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

Some embedded literal typset code might look like the following :

```
int wrap_fact(ClientData clientData,
              Tcl_Interp *interp,
              int argc, char *argv[]) {
    int result;
    int arg0;
    if (argc != 2) {
        interp->result = "wrong # args";
        return TCL_ERROR;
    }
    arg0 = atoi(argv[1]);
    result = fact(arg0);
    sprintf(interp->result,"%d",result);
    return TCL_OK;
}
```

Now we're going to cite somebody. Watch for the cite tag. Here it comes [**?**, **?**]. The tilde character (˜) in the source means a non-breaking space. This way, your reference will always be attached to the word that preceded it, instead of going to the next line.

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

	NumCriticsReview	Duration	DirectorFbLikes	Actor3FbLikes	Actor1FbLikes	Gross	Budget	Year	Actor2FbLikes	IMDBScore	AspectRatio	MovieFbLikes	ColorCat	DirectorNameCat	Actor1NameCat	Actor2NameCat	Actor3NameCat
NumCriticsRevies	1,00	0,27	0,19	0,27	0,19	0,52	0,26	0,15	0,28	0,30	0,05	0,68	-0,04	-0,30	-0,24	-0,18	-0,13
Duration	0,27	1,00	0,16	0,13	0,09	0,26	0,17	0,26	0,13	0,26	-0,02	0,20	0,02	-0,20	-0,15	-0,11	-0,10
DirectorFbLikes	0,19	0,16	1,00	0,12	0,09	0,15	0,06	0,03	0,12	0,16	0,01	0,16	0,03	-0,19	-0,09	-0,09	-0,06
Actor3FbLikes	0,27	0,13	0,12	1,00	0,25	0,32	0,13	0,03	0,56	0,05	0,02	0,28	-0,03	-0,12	-0,20	-0,20	-0,14
Actor1FbLikes	0,19	0,09	0,09	0,25	1,00	0,17	0,08	0,05	0,39	0,07	0,34	0,13	-0,01	-0,12	-0,27	-0,18	-0,11
Gross	0,52	0,26	0,15	0,32	0,17	1,00	0,31	0,09	0,29	0,18	0,02	0,40	-0,07	-0,29	-0,24	-0,18	-0,16
Budget	0,26	0,17	0,06	0,13	0,08	0,31	1,00	0,06	0,13	0,04	0,02	0,16	-0,04	-0,13	-0,10	-0,11	-0,07
Year	0,15	0,26	0,03	0,03	0,05	0,09	0,06	1,00	0,05	-0,16	-0,38	0,03	-0,03	0,14	-0,09	-0,06	-0,03
Actor2FbLikes	0,28	0,13	0,12	0,56	0,39	0,29	0,13	0,05	1,00	0,08	0,02	0,24	-0,04	-0,15	-0,25	-0,25	-0,15
IMDBScore	0,30	0,26	0,16	0,05	0,07	0,18	0,04	-0,16	0,08	1,00	0,10	0,25	0,11	-0,23	-0,07	-0,02	-0,02
AspectRatio	0,05	-0,02	0,01	0,02	0,34	0,02	0,02	-0,38	0,02	0,10	1,00	0,06	-0,04	-0,14	-0,03	-0,04	-0,01
MovieFbLikes	0,68	0,20	0,16	0,28	0,13	0,40	0,16	0,03	0,24	0,25	0,06	1,00	-0,03	-0,15	-0,13	-0,10	-0,05
ColorCat	-0,04	0,02	0,03	-0,03	-0,01	-0,07	-0,04	-0,03	-0,04	0,11	-0,04	-0,03	1,00	0,02	0,08	0,06	0,05
DirectorNameCat	-0,30	-0,20	-0,19	-0,12	-0,12	-0,29	-0,13	0,14	-0,15	-0,23	-0,14	-0,15	0,02	1,00	0,24	0,21	0,19
Actor1NameCat	-0,24	-0,15	-0,09	-0,20	-0,27	-0,24	-0,10	-0,09	-0,25	-0,07	-0,03	-0,13	0,08	0,24	1,00	0,39	0,27
Actor2NameCat	-0,18	-0,11	-0,09	-0,20	-0,18	-0,18	-0,11	-0,06	-0,25	-0,02	-0,04	-0,10	0,06	0,21	0,39	1,00	0,30
Actor3NameCat	-0,13	-0,10	-0,06	-0,14	-0,11	-0,16	-0,07	-0,03	-0,15	-0,02	-0,01	-0,05	0,05	0,19	0,27	0,30	1,00

Figure 1: heatmap of the correlation matrix

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

`findTopTen` (*category* : *String*, *language* : *String*, *actor1name* : *String*)

`findTitle`(*category* : *String*, *language* : *String*, *actor1name* : *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

In this section we present our machine learning results and evaluate our methodology. We describe in more depth the the correlation process and discuss our evalutation.

After loading our dataset and created our dataframe contaiining the 28 movie features we drop the features that we though it will have the least impact with the imdb score feature. These features was: "Gerne", "Title", "NumVotedUsers", "CastTotalFbLikes", "FacesOnPoster", "PlotKeywords", "IMDBLink", "NumUserReviews", "Language", "Country" and "ContentRating". From these features some can not been created before the movies release sush as "NumVotedUsers" and others can not affect the movie score sush as "IMDBLink".

We also need to transform the dataframe to convert the String features into numeric features. Then we procced to the correlation which is done using the "pearson" method. In figure 1 we present the heatmap extracted by the features correlation. The more green a cell is the more possitive it correlates with the feature in that row, and the more red a cell is the more negative weight gives in the correlation. Looking at the "IMDBScore" row of the matrix we found that features as "Year", "Actor2Name", "DirecotrName", "Actor1Name" and "Actor3Name" have a negative weight in the imdb score so we drop them.

Using randomSplit we split our dataframe to two datasets. A train dataset containing 80% of the movies in our initial dataset and a test dataset containing the remaing 20%. We produce our pipeline model by fitting the training data to our pipeline. The testing data will go though our model to produce the prediction results.

To evaluate our data we created a val RegressionEvaluator which we feed with our previous predictions. The room mean square error we had was 1.02 which considering our features and that the rating scale is from 0 to 10 is acceptable.

We also tested another set of features to make predictions. Based on the Chuan Sun work on the movie predictions we used another set of features. This set includes the following features: ”IMDBScore”, ”DirectorFbLikes”, ”Duration”, ”Actor1FbLikes”, ”Actor2FbLikes”, ”Actor3FbLikes”, ”FaceNumOnPostes”, ”Year”, ”Color”, ”Budget”. Using our model, because we do not know exactly the parameters used in his model, we had root mean square error 1.04 using these features.

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