PROBING THE NETFLIX PRIZE:
Recommended Algorithms for Recommender Systems

DS 5110: Big Data Systems | Final Project

Lauren Neal *(ln9bv)* | Nima Beheshti *(nb9pp)* | Nick Thompson *(nat3fa)* | Melanie Sattler *(ms9py)*

ABSTRACT

Present-day recommender systems are responsible for infusing consumerism with an almost-intimate degree of personalization. It is no longer enough to simply provide the user, the consumer, the subscriber with a dearth of choices; in order to secure a large slice of the attention economy and develop brand loyalty, companies are tasked with not only providing customers what they want, but also suggesting additional options based on individual preferences and previous selections. For this project, we secured the original data set used in the Netflix Prize Challenge, which ran from 2006-2009. We examined the ability of this data to predict how users would rate titles they had not already viewed. Variables included only user ID, title ID, and ratings; additional features (year, runtime, genre) were mapped onto the original data where possible. We experimented with linear regression, ridge regression, lasso regression, K-Means classification, and Alternating Least Squares. We conclude that classical linear (including lasso and ridge) regression and K-means classification models are ill-equipped to predict user-item or user-product ratings, especially when the available features lack heterogeneity. The non-negative matrix factorization of the Alternating Least Squares algorithm was the only method that provided substantive, sensical results: the lowest resulting RMSE (0.852610) at once met the criteria posed by the original Netflix Prize Challenge (to beat Netflix's then-best RMSE of 0.9525 by 10%) and bested BelKor's $1MM bounty-winning RMSE of 0.8558.

INTRODUCTION

Our team sought to better understand the evolution of methods employed in the burgeoning and now-ubiquitous business of recommender systems. Algorithmic recommendations have come to dominate myriad aspects of how consumers consume: from media (social, music, film/TV) to market commodities (clothing, skincare and hair products, etc). TikTok became the most popular app of the pandemic due to its addictive algorithmic design, which sacrifices aesthetics in favor of algorithmically optimal attributes. In July of 2021, TikTok announced it would be selling its algorithm for use by other developers.

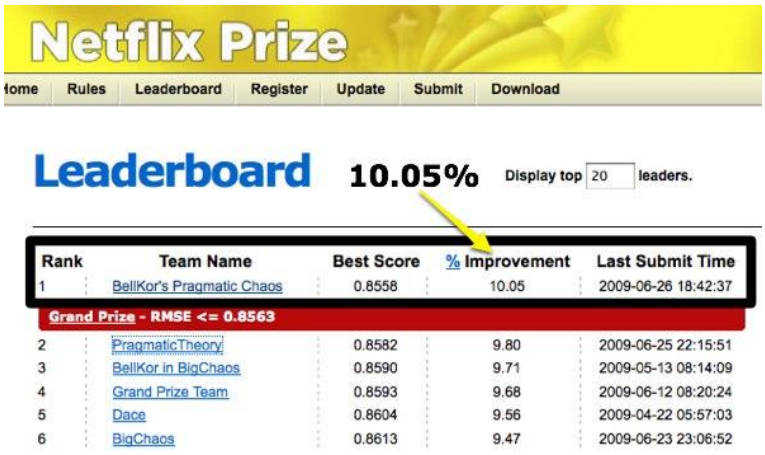
Netflix estimated that, by 2012, 75% of what its users/subscribers viewed was as a direct result of the Netflix recommender engine (Amatrain et. al, 2012). The primary goal of many contemporary apps and services, Netflix included, can be summarized this way: to capture (and retain) a consumer's attention. Keeping users engaged with a service is paramount to the success of any subscription-based business model; thus, making accurate recommendations is essential to Netflix's staying power in a fast-moving, trend-chasing industry like film and television. If users don't find the next title they are

likely to enjoy, they are liable to move on to Amazon, Hulu, Disney+, AppleTV+, or another of the quickly multiplying options for streaming entertainment.

In 2006, Netflix charged participants in the Netflix Prize Challenge with obtaining an RMSE (root mean squared error) value of Netflix's then-best 0.9525 or less on a subset of a data set of 100MM+ ratings—ideally reducing it to 0.8572 or less (a 10% reduction in the accuracy of Netflix's existing system, **Cinematch**)—, with the prize being awarded to the team that achieved the lowest RMSE on the remaining observations in a subset of the data (Töscher et al. 2009).

The objective of this project was to create a model that could replicate or best the top-performing submission to the original Netflix Prize challenge using the same data, whilst also investigating the positives and pitfalls of applying various model types to a recommendation problem.

DATA DESCRIPTION



The original Netflix Prize data set contained ratings records (out of 5-stars) for 17,770 film and television (series) titles from a total of 480,189 users. In total, the number of ratings amounted to 100,480,507. Initially, the ratings for each film were contained in individual files. This hefty data set was only a fraction of the 5 billion ratings in Netflix's complete arsenal in 2006. On Kaggle, users combined the ratings to produce a set of four text files containing all 100M+ observations. The initial data set also contained a probe.txt and qualifying.txt files: the "probe" was a subset containing 1,408,395 ratings; 2,817,131 ratings comprised the "qualifying" subset.

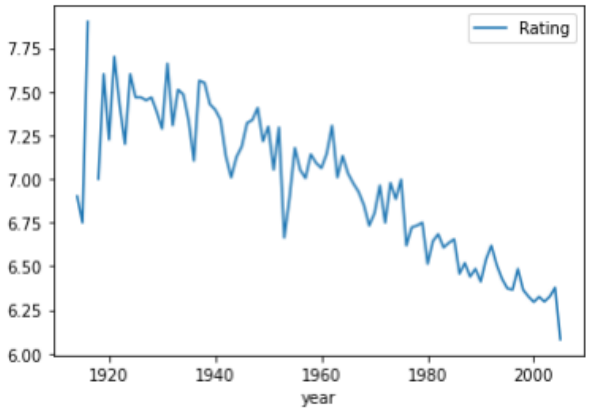
METHODS

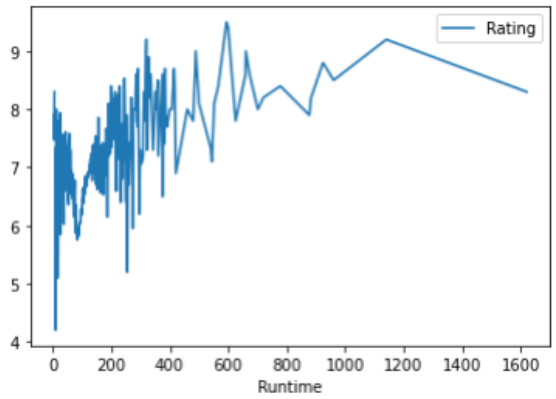
Exploratory Data Analysis

We began our exploratory data analysis by examining how the original Netflix Prize data was structured. We were given four text documents, each approximately 500MB in size. The text files were firstly broken down by Movie IDs, these were further broken down into ratings for these titles (which included both television and film works), with the ratings containing a User ID, Rating, and the rating Date. We found supplementary data for each title's ID: release year, actual title, runtime, average rating, director(s), writer(s), production companies, and genre(s). In total, the data included 17,770 movies, and just over 100,000,000 movie reviews. This data was imported into Pandas dataframes, with functions like

groupby and tools like dictionaries providing the foundation necessary to produce plots and histograms to visualize the data.

The majority of our exploratory analysis took place on the supplemental data-set; we had hoped these additional features would provide more predictive power to our model. We started by constructing graphs to analyze the average rating of these movies along with the runtime and the year released. We noticed a downtrend in average rating as the year released increases, and a slight uptrend in average rating as runtime increases. We took notice of the sharp drop in ratings for movies around 100 minutes of runtime, and further looked into what appeared to be a title with a runtime of 1600 minutes (this turned out to be a documentary TV series).





Additionally, we analyzed the genre, writers, and directors features. The main difficulty with these features was that one movie could fall under multiple genres or have multiple writers and directors. We spent time clearing and decoupling these columns for further analysis. We were able to produce a breakdown of the average rating each genre received, as well as the best and worst-rated writers and directors. Overall, most genres fell in an average rating range of 6-7, with the horror genre as the clear worst at 5.5, and talk show genre in the top spot around 8. With these features analyzed, we were ready to begin attaching these features to our main dataset and begin constructing our initial models.

Data Import & Preprocessing

One of the most challenging aspects of our data import and preprocessing was due to the nature of the text files storing the data: four separate .txt files each stored approximately 25% of the 100MM observations. Each observation included a user ID, rating value (1.0, 2.0, 3.0, 4.0, or 5.0), and date of the rating. However, the movie IDs were not contained in their own column; each movie ID appeared in its own row in the same column as the user IDs followed by a colon. Thus, each movie ID (over 17,000 in total) had to be extracted from this column and placed in a freshly created column. The colon needed to be removed, and the row that once contained that movie ID had to be deleted. All of the aforementioned steps needed to be taken before we could successfully concatenate the four separate data frames built from the quarter of text files. We had to test and audition several methods before we were able to fully pre-process the data files and combine them into one single, usable, formatted file via a combination of extracting and temporarily storing movie IDs in an array, then creating a dedicated movie ID column. Once this preprocessing was complete, the data required little additional cleaning and/or processing.

Data Splitting & Sampling

The PySpark function `randomSplit` was utilized in tandem with a consistent `seed` (314) to produce 75/25 and 80/20 train/test splits.

Feature Selection & Engineering

We had hoped that the additional features for each title/ID would provide more predictive power to our model, but we quickly discovered that it would not be prudent to map the additional features onto our data set of 100MM+ observations: the resulting file topped 15GB, nearly five times the maximum size allotted for this project. The team tested myriad methods and tried many approaches to achieve this; unfortunately, we had little success. In one instance, we thought that perhaps we could extract the unique genres from each row/observation, and then binarize the result (i.e. a title with "Mystery" in the `genres` column would have a 1 in the `Mystery` column, but a 0 in the `Family` column if "family" did not appear in `genres`. This was a creative attempt (albeit a markedly fruitless one) at one hot encoding genre in service of a deeper exploration of classification models/techniques.

While we did attempt to incorporate the additional features/predictors in various ways, it was at this stage that we resigned ourselves to primarily focusing on the idiosyncrasies of methods and models specifically designed for recommender systems: collaborative filtering, single value decomposition, non-negative matrix factorization, sparse matrices, etc., to predict individual users' preferences (specifically, `ratings` as the target/outcome/response variable) for particular titles. The one exception was our work developing a K-Means classification model.

In the case of K-Means, the team attempted to run a classification model that could accurately classify the rating a person would give to a specific movie as "Recommended" or "Not Recommended." Given that the ratings column of our data has values from one to five, we created a new binary variable (`classification`): a value of one for ratings > 4.0, and value of zero for ratings = < 3.0. The size of the data set was 100MM+ rows, so for simplicity reasons, and to allow the models to run in a reasonable amount of time, we sampled approximately 500,000 observations—after first ordering the data frame by `user_id`. By doing this we can have many observations per user rather than just a few for all users. We created training and test sets of the data using an 80/20 split. This small subset of the dataset will be used to test whether this method would work for a recommendation system before further exploration should our results be positive. In addition, we created a new dataset that included descriptive statistics on each title. From this dataset we kept the `year` the title was released in addition to its `runtime`. After merging the two datasets, dropping NaN values, and dropping duplicate columns, we derived our complete training and testing data sets.

<u>Model Construction</u>

We began with a linear regression model to establish a baseline before exploring variations on the elastic net parameter to produce lasso and ridge regression models. Developing a K-means classification model came next. Finally, we pursued a functional Alternating Least Squares model, using cross-validation and hyperparameter adjustments to tune our working model toward an optimal one.

Linear Regression

Linear regression is a standard basic model that works well with linear or normalized datasets and provides a general baseline in most cases. Since there were limited variables available for prediction in this dataset, we used both the Movie ID and User ID as predictors. The initial setup was set-up with `maxIter` (iterations) equal to 10, `regParam` of 0.3, and `elasticNetParam` of 0.8. We used the `pyspark.ml.feature` function `VectorAssembler` to produce an outputCol (`features`) from the inputCols (`user_id` and `movie_id`). The model was trained using `LinearRegression` from the `pyspark.ml.regression` package, and evaluated via `RegressionEvaluator` from `pyspark.ml.evaluation`.

Lasso

Based on the output from the Linear Regression model we tried first, we wanted to see if we could expand on it and get an improved output. The model was also set up with 10 iterations, and a `regParam` of 0.3; to make this a lasso model by definition, we adjusted the `elasticNetParam` to 1. The model was trained using `LinearRegression` from the `pyspark.ml.regression` package, and evaluated via `RegressionEvaluator` from `pyspark.ml.evaluation`.

Ridge

Our ridge regression model was constructed with identical `maxIter` and `regParam` hyperparameters (10 and 0.3, respectively). We adjusted the `elasticNetParam` to 0 in this case, per ridge regression requirements. The model was trained using `LinearRegression` from the `pyspark.ml.regression` package, and evaluated via `RegressionEvaluator` from `pyspark.ml.evaluation`.

K-Means

In order to efficiently run our models, we used the `pipeline`, `VectorAssembler`, and `StandardScaler` functions/features of the `pyspark.ml` package to create a pipeline for each model using various feature columns to see which columns and/or combinations worked best. The `StandardScaler` was used specifically to address the discrepancy in scale between the rating, release `Year`, and `Runtime` features. `VectorAssembler` allowed us to load various combinations of predictors as input columns to produce a `features` column.

Alternating Least Squares

Alternating Least Squares (ALS) is one of the more favored algorithms when approaching the problem of personalized recommendation, primarily because of its scalability: data with high-dimensionality is reduced to many fewer dimensions through the use of a limited number of unobservable factors (called **latent factors**). The ALS method requires the creation of a user-item

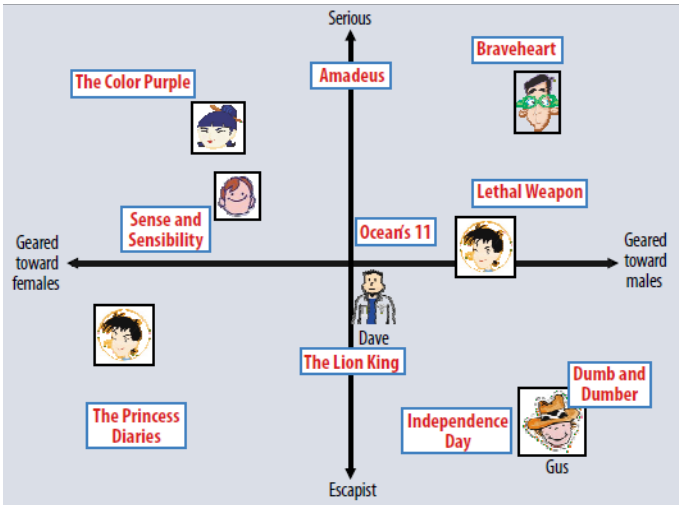


Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

interaction matrix, which then undergoes non-negative matrix factorization to produce a model. The **rank** parameter represents the number of latent factors used in the model (the number of columns in the user-feature and product-feature matrices).

Constructing our ALS model necessitated use of the `pyspark.mllib` package's built-in `recommendation` functionality. We created training and test data sets and stored them in RDDs, then used `lambda` functions in conjunction with `map` and `split` to produce training and test Rating matrices (the object form required to train implicit ALS models using Mllib).

[Image source: Koren et al. 2009]

Model Tuning & Selection via Hyperparameters

We iterated through various **rank** values to determine which minimized the MSE and RMSE. We consistently used a numIterations value of 15, and alpha = 0.01 to eliminate the possibility of overtraining/overfitting.

Cross-Validation

To cross-validate results, we imported our processed data into Spark dataframes, as required by the more up-to-date `pyspark.ml` package (as opposed to the RDDs required by `pyspark.mllib`). Initially, we encountered a problem when we attempted to implement the `CrossValidator` function from the `pyspark.ml.tuning` package (in conjunction with `ParamGridBuilder` and the `pyspark.ml.recommendation ALS` function): an Apache Spark known issue prevents the function from properly computing RMSE for ALS models. Essentially, the function returns NaN if and when a user or item (in this case, movie ID) is unknown—this is likely to occur in the context of any problem using sparse matrices, as some users and items will occur only in the training or test set, not in both. Using documentation on this issue (Apache Spark 2016) and suggested workarounds (IBM Cloud 2019), we wrote custom functions to iterate through various train/test splits and *k*-folds. Once we'd successfully written such functions, we repeatedly threw `OutOfMemory` errors/exceptions. The function would not operate on our data set unless our Rivanna session had allotted a minimum of 128GB. Ultimately, our average RMSE hovered around 0.85.

Personalized Recommendations by User

We created and employed `broadcast` variables to efficiently map the titles of films to our massive data set of ratings. These, in combination with the best ALS model we were able to produce, allowed us to identify specific film/TV recommendations for specific users, based on titles that the user had previously watched and rated. Initially, we imported a CSV file containing titles with their corresponding movie IDs (matching those in the original Netflix Prize data) and stored the data in an RDD. The team used a custom function (parseMovieIdTitlePair) to separate each `title` from its `movie_id` and mask the release `year`. The RDD was transformed to a broadcast variable via parsing performed by the custom function and use of the `flatMap` and `collectAsMap` operations. We collected titles viewed by the test users using our matrix of training data (filtered by specific user IDs and using our broadcast variable in tandem with `value.get(observation.product)`.

RESULTS / DISCUSSION OF RESULTS

Linear regression*:* The standard linear regression model did not perform well on the data, as indicated by the resulting R2 value: -5.26529e-09, an incredibly small number, in addition to being negative—indicating that the model itself was not a good choice for this problem. Mean Squared Error (MSE) of the Linear Regression Model was 1.17768. While the MSE was not extreme, the R2 value clearly indicated that the model couldn't explain the outcome well. Furthermore, the resulting RMSE could not even outperform Netflix's best RMSE in 2006, a fifteen-year-old benchmark. Lasso regression yielded similar results. The R2 was -5.26529e-09, identical to the standard linear regression model, and the MSE was also the same: 1.17768. Considering that `elasticNetParam` values of 0.8 and 1 are very close, this was not surprising. Ridge regression performed the best out of the three linear models that we tested, but the results were still poor. The MSE was 1.1775978, a minor step up from the linear regression and the lasso regression results. The R2 was 7.1522e-05 for this model—an increase, yes, but overall still very poor when considering that R2 represents the amount of variance explained by the model. Overall, ridge regression was only able to account for an insignificant amount of variance of the data, while the resulting R2 values for standard linear and lasso regression indicated that these models are entirely inappropriate for the type of problem posed by recommender systems.

Model	Train/Test Split	ElasticNetParam	MSE	RMSE	R^2
Linear Regression	80/20	0.8	1.177682	1.3869	-5.2653E-9
Lasso Regression	80/20	1.0	1.177682	1.3869	-5.2653E-9
Ridge Regression	80/20	0.0	1.1776	1.3867	7.15E-5

K-Means: When using the `rating` column alone and with `year` as a variable, the results were near zero; we determined that rating shouldn't be considered as a potential feature because in a real world setting we wouldn't have a user's rating for a movie if they haven't already seen it. For that reason using `rating` to assess model accuracy of models has severe bias and should be eliminated. The model that

performed best given the limitations we found with the K-means model involved using variables `year` and `runtime` as features. The metrics for this model overall were poor and we wouldn't consider using it for an actual recommendation system, but we decided to further explore the model by testing different 'k' values. The results can be seen below and lead us to see the optimal 'k' value to use with this model would be a value of 2. This 'k' value led to the highest precision and F1-Score metric, while also having a relatively high recall. Overall, we can conclude that K-means classification models perform poorly on recommendation system problems and other model types should be prioritized.

k_value	Precision	Recall	F1Score
2	0.8469857478842945	0.5499432240886487	0.6668829729986389
5	0.07027559450821155	0.6633776091081593	0.12708799098460477
10	0.01692564375741251	0.6223207686622321	0.03295499021526419
15	0.0179307294912256	0.7181964573268921	0.03498793857498676
20	0.047198826059862906	0.662528216704289	0.08811994520650766

ALS: Ultimately, a `rank = 15` and `alpha = 0.01` produced the lowest values: `MSE = 0.726944` and `RMSE = 0.852610`. Fascinatingly, the resulting RMSE is just under the Netflix Prize-winning RMSE of 0.8558. This is a reflection of the massive advancements made in the study and development of recommendation systems since the BelKor/Pragmatic Chaos team achieved the value in 2009, without the support of current tools available to us. From the sampling of titles and their rating predictions (featured in the image at left), it is clear that not all predictions are created equally. A 3.9 prediction on a title actually rated 4.0 is not bad; however, the predicted values call attention to another of the data set's shortcomings: a user only has five rating options from which to choose. This almost amounts to a discrete, categorical ratings framework, while the predictions being calculated are continuous values ranging from 0.0 to 5.0. Essentially, in some ways, the model is doing more work than is necessary given the nature of how Netflix allowed users to rate titles in its early years; we hypothesize that this reality contributed to Netflix's decision to change how users rate titles on its service: presently, users are asked to provide either a simple thumbs up or thumbs down.

[9]:

	rank	MSE	RMSE
0	5	0.747698	0.864695
1	10	0.727663	0.853032
2	15	0.726944	0.852610
3	20	0.732890	0.856090

```
((209573, 1), (4.0, 3.9097003814804343))
((1959936, 2), (5.0, 2.990446066638587))
((755319, 3), (3.0, 3.5455251470879565))
((596255, 3), (1.0, 0.9982883758030261))
((499971, 3), (5.0, 4.221142056113273))]
```

We were successful in our efforts to procure personalized recommendations for specific user IDs. Based on the titles that user_id = 30878 had viewed and rated highly (i.e. films from the *Star Wars* and *Transformers* franchises, *8 Mile*, *The Matrix*, *Titanic*, *Tron*, and *Apollo 13*), our model suggested: *Lost: Season 1*, the *Lord of the Rings* trilogy, *Battlestar Galactica: Season 1*, and *Dead Like Me*. While there are infinitely many dimensions from which to analyze an individual's taste in film and television, even

our sparse-matrix, ratings-based model seemed to intuit user 30878's predilection for big-budget spectacle, science fiction and space, and epic hero journeys through fantastical realms.

`User_id = 499971` had viewed significantly fewers titles than user 30878: only 54 in total. This translated into a batch of recommendations from which it was not as easy to discern content-related patterns.

FUTURE WORK / POTENTIAL PROJECT EXPANSIONS

> "...the Netflix Prize objective, accurate prediction of a movie's rating, is just one of the many components of an effective recommendation system that optimizes our members' enjoyment. We also need to take into account factors such as context, title popularity, interest, evidence, novelty, diversity, and freshness. Supporting all the different contexts in which we want to make recommendations requires a range of algorithms that are tuned to the needs of those contexts" (Amatrain et. al, 2012).

Due to the nature of the data there was a limit to the types of models we could successfully, meaningfully employ. We were also constrained by the size of the dataset and time. To expand the scope of this project, further variables could be added: director, producer, the time it took to produce the movie, actors, date of release, studio, budget and many other factors that could contribute to a model's predictive ability. Given that taste in entertainment is highly subjective, personal, and nuanced, additional features could be engineered from this bonus data, such as director-producer combination. Critical opinions might also prove a valuable predictor resource: text analysis could be performed on reviews; professional reviewer ratings could be added to an expanded slate of variables. Adding Twitter, Facebook, Instagram, or other social media statistics (possibly even streaming data via tools like Kafka) might be conducive to assessing more nebulous factors like popularity and word-of-mouth. Entertainment has an inherent cultural component: user/subscriber demographics could be combined with other features related to the global socio-political landscape. For example: in the United States, action titles perform about 15% worse than the global average; in China, action titles rake in almost 60% more at the box office than the average. The U.S. boasts most interest in documentaries, while documentaries perform 100% worse than the global average in China (no doubt due to cultural/political censorship). Drama performs best in South Korea; in Mexico, it's horror (Follows et al. 2021).

In addition to elaborated feature engineering and augmentation, recommender system implementations could vary their approach to weighting the importance of features, as well as filtering methodologies beyond collaborative: content-based, mobile, session-based, etc.

Most recently, deep learning has been applied to the recommender system problem, with Spotify and YouTube flexing muscles in the space of Convolutional and Recurrent neural networks, respectively. The original Netflix Prize-winning team used Restricted Boltzmann Machine techniques in conjunction with collaborative filtering to achieve their results in 2009. Any of the aforementioned methods, coupled with more credence and care paid to predictors, might improve the performance of our recommender system models.

CONCLUSION

This project used PySpark (with limited support in pure Python) to construct a model that predicts how specific Netflix users will rate and rank various film and television titles, using 100MM+ ratings from a Netflix-sanctioned data set. We demonstrated the pitfalls of using linear, lasso, and ridge regression models—plus K-means classification—to produce viable results. We found that contemporary tools for building a model using the Alternating Least Squares algorithm easily beat both the 2006 RMSE goal set by Netflix and the ultimate RMSE benchmark set by the Netflix Prize-winning team. Next steps in this work are to engage in more aggressive feature engineering and augmentation, to employ more robust model optimization and validation, and to explore CNNs (convolutional neural networks), RNNs (recurrent neural networks), and RBMs (restricted Boltzmann machines) as applied to recommender systems.

REFERENCES

"[SPARK-14489][ML][PYSPARK] ALS unknown user/item prediction strategy." *Github, Apache Spark,* 4 May 2016, https://github.com/apache/spark/pull/12896. Accessed 31 Jul. 2021.

Amatriain, Xavier, and Justin Basilico. "Netflix Recommendations: Beyond the 5 stars (Part 1)." *Netflix TechBlog,* 6 Apr. 2012, https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429. Accessed 30 Jun. 2021.

Amatriain, Xavier, and Justin Basilico. "Netflix Recommendations: Beyond the 5 stars (Part 2)." *Netflix TechBlog,* 12 Jun. 2012, https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-2-d9b96aa399f5. Accessed 30 Jun. 2021.

Amatriain, Xavier, and Justin Basilico. "System Architectures for Personalization and Recommendation." *Netflix TechBlog,* 27, Mar. 2013, https://netflixtechblog.com/system-architectures-for-personalization-and-recommendation-e081aa94b5d8. Accessed 15 Jun. 2021.

"How TikTok Recommends Videos For You." *TikTok Newsroom*, 18 Jun. 2020, https://newsroom.tiktok.com/en-us/how-tiktok-recommends-videos-for-you.

Jackson, Dan. "The Netflix Prize: How a $1 Million Contest Changed Binge-Watching Forever." *Thrillist*, 7 Jul. 2017, https://www.thrillist.com/entertainment/nation/the-netflix-prize. Accessed 1 Jun. 2021.

Johnston, Casey. "Netflix Never Used Its $1 Million Algorithm Due to Engineering Costs." *Wired*, 16 Apr. 2021, https://www.wired.com/2012/04/netflix-prize-costs/. Accessed 15 Jun. 2021.

Koren, Yehuda. "The BellKor Solution to the Netflix Grand Prize," Aug. 2009.

"MLA Formatting and Style Guide." *The Purdue OWL,* Purdue U Writing Lab. Accessed 4 Aug 2021.

“Movie recommender system with Spark machine learning.” *IBM Cloud*, 15 Nov. 2019.

Motion Picture Association. “The American Motion Picture and Television Industry: Creating Jobs, Trading Around the World.” *MPA Research Docs*, 8 Apr. 2021, https://www.motionpictures.org/research-docs/. Accessed 4 Aug. 2021.

“Netflix Prize Data.” *Kaggle,* https://www.kaggle.com/netflix-inc/netflix-prize-data. Accessed 10 Jun. 2021.

“Plotting data in PySpark.” 1, Nov. 2015. https://standarderror.github.io/notes/Plotting-with-PySpark/. Accessed 1 Aug. 2021.

Rahman, Was. “The Netflix Prize -- How Even AI Leaders Can Trip Up.” *Medium*, 11 Jan. 2020, https://towardsdatascience.com/the-netflix-prize-how-even-ai-leaders-can-trip-up-5c1f38e95c9f. Accessed 1 Jul. 2021.

Töscher, A., Jahrer, M., & Bell, R. M. (2009). The bigchaos solution to the netflix grand prize. Netflix prize documentation, 1-52.

Follows, Stephen, and Bruce Nash. “The Relative Popularity of Genres Around The World.” *American Film Market,* 2021. https://americanfilmmarket.com/relative-popularity-genres-around-world/ Accessed Aug. 10 2021.