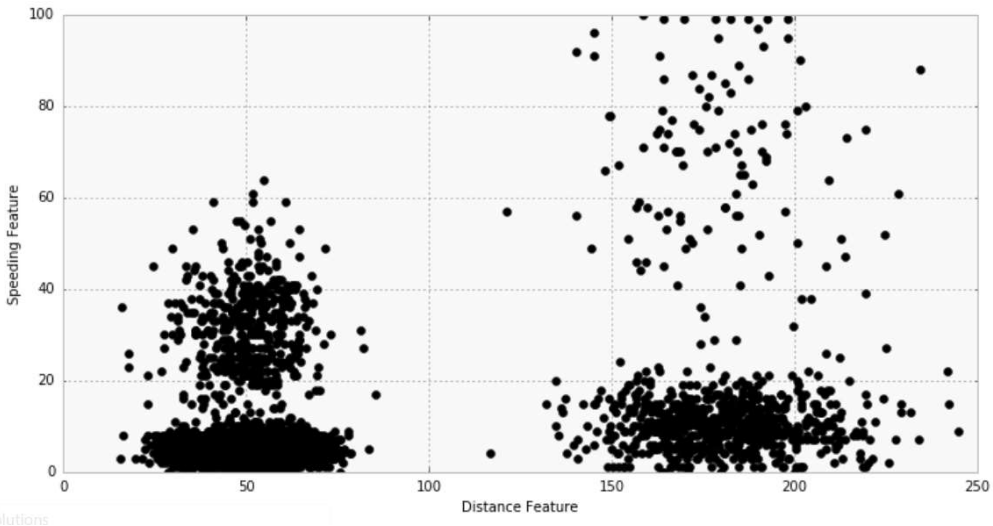

Figure 5 : K-Means model representation (Trevino, 2016)

The figure above shows the data represented in the space by the feature *distance* on the X-Axis and by the feature *Speeding* on the Y-Axis. The model will then try to find K groups (clusters) in those data based on the agglomerations of points (distance between points). The number of clusters K is the parameter we chose at the time of model building (Trevino, 2016)

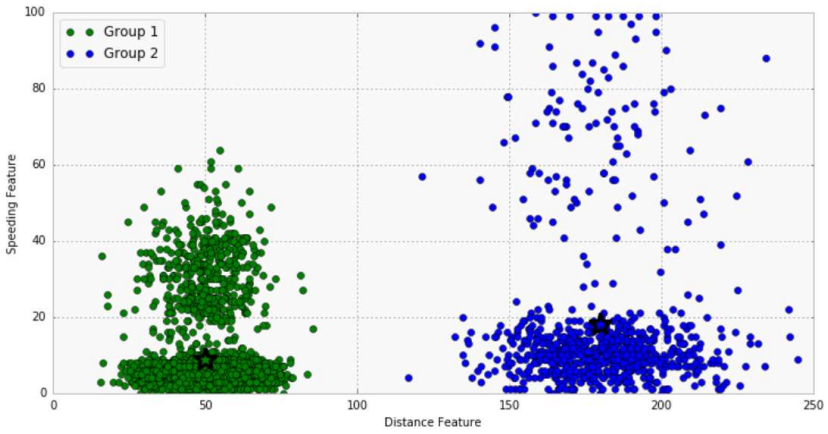

Figure 6 : clustering representation (Trevino, 2016)

The figure above shows the results of the clustering. The model found the two groups (clusters) and labelled the data as members of one of those two groups. It is now trained. (Trevino, 2016)

We can then use this trained model to know to which clusters new data that we gives to the model belong (Trevino, 2016).

There is no method to determine the good value of parameter K and we have to run the model with different values to find the good one based on the accuracy of the results (Trevino, 2016).

3.4 Programming languages

There are various programming languages that allow us to implement ML models. The main thing to consider when choosing a programming language to process Machine Learning is the number of available libraries and their purposes. (Tagliaferri, 2017; Puguet, 2016)

The most in-demand and the most used language in data science and Machine Learning is Python (Puguet, 2016).

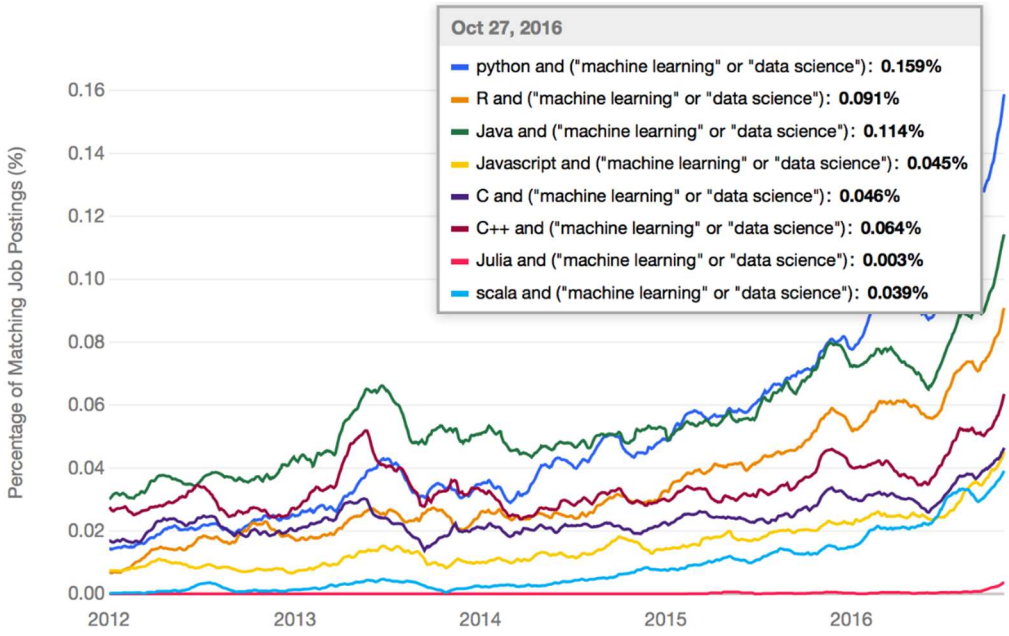

Figure 7 : statistics over time of job offers on indeed.com in ML and data science (Puguet, 2016)

The figure above shows the occurences over time of selected terms in job offers on indeed.com. Python is followed by R and Java. (Puguet, 2016)

Python has many useful libraries to process ML such as NumPy, SciPy, Matplotlib and mainly scikit-learn built on the top of them. It has good performances for preprocessing data and working with them directly. Recently the development of Deep Learning libraries such as PyTorch, Keras or TensorFlow puts Python in the best spot in the field of data science. (Tagliaferri, 2017)

3.5 Applications of Machine Learning

ML is a buzzword in the technology world right now since it represents a major step forward in the way a computer can learn. The possibilities are large and ML can be used in many different situations. (Marr, 2016)

In this section, we are going through the most known ML concrete applications.

In financial trading, Machine Learning algorithms are used by financial trading companies to optimize their business. ML can be useful to predict changes on the market at any time of the year by analyzing big amount of data and execute trades at high speeds and high volume. It can replace human analysts with more efficiency. (Marr, 2016)

In the healthcare, Machine Learning algorithms are used to process information and spot patterns to diagnose and monitor hospital patients. It can also be used to understand risks factors for disease in large populations. (Marr, 2016)

In the marketing, Machine Learning algorithms are used to better target customers and generate customized emails, coupons or offers. ML models are fed by the metadata defining the customers. (Marr, 2016)

In the banking systems, Machine Learning algorithms are used to detect fraudulent transactions. Paypal has developed a system able to distinguish between legitimate and fraudulent transactions between customers. (Marr, 2016; Morgan, 2015)

In the shopping systems, Machine Learning algorithms are used to recommend products to customers. Netflix or Amazon recommend movies to watch or items to buy to their customers by analyzing their activity and comparing it to other customers. (Marr, 2016)

Machine Learning algorithms are also used in the Natural Language Processing (NLP) to recognize speech and synthesis (understanding and pronouncing words). Siri, Alex, Google Assistant or Cortana are built with ML models to process the natural language. (Valchanov, 2018)

In the case of searching engines, Machine Learning algorithms are used to improve the accuracy of search engines. Google uses ML to improve the results of the search engine by analyzing the first links that are clicked or those that are dodged according to the input of the research. (Marr, 2016)

3.6 Process

In order to succeed in the implementation of a good ML model, there are some steps to follow successively. Yufeng G, a Google engineer, published a quick guide to introduce those steps and the path to follow. He takes as an example a bunch of glasses of alcohol with the aim to classify new inputs as *Beer* or *Wine*. In this section, we are going through this example to introduce the important steps of the implementation of a good ML model. (G, 2017)

A ML model needs lots of data to train and learn. The collecting phase is very important since the accuracy of our model will depend on those data (quantity and quality). When we want to build a model to respond to a problematic, we have to think about what kind of data we need and what are the features that matter and are relevant. (G, 2017)

In this case, the data we need are the color, the alcohol content and a label *Wine* or *Beer*. So we need to collect those information on some real examples in a shop for instance. (G, 2017)

Color (nm)	Alcohol %	Beer or Wine?
610	5	Beer
599	13	Wine
693	14	Wine

Figure 8 : extract of data collected (G, 2017)

The figure above shows an extract of the data collected for this case. They will be the data we are going to use to train our model. The color and the alcohol content are the features of the data that are labelled as *Wine* or *Beer*. (G, 2017)

The next step before starting the model building is the preparation of the data. We just have collected a lot of data and we have to work on them before feeding the model. The data need to be put together into a suitable place (a csv file for instance) and edited. (G, 2017)

In this case, the data are quite obvious and trivial, but raw data are sometimes less *ready-to-use* and need some changings. Those changings can be forms adjusting (apply

lowercase, …), manipulations (merge two cells into a single, type of cells,…), error corrections or more. (G, 2017)

The last step of the data preparation is to split the data into two subsets : training data and testing data. The training set will be used for the training phase of the model. The testing set will be used for evaluating and testing the accuracy of the model at the end. A good ratio between the training set and the testing set is 70/30 or 80/20. (G, 2017)

We have then to choose a model. We have seen before that there are many different models to implement Machine Learning. The choice of the model is crucial and depends on the problematic. There are models well suited for image data, music data, text data, … It also depends on the type of the data we have (labelled, unlabelled, …). (G, 2017)

In this example, we have data labelled as *Wine* or *Beer* with two features (color and alcohol content) and we want to predict the label of new inputs. A linear regression model suits very well for that kind of problematic. (G, 2017)

The next step is the most important one : the training. In this phase, we give the data collected and prepared to the model to train it. The model will learn how the data are connected together in order to be able to predict an output for a new input. In this case, the model learns what type of alcohol (*wine* or *beer*) a glass of color Y and alcohol content X is. We can then ask it : *I have a glass with a red and 11.5% content, what alcohol is it ?*. (G, 2017)

In this case, we have chosen a linear regression model. We have seen before that a linear regression model tries to find a function like y = ax + b. Here, the y represents the output label (*wine*, *beer*) and the x represents the input features (color and alcohol content). In other words, we have a function saying *an alcohol with a color C and an alcohol content A is a beer/wine*. The model works with the training data to find the best values for a and b (in the function previously introduced) in order to have the most closest to reality results. It finds the best possible linear relationship based on the available data. (G, 2017)

The next step is the evaluation of the model. We split the data into two subsets before. It is now time to use the testing set to evaluate the model. The testing set contains data with the inputs features (color, alcohol content) and the output label (*wine*, *beer*). The model uses the function he found during the training phase on the inputs features of the training set and checks if the output label given by the function is correct. (G, 2017)

The training phase evaluates the accuracy of the model. The accuracy is the ratio between correct and false results. The evaluation therefore gives a percentage score. (Jeevan, 2018)

In some cases, there are parameters called *hyperparameters* that are arbitrarily chosen at the time of model building. This is for example the *K* parameter that we saw before in a KNN or K-Means model. Depending on the results of our model, we can try to change them for better results. In this case, we don't have any hyperparameter. (G, 2017)

The goal of ML is to answer questions. When the model is trained and the accuracy is relevant, it is finally ready to be used. In this case, we can use the model to know whether a given drink is wine or beer, according to its color and alcohol content. (G, 2017)

3.7 In the case of a recommendation engine for movies

Recommendation engines are used everywhere today on internet. Netflix, Amazon, Youtube or Pinterest rely on recommender systems to make personalized recommendations to their users. They use those systems to filter millions of content and find the most appropriate ones for their customers. Recommender systems are well-studied and provide big values to businesses. (Liao, 2018)

The added value is so important that Netflix launched a contest in 2006 : the goal was to come up with a recommender system that could do better than theirs. Thousands of teams from 186 countries battled for three years. They finally rewarded 1 million the winners who managed to improve the system by 10%. It seems to be a lot of money but predicting the movies Netflix members will love is a key component of their service. (Lohr, 2009)

There are different approaches with different philosophies to build a recommendation engine : content-based recommendation, collaborative filter recommendation, hybrid system. In this section, we are going through those possibilities.

3.7.1 Content-based recommendation

The basic idea of a content-based recommendation system is to recommend items that are similar to other items that the user liked before. The mission of the engine is to calculate the similarity between items. (Ma, 2016)

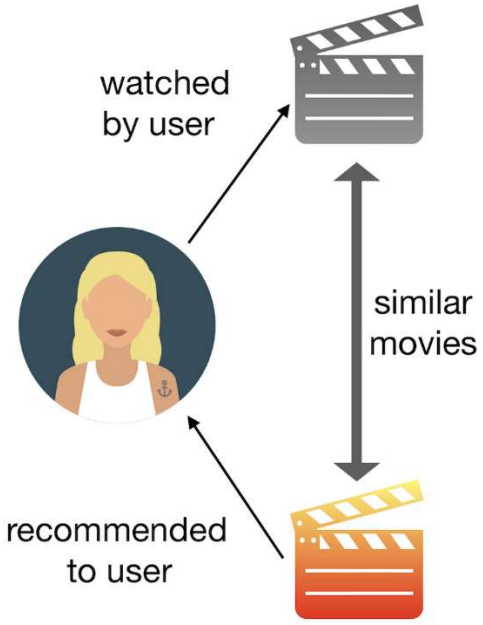

Figure 9 : content-based recommendation (Grimaldi, 2018)

This approach utilizes the characteristics of an item to find items with similar properties. Those characteristics are the keywords of an item. They can be special tags, the synopsis, director's name or actor's name in the case of movies. (Ma, 2016; Liao, 2018)

There are various ways to compute the characteristics of an item and calculate the similarity between items. The most common is to vectorize the items and mathematically calculate the similarity between the vectors. (Grimaldi, 2018)

A good way to vectorize an item is the Vector Space Model. The model extracts keywords from an item and vectorizes that item by TF-IDF. (Ma, 2016) Term Frequency-Inverse Document Frequency (TF-IDF) is an algorithm used to transform text into a meaningful representation of numbers (a vector) (Singh, et al., 2019).

To calculate the similarity between two vectors, a common way is to calculate the cosine similarity. We need first to calculate their dot product. The dot product between two vectors is equal to the projection of one of them on the other. We can then calculate the similarity with this dot product since the similarity between two vectors is the ratio between their dot product and the product of their magnitudes. (Grimaldi, 2018)

$$similarity = cos(\theta) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \|\boldsymbol{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}}$$

Figure 10 : cosine similarity between two vectors (Grimaldi, 2018)

This formula gives 1 if the two vectors are identical and -1 if they are completely opposite. The similarity is therefore finally a score between -1 and 1. (neo4j, 2019)

The final matrix of comparisons looks like the figure below :

	$Movie_1$	$Movie_2$	$Movie_3$	$\cdots$	$Movie_n$
$Movie_1$	1	0.158	0.138	...	0.056
$Movie_2$	0.158	1	0.367	...	0.056
$Movie_3$	0.138	0.367	1	...	0.049
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$Movie_n$	0.056	0.056	0.049	...	1

Figure 11 : matrix of comparisons (Grimaldi, 2018)

Content-based recommendation is a good system with good results but it suffers from *filter bubble*. It will only recommend items with similar content since it is obviously coded for that purpose. (Pinela, 2017) If a user likes *dark comedy*, the system will only recommend him *dark comedy* movies. The user could maybe also like *fantasy* but he will never know unless he gives it a try independently of the system. (Grimaldi, 2018)

3.7.2 Collaborative filter recommendation

The collaborative filter recommender systems are based on interactions between users and items. Instead of focusing on the characteristics of an item, the system compares a user's past behavior towards items (items previously purchased or rated) and similar actions made by other users. This system is then used to predict the users interest or rating of an item in order to recommend it or not. (Liao, 2018)

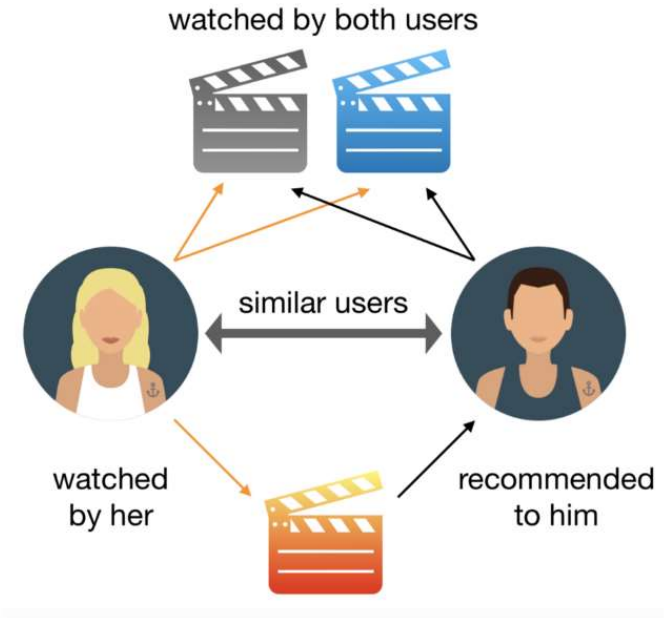

Figure 12 : collaborative filter recommendation (Grimaldi, 2018)

The collaborative filter recommendation engines can be of two different types : User-Based Collaborative Filtering (UB-CF) or Item-Based Collaborative Filtering (IB-CF) (Pinela, 2017).

User-Based Collaborative Filtering

UB-CF system finds similar users to a given user based on their tastes (the items they all purchased or rated well) and recommend to the given user the items he did not already buy or rate but the other similar users did (Pinela, 2017).

For example, user A and user B have seen the same movies and rated them identically but user A has seen and liked *Titanic* and user B has not. The system assumes that user A and B are similar and it will recommend *Titanic* to user B. (Pinela, 2017)

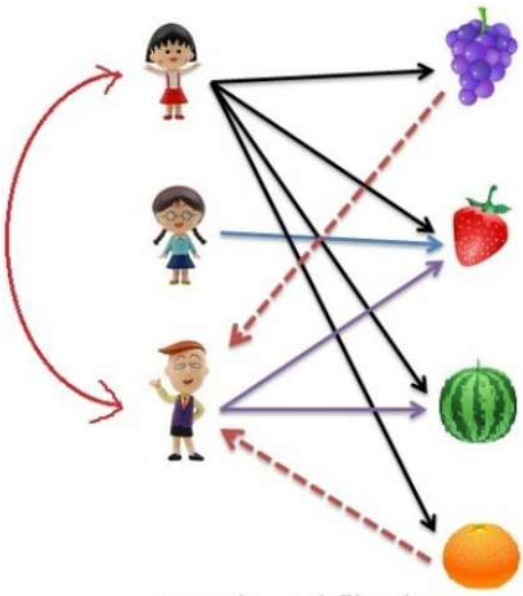

Figure 13 : UB-CF recommendation (Pinela, 2017)

A good model to implement such system is the K-Nearest Neighbors model that we have previously introduced. The KNN model finds a given user's k nearest neighbors (similar) and recommends movies to him based on those similar neighbors. (Pinela, 2017)

In order to find the users k nearest neighbors, the model needs to be fed with data that can be transformed into a User-Item matrix (Pinela, 2017).

Name	Avengers	Star wars	Thor	Spider-man	Iron Man
Alex	4	2	?	5	4
Bob	5	3	4	?	3
Tom	3	?	4	4	3

Figure 14 : User-Item matrix (王斌, 2018)

That kind of matrix makes possible the research of user's k nearest neighbors by the model.

The UB-CF system is good because it is easy to implement and gives more accurate results than a content-based system, but it suffers from various problems : sparsity (the percentage of people who rate items is low), scalability (if there are too many users and items in the matrix, the computational cost for the system is very high) and cold-start (new users have none or only few interactions to be compared with other users). (Pinela, 2017)

Item-Based Collaborative Filtering

IB-CF system recommends items to users that are similar to items they already purchased or rated well. It is therefore based on the similarity between items. But unlike a content-based system, it does not analyze the characteristics of items to define the similarity between them. It is a collaborative filtering, so it defines the similarity between items by other users interactions with items (purchase, rate,…). Basically, the system considers two items as similar if a lot of users rated them the same way. (王斌, 2018)

For example, users A,B, and C rated 5/5 *Spiderman*, 5/5 *Catwoman* and 1/5 *Gran Torino*. The system considers that *Spiderman* and *Catwoman* are similar and *Gran Torino* is different from them. Finally, if user D rated 5/5 *Spiderman* but did not watch *Catwoman*, the system recommends *Catwoman* to him because it assumes that he should like it, based on the other users behavior.

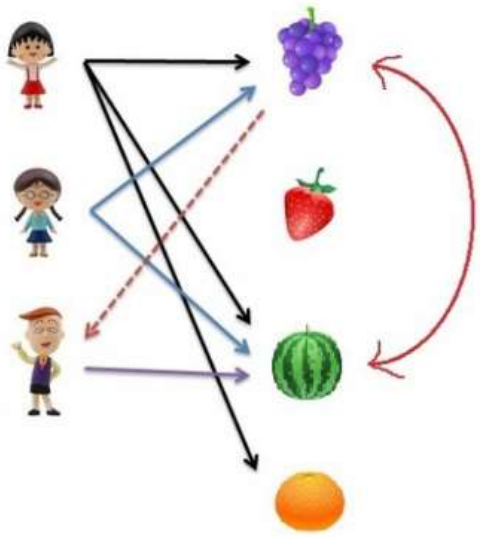

Figure 15 : IB-CF recommendation (Pinela, 2017)

Like for the UB-CF recommendations, a good model to implement an IB-CF system is the KNN model (Ma, 2016). It uses the same matrix than the UB-CF but inversed (Item-User matrix instead of User-Item matrix). We will see later in the empirical part of this thesis how it works concretely and how it can be implemented.

	Alex	Bob	Tom
Avengers	4	5	3
Star wars	2	3	?
Thor	?	4	4
Spider-man	5	?	4
Iron Man	4	3	3

Figure 16 : Item-User matrix (王斌, 2018)

IB-CF algorithms were first created by Amazon in 1998 who wanted to lower the computational cost of the UB-CF system. It has since been widely adopted by other big companies (such as Netflix or Youtube) due to the lowering of computational cost and the better scalability offered by this system compared to the UB-CF model. (王斌, 2018)

The lowering of the computational cost is explained by the fact that the model building and training can be done offline in the IB-CF algorithms and the trained model can then be used online to make predictions. It can not be done offline in the UB-CF model. The reason is that there are way more users than items on big websites (like Netflix or Amazon) : if a user changes his rate of an item, the similarity between users is way more affected by this change than the similarity between items. In order to make good recommendations, the UB-CF model must constantly re-calculate the similarities, so it has to be online. (王斌, 2018)

	Avenger	Star wars
Alex	1 (4)	1
Bob	5	2
Tom	3	3
...	2	5
...	1	3
...	4	4
...	5	5

Figure 17 : matrix when users number is larger than number of items (王斌, 2018)

The figure above illustrates the issue introduced in the previous paragraph. If Alex changes his rate of *Avenger* from 4 to 1, his similarity to other users has to be re-calculated because it changes everything (he was first similar to Bob and he is now similar to Tom). On the other hand, the similarity between *Avenger* and *Star Wars* has not to be

expressly re-calculated because there are 7 other unchanged lines assuming this comparison. (王斌, 2018)

3.7.3 Hybrid system

The hybrid approach is a combination between the systems introduced above. There are many ways to combine them (weighted : add scores from different recommenders, mixed : show recommendations from different recommenders, features combination : extract features from different sources and combine them as a single input, …). (Ma, 2016)

The power of a hybrid system is to combine the strengths and the weaknesses of all systems (Lineberry & Longo, 2018).

4 Project design

4.1 Project presentation

In this chapter and the next one, we are going to build the whole architecture of a web application supporting the integration of a ML model that recommends movies to users.

The main objectives of this project are :

- To provide a good-looking, responsive and user-friendly user interface (UI)
- To implement a well-suited ML model in order to recommend pertinent movies to users

The figure below shows the use cases the web application should support :

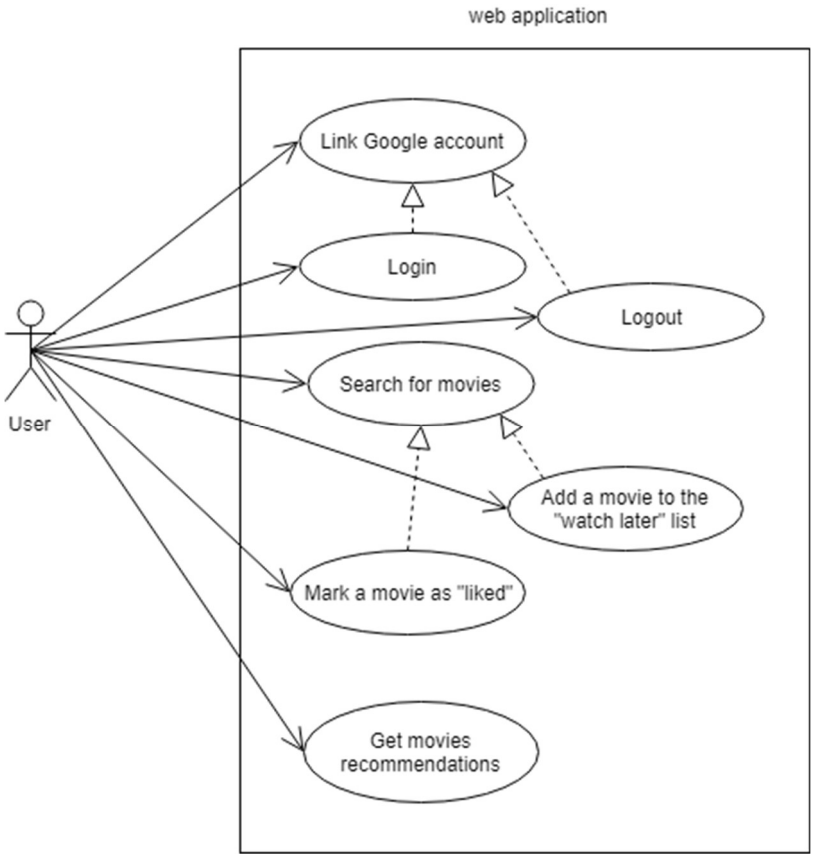

Figure 18 : Use Case Diagram

The user can use the web application to :

- Link his Google account to the platform in order to successfully login and get authenticated, and logout
- Search for all available movies across a large database
- Mark a found movie as *liked*

- Add a found movie to the *watch later* list
- Get movies recommendations based on his list of liked movies

In order to implement all those use cases, the project needs a frontend, an authentication, a database, a backend, a ML model, a dataset and several Application Programming Interfaces (API). These different parts are described in the followings sections.

This project is not commissioned by a third party. This is a personal project with the aim to implement the results of the research of the previous chapter in a concrete web structure.

4.2 Project setup

The project architecture is quite complex and many choices have to be done about the technologies of the different parts of the structure.

In this section, we are going through those different choices for each part of the project and we explain why they have been made.

4.2.1 Frontend

The frontend has to provide a user-friendly, efficient and good-looking UI. In order to achieve these goals, the ReactJS framework was chosen.

ReactJS is a JavaScript library useful for building user interfaces. It was first created by Facebook and it is now an open-source library. The power of ReactJS is that it is fast, scalable and simple. (Pandit, 2018)

Many big companies developed their frontend with ReactJS : Facebook, Instagram, Netflix, WhatsApp, Yahoo! Mail, the New York Times, … (Warcholinski, 2019)

4.2.2 API

The web application needs an API to fetch information about movies and get the whole list of existing movies.

There are many available movies APIs. We chose to use TMDB because they have a large database of movies and provide an easy-to-use API.

TMDB is an API that allows to search for and find movies, TV shows and people. This is free of charge if it is used in a non-commercial project. (TMDB, 2019)

The movies endpoint of the API gives all needed information about movies : primary info, plot keywords, cast, crew, title, synopsis, release information, genre, … (TMDB, 2019)

4.2.3 Database and users

We use the TMDB API to access their database of movies but we need our own database to store information about users : accounts, the movies they liked and the movies they added to the *watch later* list.

Since the time is a constraint in the development of this project, we chose to use Google Firebase Realtime Database and Google Firebase Authentication. They are easy to use and to implement.

Google Firebase Realtime Database is a cloud-hosted NoSQL database. Data are stored as JavaScript Object Notation (JSON). They provide a JavaScript Software Development Kit (SDK) to implement it in a web project. (Google, 2019)

Google Firebase Authentication is a backend service to authenticate users to an app. It supports authentication using email/password, phone numbers, Google, Facebook and more. They provide a JavaScript SDK to implement it in a web project. (Google, 2019)

We chose to use a Google authentication since it is quickly implemented and very powerful.

4.2.4 Backend

The backend has to run the Machine Learning model and to provide recommendations through a built-in API.

In the previous chapter, we saw that Python was the best programming language to implement Machine Learning. We chose to use the Django Rest Framework (DRF) to build our backend.

DRF is a Python framework useful to build a backend and an API. It is used and trusted by big companies like Mozilla, Red Hat or Heroku. (DRF, 2019)

4.2.5 Machine Learning model

In the previous chapter, we saw that there are many ways to implement Machine Learning into a recommender system for movies. We saw that IB-CF system and hybrid system are the best choices.

We chose to build our recommender system with the IB-CF approach since the time is a constraint and a hybrid system is very complex and long to implement.

We will use the KNN model previously introduced to build our IB-CF recommender system. This system will be integrated in the DRF backend.

We will use the Scikit-Learn library to build this model. Scikit-Learn is a Python library for implementing Machine Learning. It has a module named *Neighbors* useful to build KNN models. (Scikit-Learn, 2019)

We will also use other libraries :

- NumPy : it is a Python library for scientific computing (NumPy, 2019)
- SciPy : it is a Python library for mathematics, science and engineering (SciPy, 2019)
- Pandas : it is a Python library providing high-performance, easy-to-use data structures and data analysis tools (Pandas, 2019)

4.2.6 Dataset

In the previous chapter, we saw that data are the core of a ML model in order to train it.

We chose to build a KNN model to implement a IB-CF recommender system, so we need data of movies ratings by users.

GroupLens provides datasets of movies ratings by users extracted from MovieLens. They can be used in non-commercial and research projects. (GroupLens, 2019)

There are two datasets : the first one with 27’000’000 ratings applied to 58’000 movies by 280’000 users, the second one with 100’000 ratings applied to 9’000 movies by 600 users. (GroupLens, 2019)

In this project part of the thesis, those two datasets will be called the *small dataset* and the *large dataset*.

4.3 Project architecture and interactions

Since the project architecture is quite complex, we have to introduce its design and structure. This section is going through the whole architecture and presents the interactions between the different parts of it.

The figure below shows the global architecture of the project :

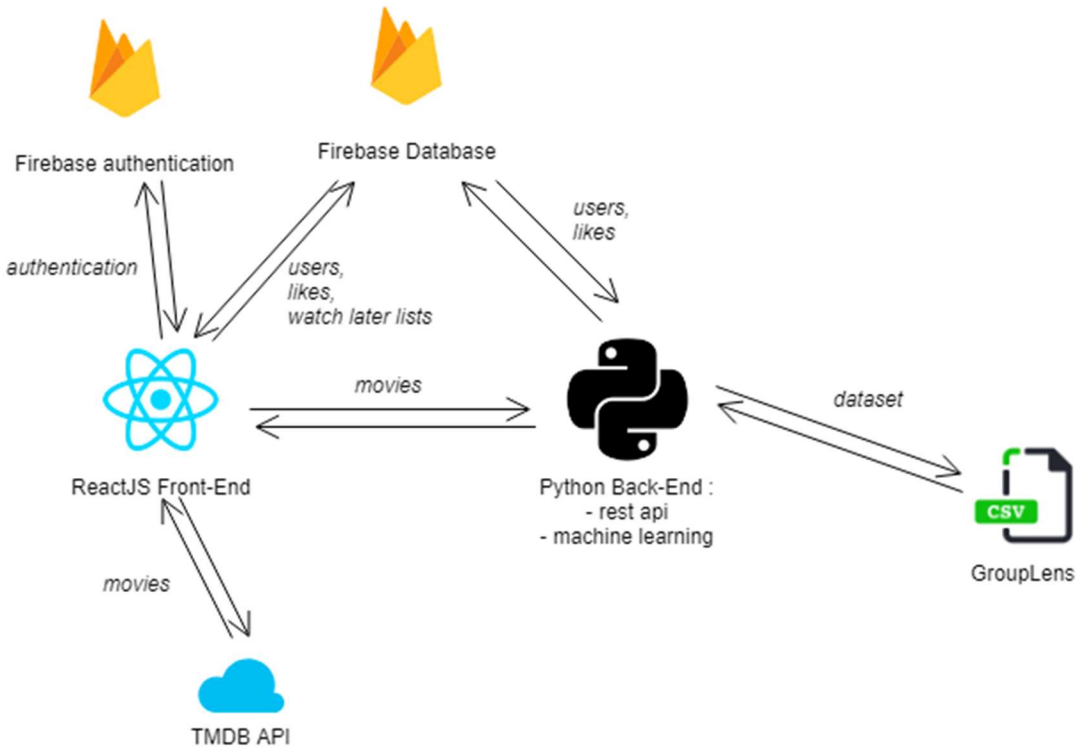

Figure 19 : project architecture

The frontend communicates with Firebase Authentication for logging in users through their Google account and giving them access to the web application. It also uses Firebase Authentication for logging out users.

The frontend communicates with Firebase Database to store, delete, update and retrieve data about users : the TMDB identifier numbers (IDs) of movies they liked and the TMDB IDs of movies they added to the *watch later* list.

The frontend communicates with the TMDB API to get data and metadata about all existing movies. It needs to call the API to display the movies and their information on :

the home page, the *like*/*watch later* lists pages, the search for movies page and the recommendations page.

The frontend communicates with the backend API to get movies recommendations for the users. It sends the list of the TMDB IDs of movies a user liked and the backend returns a list of recommendations after having found similar movies through its ML model.

The backend uses the dataset of GroupLens, which is provided in Comma-separated values (CSV) files, to build the Machine Learning model.

The backend communicates with Firebase Database to get the lists of movies liked by users and add them into the CSV files. It will periodically do this process to improve the accuracy of the model by feeding it with more data.

5 Project implementation

5.1 Frontend

Since the main purpose of this thesis is not to build a ReactJS application, we are not going through the whole design of the frontend. This section focuses on the global structure of the frontend and some important points in order to have a good general understanding of its design.

Global structure

The figure below shows the structure of the frontend application :

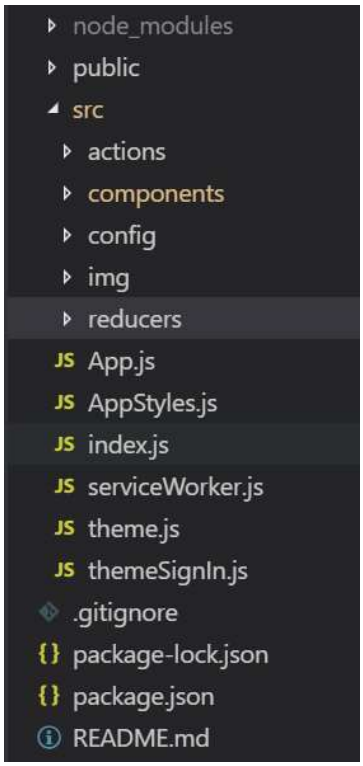

Figure 20 : Frontend structure

The *node_modules* folder contains all needed libraries to run a ReactJS app and other dependencies included in the *package.json* file. The *src* folder contains the application :

- *actions* and *reducers* folders contain all Redux files. Redux is a React library used to simplify the management of the React states in the application (Redux, 2019). We are not covering this part in this thesis.

- *components* folder contains all concrete components used to build the pages of the application
- *config* folder contains configuration files (api keys, Firebase configuration, …)
- *theme.js* and *themeSignIn.js* files are customizations of the rendered styles
- *index.js* and *App.js* are the entry points of the application (main template and routing)

Routing

The *App.js* file contains the BrowserRouter of the React Router library. Its goal is to load components to render according to the Uniform Resource Locator (URL) accessed by the user.

```
{/* routing */}
<Route exact path="/" component={SignIn} />
<Route path="/home" component={requireAuth(Home)} />
<Route path="/mylikes" component={requireAuth(MyLikes)} />
<Route path="/watchlater" component={requireAuth(WatchLater)} />
<Route path="/search/:param" component={requireAuth(Search)} />
```

Figure 21 : Routing

The router will load :

- the *SignIn* component when the accessed URL is */*, that component renders the SignIn page where the user can log in
- the *Home* component when the URL is */home*, that component renders the home page of the application
- the *MyLikes* component when the URL is */mylikes*, that component renders the page where the user can see his list of liked movies, remove them or get recommendations
- the *WatchLater* component when the URL is */watchlater*, that component renders the page where the user can see his list of ”Watch Later” movies or remove them
- the *Search* component when the URL is */search/xxx*, that component renders the results of the search on an appropriate page when the user search for movies

The *Home*, *MyLikes*, *WatchLater* and *Search* components are wrapped into a function called *requiredAuth* that checks if the user is authenticated before rendering the component. This process is detailed in the next point.

Authentication

The application should be only accessible by authenticated users. In order to achieve this goal, we have a custom functionnal component that checks it for specified components (listed in the previous point) :

```
export default function (ComposedComponent) {
    class Authentication extends Component {
        static contextTypes = {
            router: PropTypes.object
        };

        // redirect to "/" if not authenticated
        componentWillMount() {
            if (this.props.authenticated === null) {
                this.context.router.history.push("/");
            }
        }

        // redirect to "/" if not authenticated
        componentWillUpdate(nextProps) {
            if (!nextProps.authenticated) {
                this.context.router.history.push("/");
            }
        }

        // render the component if authenticated
        render() {
            if (this.props.authenticated) {
                return <ComposedComponent {...this.props} />;
            }
            return null;
        }
    }

    function mapStateToProps(state) {
        return { authenticated: state.auth };
    }

    return connect(mapStateToProps)(Authentication);
}
```

Figure 22 : Authentication check

This solution is inspired by the tutorial *Adding Authentication To React Redux Firebase App* written by Bernard Bado on medium.com (Bado, 2018).

This function receives a component as a parameter and then renders it if the user is authenticated or redirect to the URL */* if he is not. The *componentWillMount()* method checks it on the first time the component is rendered and the *componentWillUpdate()* method checks it each time the component is updated. (Bado, 2018)

The user can log in by clicking on the button *Sign in with Google* on the SignIn page (URL */*) :

```
<Button variant="contained" color="primary" className={classes.signInButton} onClick={this.props.signIn}>
    <SignInIcon className={classes.signInIcon} />
    Sign in with Google
</Button>
```

Figure 23 : Sign In button

This button calls the *signIn()* method on click : (Bado, 2018)

```
export const signIn = () => dispatch => {
    authRef
        .signInWithPopup(provider)
        .then(result => { })
        .catch(error => {
            console.log(error);
        });
};
```

Figure 24 : signIn() method

This method calls the *signInWithPopup()* method of Firebase Authentication which opens a pop up in the user's browser to allow him to sign in with his Google account.

Writing and reading data in Firebase Database

The figure below shows the structure of our Firebase Database :

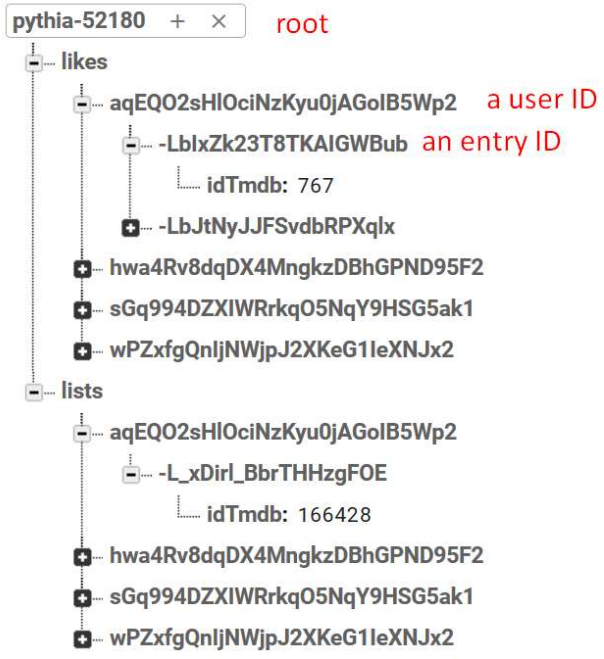

Figure 25 : Firebase Database structure

The first node *pythia-52180* is the root node. Under the root, there are two nodes : *likes* and *lists*. The *likes* node is the lists of movies the users liked. The *lists* node is the lists of movies the users added to their *Watch Later* list.

Under those two nodes, the structure is the same : there is a node for each user. And finally under the users, there is an entry for each movie and its ID in the TMDB database.

The following code is inspired by the tutorial *How to Integrate React Redux and Firebase in 3 Simple Steps* written by Bernard Bado on medium.com (Bado, 2018).

The figure below shows how we can add a liked movie for a given user in the database from our frontend : (Bado, 2018)

```
// add a like
export const addLike = (newLike, uid) => async dispatch => {
    likesRef
        .child(uid)
        .push()
        .set(newLike);
};
```

Figure 26 : Add a liked movie

This code adds a new entry under the given user ID node that is under the *likes* node.

The figure below shows how we can remove a liked movie in the database from our frontend : (Bado, 2018)

```
// remove a like
export const removeLike = (likeId, uid) => async dispatch => {
    likesRef
        .child(uid)
        .child(likeId)
        .remove();
};
```

Figure 27 : Remove a liked movie

This code removes an entry under the given user ID node that is under the *likes* node.

The figure below shows how we can get all the movies a user liked from our frontend : (Bado, 2018)

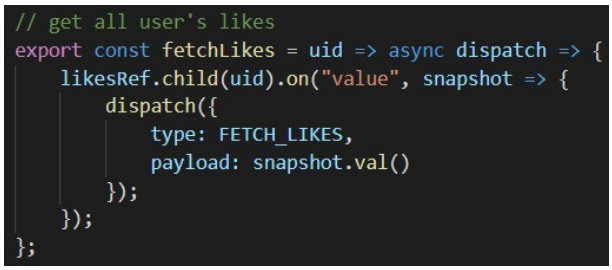

Figure 28 : Get all movies a user liked

This code get all the movies a given user liked and dispatch them in the global state of our application. It means that we can then get them from everywhere in our application. (Bado, 2018)

The process is exactly the same for the *lists* (*Watch Later* movies) node :

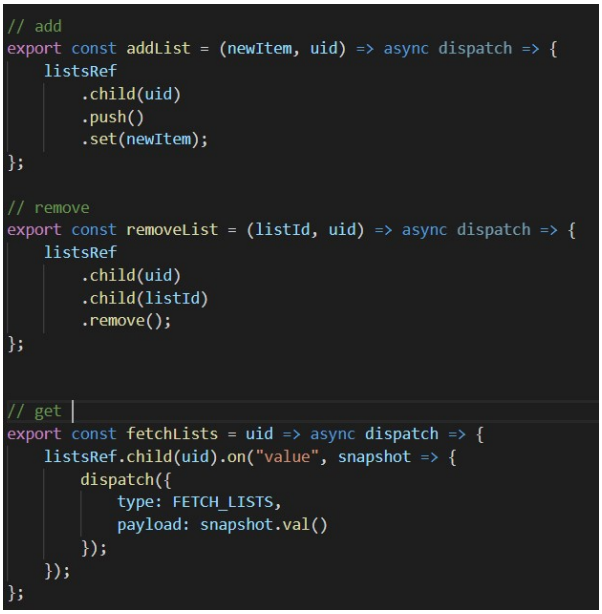

Figure 29 : Database actions for "Watch Later" movies

Fetching movies data from TMDB

The Firebase Database only stores the ID of a movie of the TMDB database. In order to display the movies with the full information (director's name, cast, title, plot, poster, …), we need to fetch the data from TMDB.

The figure below shows how the frontend fetches the data from TMDB for the movies a user liked :

```
// fetch data from tmdb api
loadDataMovies = () => {
    let dataMovies = new Array;
    const pushes = []
    // go through the idTmdb of the likes and fetch the movie and push to the array
    for (var like in this.props.likes) {
        const url = `https://api.themoviedb.org/3/movie/${this.props.likes[like].idTmdb}?api_key=${tmdbApiKey}`;
        pushes.push(
            fetch(url)
                .then(response => response.json())
                .then(json =>
                    dataMovies.push(json)
                )
        );
    }
    //check if the fetches are over and set the array to the state of dataMovies
    Promise.all(pushes).then(() => {
        this.setState({ dataMovies: dataMovies }, () => {
            if (dataMovies.length == 0)
                this.setState({ noResult: 1 })
            else
                this.setState({ noResult: -1 })
        })
    });
};
```

Figure 30 : Fetching data from TMDB

This is the workflow of the code :

1. we create a temporary array that will contain the movies fetched from TMDB
2. we fetch the data from TMDB for each movie a user liked and push them into the temporary array
3. when the movies are all fetched, we assign the temporary array to the state *dataMovies* of the component
4. if there are not any result in the temporary array we assign 1 to the state *noResult* of the component, if there are we assign -1

It works similarly for the other pages that need to fetch data from TMDB.

Get recommendations

The figure below shows how the frontend gets movies recommendations from the backend :

```
// fetch ids of suggestions
fetch(backendUrl + '/suggestions',
    {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Api-Token': backendApiToken,
            'Api-Secret-Key': backendApiSecretKey
        },
        body: JSON.stringify(idsToFetch)
    })
    .then(response => response.json())
    .then(json => this.setState({suggestionsDialogOpen : true}, this.fetchSuggestionsFromTmdb(json.results)))
    .catch(err => console.error(err))
```

Figure 31 : Fetching movies recommendations from the backend

This code sends the TMDB IDs of movies the user liked to the backend and the backend sends back movies recommendations based on those IDs. The backend only sends the TMDB IDs of recommendations, so we need then to fetch the full data from TMDB based on those IDs :

```
// fetch suggestions from TMDB with the ids of suggestions
fetchSuggestionsFromTmdb = (results) => {
    let suggestionsFetched = new Array;
    const pushes = []
    // go through the tmdbId of the results and fetch the movie and push to the array
    for (var item in results) {
        const url = `https://api.themoviedb.org/3/movie/${results[item].tmdbId}?api_key=${tmdbApiKey}`;
        pushes.push(
            fetch(url)
                .then(response => response.json())
                .then(json =>
                    suggestionsFetched.push(json)
                )
        );
    }
    //check if the fetches are over and set the array to the state of suggestions
    Promise.all(pushes).then(() => {
        this.setState({ suggestions: suggestionsFetched })
    });
}
```

Figure 32 : Fetching data from TMDB

This code works the same way as we just detailed in the last point.

Once it is done, the recommendations are saved in the state of the application and shown to the user in a dialog popup in the application.

This is the point of view from the frontend, we will see later in the next sections how it works inside the backend.

Search for movies

When a user types the name of a movie in the search bar, the frontend fetches the results from TMDB. If there are too many results, TMDB splits the response into several pages and we have to call each page of results separately :

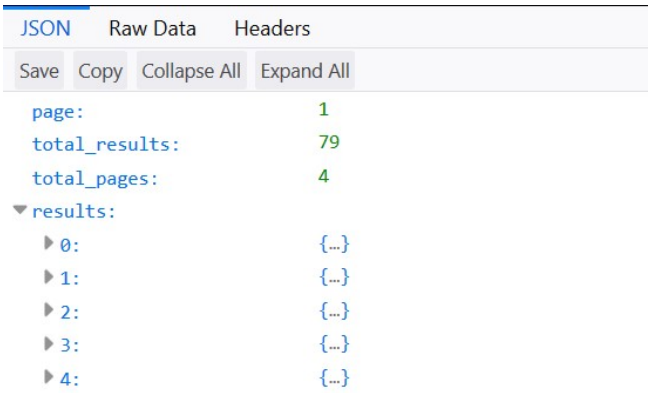

Figure 33 : Results of a search from TMDB

The figure above shows the results when we search for *Gladiator*. There are 79 results split into 4 pages and this first response shows the 20 first results. If we want the 20 next results, we have to call again the TMDB API and add the parameter *page=2* in the request :

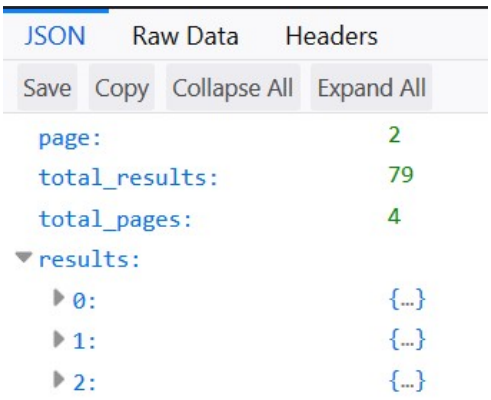

Figure 34 : This is the second page of results

This how the frontend does this process :

```
// fetch data from tmdb api
loadDataMovies = () => {
    const { page, dataMovies } = this.state;
    const url = `https://api.themoviedb.org/3/search/movie?api_key=${tmdbApiKey}&query=${this.props.match.params.param}&page=${page}`;
    fetch(url)
        .then(response => response.json())
        .then(json =>
            this.setState({
                dataMovies: [...dataMovies, ...json.results], // append to existing list
                scrolling: false,
                total_pages: json.total_pages,
                total_results: json.total_results
            })
        );
};
```

Figure 35 : Fetching results of a search from TMDB

The *loadDataMovies()* method fetches the URL endpoint of the TMDB API with two parameters : the query (the name of a movie the user typed into the search bar) and the current number of the results page. It starts by fetching the first page of results. It sets

then the response to the state of the component : the results, the total number of pages and the total number of results.

The first results are shown to the user and then if he scrolls down the page, the frontend calls the next page of results to the API :

```
// fetch the next page of results
loadMoreDataMovies = () => {
    this.setState(
        prevState => ({
            page: prevState.page + 1,
            scrolling: true
        }),
        this.loadDataMovies
    );
};
```

Figure 36 : Calling the next page of results

This code updates the current number of the results page in the state of the component and call again the *loadDataMovies()* method just detailed before that appends the new results to the existing state. The component is then refreshed and the new results are shown to the user next to the previous ones.

Screenshots of the final render

Here are a few screenshots of the full implementation of the frontend :

Figure 37 : Sign In page

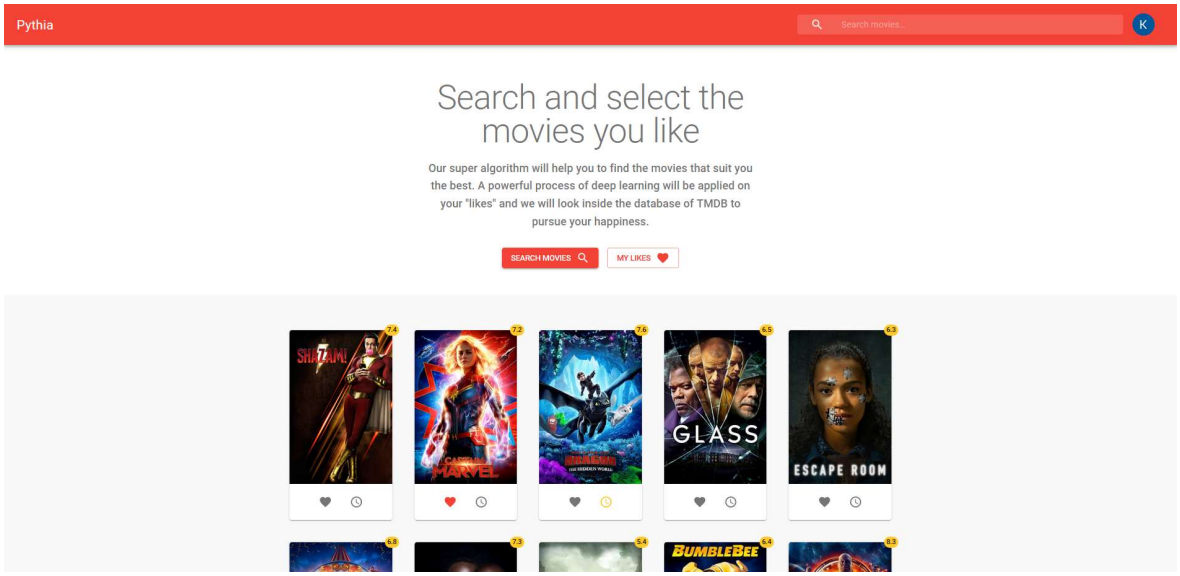

Figure 38 : Home page

The Home page shows the last popular movies on TMDB.

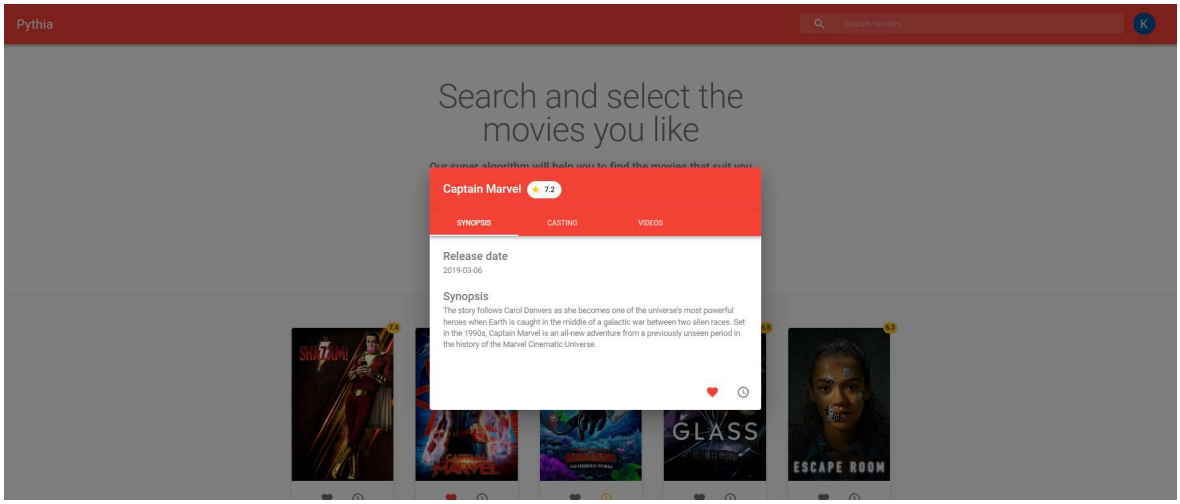

Figure 39 : Details of a movie I

When a user clicks on a movie, he can see the its details, such as the synopsis, the casting or the videos (trailers, teasers, …).

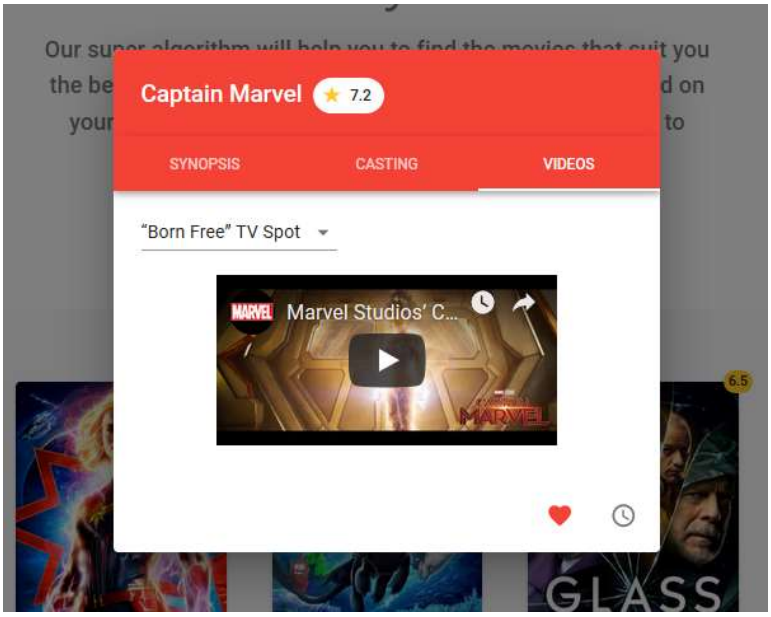

Figure 40 : Details of a movie II

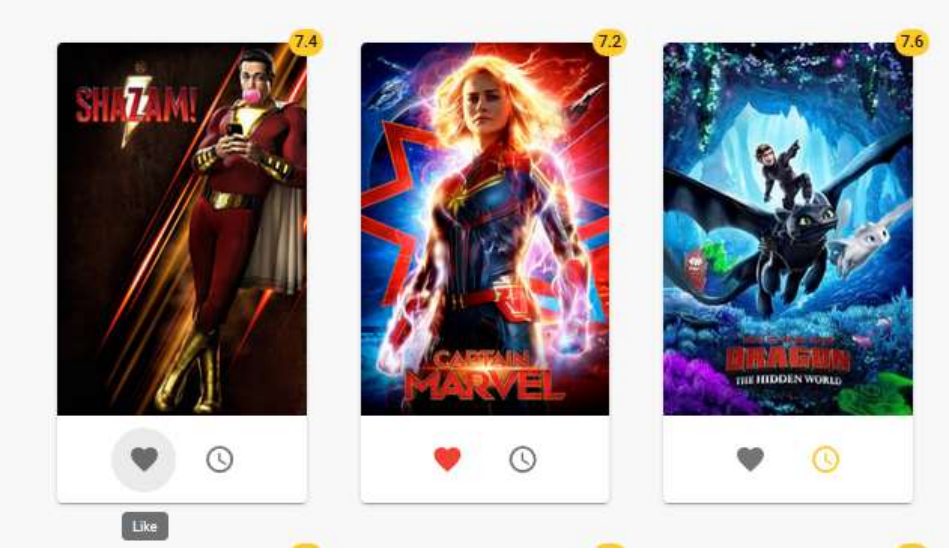

Figure 41 : How to like a movie

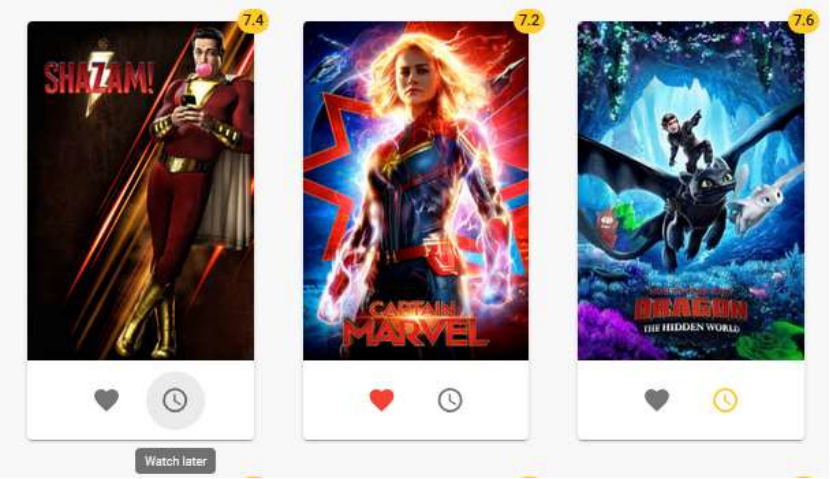

Figure 42 : How to add a movie to the "Watch Later" list

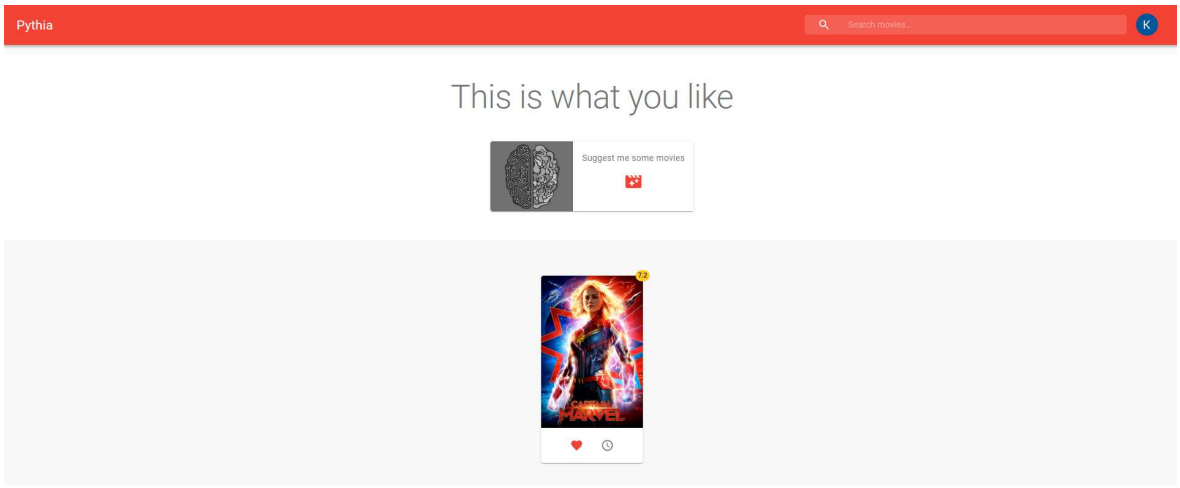

Figure 43 : The page of the liked movies

This page shows all the movies a user liked. He can then click on *Suggest me some movies* to fetch recommendations from the backend. We will see in the next sections the results and how it works.

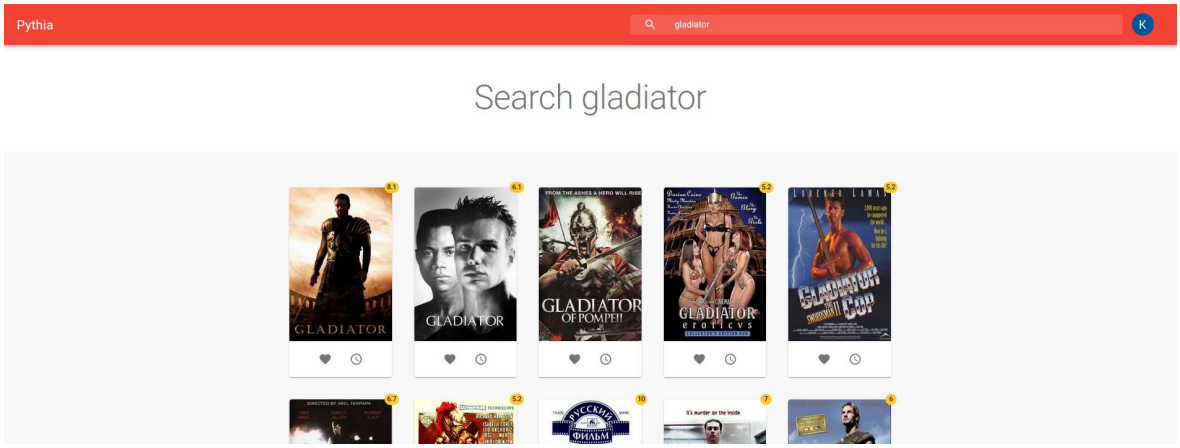

Figure 44 : The page of the search results

Thie figure above shows the results of a research.

The frontend is responsive and adapts its display if the user is on a smartphone :

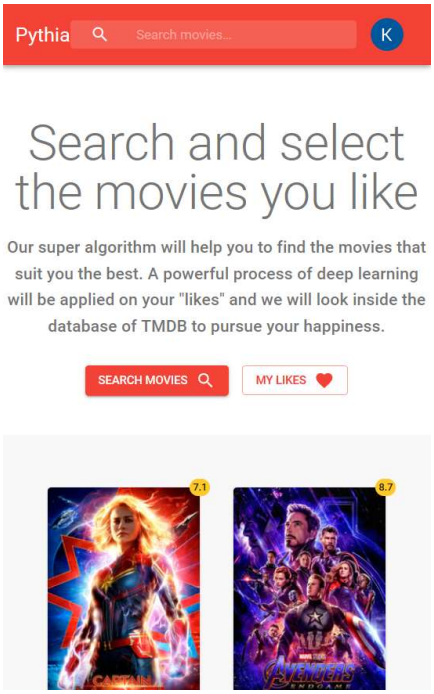

Figure 45 : responsive design I

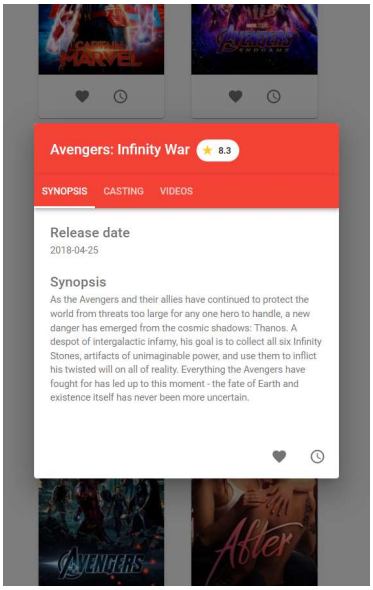

Figure 46 : responsive design II

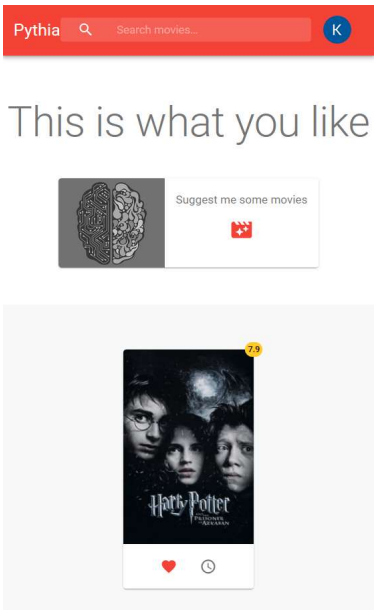

Figure 47 : responsive design III

5.2 Backend and API

In this section, we are going through the Django Rest Framework backend structure.

Global structure

The figure below shows the structure of the backend :

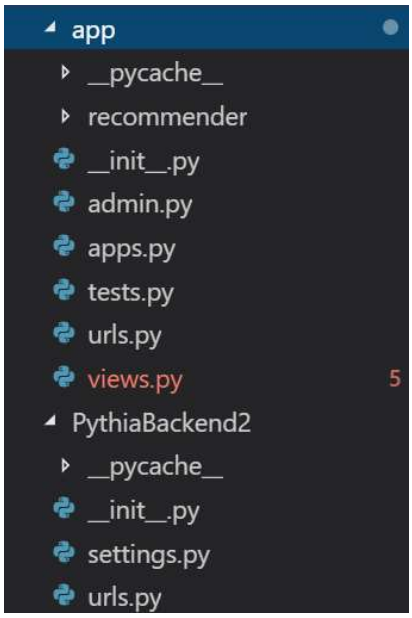

Figure 48 : Backend structure

The *PythiaBackend2* folder contains the global configuration of the backend :

- The *settings.py* file configures the API

- The *urls.py* file sets up the root of the URLs

The app folder contains the application files and the API :

- The *recommender* folder contains the ML model and the data. We will see it in the next section.
- The *urls.py* file sets up the URLs of the API and the endpoints
- The *views.py* files contains the logic of the API

The goal of our backend is to receive the TMDB IDs of movies liked by a user from the frontend and to send it back the TMDB IDs of movies recommendations. The backends receives those IDs on the endpoint */api/suggestions* of its API by the POST method.

Routing

The root of the URls is defined in the file *PythiaBackend2/urls.py* as :

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('app.urls'))
]
```

Figure 49 : Routing root

The URL */admin* allows to access the administration of the backend in a browser :

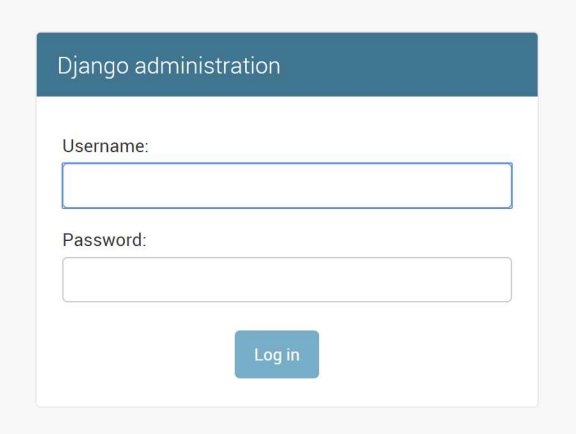

Figure 50 : Administration log in

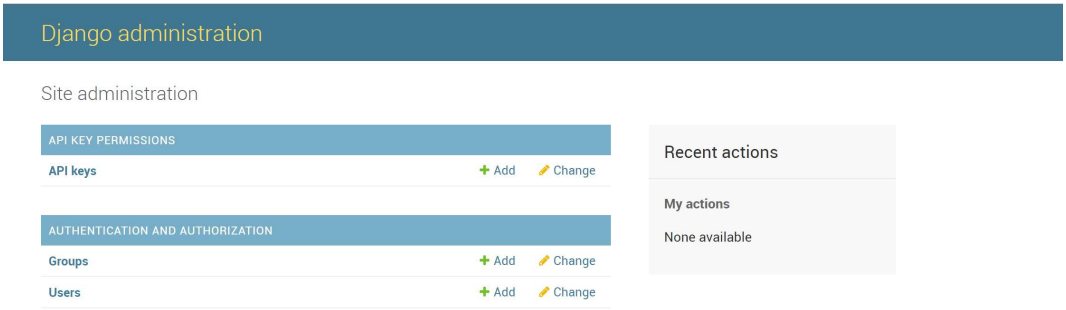

Figure 51 : Administration dashboard

The URL */api* allows to call the API endpoints included in the file *app/urls.py* with the URL */api/xxx* :

```
urlpatterns = [
    path('suggestions', views.Suggestions.as_view()),
]
```

Figure 52 : API endpoints

So the endpoint */api/suggestions* is finally treated by the logic inside the *views.py* file.

API logic

When the URL */api/suggestions* is accessed by the POST method, the following code inside the *views.py* file is called :

```python
class Suggestions(APIView):
    """
    Suggestions API
    """

    # get recommender
    recommender = Recommender()

    def post(self, request, format=None):
        data = request.data
        # prepare list of results
        results = []
        checkedIds = []
        # extend the list of results of the recommendations for each movie sent by the user
        for item in data:
            results.extend(self.recommender.make_recommendations(item['tmdbId'], 10))
            checkedIds.append(item['tmdbId'])
        # filter checkedIds from results
        filteredResults = [x for x in results if x not in checkedIds]
        # filter unique values
        uniqueResults = set(filteredResults)
        # format results
        formatedResults = {'results' : [{"tmdbId": i} for i in uniqueResults]}

        return Response(formatedResults)
```

Figure 53 : Suggestions API

This code does the following things :

1. Instantiate our *Recommender* object in order to make recommendations
2. Get the TMDB IDs of liked movies sent by the frontend and store them in the variable *data*
3. Declare an empty array *results* that will be filled by the TMDB IDs of the recommended movies
4. For each TMDB ID in *data*, get 10 recommendations by calling the method *make_recommendations* of the *Recommender* object, and push them into the array *results*
5. Remove duplicate movies and movies already seen by the user from the array *results*
6. Format the array *results* as a JSON
7. Return the JSON of results to the frontend

Security

The backend is secured by an API key and can only be accessed if the frontend adds that key to its requests.

This API key is set in the administration :

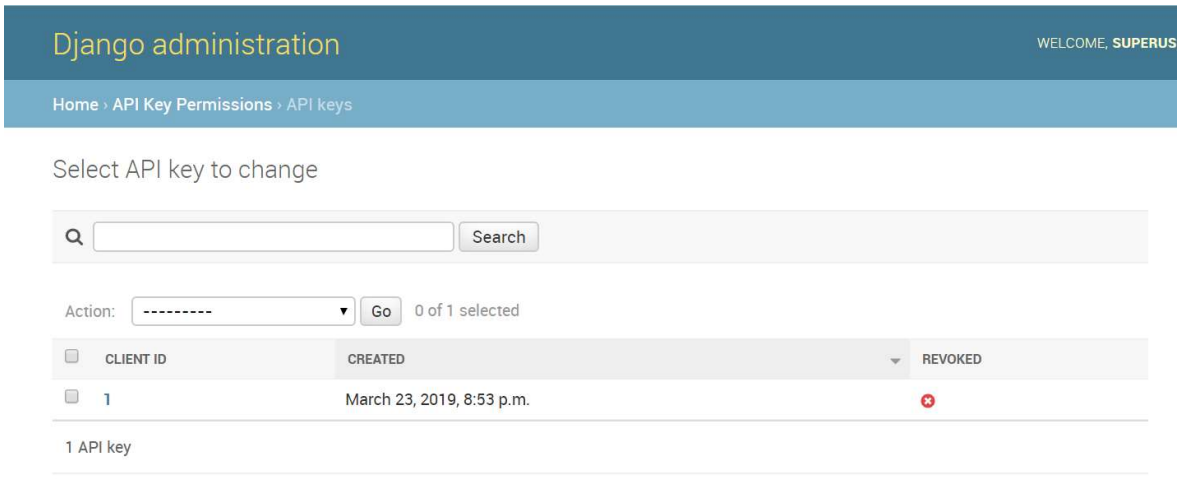

Figure 54 : API keys

5.3 Machine Learning model

We saw that the backend contains a *Recommender* folder and uses a *Recommender* object to make recommendations. In this section we are going through the implementation of that recommender system built in the backend.

This implementation and code are partially inspired by a guide about ML written by Kevin Liao on Towards Data Science (Liao, 2018).

In order to go through the implementation of the system, we are following the process of ML implementation introduced in the previous section *Process*.

Collecting the data

The GroupLens dataset contains 4 CSV files :

- *links.csv*
- *movies.csv*
- *ratings.csv*
- *tags.csv*

We need the three first files :

	A	B	C
1	movieId	imdbId	tmdbId
2	1	114709	862
3	2	113497	8844
4	3	113228	15602
5	4	114885	31357

Figure 55 : links.csv

The *links.csv* file contains the correspondance between the ID of the movie in the dataset, its ID on IMDB and its ID on TMDB.

	A	B	C
1	movieId	title	genres
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
3	2	Jumanji (1995)	Adventure|Children|Fantasy
4	3	Grumpier Old Men (1995)	Comedy|Romance
5	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
6	5	Father of the Bride Part II (1995	Comedy

Figure 56 : movies.csv

The *movies.csv* file contains some information about the movies, such as the title and the genres.

	A	B	C	D
1	userId	movieId	rating	timestamp
2	1	1	4	964982703
3	1	3	4	964981247
4	1	6	4	964982224
5	1	47	5	964983815
6	1	50	5	964982931

Figure 57 : ratings.csv

The *ratings.csv* file contains the user's ratings of movies. It also contains the date (timestamp) of the submitted rate.

The structure of the dataset is exactly the same for both sets : the small one (100'000 ratings) and the large one (27'000'000 ratings).

Data preparation

Before feeding the model, we need to apply some transformations on the raw data retrieved on GroupLens.

First, we merge the TMDB ID column of the *links.csv* file into the *movies.csv* file :

	A	B	C	D
1	movieId	title	genres	tmdbId
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	862
3	2	Jumanji (1995)	Adventure|Children|Fantasy	8844
4	3	Grumpier Old Men (1995)	Comedy|Romance	15602
5	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	31357
6	5	Father of the Bride Part II (1995	Comedy	11862

Figure 58 : new movies.csv

It will be lighter for the infrastructure to only use one file to get the movies information.

Now we have to prepare the data in the backend. All the preparation process (and then the model building) is done in a python class *Initializer* that we create. This class has some parameters that we set first in its constructor (Liao, 2018) :

```
self.path_movies = 'app/recommender/ml-latest-small/ml-latest-small/movies.csv'
self.path_ratings = 'app/recommender/ml-latest-small/ml-latest-small/ratings.csv'
```

Figure 59 : parameters in constructor

The *path_movies* and *path_ratings* parameters are both system pathes to the needed CSV files.

Then we can start to prepare the data in a method *_prep_data()* of the class. Firstly, we retrieve the data from the CSV files with the pandas (pd) library : (Liao, 2018)

```
# read data :
df_movies = pd.read_csv(
    os.path.join(self.path_movies),
    usecols=['movieId', 'title', 'tmdbId'],
    dtype={'movieId': 'int32', 'title': 'str', 'tmdbId': 'int32'})
df_ratings = pd.read_csv(
    os.path.join(self.path_ratings),
    usecols=['userId', 'movieId', 'rating'],
    dtype={'userId': 'int32', 'movieId': 'int32', 'rating': 'float32'})
```

Figure 60 : retrieve data from CSV files

The *movies.csv* is stored in the variable *df_movies* and the *ratings.csv* is stored in the variable *df_ratings* (Liao, 2018).

To build the IB-CF system we need to feed a KNN model with a Item-User matrix. In this case we build a Movie-User matrix based on the *ratings.csv* file : (Liao, 2018)

```
# pivot and create movie-user matrix
movie_user_mat = df_ratings.pivot(
    index='movieId', columns='userId', values='rating').fillna(0)
```

Figure 61 : Movie-User matrix

The *pivot* method of the pandas library transforms the *ratings.csv* file into the Movie-User matrix and fills the empty values (the movies that are not rated by a user) with a 0. The created matrix looks like that : (Liao, 2018)

```
In [31]: df_movie_features.head(5)
```

Out[31]:

userId	4	5	10	14	15	18	19	26	31	34	...	283199	283204	283206	283208	283210	283215	283219	283222	283224	283228
movieId																					
1	4.0	0.0	5.0	4.5	4.0	0.0	0.0	0.0	5.0	0.0	...	5.0	0.0	0.0	4.5	0.0	4.0	4.0	0.0	0.0	4.5
2	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	4.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0

Figure 62 : Movie-User matrix (Liao, 2018)

The matrix is very sparse (lots of 0 values) and it would lower the calculation performances of the KNN model. To minimize that negative impact, we transform the matrix into a SciPy sparse matrix : (Liao, 2018)

```
# transform matrix to scipy sparse matrix
movie_user_mat_sparse = scipy.sparse.csr_matrix(movie_user_mat.values)
```

Figure 63 : matrix transformation

The last thing to do is to create a hashmap that allows to retrieve the TMDB ID of a movie in the *movies.csv* based on its ID in the Movie-User matrix : (Liao, 2018)

```
# create mapper from movie tmdbId to index of the movie in the movie_user_matrix
hashmap = {
    movie: i for i, movie in
    enumerate(list(df_movies.set_index('movieId').loc[movie_user_mat.index].tmdbId))
}
```

Figure 64 : hashmap

This hashmap associates the ID of a movie in the matrix to its TMDB ID in the *movies.csv* file. It is useful since the backends has to return the TMDB IDs to the frontend.

The *_prep_data()* method finally returns the matrix and the hashmap :

```
return movie_user_mat_sparse, hashmap
```

Figure 65 : values returned by the method

The method can now be called in the constructor of the class and save the matrix and the hashmap in two new attributes :

```
self.movie_user_mat_sparse, self.hashmap = self._prep_data()
```

Figure 66 : *_prep_data()* called from the constructor

Model building and training

The constructor of the *Initializer* class can now build the KNN model and train it by passing the matrix just created : (Liao, 2018)

```
self.model = NearestNeighbors(
    n_neighbors= 20,
    algorithm= 'brute',
    metric= 'cosine')
self.model.fit(self.movie_user_mat_sparse)
```

Figure 67 : model building and training

The *NearestNeighbors* class of the Scikit-Learn library builds a KNN model. The first parameter set the default K neighbors to 20 (it can be override later), the second one sets the algorithm used to compute the nearest neighbors to *brute* (it will use a brute-force search) and the last one sets the metric to use for distance computation between

neighbors to *cosine* (it will use the cosine similarity to calculate the distance). (Scikit-Learn, 2019)

We use then the *fit* method of the class to train the model with the Movie-User matrix.

The model is now trained and it can be used to make predictions (recommendations). We saw before that in a IB-CF system, we can train the model offline and then save it in order to use it online after that. So we finally save it on the disk :

```
def _saveData(self):
    """
    Save data to the disk
    1. hashmap
    2. scipy movie-user sparse matrix
    3. trained model
    """

    #save hashmap
    np.save('app/recommender/hashmap.npy', self.hashmap)

    #save scipy parse matrix
    scipy.sparse.save_npz('app/recommender/matrix.npz', self.movie_user_mat_sparse)

    #save trained model
    dump(self.model, 'app/recommender/model.joblib')
```

Figure 68 : save the model in the backend

The method *_saveData()* of the *Initializer* class saves the model, the hashmap and the matrix into files in the backend. We will need later to use those three files to make recommendations.

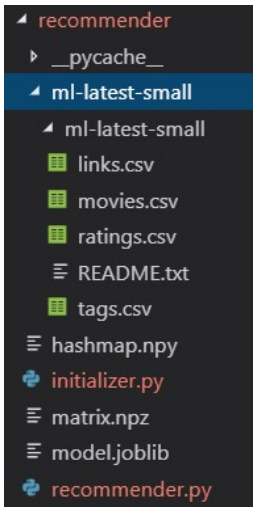

Figure 69 : *recommender* folder in the backend

The figure above shows the *recommender* folder in the backend. It contains the CSV, the *Initializer* class in the *initializer.py* file, the *hashmap.npy* file that contains the saved

hashmap, the *matrix.npz* file that contains the saved matrix, the *model.joblib* file that contains the trained and saved model and a *recommender.py* file. The *recommender.py* file contains the *Recommender* class that will use the built KNN model to make predictions as detailed in the next section.

The *Initializer* class can be called each time we want to update the model and train it with new data. It will replace the three saved files with new ones.

Predictions

The *Recommender* class is used to make recommendations for a given TMDB ID of a movie. We need first to load the hashmap, the matrix and the model from the files where they are saved :

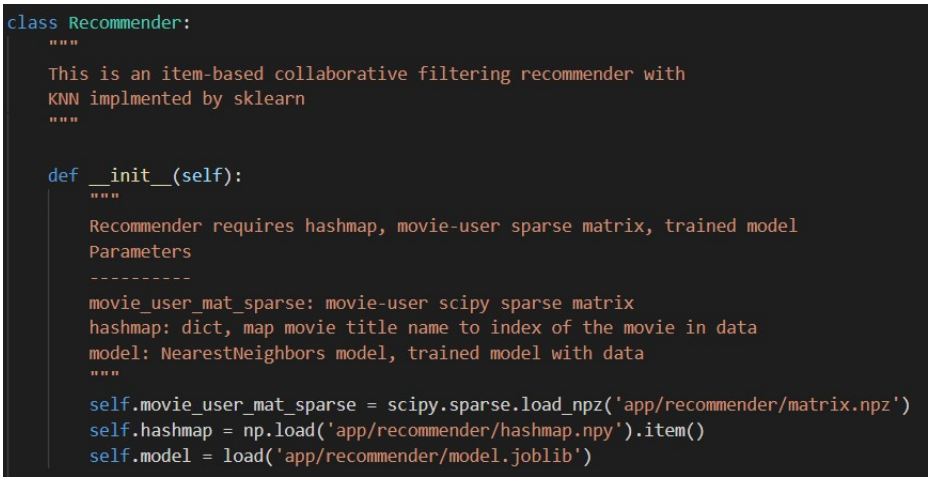

Figure 70 : constructor of the *Recommender* class

The figure above shows the constructor of the *Recommender* class that loads the three mentioned files.

Once the *Recommender* is instancied and the files are loaded, we can use it to make recommendations. The figure below shows the method *make_recommendations* of the class :

```python
def make_recommendations(self, inputTmdbId, n_recommendations):
    """
    make top n movie recommendations
    Parameters
    ----------
    inputTmdbId: int, tmdbId of user input movie
    n_recommendations: int, top n recommendations
    """
    # get recommendations
    indexes = self._inference(inputTmdbId, n_recommendations)
    # return results
    results = []
    reverse_hashmap = {v: k for k, v in self.hashmap.items()}
    for i in indexes[0]:
        results.append(reverse_hashmap[i])
    return results
```

Figure 71 : *make_recommendations* method

The method takes two arguments : the TMDB ID of the movie we want to search for similar movies and the number of similar movies that we want to search for. The method does the following things : (Liao, 2018)

1. It gets the similar movies by passing class the two arguments to the method *_inference* of the class
2. It declares an empty array *results*
3. It fills the array *results* with the TMDB IDs of similar movies. The TMDB IDs are retrieved by seeking inside the hashmap the TMDB ID corresponding to the ID of the movie in the Movie-User matrix.
4. It returns the array *results* (it contains the recommendations)

The figure below shows the *_inference* method :

```python
def _inference(self, inputTmdbId, n_recommendations):
    """
    return top n similar movie recommendations based on the movie tmdbId given by the user
    Parameters
    ----------
    inputTmdbId: int, tmdbId of user input movie
    n_recommendations: int, top n recommendations
    Return
    ------
    list of top n similar movie recommendations
    """
    # use the input movie tmdbId to get movie index in the hashmap (corresponding to the index in the movie-user matrix)
    movieIndex = self._findMovieIndex(inputTmdbId)
    # inference
    distances, indexes = self.model.kneighbors(
        self.movie_user_mat_sparse[movieIndex],
        n_neighbors=n_recommendations+1)
    # return recommendations
    return indexes
```

Figure 72 : *_inference* method

The method takes the two same arguments : the TMDB ID of the movie we want to search for similar movies and the number of similar movies that we want to search for. It does the following things : (Liao, 2018)

1. It gets the ID of the movie in the matrix by calling the method *_findMovieIndex* with the TMDB ID of the movie. The method *_findMovieIndex* uses the TMDB ID of a movie and the hashmap to get the movie ID in the Movie-User matrix.
2. It calls the method *kneighbors* of the model with two arguments : the ID of the movie in the matrix it just got and the number of similar movies we want. The method *kneighbors* finds the K-nearest neighbors of an item in the matrix (Scikit-Learn, 2019). In this case, it finds the 10 nearest similar movies of the given ID.
3. It returns the IDs of the similar movies.

Get recommendations

The model is now trained and the *Recommender* class is ready to be used in our API.

We can test it with Postman :

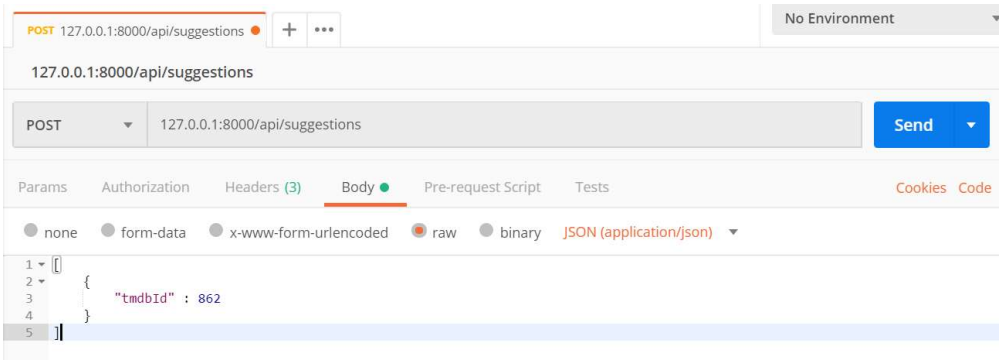

Figure 73 : Postman request to API

We call the API endpoint */api/suggestions* for the movie of TMDB ID 862. The results are :

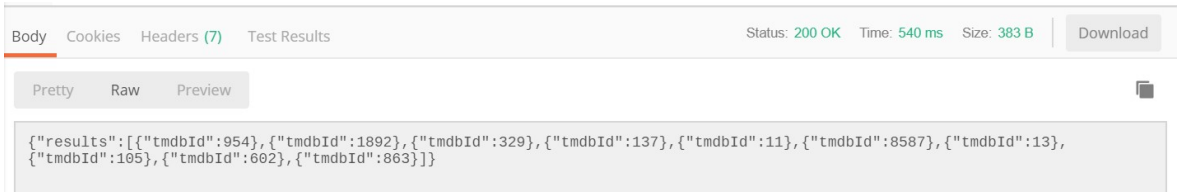

Figure 74 : Postman response

The backend sends back the TMDB IDs of the 10 most similar movies found by our KNN model.

Improving the model by training it with new data from users ratings

The KNN model is built with the fixed dataset from GroupLens. Once we have users on our application, they will like some movies and everything will be stored in our database.

In order to improve the ML model, we could append those new data to the dataset from GroupLens. We saw that the more data a ML model has, the greater the accuracy is. The idea is to periodically fetch the new data from our database, append them to the old dataset and train the model with the new dataset.

The code below sets up the process :

```
firebase = pyrebase.initialize_app(config)
db = firebase.database()

# load the hashmap
hashmap = np.load('app/recommender/hashmap.npy').item()

# clone the ratings file in order to keep a clean backup
path_ratings_src = 'app/recommender/ml-latest-small/ml-latest-small/ratings_base.csv'
path_ratings_dest = 'app/recommender/ml-latest-small/ml-latest-small/ratings.csv'
copyfile(path_ratings_src, path_ratings_dest)
```

Figure 75 : updater I

This code does the following things :

- it initializes the connection to Firebase Database and get the database in the variable *db*
- it loads the hashmap (previously introduced) because we need it in the next portion of code (cf below)
- it clones the *ratings.csv* file in order to keep a safe backup without our own data appended to it

Then we can append the new data :

```
# get all likes in the database
likes = db.child("likes").get()

# add the likes to the cloned ratings file
for like in likes.each():
    #get userId
    userId = like.key()
    for i in like.val():
        #get tmdbId
        tmdbId = like.val()[i]["idTmdb"]
        #get movieId
        movieId = findMovieIndex(tmdbId)
        # write into the file : add a new row to the CSV
        with open(path_ratings_dest,'a') as fd:
            fd.write(str(userId) + ',' + str(movieId) + ',5,' + str(time.time()) + '\n')
```

Figure 76 : updater II

This code does the following things :

- it gets all the *likes* from the db in the variable *likes*
- for each *like* it opens the *ratings.csv* file and add a new row with those information : the user's ID of the *like*, the ID of the liked movie, that corresponds to its TMDB ID in the Movie-User matrix (in order to achieve this we call the method *findMovieIndex*, previously introduced, that looks inside the hashmap to find the corresponding ID for a given TMDB ID), the rating 5 (on our application, a user can like or not a movie. If he does, we assume that the rating is 5/5) and the current timestamp

6 Results

Now that everything is implemented, we can evaluate and discuss the results. The objectives were to implement a ML algorithm and to provide a good-looking and user-friendly UI.

This section will focus on those two objectives and dicuss the results separately.

6.1 Usability of the application

We asked 4 different users to test the application and to give a feedback in a Excel file. We gave them the application and asked them to go on (without any help or question). The Excel file contains a few formulated questions asking to give a grade from 1 to 5 and a free space for commentaries or suggestions.

Here are the results :

Profile		A	B	C	D
	background	graphic designer formation and IT formation	IT formation	no IT background	no IT background
	gender	F	M	F	F
	age	27	25	49	22
	daily use of the web/mobile applications (from 1 to 5)	4	5	1	3
Usability					
	Do you understand the main goal of the application ? (from 1 to 5)	5	5	4	5
	Do you understand how it works ? (from 1 to 5)	5	5	2	3
	Do you know how you can search for movies ? (from 1 to 5)	5	5	5	5
	Do you know how you can get some information about a movie ? (from 1 to 5)	5	5	5	5
	Do you know how you can add a movie to your list of "likes" ? (from 1 to 5)	5	5	4	5
	Do you know where you can remove or see your "likes" ? (from 1 to 5)	5	5	3	4
	Do you know how you can finally get some recommendations ? (from 1 to 5)	5	5	3	4
	How do you globally evaluate the UI ? (from 1 to 5)	5	5	5	5
	Free commentaries / suggestions	introduction text is too long, as a user I don't care about the system behind. I will also prefer juste one verb for the first sentence:either search or select little title for the most popular movie	The name of the movie might be unclear sometimes. As a user, I want to know what is the complete name of the movie without clicking on it, and without having to read on the image. Easy to use because of the material	once I got the concept, the interface is very easy to use where can I see the movies ?	the concept is clear but we are dropped into the main application and it is not easy to know how to start to use it

Figure 77 : users' feedback

The users have been chosen because they have different backgrounds and they are people of different ages :

- The user A is a female, 27 years old and has a graphic designer and IT background. She estimates her daily use of the web or mobile applications by 4/5.

- The user B is a male, 25 years old and has a strong IT background. He estimates his daily use of the web or mobile applications by 5/5.
- The user C is a female, 49 years old and has no IT background. She estimates her daily use of the web or mobile applications by 1/5.
- The user D is a female, 22 years old and has no IT background. She estimates her daily use of the web or mobile applications by 3/5.

The first conclusion is that for people with an IT background, the goal of the application is very clear and the UI is very easy to use. They got immediately everthing and directly knew how to use the application in order to reach its goal. According to user B, the application is easy to use because it is done with the Google Material Design concept and it looks like every applications of Google or Android.

As a graphic designer, the user A suggested to add a title on the homepage to inform the users that the displayed movies are the last popular movies and not any recommendation yet. She also suggested to shorten the main text on the homepage because it is too long and the user does not need to know how it works in background.

The user B suggested to directly add the title of the movies on the cards. It is not user-friendly to have to read the posters of the movies or to click on a movie to get its title if we do not know the poster.

The users without any IT background easily understood the goal of the application but they had some difficulties at the beginning to get how it works and how to start to use it. They easily understood how to search for movies, like a movie and how to get information about a movie because it is obvious but they had difficulties to jump to the next step of the process (manage the *likes*, get recommendations). The user D said that the users are dropped into the application without any context or help at the beginning. But once they explored by themselves the application, the menu and the buttons, they finally got the concept and it was then easy to use due to the good UI.

The user C asked where she can watch the movies because she did not understand that the application just provides some recommendations and do not stream any content. This information is nowhere.

We will see how we can take those information into consideration in the conclusion and the final discussion of this thesis.

6.2 Functionnal results of the algorithm

The KNN algorithm is a lazy algorithm : it means that there is not a training phase and a testing phase to evaluate the accuracy. All the data are used in the training phase and the model is then ready to be used. (Navlani, 2018) So we can not evaluate the model by mathematically testing its accuracy.

We are going to test our algorithm by analyzing its behavior with real cases. Let's see some concrete examples.

We start by getting some recommendations for a popular movie : *Iron Man* :

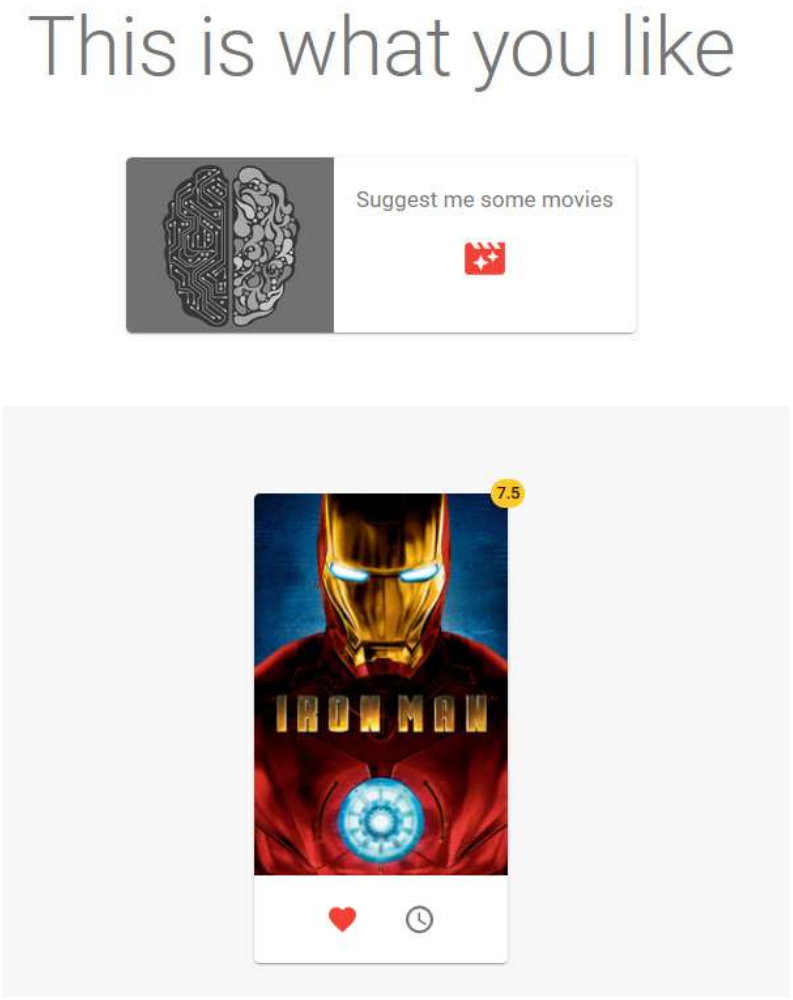

Figure 78 : Movies liked I

The recommendations are :

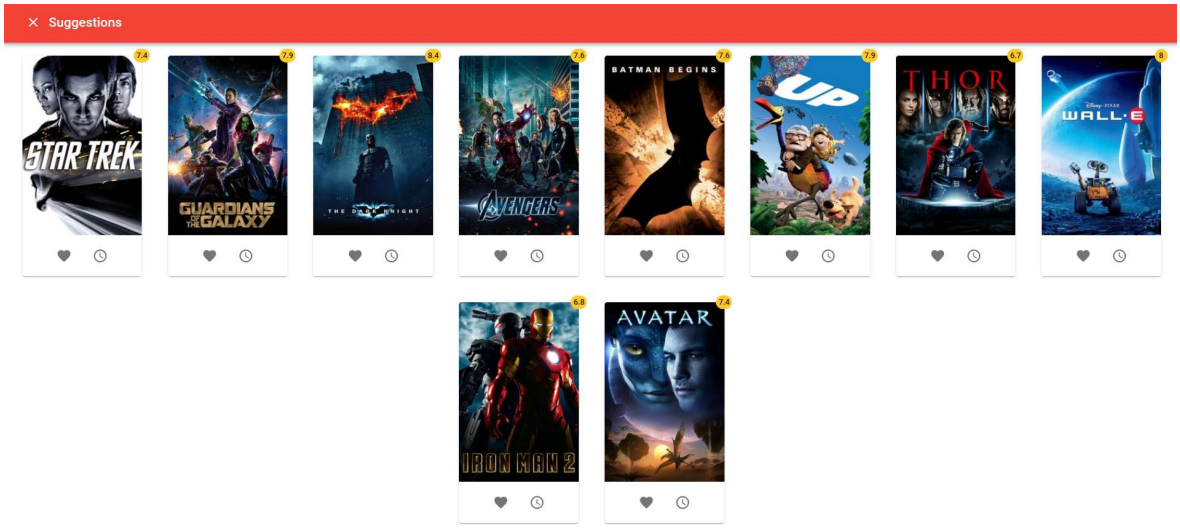

Figure 79 : recommendations I

The figure above shows the recommendations : *Star Trek, Guardians of the Galaxy, Batman The Dark Knight, Avengers, Batman Begins, Up, Thor, Wall-E, Iron Man 2, Avatar.*

It seems to work pretty good since *Iron Man* is a super-hero movie and a Marvel movie : the recommendations are some Marvel movies or other super-heroes movies. There is only *Wall-E* and *Up* that are not very relevant.

Let’s add now a new super-hero and Marvel movie to our list of *likes* :

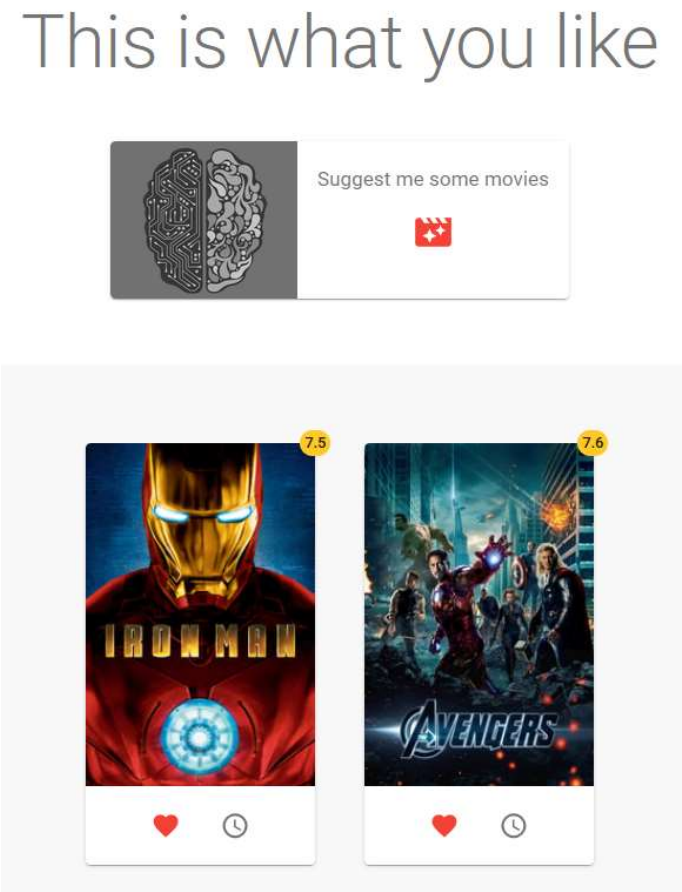

Figure 80 : movies liked II

The recommendations are now :

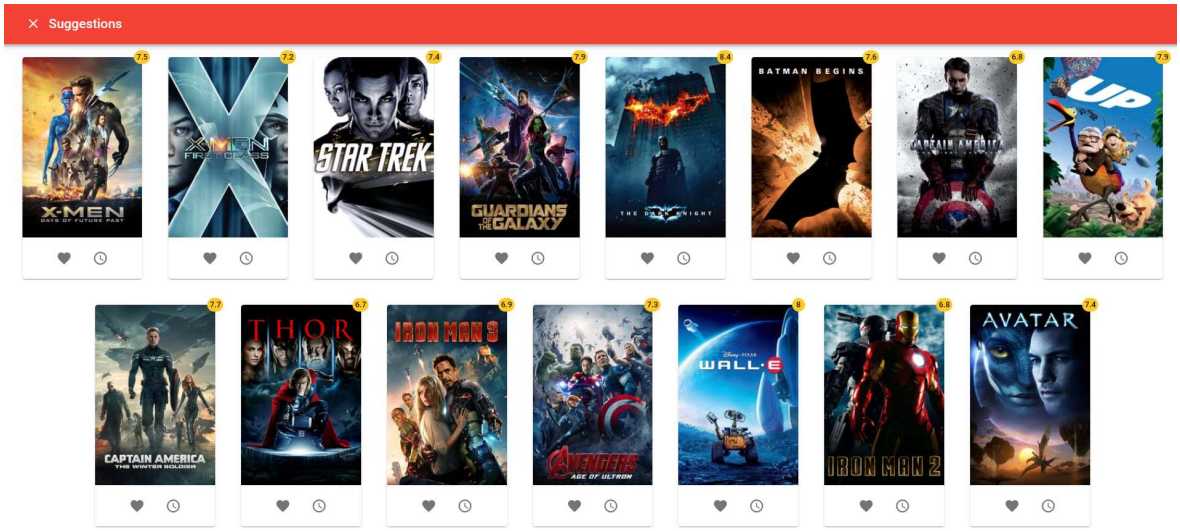

Figure 81 : recommendations II

The figure above shows the recommendations : *X-Men : Days of Future Past, X-Men : First Class, Star Trek, Guardians of the Galaxy, Batman : The Dark Knigt, Batman : Begins, Captain America : The First Avenger, Up, Captain America : The Winter Soldier, Iron Man 3, Avengers : Age of Ultron, Wall-E, Iron Man 2, Avatar.*

The algorithm kept the same recommendations as in the first example and added some other Marvel or super-heroes movies (*Captain America, X-Men*, …).

Let's test it now with an other popular movie : *Harry Potter and the Prisoner of Azkaban*. The recommendations are :

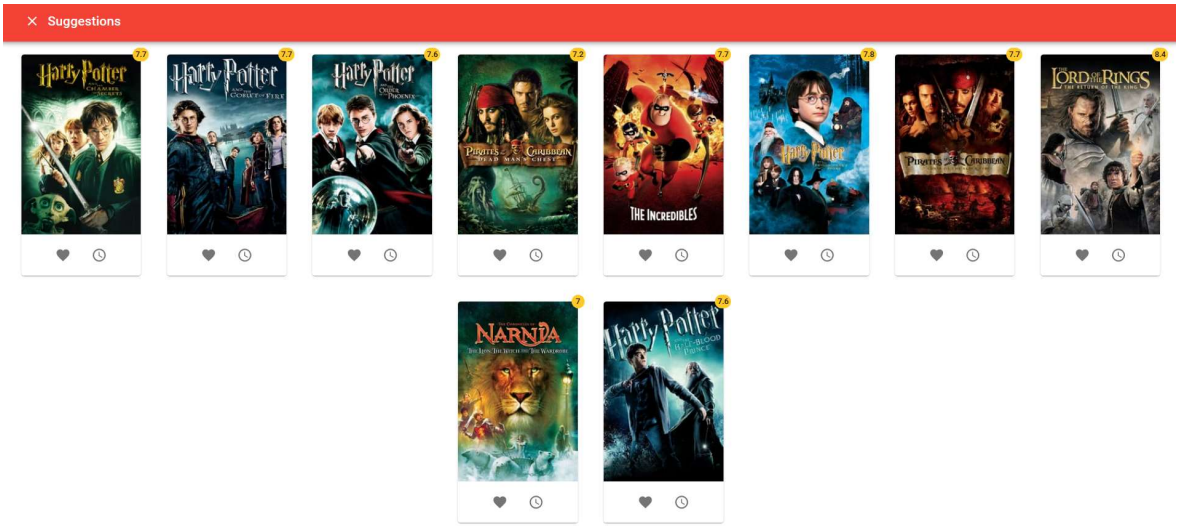

Figure 82 : recommendations III

The figure above shows the recommendations : *Harry Potter and the Chamber of Secrets, Harry Potter and the Goblet of Fire, Harry Potter and the Order of the Phoenix, Pirates of the Caribbean: Dead Man's Chest, The Incredibles, Harry Potter and the Philosopher's Stone, Pirates of the Caribbean: The Curse of the Black Pearl, The Lord of the Rings: The Return of the King, The Chronicles of Narnia: The Lion, the Witch and the Wardrobe, Harry Potter and the Half-Blood Prince.*

It also works pretty good : all the *Harry Potter* movies are recommended and some other fantasy movies liked by the kids or the teenagers are also recommended. There is only *The Incredibles* that is not relevant.

Now let's test it with a less popular movie but with a strongly identified theme : *The Hurt Locker*. This is a war movie, so we expect to get a lot of famous war movies as recommendations :

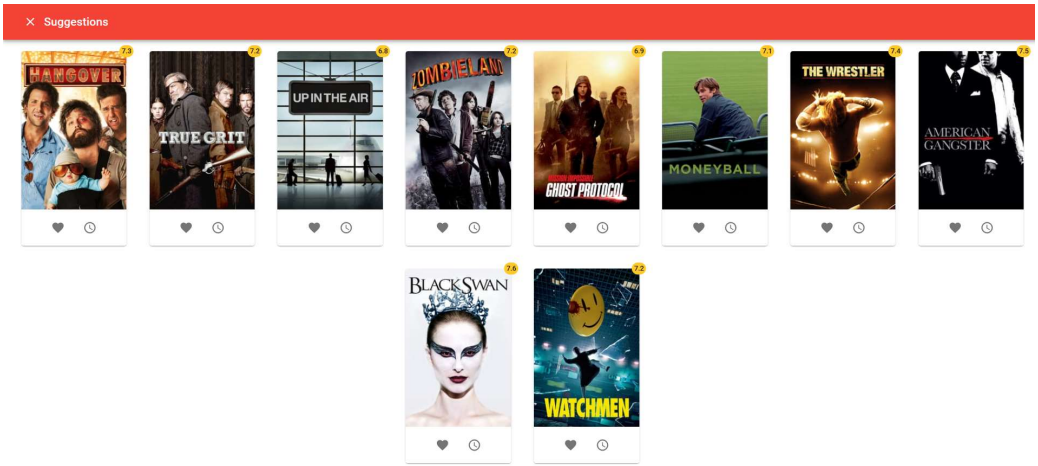

Figure 83 : recommendations IV

The figure above shows the recommendations. There is not any war movie. The recommendations are not relevant and even sometimes very distant from *The Hurt Locker*.

We will discuss it in the conclusion.

7 Conclusion and discussion

In this thesis, we have introduced the basics of the Machine Learning and we have studied and explained how it works and how it can be used in the case of a recommender system for movies. We have introduced the different ways it can be done and how the big companies deal with those systems.

We chose then to implement an Item-Based Collaborative Filtering system with a KNN algorithm in a project because it is one of the most powerful recommender system and it is trusted by many people that need to build such engines. The project had to provide everything in order to allow users to get good recommendations for every movie they want on a user-friendly interface.

We can now say that the objective of the UI is reached since the tests with real users are convincing. People with an IT background performed very well with the application but it was expected. The most significant result is that people without any IT background performed well with the UI. They take more time to handle the process and the UI but they finally managed to understand everything. So it is not a perfect result but it is quite good. We could go further to improve the lambda users' experience by adding a stepper at the first connection on the application with a virtual guide (pop-ups, avatar, …) explaining the goal of the application and showing the different successive steps in order to get the first recommendations. By adding that we could then remove the message on the homepage which does not seem to be relevant enough and clear enough to offer a perfect experience to everyone. We could finally improve the information about the movies by adding on each movie some links to Netflix, Amazon or some other streaming companies that offer this item.

Regarding the functionnal results and the algorithm, we can say that the objective is also reached since the recommendations are quite good in general. The recommendations are very good and very relevant for popular movies but they are less convincing for less popular movies. This is due to the Item-Based Collaborative Filtering system and the KNN algorithm. We saw that this system works with the users' data and the more data it has, the more accurate it is. Since the popular movies are watched by a lot of people and rated by a lot of people, the system gets lots of data about them so it performs very well with that kind of movies and on the other hand the less popular movies suffer from a lack of significant data since they are not watched by a lot of people. We could go further to improve the system by adding a content-based system and merging them into a hybrid sytem. The content-based system would balance the recommendations for less popular

movies since it would easily identify a war movie as a “war theme” movie by reading its synopsis for example and would recommend other movies that have the same theme. The Item-Based Collaborative Filtering system would meanwhile recommend movies for popular movies since it performs very well. This hybrid system could detect if a movie is popular or not by anlayzing the number of reviews or ratings and then call the right algorithm. Or it could also everytime merge the recommendations that come from the two systems and give the users some balanced results.

Regarding the self-learning, the thesis author learned what the Machine Learning concretely is, how it works in general and how it can be implemented in a movie recommender system. He learned the Python programming language since it is the most powerful language for Machine Learning and since he did not know anything about it at the beginning. He also learned how to develop a backend server with Python and how to integrate a ML model inside it. Finally he learned how to build a fully functionnal architecture which is able to run the full project application.

8 Table of Figures

9 References

Bado, B., 2018. *Adding Authentication To React Redux Firebase App.* [Online]
Available at: https://medium.com/quick-code/adding-authentication-to-react-redux-firebase-app-f0efcb1c519a
[Accessed 9 April 2019].

Bado, B., 2018. *How to Integrate React Redux and Firebase in 3 Simple Steps.* [Online]
Available at: https://medium.com/quick-code/how-to-integrate-react-redux-and-firebase-in-3-simple-steps-c44804a6af38
[Accessed 9 April 2019].

Bronshtein, A., 2017. *A Quick Introduction to K-Nearest Neighbors Algorithm.* [Online]
Available at: https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7
[Accessed 2 April 2019].

Castrounis, A., 2019. *Machine Learning: An In-Depth Guide - Overview, Goals, Learning Types, and Algorithms.* [Online]
Available at: https://www.innoarchitech.com/machine-learning-an-in-depth-non-technical-guide/
[Accessed 1 April 2019].

DataRobot, 2019. *Unsupervised Machine Learning.* [Online]
Available at: https://www.datarobot.com/wiki/unsupervised-machine-learning/
[Accessed 1 April 2019].

DRF, 2019. *Django Rest Framework.* [Online]
Available at: https://www.django-rest-framework.org/
[Accessed 8 April 2019].

Google, 2019. *Firebase Authentication.* [Online]
Available at: https://firebase.google.com/docs/auth/
[Accessed 8 April 2019].

Google, 2019. *Firebase Realtime Database.* [Online]
Available at: https://firebase.google.com/docs/database/
[Accessed 8 April 2019].

Google, 2019. *Plongée dans le ML : régression linéaire.* [Online]
Available at: https://developers.google.com/machine-learning/crash-course/descending-into-ml/linear-regression?hl=fr
[Accessed 2 April 2019].

Grimaldi, E., 2018. *How to build a content-based movie recommender system with Natural Language Processing.* [Online]
Available at: https://towardsdatascience.com/how-to-build-from-scratch-a-content-based-

movie-recommender-with-natural-language-processing-25ad400eb243
[Accessed 4 April 2019].
GroupLens, 2019. *MovieLens.* [Online]
Available at: https://grouplens.org/datasets/movielens/
[Accessed 8 April 2019].
G, Y., 2017. *The 7 Steps of Machine Learning.* [Online]
Available at: https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e
[Accessed 3 April 2019].
Jeevan, M., 2018. *How to evaluate a machine learning model- part 1.* [Online]
Available at: https://www.edvancer.in/how-to-evaluate-a-machine-learning-model-part-1/
[Accessed 3 April 2019].
Kurama, V., 2017. *Introduction To Machine Learning.* [Online]
Available at: https://towardsdatascience.com/introduction-to-machine-learning-db7c668822c4
[Accessed 1 April 2019].
Kurama, V., 2018. *Unsupervised Learning with Python.* [Online]
Available at: https://towardsdatascience.com/unsupervised-learning-with-python-173c51dc7f03
[Accessed 1 April 2019].
Liao, K., 2018. *Prototyping a Recommender System Step by Step Part 1: KNN Item-Based Collaborative Filtering.* [Online]
Available at: https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea
[Accessed 4 April 2019].
Lineberry, A. & Longo, C., 2018. *Creating a Hybrid Content-Collaborative Movie Recommender Using Deep Learning.* [Online]
Available at: https://towardsdatascience.com/creating-a-hybrid-content-collaborative-movie-recommender-using-deep-learning-cc8b431618af
[Accessed 4 April 2019].
Lohr, S., 2009. *Netflix Awards $1 Million Prize and Starts a New Contest.* [Online]
Available at: https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/
[Accessed 4 April 2019].
Ma, K., 2016. *Content-based Recommender System for Movie Website.* [Online]
Available at: http://www.diva-portal.org/smash/get/diva2:935353/FULLTEXT02.pdf
[Accessed 4 April 2019].
Marr, B., 2016. *The Top 10 AI And Machine Learning Use Cases Everyone Should Know About.* [Online]

Available at: https://www.forbes.com/sites/bernardmarr/2016/09/30/what-are-the-top-10-use-cases-for-machine-learning-and-ai/#1fa94e4494c9
[Accessed 2 April 2019].
Mishra, M., 2018. *Understanding Linear Regression in Machine Learning.* [Online]
Available at: https://medium.com/datadriveninvestor/understanding-linear-regression-in-machine-learning-643f577eba84
[Accessed 2 April 2019].
Morgan, L., 2015. *11 Cool Ways to Use Machine Learning.* [Online]
Available at: https://www.informationweek.com/strategic-cio/executive-insights-and-innovation/11-cool-ways-to-use-machine-learning/d/d-id/1323375
[Accessed 2 April 2019].
Navlani, A., 2018. *KNN Classification using Scikit-learn.* [Online]
Available at: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn
[Accessed 2 April 2019].
neo4j, 2019. *8.2. The Cosine Similarity algorithm.* [Online]
Available at: https://neo4j.com/docs/graph-algorithms/current/algorithms/similarity-cosine/
[Accessed 4 April 2019].
NumPy, 2019. *NumPy.* [Online]
Available at: http://www.numpy.org/
[Accessed 8 April 2019].
Pandas, 2019. *Python Data Analysis Library.* [Online]
Available at: http://pandas.pydata.org/
[Accessed 8 April 2019].
Pandit, N., 2018. *What is ReactJS and Why Should We Use It ?.* [Online]
Available at: https://www.c-sharpcorner.com/article/what-and-why-reactjs/
[Accessed 8 April 2019].
Pinela, C., 2017. *Content-Based Recommender Systems.* [Online]
Available at: https://medium.com/@cfpinela/content-based-recommender-systems-a68c2aee2235
[Accessed 4 April 2019].
Priy, S., 2019. *Clustering in Machine Learning.* [Online]
Available at: https://www.geeksforgeeks.org/clustering-in-machine-learning/
[Accessed 1 April 2019].
Puguet, J.-F., 2016. *The Most Popular Language For Machine Learning Is ....* [Online]
Available at: https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en
[Accessed 2 April 2019].

Redux, 2019. *Getting Started with Redux.* [Online]
Available at: https://redux.js.org/introduction/getting-started
[Accessed 9 April 2019].
Rouse, M. & Burns, E., 2018. *Machine Learning (ML).* [Online]
Available at: https://searchenterpriseai.techtarget.com/definition/machine-learning-ML
[Accessed 1 April 2019].
Scikit-Learn, 2019. *1.6. Nearest Neighbors.* [Online]
Available at: https://scikit-learn.org/stable/modules/neighbors.html
[Accessed 8 April 2019].
SciPy, 2019. *SciPy.* [Online]
Available at: https://www.scipy.org/
[Accessed 8 April 2019].
Shukla, S., 2019. *Regression and Classification | Supervised Machine Learning.* [Online]
Available at: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/
[Accessed 1 April 2019].
Singh, G., Jain, A. & Dhandhania, K., 2019. *TF-IDF: Vector representation of Text.* [Online]
Available at:
https://www.commonlounge.com/discussion/99e86c9c15bb4d23a30b111b23e7b7b1
[Accessed 4 April 2019].
Tagliaferri, L., 2017. *An Introduction to Machine Learning.* [Online]
Available at: https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning
[Accessed 1 April 2019].
TMDB, 2019. *API Overview.* [Online]
Available at: https://www.themoviedb.org/documentation/api
[Accessed 8 April 2019].
Trevino, A., 2016. *Introduction to K-means Clustering.* [Online]
Available at: https://www.datascience.com/blog/k-means-clustering
[Accessed 2 April 2019].
Valchanov, I., 2018. *Machine Learning: An Overview.* [Online]
Available at: https://www.datascience.com/blog/machine-learning-overview
[Accessed 1 April 2019].
Warcholinski, M., 2019. *10 Famous Apps Using ReactJS Nowadays.* [Online]
Available at: https://brainhub.eu/blog/10-famous-apps-using-reactjs-nowadays/
[Accessed 8 April 2019].

王斌, 2018. *Comparison of User-Based and Item-Based Collaborative Filtering.* [Online]

Available at: https://medium.com/@wwwbbb8510/comparison-of-user-based-and-item-based-collaborative-filtering-f58a1c8a3f1d
[Accessed 4 April 2019].