

1<sup>st</sup> Capstone project milestone report:  
**"Exploring Netflix's movie recommendation system"**

**1. The Problem:**

Recommender systems are a class of information filtering that is employed in multiple online services/platforms (e.g. movies, music, books, etc.), where the extremely large number of available options makes it impossible for a single consumer to have explored and evaluated all of the possible products. In this capstone project, I plan to explore algorithms used in a movie recommender system.

**2. Who is the client?**

Netflix is one of the most popular online entertainment platforms. The main product of Netflix is its on-demand video streaming service, which allows users to stream movies or television series through multiple electronic devices. As such, Netflix relies on a movie recommender system to provide users with a selection of digital content that users are most likely to enjoy. Therefore, having an accurate movie recommender system is of special interest to Netflix.

**3. Where can the data be obtained?**

The initial dataset comprising the Netflix prize dataset can be downloaded from kaggle (link: <https://www.kaggle.com/netflix-inc/netflix-prize-data>)

The original dataset was preprocessed and regrouped into 4 .txt files, comprising a total of 100480507 ratings by 480189 users for a collection of 17770 movies. A separate .csv file contains the list of name of movies and year of release.

**1. Data Wrangling Steps:**

**(\*\*\*: all of the ipython notebooks mentioned here are included in the same folder as this report)**

The data comes in the form of four .txt files, each with the following format:

Movie X:

Customer ID\_A, rating, date of rating

Customer ID\_B, rating, date of rating

....

Movie Y:

Customer ID\_C, rating, date of rating

Customer ID\_D, rating, date of rating

My first step was to use the code in "**data\_processing.ipynb**" to compile the 4 txt file into a single dataframe, where each row corresponds to 1 movie rating by one specific user. The columns of the resulting data frame are: "User ID", "Rating", "Date", "Movie ID".

The following image shows how this dataframe looks like:

|   | User ID | Rating | Date       | Movie ID |
|---|---------|--------|------------|----------|
| 1 | 1488844 | 3      | 2005-09-06 | 1        |
| 2 | 822109  | 5      | 2005-05-13 | 1        |
| 3 | 885013  | 4      | 2005-10-19 | 1        |
| 4 | 30878   | 4      | 2005-12-26 | 1        |
| 5 | 823519  | 3      | 2004-05-03 | 1        |

I also used the code in **"movie\_omdb\_request.ipynb"** to retrieve the genre to which each movie belongs. For this, I used the "omdb" python package, which allows performing queries to the "Open Movie Database (OMDb)" api (<http://www.omdbapi.com/>). Unfortunately, this database does not contain information for all of the movies in the Netflix Prize data set. Regardless, I was able to retrieve the genre information for ~72 % of the all the movies (17770), which corresponds to ~90% of the total ratings.

The final steps for data wrangling are contained in **"data\_processing\_2.ipynb"**. The genre information retrieved from the omdb api came in the form of a single string containing the respective genres. That information was added to the original csv file with the list of movies. The resulting dataframe is shown below:

|    | Name                         | Year | Genres                         |
|----|------------------------------|------|--------------------------------|
| 1  | Dinosaur Planet              | 2003 | Documentary, Animation, Family |
| 2  | Isle of Man TT 2004 Review   | 2004 | None                           |
| 3  | Character                    | 1997 | Crime, Drama, Mystery          |
| 4  | Paula Abdul's Get Up & Dance | 1994 | None                           |
| 5  | The Rise and Fall of ECW     | 2004 | None                           |
| 6  | Sick                         | 1997 | Short, Drama                   |
| 7  | 8 Man                        | 1992 | Action, Sci-Fi                 |
| 8  | What the #\$*! Do We Know!?  | 2004 | None                           |
| 9  | Class of Nuke 'Em High 2     | 1991 | None                           |
| 10 | Fighter                      | 2001 | Action, Drama                  |

For subsequent data analysis, I opted to add to the list of movies dataframe one column for each available genre. For this columns, 'Y' means that the respective movie belongs to that genre, and 'N' that it doesn't. The following image shows what the resulting data frame looks like:

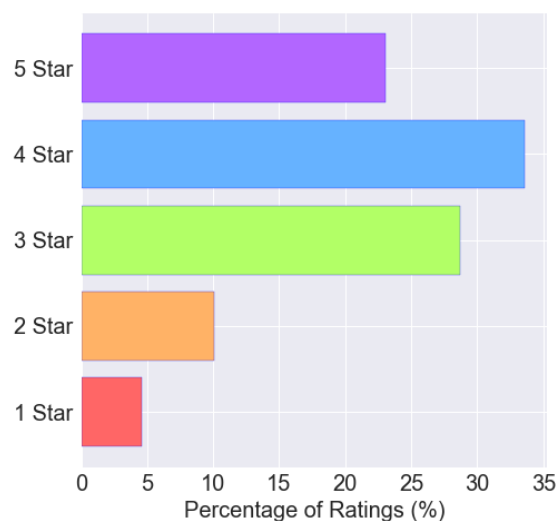
|   | Name                         | Year | Genres                         | Sci-Fi | Crime | Romance | Animation |
|---|------------------------------|------|--------------------------------|--------|-------|---------|-----------|
| 1 | Dinosaur Planet              | 2003 | Documentary, Animation, Family | N      | N     | N       | Y         |
| 2 | Isle of Man TT 2004 Review   | 2004 | None                           | N      | N     | N       | N         |
| 3 | Character                    | 1997 | Crime, Drama, Mystery          | N      | Y     | N       | N         |
| 4 | Paula Abdul's Get Up & Dance | 1994 | None                           | N      | N     | N       | N         |
| 5 | The Rise and Fall of ECW     | 2004 | None                           | N      | N     | N       | N         |

The index of this dataframe corresponds to the movie id from the original dataset.

## 2. Exploratory data analysis:

The purpose of this exploratory data analysis is to identify possible variables in the Netflix Prize Data set that could potentially act as important factors in predicting movie ratings. The specific details of this analysis are contained in "**EDA.ipynb**".

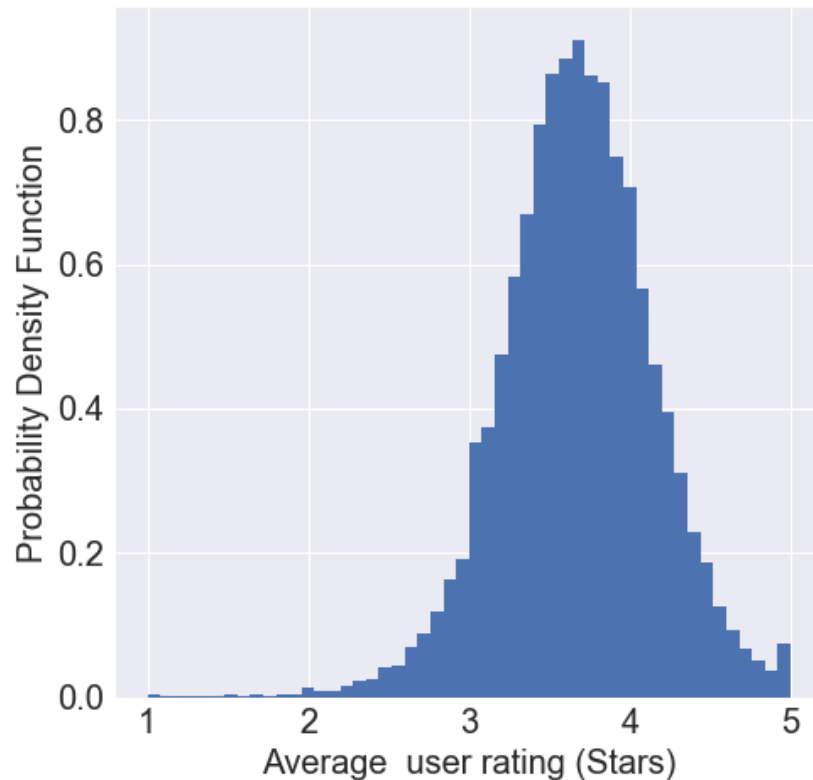
To start with, since we want to eventually predict movie ratings, I looked at the overall distribution of the movie ratings, as shown in the following bar graph:



The previous bar plot shows that the most frequent movie rating is 4 stars. I also calculated that the majority of the ratings (~ 56%) received a good rating (4 or 5 Stars). Overall, the average movie rating is ~ 3.6 Stars, with a standard deviation of ~1.09 Stars.

### **Do some users have bias when giving ratings?**

To answer this question, I looked at the distribution average user rating (plot below):

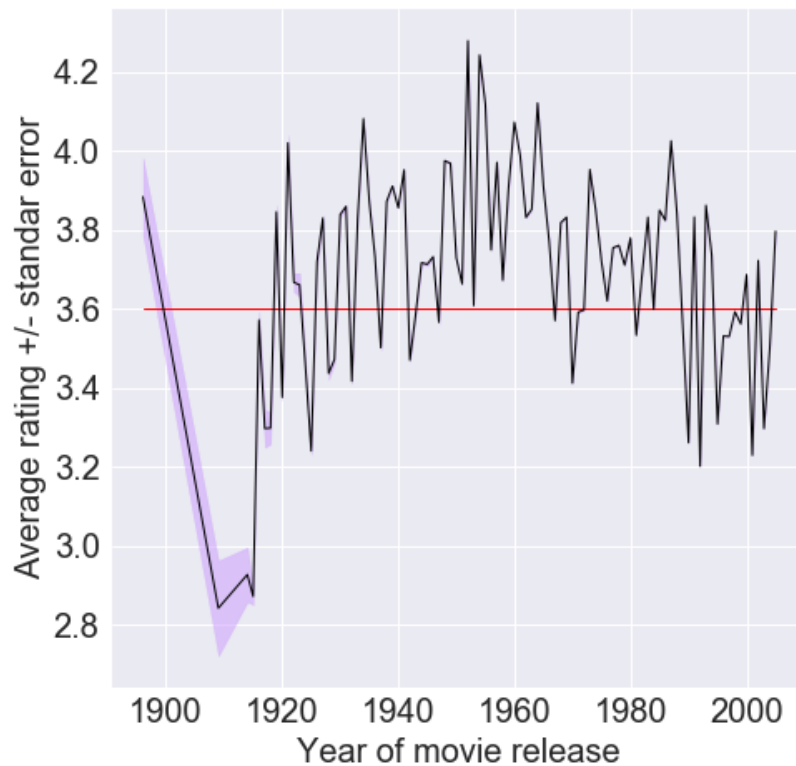


Looking at the above histogram, it seems that the average user rating is normally distributed. Indeed, running a normality test ( $pvalue < 0.05$ ; `normaltest()` from `scipy.stats`), confirms this observation. If we assume that users with an average ratings above 4 Stars or below 3 Stars average have a positive or negative bias when evaluating movies, I saw that ~30% of users have a bias when rating movies. Thus, it could potentially be informative to include a user-specific bias in a movie rating predictive model.

Of note doing a quick calculation, turns out that on average each user gave ~209 ratings. Surprisingly, the biggest number of ratings given by a single user was 17653!!!

### Does the movie release year have an effect on ratings?

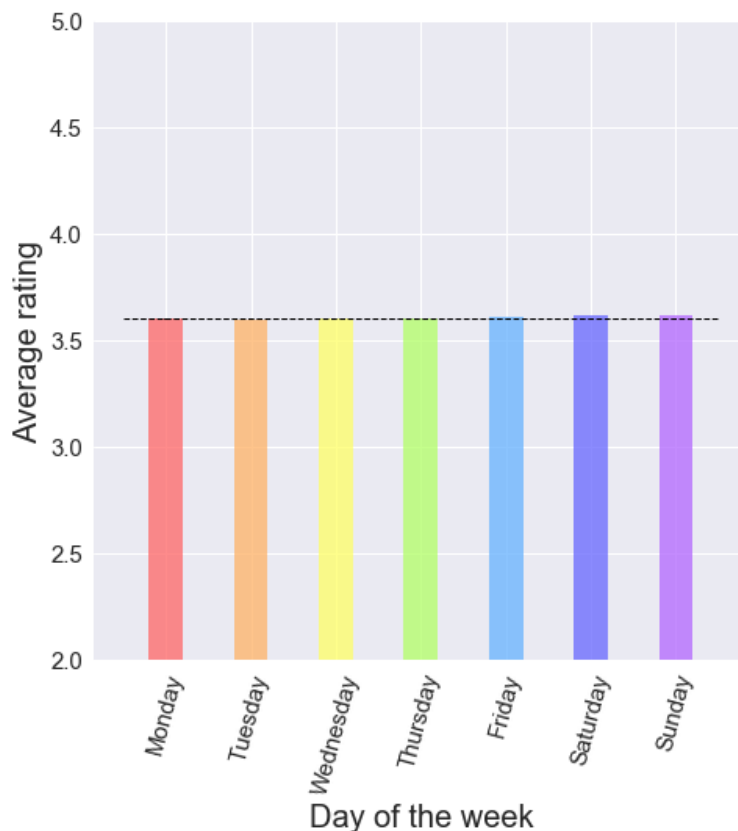
Since the Netflix prize data set also includes the year in which a movie is released, I also analyzed if movies released on different years had different ratings. For this I grouped the data by movie's release year, and plotted the average rating for each year.



The previous plot shows the average rating for each year (black line) +/- the standard error (shaded region around black line). The red line represents the overall average rating (~3.6 Stars). The small standard error for most years shows that the average rating on that year is different than the overall average rating. This suggests that the year in which the movie was released may be an important factor affecting the rating predictions. In fact, using a one-sample t-test to compare them to the overall average (considering pvalues < 0.001 as indicating significance), showed that 89 out of the 94 different movie release year are significantly different than the overall average. Thus, the year in which a particular movie was release may have significant effect on the predictions of movie ratings.

## Does the day of the week have an effect on the ratings?

Another piece of information on the Netflix prize dataset is the date in which a rating was given. I used this information to test if the days of the week could have a significant effect in the movies rating. For this, I used the "datetime" module to convert the date into a day of the week, and grouped the data accordingly.



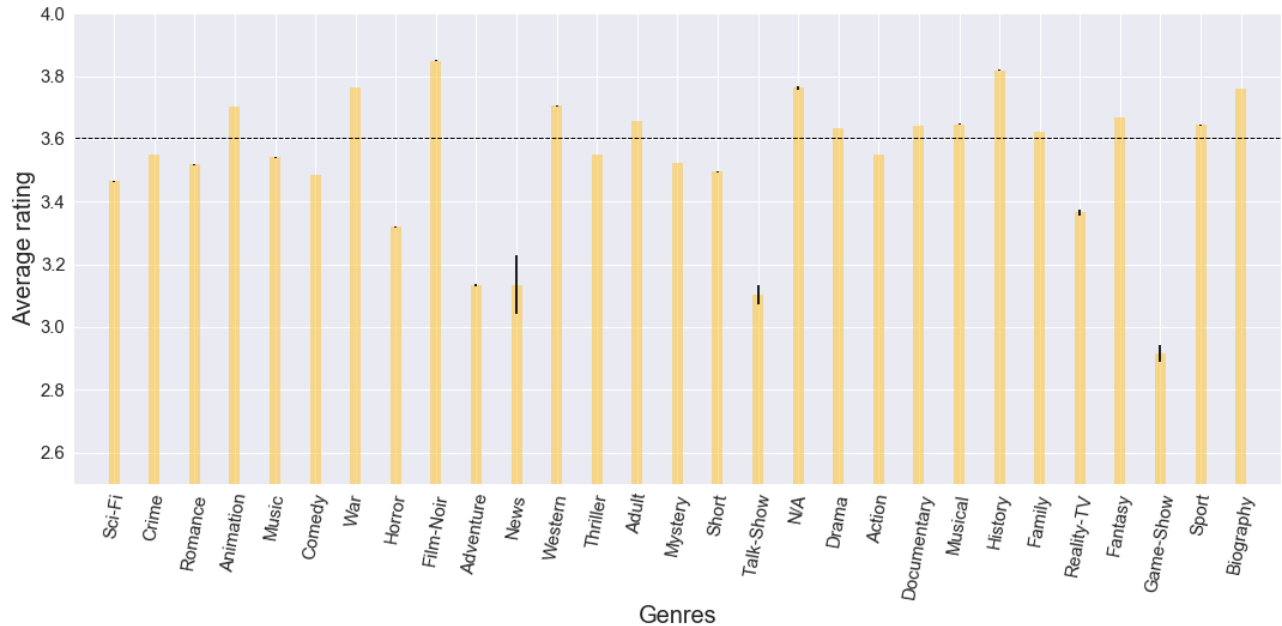
When extracting the data from the main dataframe, a one sample t-test was used to assess if the ratings for a respective day of the week were different from the overall mean (~3.6). Significant differences were considered when  $p\text{-value} < 0.001$ .

The one sample t-test revealed that Monday, Tuesday, Friday, Saturday and Sunday were different than the overall average. However when we look at the bar plot above we can see that, although some days had ratings different than the average (~3.6; dashed black line in the bar plot graph), the average rating for all days of the week is very similar to the overall average (~3.6; see the printed values above).

This suggests that the day of the week might not be an influential factor when predicting movie ratings.

## Is the rating of a movie affected by its genre?

I originally retrieved the movie's genre from the Open Movie database API (OMDb) with the idea of investigating if different genres tend to have different ratings.



The bar plot above (average genre rating  $\pm$  standard error), strongly suggests that the genre of a movie affects the rating of that type of movie. In fact running a 1-sample t-test, it was possible to determine that all of the genres differ significantly ( $p\text{-value} < 0.001$ ) from the overall mean (black dashed line). While some genres, like Family, don't differ much from the overall average rating, other genres, like Game-Show, show much bigger differences.

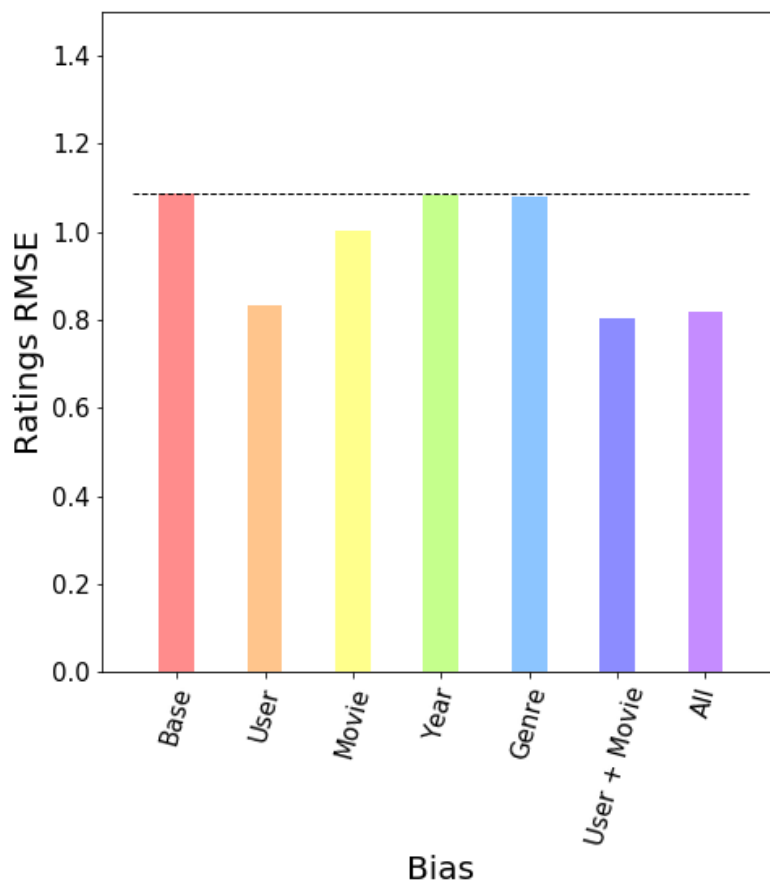
After performing this EDA, it can be concluded that factors such as 'User specific bias', 'Year of movie release' and 'Movie's Genre' significantly affect the rating of movies, thus, they can potentially be incorporated into a predictive model for movie preferences and ratings. In further steps, I would like to also calculate more specific user biases. For example, one could calculate the bias that a user might have when rating a particular genre.

### 3. Raw average as ratings predictor:

\*\*\*: \*Trying to analyze the entire NetFlix prize data set on a single computer proved to be too computationally intensive. Instead, I opted to take a random sample ('ratings\_subset\_plus\_biases.csv') of 1 million rating to work with.

A simple idea that can be implemented to start making movie predictions is the “Raw average”. Here, we calculate the predicted rating as the average rating of the entire data set. To measure the errors in the predictions I use the root mean squared error or RMSE. In addition, as suggested by the EDA presented previously, I incorporated user, movie, year of release and genre specific rating biases. For example, to calculate a user-specific bias I subtracted the data set average from the rating average for a specific user. The other biases were calculated similarly.

The resulting errors, measured by RMSE, are shown in the plot below:



The figure shows that incorporating specific rating biases gives a modest reduction to errors of the predicted ratings. We can see that a user-specific or movie-specific bias gives the greatest improvement. However, combining multiple biases (for example User-specific bias + Movie-specific bias) does not further improve the predicted rating.



#### 4. Using alternate least squares (ALS) for rating predictions.

Collaborative filtering (CF) approaches are commonly used in recommender systems.


























|   |  |  |  |  |
|---|---|---|---|--|
|  |  |  |  |  |
|  |   |  |  |  |
|  |  |  |  |  |
|  |  |   |  |  |
|  |  |  |  |  |

image source: <http://kurapa.com/wp-content/uploads/2014/08/image54.png>

The main idea of the techniques used in CF, as exemplified in the previous image, is to complete the missing entries (ratings) of a sparse users-by-items matrix, based on the ratings of similar users. One of such techniques, included in the pyspark MLlib, is Alternate Least Squares (ALS). In ALS, a number of latent (hidden) factors are used for predicting the missing entries (ratings) in the sparse user-by-movie matrix

The ALS model from MLlib has the following parameters:

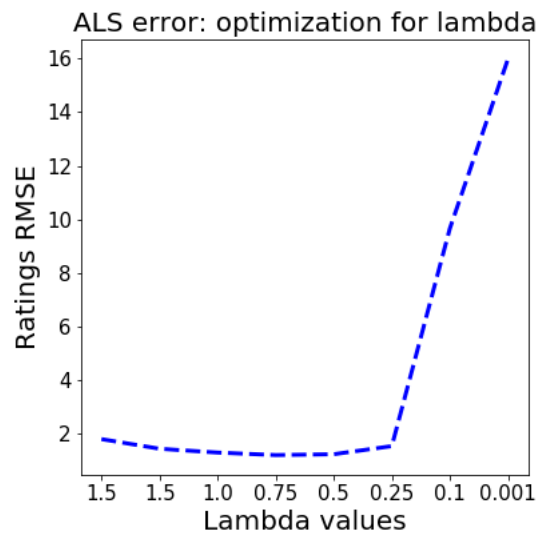
- ) numBlocks: is the number of blocks the users and items will be partitioned into in order to parallelize computation (defaults to 10).
- )rank: is the number of latent factors in the model (defaults to 10).
- )maxIter: is the maximum number of iterations to run (defaults to 10).
- )regParam: specifies the regularization parameter in ALS (defaults to 1.0).
- )implicitPrefs: specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data (defaults to false which means using explicit feedback).
- )alpha: is a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations (defaults to 1.0).
- )nonnegative: specifies whether or not to use nonnegative constraints for least squares (defaults to false).

More details on the analysis shown here can be found at:

[https://github.com/masaver/springboard/blob/master/capstone1\\_explring\\_netlix's\\_movie\\_recomender\\_system/final%20report/movie\\_recomender.ipynb](https://github.com/masaver/springboard/blob/master/capstone1_explring_netlix's_movie_recomender_system/final%20report/movie_recomender.ipynb)

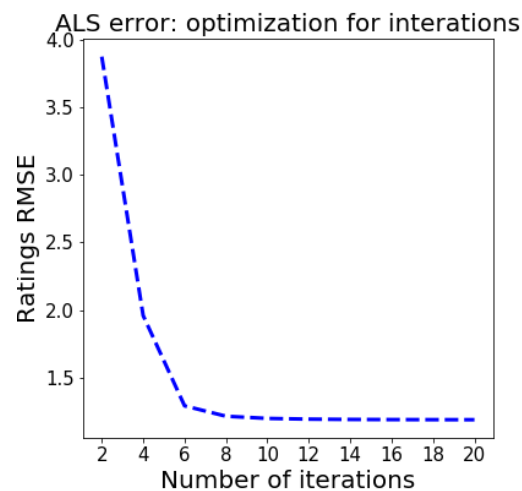
As a first step in implementing an ALS model was to optimize for the hyperparameters rank, maxIter and regParam. To do this I kept two of the parameters constant, while changing the third one. As before, to measure the errors I used RMSE.

- **ALS implementation: optimizing for regParam:**



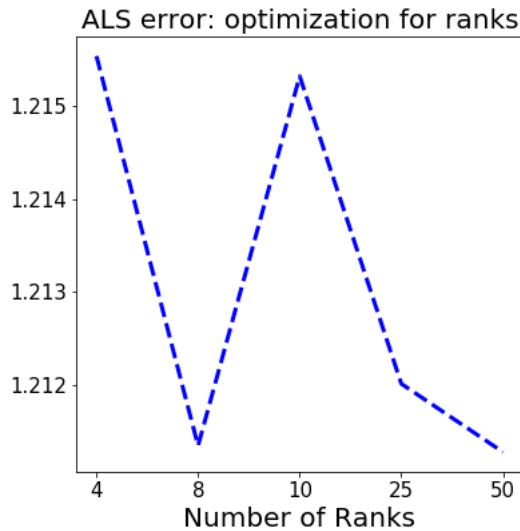
When optimizing for regParam (or lambda), the lowest RMSE was achieved with an regParam of 0.5.

- **ALS implementation: optimizing for number of iterations:**



The RMSE doesn't seem to significantly decrease after 8 iterations. Hence, I'll take 8 as the optimal number of iterations.

- **ALS implementation: optimizing for number of ranks:**

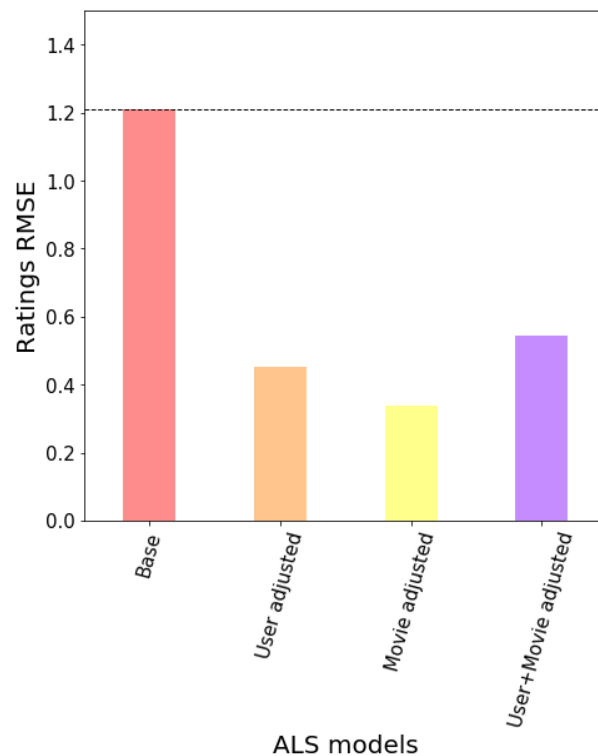


In this case the lowest RMSE was obtained when using 8 ranks. Hence, I'll take 8 as the optimal number of iterations.

- **ALS implementation optimal parameters:**
  - **regParam** = 0.5
  - **interactions** = 8
  - **ranks** = 8

Once the optimal parameters were determined, I trained the ALS model I used for predictions. As suggested from the previous analysis, I also adjusted the predicted ratings from the ALS model using user-specific and movie-specific biases. Of note, I noticed that, for a reason I could not determine/understand, the ALS outputs predictions outside the 5-Star scale used in Netflix's ratings. Therefore, I adjusted the ratings so that a predicted rating higher than 5 would be set to 5, and ratings lower than 1 would be set to 1.

The following figure shows the respective errors (measured by RMSEs) for the base and adjusted ALS models:



We can see from the plot shown above that we obtain a lower error in ratings prediction, when we adjust the predictions from our ALS model using the user and movie specific rating bias alone. Hence, for generating movie predictions I will only take into account the movie-specific bias.

To predict movie preferences for a new user (not included in the dataset), I created a fake list of ratings in the format (User ID, Movie ID, Rating). This list is shown below:

```
new_ratings = [  
    (0,14941,4),# The Matrix  
    (0,14928,4),# Dead Poets Society  
    (0,5344,5),# Fullmetal Alchemist  
    (0,10463,4),# Pokemon: The First Movie  
    (0,10453,4),# Pokemon Advanced  
    (0,5732,4),# Good Will Hunting  
    (0,15096,5),# The Notebook  
    (0,17132,5),# Waking Life  
    (0,11763,3),# Serendipity  
    (0,178,5) #A Beautiful Mind  
]
```

This list of ratings was added to the existing data set. The updated data set was used to train an ALS model incorporating a movie-specific bias only. As before, the output ratings were adjusted to stay within the 1 to 5 stars scale from Netflix. I will consider that any movie with a predicted rating of 4.5 Stars or more should be recommended to the user. The image below shows a sample of 20 recommended movies, with a rating of at least 4.5 Stars and a minimum number of 20 reviews on the dataset:

Sample list of recommended movies:

Angel: Season 5. Predicted rating: 5 Stars  
Spirited Away. Predicted rating: 4.67015305531 Stars  
Batman Begins. Predicted rating: 4.80449606347 Stars  
I. Predicted rating: 4.69739734497 Stars  
Buffy the Vampire Slayer: Season 4. Predicted rating: 5 Stars  
The Shield: Season 3. Predicted rating: 5 Stars  
Home Improvement: Season 1. Predicted rating: 4.69458404747 Stars  
Dark Angel: Season 1. Predicted rating: 4.79938823529 Stars  
Stargate SG-1: Season 4. Predicted rating: 4.76801740771 Stars  
Million Dollar Baby. Predicted rating: 4.64580944925 Stars  
Freaks & Geeks: The Complete Series. Predicted rating: 5 Stars  
Sex and the City: Season 1. Predicted rating: 4.74706793216 Stars  
Absolutely Fabulous: Series 1. Predicted rating: 4.64477690002 Stars  
Star Trek: The Next Generation: Season 6. Predicted rating: 4.94167663198 Stars  
Curb Your Enthusiasm: Season 4. Predicted rating: 5 Stars  
CSI: Season 3. Predicted rating: 5 Stars  
Braveheart. Predicted rating: 4.77114929443 Stars  
Finding Nemo (Widescreen). Predicted rating: 5 Stars  
Six Feet Under: Season 1. Predicted rating: 4.91393002626 Stars  
Life Is Beautiful. Predicted rating: 4.65389538027 Stars

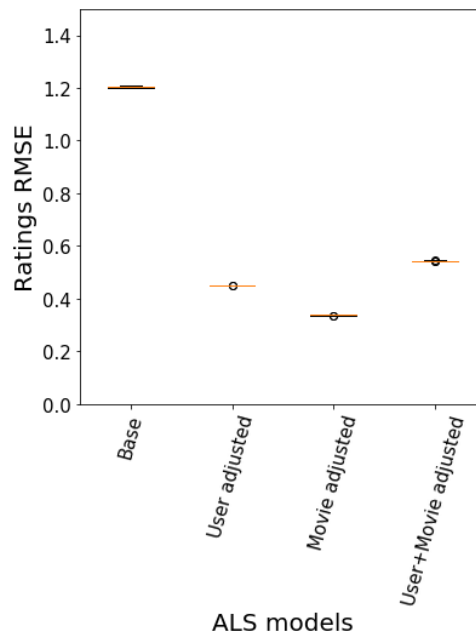
Total number of recommended movies: 419

From this sample of recommended movies, out of a total of 419 recommendations for the new user, we can see that there is some consistency in the type of movies being recommended, with respect to the original list of ratings for the new user. For example, if you liked "The Matrix" you might be likely to also like "Batman Begins" or "Stargate SG-1", or if you liked "A Beautiful mind" and "Good Will Hunting" you might be likely to also like "Braveheart" or "Life Is Beautiful". In addition, since the new user seemed to have liked anime media such as "Full Metal Alchemist" and "Pokemon", it would make sense to recommend "Spirited Away"

The only recommended movie/series in the sample list that seems a bit out of place, with respect to the original list of ratings for the new user, might be "Sex and the City".

## 5. How general are the results previously shown?

I also wanted to validate the ALS model using cross-validation. To answer this, I created 10 random samples from the original DataSet, each containing 1 million ratings. For each of this samples an ALS model was created, and also calculated the respective errors (RMSE).



The previous figure shows we can obtain a consistent error when training an ALS model, even if we use different samples of the original data set. This is evidence by the very small (almost non-existent) spread of values (ratings RMSE) in the boxplot above.

Furthermore, if we use one of these random samples to make movie recommendations to the new user we see that ~82% of the recommended movies are the same as in the first set of recommendations we obtained.

## 6. Can we obtain similar results/performance by using Singular Value Decomposition (SVD)?

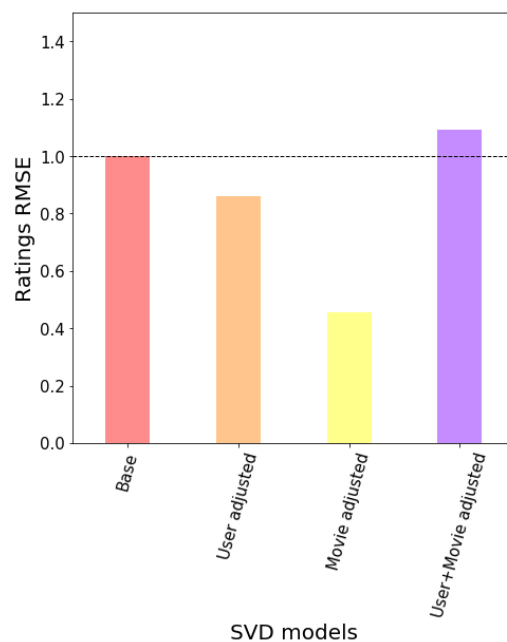
Many other methods can be used to estimate the predicted rating for a specific movie. In Singular Value Decomposition (SVD), for example, we start with a sparse matrix  $R$ , containing the available ratings for each movie/user pair. The matrix  $R$  can be decomposed into two unitary matrices,  $U$  and  $V$ , and a diagonal matrix  $D$ , such that:

$$R = UDV^T$$

In a general sense,  $U$  represents how much users “like” each feature and  $V$  represents how relevant each feature is to each movie. By taking the top  $k$  features (columns) of  $U$  and  $V$ , one can calculate a lower rank approximation of the original matrix, which is then used for making predictions of movie ratings.

Conveniently, 'Surprise' (<http://surpriselib.com/>) is a python sci-kit for recommender systems that easily allows to implement SVD.

Again, for implementing a SVD model, I used a random 1-million rating sample from the original dataset. I also used user or movie specific biases to adjust the predicted ratings. As before, the prediction errors are measured as RMSE.



As shown in the previous figure, the RMSE obtained without bias adjustments is  $\sim 1$ , which is slightly better than the basal RMSE ( $\sim 1.2$ ) obtained in the ALS model we implemented before. Once again, adjusting the predicted ratings by user or movie specific biases, greatly improves (lowers) the predictions error. Of note, a SVD model with both user and movie specific biases does worse than a SVD model with no adjustments.

Furthermore, as shown below, using a 5-fold cross-validation shows that we can obtain consistent prediction errors when implementing a SVD based recommender:

|                | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Mean   | Std    |
|----------------|--------|--------|--------|--------|--------|--------|--------|
| RMSE (testset) | 0.9974 | 0.9962 | 1.0004 | 0.9994 | 0.9993 | 0.9985 | 0.0015 |

Like before (see below), I added the same list of new ratings for a new user (User ID = 0) not previously present on the dataset, and proceeded to train a new SVD model with the updated dataset. To make recommendations, I used this new SVD model to predict ratings for movies not previously rated by the new user. Below, is a list of the top recommended movies:

Recomended movies for new user:

Dead Like Me: Season 2. Predicted rating: 5 Stars  
Lost: Season 1. Predicted rating: 5 Stars  
Lord of the Rings: The Return of the King: Extended Edition. Predicted rating: 5 Stars  
The Lord of the Rings: The Fellowship of the Ring: Extended Edition. Predicted rating: 5 Stars  
Seinfeld: Season 3. Predicted rating: 5 Stars  
Lord of the Rings: The Two Towers: Extended Edition. Predicted rating: 5 Stars  
Angel: Season 5. Predicted rating: 5 Stars  
The Simpsons: Season 6. Predicted rating: 5 Stars  
The Shawshank Redemption: Special Edition. Predicted rating: 5 Stars  
Firefly. Predicted rating: 5 Stars  
Cinderella: Special Edition. Predicted rating: 5 Stars  
Family Guy: Vol. 1: Seasons 1-2. Predicted rating: 5 Stars  
Nip/Tuck: Season 2. Predicted rating: 5 Stars  
Raiders of the Lost Ark. Predicted rating: 5 Stars  
Anne of Green Gables. Predicted rating: 5 Stars  
Gladiator: Extended Edition. Predicted rating: 5 Stars  
The Twilight Zone: Vol. 42. Predicted rating: 5 Stars  
Family Guy: Vol. 2: Season 3. Predicted rating: 5 Stars  
The West Wing: Season 3. Predicted rating: 5 Stars  
Gilmore Girls: Season 4. Predicted rating: 5 Stars  
Star Wars: Episode V: The Empire Strikes Back. Predicted rating: 5 Stars  
The Simpsons: Season 4. Predicted rating: 5 Stars  
The O.C.: Season 1. Predicted rating: 5 Stars  
Buffy the Vampire Slayer: Season 2. Predicted rating: 5 Stars  
CSI: Season 3. Predicted rating: 5 Stars

Doing a quick overview of the recommendations from both the ALS and SVD based recomenders, we see we can obtain similar results for the same new user. For example, in both cases we see recomendations such as "The Lord of the Rings", "Gladiator", "The Simpsons" or "Family Guy".



## 7. Conclusion and Future directions.

The NetFlix prize competition aimed at improving the accuracy of Netflix recommendation system, called CineMatch. The RMSE for the recommendations produced by this method as  $\sim 1$ . Both the ALS (basal RMSE:  $\sim 1.2$ ) and SVD (basal RMSE:  $\sim 1$ ) produced similar predictions errors. For simplicity, the analysis presented here was performed using only a subset (1 million) of the ratings in the NetFlix Prize, therefore a higher accuracy (lower prediction error) could be achieved by analysing the entire Netflix Prize data set on a system with higher computing power.

The NetFlix prize competition was won by combining multiple predictive models, with a RMSE of  $\sim 0.85$ . Surprisingly, here we obtained an even better (lower) prediction error ( $\sim 0.4$  RMSE), by simply adjusting the predicted ratings by the overall user or movie specific rating biases. Therefore, it is suggested that future recommender systems would greatly benefit from incorporating this type of biases in ratings predictions.

Of note, this type of movie recommender systems will most likely need to be revised, as Netflix has changed its 5 Star rating scale to a "Liked" or "Not Liked" system.

## 8. References.

"Collaborative Filtering - RDD-based API",

Link: <https://spark.apache.org/docs/2.2.0/mllib-collaborative-filtering.html>

"An on-line movie recommending service using Spark",

Link: <https://github.com/jadianes/spark-movie-lens/blob/master/notebooks/building-recommender.ipynb>

"Surprise: A python scikit for recommender systems",

Link: <http://surpriselib.com/>

"NetFlix Prize",

Link: [https://en.wikipedia.org/wiki/Netflix\\_Prize](https://en.wikipedia.org/wiki/Netflix_Prize)

"Winning the Netflix Prize: A Summary",

Link: <http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/>