Reinforcement learning approaches to movies recommendation

Antoine Carpentier[1], Pierre Gérard[2] and Julian Schembri[2]

[1]Vrije Universiteit Brussel, Brussels
[2]Université libre de Bruxelles, Brussels
antoine.carpentier@vub.ac.be, pierre.gerard@ulb.ac.be, julian.schembri@ulb.ac.be

Abstract

The purpose of our research is to study reinforcement learning approaches to building a movie recommender system. We formulate the problem of interactive recommendation as a contextual multi-armed bandit, learning user preferences recommending new movies and receiving their ratings. We show that using reinforcement learning solves the problem of exploitation-exploration trade-off and the cold-start problem. We integrate the novelty of movies to the model. We explore a content based approach as well as a collaborative filtering approach and both yield viable recommendation results.

Introduction

Individuals sometimes have to settle on decisions without sufficient personal experience of the several choices. On a regular daily basis, their decisions depend on recommendations, advices and opinions from other individuals either by listening in conversations or by direct suggestion of movies, TV shows, holiday destinations, etc. Recommender systems help and enlarge this normal social process.

In a recommender system, individuals usually give behaviour information as data sources, which the system then use to learn users preferences in order to provide recommendation. The recommender's value lies in its capacity to make great matches between the recommenders and those looking for suggestions. Such systems are the strength of content providers such as Netflix Gomez-Uribe and Hunt (2015) and YouTube [1], allowing them to make their content more attractive in order to increase views and to optimize the monetisation of their videos. It allows to infer the customer's center of interest and, if customers share some general behaviour, to detect improvements in content that can be made to better meet the expectations of the public.

This work follows the steps of Wang et al. (2014) who have implemented a music recommendation system using reinforcement learning. In Wang et al. (2014), they explain what could be done to adapt their work to movies recommendation and suggest to try a collaborative filtering approach. In our work, we first adapt their strategies to movies recommendation and then we try on some approaches they suggest.

The problem of movie recommendation is modeled as a contextual multi-armed bandit [Sutton and Barto (1998)] [Lu et al. (2010)] which gives us the following advantages.

- Learning incorporates user feedback (i.e. ratings of movies) into recommendations thus fitting diversity in users preferences, as opposed to *context-free* multi-armed bandit,
- The model can take into account both a content based approaches and a collaborative filtering approaches,
- The model can use any type of features,
- Using a good policy balances exploration and exploitation thus mitigating the problems of cold-start and novelty.

The multi-armed bandit is a intensively studied reinforcement learning dilemma. It consists of a bandit (usually a casino slot machine) with K arms, each giving a payoff r sampled from an unknown probability distribution p_i. The player has a defined n number of pulls and her goal is to choose wisely which arm to play in order to maximize the total payoff.

The contextual multi-armed bandit is a generalization of the multi-armed bandit where the reward obtained from each arm depends also on a *context*. The context contains information about the arm and the player.

In the following sections, we will first introduce the methods used. Then we will review the results we obtained before discussing them.

Methods

Data mining

The data used is originated from IMDb [2]. It is two-folds. The first part is a movie catalog containing relevant information about each movies such as the genre, keywords describing the content, the year, the mean rating and number

[1]http://www.youtube.com

[2]http://www.imdb.com/

of votes. That part is publicly available from IMDb FTP in a homemade format. The second part of data is RSS flux of movies watched and rated by IMDb users containing the title of the movie along with the rating and the time it was watched.

Features extraction and selection

The first challenge was to extract the features from the IMDb homemade format and convert it into a clearer *csv* format.

In order to reduce the complexity of the algorithm, to focus on the learning matter and to avoid transforming the problem into a big data and parallelization problem, a subset of movies and users was picked, counting up to about 430.000 user-movie pair. We picked the 1000 movies with the most votes and all users who have at least 50 ratings for those movies. This create a bias in the data, that we will need to evaluate in a future work based on full real-life data.

To keep the features vector simple, we selected only the genre and year for each movie. Those two features are categorical variable and in order to learn a single weight per feature (i.e. drama, crime, ..), a one-hot encoding has been applied resulting in a blow up of the feature space to n features, n being the number of category.

Name	Drama	Crime	Thriller	...
Movie 1	0	0	1	...
Movie 2	1	1	1	...
Movie 3	0	0	0	...
...	...	...	...	...

Table 1: Movies feature space

Dimension reduction

The feature space resulting from the feature extraction process is quite large, making the recommendation computational intensive. In order to deal with that problem, the feature space size has been reduced using Principal Components Analysis (PCA) [Jolliffe (2002)].

The principal components analysis is a machine learning procedure that apply an orthogonal transformation to the dataset in order to obtain a new set of values that minimizes the covariance. Those new variables are called principal components. There are ordered in term of variance, the first principal component having the highest variance and so on. When the procedure is applied a reduction to n-dimension consists of keeping the n-first principal components thus removing first the components with lesser variance.

For our dataset we kept the 20-first principal components out of about 50, keeping about 83% of the variance of the original dataset.

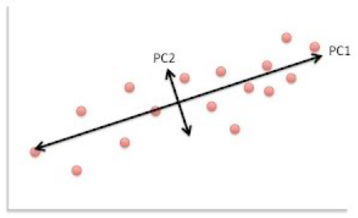

Figure 1: Example of a 2 dimension PCA. A reduction to one dimension would keep only PC1.

Contextual multi-armed bandit modelling

The movie recommendation problem can be seen as a contextual multi-armed bandit problem where each arm is a movie and a user can pull it (be recommended to watch it). The context for each arm (movie) is a features vector containing the user preferences and the movie features. The reward (rating) obtained from pulling the arm (watching the movie) depends on this context. When the rating is returned, the user preferences are updated depending on the movie features and the rating.

This raises one concern : as the number of movies is gigantic, some movies might not be watched. To ensure that movies are not repeated too often, a *novelty* parameter must be used.

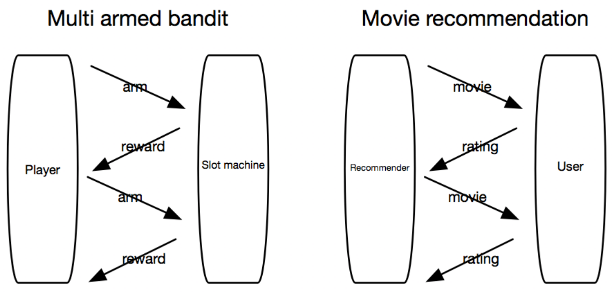

Figure 2: Seeing a movie recommender as a contextual multi-armed bandit problem

Content based The content based model tries to find a movie that is close to the learned user preferences $\boldsymbol{\theta}$. It takes into account the movie genre, movie year and novelty. Novelty can be defined as if the movie as been recently watched or not.

The movie genre and movie year can be considered as a one-hot encoded vector $\boldsymbol{x}$. Without considering other factor, the utility of a user for a movie can be represented as the following.

$$U_c = \boldsymbol{\theta}^T \boldsymbol{x}$$

Distinctive users have different preferences $\boldsymbol{\theta}$. Also, we do the hypothesis of a stationary true value for $\boldsymbol{\theta}$, meaning that while, we learn the user preference, its preferences stay the same.

The novelty is defined using the forgetting curve proposed by Ebbinghaus (1913) and defined as $e^{\frac{t}{s}}$ with t being the current time and s an hyper-parameter defining how fast we forget and want a repetition. Obviously if we want to recommend movies instead of music, an higher value of s is needed. The utility for a user in term of novelty can be defined as follows.

$$U_n = 1 - e^{\frac{t}{s}}$$

The combined utility (i.e. the expected rating for a movie and a user) is defined as follows.

$$U = U_c \times U_n = (\boldsymbol{\theta}^T \boldsymbol{x}) \times (1 - e^{\frac{t}{s}})$$

The task of the recommender is to propose a movie to the user depending on the expected ratings computed for each movie and to update the user preferences $\boldsymbol{\theta}$ according to the rating received from the user. We use an iterative mean.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + r_t \times \boldsymbol{x}_t$$

where $\boldsymbol{\theta}_t$ is the user preferences at epoch t, r_t is the rating received for the movie recommended at epoch t and $\boldsymbol{x}_t$ is the movie features for the movie recommended at epoch t.

Collaborative filtering The collaborative filtering model tries to find a movie that a user with similar preferences gave a good rating to. It takes into account other users rating and the novelty of movies.

The algorithm will start by finding a subset of the user and the k other users with the most similar preferences. The distance between users is defined as the Pearson product-moment correlation coefficient on preferences $\boldsymbol{\theta}$ of the targeted user and every other users. We obtain a list of all movies watched by those other users. We consider for each movie only the best rating. Those ratings become the expected ratings for each movie for the targeted user. To avoid any bias due to users over- or under-rating movies, other users ratings are weighted by the mean of all their ratings.

$$U_f = u$$

The novelty if defined similarly as above, so the expected rating is :

$$U = U_f \times U_n = u \times (1 - e^{\frac{t}{s}})$$

Bandit strategies

Many strategies to find the optimal solution have been developed over the years. A review of them is explain by Kuleshov and Precup (2014).

Random The random strategy is not a real strategy per se. It is a strategy where the player will only do exploration. It is mostly used as a basis to compare the effectiveness in term of reward or regret of an other algorithm.

Greedy-epsilon The $\epsilon - greedy$ algorithm is widely used because it is fairly simple. For each round of the simulation, the algorithm selects the arm (movie) with the highest rating with a probability of $1 - \epsilon$ and selects a random arm with a probability ϵ.

Greedy time dependent The greedy time dependent is a variation of the previously mentioned greedy where ϵ variate over time in order to maximize the exploration at first and reduce it in order to increase exploitation over time.

$$\epsilon = \frac{1}{\sqrt{t+1}}$$

However, the effectiveness of variating ϵ over time is disputed by Vermorel and Mohri (2005) who did not find any practical advantage to using these method. As a result, we will test both approaches.

Greedy UCB The problem with the two previous greedy algorithms is that they estimate the mean for each arm but not the standard deviation. The greedy algorithm with upper confidence bounds (UCB) solves that problem by computing for each arm a UCB such that the probability that the rating stands in the interval $[\hat{r} - U, \hat{r} + U]$ is high.
We define the UCB using Hoeffding's inequality such that $p = t^{-4}$.

$$P[r > \hat{r} + U] \leq e^{-2NU^2}$$
$$t^{-4} = e^{-2NU^2}$$
$$U = \sqrt{\frac{2 \log t}{N}}$$

where N is the number of times that this arm has been pulled and t the total number of pulls.

Exact Bayesian methods With the exact Bayesian method, we want to predict the user's rating of a movie through the inference on the user's set of parameters $\Omega = \{\boldsymbol{\theta}, s\}$. For the i-th recommendation, we assume that we have $l = i - 1$ history recommendations $\mathcal{D}_l = \{(x_j, t_j, r_j)\}_{j=1}^{l}$. Where x_j is the j-th recommendation's features vector, t_j is the time elapsed since the preceding recommendation of this movie and r_j is the user's feedback of the j-th recommendation.

We can predict the expected rating of movie k as:

$$\lambda_k^l = p(U_k|\mathcal{D}_l) = \int p(U_k|\Omega)p(\Omega|\mathcal{D}_l)d\Omega$$

Where $p(U_k|\mathcal{D}_l)$ is the posterior distribution of U_k and $p(\Omega|\mathcal{D}_l)$ is the posterior distribution of Ω. We recommend song $k^\star = arg \max_{k=1\ldots|S|} Q(\alpha, \lambda_k^l)$ where Q is the quantile function that satisfies $\mathbb{P}[U_k \leq Q(\alpha, \lambda_k^l)] = \alpha$ and S is all the movies. Hence, we balance exploration and exploitation.

The Bayesian model is developed with a composition of random variables as shown in Wang et al. (2014). Because there is no closed-form solution to λ_k^l, Markov Chain Monte Carlo (MCMC) can be used as an approximation procedure.

Bayes inference To get around the bad performance of the exact Bayesian model, we will use an approximated Bayesian model. The complexity of computing the the exact Bayesian model is due to the irregular form of the function $U_n(t)$. To do so, we discretize the time t into a K intervals: $[0, \xi), [\xi_1, \xi_2), ...[\xi_{K-1}, +\infty)$ and then map t into a vector $\boldsymbol{t} = [(t - \xi_1)_+, ...(t - \xi_{K-1}), t, 1]$. We approximate $U_n(t) \approx \boldsymbol{\beta}^T \boldsymbol{t}$ where $\boldsymbol{\beta}$ is a vector of parameters that will be learned. We can rewrite $U = U_c U_n \approx \boldsymbol{\theta}^T \boldsymbol{x} \boldsymbol{\beta}^T \boldsymbol{t}$. From the history of an user, we try to learn the expected values of $\boldsymbol{\beta}$, $\boldsymbol{\theta}$ and $\tau = 1/\sigma^2$ that represent user's data. Then with these data, we can generate samples and approximate with MCMC the expected rating of each movie candidate for recommendation.

Simulation

We conducted an empirical efficiency study of the algorithms mentioned above as well as the content based and collaborative filtering modelling.

Experiments were conducted on a personal computer and the programming language $python$ was used to implement models and algorithms.

The simulation consists in the interaction between a recommender and a user. To simulate online learning, ratings from users are known in advance and retrieved at each epoch. We ran each algorithm for several hundred epochs and averaged the results over more than 100 users.

Results can be found in section .

Metrics

In order to evaluate our algorithms and models in term of movie recommendation performance, an assortment of metrics retained. They could be formalized as follows.

Cumulative regret For the l-th recommendation, the regret is defined as the difference between the maximum expected rating $E[R^l] = max_{k=1...|S|} U_k$ and the expected rating $E[\hat{R}^l]$ of the recommended song is $\Delta_l = E[\hat{R}^l] - E[R^l]$. The cumulative regret for the n-th recommendation is then $\sum_{l=1}^{n} \Delta_l = E[\hat{R}^l] - E[R^l]$

Accuracy (RMSE) The root-mean-square error is a frequently used measure of the difference between the predicted value and the actually observed value. In our case, the root-mean-square error will be the square of the difference between the expected value for the rating and the actual rating given by the user.

Average rating The average rating is the simplest metric consisting of only the observed rating. It is interesting because it is a good indicator of how the user liked the recommendation.

Computation time The last metric is the time needed to compute a recommendation. Indeed, usually a recommendation should be computed in a reasonable time, the system can't afford to make the user wait for several minutes before giving him a recommendation.

Results

This section will explore the different results obtained throughout all experiments made.

Adaptation of Wang et al. (2014) to movie recommendation

Here are the results of the adaptation of Wang et al. (2014) to movie recommendation using a greedy algorithm and the same content based model as described in that paper. Let's look at metrics defined in .

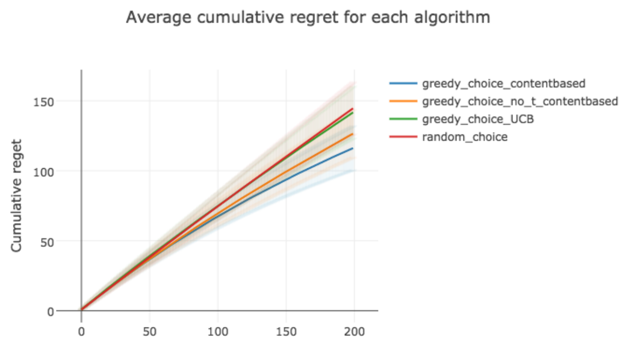

Figure 3: Average cumulative regret for greedy algorithm on the content based approach

This graph tells us that on the long term or mid term, the recommender will give us better result than a random selection of movie. Also it underlines the importance of balancing exploration and exploitation. Indeed, the result difference between different greedy approach is quite significant. We can see that greedy UCB does not seem to work well.

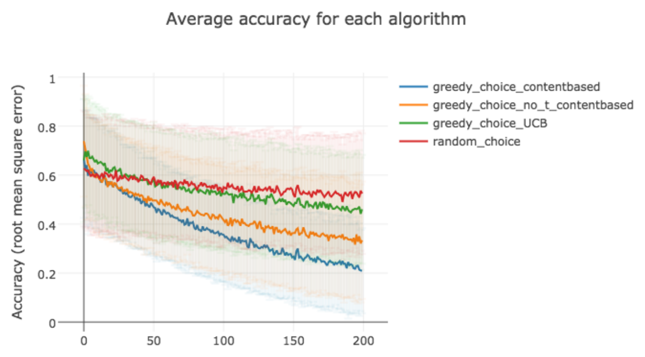

Figure 4: Average mean square error for each algorithm on the content based approach

The graph above tends to show that it takes about 15 recommendations before starting to actually exploit the user preferences for two greedy except greedy-UCB which seems to learn more slowly. After 15 recommendations, those two greedy algorithm error are getting smaller meaning that they tends to predict the rating the user will give to the algorithm better. The explanation to why greedy will get a smaller error than random is because greedy will exploit the result and recommend movie with features similar to user preferences thus predicting the utility with more accuracy. The explanation to why random get more accurate in term or prediction of rating over time without doing exploitation is because at first the user feature are initialized to zero and then are updated to match what the user like.

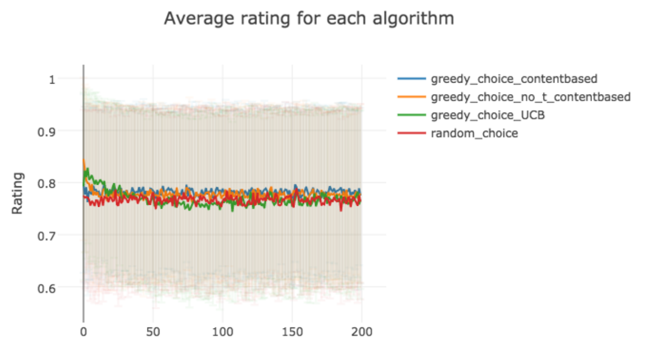

Figure 5: Average rating given by the user as a feedback to a recommendation

On the graph above the feedback given by the user for two greedy different from the greedy UCB seems to dominate ”random”, hence it tends to indicate that two to algorithms recommend movies that the users likes more.

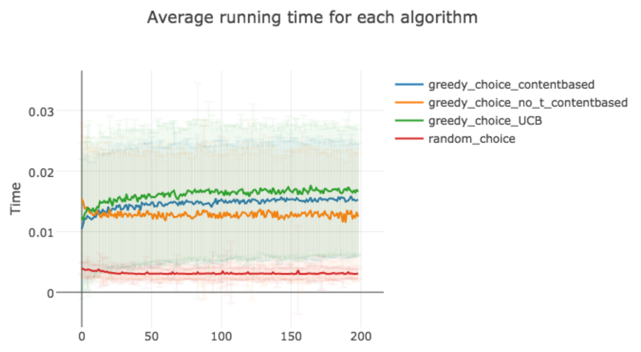

Figure 6: Average time in second needed to produce a recommendation to the user

On the graph above, we can see that the slowest version of greedy takes less than 0.02 seconds to produce a recommendation which is a score than would be suitable for a web interface where recommendation would be computed on request.

Bayesian methods

Exact Bayesian Due to the Markov Chain Monte Carlo approximation method and the high number of random sampling it need to compute, the exact Bayesian method take a long time as show below.

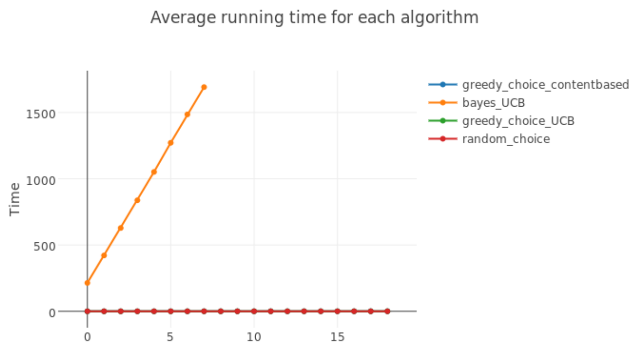

Figure 7: Cumulative sum of the time needed to produce the recommendation

On the graph above, we can see that the exact Bayesian take about 4 minutes to produce a single recommendation. Therefore this model does not seems to be suitable for a movie recommender system if not adapted to widely parallel way to compute each utility for a recommendation.

Bayesian inference The Bayesian inference method shown in (Wang et al., 2014) involve complex computations. Some parts of the algorithm are partially incomplete and left for interpretation. Furthermore the hyperparameters presented in the paper are used for music recommendation and hence might not be suitable for movie recommendation. Attempts of implementation of this method have not provided expected results shown in the paper. The debugging of code containing partial interpretation is a highly complicated task, on which we worked extensively without success.

Collaborative filtering and content based comparison

Following, Wang et al. (2014) suggestion, we then looked at the collaborative filtering model to movie recommendation and compared it to content based. Here are the results.

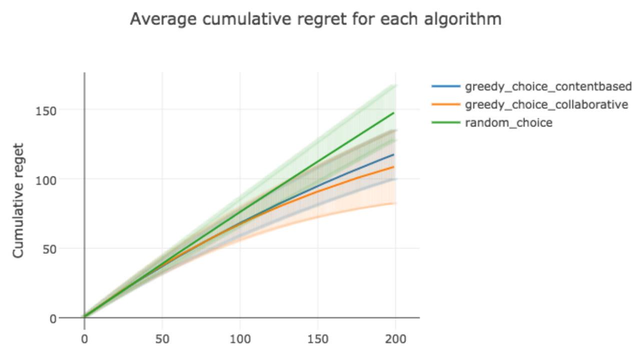

Figure 8: Comparison of the average cumulative regret for greedy algorithm on the content based model and collaborative filtering model

The graph above tends to show that the collaborative filtering tends to return better longer term result at least for a greedy algorithm. Indeed the cumulative regret is lower for collaborative filtering than it is for a content based model.

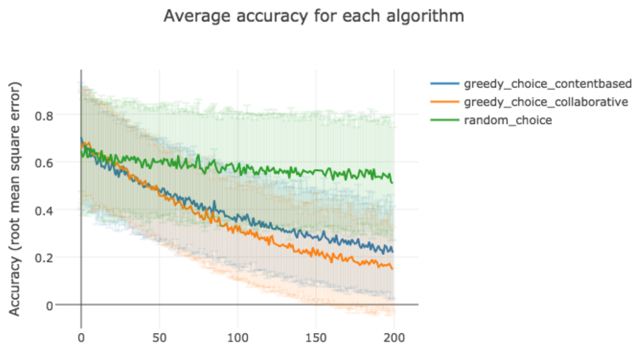

Figure 9: Comparison of the mean square error for the content based model and collaborative filtering model

The graph above seems to show that the collaborative filtering model is slightly more accurate than the content based model having a inferior mean square error.

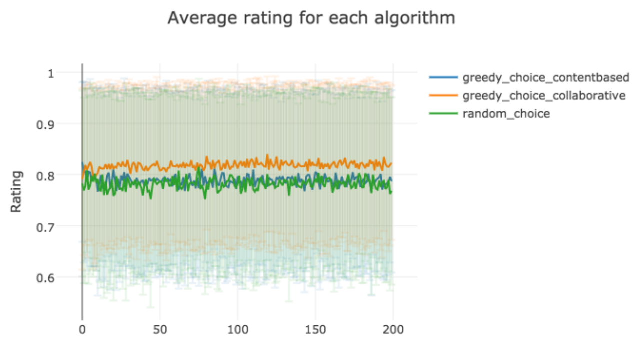

Figure 10: Comparison of the average rating given as feedback by the user for each algorithm

Maybe the more important graph for the comparison is the one above. It tends to show that there is a significant different on the feedback given by the user depending on the model used for the greedy algorithm. User tends to give a higher feedback to movie suggested by the collaborative filtering approach.

Overall performance

To conclude the results section, let's look at the overall performance of all models and algorithms

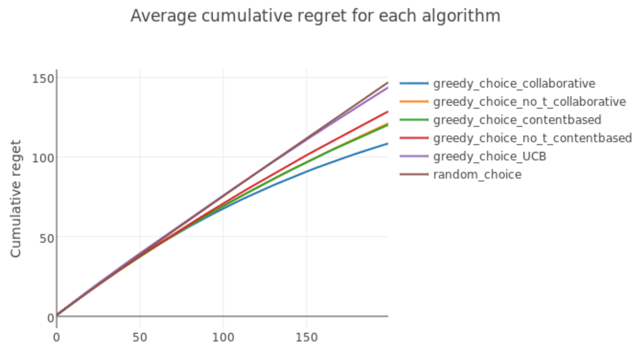

Figure 11: Comparison of all model and algorithm

Obviously, the random random choice is the worst.

The best techniques seems to take a greedy that explore a lot at the beginning and reduce the exploration in profit of exploitation over time while using a collaborative filtering model.

Discussion

Exploring user preferences is a central piece of a personalized recommender system and can be used as a context in a multi-armed bandit modelization of the recommender as mentioned by Wang et al. (2014).

As shown in the result the Bayesian way is not viable as it is. Indeed, the exact Bayesian takes way too much time to compute a single recommendation and attempts of implementation of the Bayesian inference have not yield the expected results. However, for the exact model, one could argues that the computation could be parallelised on multi-core or on a big data cluster.

The comparison of the different viable algorithms have shown the following :

- $\epsilon - greedy$ give on of the poorest result,
- a greedy that variate the exploration and is content based give satisfactory results,
- a greedy that variate the exploration and use collaborative filtering give the best results of all.

The reason why the greedy-UCB algorithm does not seem to work well is that the UCB is difficult to gauge and to fit in our contextual model. Bayes-UCB should have solve this problem by representing the rating as a random variable but as said above, is not viable either.

We ask the following question : does collaborative filtering models work better than content based models because we recommend movies or are the results generalisable to any kind of recommendation ? Moreover, one must keep in mind that a collaborative filtering approach tends to recommend to user popular item thus discarding less known item, meaning that for movies, it will suggest blockbusters over a small budget productions.

For future direction, we suggest to research an hybrid approaches between collaborative filtering and content based. We would also suggest to compare the result of the reinforcement learning with a more traditional supervised learning approaches. We would also recommend a more scalable approaches on a big data cluster in order to include more movies features, more movies and more user.

Conclusion

We built a thorough functional movie recommender system including data mining, features engineering, dimension reduction and several models and content based as well as a collaborative filtering approaches to reinforcement learning.

Following the step of Wang et al. (2014) who worked on music recommendation, we used the advantages of reinforcement learning to build such a system taking advantages of the exploitation-exploration trade-off to solve the usual recommender cold-start problem.

We also successfully integrated novelty in our model to handle and avoid too much movie repetition. Moreover, we followed Wang et al. (2014) advice to explore collaborative filtering and demonstrated that it outcomes better results than the content based approaches. Finally we defined future possible research.

References

Ebbinghaus, H. (1913). Memory: A contribution to experimental psychology, translated by henry a. *Ruger & Clara E. Bussenius. New York: Teachers College, Columbia University.*

Gomez-Uribe, C. A. and Hunt, N. (2015). The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4):13:1–13:19.

Jolliffe, I. (2002). *Principal component analysis*. Wiley Online Library.

Kuleshov, V. and Precup, D. (2014). Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*.

Lu, T., Pál, D., and Pál, M. (2010). Contextual multi-armed bandits. In *AISTATS*, pages 485–492.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Vermorel, J. and Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer.

Wang, X., Wang, Y., Hsu, D., and Wang, Y. (2014). Exploration in interactive personalized music recommendation: a reinforcement learning approach. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(1):7.