MovieLens Data Recommendation System

Prasham Sheth (pds2136), Shreyas Jadhav(sj3006) and Manas Dresswala (mad2306)

INDEX

1. Introduction **1**

2. Data Analysis **1**

3. Business Rule **1**

4. Data Sampling **3**

5. Objective **4**

6. Recommendation Systems **5**

7. Evaluation Metrics **5**

8. Hyperparameter Tuning **6**

9. Coverage **6**

10. Implementation **7**

10.1. KNN 7

10.1.1. Hyperparameter Tuning 7

10.1.2. Evaluating the systems (RMSE and R2) 11

10.1.3. Runtime Scalability 12

10.1.4. Coverage 12

10.2. ALS 14

10.2.1. Hyperparameter tuning 14

10.2.3. Runtime Scalability 16

10.2.4. Coverage 16

11. Future Scope **17**

1. Introduction

In today's world, where data has become the new oil, recommendation systems has taken a huge leap in terms of research and real time implementation.
With the growing technology industry, the amount of competition between these companies to retain users has grown tremendously. Everyone is behind personalizing their experiences based on the user so that the user actually comes back to their side.
What is stopping a user from simply switching tabs and going for movie recommendations to Netflix or Imdb? What does it take to actually understand the users, keeping in mind his/her preferences, and recommending them the right movie? Keeping in mind one of the most important business goals, wherein, the company needs to make some profits out of it. If the company doesn't make profits by implementing the recommendation systems, what was the point of doing it in the first place?

So let's try and understand what we are trying to do here.
We have been given this data where we have ratings of over x users for y movies. Using this rating dataset we will be trying to recommend movies to users.

2. Data Analysis

After doing some market research, we have come to the conclusion that the most active users (that is the users that have rated more than z number of movies) have a lot of influence on the recommendation of movies. Their ratings are more important according to us, as they have rated a lot of movies. Also, the influence of the users who haven't rated many movies would be very less as they wouldn't be generalizing the trends that we need to consider while suggesting a movie to one user.
The movies also follow a similar trend as the popular ones would be having high frequency of being rated. We only take into consideration such movies which have high frequency of getting rated.

3. Business Rule

Let's take a minute here to understand how our recommendation systems are actually recommending movies to the user, that is the business rule we have considered here.
After each of the implementations of the recommendation system, we will have the predicted rating values of the movies that the user would have given to the movie had he seen it. This predicted value is found based on our implementations explained above. The predicted values are for all movies the user hasn't seen before. From these predicted rating values, we will find the top 10 movies that we would recommend based on these values. If this value is greater than or equal to 3.5, then we assume that the user would like the movie. We only recommend the movies that the user likes (that is the ones with ratings 3.5 or greater).
We have selected 3.5 as the threshold value here as that is the mean movie rating of our dataset.

Keeping this in mind, let's move forward.

Next, let's talk about movies. We want to recommend the movies that our users will actually like, so that they come back to our website for more recommendations.

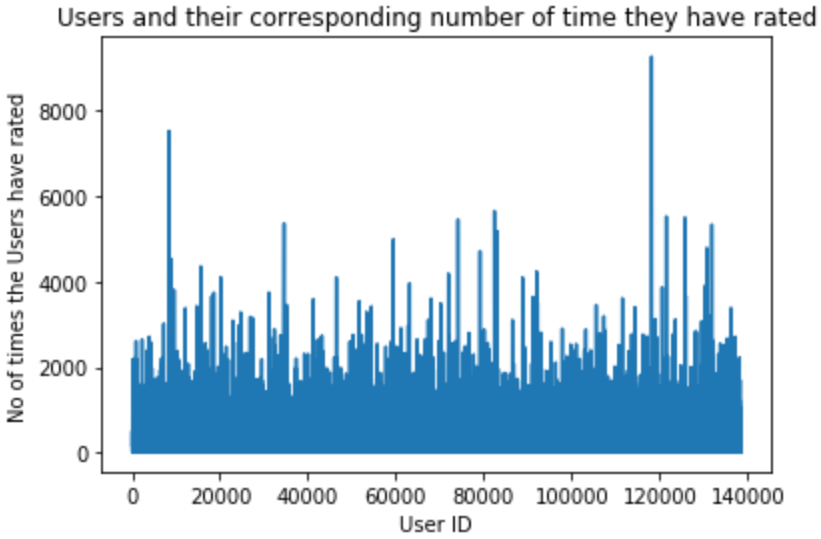
To simply solve this problem, let's assume we recommend the top 10 most popular movies.
You might ask, how do we define “popular” in this case? For this problem, we have defined popular as the movies that have been recommended more than a number of times and then we sort these movies based on their mean rating. The top 10 movies that we get in the end, are the top 10 most popular movies.
Now, we think that just recommending the top 10 most popular movies is not a good thing to do, as we are not considering the users preferences in this case and also that we are letting our data science skills go in vain.

So what we try to do here, is that we build 2 recommendation systems based on our knowledge of collaborative filtering.
The first one is a memory based recommendation system where we find similar items/users using k-NN (k nearest algorithms) in which similarity is used to determine the nearest neighbours .
The second one is a model based approach where we are using ALS (alternating least squares) model to get the ratings and use those eventually to recommend movies.

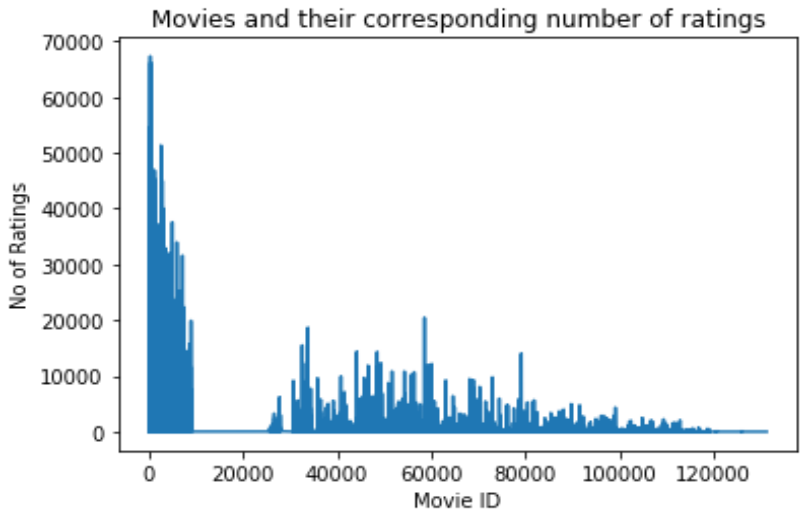
4. Data Sampling

Before we jump into the implementation of these systems, we would like to discuss how we have sampled our data.

Plot(1)



Plot(2)



Sampling of the data was important here as the entire dataset had over 20 millions ratings, and would have a lot of sparsity in it. To find the correct recommendations, we first sampled our data based on what mattered more for us and then tuned our hyper parameters accordingly.

5. Objective

As we have explained above,
1. We care more about the movies that have been rated more (as these are the movies that have been seen more than others, so the chances of a user liking them would be higher).
    - Therefore, as seen in plot(1), we consider movies at the left of the plot i.e the most popular movies.

2. We care about the users that have rated more than a certain number of movies (as these are our active users).
    - Therefore, as seen in plot(2), considering our business model and desired sample we consider only the active users. I.e users that have rated the most.

We are trying to make sure that the users get the recommendations that they will like, keeping in mind the popular movies (so that we decrease the risk of recommending the less popular movies).

Next, to know if we have built a good recommendation system, we first created a baseline and noted its accuracy. We calculate the baseline for the two evaluation metrics: RMSE and R-Squared.

For the baseline for RMSE, we take the mean rating for each of the four samples as the predicted values and calculate the baseline RMSE for the corresponding samples.

For the baseline for R-squared, similar to RMSE we take the mean rating for each of the four samples as the predicted values and hence the R-Squared values is zero. Therefore, the baseline is the horizontal line i.e (y = 0) in this case.

6. Recommendation Systems

Now let's talk about the systems that we have built -

1. **Memory based recommendation system using KNN** - For this case we use different kinds of similarity metrics to calculate the similarity and hence, find the “k” nearest neighbours for the user/item that are into consideration. Based on the average of those neighbours’ ratings we predict the ratings for a user. We built both user-user similarity model as well as item-item similarity model.

   As expected the item-item model performed better than the user-user one and is appropriate for the case we have in hand as adding items to the database makes more sense than adding a new user and computing similarity amongst each of the existing ones. The items being lesser in magnitude makes sense to be compared with each other for computing the similarity.

2. **Model based recommendation system using ALS** - In this case we are using the spark MlLib library used for collaborative filtering that is ALS. Using the API given, we are trying to recommend 10 movies to each of the users.
    1. First we run our model on the sample dataset. Based on the rmse values (we will explain this in just a minute) we are trying to tune the parameters of the ALS model.
    2. The different parameters that we consider here are -
        1. Number of iterations (numIters)
        2. Rank
        3. Regression parameter (lambda)
    3. After finding the best values of the parameters using cross validation, we use different sample sizes to increase the size of our datasets and answer some important questions. In the real-world, the data we have would be much more than this, so we need to make sure that we have made our model good enough to work on the large dataset.

7. Evaluation Metrics

As we promised above, before understanding anything else, lets see how we have evaluated the accuracy of our systems.
To know if we have built a good recommendation system, we first created a baseline and noted its accuracy.
You must be wondering, how we have defined accuracy here.
We have used to evaluation metrics to define the accuracy -

1. RMSE (Root mean squared error) - Primary
2. R2 (R - squared) - Secondary

8. Hyperparameter Tuning

After we have understood the coverage of our systems, we will go on to tune our parameters for the different samples.
We have 4 samples into consideration here -
1. Take the ratings of the top 500 movies, from users that have rated more than 1500 movies
2. Take the ratings of the top 1000 movies, from users that have rated more than 1500 movies
3. Take the ratings of the top 1500 movies, from users that have rated more than 1500 movies
4. Take the ratings of the top 2000 movies, from users that have rated more than 1500 movies

9. Coverage

Because of our business rule (explained in the beginning), we might end up not recommending some movies and we also might end up not recommending 10 movies for some users. Here, we introduce the coverage concept, where we try and understand how much of the sample dataset are we actually covering.
There are two types of coverage that we have found -

1. User coverage = The number of users that we are recommending 10 movies / The total number of users in our sample.
2. Catalogue coverage(items) = The number of unique movies recommended to all users / The total number of movies in our sample.

10. Implementation

10.1. KNN

What KNN does is as said before finds the k nearest similar user/items on the basis of a distance or similarity metric we take. The average of the similar item/users can be then further used to get the predictions.

This can be thought of as getting k similar users or sk similar movies and then using those and the previously known likings of the users by the given ratings, and using those to predict the recommendations for the users.

For the case of KNN we can use different types of similarity metrics and different values of k and can get different outputs and hence, need to tune the values for both these things. We implement the KNN algorithm using the function available in sklearn library.

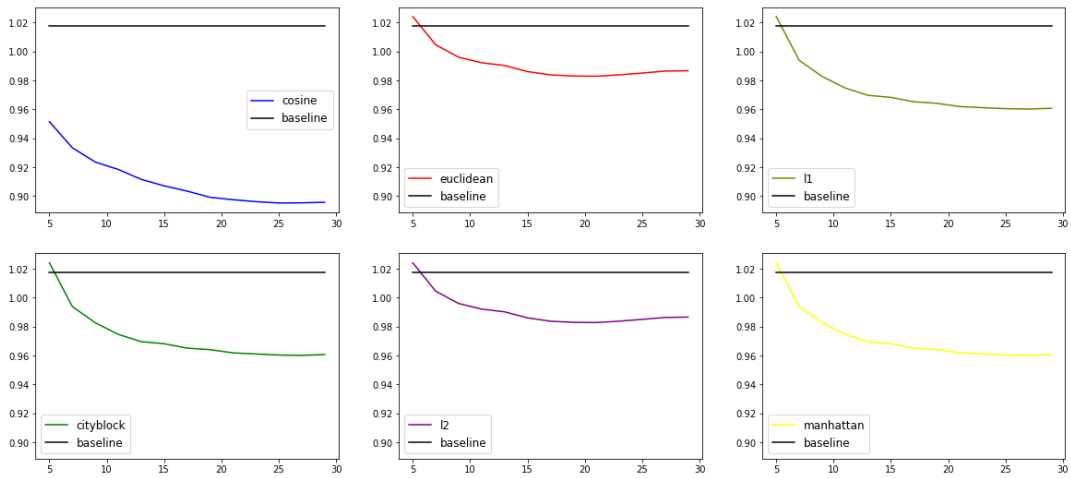
10.1.1. Hyperparameter Tuning

We consider these different types of metrics for considering the distance between any two things in the KNN :
1: ‘cityblock ‘
2: ‘cosine’
3: ‘euclidean’
4: ‘l1’
5: ‘l2’
6: ‘manhattan’

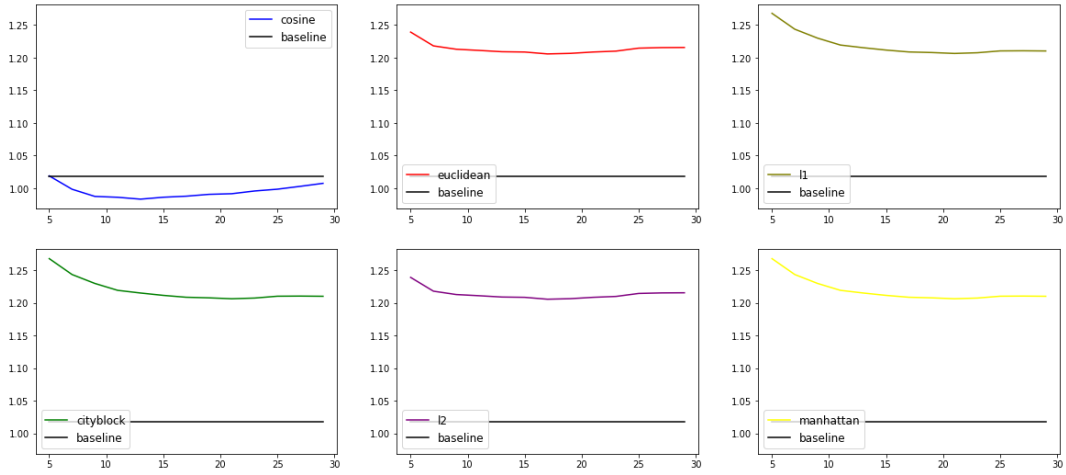
For the values of k we take into consideration the values from 5-30.

We then apply the grid search over the values into consideration and find the best values of these hyperparameters for each of the different samples.

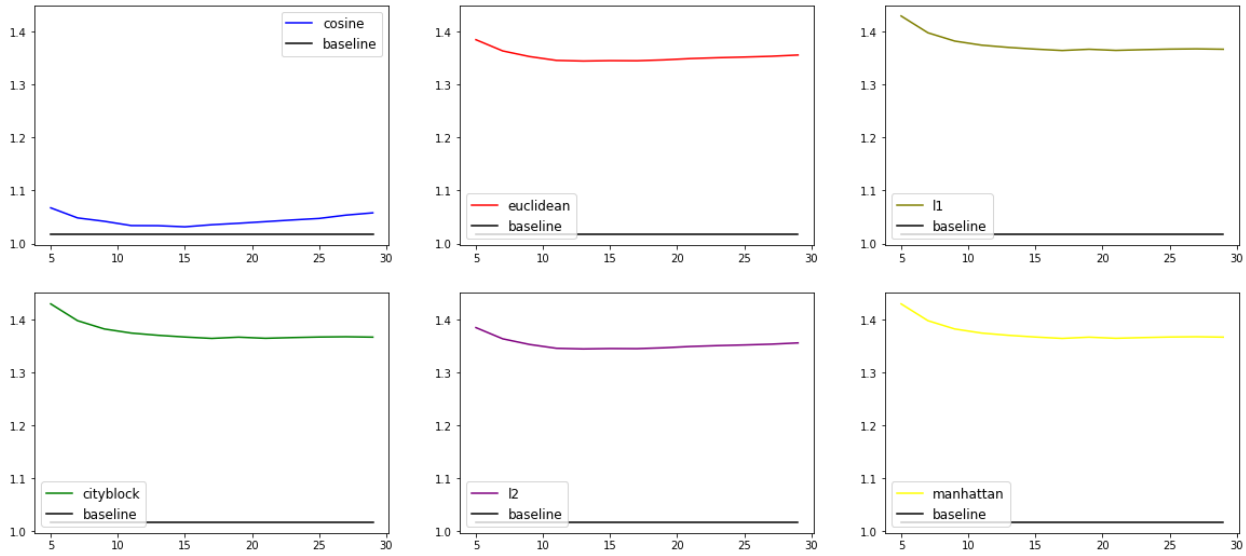
Following 4 graphs show the behaviour of RMSE for **user-based** nearest neighbours method. The graphs show RMSE values for different metrics and are ordered in the manner of the samples as described above.



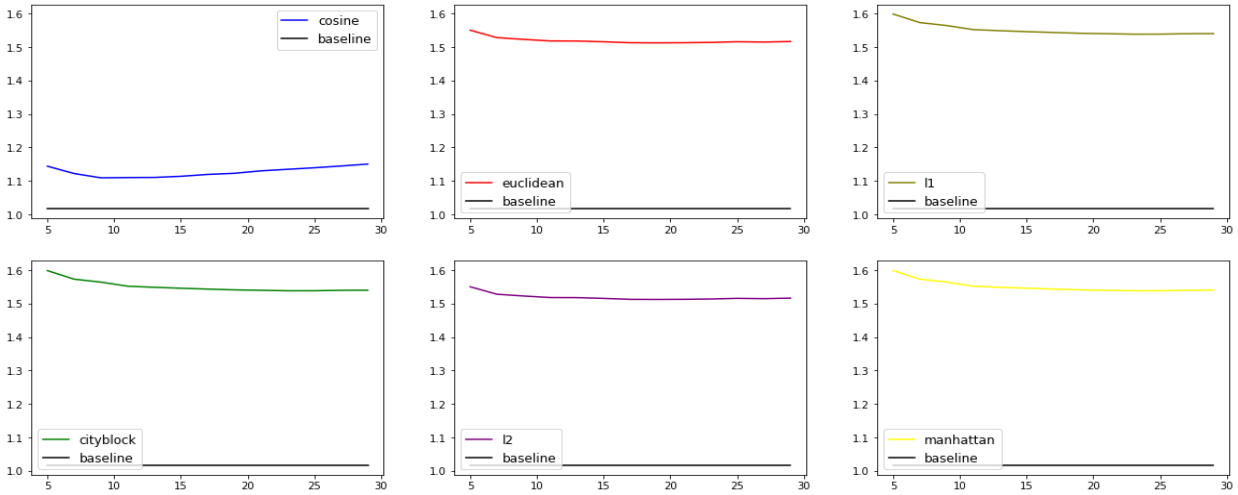
Best value of k:25 Best RMSE:0.895



Best value of k:13 Best RMSE:0.983

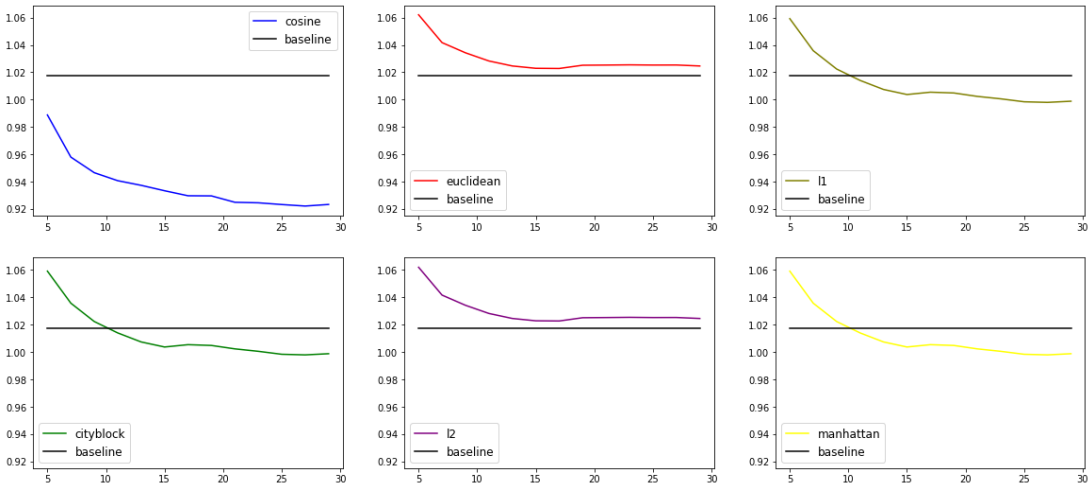


Best value of k:15 Best RMSE: 1.031

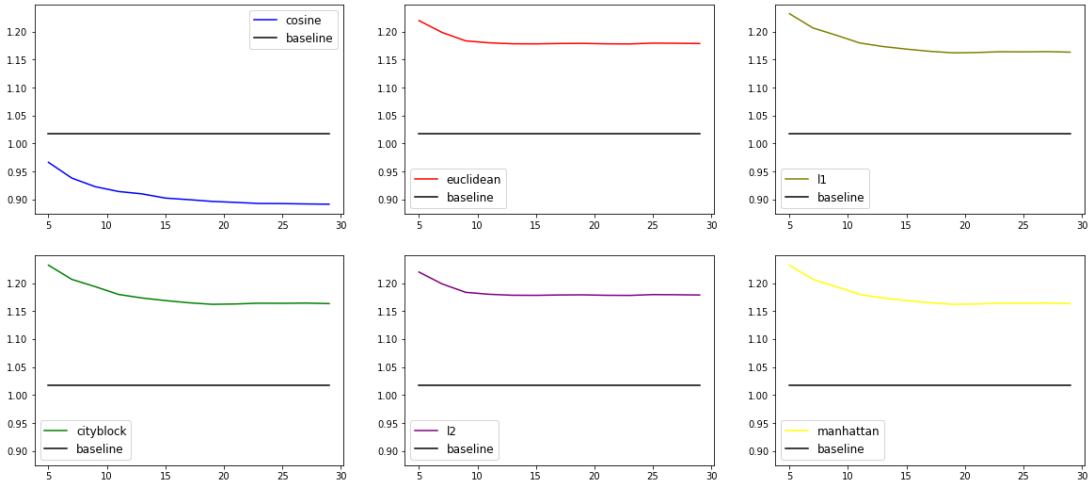


Best value of k:9 Best RMSE: 1.109

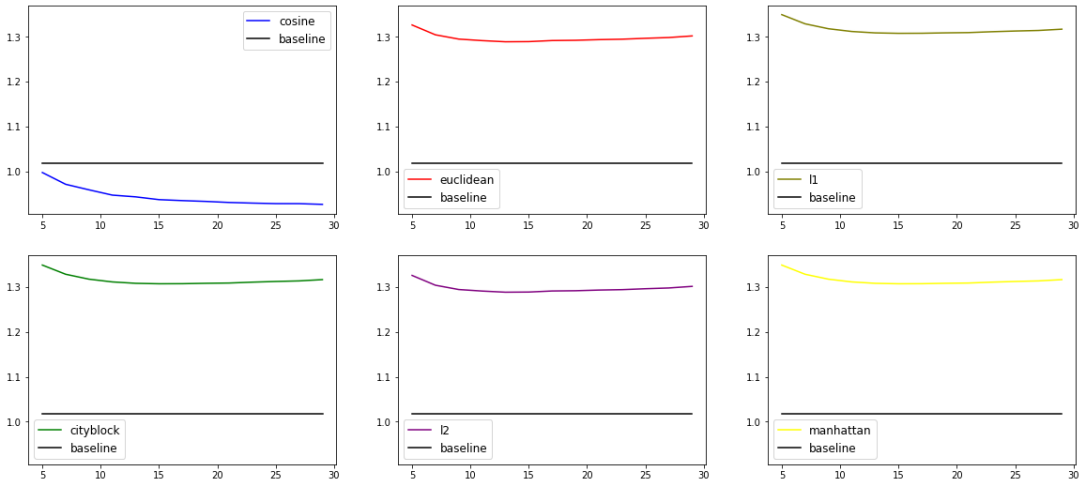
Following 4 graphs show the behaviour of RMSE for **item-based** nearest neighbours method.



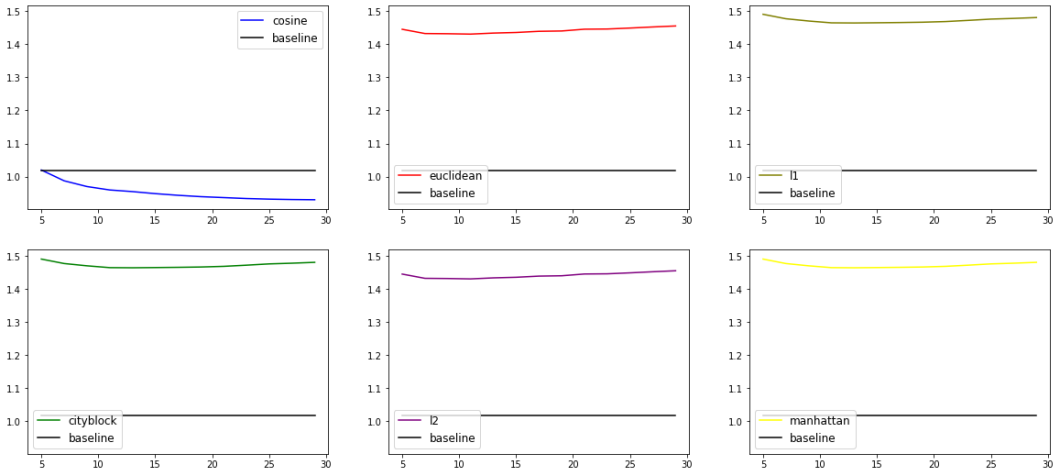
Best value of k:27 Best RMSE:0.922



Best value of k:29 Best RMSE:0.891



Best value of k:29 Best RMSE: 0.926

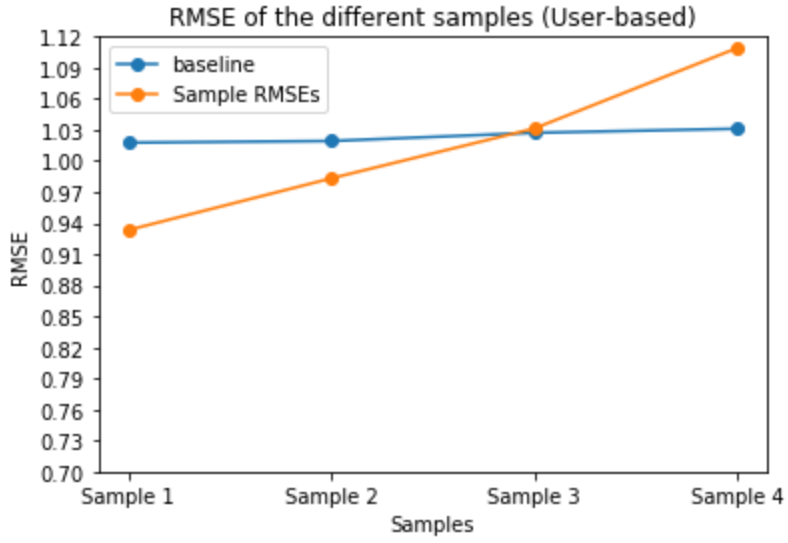


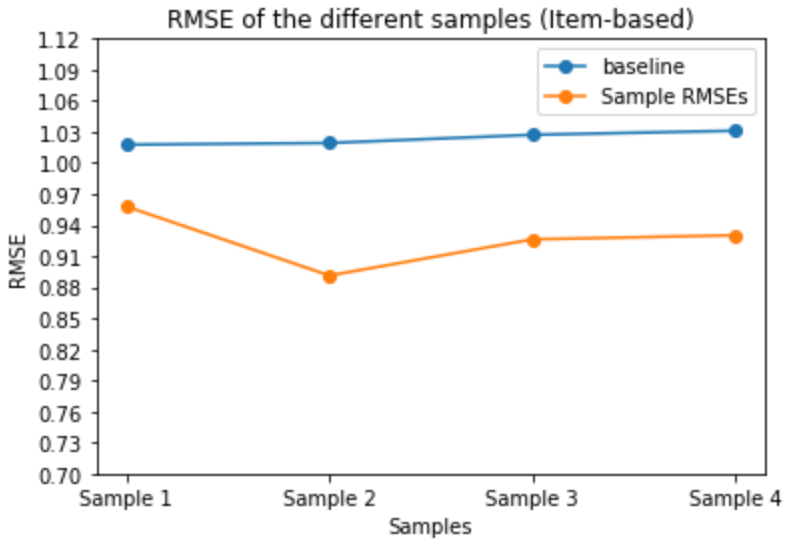
Best value of k:29 Best RMSE: 0.93

As seen from the graphs we obtain the **best value of RMSE when using cosine similarity** and it follows the intuition as the cosine similarity does consider the directions while eliminating the dependence on the magnitudes. Also, the item based similarity performs better than the user-based similarity for filtering.

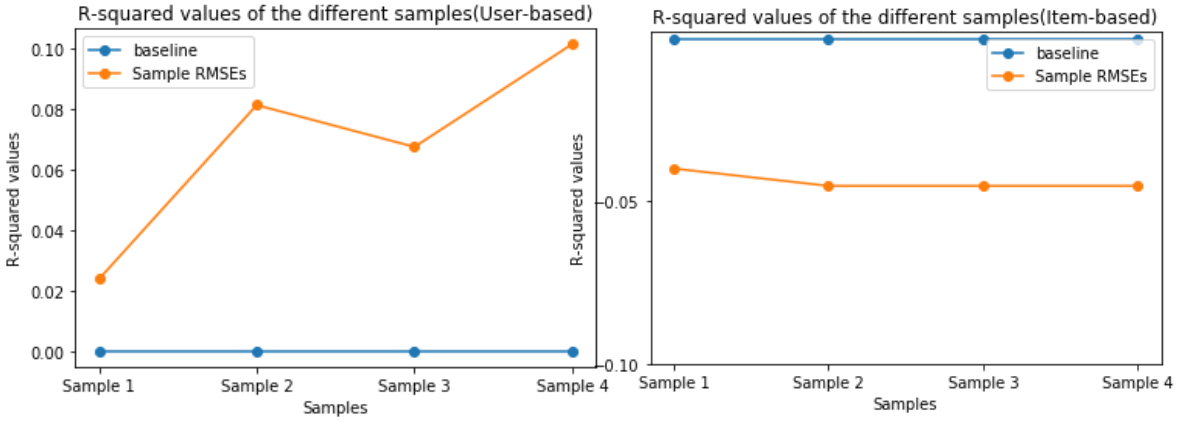
10.1.2. Evaluating the systems (RMSE and R2)

The following two graphs shows the change in RMSE for the samples considering the best values of the hyperparameter. The baseline for each sample is also plotted to compare the performance of the model to the baseline





As, it can be seen the RMSE values for the Item based model is the lowest for the first sample and as we change the sample size making it smaller from left to right (as we are increasing the number of movies into consideration for filtering the ratings)

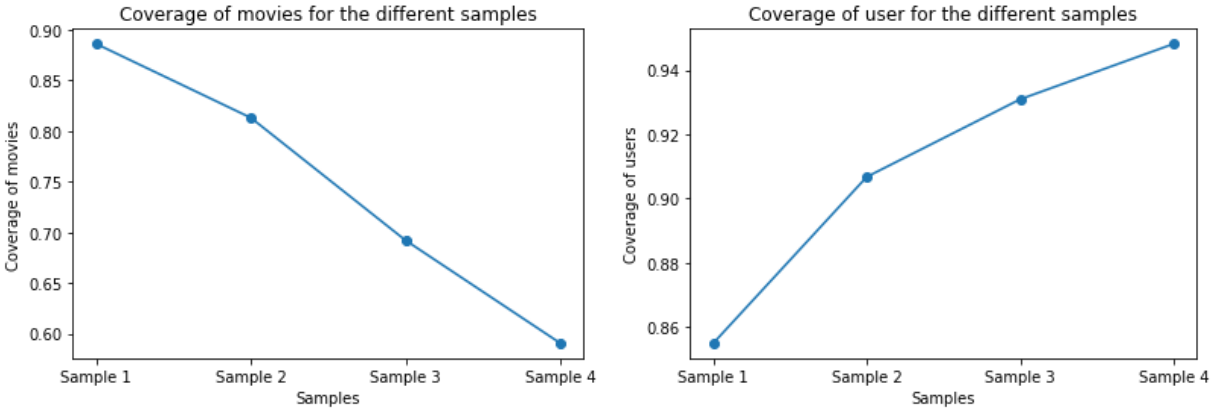


10.1.3. Runtime Scalability

The technique of using KNN won't be scalable when using more data points as it requires to compute the pairwise distance for each and every pair of users/items and as and when the number of user/items increase it would linearly increase the number of computations and hence, the technique won't be feasible to scale with our input.

10.1.4. Coverage

The coverage for the models are as shown in the following two graphs:



We can see here that we are covering more and more users when we increase the sample size.

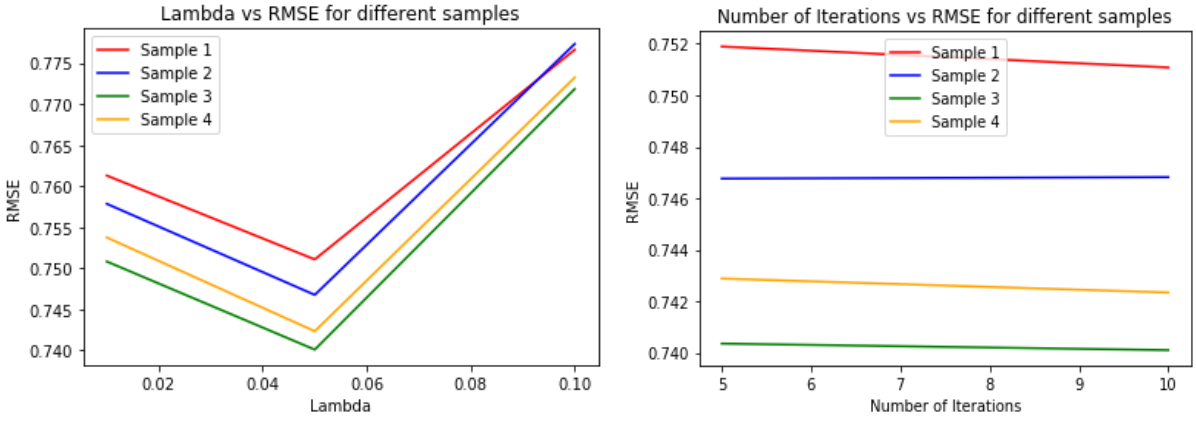
But, for the movies, we are covering lesser number of movies as the sample size increases. This is because our system is built towards recommending more popular movies, because of which we are leaving out the other ones. An important future scope for this movie could be tuning our model for the movies that have been rated lesser number of times. The second plot can be thought of as a trade-off to cover most of the users. We want to make sure that our model recommends 10 movies to the users, thus we are concentrating more on the top movies.

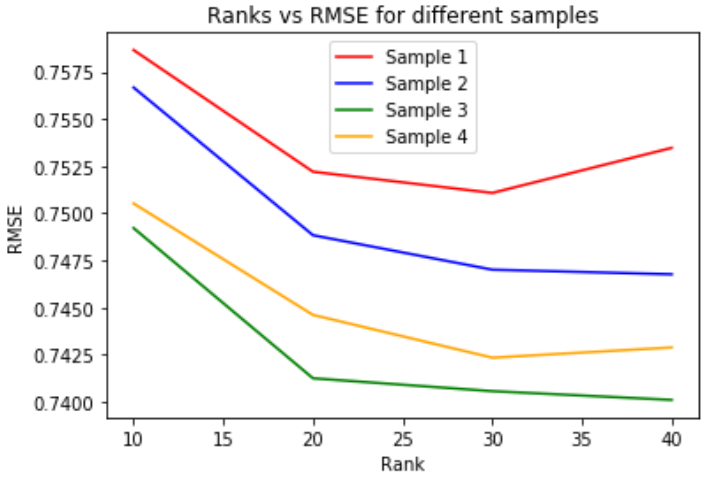
10.2. ALS

Once we have our samples in place, we divide them into 3 datasets, namely, training, tune and test.
We use the 'tune' dataset to tune our hyperparameters (that is, Ranks, Regression Parameter (lambda) and Number of Iterations (numOfIters)).
Note here that we tune our hyperparameters based on our primary accuracy metric that is RMSE.

10.2.1. Hyperparameter tuning

Below are some plots that depict the change in RMSE for different values of the hyperparameters -

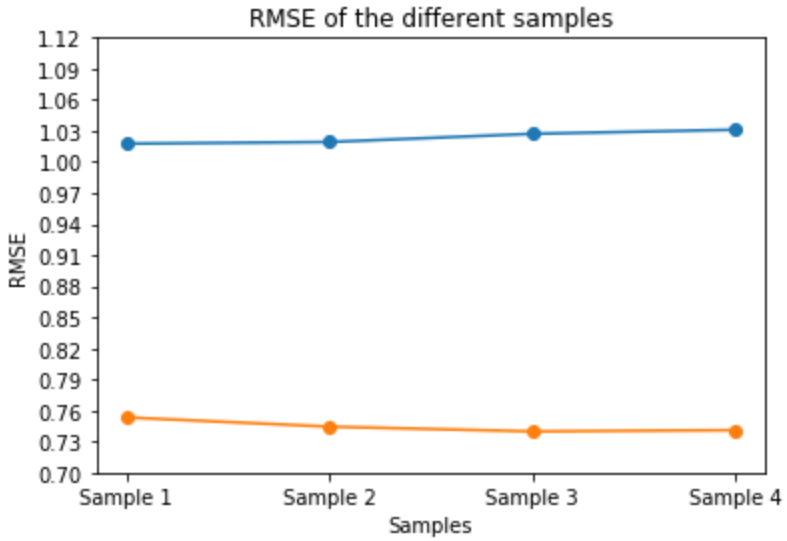




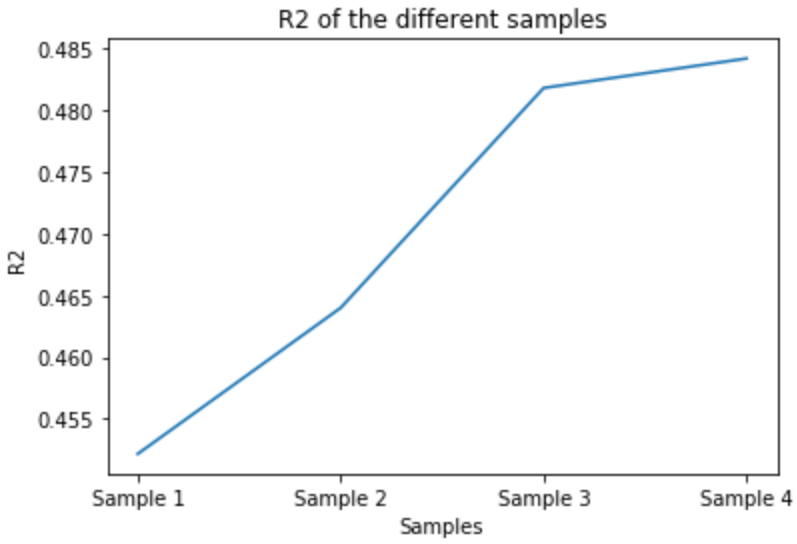
From the above graphs we can see the best values (of the hyperparameter) for the different samples.

10.2.2. Evaluating the model (RMSE and R2)

Now, we use these values to test our model using the 'test' dataset. After testing our model, we note the RMSE and the R-Squared (R2) values to evaluate our model. Note - we calculate the RMSE of our baseline model and plot it against the RMSE of the different samples to see how good model is performing.



NOTE: Baseline(Blue); Sample RMSE(Orange)



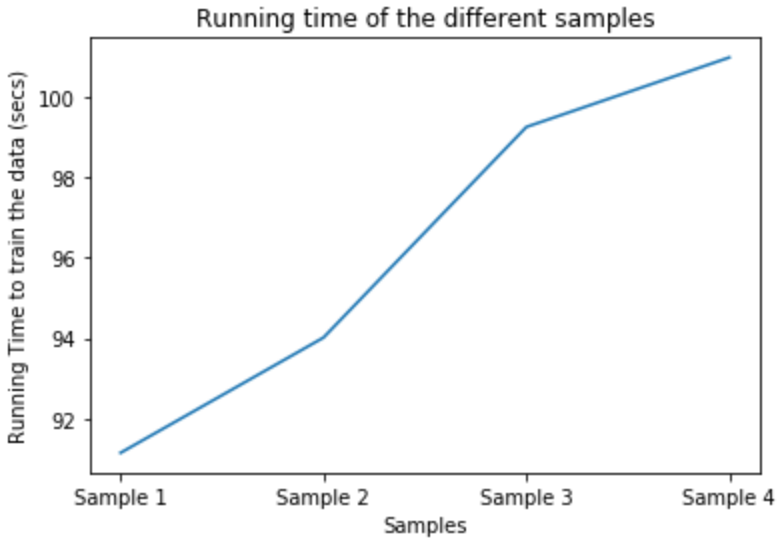
We can clearly make some conclusions here -

1. Firstly, we can see in the RMSE plot that our model is performing better than the baseline(coloured in blue). The best RMSE value we are getting is - 0.74

2. We can see that the accuracy of our model is becoming better on increasing the sample size. However the change is very minute.

10.2.3. Runtime Scalability

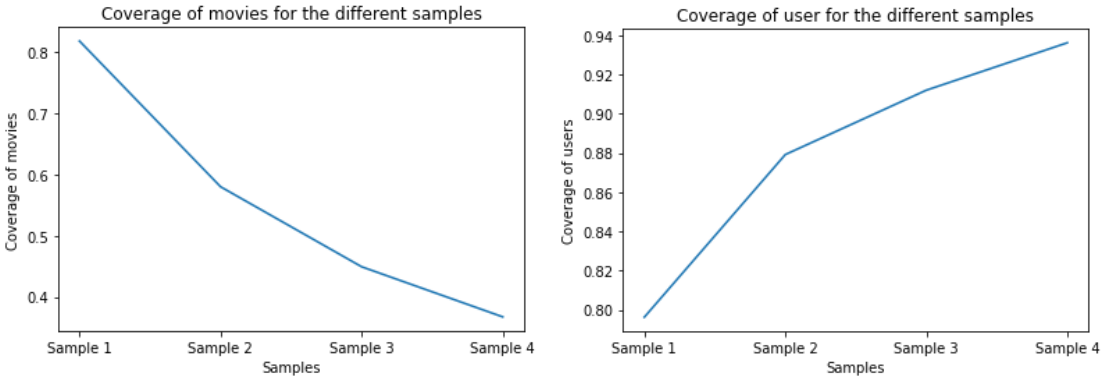
To understand how the run-time is scaling with the size of the sample, we plot the running time to train our model for the different samples -



After looking at the plot we can conclude that the running time of the model is increasing as the sample size increases. This shows that if we increase the sample size to around 1M rows (that is make it very big), our model will take a lot of time to run.

10.2.4. Coverage

After finding the coverage for the different samples, we get the below plot -



We can see here that we are covering more and more users when we increase the sample size.

But, for the movies, we are covering lesser number of movies as the sample size increases. This is because our system is built towards recommending more popular movies, because of which we are leaving out the other ones. An important future scope for this movie could be tuning our model for the movies that have been rated lesser number of

times. The second plot can be thought of as a trade-off to cover most of the users. We want to make sure that our model recommends 10 movies to the users, thus we are concentrating more on the top movies.

11. Future Scope

After seeing the plots and analyzing our accuracy metrics, we come to the conclusion that there are many areas in which we can improve our models. Some of them are as follows -

1. The first way to improve the accuracy could be to use implicit features also while building our model. For example, we can try and find similarity between movies based on their genres. Or we can find some other implicit features from the links to the different websites provided in the dataset. This can be done using web scraping.
2. As the data is very big, we can use the concept of approximate nearest neighbors and improve our accuracy.
3. We can also edit our code to handle the cold start problem. We believe that after handling this problem, we might be able to improve the performance of our systems when implemented in real time.