REPORT ON MOVIE RECOMMENDATION

Amogh Mishra, Siddhant Shandilya, Swarna Bhartiya
amogh.mishra@columbia.edu, siddhant.shandiya@columbia.edu, sm4776@columbia.edu

1. Problem Statement: We are a team in the digital media company responsible for building a recommendation system for our stakeholders. The dataset to leverage for this task is 20M movielens dataset.

2. Objective: Develop v1.0 of the movie recommendation system which provides top-k movies to watch. We care the most about our users and try to recommend the best possible movies to them using the best technique. While RMSE and MAE plays an important factor in evaluating our model, error-based metrics weigh all errors equally, but errors in the less-relevant tail of items are probably not as important as the rank of the most relevant items. Therefore we are also concerned about the accuracy in the top k ranked movies that the end user is able to see.

3. Assumptions:

a) We have assumed that our stakeholders are not interested in the cold-start problem.
b) The user taste profile follows a decay with time on an annual level.
c) Targeted audience is active users (defined in [7]) for v1.0.
d) Movies considered are above the median level of popularity. (defined in [7])

4. Metadata available: The Movie Lens dataset consists of movies, genre, users, user's rating and timestamp of the rating.

5. Models Used: There are several techniques that were considered to develop a recommendation system with varying levels of difficulties. The models are as follows:

a) User-User collaborative filtering
b) Item-Item collaborative filtering
c) Matrix Factorization
d) Factor Machines
e) Collective Factor Machines
f) Field-aware Factorization Machines
g) Content-Collaborative based Hybrid Recommendations
h) Knowledge-Graph based Recommendations
i) Deep Learning based Recommendations

Given the multitude of options available, our RecSys 1.0 is developed using fundamental algorithms:

a) Matrix Factorization using PySpark
b) Item-Item Collaborative Filtering using Annoy Index
c) Factor Machine using LightFM package

Note: The following content is this report compares the strategy between ALS and Item-Item collaborative filtering. Factor Machines (FM) were used in a limited capacity of running on small dataset. We ran FM on movie-latest-small.zip and achieved AUC score of 0.99. We also tested the model to provide top 10 items to recommend for a sample user.

6. Metrics Used for performance evaluation (both quality and scalability):

a) **RMSE**: Root Mean Square Error (**RMSE**) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the actual ratings, the predicted ratings are. **RMSE** is a measure of how spread out these residuals are. It gives more weight to outliers.

Formula: $$\sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

b) **MAE**: Mean Absolute Error (MAE) is the average vertical distance between each actual rating and the predicted rating. Formula: $\frac{1}{n}\sum_{i=1}^{n} \left|Predicted_i - Actual_i\right|$

c) **Coverage**

i) **User coverage**: First we define what a good recommendation is (eg, a predicted rating >3.5). Then, user coverage is the proportion of users for which at least k movies can be recommended well. First, we define what a good recommendation is and fix a value of k to calculate the coverage value.

ii) **Catalogue coverage:** First we define what a good recommendation is (eg, a predicted rating >3.5). Then, Catalogue coverage is the proportion of movies for covered in the recommendation for all the users. In other words, the fraction of items that are in the top-k for at least 1 user.

d) **Time:** Time taken to run the model, i.e parameter tuning and calculating the evaluation metrics.

e) **Model size**: Sample size from the original data to train the model including the learning of hyperparameters.

7. Sampling Methodology

The entire movielens dataset has 20 Million ratings with 138000 users and 27000 movies spread across time from the year 1995 to 2015. The intuition behind the sampling process is that the user's interest is influenced by external factors such as seasonality, trending items etc. Therefore, we filter the sample on an annual basis starting from 2015 and moving backward in time. As we go back in time, we append more user history to provide with more signals to the model. At each experiment step, we report our metrics as discussed above in [6].

A second level of filtering is accomplished by adding an assumption that our stakeholders are only interested in recommendation over active users on above average movies on popularity index. We define an active user if the user rates more than 30 movies and movies which have more than 100 ratings and popular movie if it consists of more than 100 ratings. This assumptions also helps in mitigating the user not found exception in the test dataset.

8. Model Explanation:

I. Baseline

For any recommendation system we build or any algorithm we use, It is important for us to test if it is performing well. Since we are short of a relative metric most of the time, the simplest way to measure the performance is to test our algorithm against a baseline model. A baseline model is one which is the simplest model and we don't put in extra effort to make any prediction for each user-item pair. We have incorporated two baseline models in our analysis:

a) **ALS based baseline model**: In this, we are predicting the baseline estimate for given user i and item j by using:

$\hat{r}_{ij} = \mu + b_i + bj$ .Here, $\hat{r}_{ij}$ is the estimated rating of the ith user and jth item. b_i is the bias of the ith user and b_j is the bias of the jth item from the ALS model.

b) **Baseline average model**: In this, we are we are predicting the baseline estimate for given user i and item j by using:

$$\hat{r}_{ij} = (\mu_i + \mu_j)/2$$

Here $\hat{r}_{ij}$ is the estimated rating of the ith user and jth item. μ_i is the average rating given by the ith user and μ_j is the average rating of the jth item in the dataset

Both our baseline models have been created using the training dataset and consequently validated on the tune dataset to report the RMSE and MAE which can be used to test against the various Collaborative Filtering models we have used.

For Baseline models, the final results on the test set are:

ALS Baseline

RMSE on the test set	0.88
MAE on the test set	0.66

Average Baseline

RMSE on the test set	0.91
MAE on the test set	0.83

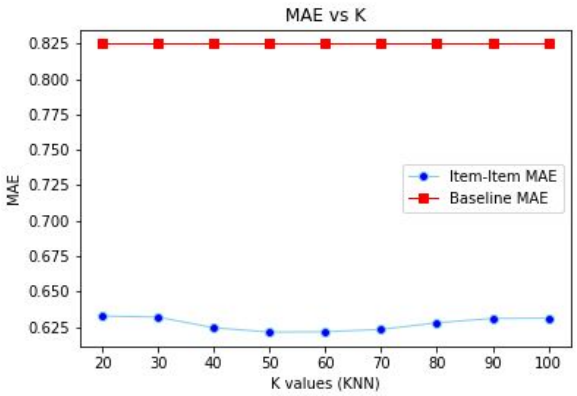
II. Item-Item Collaborative Filtering

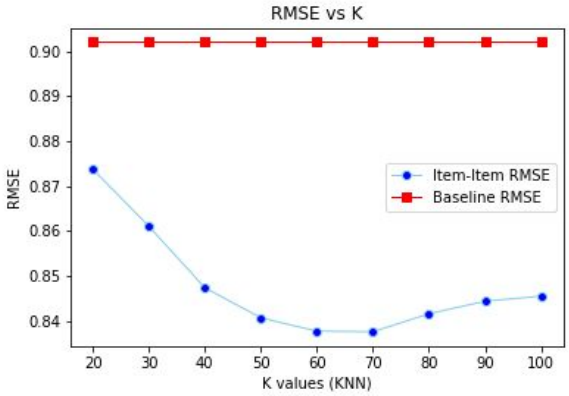
Item-Item Collaborative filtering refers to recommending items to a user based on item similarity. To implement this idea, we used Annoy (Approximate Nearest Neighbors) which helps to get the K approximate nearest neighbours of a particular item. Annoy is an open source library written internally in C++ to provide the production-level scalability along with ease of use via Python binding. Our operation was more read-heavy (getting K nearest items) than write-heavy (creating annoy index) which provides superior performance on an amortized level. Therefore, we used annoy index to fetch the top K nearest neighbours. For each user-item pair in the test_set, K (K approximate nearest neighbours) items similar to that item were selected and then the following weighted sum prediction technique was used.

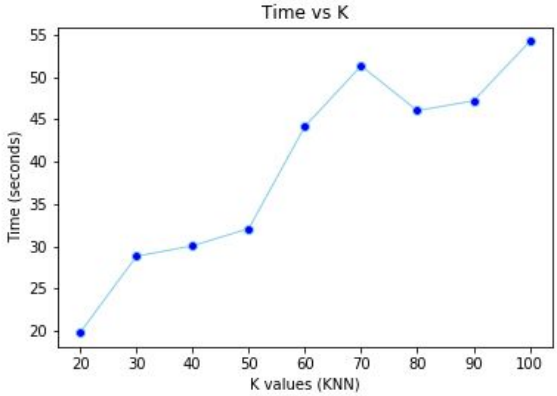
$$P_{u,i} = \frac{\sum_{\text{all similar items, N}}(s_{i,N} * R_{u,N})}{\sum_{\text{all similar items, N}}(|s_{i,N}|)}$$

Where N is the set of items similar to item i, $s_{i,N}$ is the similarity of i with item in N. $R_{u,N}$ is the ratings given by user u to item in set N. If the user u has not rated any item in N, then that particulate user-item pair was given a rating that is the average rating given by the user to the items he/she rated.

As mentioned earlier, the initial dataset corresponds to the year 2015. The following plots, table show the performance of the item-item collaborative filtering method developed.



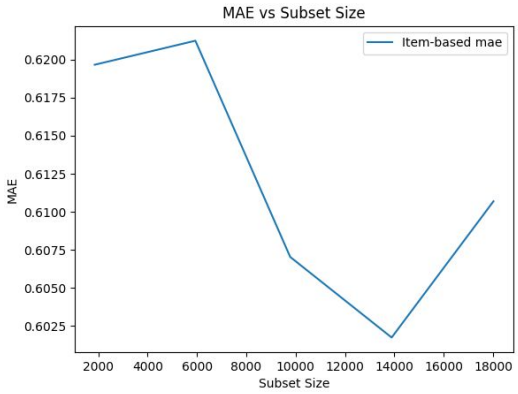


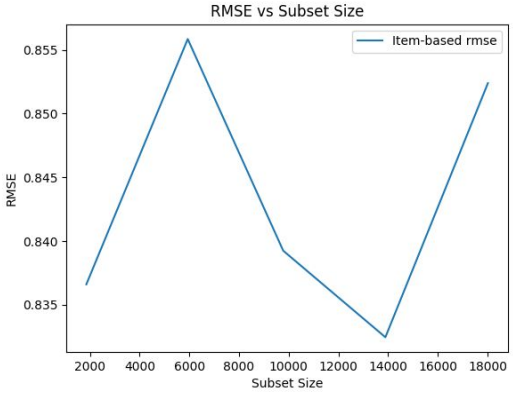


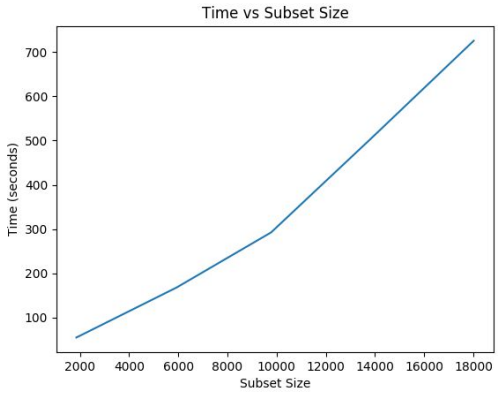
K (knn)	20	30	40	50	**60**	70	80	90	100	Baseline
MAE	0.632	0.630	0.625	0.624	**0.624**	0.625	0.627	0.630	0.631	0.83
RMSE	0.874	0.861	0.847	0.842	**0.837**	0.838	0.843	0.845	0.846	0.91
Time(sec)	20.15	29.58	30.33	31.45	44.19	52.10	46.63	47.21	54.88	

As we can observe, item-item collaborative filtering has lower (better) MAE (Mean Absolute Error) and RMSE (Root Mean Square Error) values compared to the MAE, RMSE of Baseline MAE and RMSE (average model). Least MAE and RMSE of item-item model are observed for K value 60.

Next, we compared the performance by scaling the dataset. Following are the plots obtained comparing datasets for K = 60. K was chosen 60 in particular because it gave the best MAE, RMSE values for the initial dataset.







Note: Here Subset size refers to the size of the testset of the subset.

The MAE vs Subset Size and RMSE vs Subset Size plots above seem to be random. The reason for such plots can be the rating entries of each user-item pair or the testset chosen from the subset for prediction. These can affect the K nearest neighbours collection and may vary the error values.

The time plot is as expected. The time for prediction over the test dataset of each of the subset increases with the size of the test dataset

III. ALS

Here, we have used the ALS method from pyspark. Spark Machine Learning currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. spark.ml uses the alternating least squares (ALS) algorithm to learn these latent factors.

Matrix factorization is a technique in which we create latent factors to build a recommendation system where we try to jointly factor the users and items in the same latent space. This significantly reduces the size of the matrices which gives an enormous computational advantage over the behaviour based models like user-user/item-item collaborative filtering. Apart from scalability, it also reduces the possibility of overfitting by regularizing the model so that it can be generalized over the data set.

The equation which the Matrix factorization attempts to minimize is given by:

$$\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 = \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2$$

$$\text{subject to:}$$

$$\text{No constraints on } U \text{ and } V$$

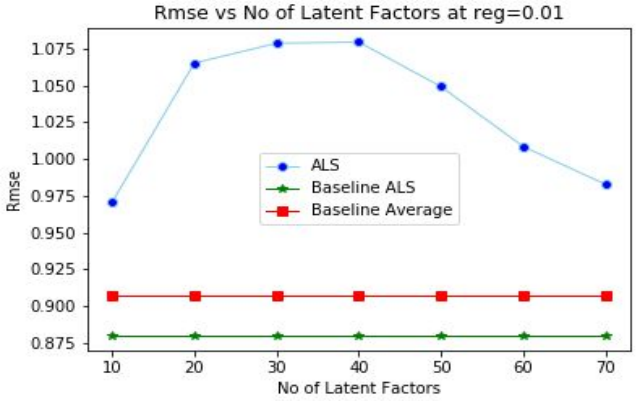
This uses a technique known as alternative least squares to minimize J by alternatively optimizing for u and v in subsequent steps. Here are the results from our ALS model.

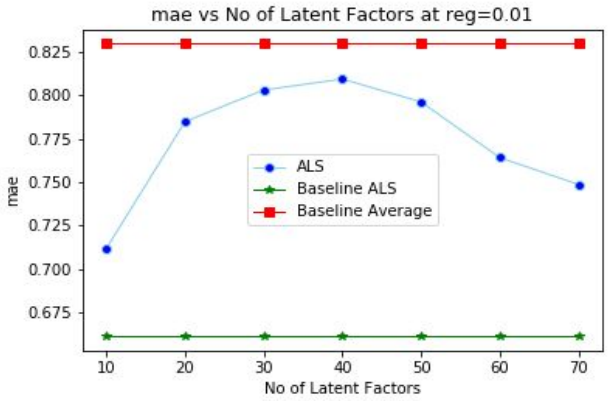
For **training set**,

RMSE on Training Data	0.63
MAE on Training Data	0.48

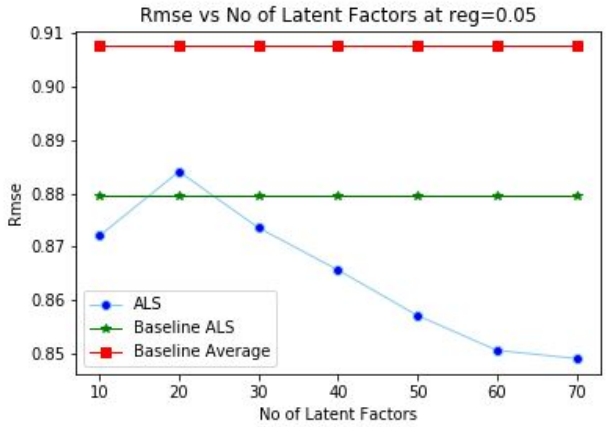
User Coverage on Training Data	91%
Catalogue Coverage on Training Data	90%

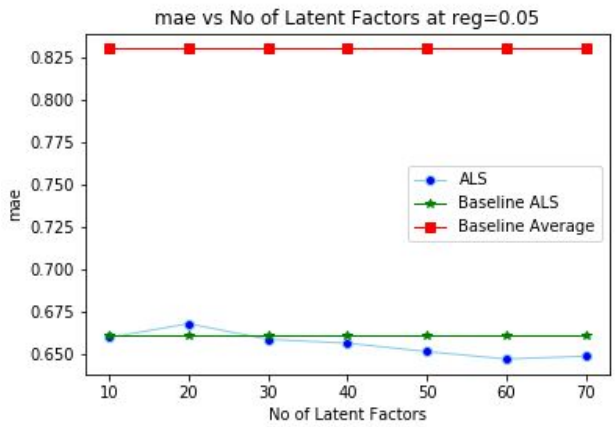
Parameter Tuning gave the following results which can be better interpreted using plots:



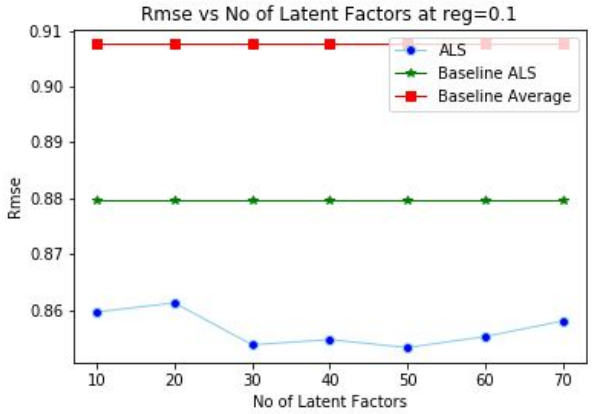


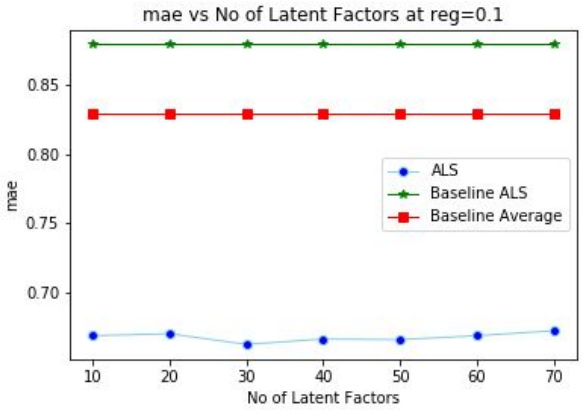
Plot of **RMSE** and **MAE** compared to the **number of latent factors** after fixing the regularization parameter at 0.01. Here can see that the baseline is performing better than our ALS model.





Plot of **RMSE** and **MAE** compared to the **number of latent factors** after fixing regularization parameter at 0.05. Here we can see that for most values of the latent factor, our ALS model has a better RMSE than the baseline models and the RMSE decreases as the number of latent factor increases. Incase of MAE, again our ALS model is performing better than both the baselines as the number of latent factor increases after 30.





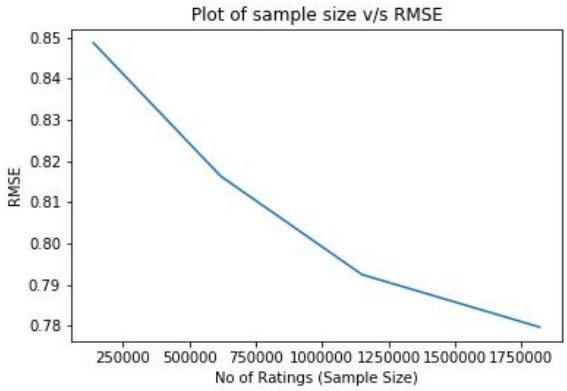
Plot of **RMSE** and **MAE** compared to the **number of latent factors** after fixing the regularization parameter at 0.1. Clearly, our ALS model outperforms both the baseline models.

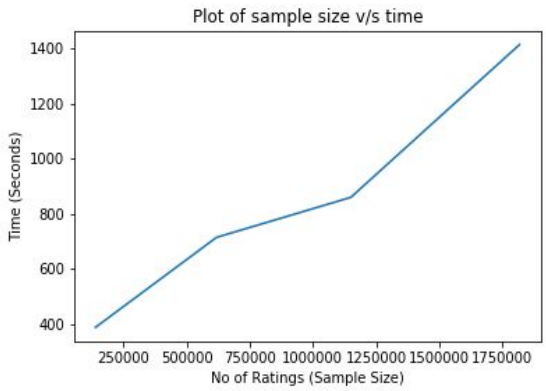
Parameter tuning gave the best results for regularization parameter = 0.05 and no of latent factors =70. For **test set** we have:

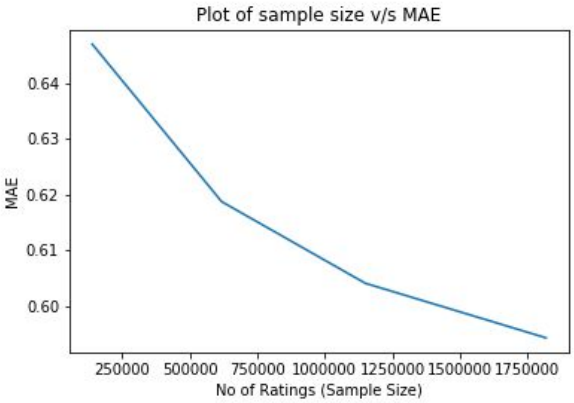
RMSE on Test Data	0.84
MAE on Test Data	0.65

User Coverage on Test Data	90%
Catalogue Coverage on Test Data	89%

Hereafter, we subsequently increased the size of the dataset to train our model and check if there is any change in accuracy and how well does the model scale (i.e. we measure the time taken to run everything)







When we increase the sample size, eventually our model starts to show better results in terms of accuracy as the RMSE drops down by 8.2% and MAE drops down by 8.7% but the time for model and hyperparameter training increase by 3x which might be a potential issue if we actually have to train our model on the entire dataset. Also, since our local machine has limited computational power, we could not increase our sample size beyond 1.8 Million as it was throwing OOM error.

9. ALS vs Item-Item collaborative filtering (Comparison):

Based on the plots, tablels, and data included in the Item-Item and ALS section, we can see that for the initial dataset we created, Item-Item collaborative filtering returned slightly better (about 0.02 difference) MAE, and RMSE values. This is in contrast to what is usually observed for large datasets. So, this result could be due to the comparatively smaller size of the dataset we created.
As we scale the dataset, ALS performs much better compared to Item-Item collaborative filtering. It shows a trend of decrease in the MAE, RMSE as the size of the subset increases. As Item-Item collaborative filtering did not generate a proper trend, we might not be able to guarantee good performance as the subset size increases.
This is observed from the plots included.

10. Sample User Case Study on ALS only

Movies watched by a random user

userId	title	genres
1644	Sweeney Todd: The Demon Barber of Fleet Street (2007)	Drama|Horror|Musical|Thriller
1644	Interstellar (2014)	Sci-Fi|IMAX
1644	Mr. & Mrs. Smith (2005)	Action|Adventure|Comedy|Romance
1644	Pirates of the Caribbean: At World's End (2007)	Action|Adventure|Comedy|Fantasy
1644	Hitchhiker's Guide to the Galaxy, The (2005)	Adventure|Comedy|Sci-Fi

Movies recommended to the random user

userId	title	genres
1644	Cinema Paradiso (Nuovo cinema Paradiso) (1989)	Drama
1644	Pianist, The (2002)	Drama|War
1644	Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)	Action|Adventure|Western
1644	Once Upon a Time in the West (C'era una volta il West) (1968)	Action|Drama|Western
1644	Shining, The (1980)	Horror

We can say that our model is able to capture the genres which this user likes, eg. Drama, Action, Adventure and Comedy. Apart from this, Western is a new recommendation which we can attribute to novelty. Therefore, diversity and novelty is somewhat captured by our model. To confirm this and evaluate using some metric, we need more information regarding the users such as their implicit feedback and use other algorithms like SVD++ to incorporate the data and use ranking metrics like NDCG.

11. FURTHER QUESTIONS?

I. When to use what? Item-Item or ALS?

Model-based systems are likely to be faster, at least in comparison to memory-based systems because the time required to query the model (as opposed to the whole dataset) is usually much smaller than that required to query the whole dataset.

II. *Other Design choices:*

1) A web-service based recommendation service to demo stakeholders.
2) Develop an API and On-prem based solution covering different clients needs.
3) Incorporate user's social media activity to add features into a deep learning model.
4) Extensibility to other languages using transfer learning using a deep neural model.

III. Business rules:

We can use guardrails while recommending movies in the following cases:

1) While recommending movies, It is important that the top recommended movies which the end user looks at should be according to his or her taste. But, diversity is one important thing which should be included. For eg, recommending only “Action” movies to a user who also likes “Adventure” movies could have a negative impact in the sense that user doesn’t get to see novel genres. Here we can introduce a business rule to make diverse recommendations to the end user.

2) While recommending movies to children or a person less than a specific age, we would not want to recommend a movie which has an explicit content even though it matches with the preferred taste of the user.

IV. *How does your recommendation system meet your hypothetical objectives? Would you feel comfortable putting these solutions into production at a real company?*

Our recommendation system meets the hypothetical objectives in the sense that it has low RMSE and MAE for both item-item and Matrix Factorization using ALS. We were also able to attain a high user and catalogue coverage. Since we are concerned more about the user, this model can be deployed after a few checks for which we need additional data and one or two other metrics to check the accuracy in the top k recommended movies which the end user is able to see. We feel comfortable because after checking for a few sample cases (eyeballing for now) we were able to see diverse and accurate recommendation for the user based on his or her taste.

V. *Potential watchouts:*

The v1.0 of the recommendation system operated on the active users and popular movies. However there are the following scenarios which are not addressed in this version:

1. Cold Start: How would the recommendation behave when there is a new user?
2. Scalability: Would we perform batch-processing for training? If not, how can we approach a near real-time recommendation.
3. Sparsity: How to make the computation efficient with increasing data sparsity?
4. Grey - Sheep User Problem: How to provide recommendation to unpredictable user’s activity?
5. Synonymy - How to prevent not recommendation the semantically same movie?
6. Shilling Attack - How to make recommendation robust against attacks to throw off the model?
7. Adaptability - How to construct a model which quickly adapts to new user’s taste?
8. Discover - How to discover the latent user’s taste and provide a new/not-so-popular items yet highly recommendable item. (~similar to Spotify)
9. Social Sensing - How to incorporate the signals from the user’s social media activities to improve the robustness of the model?