Andrew Zinkan

5315 Big Data Management

3/14/2022

**Use Case Name**

“Ripe-Pumpkin: An Approach to Collaborative Filtering For Movie Recommendations”

**Use Case URL**

GIT HUB LINK

**Introduction**

Ripe-Pumpkin a new start up, is an entertainment focused company rooted in movie review aggregation services. Currently they are in the position to review what they trademark as the ‘Pumpkinmeter’, which is a catchy way of adding a score the movies they’d recommend an individual see. This recommender while a simple prescriptive output to the end user is produced by a sophisticated measurement of collaborative recommendations by millions of users. By using this type of collaborative filtering to produce recommendations can lead to accurate and customized movie suggestions for an individual. This feature hopes to add value to the end customer by displaying only tailored results they will likely enjoy. Ripe-Pumpkin is currently reviewing the potential benefit associated with a move review-aggregation service. Across the industry Ripe-Pumpkin has seen the recent success of others that have started providing suggested streaming content. Ripe Pumpkins realized they too need to make a business conscious decision and evaluate the potential of introducing their own the ‘Pumpkinmeter’ to increase their capabilities as a movie recommendation service.

**Dataset used**

Data used for this analysis is provided from GroupLens, a research group that collects ranking & preference data from their partner MovieLens and makes it available for public consumption. The data collected in ‘ml-latest.zip’ includes 27,753,444 rating records, for 283,228 unique user profiles, and 58,098 unique movies (Harper & Konstan, 2015). The data has been collected over various periods of time and stored as structured data in the form of comma separated data files (CSV), broken up by ratings and movie content. Specifically applicable to this case study are the ‘Movies.csv’ file (Figure 1), and the ‘Ratings.csv’ file (Figure 2). These files contain key pieces of data that are at the core of this analysis, specifically ‘Movies.csv’ captures the ‘movieid’, ‘title’, and ‘genres’ of the movie, while the ‘Ratings.csv’ file contains pertinent information regarding ‘userId’, unique ‘movieId’, ‘rating’ of the film, and ‘timestamp’ represented as seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970 (Harper & Konstan, 2015). In short, these two files provide a massive amount of data that is directly applicable to the problem at hand and will help insights about individuals and their movie preferences. For additional information and data dictionary please refer to [grouplens.org](https://files.grouplens.org/datasets/movielens/ml-25m-README.html).

**Technical Details**

To develop a movie recommender, we need to use a machine learning model, in this case a collaborative filter model. This model uses the pyspark library, and its pyspark.mllib.recommendation import of the Alternating Least Squares (ALS) package to do predivitive modeling through a RDD collaborative filtering approach. This methodology is a machine learning approach that essentially identifies the nearest neighbors based on similarity (*in this case movie rating*). “The underlying assumption is that if a user A has the same opinion as a user B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a user chosen randomly” (Dianes, 2015). For a simplified example, Bob likes the same 10 movies that Jim does, and Bob hates the same 10 movies that Jim does. Therefore, if there is a movie Jim likes that Bob hasn’t seen, its highly likely that Bob will like that movie too, seeing that Bob ad Jim have similar preferences. This ML approach searches for the most similar profiles based on movie preference and then recommends their likeness score or ‘Pumkinmeter’ as a reasonable expectation (or prediction) that someone similar with like it too. Pysparks’ powerful RDD collaborative filtering approach is widely used as a common recommendation model and is likely very similar to what Ripe-Pumpkin’s competitors are doing too.

**Debugging Details**

During this project there were a few challenges that came up along the way, primarily due to the size of the data and memory required for pysparks temporary storage. When doing training on a data set consisting of 27,753,444 records in local mode, even taking a 60/20/20 train, validate, and test split lead to resource constraints and raised low disk space errors. I first started out with a t2.mico EC2 instance with a storage size of 30GiB initially. This instance type did not seem to be able to handle the data size so I later adopted a t3.large EC2 instance which substantially helped run time. While this seemed to help increase the run time it still didn’t fix the disk space or temporary files size constraints. So, in order to decrease model run time further, and in an attempt to minimize the temp data storage I reduced the data set. To reduce the data set I selected to use a random sample of responses, and because the data set and problem at hand does not seem to be related to a time series problem a truly random and unordered sample would be sufficient. Initially I tried to use a random 60% of the total data set, but continued to use a smaller and smaller sample sizes as the space constraints persisted. Eventually I landed on only using 10% of the overall data leaving approximately 2.77 million records for training, validating, and testing. While this is a small representation of the overall population this random sampling without replacement would imply that I get a mean sample population centered around the mean of my total population. Essentially meaning that my sample data should be representative of the total population, and that our model could be applied to the rest of the population with similar performance. While we would prefer to use as much data as possible to do training, we would need to expand the resources available on our EC2 instance further and not run pyspark in local mode.

**Results**

My model when tested on the test dataset resulted in root-mean-square error (RMSE) of 0.982 indicating that the model developed on the training data results in low performance and a high degree of difference between our observed and predicted values. Essentially this tells us that the model does not help us much when trying to accurately predict movie preferences of individuals. Typically for this type (or any ML model) the more data that you can provide it the more observations it has to recognize a pattern. Had I been able to process the full data set for all 27M records this model may have received higher predictive accuracy, however this alone would not drastically increase its predictive power in a business context, indicating that there are likely model adjustments I should continue to pursue.

As for the scenario results, I found that in scenario 1 *My* movie preferences for the >=25 reviews (Figure 4) were less appealing to me than the >= 100 reviews results (Figure 5). In the attached figures you can see the highlighted records which are movies I like, in these cases the model reformed correctly. Its visibly apparent that in the results that used movies with more than 100 reviews (Figure 5) that there are more recommendations that I would consider a true positive (*The model accurately identified what like*). The same pattern was noticed for my second user. Intuitively and mathematically, this makes sense because the more reviews and more positive feedback a movie receives the more likely another individual is to like it as well, thus the more likely the recommendation model is to recommend it. Example: if 100 people with different movie taste all like a single movie, and 1 person dislikes it, I have a higher probability of being more like one of the 99 (*movie likers*) than the 1 (*movie disliker*). Because preferences are built based on association our model will tend to steer us to more popular results because there is a higher probability, I like it too. This can create a feedback loop that is seen in our world as “going viral”. Looking at Netflix for example, once a movie becomes supper popular “Squid Games” for example, Netflix’s algorithm continues to make that recommendation to others. As more recommendations are created more views and reviewers are created there by reenforcing this feedback loop to recommend it to more users.

**Insight**

With the lack of promising results from the initial model, no established or reoccurring data feed or API connection, and no mention of how to deploy this product within the platform it seems that Ripe-Pumpkin needs to reevaluate the implementation of this model into their business strategy. This is not to say that the model is a failure by any means, however, there needs to be additional research on the topic and additional features need to be added to the model to increase is predictive potential. As is, this model would not bring substantial value to that of the company and or its end user do the high error rate in the models testing. One area worth looking into for Ripe-Pumpkin would be to test the predictive power of the model base on binary results or taking a rounded version of the prediction for testing accuracy. For example, observed rating from users always rank from 1 – 5 in 0.5 increments, you would never see a rating for ‘3.467’ from an individual user. Because the data is structured as such it would be worthwhile testing the question “Can we simply recommend a movie you will like (>3 *like*, <= 3 *dislike)?*” vs the current question which is more centered around how accurate is the predicted score vs the real score? This reevaluation of the business question maybe critical, as ratings are not normally distributed (Figure 7). Additionally, to increase model predictive ability and suggest more relevant recommendations it would be wise for Ripe-Pumpkin to do some outlier removal for certain movie choices with few reviews. By providing recommendations that have reviews by many we are more likely to get better movies rather than allow a handful individuals to sway it rating for niche films. Lastly while I would not be ready to give up on the collaborative filtering ALS approach just yet there are other models that might be worth looking into that may produce superior results such as matrix factorization models, which harness additional information such as implicit feedback, temporal effects, and confidence levels (Koren, Bell, & Volinsky 2009). It might also be worth while to explore a Hybrid Deep Learning algorithm centered around a Content-Based models (ITNEXT, 2020). This type of model would make recommendations to a user based on the content of the film instead of what other users are watching, this however would require entirely different data to be available to Ripe-Pumpkin’s use.

**References** (at least 3 references including the main project source, lecture, presentation, etc.)

Dianes, J. 2015 *Building a Movie Recommendation Service with Apache Spark & Flask-Part 1:* CodeMentor.io <https://www.codementor.io/@jadianes/building-a-recommender-with-apache-spark-python-example-app-part1-du1083qbw>

F. Maxwell Harper and Joseph A. Konstan. 2015. *The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19.* <https://doi.org/10.1145/2827872>

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. *Matrix Factorization Techniques for Recommender Systems. Computer 42, 8 (August 2009), 30–37*. <https://dl.acm.org/doi/10.1109/MC.2009.263>

ITNEXT 2020 *What are today’s top recommendation engine algorithms?* <https://itnext.io/what-are-the-top-recommendation-engine-algorithms-used-nowadays-646f588ce639>

**Appendix**

**Figure 1**: -- Movies.csv sample as a data frame

Table

Description automatically generated with low confidence

**Figure 2:** -- Ratings.csv sample as a data frame

Table

Description automatically generated

**Figure 3:** -- Number of records for each file, average rating for the population, and the number of unique contributors adding to each ranking section. Note that whole numbers tend to have more common selections.

Text

Description automatically generated

**Figure 5:** -- My movie recommendation results with more than 25 ratings

Text

Description automatically generated

**Figure 6:** -- My movie recommendation results with more than 100 ratings

Text

Description automatically generated

**Figure 7:** – Non normal distribution of moving ratings (Left Hand Skew)

Chart, histogram

Description automatically generated