A Movie Night with Machine Learning

Michail G. Pachilakis
csd3077@csd.uoc.gr

Iordanis P. Xanthopoulos
csd3161@csd.uoc.gr

Abstract

As the number of movies released continuously grows, viewers are flooded with information about several movie productions, making the simplest questions like "What movie to see tonight?" really hard to answer. Also movie production studios would like to know if a movie could be a commercial success in order to invest money in it.

In this work **i.** we provide a simple question interface so the users can find movies matching on his/her creteria, **ii.** we provide an interface to answer statistic related questions about any movie dataset and **iii.** we provide predictions about a movies imdb score based on Machine Learning.

Keywords Machine Learning; imdb; prediction; spark; statistics; recommendations

1 Introduction

A paragraph of text goes here. Lots of text. Plenty of interesting .test.text.

More fascinating text. Features[1] galore, plethora of promises.

2 Dataset

The dataset was created at 2016 by [NAME], who used it on an article at [SITE NAME][CITE] and can be found, copyright-free and free of charge, at Kaggle[CITE]. The dataset is consisted of 5043 lines, each line representing a movie, and 28 columns, depicting its characteristics. The size of the dataset is 579.78KB and the format of the data is text, containing both alpharithmetic and numeric values.

Since the data are originated from IMDB, we do not expect any biases, regarding the scores and the validity of movie characteristics such as the actors' lineup and the director. Still, even though the dataset has enough data for the small-scaled experiments we conducted, some of its flaws are mentioned by its creator and we ought to point them out as well. In some cases, the columns have blank fields, which gave an error output, if left unhandled prior to the construction of the ML model. Moreover, the currency depicting the gross income of the movies is, in some cases, not US dollars, but the local currency used in the country the movie is originated from (i.e. Chinese movies' gross has the Chinese RMB currency). The inflation factors are not taken into consideration, it goes without saying that 1000US dollars did not have the same value at 1920s as they do nowadays, and we do not take any particular actions regarding this matter, due to the high complexity of the conversion and the little impact to the resulting ML model.

Below, we present the 28 characteristics of the datasets, with a minor description for each one as to its type and what it describes:

- **color**: Binary alpharithmetic variable with possible values Color or Black and White.
- **director_name**: Categorical alpharithmetic variable, with its value describing which is the director of the movie.
- **num_critic_for_reviews**: Numeric variable, with its value describing how many reviews are written for the movie by critics.
- **duration**: Numeric variable, with its value describing how long the film lasts.
- **director_facebook_likes**: Numeric variable, with its value describing how many likes the director of the movie has on facebook.com.

- **actor_3_facebook_likes**: Numeric variable, with its value describing how many likes the third actor (in lineup order) has on facebook.com .
- **actor_2_name**: Categorical alpharithmetic variable, with its value containing the name of the second actor (in lineup order).
- **actor_1_facebook_likes**: Numeric variable, with its value containing how many likes the first actor (protagonist) has on facebook.com .
- **gross**: Continuous numeric variable, with its value containing the total gross income of the movie.
- **genres**: Alpharithmetic variable, contains a number of predefined strings, such as Action, Adventure.
- **actor_1_name**: Alpharithmetic variable, contains the fullname of the first actor (protagonist).
- **movie_title**: Alpharithmetic variable, with its value containing the name of the movie.
- **num_voted_users**: Numeric variable, with its value containing the number of users who rated this film.
- **cast_total_facebook_likes**: Numeric variable, with its value containing the total number of likes of the whole cast in the facebook.com .
- **actor_3_name**: Alpharithmetic variable, contains the full name of the third actor.
- **Plot_keywords**: Alpharithmetic variable, contains the plot keywords separated by lines "|".
- **Movie_imdb_link**: Alpharithmetic variable, contains the IMDB link for the movie.
- **language**: Alpharithmetic variable, contains the main language that the actors speak in the movie.
- **actor_2_facebook_likes**: Numeric variable, with its value describing how many likes the second actor (in lineup order) has on facebook.com .
- **num_user_for_reviews**: Numeric variable, with its value describing how many reviews are written for the movie by users.
- **face_num_in_poster**: Numeric variable, with its value describing how many actor faces are in the movie poster. **Not applicable to our model**
- **movie_facebook_likes**: Numeric variable, with its value describing how many likes the movie has on facebook.com.
- **country**: Categorical alpharithmetic variable, contains the name of the country that the movie was shot.
- **content_rating**: Categorical alpharithmetic variable, describing the viewers minimum age which the movie is appropriate.
- **budget**: Numeric variable, with its value describing how much money the movie production costs.
- **title_year**: Numeric variable, with its value describing when the movie released in cinemas.
- **imdb_score**: Numeric variable, with its value describing when the movie released in cinemas.
- **aspect_ratio**: Numeric variable, with its value describing the aspect ratio of the movie.

3 Implementation

We select to implement three different functionalities. The first one targets mostly the movie production studios and is about movie score predictions. The other two are statistics about the movies contained in the dataset and movie recommendations and they mostly target the simple viewers.

For our implementation we used apache spark and the scala programming language. In the subsection 3.1 is described how we implement the movie predictions and in the subsetion 3.2 are described the implementation of movies statistics and the recommendations.

3.1 Machine Learning

We implementated the movie score prediction using Machine Learning for this reason we select to use apache spark and scala. Spark provided us with MLlib a scalable machine learning library consisting common learning algorithms and utilities. Also it provides us with spark.ml which aims to provide a uniform set of high-level APIs that help users to create and tune a practical machine learning pipelines.

Firstly we created a case class containg 28 fields, as many as the dataset features, and after parsing the initial dataset we set every feature to it's coresponding class field. We removed from the dataset movies containing commas in their titles because during splitting they generated more features than they should and they generated errors on our comma seperated csv file.

From our initial 28 features we select to keep only those who had meaning for our predictions, so features like "Title", "IMDBLink", "Country" etc. were dropped. We dropped in total 11 features that we thought it would had the least affect in the movie score prediction.

Because our dataset contained a mix of string, double and integer features we had to transform our string features to numeric values. The package spark.ml provides us with the StringIndexer function. StringIndexer converts String values into categorical indices which could me used by machine learning algorithms in ml library.

From our remaing 17 features we had to select only those that would positevly affect prediction of the movie imdb score. To achieve this we correlated our remaining features and excluded those which negatively affected the "imdb_score" feature. Again the MLlib package provided us with the Statistics package which contained the necessary functions for the feature correlation.

In order to create our machine learning pipeline we had to create some transformers to produce our output dataframe. We created a VectorSlicer which takes a vector as input column and creates a vector containing only the features that we set from the original vector. We also created a StandardScaler which is used to scale a vector to a vector which values are in similar scale and a VectorAssembler which is a feature transformer that merges multiple columns into a vector column.

We initialize the Linear Regression Classifier estimator and then chain it with the VectorSlicer, StandardScaler and VectorAssembler transformers in the machine learning pipeline. When the pipeline is used to fit training data, the transformers and the estimator will apply to the dataframe with the sequence defined in the pipeline.

3.2 Statistics and Recommendations

For the Statistics and Recommendations we used map and reduce functions. We used the same case class we described in the section 3.1 to create a RDD with 28 features named **parsedPointsRdd**. Then we applied filter and ordering functions on it to produce the recommendations. For the statistics we applied filter functions to the **parsedPointsRdd** and then calculated the statistics.

For statistics we defined the following functions:
`GreatMoviesStatistics():Double`

`DirectorStatistics(`*director* : *String*`):Double`

`GerneStatistics(`*gerne* : *String*`):Double`

The **GreatMoviesStatistics** calculates the percentage of the movies that have IMDB score above 8, these are movies that considered top and everyone should probable see them. The **DirectorStatistics** calculates the percentage of movies that a specific director have directed. The **GerneStatistics** calculates the percentage of movies included in a specific gerne.

For recommendations we defined the following functions:

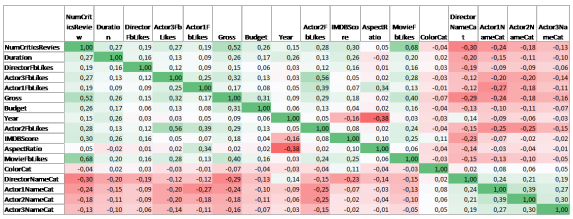

Figure 1: heatmap of the correlation matrix

`BestMoviesInCategory(`*category* : *String*`)`

`findTopTen (`*category* : *String*, *language* : *String*, *actor*1*name* : *String*`)`

`findTitle(`*category* : *String*, *language* : *String*, *actor*1*name* : *String*, *directorname* : *String*`)`

The **BestMoviesInCategory** prints the top ten movies in a gerne, specified by the user. The **findTopTen** prints the top 10 movies based on the gerne, the movie language and the main actors name, these arguments are user specified. Finally the **findTitle** prints the titles of the movies that match some user specified creteria. The user can specify the gerne of the movie, the language of the movie, the main actors name and the directors name. If some of that arguments are not known by the viewer it can be left empty by just passing as argument the empty string "". More than one movies can fit the above creteria so we limit the function to return the first ten movie titles based on their imdb score ordering.

4 Results

5 Conclusion

A polite author always includes acknowledgments. Thank everyone, especially those who funded the work.

6 Availability

You can download the code we used for the machine learning predictions, recommendations and statistics from

`https://github.com/mipach/imdb/tree/master/src`

You can run our code with little or no modications at all, using apache spark. This code is under no license so you can use it free but remeber to refer our work.

References

lalalalala