CONTENT-BASED MOVIE RECOMMENDER

MASTER'S DISSERTATION

Author: José Mario Fernández González

FEBRUARY, 2022

Index

Introduction

The final task of Master in Data Science at Kschool is to make a Data Science project about one topic of multiple subjects we learnt during the academic year in order to achieve the goal of master Data Science skills, the chosen one in this Master's Dissertation was a *Content-Based Movie Recommender.*

This project combines NLP with a recommendation engine to build an alternative engine away from typical recommender engines which are based in opinions or rating, in fact this recommendation is based on "content", specifically this is a movie recommender, so we want content like description, genres, actors, etc.

The project is completely done in python, all code is in a Jupyter Notebook which include the reading of the data to an example of the output. The data and the notebook can be found at:

https://github.com/marioferg7/Content-Based-Movie-Recommender

The project needs a high-performance machine to run all the lines, it was used a machine with 6vCPU and 64 GB of memory provided by IBM, so it's highly recommended a high-performance computer or cloud storage with same specifications or better.

Recommender Engine

First of all, we have to talk about how this engine was built and how it works, it's important to describe some theoretical concepts about it so we can understand how a recommendation is given at evaluate how good is the recommender because we can't evaluate the model with numbers or plots because the output given is subjective.

The engine was built following a method which tokenize every word from a text and return a matrix with information of each word of every movie in the dataset. This matrix is called TF-IDF matrix and is defined as:

$$\text{tfidf}_{i,j} = \text{tf}_{i,j} \times \log\left(\frac{N}{\text{df}_i}\right)$$

$\text{tf}_{i,j}$ = total number of occurences of i in j

df_i = total number of documents (speeches) containing i

N = total number of documents (speeches)

This formula is too easy to calculate if we have a few words and a few documents, for example, if we have 2 phrases like:

- The white dog is white
- The white cat is eating

TF is:

The	white	dog	cat	is	eating
1	2	1	0	1	0
1	1	0	1	1	1

IDF is:

The	white	dog	cat	is	eating
log(2/2)	log(2/3)	log(2/1)	log(2/1)	log(2/2)	log(2/1)
Log(2/2)	log(2/3)	log(2/1)	log(2/1)	log(2/2)	log(2/1)

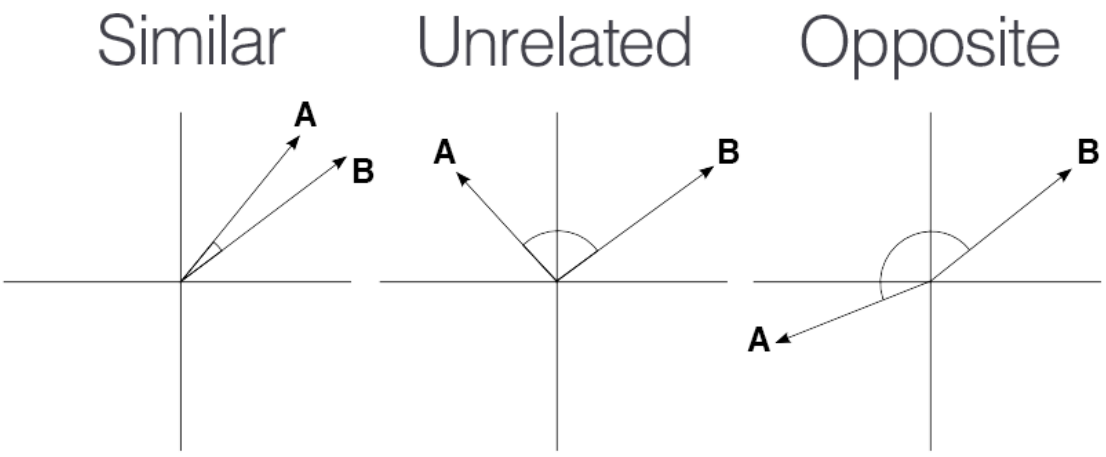
So TF/IDF is $TF_{i,j}$*$IDF_{i,j}$:

The	white	dog	cat	is	eating
0	-0.3520	0.3010	0	0	0
0	-0.1760	0	0.3010	0	0.3010

The main point of this method is to give a number of an importance of each word and it not mean that higher the number, higher the importance, this method is about how similar are the elements in the same column, because we will go to next step that is compare every row with cosine similarity which is defined as:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

This is a method to compare each row with its cosine we can see clearly in the next image:



For example, in the TFIDF example the cosine between sentence 1 and 2 is 0.29 which means they are not similar (cos = 1), but not completely unrelated (cos = 0), the cosine can take values from -1 to 1.

The problem is, how we can use this method to recommend similar films? First of all, we need data able to use with this method, content data like a description of a film can fit with this method, but I amplified this method to use with genres or main actors, tokenizing each name like a word and find the most similar movies sorting the cosines between them.

Data Information

The data was obtained from:

https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system/data

This project started with another dataset which contains information about 1000 movies, however this is not enough to build a good recommender system because the lack of movies and information about them makes a recommender engine worse, so this dataset which has information about 46628 movies was proposed to improve how many options the engine has in the pool to get a good recommendation.

This dataset has 19 columns of information about each film, but only considered as content the following ones:

- **Adult:** binary variable, if "False" it means the film is recommended to people under adult age, and "True" is the opposite.
- **Genres**: list of genres each movie is labeled, it has multiple choices, we found a total of 22 different genres and up to 9 genres in one movie.
- **Title**: the name of the film.
- **Overview**: A detailed description of the film.
- **Release date**: the day, month and year of the launching date.
- **Keywords**: Some words which contains important information.
- **Actor names**: 3 main actors of the movie.
- **Director name.**

Data Handling

When the recommender was built with the 1000 rows dataset I didn't find any issue taking every content of the dataset, but when I put the "movies_final" with 46628 rows it has a serious performance issue, because we have to handle a TFIDF matrix with shape around 100000 or more columns, so I had to remove multiple rows with lack of information.

The final number of films of the dataset is 33314, probably is a little number compared to the amount of film produced in the last 140 years, nevertheless the main goal is to get a recommendation and we lack in resources to handle a dataset with millions of movies.

Preprocess task is to reduce the amount of content that is considered irrelevant, so the list of preprocessing task was deleting irrelevant rows, tokenize the data and cleaning irrelevant words.

Note: this task is better described in the section 1.2 and 1.3 of the notebook.

Content features treatment

Now when have tokenized every single content column and make a list of each one in each film, we have to build a TF-IDF matrix with the whole content, but we didn't do that for 2 main reasons.

First is if we make transformations of a matrix with the order of 100k+ columns we will not be able to go ahead from this point because we need large amount of memory which I am not able to access so we have to separate each content and make its transformations separately to join each TF-IDF matrix later.

Second is if we have independent matrices, we can see that they have a big difference of shape, which means that cosine similarity is biased in the matrices with more columns, as we can see in the next images:

```
len(fd_nam)
```

```
54656
```

```
len(fd_gk)
```

```
22
```

First image is the number of different names the "33314 movies" dataset has counting actor names and director names and in the second image is the number of different genres in the whole dataset. So, it means that TF-IDF matrix has the same number of columns of these lengths.

So, we can see that names TF-IDF matrix is 2484 times larger than genres matrix, and if two TF-IDF matrices are joined it means that the matrix with more columns has more influence in the final result in the order of how many dimensions it has more than the other, in this example the names of directors and actors has 2484 times more influence than genres, if we use our experience in watching movies, probably a genre is more influential than the names.

We have to change the shapes of the matrices and it was proposed to reduce the number of words of big matrices to balance the results and reduce the amount of memory needed (54656 columns and 33314 rows is a gigantic matrix so we have to reduce it.

So, we proposed some adjustments for each content variable to get a good way of a correct number of columns and adjust the wight using a common sense (and not mathematically) to get a good TF-IDF matrix.

Adult

In this variable we found that there was an unbalanced class, we found that:

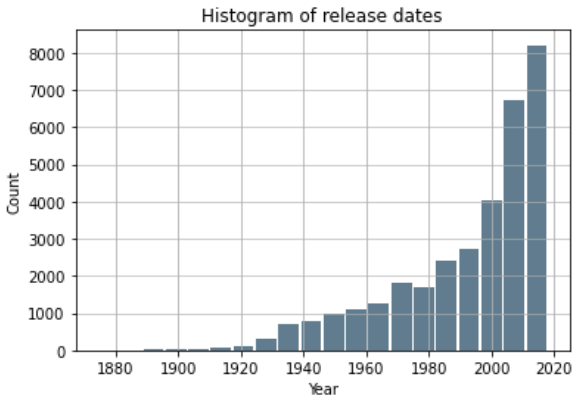
```
df.groupby('adult').size()
```

```
adult
0    33109
1        5
```

Only 5 movies were labeled as adult films, so we have to remove this variable because it has no use to add it to the recommendations.

Release date

This variable is so relevant because we found that there are films from over a 100 years ago:



To solve this, we proposed to make intervals of time, these intervals are so hard to make properly because there is too much to consider to group a film with another and separate films with the difference of months in release date, so it was proposed a quartile separation and label each interval from 0 to 3. The years of boundaries were:

```
df.year.quantile([0.25,0.5,0.75])
```

```
0.25    1979.0
0.50    2001.0
0.75    2010.0
```

Now we can see that probably is an acceptable separation of the years.

Genres & Names

As we said before, we have of columns in genre TF-IDF matrix, so we have to reply the TF-IDF matrix multiple times to solve this issue.

For names and overview, we have the opposite issue, we have to reduce them and ovoid to lose to much information, so we proposed first a filter.

For names we proposed that if a name appear less than “X” times it will be removed from content lists. I tried with multiple number with I give priority to reduce the biggest number of names to reduce the number of columns of TF-IDF matrix, the number I proposed was 5 after many tests and it leave me with 6291 names that appear in at least 5 films, so we reduced the amount of TF-IDF matrix columns from 54656 to 6291 and reduced the irrelevant information too.

Overview

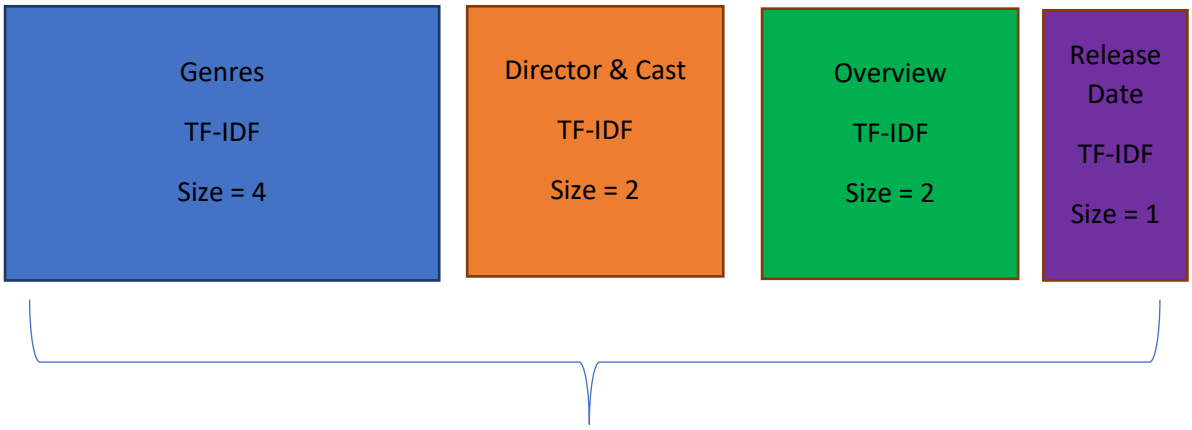
The last TF-IDF matrix is from the descriptions, first of all we have to say that is the more complex variable of all of the dataset because for tokenization we had to apply multiple NLP methods to clean a description to make a list of each description about that. After that we considered the same method used in Names, we deleted the words that appeared less than “X” times and more than “Y” times, that’s why we have more than 100000 different words in each description which means that TF-IDF is impossible to handle in memory, so we decided to apply filters to low appearance for the same reasons said in the last section and high appearance, because we don’t want to add column which has many similar movies, we want to make a recommendation based on small group.

Note: The method for reduction of names and overview was not mathematically tested its efficiency, but I did to optimize the amount of memory needed, because I broke the kernel multiple trying to make transformations in large matrices.

Building TF-IDF Matrix & Recommender

Next step is to join all matrices developed in “Content features treatment”, as we said before we have to let them a good number of columns to give the correct weight to each feature.

First of all, we set that we required 2 times the number of columns in genres than names and overview and 4 times the number of columns of release date, this was proposed by the author and was not based on any special method, only in his experience in watching movies.



Final TF-IDF Matrix

In the test we find that overview and names have a big shape to process the final cosine similarity so I had to reduce the columns, for that I proposed a PCA to reduce the amount of columns with cost of a % of information (this is described better at “Reduce the data” section) and found that at 80% of information I could reduce the matrices to 2900 columns in the case of names of actors and directors and 3050 with same information reduction to overview.

Now we have a TF-IDF matrix with 22 columns in genres and a column of labels in release date, so we have to reply their column multiples to get the desired shape, in the case of genres I replied it 290 times and got a shape of 6380 (It’s not necessary to get exactly 2 times the shape of the other matrices). And for release date I just replied the column 1500 times to get the desired shape. The final matrix will be the join of 4 matrices of 6480 columns of genres (size 4), 2900 columns from names matrix (size 2), 3050 columns from overview matrix (size = 2) and 1500 from date (size 1) makes a total of 13830 columns.

With the final TF-IDF matrix, we have to build cosine matrix, and build a recommender function that provide us a recommendation based on cosine similarity matrix. This recommender function was defined in section 4 “Build & Run the Recommender” and we describe there every step we took to get a list of 15 recommendations.

This process can be summarized in these steps:

- Select multiple films watched by 1 user (user feed) and take their cosines versus the rest of the films.

- Take 3 most common genres and then weigh the cosines of recommendation if a movie match with 0, 1, 2 or 3 genres, with more genres, more relevant is recommendation.

- Then we take the mean cosine of every film recommended and then sort it descending, taking the highest 15 values and their 15 names.

Note: We can see this process in the function “user_recomendations” from section 4.

The reason of this recommendation is we want to make an innovative method from TFIDF recommenders, we want to make a recommendation from multiple movies, but if we have multiple films we have to combine every score, so I though the mean value was a good option, however mean value is not good enough because some movies probably has high scores and other has low scores, so it’s the problem that 1 movie with high resemblance with the top recommendation can distort the recommendation, so we add a weigh by top genres watched and add to the mean. So, it will reduce the bias from these examples.

Results

Then we have to see how has it performed the recommendations. We will see the example from the notebook:

```
usuario_1 = ['i am legend', 'wonder women','waterloo bridge']
```

```
recommendation_1 = user_recommendations(usuario_1)
recommendation_1
```

```
3293            phantasm ii
3624           deepstar six
7687     dead end drive-in
2214              existenz
18007          full eclipse
4049              def-con 4
1024                 aliens
3929               watchers
16997            brain dead
3060             the hidden
5989          death machine
3873              dead heat
2009          godzilla 1985
3924              they live
```

It's hard to evaluate if it's a good recommendation, if it was only one film it will be easier to compare, but a common user normally had seen multiple films, so we want his feed, so they will be compared with the data from movies dataset:

	genres	original_title	overview	release_date	director_name	cast	names	year	date_label	descripcion
10306	[drama, horror, action, thriller, science, fic...	i am legend	[robert, nevill, scientist, unabl, stop, sprea...	2007-12-14	[FrancisLawrence]	[WillSmith, AliceBraga, CharlieTahan]	[WillSmith, AliceBraga, FrancisLawrence]	2007.0	2.0	[last, york, immun, man, spread, survivor, inc...
30770	[action, horror, science, fiction, thriller]	wonder women	[dr, tsu, brilliant, surgeon, exot, island, co...	1973-04-25	[RobertVincentO'Neill]	[NancyKwan, RossHagen, MariaDeAragon]	[NancyKwan]	1973.0	0.0	[insur, exot, transplant, sell, becom, involv,...
8474	[drama, romance]	waterloo bridge	[eve, world, war, ii, british, offic, revisit,...	1940-05-17	[MervynLeRoy]	[VivienLeigh, RobertTaylor, LucileWatson]	[LucileWatson, MervynLeRoy, VivienLeigh, Rober...	1940.0	0.0	[stay, front, noth, action, fall, hope, famili...

Now we will see the same example in top 6 recommended movies:

	genres	original_title	overview	release_date	director_name	cast	names	year	date_label	descripcion
3293	[action, horror, science, fiction, thriller]	phantasm ii	[mike, releas, psychiatr, hospit, team, old, p...	1988-07-08	[DonCoscarelli]	[JamesLeGros, ReggieBannister, AngusScrimm]	[AngusScrimm, JamesLeGros, DonCoscarelli]	1988.0	1.0	[hospit, man, becom, mike, reggi, tall, must, ...
3624	[action, horror, thriller, science, fiction]	deepstar six	[crew, experiment, underwat, nuclear, base, fo...	1989-01-13	[SeanS.Cunningham]	[TaureanBlacque, NancyEverhard, GregEvigan]	[SeanS.Cunningham, GregEvigan]	1989.0	1.0	[underwat, live, creatur, base, experiment, th...
7687	[action, drama, horror, science, fiction, thri...	dead end drive-in	[near, futur, drive, theatr, turn, concentr, c...	1986-08-13	[BrianTrenchard-Smith]	[NedManning, NatalieMcCurry, PeterWhitford]	[BrianTrenchard-Smith]	1986.0	1.0	[carmen, live, better, realli, camp, place, wa...
2214	[action, thriller, science, fiction, horror]	existenz	[game, design, run, assassin, must, play, late...	1999-04-14	[DavidCronenberg]	[JenniferJasonLeigh, JudeLaw, IanHolm]	[DavidCronenberg, IanHolm, JenniferJasonLeigh,...	1999.0	1.0	[design, assassin, latest, must, determin, dam...
18007	[horror, action, thriller, science, fiction]	full eclipse	[la, polic, depart, special, team, offic, tale...	1993-11-27	[AnthonyHickox]	[MarioVanPeebles, PatsyKensit, BrucePayne]	[MarioVanPeebles, AnthonyHickox, BrucePayne, P...	1993.0	1.0	[depart, track, special, unconvent, time, crim...
4049	[horror, science, fiction, thriller, action]	def-con 4	[two, men, woman, circl, globe, satellit, arm,...	1985-03-15	[PaulDonovanDigbyCook]	[LenoreZann, MauryChaykin, KateLynch]	[MauryChaykin]	1985.0	1.0	[stay, remain, later, war, man, survivor, kill...

As we see their genres match very well, and the dates fit in about its mean, only we don’t see any match in names, because the influence of genres is too high. So, we can’t evaluate properly unless we see these 18 movies so we can conclude that is a good recommender superficially.