# Million $ Netflix Challenge

Celal Karakoç / [redacted] / 4342933 / KaggleID: Celal Karakoc / KaggleScore: 0.84147
[redacted]

**STRICT 4 PAGE COUNT**

## I. Introduction

Within this report we will try to give our solution to the movie recommendation problem. That is, given a set of users and movies, we will seek to predict the rating that a user would give to a movie.

The relevancy for such a solution stems from the fact that there is too much data for a single user to go through to find what he or she likes. This can be applied from movies to products to connections with people. A recommender system in such cases brings convenience to the searching process and creates more potential customers. As such, these recommender systems are used by a vast amount of companies, like Netflix (movies), LinkedIn (Professional Network) and Amazon (products) to name a few. There is also a huge financial benefit to it, to quote Netflix executives Carlos A. Gomez-Uribe and Neil Hunt

> "We think the combined effect of personalization and recommendations save us more than $1B per year."**??**

methods: Collaborative Filtering Latent Factors

For my final predictor I only use the ratings. The age, profession and gender didn't have any effect on the outcome. **??**

Read up on several previous methods of your choice and discuss their approaches, including why they were successful. Connect these approaches to your own idea and motivate why you built your final predictor the way you did. For instance: it was shown that some user characteristics, such as gender, contain useful information as the performance of a user-user collaborative filtering model improved when it was added. Therefore, we decided to include such characteristics in our latent factor model as well. Make sure to specify how your approach differs from these existing approaches and do not forget to cite them in your references.

Finally, spend one or two lines describing how you structured your report (e.g. in section 2.2 we describe our deviation on a standard latent factor model).

## II. Methods

### A. Data

Within my Collaborative Filtering we ended up using only the ratings. As seen in **??** the addition of the age, profession and gender didn't improve upon the original approach. We did not gather additional data to train our recommender due to time constraints.

...

### B. Speedup gains

Throughout the project we also made several code changes to improve upon the speed of execution. This made our execution go from 2 hours to a mere 20-25 seconds. The first step that we took for achieving this result would be precalculating the similarities into a double array. The results of these similarities were serialized and saved into a *.dat* file. Furthermore, we also make use of parallel execution through streams. By both multithreading and precalculating our time was cut considerably. At last we had a improvement in our datastructures. To understand this speedup we need to look further into the code. Because we decided to choose for the top-N neighbourhood approach there was a bottleneck of getting the top-N neighbours within the list. In pseudocode:

For i to n in predictions: List similarities; // we get the list of similarities in O(n), because of the precalculation List topN; // Here we need to get the top-N Neighbours // further calculations...

The loop through the predictions will in a real world scenario be through millions of predictions. This would mean that getting the top N neighbours in $\mathcal{O}(n^2)$ time (by checking them n times for the greatest similarity) will scale really badly. A better approach would be to either sort the list and get the items 1 through N. Or we could also save our similarities in a PriorityQueue instead of a List. Both these methods would ensure a better performance, $\mathcal{O}(n \log n)$. This is still not ideal. Ideally, we would like to have a $\mathcal{O}(n)$ way of doing this. Hereby, our final improvement: the introselect algorithm. This algorithm combines both quickselect and the median of means. Basically, what happens is that we find the N'th largest value within the list in Linear time. And then when we have that value, we go through the entire list and grab all values greater than the N'th largest value. This ensures that our execution time will be shortened drastically and has resulted in the greatest speedup of our recommender.

### C. Measuring error

The Error we used to compare/measure our results is the Root Mean Square Error.

$$RMSE = \sqrt{\frac{1}{|R|}\sum_{(i,x \in R)}(\hat{r}_{xi} - r_{xi})^2} \tag{1}$$

Other evaluations we could use was the Mean Absolute Error:

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_i - \hat{y}_i| \tag{2}$$

I did not use this metric, but the difference lies mainly in that RMSE gives a relatively high weight to large errors, while MAE is steady in that regard.

### Collaborative Filtering

*1) User-User vs. Item-Item:*

*2) The Naive Approach:*

*3) Local and Global Effects:* Each user has a certain bias in rating things. Some are more generous with their ratings than others. We will try to solve this by having a certain bias added on top of our estimation. This means that for each rating for a movie $y$ by a user $x$, we will need to add a baseline estimate $b_{xy}$ on our final estimation:

$$b_{xy} = \mu + b_x + b_y \tag{3}$$

Within this we have the mean of all movie ratings ($\mu$), the (average rating of $x$) - $\mu$ ($b_x$) and the (average rating of $y$) - $\mu$ ($b_y$).

*4) Computing Similarities:* To compute the similarity between item $i$ and item $j$, we tested **??** similarity measures. In the end, we noticed that the best results were gained by the Pearson Similarity. The Pearson correlation coefficient (PCC) is calculated by the formula:

$$s(i,j) = \frac{\sum_{k \in I \cap J} (r_{ki} - \mu_i)(r_{kj} - \mu_j)}{\sqrt{\sum_u (r_{ui} - \mu_i)^2 \sum_u (r_{uj} - \mu_j)^2}} \tag{4}$$

*5) Neighbourhood Selection:* There were two ways we could decide the neighbourhood selection.

1) **Top-N filtering**: A list of N nearest-neighbors is selected according to the similarity.
2) **Threshold filtering**: A list of all the neighbors is selected that has a similarity that is larger than a particular threshold.

The most critical aspects of both of these are selecting either the threshold value for 2 or the value of N for 1. Since I decided to go with the first I found out that a neighborhood with N=17 is the best. This N has come out of trial and error with several N-values, once, ranging from 100-15.

*6) Final rating:* With all this in mind, and all of the above, the formula we acquire for our final CF algorithm is:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}} \tag{5}$$

Where $N$ is the set of top N neighbours, $s_{ij}$ is the PCC of item $i$ and $j$ and the $b_{xi}$ and $b_{xj}$ are the local and global baselines respectively.

### E. Latent factors

.. TODO

## III. Results

.. TODO

Describe how your algorithm performs; do not just give a single number but tell us what the effect is of adding certain components (e.g. how much it improved by adding temporal biases). What was its cross-validation performance and how much did that differ from its Kaggle leaderboard performance?

This section should include a table or a figure describing your results. Remember that those need to have legends and descriptions on the x- and y-axis. It should be so clear that anyone, including non-computer science students, can interpret it correctly.

This section should be roughly 1.5 page long.

## IV. Discussion

.. TODO

Summarize the results of your efforts; do not simply repeat but tell us what worked and what did not. Use your summary as a reasoning to explain what would be a good step to take in the future. For instance: We noticed that item-item works better than user-user collaborative filtering and after inspection of the cosine distances, we saw that the movies are more similar to each other than the users. Therefore, we believe that an interesting future direction would be to include more movies in the database to increase the chance of finding very similar movies to the one that needs to be predicted.

Reflect on how you approached the project, what you liked and did not like, and what you could have done to be more efficient. In particular, highlight what you have learned through your efforts.

This section should be roughly 0.5 page long.

## V. References

APA STYLE - Barrass, Robert. Scientists Must Write: A guide to better writing for scientists, engineers and students. Psychology Press, 2002.

- Burchfield, Robert William, ed. The new Fowler's modern English usage. Oxford: Clarendon Press, 1996.

- Dawson, Christian W. The essence of computing projects: a student's guide. Prentice Hall, 2000.

## VI. Introduction

dioajefisjoefijseof

## VII. Methods

### A. *Data*

dioajefisjoefijseof

### B. *Benchmarking*

dioajefisjoefijseof

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (\hat{r_i} - r_i)^2}{n}} \tag{6}$$

### C. *Implementation*

### D. *Collaborative Filtering using Near-Neighbor Search*

hbdioajwdi

*1) Baseline:* Each rating for movie $m$ by user $u$ consists of a baseline $b_{mu}$ estimate and a similarity measure. The baseline estimate has three components which include the average movie rating $\mu$. A user bias $b_u$ that shows how the users average rating compares to the global average and is defined as the weighted average of the users ratings minus the global average. A movie bias $b_m(g)$ that shows how a movies average rating compares to the global average corrected for gender $g$. It is computed as the difference between the average rating given to the movie by either males or female minus the global average. This latter results in the following baseline estimate:

$$b_{um} = \mu + b_u + b_m(g) \tag{7}$$

*2) Computing similarities:* To compute the similarity between two items $(i, j)$, a regularized version of the cosine similarity $s_{ij}$ was used as a metric. It provides a measure between [0,1] where a higher similarity indicates two movies are more similar and can easily be cached after computation. For movies, the actual value is computed by taking the inner product of the intersection of their ratings and subsequently dividing this by their respective L2-norm. The ratings of both movies where reduces by their average prior to computation providing a form of regularization. The latter reduces to the following formula:

$$s(i,j) = \frac{\sum_{k \in I \cap J} (r_{ki} - \mu_i)(r_{kj} - \mu_j)}{\sqrt{\sum_u (r_{ui} - \mu_i)^2 \sum_u (r_{uj} - \mu_j)^2}} \tag{8}$$

*3) Computing the final rating:* The combination of both formulas results in the final NN model were the rating $r_x i$ of movie $i$ by user $x$ is defined by a baseline for $r_x i$ added to the weighted average of the similarity measure and baseline deviation from the neighborhood $N$ of $i$. Which in effects models both local and global effect [**?**] and can be reduced to the following formula:

$$\hat{r}_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij}(r_x j - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}} \tag{9}$$

### *Latent Factors*

The latent factor (LF) model used is primarily based upon the work of the Netflix prize winners [?]. The model makes use of matrix decomposition to express the ratings as the inner product of two matrices $P$ and $Q$ which dimensions are partially defined by the number of factors $f$. Each user is presented by a vector $p_x \in \mathbb{R}^f$ and each movie by a vector $q_i \in \mathbb{R}^f$. The ratings of user $x$ for movie $i$ can be expressed as the inner product between them: $r_{xi} = p_x^{'} q_i$. The latter approach is very suitable to model interactions but makes it more difficult to model the behavior of individual users. To remedy this, a baseline estimate is added which contains the global average, user-bias, and movie-bias. This reduces to the following formula:

$$r_x y = \mu + b_x + b_y + p^{'} q \tag{10}$$

To train my model I have chosen to use stochastic gradient descent (SGD). If we were to simply start training the model without additional improvements we would experience severe overfitting. Therefore, a regularization term consisting of L2-norms is added where $\lambda_i$ controls the effect of the regularization term. The objective of the training process now becomes the minimization of the resulting objective function defined as:

$$min_{qp} \sum_{x,y} (r_x i - (\mu + b_x + b_i + p^{'} q)^2 \tag{11}$$

$$+ \lambda_q ||q||^2 + \lambda_p ||p||^2 + \lambda_x b_x^2 + \lambda_i b_i^2 \tag{12}$$

Using SGD, for individual rating, each parameter (note that this includes the user and movie biases) is updated using the gradient of the error that can be derived by taking the derivative of the objective function for the respective parameter. This results in the following update rules where $\alpha$ represent the learning rate:

$$b_x = b_x + \alpha(err - \lambda_x b_x) \tag{13}$$

$$b_i = b_i + \alpha(err - \lambda_i b_i) \tag{14}$$

$$p_x = p_x + \alpha(err * q_i - \lambda_p p_x) \tag{15}$$

$$q_i = q_i + \alpha(err * p_x - \lambda_q q_x) \tag{16}$$

*1) Choosing the learning rate:* It turns out that fine tuning the learning rate is quite difficult. I wanted to make the learning rate $\alpha$ adaptive to improve both convergence speed and accuracy. My initial idea was to take a constant and scale it by the inverse square root of the current iteration $t$ such that $\alpha = \alpha_0 * \frac{1}{\sqrt{t}}$. This turned out be less effective than a simple fixed learning. I then researched several methods including but not limited to: AdaGrad, AdaDelta, RMSprop and eventually found an interesting method to automatically adjust multiple learning rates so as to minimize the expected error at any one time [?] which I decided to implement. Unfortunately, my programming skills and mathematical foundation was not strong enough to achieve a working implementation. Eventually I, chose a learning rate of $0.005$ which was reduced to $0.001$ upon reaching a RMSE of $0.845$ on the validation set.

*2) Adding temporal biases:* Researched showed that the addition of temporal biases is quite effective [?]. Unfortunately to use the method described a rating date was required which was not present in the provided dataset. However, I did add a temporal bias using the movies release year which was available. To incorporate the latter into my model I first normalized all release years $y_m$ to a $[0, 1]$ range such that $y_i^{'} = (y_i min_y)/(max_y - min_y)$. A new bias vector $T$ consisting of $v$ Buckets was added. For each movies release year, the corresponding bias was selected using $T[y_i^{\prime} * v]$. The value of the biases for bucket $t$ was updated using SGD identical to the user and movie biases $t_i = t_i + \alpha(err - \lambda_t t_i)$. The resulting model after adding the temporal biases is defined as:

$$r_x i = \mu + b_x + b_i + T(y_i v) + p^{'} q \tag{17}$$

*3) Adding IMDB:* Lastly, I incorporated the IMDb / RT data from section VII-A into my LF model in a manner similar to a bias. Resulting in the final model:

$$r_x i = \mu + b_x + b_i + b_{imdb} + b_{rt} + T(y_i v) + p^{'} q \tag{18}$$

The $b_{imdb}$ and $b_{rt}$ are both defined as the rating that movie has received minus the average on each site respectively. Note, that both biases are not training using SGD.

*4) IMDb and Rotten Tomatoes:* Two additional basic predictors were created using the data retrieved from IMDb and Rotten Tomatoes as discussed was section VII-A. The predictors estimate the $r_{xm}$ as the rating average rating of for movie $m$ on each site respectively. Individually these ratings are not of much value. However, they do add value when combined with other models using ensemble methods.

*5) Ensemble strategy:* To an ensemble improve accuracy, I combined the prediction of all previously discussed models. The latent factor model was used multiple times with a different number of factors. The combiner that was used predicted the final rating as the inner product of the set of predictors $P$ and weights $W$ were $w_i$ determines the weight of predictor $p_i$. Resulting in the following estimate: $r_{xi} = P^{'} W = p_1 w_1 + ... + p_n w_n$.

Due to an increasing number of predictors choosing weight soon became tedious. Therefore an evolutionary strategy (ES) [?] was used to train the combiner. It works by randomly mutating the weights using a Gaussian distribution, computing the fitness of the new population and replacing it with the old population if the fitness has improved. Fitness is defined as $1/RMSE$ such that higher is better. Each run converged to approximately the same fitness however they might do so using a very different combination of weights. I used these as a guideline to tweak my final weights. Though ES isnt perfect it does provide useful insights e.g. suggesting a combination where some weights are negative which is not something one might think of right away.

The second combiner I used was based on votes corrected for confidence. My idea was that if a prediction would output a rating $r_i$ of e.g. 3.56 it was not very confident. The predictor's vote $v_i$ was defined as the rounded rating. The confidence predictor $i$ was then defined as $c_i = (0.5 - |v_i r_i|)^2$ such that a rating close to their respective vote would receive a large confidence. The vote which accumulated the most confidence became the final prediction. However, unfortunately the combiner did not perform better than the ES trained combiner.

*6) Cross Validation:* Cross-validation is essential when evaluating the performance of the model. If we were to report the RMSE of the training set it might be that our model performs very well on paper but does not actually generalize when used with different data. To gain reasonably accurate estimate I used a train-test split. Each run a fraction of the data e.g. 0.9 was selected for training while the remaining data was used for testing. Before each run, the data was shuffled such that a different permutation was used each time.

## VIII. Results

### A. *Basic Predictors*

TABLE I
RMSE of basic predictors

| **Predictor** | **RMSE** |
|---|---|
| IMDB | 1.1049 |
| Rotten Tomatoes | 1.2568 |
| Movie Mean | 0.9432 |

The predictors in table I show the results of the data that was retrieved from IMDB, Rotten Tomatoes and Movie Mean predictor with is just the average of all the ratings a movie has received. These predictors are as discussed earlier used in the LF model and in ensemble learning.

### B. *Collaborative Filtering using Near-Neighbor Search*

TABLE II
RMSE of CF-NN model on Kaggle

| **Method / Neighborhood** | **10** | **25** | **50** |
|---|---|---|---|
| User-User no biases | 0.98866 | 0.94705 | 0.94966 |
| No biases | 0.89882 | 0.88304 | 0.89821 |
| Movie, user biases | 0.84341 | 0.84136 | 0.85475 |
| Movie, user gender biases | 0.84487 | 0.84090 | 0.84712 |

Table II shows the RMSE of my NN CF model on Kaggle. I choose to show the Kaggle results since they are most consistent and show the best performance. Increasing the size of the local validation test increases consistency but also significantly reduces performance due to a reduction in data available for training. Hence, I chose to show only Kaggle RMSE to optimize both. A grid search was used to test all neighborhood size between $[1, 50]$. The result was rather anticlimactic since the sense that the default N=25 or N=26 achieved the best results. My first implementation used user-user collaborative filtering with no additional biases or other improvements and achieved an RSME of roughly $0.94$. The relatively poor performance caused me to explore item-item collaborative filtering with no additional biases which performed much better reducing the RMSE with approximately $0.5$ to $0.89821$ on Kaggle. Due to the gain in performance, I abandoned user-user CF and focus fully on item-item CF. After adding user and movie biases as discussed in [section] the RMSE was further reduced to $0.84136$ (N=25). Finally adding gender biases slightly improved performance result is final RMSE of $0.84090$ for $N = 25$.

TABLE III
RMSE of LF model on a local validation set

| **Method / Factors** | **50** | **250** | **750** | **2000** |
|---|---|---|---|---|
| *Basic* | 0.9421 | 0.9974 | 0.9491 | 0.9713 |
| *Regularization* | 0.8821 | 0.8968 | 0.8927 | 0.8910 |
| *User / Movie Bias* | 0.8323 | 0.8394 | 0.8393 | 0.8389 |
| *User / Movie / Temporal Bias* | 0.8402 | 0.8389 | 0.8388 | 0.8392 |
| *User / Movie / Temporal / IMDB / RT* | 0.8392 | 0.8358 | 0.8356 | 0.8352 |

### C. *Latent Factors*

Table III shows the RMSE of my LF model on a local validation set (note that all models expect the basic version include regularization though not explicitly in [table] due to space limitations). For each result, iteration was terminated once the RMSE on the tests test no longer decreased for five consecutive iterations or iteration 250 was reached. Without biases and further optimization, the model very quickly converges on the training data and performs poorly. By adding regularization as described in section VII-E overfitting on the training is and training yields more consistent results. After adding biases (regularized) biases as per [equation] a significant gain in performance of approximately $0.04$ over all factors.

The training data movie release years span approximately 80 years. After a grid-search, I decided upon a 36-bucket temporal bias vector which gives in a bucket size of approximately $80/36 = 2.2$ years. Adding the bias showed a slight gain in performance in the all LF model expect for the LF-50 model which experience a RMSE increase of nearly $0.01$.

The full LF model which includes IMDb and RT data in according with equation 11 yields the best performance. There seems to be little variance between two 250-750-2000 factor variants. Each of them achieving an RMSE of approximately $0.8360$.

Increasing the number of factors has a positive effect on each model except for the unregularized basic LF model. Though not included in table 2, I have tested the model using 4000 factors which was as high as my computers RAM would allow. The latter did not yield a significant improvement ($0.00004$) over the 2000 factor model.

### D. *Ensemble*

The combiner that yielded the most optimal result is essentially a weighted sum of predictors optimized by an evolutionary strategy and some final manual tweaking as describe in section VII-E5. The ES trained combined was very useful in finding creative combinations of weights which

TABLE IV
RMSE OF DIFFERENT SETS OF PREDICTOR COMBINATIONS

| **Model** | **RMSE** |
|---|---|
| NN-25 / LF-50 | 0.83250 |
| NN-25 / LF-50-250 | 0.83044 |
| NN-25 / LF-50-250-750 | 0.82835 |
| NN-25 / LF-50-250-350-750 | 0.83007 |
| NN-25 / LF-50-250-350-750-2000 | 0.83372 |
| NN-25 / LF-50-250-750-2000 | 0.82796 |

could sometimes be manually be improved. To choose the right set of models several selections were tested. Table IV shows the performance of several ensembles a local validation set (note that the performance is computed using the weighted average of 5 runs per configurations). Where LF-[f] stands for a LF model with f factors and NN-[n] stands for a NN model with a neighborhood size of $n$. Which is also the combination that yielded my best Kaggle score (0.82714)

The second combiner mentioned in section VII-E5 did not perform very well. The best score being 0.83982 when used with just a NN-25 and LF-2000. When more predictors were added its performance quickly decreased. I suspect that this was due to similar nature of the predictors.

## IX. Discussion

In this section I will discuss a number of things I learned, did wrong, failed at and plan to improve.

The results showed that user-user CF performed poorly compared to item-item CF. However, I believe that a large component of the performance is related to the fact that the model has only been trained on a fraction of the original Netflix data. Given the current amount of data, there are many users for which there are few or even no users that are similar which leads to bad performance. If the size of the dataset would increase user-user CF might be more useful and perhaps contribute positively to ensemble learning.

The temporal biases I have implemented are not very useful compared to the work of [**?**]. In the future, I would like to supplement the rating data with timestamp such that temporal biases can be utilized properly.

LF models are powerful but tweaking them in such a way as to avoid overfitting turned out to be quite difficult. Therefore I wish to further explore methods to make learning rates adaptive which I believe will be useful in many project to come.

I suspect that k-fold cross-validation might have improved the results of my latent factor model. Though k-fold cannot be seen as complex on its own I organized my code in such a way that the Main class which loaded all the data did so with a help of a number of static attributes and did not encapsulate the process very well. Rewriting the application in such a manner that it could easily run on the different result while maintaining intermediate result was not possible due to time constraint.

I found ensemble learning the biggest surprise during the project. I did not expect that simply taking the weighted sum of two predictors would yield such significant gains in performance. In the future, I would like research more complex ensemble techniques which I believe can significantly improve performance.

I do feel that the ensemble section is rather short, certainly given the time I spent on ensemble methods. Even though I attempted a fair number of methods not described in this report (strict page count) I was unsuccessful (or not as successful as the ES combiner) at implementing them. Most introductory literature deals with binary classification and has predictors with well-defined confidence. For any future projects, I wish to do more into methods such as Adaboost.

Overall I enjoyed, the project even though it was frustrating sometimes. Since I really wanted to reach and RMSE below 0.82, which unfortunately I did not achieve. I think I might have spent too much time on it which will probably increase my RMSE on the finals. Lastly, next time I will definitely read more papers before starting to code.

I found writing this report useful, generally tend to be every bad at writing and it always cost me a lot of time. I did find the strict page count of four 4 bit annoying. As you can see I have filled up all space available and cheated a bit with the multicolumn layout. However, to make it fit on four page I still had to remove my graphs and a number of sections and replace them with more compact tables to make sure everything fits on four pages.