Universitat Oberta de Catalunya (UOC)

Master's Degree in Data Science

FINAL MASTER THESIS

Area: Predictive Models

Predictions of new graph relationships

The Movie Database dataset

Author: Jose Luis Dolz Fernández

Tutor: Elisenda Bonet Carne

Professor: Jerónimo Hernández González

Barcelona, January 7, 2020

Master Thesis profile

Thesis title:	Predictions of new graph relationships: The Movie Database dataset
Author's name:	Jose-Luis Dolz Fernández
Professor:	Jerónimo Hernández González
Delivery date:	08/01/2020
Master's degree:	Data Science
Master Thesis area:	Predictive models
Thesis language:	English
Keywords	Graph analysis, Relationship prediction, Movies

“It’s still magic even if you know how it’s done.”

“Real stupidity beats artificial intelligence every time.”

– *Sir Terry Pratchett*

Acknowledgements

To Jerónimo,
my professor,
for giving me support
and being so sympathetic.
Lots of thanks for encouraging me
so hard and being so helpful.

To my parents,
for being always there,
in good and bad moments.
I couldn't be here
without your help.
You have given me more
than I could ever give you back.

To Beatriz, that in this semester
has done such a crazy thing
as getting married with me.
Thanks for your strength
and your willingness to make a better world.
I promise you this is my last course
and will spend the rest of my time with you.
Love you.

Abstract

Thanks to the huge amount of data that is collected nowadays, models can be created to make all kinds of predictions. Graphs are a specific type of model that can connect this data through relationships and predict new ones. A clear example is the suggestions of new people to connect with in social networks.

In this project, the information contained in The Movie Database of almost 5000 films from 1916 to 2017 is used to make a graph model and to predict brand new relationships: which actors will work together, who will be the director of a new blockbuster, etc. These new predictions are created by using machine learning over the relationships. The results obtained with best prediction algorithm used show an accuracy of 60%. Hence, further work is needed to tweak features extraction out from the graph model to improve the precision of these relationship predictions.

Keywords: Final Master Thesis, Graphs analysis, Relationships, PageRank, Path Ranking Algorithm, Prediction, Machine Learning, Python, Networkx, Movies, Actors, Directors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Proposal description and justification

Data can be modelled in several ways. The most known type is the table. Every table represents an entity: persons, cars, buildings, etc. Every tuple (*row*) represents an existing fact or object of that entity with its own attributes (*columns*). The problem with this kind of model is that it cannot relate tuples in the same tables unless new columns are created for every type of relationship.

This problem can be solved by using **graphs**. Every entity object or fact can be represented as a **node**. Then, every object is related with its own attributes –that are also nodes– by using **labelled edges**. Additionally, pairs of objects can be linked together with these edges labelling their relationship. Creating this model in a computing environment allows to query these relationships but also predict future links.

One such example of predicting new relationships is the suggestions of new users to follow in the social networks: depending on the people a user is connected with, the system will suggest a new user that is followed by those connected to the user. The more connections a user has, the bigger his relevance. In addition, the more relevant his followers are, the more his own relevance will increase. This used to be also the behavior of Google Page Rank when querying for information (Page et al., 1999).

With this in mind, almost every knowledge database can be converted into a graph –or directed graph– model to predict future relations. The purpose of this work is to transform The Movie Database dataset (TMDb, 2017) into a labelled graph model. After that, a feature extraction will be performed based on the relevance of every node and the amount of connections it receives. In a second approach, a path ranking algorithm will be used to find relevant ways to reach targeted nodes to execute new edges predictions. Finally, tests will be made by removing existing edges between entities and predict them.

There are a lot of questions around the film's ecosystem: Who is going to be the next big star? Will a producer get profit if investing in a specific genre? Which actors should be casted to create the next big hit? Are there specific words for a tagline that will attract more people to theaters? This work tries to answer all the questions cinema studios ask.

1.2 Personal motivation

For the great majority of people, it's difficult to find a balance between their dreamed job and their actual job. Seventeen years ago, I was sitting on a college classroom, totally bored, and wondering if I wanted to spend the rest of my life in front of a computer. I decided to quit from computer science degree and start a new career in dramatic art.

After completing four years of career, I kept on attending several courses and seminars: verse theater, casting preparation, voice dubbing, etc. Never been as happy as on a stage or in front of a camera. The problem was that no professional job –also known as paid job– was given to me. After some years trying my best on attending lots of castings and working as a logistics and warehouse specialist, I returned to the computer career through online university. Well, it was the best job of my non-dreamed jobs. Moreover, the offers for professionals in this sector kept growing.

But something unexpected happened by the end of the degree: while doing my specialization itinerary, I discovered data mining and machine learning. I learnt how to discover people behavior patterns based on their data as well as to predict them! I was so excited that when I coursed the Statistics subject again –that I repeated it three times at college– I passed it with honors. That love for the data and human behavior patterns took me to this Master's degree.

However, there was still a question to answer. How could my two professional worlds –actor and computer engineer– meet each other? The answer is in this Master thesis: using a huge amount of data to predict new professional relationships in cinema. Furthermore, this final work gives me the opportunity not only to show what I have learnt in the Master's degree but to learn new things as in any other subject.

1.3 Objectives definition

1.3.1 Main objectives

- Understand the behavior and problematic of graphs.
- Realize how to split the dataset between important data and useless data for the problem.
- Convert the dataset into a graph model.
- Extract important feature relationships from the model.
- Make predictions based on feature extraction.
- Tune machine learning model to improve precision.
- Optionally, create data and results visualizations.

1.3.2 Secondary objectives

- Improve knowledge and coding in Python language.
- Learn and master scientific writing in LaTeX.
- Improve English skills.

1.4 Methodology description

First of all, it is important to understand the context and objectives. This project starts from a known dataset, builds a labelled graph model and, from that point, tries to make predictions. At first sight, the methodology to reach the targets seems to be incremental. This means that a stage cannot be started since the previous one is finished. This work can be divided in the next phases:

Data: gathering, discovering and cleansing

↓

Modelling

↓

Machine Learning: prediction

↓

Visualization

Nevertheless, when talking about Data Science projects, there is an extra stage between *Machine Learning* and *Visualization* phases: the model tuning. This means that the model parameters are tweaked in order to improve model's precision and accuracy. So instead of following an incremental methodology, this project would perform an iterative one. Due to lack of time, the *Visualization* stage is optional in this work.

Apart from this, a comprehensive research will be done in the State of Art section below. As mentioned in section 1.1, nodes and relationships relevance must be taken into account for predicting new possible links in the graph. There is a lot of literature that can be related with this topic. In consequence, choosing and understanding the right papers, articles and books will be part of the research methodology.

1.5 Planning

These are the different stages for the development of this work and can be also followed at Figure 1.1:

- **Work definition and planning**: from September 18th to September 29th. The first stage of the project introduces the project topic. The work scope, methodology and interests for the general public are also featured in this phase. Additionally, the author personal motivation is presented as well.

- **State of Art**: from September 30th till October 20th. In this stage, a comprehensive research will be done to get the actual level achieved in relations prediction on labelled graphs and works on the Movies dataset. Here, algorithms and researches close to this project will be reviewed. Hence, almost the whole bibliography will be collected at this point.

- **Design and implementation**: from October 21st to December 21st. This stage is the core of the whole project. It is divided in three sub-stages as follows:
  - **Data analysis and cleansing**: two weeks. An exhaustive analysis of TMDb dataset will be performed in this stage. Then, only useful attributes will remain in the dataset. Next, a data cleansing will be made to get rid of unusual characters, wrong typos and possible outliers. Ultimately, the dataset will be split in two: the training set and the test set.
  - **Model building**: two weeks. A graph model will be built with the training set in this stage. This model will be tested with results-known queries to check that

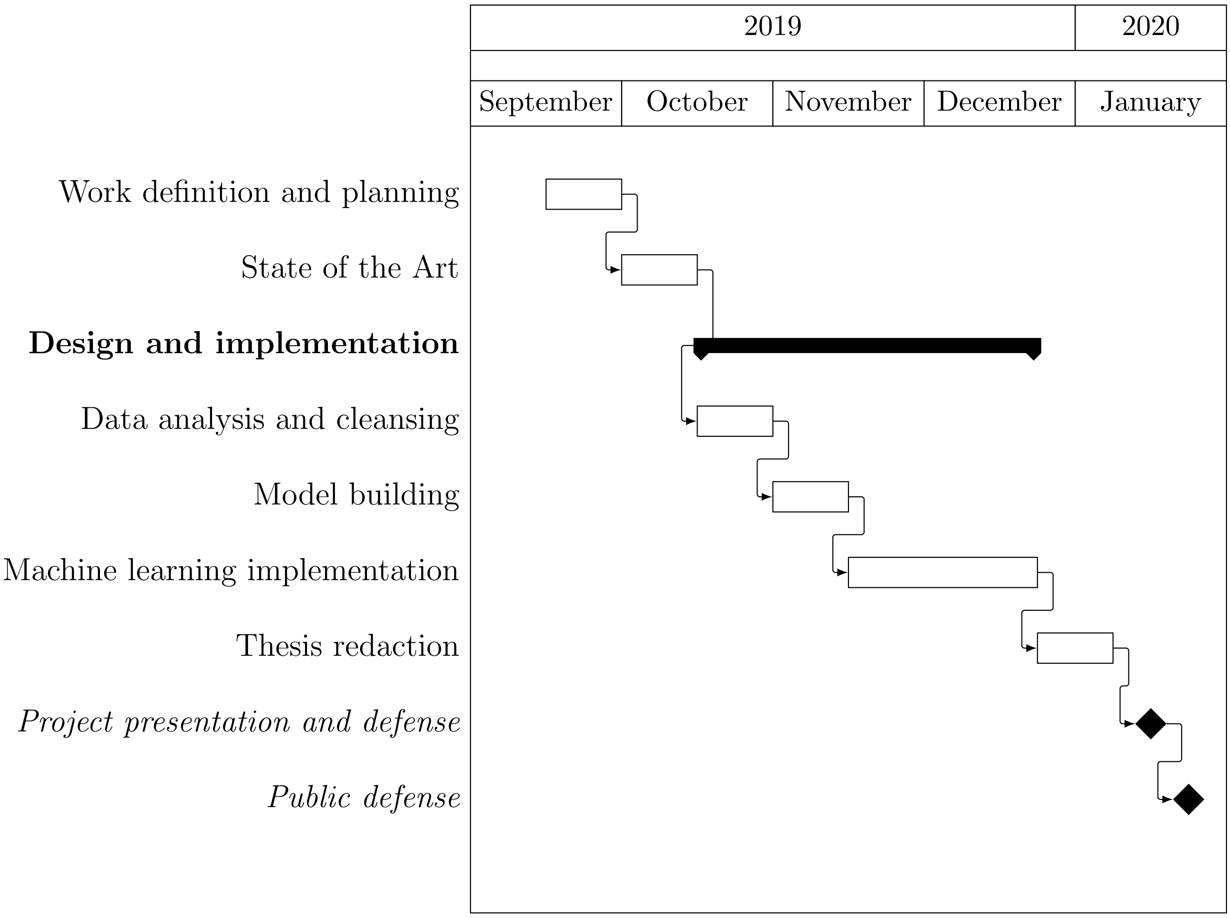


Figure 1.1: Gantt chart of Work planning

  labelled edges point to the correct nodes. In other words, model integrity, consistence and correctness will be proved. If there is time left, a graph visualization will be made so the model can be explored.

  - **Machine learning implementation**: five weeks. The biggest time-consuming stage is dedicated to creating the core code. By this time, the feature extraction will be programmed as well as the prediction system. Additionally, the test dataset will be used with several queries to get the Mean Average Precision (MAP) of the model.

- **Thesis redaction**: from December 22nd till January 8th. This is where all the previous deliveries meet. Every previous writing must be reviewed and corrected. In addition, new sections must be written: experiments and results, conclusions and future work. This is the documentation that will be uploaded to the university repository and available for everyone.

- **Project presentation and defense**: from January 9th to January 14th. A presentation for general public, in a unique language, must be delivered. It is mandatory to create a clip for the presentation up to 20 minutes long and up to 30 slides.

- **Public defense**: any time from January 15th till January 22nd. During this week, the Master Thesis Tribunal can ask questions to the project author and must be answered in the next 24 hours.

Chapter 2

State of the Art

This chapter outlines the different works and investigations related to this project topic so far. An exhaustive review of literature has been performed to find common algorithms and their improvements, computational problems and their solutions. This State of the Art has been divided into three different sections: nodes importance, where the entity relevance algorithms are presented; nodes and links prediction section, entities and relations forecasting based on works of the previous section; and, finally, a section with related prediction works and projects bases on The Movie database dataset.

2.1 Node importance

Investigations about ranking the relevance of an entity inside a network have increased in the last two decades due to Internet search engines. These crawlers need to offer the exact matches based on user queries as fast as possible.

Google is the most famous search engine and his PageRank algorithm (Page et al., 1999) is the starting point of later investigations. In this method, the rank of a page is based on the number of other pages backlinks and those pages relevance. In other words, a page that is pointed by hundreds of irrelevant pages would have the same rank as a page with only one backlink of a high rank page. This PageRank builds a Markov chain with a transition probability matrix –also known as adjacency matrix– executing random walks over the graph that represents the network. However, there is an issue with this algorithm: the rank can sink if the *surfer*, that performs the random walk, falls into a loop with no outgoing links.

To overcome this rank sink, the PageRank authors mention the possibility of creating a

Personalized PageRank (PPR) depending on the initial node. Further developed in Langville & Meyer (2004), this enhancement suggests a personalized vector with jump (or teleportation) probabilities. One instance could be a user interested in data science that is more likely to jump to pages with the same topic. In addition, this personalization can avoid cheating of web pages with thousands of links that try to rise pages ranks.

Further investigations based on PPR have been achieved, like Chakrabarti (2007) and Clements et al. (2008). One interesting version is the Topic-Sensitive PageRank (Haveliwala, 2002). In this approach, the vectors are biased offline using a set of representative topics. In addition, artificial links are created during rank computation with a probability of $\frac{\alpha}{N}$, being α the same damping factor as the original PageRank algorithm. This way, quality is improved and rank sinks are more limited. During query time, the ranks are combined with the query topic to build a composition that match the query. Topic-Sensitive PageRank performs a higher precision than the non-biased one.

Last but not least, an attractive algorithm using PPR within a specific domain is PopRank (Nie et al., 2005). This method uses a *popularity propagation factor* (PPF) to each edge type and computes rank taking into account object relevance and its relationships. To calculate PPF, the best neighbor combination is chosen, with a small probability of choosing the worse one to avoid traps as in rank sink. This would be computationally expensive for large graphs with big number of relationships. The authors recommend using subgraphs to reduce time based on circles with the training object in the centre. The maximum distance explored is determined by k. In consequence, the subgraph obtained is called *k-diameter subgraph*.

2.2 Nodes and links prediction

After achieving the node rank, it is time to predict new nodes and relationships. In most of the literature reviewed, the combinations of *node–edge–node* are known as *triples* o *triplets* and the prediction consists in finding one missing element.

One interesting investigation is Taskar et al. (2003) that tries to predict the existence of links and their type by using *relational Markov networks* (RMNs) based on a previous work (Taskar et al., 2002). In this work, every tuple of nodes has link, even if it exists or not. Then, every link is marked with attribute *Exists* to denote if the nodes are connected or not. With RMNs, the authors classify nodes in a set called *clique*. This way, they can learn the parameters of every clique and, therefore, predict links existence by entity attributes correlation. Every clique or subgraph has a template type. On one side, the *similarity* explains that nodes with similar properties are likely to have the same label. On the other hand, another type of template can show *transitivity*, where if exists a link between X-Y and other between Y-Z, there is a high probability for a X-Z link to exist.

In Jiang et al. (2012), *triplets* are represented as *subject–predicate–object* in multi-relational graphs. The statemates are represented as directed labelled links that starts in a subject node and point to an object node. This investigation takes into account the adjacency matrix seen in previous section literature. This model can derive new link types by aggregation that can predict links by transitivity. The computational cost of this algorithm is expensive, but the authors suggest using sub-matrices based on particular domain (*i.e.*, searching the most profitable actress in drama genre).

Relation learning can also be performed using Path Constrained Random Walks or *Path Ranking Algorithm* (PRA) (Lao & Cohen, 2010). In this method, the constraint is that the surfer can only walk through specific link types. It can also walk in reverse sense. As an example, the reverse relation of *stars in* can be read as *is being starred by*. Then, probability for every path from a source node –with direct and/or reverse relations– is calculated. In addition, query independent paths are computed, as seen in PageRank algorithm, and later combined with query dependent path. Moreover, the authors describe an extension by biasing popular entities of a particular type.

Finally, the work of Gardner & Mitchell (2015) presents the *Subgraph Feature Extraction* (SFE) algorithm, as an evolution of preceding paper. While PRA is a two-step process, SFE is similar as only doing the first step. SFE improves the mean average precision in less time. Furthermore, it can extract ever more features, including some that are not representable as paths. For each node constructs subgraphs with k random walks, as seen before in Nie et al. (2005). After completion, subgraphs with same intermediate nodes are merged. Then, new relations between nodes can show up.

2.3 The Movie DB dataset predictions

Most of the literature based on movies prediction that can be found are not exactly in the same line of this project. The bulk of the works are movie recommendation for Video On Demand platforms users. For instance, (Diao et al., 2014), tries to recommend movies by using a combination of movie rating and an analysis of *sentiments* or reviews written by users. On the other hand, (Viswanathan, 2015), tries to predict movies success by using three different models not based on graphs: Support Vector Machine (SVM), K-Nearest Nieghbours and Logistic regression. In addition, several works on this kind of predictions –revenue or rating prediction– can be found at Kaggle competition[1]. Most of these projects are based on linear o multilinear regressions.

An interesting work is Bogers (2010). It describes a recommendation algorithm called ContextWalk using random walks to extract user unseen movies. In a first step, a contextual graph with self-transitions is built out of data and creates its probability matrix. Then, this algorithm computes the transition probabilities starting on a fixed node and gets a personalized vector of probabilities as seen in PPR. The importance of this model is that it can be extended as a genre or actor recommendation. Modifying these extensions with models seen in the previous sections 2.1 and 2.2 for link and node prediction, the model could be fitted in this thesis project.

[1] https://www.kaggle.com/tmdb/tmdb-movie-metadata

Chapter 3

Data Analysis

The Movie Database dataset provided by Kaggle is divided in two different comma separated values (CSV) files: **movies**, where all the information about the profile of the movie can be found, such as revenue, languages, genres, etc.; and **credits**, data related with the people that worked in the movie such as the cast –actors and actresses– and crew, like the director, sound editor, producer, etc. This chapter describes the fields and attributes inside these datasets. After that, the decisions taken by the author to build the data model are explained.

3.1 Movies dataset

This dataset contains all data about the attributes of the movies. It has information of 4803 movies, released between 1916 and 2017. For every movie are 20 different fields as explained below. An example of information about a movie in this dataset is shown in table 3.1.

Budget

The budget is the money invested in making the movie. It is shown as an integer that outlines the amount of dollars spent. There are 1037 films with no budget.

Genres

This is the category of the style of the movie. It is presented as a list of tuples where every tuple has a unique identifier and the name of the genre. As an example, one of the movies with the highest number of genres is *Jimmy Neutron: Boy Genius* with seven different items: action, adventure, animation, comedy, family, fantasy and science fiction. In this dataset, there are 28 movies without any genre.

Field	Value
budget	237000000
genres	[{”id”: 28, ”name”: ”Action”}, {”id”: 12, ”nam...
homepage	http://www.avatarmovie.com/
id	19995
keywords	[{”id”: 1463, ”name”: ”culture clash”}, {”id”:...
original language	en
original title	Avatar
overview	In the 22nd century, a paraplegic Marine is di...
popularity	150.438
production companies	[{”name”: ”Ingenious Film Partners”, ”id”: 289...
production countries	[{”iso_3166_1”: ”US”, ”name”: ”United States o...
release date	2009-12-10
revenue	2787965087
runtime	162
spoken languages	[{”iso_639_1”: ”en”, ”name”: ”English”}, {”iso...
status	Released
tagline	Enter the World of Pandora.
title	Avatar
vote average	7.2
vote count	11800

Table 3.1: Example of fields in a single row of Movies dataset

Homepage

The URL of the movie webpage. Only 1712 movies (35.64%) has this field fulfilled.

Id

A unique identifier for every single movie. Using this field, we can relate the movie information with its credits information using the *movie_id* field of the Credits dataset as explained in below section 3.2.

Keywords

These are the words that define the movie plot or what is shown in the movie. Said another way, they describe the whole movie in a few important words or expressions. This field is a list of keywords that are presented as a tuple with a unique id and a name being the word or expression as the keyword. *My Date with Drew* has the highest number of keywords with 97 and some of them are: male nudity, female nudity, tattoo, card game, wife husband relationship, etc. 412 films are found without any keyword.

Original language

Despite the obvious sense of this field name, it can lead to wrong conclusions. Normally, the value of this field matches the original language of the movie. Sometimes, the value represents the language of the first producer or director. Anyway, it normally matches the language of the original title of the film. In some cases, there are errors and the original language does not match the spoken languages or the production countries. Incidentally, there is an error with *Rugrats in Paris: The Movie*: its original language is marked as Italian, but the spoken language is English and the production countries are Germany and United States.

Original title

As the field name says, this is the name of the movie in its original language with the character set in this language. This means that the title can be found in kanji characters if the movie is Japanese (i.e. Shin Gojira is shown as シン・ゴジラ).

Overview

The overview is the synopsis of the film. It is a kind of summary –without spoilers– written to attract the attention of the possible audience. It may contain some of the keywords of the film. The maximum number of characters for this field is 1000. There are 3 movies without overview: *Chiamatemi Francesco - Il Papa della gente*, *To Be Frank, Sinatra at 100* and *Food Chains*.

Popularity

This is a field created by the developers of the Movie Database. They use an own algorithm to rank every movie. As they explain in their API documentation[1], the popularity field is based on:

- Number of votes for the day
- Number of views for the day
- Number of users who marked it as a "favorite" for the day
- Number of users who added it to their "watchlist" for the day
- Release date
- Number of total votes
- Previous days score

The minimum score for popularity is 0.0 and the highest is 875.581305. There is only film with popularity equal to 0: *America Is Still the Place*.

Production companies

As its name indicates, this is the list of the companies that are in charge of the production of the film. This field is presented as a list of tuples, like previous fields, composed of a unique identifier and a name. There is a total of 351 films with no information about their production companies.

Production countries

Along the same line of the production companies, this field list the countries that takes part of the production of the film. In most cases, the countries match the location of the production companies' headquarters. For this field, 174 movies have no information about their production countries.

[1] https://developers.themoviedb.org/3/getting-started/popularity

Release date

This field indicates when the film was commercially available for general public. The date matches when the movie was released nationwide in its original country or worldwide. Only one movie has no value for this field. Again, it is *America Is Still the Place*.

Revenue

The revenue outlines the amount of money obtained worldwide gross for the exhibition of the film. This means that merchandising or home movies purchases are not included. The highest revenue in the dataset is for *Avatar* with a total of 2.787.965.087 dollars. The lowest revenue is 0 and there is a total of 1427 films with this amount. So this means they don't have this information. As proof, *Volcano* (1997) was a profitable movie and searching on the Internet can be found that it obtained more than 122 million dollars gross revenue.

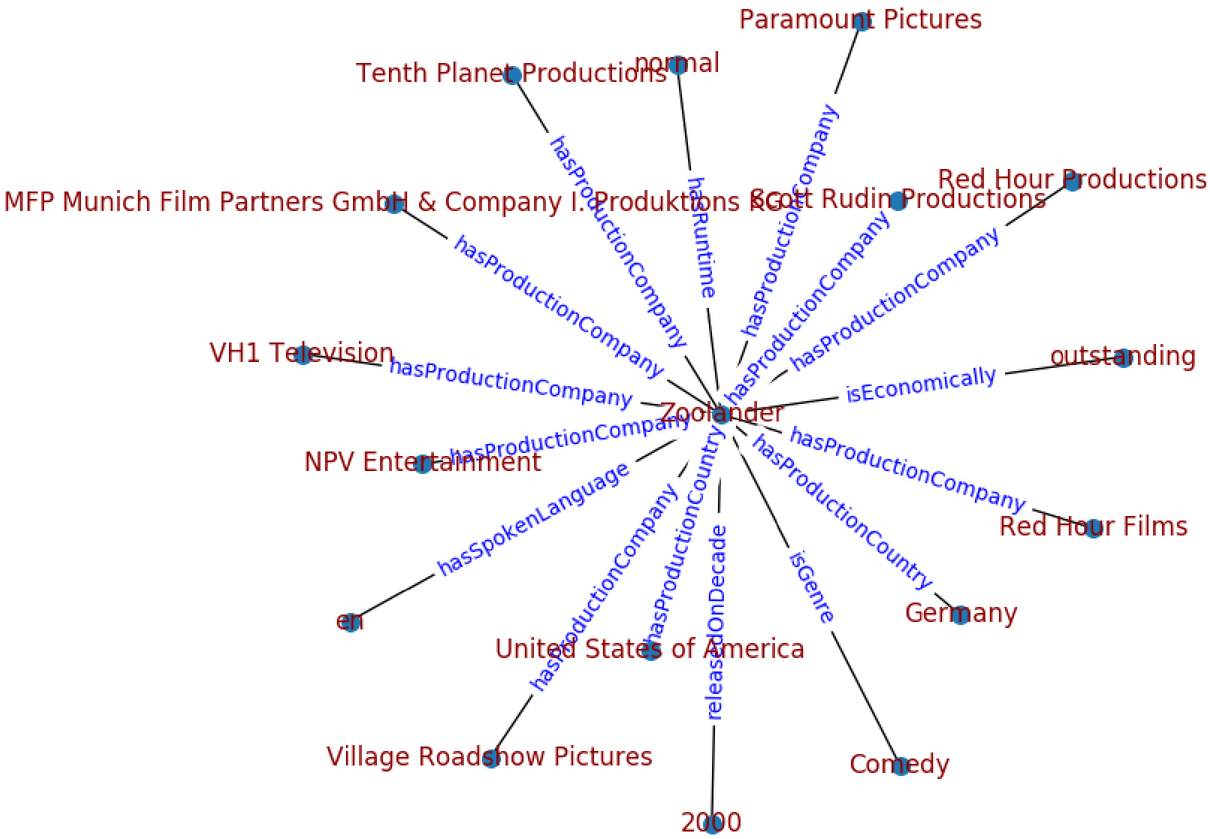


Figure 3.1: Example of graph with one movie and selected attributes

Runtime

This field describes how long the movie is in minutes. It starts counting from the beginning of the film to the very end of the credits at the end. There are 5 movies with no information or, in other words, its runtime equal to zero. The average runtime is almost 107 minutes and the longest film in the dataset is *Carlos* and it lasts 338 minutes.

Spoken languages

As its name suggests, this field indicates the languages that are used in the movie. It is presented as a list of tuples, where its first field is the ISO code of two characters for the language and the second one is the name of the language in its own language. This can cause some issues. If any letter is not present in the English alphabet, it appears with its Unicode. As example, French appears as *Fran\u00e7ais* instead of *Français*.
There are 86 movies without information of their spoken languages and the film with the largest number of spoken languages is *My Date with Drew* with 9: English, French, Hindi, Italian, Latin, Chinese, Portuguese, Russian and Bo (an ancient and extinct language of India).

Status

This field explains the state of the film in its life cycle. The status of the vast majority of the movies is *Released* (a total of 4597 items). But there are some that are *Rumored* (5 items) and others in *Post production* (3 items).

Tagline

The tagline is a phrase or group of small sentences that can be found in movies banners or posters. It works as a catchy slogan to attract the attention of the public. There are 844 movies without tagline and the longest one has 6 phrases with a total of 252 characters, achieved by the movie *Coal Miner's Daughter*.

Title

This fields shows the official translation of the original title (*see above*) in English. In general, the titles match the original titles. Some titles, as *El Mariachi*, do not need translation as they are word commonly known in the United States. Notice that the translation may not be exact. Sometimes the original title is translated –by producers or country distributors– to a catchier title. For instance, *Un Plan parfait* (*A perfect plan* in English) is translated as *Fly me to the moon*).

Vote average

As its name suggests, the vote average is the mean of all ratings given by users. Number range goes from 0 to 10. There is a total of 63 films with a zero, but only one of has one vote: *Sparkler*. In the other hand, there 4 movies with a 10 but they only have one or two votes.

Vote count

Finally, the vote count is the number of users that has rated a film. The most voted film is *Inception* with 13.572 votes. The vast majority of the movies have less than one thousand votes, followed by almost five hundred movies that have between one thousand and two thousand votes. As shown in Figure 3.2, 90% of the movies has less than 2.000 votes.

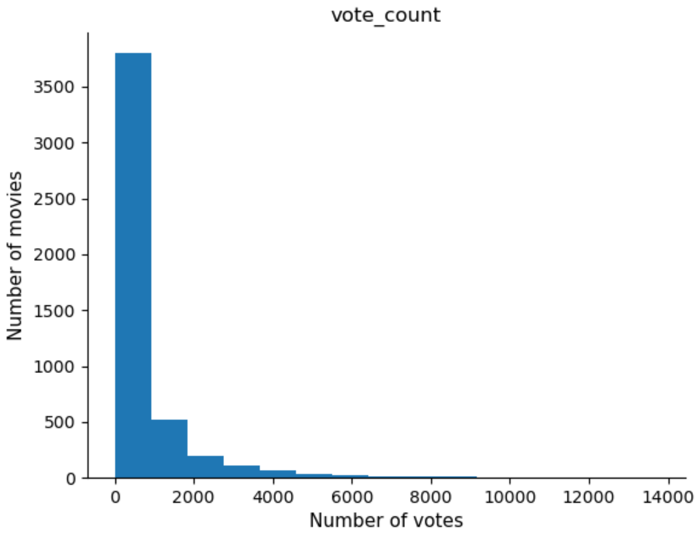


Figure 3.2: Histogram of number of movies by vote count.

Some of the attributes explained above will have to be discarded. Some others will have to be transformed. Every decision taken about data is further discussed below in section 3.3. To get a basic idea of how attributes will connect with their entities, an example of a limited model graph centered on *Zoolander* movie is shown in Figure 3.1.

3.2 Credits dataset

This dataset contains the data related with the people working on the film. Like the previous dataset presented in section 3.1, it has information of 4.803 movies. In this case, there 4 different fields for every film.

The first field found is *movie_id*, a unique identifier that matches with the same movie identifier seen in the previous dataset. The second field, *title*, matches its corresponding title as well in that dataset. This section will explain the other two attributes: *cast* and *crew*.

Cast

This field outlines the people that acts in the movie. They are registered as a list of tuples with the 7 attributes as below:

- **cast_id**: this unique identifier, shown as an integer, is a local id for the character. In other words, is a role identifier inside a particular movie. This means that different characters in other films could have the same identifier.
- **character**: this is the name of the character played by the actor. E.g. *King Arthur* in *Excalibur* or *Mighty Steel Leg Sing* in *Shaolin Soccer*.
- **credit_id**: this field is a unique identifier, as an object ID, for this particular credit in TMDB database. This means that querying a *credit_id* on the TMDB API, the response will be all the information for a particular character/actor of a precise movie. I.e, querying the credit id '52fe47c99251416c91075af7' will return all the information about Tom Cruise's character *Stacee Jaxx* on the film *Rock of Ages*.
- **gender**: an integer that outlines the gender of the actor. Number 1 is for female and 2 is for male. In addition, number 0 can be found for 'not set' gender, as discussed in the TMDB Kaggle forum[2].
- **id**: This is a unique identifier, shown as an integer, for every person in the database, not only actors or actresses. Like *credit_id*, it is used to query people information on the TMDB API.
- **name**: a string with the name –artistic or real– of the actress or actor that plays the character in the movie.

[2] https://www.kaggle.com/tmdb/tmdb-movie-metadata/discussion/58203

- **order**: this field, shown as an integer, can be confusing as it can be viewed as the order of actor appearance in the film. But what it really means is the importance of a particular actor in the movie. In other words, the actor with the lowest value in this field is the main character of the movie and the one in the 224th position could be one playing an Extra in the background of a very crowded scene.

There is a total of 43 movies without information of the cast. The largest cast appears in the film *Rock of Ages* with 224 actors.

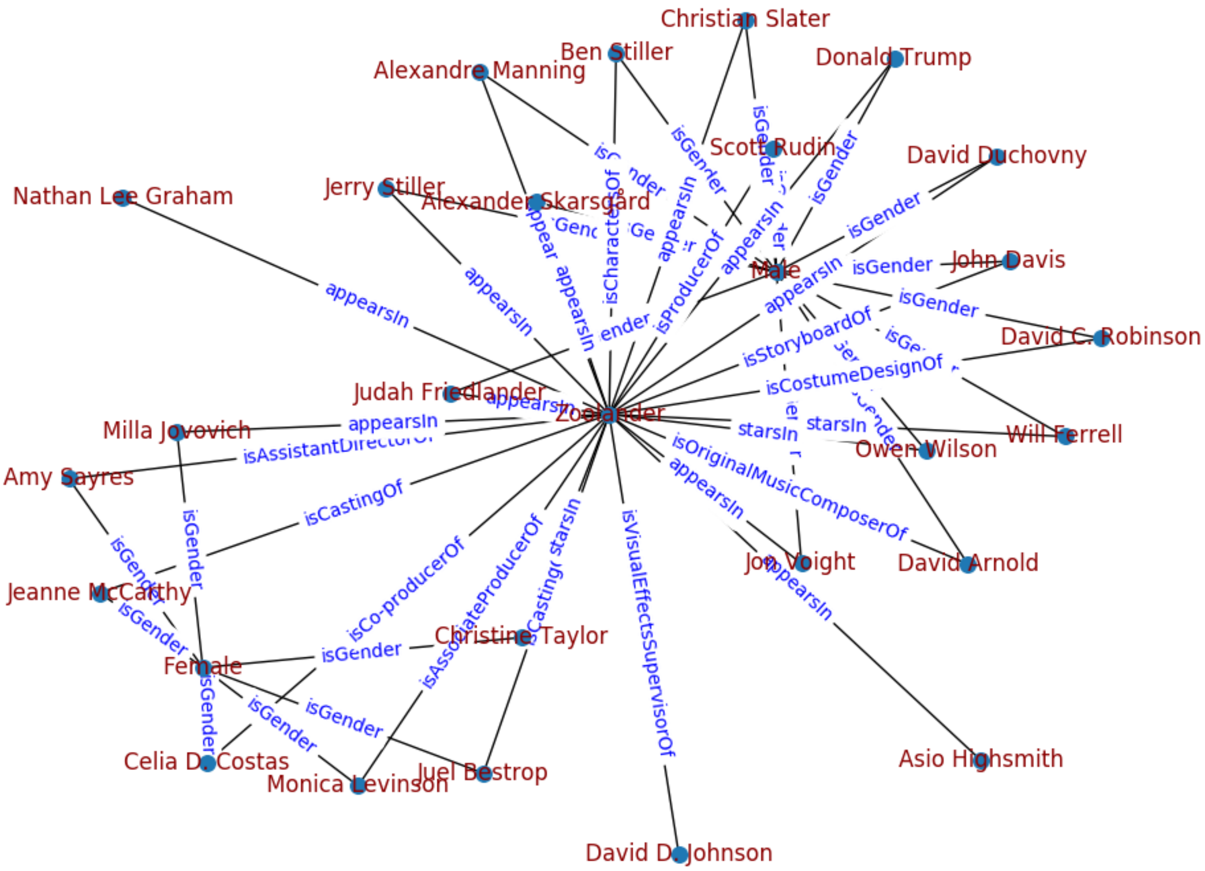


Figure 3.3: Example of graph with one movie and selected people of cast and crew

Crew

This field describes the staff that, normally, does not appear in frame of the film. In general, they are the 'technicians' behind the camera. Like the *cast* field, it is presented as a list of tuples but with 6 attributes as explained below:

- **credit_id**: this attribute has the same purpose as in *cast* field, a unique identifier –as an object ID– for this particular credit in the database.
- **department**: a string that outlines the department name to which the person belongs. E.g, Art, Camera, Costume and make-up, etc. When there is not a particular department to fit the staff, it has the value 'Crew'.
- **gender**: the people gender showed as an integer, as seen in the *cast* field.
- **id**: like in *cast* field, an integer as unique identifier for the whole TMDB database for every person.
- **job**: this string describes the specific job position inside the movie and/or department. For instance, supervising art director, set designer, production design, etc... can be found in the Art department.
- **name**: a string with the name –artistic or real– of the person that performs the job.

There is a total of 28 movies without crew information. The largest crew appears in the film *Jurassic World* (2015) with 435 people. Lastly, in the same way as in Movie dataset section, a limited graph model can be seen in Figure 3.3. This time the graph is limited to 30 people, half are part of the movie cast and the other half from the crew. As can be observed, some people whose gender information is missing are not connected to any of the gender nodes.

3.3 Data model, decision and cleansing

This section explains the decisions taken to build the data model from the Kaggle datasets. Not all the information provided can be included for the sake of simplicity. Higher complexity means more resources and time for the model to learn.

The data model will be a graph $\mathcal{G}$. This graph will be composed of a set of nodes $\mathcal{N}$ that will be related with edges $\mathcal{E}$. Every edge will have a label outlining a binary relationship. Nodes should represent the dataset attributes such as movie title, genre, country, etc. The edges will relate these attributes for every movie and actor and staff people in the film. That is the basic idea but there are some changes that will be applied to make a better model.

The first decision taken is to not create any relationships if the data is missing. If a *Null* node is created, lots of nodes with several relationships will be related with it, turning this node in the most related one. This could induce the machine learning algorithm to consider this node as the most important and be the first option for predicting new relationships. In addition, gender 0 will not be used as it is not clearly defined.

Movie name will be taken from the *Title* attribute. As seen in the previous section, *Original Title* can be found with a character set of the original language, but this can cause problems for drawing the model. In the other hand, the *Title* field is normally written in English and uses ASCII characters set.

The homepage URL is only available for the third part of the films. Boolean nodes could be created ($True,\ False$) and relate every movie with $\mathcal{R} = \{'hasHomepage'\}$. But this information is not relevant enough and these Boolean nodes should be avoided. In consequence, homepage information will not be used.

Numerical attributes can be used for regression models. But, for the purpose of this project, they have to be discretized. This means that instead of creating a node for every number that appears as attribute, it will be classified into a category bin. The numerical attributes will be categorized as follows:

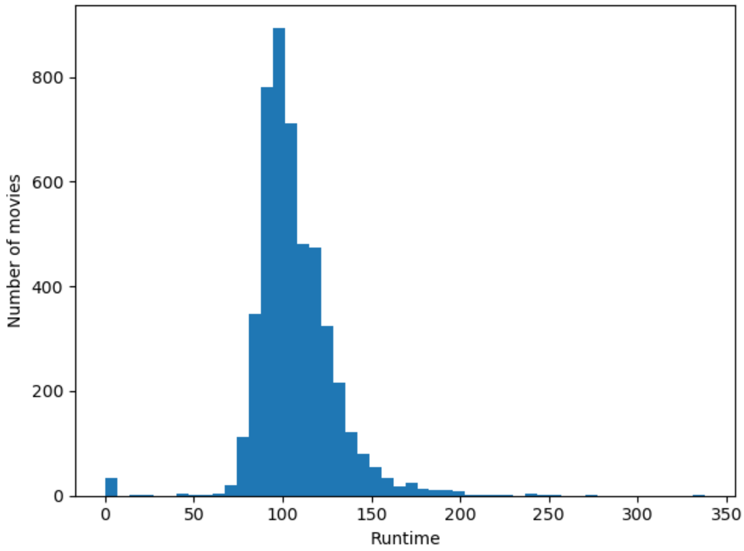


Figure 3.4: Histogram of number of movies by runtime.

- **Runtime**: A short film, as described by the Awards Academy (Oscars, 2019), has a runtime of 40 minutes or less. For the Actors Guild (SAGA, 2019), a short film is below 60 minutes. This means that any movie with a runtime over those minutes can be considered as a feature film, so technically there are not *long* movies. Figure 3.4 shows that the vast majority of movies are around the average runtime (107 minutes). A zoomed histogram can be drawn by discarding movies below 40 minutes and over 180 minutes long. Figure 3.5 indicates that most of the films has a runtime attribute between 80 and 140 minutes. The average runtime, as seen before, is 107 minutes.

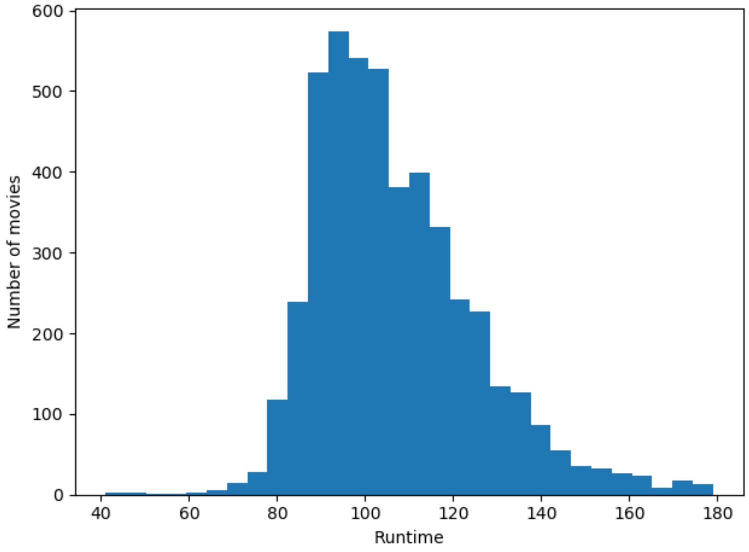


Figure 3.5: Number of movies by runtime, with a runtime between 40 and 180 minutes.

Taking all this information into account, *short* runtime category will be for movies that last 60 minutes or less. *normal* film runtime will be over 60 minutes and less that 140. Any film longer that 140 minutes will be classified as a *long* movie.

- **Budget** and **Revenue**: these two attributes can be converted into a margin percentage to see if the movie has produced benefits or not. There is a total of 1574 movies that has no information about its budget, its revenue or none of both. In these cases, margin relationship will be discarded.
  Revenue is the total amount of money obtained on theatres all around the world. The problem is that the taxes and fees must be subtracted from that amount. As it may vary from one country to other –even between states of the same country–, this project will consider that 10% of the revenue is for taxes and another 50% will be for distributors, theater fees, etc. In addition, the production and advertising costs of the movie –budget in this case– must be subtracted as well. In consequence, the margin could be formulated as:

$$Margin = \frac{0.6 \times Revenue - Budget}{Budget}$$

Four categories will be created for this margin attribute: a film will be an economic *disaster* if its margin is less than -20%; *non-profitable* when its margin is between -20% and 0% (included); *profitable* when it gets a benefit margin up to 20%; and *outstanding* when higher.

- **Vote average** and **Vote count**: how much a movie is liked can be determined with *vote average* attribute as a rating. This way, excellent films can be marked. Despite the maximum vote is 10, there are only six movies with an average greater or equal to 9 and with a very low vote count (Table 3.2). The votes of one or two person cannot be considered as a good average tendency. In consequence, the first condition for a film to be eligible as an *Excellent movie* is if it has 100 or more votes. Filtering this way, the maximum vote average is now 8.5. Then, the second condition for the criteria will be to have a rating greater or equal to 8. Using these conditions to filter the dataset results in 59 movies such as Kurosawa's *Seven Samurai*, Chaplin's *Modern Times* or Spielberg's *Schindler's List*.

Title	Vote count
One Man's Hero	2
Stiff Upper Lips	1
Sardaarji	2
Dancer, Texas Pop. 81	1
Me You and Five Bucks	2
Little Big Top	1

Table 3.2: Movies with *vote average* ≥ 9.

Popularity attribute has been discarded. As seen in section 3.1, the number is determined by their own algorithm. This value can change every day, so it is not a permanent fact that describes the movie over time. Moreover, its distribution shows several outliers over the maximum value with a low mean value, as can be seen in the boxplot of Figure 3.6.

The *Keywords*, *Overviews* and *Tagline* attributes will not be used in this project. They need a Natural Language Processing (NLP) treatment that is out of the scope of this work. Keywords could be used as nodes, but there are a lot of different words and the resulting graph would be too complex.

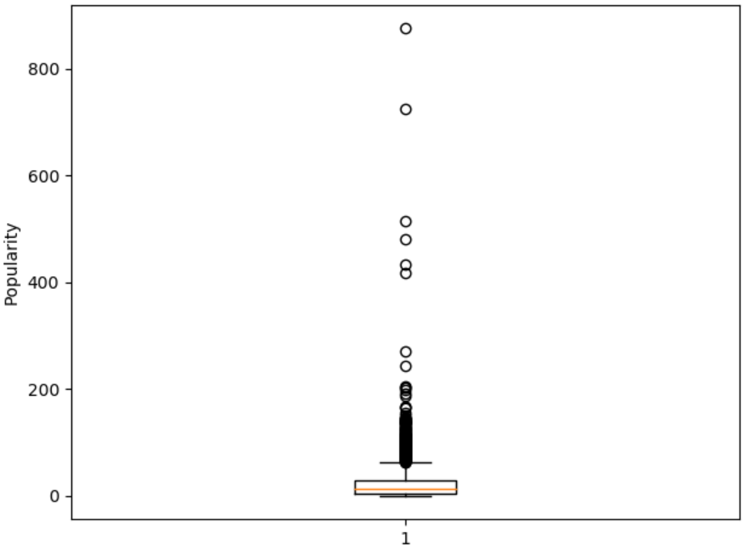
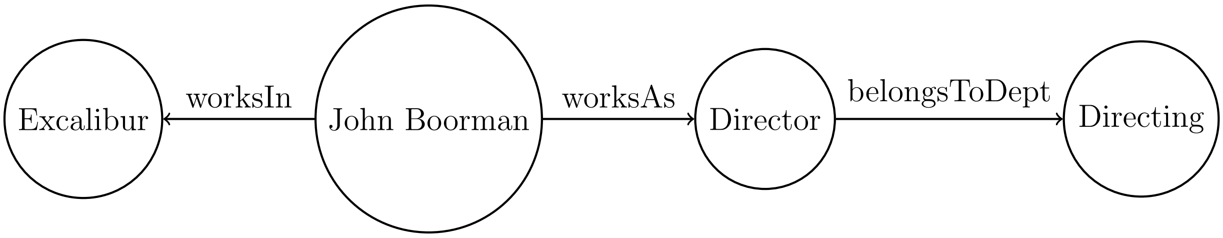


Figure 3.6: Popularity boxplot representation.

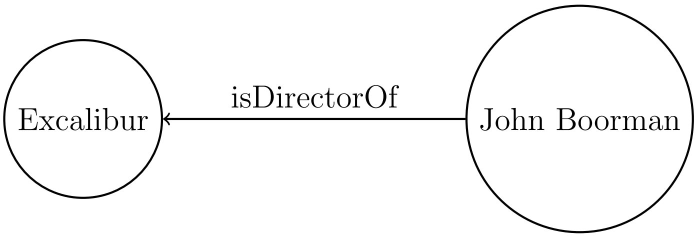
Regarding the *Release date* attribute, creating a node for every day will increase model complexity dramatically. A first attempt to discretize this field is to classify every date in different ages of cinema history. For example, classify as *silent movie* every movie before 1927, the year when the first sound movie –*The Jazz Singer*– was released. The problem with this classification is that there are different kind of ages depending on the year and the country. I.e., when it was the Golden Age of Hollywood film studios, the neorealism movement was taken place in Italy. So, for the sake of simplicity, the release date will be categorized in cultural decades, that is from year ending in 0 to year ending in 9. Century will also be taken into account. In consequence, a movie released on 1916 will belong to decade 1910s and other released on 2015 will be classified in 2010s decade.

Actresses and actors in casting has different importance, as seen in section 3.2. For this project, the first four actors –sorted by the *order* field– will be considered as the main characters (lead actors and supporting roles) and will be related with the movie with $\mathcal{R} =' starsIn'$. The rest of the cast will be linked with $\mathcal{R} =' appearsIn'$.

For crew's people attributes *job* and *department*, a first design was made considering both fields like nodes as follows:



The problem with this model is when a person works in different films or has several jobs in the same field. The model will show that this person has worked on different movies what will not differentiate what kind of job performed in a particular film. To solve this particular issue, the next model is proposed:



In this model, *job* node is replaced by a relationship. This way the model can know the job position of the technical staff in a particular movie. Unfortunately, the *department* information is lost because the node cannot be related to the *is_JobPosition_Of* link.

Like in any other data science projects, the identifiers attributes are dumped with two exceptions: movie id and actor/person id. This fields are needed to identify movie and person nodes as explained in next chapter, in section 4.1.

Finally, the film *America Is Still the Place* will be discarded from the dataset. Almost all of its information is missing and can be considered as an outlier. Additionally, movies in *Rumored* and *Post production* status will also be dropped from the set because the lack most of the information needed. As seen in section 3.1, the number of films with these statuses are 8. In consequence, the **total number of movies remaining in the dataset will be 4795**. A graphic example of the final model is shown in Figure 3.7, limited to one movie and eight persons, four belonging to cast and the other four belonging to the technical crew.

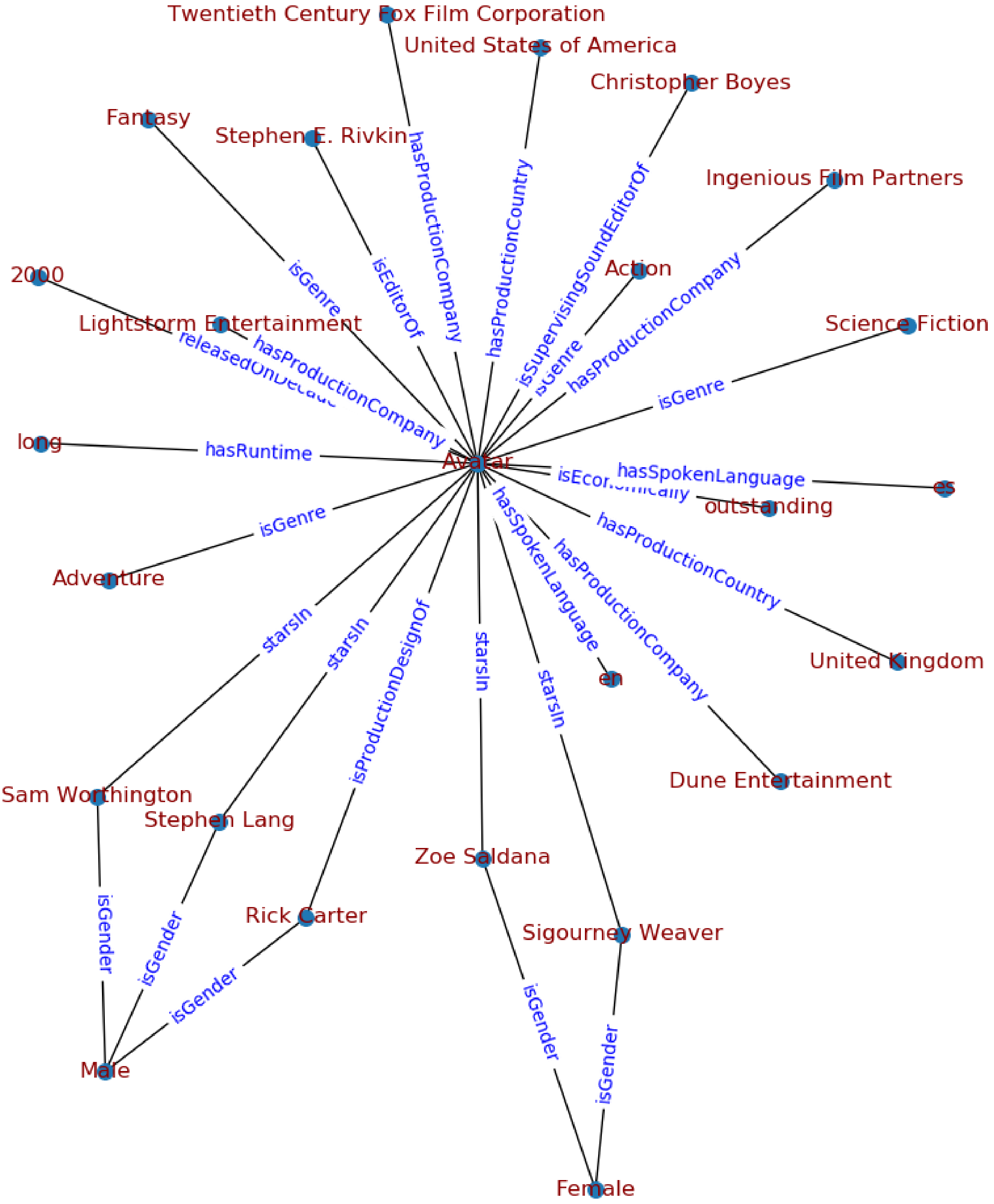


Figure 3.7: Graph model with limited nodes

Chapter 4

Implementation

Once data has been analysed and it has been decided which attributes are going to be included and which are not, it is time to build the model and work on it. This chapter outlines, in first place, how the model is built out from The Movie Database dataset once cleaned and the properties it has. After that, different machine learning techniques will be explained and executed to perform relationship predictions.

4.1 Model building

The way the model is built is not trivial. There are a lot of languages right now that could be used to produce the graph to work with. Besides that, nowadays exists powerful graph databases like Neo4j[1] that allow to construct the model and to query it easily.

For this project, the chosen language is **Python**. The main reason for choosing it over others –like R or Java– is that it has been the main language used in the Master's degree subjects. Furthermore, it is a very easy language to learn for those who has a computer science background. This language has one of the biggest communities around the world and it has been breaking in the past years. This is why learning it is the first of the secondary objectives of this project.

There could be infinite ways to build the graph in Python. One way is to create new data structures, classes and methods, to allocate the model in the most desired way. However, there is not enough time to implement it and it is out of this project main purposes. There's no need to reinvent the wheel. Consequently, a graph package has been chosen from the wide range

[1] More information at neo4j.com

existing for Python: **NetworkX**.

NetworkX is a Python package developed by Aric Hagberg, Dan Schult and Pieter Swart to create and manipulate complex networks. This package has lots of interesting and powerful functions. It can build a graph easily in just for steps: first, declare a new graph; then, add two nodes to the brand new graph; finally, add an edge to the graph between the nodes. A code snippet in Python will look as follows:

```
G = nx.Graph()

G.add_node(node1)

G.add_node(node2)

G.add_edge(node1, node2)
```

There are four type of graphs to choose from: undirected, directed, undirected multiedge and directed multiedge. For the purpose of this project, the chosen one will be an undirected graph. This way, edges can be walked in any direction. To know if the direction if reverse will depend on the types of origin and destination node.

Every **node** added to the graph **must have an identifier**. To keep it simple, the node id will be a word that can easily identify the node, followed by an underscore and a number or classifying name. For movies and people, for example, the number used will be their own *id* attribute. On the other hand, 2-digit ISO code will be used to identify the languages. The proposed node identifiers for all nodes are shown in table 4.1.

Cast and crew are not separated by specialized types but generalized as *Person_ID* node. This is due to people that can perform different jobs, acting in one film and producing or directing in another one (or even in the same one!). To know whether if that person is making one job or another, the edge label will be taken into account.

In addition, nodes can have any desired attribute: label, weight, etc. For this project, all nodes will have the attribute *name* that will keep the information of the node, whether is a person name, movie title or an adjective. This will make it easier for the package to loop over the graph and draw it. **Edges**, like nodes, can have any desired attribute. In this project, the

Type of node	Node identifier	*Example*
Movie	Movie_{movie_id}	*Movie_559 (for Spider-Man 3)*
Language	Language_{2-digit code}	*Language_en*
Genre	Genre_{genre_id}	*Genre_14 (for Fantasy)*
Decade	Decade_{first year of decade}	*Decade_2000*
Country	Country_{ISO code}	*Country_US*
Production company	Company_{company_id}	*Company_5 (for Columbia Pictures)*
Gender	Gender_{number}	*Gender_1 (for female)*
Person	Person_{person_id}	*Person_10205 (for Sigourney Weaver)*
Runtime	Runtime_{adjective}	*Runtime_normal*
Margin	Margin_{adjective}	*Margin_out (for outstanding margin)*
Rate	Rate_S (unique)	*Rate_S (for excellent movies)*

Table 4.1: Node identifiers

only attribute that edges will have is the *label* and it will be self-explanatory and taking into account the decisions taken in section 3.3.

Last but not least, NetworkX has its own function to render graphs. All graph figures with limited nodes shown in previous sections, like Figure 3.7, are rendered using this feature.

4.2 Machine learning algorithms

Now that the model has been created, it is time to carry out the main purpose of this project. In this section, different algorithms will be used to perform new relationships predictions. The order of execution chosen is not accidental. An algorithm can use the previous one, be based on it or its evolution. All of them have been mentioned in the State of Art, in chapter 2.

4.2.1 Personalized PageRank

The first algorithm to be applied is the PPR (*Personalized PageRank*), also known as Random Walk with Restart (RWR). As mentioned before, this algorithm is an evolution of Google's PageRank and it was further developed in Langville & Meyer (2004). This method performs random walks starting on a particular node with a low probability of jumping and starting again from the initial node.

The process begins in the picked node as starting point. Then, any of the node's neighbors is chosen randomly. A node is considered a neighbor if it is connected with the original node with an edge. In other words, two nodes are neighbors if they have a relationship between them. After moving to the next node, another random neighbor node is selected. This process continues till reaching the maximum number of steps or jumps desired. At any moment, before moving to the next node, the probability of jumping to the initial node is pondered. If a random calculated probability if less or equal than the *jump threshold*, the next node is the starting node. If not, the process continues. Every time a node is visited its rank –or weight– increases.

Once the number of steps is completed, a vector is obtained. The vector indexes correspond to graph nodes and every value is the rank obtained. This vector is personalized for node chosen as starting point. This means that the nodes that are closer to the start will have higher rank compared to those that are far or not connected at all. For these cases, the rank value is 0.0 by default. In addition, those nodes with a high degree –a high number of relations– should have a higher rank because they are more likely to be visited from several neighbor nodes. As a last step, the resulting vector of ranks is divided by the total number of graph nodes to normalize ranks. Pseudo-code for this process is shown in Algorithm 1.

To perform a prediction with this method, first of all a known node must be chosen and execute PPR on it. Then, search for a known relation and get the rank value of the related node. After that, the graph is cloned and the known relationship removed. In this new graph without the old edge, a new PPR is launched for the same initial node. Then, the value obtained in the same vector index is compared. After executing this process on several nodes, new rank values on target nodes can be studied to calculate a threshold to predict if a relation between pair of nodes should exists.

To conclude, experience tells that ranks should decrease after removing any relationship. The first reason is that removing an edge should decrease most of the ranks if nodes were connected through the removed edge. On the other hand, the rank for the target node can even be zero if there is no longer a connecting path between the pair of nodes.

Algorithm 1 Personalized PageRank

```
Require:
    G as graph
    Number of Random Walks
    Number of steps
    Initial node
    Probability of restart

$S = G.number\_of\_nodes()$
$v = vector.zeros(S)$
for $i = 1 \rightarrow N\_RandomWalks$ do
    $n = initial\_node$
    for $j = 1 \rightarrow N\_Steps$ do
        $p = random()$
        if $p < restart\_probability$ then
            $n = initial\_node$
        else
            $n = walk(g, n)$                    ▷ Return random related node
        end if
    end for
    $v[n] = v[n] + 1$
end for
$v = v/S$                                       ▷ Normalize values
return $v$
```

4.2.2 Path Ranking Algorithm

As explained in chapter 2, Path Ranking Algorithm or PRA (Lao & Cohen, 2010) is a process of two steps to perform link prediction.

Firstly, this process finds possible path types to be used as features. A path is composed of the node types and labelled relationships between a node pair (A, B). In consequence, the first thing to do is finding node pairs that are related by the desired relation to predict, for example $Person \rightarrow starsIn \rightarrow Movie$. Once a list of node pairs that fits with the requirement is found – called *positive pairs*– every pair is treated in isolation in the graph. For each pair, a copy of the graph is made and the relationship is removed. Then, a limited number of new paths that connects node A with node B are searched with a maximum of step/jumps of depth. New paths are known as **feature path**. An illustrative example can be found in Figure 4.1: once the original *starsIn* relationship is erased, a possible path to reach node B (*Movie 1*) could be $Person1 \rightarrow starsIn \rightarrow Movie2 \leftarrow appearsIn \leftarrow Person2 \rightarrow appearsIn \rightarrow Movie2$. Once new feature paths are discovered, they are ranked being the ones with more weight the most important. One way to rank the feature paths is to count the number of occurrences that every paths has in the resulting list.

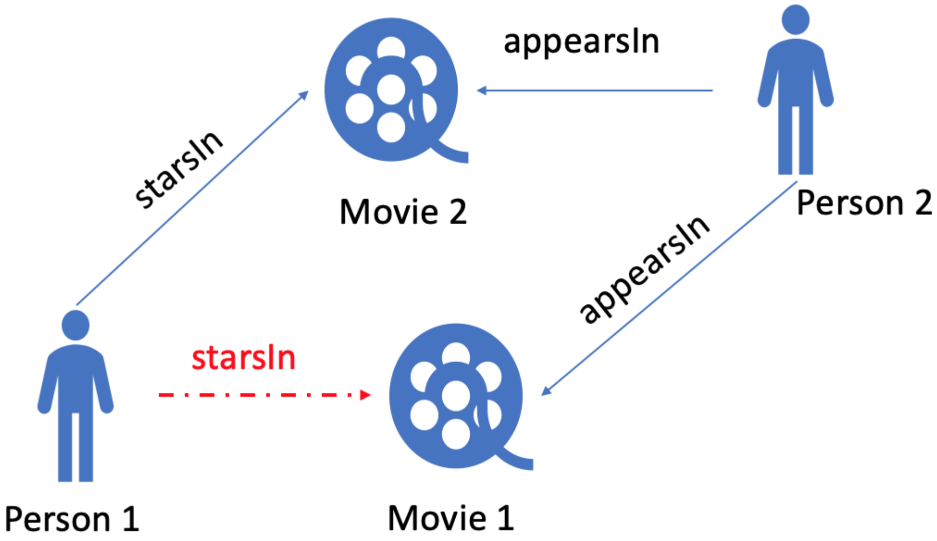


Figure 4.1: Feature path discover

The second step of the process is to compute Personalized PageRanks for every node pair and every chosen feature path to find the probability of reaching node B from initial node A. In addition, negative node pairs should be added to the node pairs bucket. A node pair is considered negative is the relation is not equal to the positive one but the node type for A and B are the same. Hence, the number of node pairs to loop over is doubled. Once a PPR for a particular node pair and feature path is computed, the rank value of node B is stored in a matrix where node pairs will be the row index and feature paths will be the columns. Once the whole process is finished, a last column is added to outline if the node pair is positive or node (1 or 0). Therefore, a matrix with binary target labels is obtained. Since the target labels follow a Bernoulli distribution, logistic regression can be used to perform predictions.

As outlined in Gardner & Mitchell (2015), this second phase consumes a lot of resources, as shown below in the experiment in section 4.3.2. As displayed in the pseudo-code in Algorithm 2, the PPR algorithm described in section 4.2.1 must be modified to walk through the graph according to the marked path. This means that selected edges and node types to walk through must have the same labels and types that appear on the feature path. Because of this, trying to reach the final node through random walks is very complex and it is not guaranteed unless thousands or millions of Random Walks with Restart are performed over the graph.

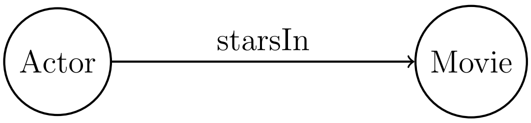
Algorithm 2 Path Ranking Algorithm

```
Require:
    G as graph
    Tuple_to_predict: (Initial node type, Relation, Final node type)

$node\_pairs = G.get\_pairs(tuple\_to\_predict)$
$path\_list = []$
for each $pair$ in $node\_pairs$ do
    $F = G.removed\_edge(pair)$
    $paths = F.get\_paths(pair, n\_paths, depth)$
    $path\_list.append(paths)$
end for
$top\_paths = path\_list.most\_common(10)$
$negative\_pairs = G.get\_negative\_pairs(tuple\_to\_predict)$
$node\_pairs.append(negative\_pairs)$
for each $pair$ in $node\_pairs$ do
    for each $path$ in $top\_paths$ do
        $F = G.removed\_edge(pair)$
        $initial\_node = pair\,[0]$
        $final\_node = pair\,[1]$
        $v = \textbf{modified\_ppr}(F, n\_RWs, steps, initial\_node, path, restart\_prob)$
        $\textbf{probability\_matrix}\,[pair]\,[path] = v\,[final\_node]$
    end for
end for
$probability\_matrix\,[label] = labels\_list$
return $probability\_matrix$
```

4.3 Experiments and results

After completing algorithm development, it is time to perform some experiments. In the same way as section 4.2, Personalized PageRank will be the first algorithm to be tested followed by Path Ranking Algorithm. For the sake of simplicity and complexity, the relationship to be predicted will be that of an actor starring a film. Expressed graphically:



4.3.1 PPR experiment

To illustrate the method, Person type nodes will be picked as initial nodes. For this experiment, the actors picked will be Antonio Banderas, Meryl Streep and Ian McKellen that correspond to node identifiers $Person\_3131$, $Person\_5064$ and $Person\_1327$ respectively. For each of these nodes, the PPR algorithm is executed with 1000 RWs with 50 steps for each walk. The probability of jumping to the initial is set to 10%. The process will return a vector with the normalized rank values calculated by walking the graph from the initial node. This means that if a node is not visited, its value will be 0.0.

Once the vector for every actor is obtained, the index of a movie starred by the actor must be found to get its rank. The selected films for Banderas will be *Puss in Boots* (Movie_417859), *Four Rooms* (Movie_5) and *Once Upon a Time in Mexico* (Movie_1428); for Streep, *The Hours* (Movie_590), *The Iron Lady* (Movie_71688) and *Death Becomes Her* (Movie_9374); finally, for McKellen, the chosen films are *The Hobbit: the Desolation of Smaug* (Movie_57158), *Gods and Monsters* (Movie_3033) and *X-Men 2* (Movie_36658). The process of calculating PPR algorithm for a single node takes 5 minutes on a personal computer.

After collecting the rank values for those movies, it is time to predict with this algorithm. For every node and movie, the graph must be cloned and the edge that joins those nodes removed. Next, a new PPR algorithm is executed again on the cloned graph without the relationship with the same initial node and parameters. At last, the new rank values for the films are collected and compared with the old ones. Results are shown in table 4.2.

Actor	Movie	Original graph	Edge removed
Antonio Banderas	Puss in Boots	0.0000087075	0.0
Antonio Banderas	Four Rooms	0.0000522452	0.0
Antonio Banderas	Once Upon a Time in Mexico	0.0000348302	0.0
Meryl Streep	The Hours	0.0000261226	0.0
Meryl Streep	The Iron Lady	0.0000261226	0.0
Meryl Streep	Death Becomes Her	0.0000261226	0.0
Ian McKellen	The Hobbit: Desolation of Smaug	0.0000870754	0.0000087075
Ian McKellen	Gods and Monster	0.0000870754	0.0
Ian McKellen	X-Men 2	0.0001131980	0.0000087075

Table 4.2: PPR ranks

4.3.2 PRA experiment

In this section, an experiment with Path Ranking Algorithm is performed over the TMDb graph model. The first step is to choose what kind of relationship is going to be predicted. Taking into account the same kind of prediction as in PPR experiment, the triplet used in this process will be $Person \rightarrow starsIn \rightarrow Movie$.

As commented in section 4.2.2, this is a two-step process. In the first step, a bucket of 200 positive node pair is used to search feature paths. For every positive pair, the graph is cloned and their relationship removed. Then, a total of 200 iterations are made to find a maximum of 10 feature paths. After completing the whole list of positive pairs, a set of 75 valid paths –repeated or not– is obtained in the first place and then counted. After that, the top 10 most common feature paths are chosen. These paths ordered by number of occurrences can be seen in table 4.3.

Weight	Path
3	$Person \rightarrow starsIn \rightarrow Movie \leftarrow appearsIn \leftarrow Person \rightarrow appearsIn \rightarrow Movie$
3	$Person \rightarrow appearsIn \rightarrow Movie \leftarrow isDirectorOfPhotographyOf \leftarrow Person \rightarrow isDirectorOfPhotographyOf \rightarrow Movie$
3	$Person \rightarrow appearsIn \rightarrow Movie \leftarrow isCastingOf \leftarrow Person \rightarrow isCastingOf \rightarrow Movie$
2	$Person \rightarrow appearsIn \rightarrow Movie \leftarrow hasProductionCompany \leftarrow Company \rightarrow hasProductionCompany \rightarrow Movie$
2	$Person \rightarrow appearsIn \leftarrow Movie \leftarrow isHelicopterCameraOf \leftarrow Person \rightarrow isHelicopterCameraOf \rightarrow Movie$
2	$Person \rightarrow appearsIn \rightarrow Movie \leftarrow isTechnicalSupervisorOf \leftarrow Person \rightarrow isTechnicalSupervisorOf \rightarrow Movie$
1	$Person \rightarrow appearsIn \rightarrow Movie \leftarrow appearsIn \leftarrow Person \rightarrow isGender \rightarrow Gender \leftarrow isGender \leftarrow Person \rightarrow isArtDirectionOf \rightarrow Movie$
1	$Person \rightarrow isGender \rightarrow Gender \leftarrow isGender \leftarrow Person \rightarrow isScreenplayOf \rightarrow Movie \rightarrow isGenre \rightarrow Genre \leftarrow isGenre \leftarrow Movie$
1	$Person \rightarrow isGender \rightarrow Gender \leftarrow isGender \leftarrow Person \rightarrow appearsIn \rightarrow Movie$
1	$Person \rightarrow appearsIn \rightarrow Movie \leftarrow isCompositorsOf \leftarrow Person \rightarrow isVfxArtistOf \rightarrow Movie \rightarrow hasSpokenLanguage \rightarrow language \leftarrow hasSpokenLanguage \leftarrow Movie \leftarrow isDirectorOf \leftarrow Person \rightarrow isDirectorOf \rightarrow Movie \rightarrow hasProductionCountry \rightarrow Country \leftarrow hasProductionCountry \leftarrow Movie$

Table 4.3: Ranked paths

Once the first step is completed, it is time to perform PPR using the chosen feature paths as a constraint to walk over graph. Firstly, 100 positive node pairs are randomly selected from the original bucket. Next, 100 random negative node pairs are picked from the model and added to the list of node pairs to look for. These node pairs, both positive and negative, will play the role of row index in a later matrix of 200 rows per 10 columns. After completing the node pairs list, the PPR is launched for every node pair and every selected feature path with the next parameters: 150 random walks, 20 steps depth from the initial node and a restart probability of 15%. As outlined before, once a PPR for a selected feature path $Type1 \rightarrow Edge1 \rightarrow Type2 \rightarrow Edge2 \rightarrow ... \rightarrow EdgeN \rightarrow TypeN$ is completed, the rank value for node B starting from node A is stored in the final matrix in row index (A, B), column $(Type1, Edge1, Type2, Edge2, ..., EdgeN, TypeN)$. Finally, once the all values are calculated for every node pair and feature path, a label column is added to the matrix labelling positive pairs with 1 or 0 for negative pairs. The whole process took 6 hours to be completed on a personal computer. An example of the first 5 rows of the resulting matrix can be seen in table 4.4, where path order is the same as in table 4.3. Most of the values in the matrix is 0 because reaching the final node using random walks with a path constraint is not guaranteed and very complex.

node pair	path 1	path 2	path 3	path 4	path 5	path 6	path 7	path 8	path 9	path 10	label
$(Person\_13247, Movie\_327)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
$(Person\_51641, Movie\_12088)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
$(Person\_171472, Movie\_46705)$	0.000548575	0.000261226	0.000130613	0.000452792	0.000287349	0.000313471	0.000191566	0.000322179	0.0	0.0	0
$(Person\_3610, Movie\_43867)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
$(Person\_118403, Movie\_27576)$	0.000949122	0.0000522452	0.0	0.0000870754	0.000626943	0.000592113	0.000322179	0.00060082	0.0	0.0	0

Table 4.4: First 5 rows of PRA resulting matrix

As commented before, the label column shows a Bernoulli distribution. Consequently, a logistic regression model can be built to predict possible relationships. For this purpose, the matrix data is divided 75/25 for the training and testing sets. This means that 150 rows are used to train the model and 50 to test it. Once the logistic regression model is built and the test set used to predict labels, a confusion matrix is obtained (table 4.5).

		True diagnosis		
		Negative	Positive	Total
Screening test	Negative	6	0	6
	Positive	19	25	44
	Total	25	25	50

Table 4.5: Logistic Regression confusion matrix

Once known how the classification is, accuracy, precision, recall and F1 score metrics can be calculated. The **accuracy** is the ratio of correctly predicted observation over the total observations to be predicted. For this case, the accuracy achieved is 62%.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Observations} = \frac{6+25}{50} = 0.62$$

Precision is the proportion of positive observations that are correctly predicted. In other words, how many of the node pair labelled as positive have its actor starring the movie. The precision obtained in this model 56.82%.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} = \frac{25}{25+19} = 0.56\widehat{81}$$

Recall, also known as *sensitivity*, is the ratio of correctly predicted positive observations over all observations predicted as positive class. Particularly, it shows how many of the truly positive node pairs were correctly labeled. For this metric, the value achieved is 100%.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Positives} = \frac{25}{25+0} = 1$$

The **F1 Score** is a metric to balance Precision and Recall, the weighted average of this two metrics. This score takes into account both false positives and false negatives. The F1 score obtained for this particular model is 0.725.

$$F1\ Score = 2 \cdot \frac{(Precision \cdot Recall)}{(Precision + Recall)} = 0.7246$$

Lastly, some predictions with the same positive node pairs of PPR experiment section are performed. To obtain predictions from the Logistic Regression model, a PPR for every node pair must be performed for every feature path. Original graph model must be cloned before removing the relationship between the nodes pair. Then, the PPR can be calculated with the same parameters as before. Prediction results are shown in table 4.6. It has to be noted that the value for each pair and each feature path was 0.0. In other words, PPR did not find the path between the actors and their respective films.

Actor	Movie	Prediction label
Antonio Banderas	*Puss in Boots*	1
Antonio Banderas	*Four rooms*	1
Antonio Banderas	*Once Upon a Time in Mexico*	1
Meryl Streep	*The Hours*	1
Meryl Streep	*The Iron Lady*	1
Meryl Streep	*Death Becomes Her*	1
Ian McKellen	*The Hobbit : Desolation of Smaug*	1
Ian McKellen	*Gods and Monsters*	1
Ian McKellen	$X - Men\ 2$	1

Table 4.6: Prediction label for positive node pairs

Chapter 5

Final results evaluation

It is time to analyze the results of every algorithm predictions. The evaluation has to go beyond the values obtained in the experiments because some results can give contradictory ideas.

As expected, **Personalized PageRank** could not predict the relationship. This method can be considered as *naïve* as it is expected to find the final node by using random walks and no constraints. With an edge joining initial and final node, it is very possible that the final node is visited several times, increasing its rank. But once the relationship between the nodes is removed, it is difficult for the algorithm to find the final node if it has no feature path to be guided for. Additionally, the PPR ranks will depend on how the model is built. For a fully connected graph, the possibilities to find the final node are greater that for one that is not. This is the case of this project graph. Movie and Genre type nodes have the highest degree and can be considered as the main nodes to walk through when touring the model from a Person type node. This statement can be corroborated by looking the top 10 feature paths in table 4.3.

In consequence, removing the relationship of a node with a main node makes its ranks to drastically decrease or become zero, as shown in table 4.2. Not only that but removing one edge from the initial node decreases the rank of other related nodes. The reason for Ian McKellen to have higher ranks is that his node has a lower degree (21) than Banderas (33) or Streep (31). Having fewer options to choose from, the most likely to repeat the same nodes on restart. On the other hand, the more connections a node has, the more chances to be visited from any other node. That is why Banderas' films has higher values than Streep's despite having a lower degree for himself. His films have, in order, 51, 109 and 61 connections and Streep's have 45, 62 and 35, respectively. Furthermore, the high amount of connections of Movie nodes makes possible for McKellen to maintain a small rank value for two movies once the joining edge is removed. For instance, *The Hobbit* has a degree number of 147 and *X-Men 2* has 70. On the contrary, Gods and Monster has only 34 connections.

As seen above, Personalized PageRank algorithm is not an accurate process to predict new relationships for the particular model built in this project. In other cases, like proposing new contacts in social networks, could be useful if the graph model that represents the network is walked every time a neighbor adds a new edge and all the rank values are revised. Nonetheless, this algorithm is useful to understand how graph nodes, edges and paths can be ranked to obtain the most important ones and has been convenient as a basis to learn and use the **Path Ranking Algorithm**.

Results achieved by PRA can be confusing and deceiving. At first, seems that the algorithm always finds the positive node pairs. But, as explained below, this is misleading. Taking a look at logistic regression confusion matrix in table 4.5 seems that the classification problem comes with the negative pairs as true positive are always right, like Recall metric confirms. The high value of this metric makes the F1 Score to have a great value as well. This Score outlines that positive pairs are detected correctly but some negative are labeled as positive. In contrast, Precision metric tells that predicting positive node pairs is like tossing a coin. This metrics describes that almost all the tested nodes where classified as positive where few more than a half were correctly labeled.

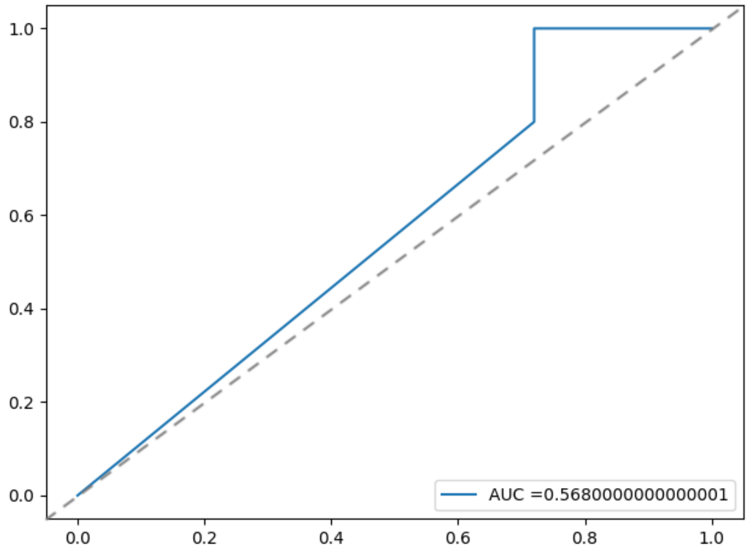


Figure 5.1: Logistic Regression ROC Curve

The best-known metric and easiest to understand is Accuracy. It describes, in a simple way, how well the predicting model is performing. As it has been studied throughout the Master's degree, a model is considered good enough if it has an accuracy equal or greater than 80%. For this particular model, Accuracy outlines that the model is not bad at all but not as good as it should be. This can be confirmed with the Receiver Operating Characteristic curve. The ROC curve in figure 5.1 shows that model have some difficulty to discriminate whether a relation should exist or not. The curve is almost parallel to the fifty percent diagonal but keeps a distance that grows very slowly. What is more, another metric that tells that this predicting model is not good enough is the Area Under the Curve (AUC) value. If this value is 0.5 or it is closer to this number, as is the case, it means that the model has no discriminatory capacity.

The main reason to have obtained this poor classifying model are the rank values obtained in PRA. As mentioned before, it is really complex and not guaranteed to reach the final node using PPR with a constrained path. For a not fully connected graph, the chances are even lower. As a consequence, almost all matrix values obtained are 0.0. For this reason, the Logistic Regression model ensures that if every value for every feature path of node pair is 0.0, then the label should be 1. That is why this model is considered as deceptive. Instead of achieving some values to be considered as threshold to determinate if a relationship should exist or not, it considers that if no values are obtained the relationship should exist by default.

Chapter 6

Conclusion

Once this project is completed, it is time to evaluate if the objectives has been met and the lessons learned. The main purpose of this work was not only to show was has been studied during the Master's degree but to gather new knowledge by exploring new data models. Despite that the prediction models have not proven to be as good as expected, positive lectures can be extracted from this work.

6.1 Project results

In first place, working with graph models is not as easy as it can seem at the beginning. The most important thing to take into account is how well the graph is built. For this project, the poor connections in some points and the high number of nodes made it very difficult to obtain good rank values while random walking. Furthermore, some type of nodes –like Movies, Genres, Persons or Genders– seemed to be the only connection points with the rest of the graph. Figure 6.1 shows a good example of this. In this graphic, paths seem to come together in Movie nodes and in Genre nodes also. As seen, high degree on nodes can be a double-edged sword. On one side, when starting the random walk, it is difficult to visit the same node if there are lots to choose from; on the other hand, while walking through the graph, the node has a bigger chance to be visited from anywhere. Having a well-balanced graph is not trivial.

Regarding the algorithms used, PPR has been useful to understand how some recommendation systems works. Despite the fact of not being favorable as a relation predictive model, knowing how it works is the base to understand PRA. Path Ranking Algorithm is a really interesting process to try to predict new relationships. After having studied it, there is no doubt that this two-step process is one of the best algorithms to predict relationships in a graph. The problem with it is the amount of resources it needs. Executing this process sequentially on a

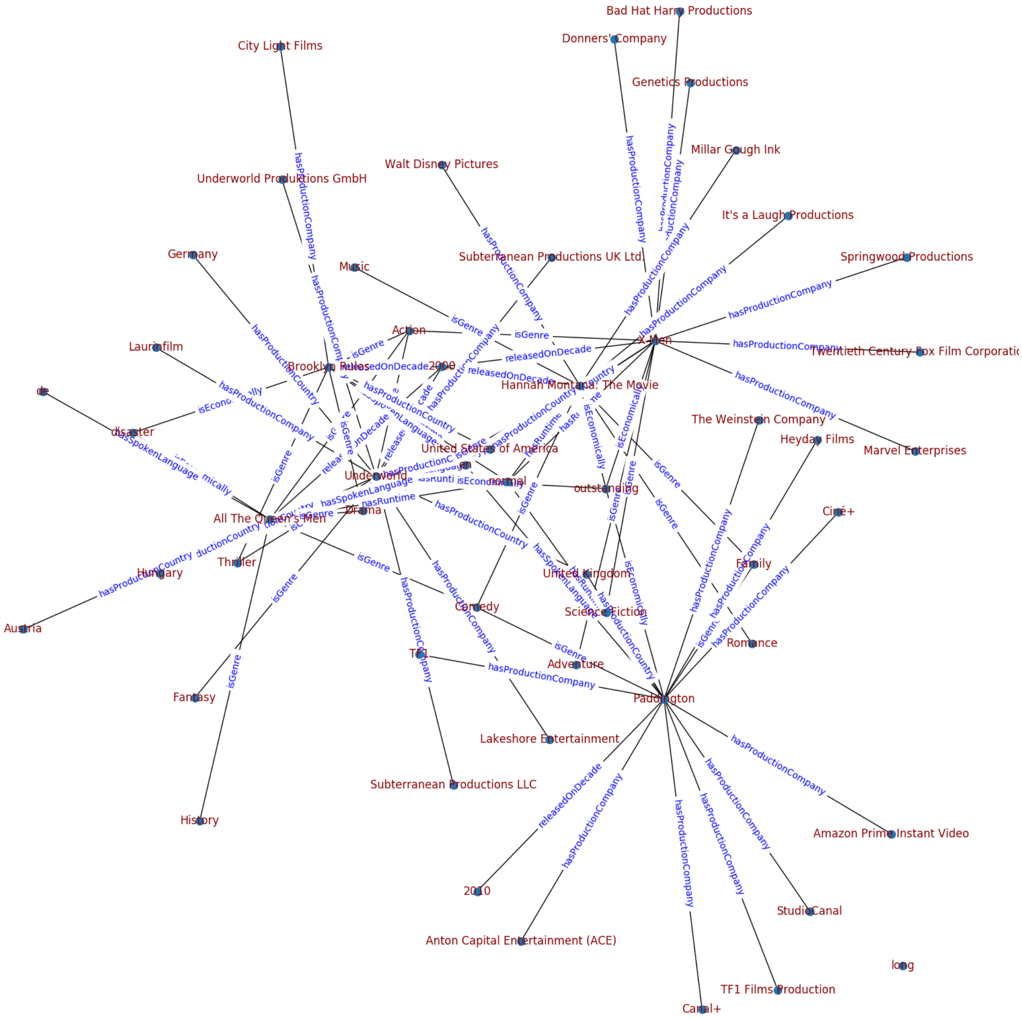


Figure 6.1: Graph model with information of six random films

domestic computer is very time consuming. It would have been good to execute this algorithm with thousands, or even millions, of random walks. But for this case, dedicated and scalable machines –like Amazon Web Services ParallelCluster– should be used. This way, several of random walks could have been executed at the same time on different machines. In consequence, thousands of feature paths would have been selected and the top ten paths would have been clearer and more useful. Needless to say, the rank values stored in resulting matrix would have been different. With any non-zero value, the Logistic Regression model would have learned to discriminate in a much better way instead of having values that train a model with a large bias after waiting hours to conclude.

6.2 Future lines of work

As commented in previous section, one future line of work is getting non-zero values for PRA resulting matrix. This could be done in two different ways: on one hand, as outlined before, using dedicated scalable machines to perform random walks jobs in parallel; another way is to reduce the amount of data that is introduced in the graph. But this last option means to discard even more attributes that are considered influential. Instead of discarding more fields, it would be better if the dataset had facts from the last ten or twenty years. This way, predictions for current time could be more accurate.

Another process to predict new relationships that should be tested in a near future is the Subgraph Feature Extraction (Gardner & Mitchell, 2015). As described in chapter 2, this algorithm is an evolution of PRA but changing the second step to another that, in theory, is less resource consuming. In this process, the feature path extraction is made by using subgraphs centered around nodes. Subgraph depth is determined in advance. Every walk leaving the node can be considered as a small path. Once enough subgraphs are obtained, different node subgraph paths are compared. Those that share common nodes are merged and gathered as feature paths. Additionally, the interesting investigation of Bogers (2010) should be tested as well. As outlined before, the process can be extended to genre or actor recommendation and, from that point, modified for be used as an edge predictive model.

Added to that, textual attributes that has been discarded –like *tagline* and *overview*– could be taken into account. This kind of data should be treated with Natural Language Processing (NLP). By using this process, the text can be labeled with a particular feeling: positive, negative, neutral, sadness, aggressiveness, etc. This way, a new kind of node type can be created and shared by Movie nodes. At the same time, the same textual attribute could be compared for several entities and try to figure out their similarity. Then, a threshold could be determined, as 80% or greater similarity between texts, to connect different movies through a new node.

Last but not least, another way to represent the graph could be considered. Graph databases, like Neo4J, make model exploration easier. Although this database has its own viewer, it has a good group of third-party tools to render the model. Additionally, it would be a good idea to add pictures for every actor. Most of the people do not seem to remember their names, but their faces or the characters they have played.

6.3 Lessons learned

For a cinema lover, the TMDb dataset is delightful. It has a lot of interesting facts. Discarding some of its attributes is always painful but if not the complexity of the project would grow exponentially. Despite discarding some of them, the graph model has become huge. The dataset heterogeneity is not as good for a data scientist as it is for a fan. As outlined in previous section, to predict new actual relationships should have been better to have half the number of movies on the dataset but from the last ten or twenty years. There is no sense to take into account actors that has passed away or production companies that does not exist anymore. Moreover, consumer tastes change through time and they are more similar to those of twenty years ago than those of a century ago. Anyway, this heterogeneous dataset can show some interesting facts for a cinephile when looking to nodes degrees. For example, Gender male degree (25729) is almost double than female (13166); Drama is the most connected genre with 2292 edges while the second one is comedy with 1718; almost all the movies has a normal runtime (4467) and the most connected node when talking about benefits margin is Outstanding with 1804 connections. Data analysis –where data scientists spend the 80% of the time– has been time consuming but very comforting as well.

Converting the dataset into a graph model has been an easy task thanks to NetworkX library for Python, as seen in section 4.1. Its possibility to render the graph has been very useful to understand nodes connections. On the contrary, some functions are not as intuitive as they should be. For instance, getting a node degree is performed by getting an array of the degrees of all nodes and then selecting the target node by its index. Getting neighbor nodes of a particular type was tricky as well. Nevertheless, the library has come in handy and it has pushed me to sharpen my knowledge about Python.

On top of that, reproducing PPR and PRA algorithms has improved my algorithmic proficiency as well. Personally, make algorithms efficient and using less computer resources is a challenge always sought. In this line, Python delights any developer. For instance, its capacity to return several values per function makes coding much easier. Definitively, it is the recommended coding language to be used in data science projects.

Before finishing, it should be noted that other secondary objectives have been met. Learning LaTeXwas easier than expected at first moment. Of course, having knowledge in other markdown languages as HTML is an advantage. Additionally, having such a great community behind that develops interesting packages and add-ons, makes this document preparation system very worthwhile. From now on, it will be my editor for professional documentation.

Regarding the English language, this project has enlarged my vocabulary. Indeed, most of the technical words are learned throughout the Master's degree as they are shown in English. But searching for their synonyms so the text was not repetitive has been tough but gratifying.

In conclusion, despite the fact that predictive models built have been a little disappointing by the results they threw, the overall process has been satisfying. Creating the best predictive model for cinema industry was not guaranteed, although a big effort has been made to achieve it. However, elaborating this project has taught me new concept and skills in many different areas and have strengthened my passion for science.

Bibliography

Bogers, T. (2010). Movie recommendation using random walks over the contextual graph. In *Proc. of the 2nd Intl. Workshop on Context-Aware Recommender Systems.*

Chakrabarti, S. (2007). Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (pp. 571–580)., New York, NY, USA. ACM.

Clements, M., de Vries, A. P., & Reinders, M. J. T. (2008). Optimizing single term queries using a personalized markov random walk over the social graph. In *Workshop on Exploiting Semantic Annotations in Information Retrieval (ESAIR).*

Diao, Q., Qiu, M., Wu, C.-Y., Smola, A. J., Jiang, J., & Wang, C. (2014). Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars). In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 193–202). ACM.

Gardner, M. & Mitchell, T. (2015). Efficient and expressive knowledge base completion using subgraph feature extraction. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (pp. 1488–1498)., Lisbon, Portugal. Association for Computational Linguistics.

Haveliwala, T. H. (2002). Topic-sensitive pagerank. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, (pp. 517–526)., New York, NY, USA. ACM.

Jiang, X., Tresp, V., Huang, Y., & Nickel, M. (2012). Link prediction in multi-relational graphs using additive models. In *Proceedings of the 2012 International Conference on Semantic Technologies Meet Recommender Systems &#38; Big Data - Volume 919*, SeRSy'12, (pp. 1–12)., Aachen, Germany, Germany. CEUR-WS.org.

Langville, A. N. & Meyer, C. D. (2004). Deeper inside pagerank. *Internet Mathematics*, *1*(3), 335–380.

Lao, N. & Cohen, W. W. (2010). Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, *81*(1), 53–67.

Nie, Z., Zhang, Y., Wen, J.-R., & Ma, W.-Y. (2005). Object-level ranking: Bringing order to web objects. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, (pp. 567–574)., New York, NY, USA. ACM.

Oscars (2019). *Rules and Elegibility* (92 ed.). Academy Awards of Merit.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120.

SAGA (2019). *Elegibility Criteria* (26th ed.). 5757 Wilshire Blvd., Los Angeles, CA 90036: Screen Actors Guild Awards.

Taskar, B., Abbeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI'02, (pp. 485–492)., San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Taskar, B., Wong, M.-F., Abbeel, P., & Koller, D. (2003). Link prediction in relational data. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, (pp. 659–666)., Cambridge, MA, USA. MIT Press.

TMDb (2017). The movie database 5000 movie dataset.

Viswanathan, A. P. (2015). Movie success predictor. *International Journal of Computer Trends and Technology (IJCTT)*, *28*(2), 81–82.