Modeling Ordinal Data For Recommendation System

by

Anupam Srivastava

B.Tech., Indian Institute of Information Technology, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES

(Computer Science)

The University Of British Columbia
(Vancouver)

September 2014

Abstract

In this work we investigate the problem of making personalized recommendations by creating models for predicting user-item rating, such as in movie recommendations. The study is based on the Movielens data set [16] which has ratings on an ordinal scale. In the past, partly due to motivation gained by the Netflix challenge, researchers have constructed models that make point predictions to minimize the root mean square error (RMSE) on test sets, typically by learning latent user and movie feature structure. In such models, the difference between ratings of 2 and 3 stars is the same as the difference between ratings of 4 and 5 stars, etc., which is a strong prior assumption. We construct probabilistic models which also learn latent user and movie feature structure but do not make this assumption. These models interpret the ratings as categories (nominal and ordinal) and return a probability distribution over the ratings for each user-movie pair instead of making a point prediction. We evaluate and compare our models with other models for making personalized recommendations for the top-n task and comparing the precision vs recall, receiver operating characteristic and cost curves. Our results show that our ordinal data model performs better than a nominal data model, a state-of-the-art point prediction model, and other baselines.

Preface

This dissertation is original and independent work by the author, A. Srivastava under the supervision of Professor David Poole. The work has been submitted as a research paper for publication in AAAI-2015 under the title "Modeling Ordinal Data for Recommendation System" with A. Srivastava as the main author and David Poole as the co-author.

Ethics approval was not required for this work as the data sets used were publicly available[16].

Table of Contents

List of Tables

List of Figures

Acknowledgments

First of all, I thank my research supervisor Professor David Poole for his continuous support and guidance. This work would not have been possible without his invaluable supervision and encouragement.

I thank Professor Mark Schmidt for being the second examiner for my thesis. I am grateful to all my fellow researchers at STAR-AI Lab.

Finally, I am extremely grateful to my parents and my brother for their continuous support and encouragement.

Dedication

I dedicate my work to my parents for their endless love and support.

Chapter 1

Introduction

In recent years, the amount of information on the internet has exploded and so has the choices available to a consumer. Modern e-commerce websites offer a wide array of products to its users, for example amazon.com offers over 250 million products[1]. This information overload is overwhelming for the user and makes it difficult for them to make decisions. The same problem is faced in case of music and movie streaming websites. In such an environment, it is beneficial for both the user and website for the website to come up with a personalized list of item recommendations which would be consumed by the user with a high probability. This makes the need for a recommendation engine in these websites imperative.

In the literature, three types of recommendation systems have been studied extensively. Content based recommendation systems which recommend items based on user specific features like age, occupation, sex, etc., and item specific features like price, age appropriateness, etc., [4, 8, 25]. Collaborative filtering based recommendation systems recommend item by studying and identifying patterns in user consumption history and finding similar user and items by using some similarity notion which differs from algorithm to algorithm [18, 24, 29]. Often researchers bring these two types of recommendation systems together into the third type called hybrid recommendation systems where both usage history and user and item features are used to make recommendations [3, 7].

[1] The number was obtained by asking amazon to return product list which do not match to a nonsense query

User ID	Movie ID	Rating	Time stamp
192	1021	2	14211329
192	1109	5	14312461
184	2013	4	14319122
184	1021	4	14321232
203	2207	5	14322923

Table 1.1: Netflix data set format. Tuple 1 means that user with ID 192 gave a rating of 2 to the movie with ID 1021 at t = 14211329.

In October 2006, Netflix made public a 100 million rating data set [6]. The data set contained tuples like those shown in Table 1.1. Each entry represented the rating given by a specific user to a movie, both recognized by unique IDs. The challenge posted by Netflix was to make predictions on the held out user and movie ID pairs and improve the root mean squared error (RMSE) measure by 10 percent over their existing recommendation system. The challenge brought a surge in research in this field and many submissions and subsequent publications were made. The winner of this challenge was an ensemble method bringing together many different solutions [20]. The latent factor based model proposed in [5] was part of this ensemble method and performed very well on the RMSE measure [20]. We will use this model as a baseline and it will be referred to as the BellKor solution.

Since the challenge was to improve on the root mean square error measure, the proposed recommendation models treated the ratings as numbers [5, 34]. But we don't always have the luxury of numerical ratings. If you look at most e-commerce, music and movie streaming websites, user preferences towards items is perceived through rather implicit signals like searching for an item, bookmarking an item, using an item, etc. Its not easy to assign a number to these signals. However, they can be assigned categories. Moreover we can observe that we can often order these categories, for example, *bookmarking an item* is a stronger signal of user affection towards a product compared to just *searching for an item*.

There is no intuitive way of using recommendation systems which interpret ratings as numbers for such data sets. In addition, even though treating the movie ratings (given that RMSE is to be optimized) as numbers is intuitive, it is misin-

terpretation of the data. Users don't assign numbers to movies, rather they assign categories represented by numbers. Treating them as numbers means we are treating the difference between 1 and 2 stars, the same as the difference between 4 and 5 stars. This is a strong prior assumption and hinders recommendation qualities as shown by our experimental results.

In this work, we propose two probabilistic recommendation models which are based on collaborative filtering. The models don't treat the movie ratings as numbers but as categories. In one model, we treat the categories to be nominal whereas in the second model we treat the categories to be ordinal. Both models are based on a latent factor model similar to the BellKor solution [5]. However, they differ from the BellKor model in more ways than just interpretation of input. In the proposed ordinal model we learn an additional set of parameters per user which allows us to learn user rating pattern. The proposed model output a probability distribution over the set of possible ratings instead of just one number as done in BellKor model. We evaluate these based on the top-n task [19] where the aim is to produce n recommendations and evaluate which ones of these the user likes.

We discuss collaborative filtering, latent factor models, and other related work in the next chapter. In Chapter 3, we describe our two proposed probabilistic recommendation models. We test our model against the BellKor solution on data sets of varying sizes and probability distribution. We use the MovieLens data set [put reference] for our evaluation, since the Netflix data set is no longer available. We give the details about our experiments and results in Chapter 4. Finally we conclude in chapter 5 with some discussion and some pointers to future work.

Chapter 2

Background

2.1 Collaborative filtering

The biggest problem faced by recommendation systems is data sparsity. An average user rates only a small percentage of the entire set of items available. This makes the prediction task difficult because there are very few positive samples and a very large number of negative samples. In collaborative filtering algorithms, this problem is solved by trying to collect and understand preferences from multiple users and items to make recommendations for one user or item. One method is to form clusters of *similar* users and items. Then predictions are made for a user by looking at the usage history of a *similar* user or by looking at the set of items *similar* to the item already used by the user.

As an example, look at Table 2.1 which displays the consumption history of five users. Looking at the table, we can tell User 3 and User 5 have similar taste in items as they both have consumed Item 1 and Item 2 and rejected item 3. Now, if we were to make prediction for User 3, we can look at the consumption history of User 5 and recommend Item 4 to User 3. Similarly, Item 2 can be recommended to User 1. However, our data is not always a simple "consumed - rejected - not seen" data set. We are going to train and test our models on movie[1] rating data sets. The movie ratings are ordered over 5 categories which makes the problem of making recommendation more complex. Moreover, in this example we have used

[1] From now on, we are going to use movie and items interchangeably

	Item 1	Item 2	Item 3	Item 4
User 1	Yes	?	No	Yes
User 2	Yes	?	?	?
User 3	Yes	Yes	No	?
User 4	?	?	Yes	?
User 5	Yes	Yes	No	Yes

Table 2.1: User consumption history."Yes" represents that an item has been consumed by the user, "No" represents that an item was rejected by the user whereas "?" represents the item has not been seen by the user

a very simple and stripped down notion of similarity. As we see in the literature, researchers have proposed more complex and more effective ways of interpreting similarity between users and items.

2.2 Latent factor model

In the latent factor model, we assume that there are a set of unobserved characteristic traits which explain why a user gave a particular rating to a movie. These unobserved characteristic traits may belong to user or item. For example: a user trait may be their affinity towards action movies and a movie trait might be the degree to which it is an action movie. Learning these latent factors is similar to clustering users and items. Each latent factor can represent a cluster and the value of the latent factor for a particular user/item can represent the degree to which a user/item is associated with a cluster. This way a user/item can be associated with multiple cluster with different degrees of association and then the ratings given out by the user can be explained in terms of these associations.

In Figure 2.1, we can see a toy example of a post hoc interpretation of a 2-dimensional latent factor model for items. The X-axis captures whether the movie is comic or tragic whereas the Y-axis captures if the movie is romantic or thrilling. For example, "Armageddon" is a thriller with a comic touch and hence is placed in the second quadrant. Now a user depending on his own set of latent factors would be attracted to certain portions of this state-space. For example, a user may lie in the top right quadrant based on its own latent factors as shown in Figure 2.1. Now

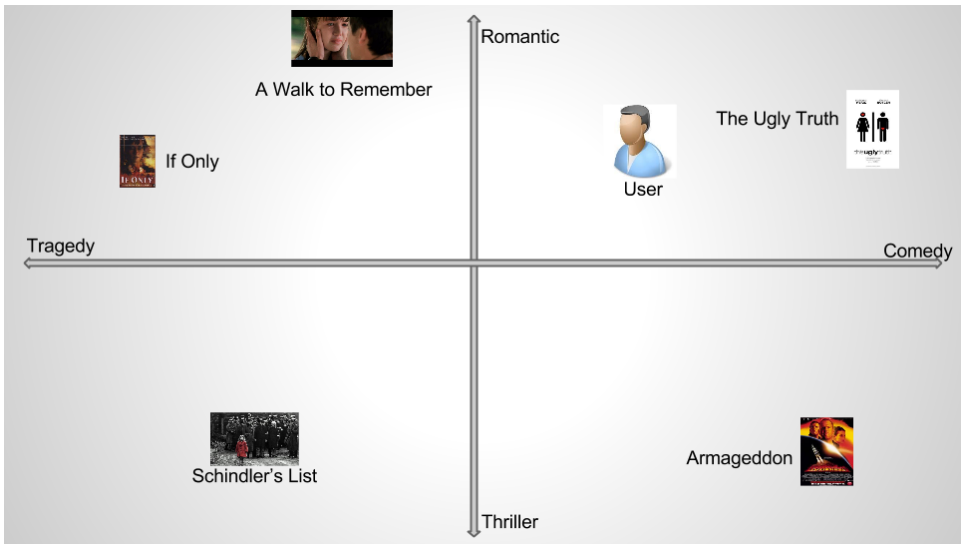


Figure 2.1: Latent factor model

the user would prefer romantic comedies like "The Ugly Truth". The idea behind the latent factor based recommendation system is that the user would give high ratings to movie lying in corresponding portions of the state space and low ratings to movie lying far away from there.

2.2.1 BellKor model

The Bellkor model is based on a latent factor model [5]. The model assumes that each user and each movie has a set of unobserved latent factors. The interaction between these latent factors linearly combined with appropriate offsets explains the rating given by the user to the movie.

$$\hat{r}_{u,i} = \mu_{avg} + b_u + b_i + \sum_f p_{u,f} q_{f,i} \tag{2.1}$$

where $\hat{r}_{u,i}$ is the rating which user u would give to item i according to the model, μ_{avg} is the average rating over the data set, b_u is a user bias, b_i is a item bias, f is the latent factor, $p_{u,f}$ is the value of latent factor for user u and $q_{f,i}$ is the value of item latent factor for item i.

Using the definition in Equation 2.1, the cost function can be defined as:

$$C(\theta) = \sum_{\langle u,i,r \rangle \in D} \left(\hat{r}_{u,i} - r_{u,i}\right)^2 + \lambda \|\theta\|_2^2 \tag{2.2}$$

where D is the data set consisting of tuples <u, i, r>, $\theta = \left(b_u, b_i, p_{u,f}, q_{i,f}\right)$, $r_{u,i}$ is the rating given by user u to item i and $\hat{r}_{u,i}$ is the model prediction. λ is the regularization constant and the second term is introduced to avoid over-fitting. The l_2 regularizer makes our model more numerically stable [12, 27].

The model learns the user and item bias along with the latent factors by minimizing Equation 2.2:

$$\hat{\theta} = argmin \sum_{\langle u,i,r \rangle \in D} \left(\hat{r}_{u,i} - r_{u,i}\right)^2 + \lambda \|\theta\|_2^2 \tag{2.3}$$

The minimization can be done using stochastic gradient. The algorithm can been seen in Algorithm 1. The model parameters are initialized to random values sampled from uniform random distribution. The termination criteria depends on the improvement gained in the cost function. In our experiments, if the improvement drop below .01%, then the algorithm is stopped. The derivatives of the cost function with respect to model parameters are:

$$\begin{aligned}
\frac{\partial C(\theta)}{\partial b_u} &= 2\left(\hat{r} - r\right) + \lambda 2 b_u \\
\frac{\partial C(\theta)}{\partial b_i} &= 2\left(\hat{r} - r\right) + \lambda 2 b_i \\
\frac{\partial C(\theta)}{\partial p_{u,f}} &= 2\left(\hat{r} - r\right) q_{i,f} + \lambda 2 p_{u,f} \\
\frac{\partial C(\theta)}{\partial q_{i,f}} &= 2\left(\hat{r} - r\right) p_{u,f} + \lambda 2 q_{i,f}
\end{aligned}$$

2.2.2 Ordec model

In [21], the authors extended the BellKor model to create a probabilistic recommendation model called Ordec which returns a probability distribution over ratings for each user-item pair. In addition to the parameters defined in (2.1), the model in-

Algorithm 1 BellKor latent factor model

```
INPUT: data set of ⟨UserID, MovieID, Rating⟩
OUTPUT: b_u, b_i, p_{u,f}, q_{i,f}
Initialize model parameters
while Termination criteria not met do
    for each data point: user u, item i, rating r do
        r̂ = μ_avg + b_u + b_i + Σ_f p_{u,f} q_{f,i}          ▷ Make prediction
        θ^new = θ − step_size ∂C(θ)/∂θ                        ▷ Update model parameters
        θ = θ^new
    end for
end while
```

troduced S - 1 ordered thresholds, associated with each of the rating values besides the last one

$$t_1 \leq t_2 \cdots \leq t_{S-1} \tag{2.4}$$

where t_1 is an actual model parameter but other thresholds are defined using another set of parameters $\beta_1, \beta_2, \cdots, \beta_{S-2}$:

$$t_{r+1} = t_r + exp(\beta_r) \tag{2.5}$$

where $r \in [1, S-2]$. This ensures that the difference between consecutive thresholds is non-negative. Given these model parameters, the authors defined:

$$P(r_{u,i} \leq r|\theta) = \frac{1}{1 + exp(\hat{r}_{u,i} - t_{u,r})} \tag{2.6}$$

where θ represents the biases and latent factors defined in (2.1) along with the thresholds explained in (2.5). The learning was done by using stochastic gradient ascent on the log likelihood function.

2.3 Evaluation of recommendation systems

Motivated by the Netflix challenge, recommendation systems are being evaluated on the basis of how well they do on the root mean square measure (RMSE).

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} \left(\hat{r} - r\right)^2}{n}}$$

where $\hat{r}$ is the prediction and r is the actual rating. However, over the years the focus of recommendation systems have shifted from prediction accuracy to ranking of items. The recommendation system is expected to return a list of items ranked according to the probability of them being consumed by the user. Ranking of items give a more accurate representation of how the recommendation system is actually used in the real world [22].

In [9], authors showed that methods which do well on the RMSE measure might actually not do that well on predicting highly ranked items. This is because while optimizing RMSE, the model gets equal incentive for doing well on predicting low and high ratings. However, a system designed to return a list of items which would be ranked highly by the user can achieve its goal just by doing well on the higher ranked items even though it might be very bad in predicting or distinguishing between lower ranked items.

In our study, we evaluate the system on the basis of how well they do in predicting items which would be consumed by the user with a high probability. We didn't carry out an actual user study, so we don't have a good way of measuring or evaluating this phenomenon. To overcome this, we use a surrogate; we ask the recommendation models to present users with K items and say an item is *liked* if it was given a rating of 5 by the user.

2.3.1 Precision at n vs recall at n

Recommendation systems are often compared against each other using the precision at n vs recall at n metric [32]. Precision at n is the precision of the model when it was used to generate a recommendation list of size n. Recall at n is the recall of the model when it was used to generate a recommendation list of size n. They are calculated as follows:

$$\text{Precision at } n = \frac{\#TP(n)}{\#TP(n) + \#FP(n)} \tag{2.7}$$

$$\text{Recall at } n = \frac{\#TP(n)}{\#TP(n) + \#FN(n)} \tag{2.8}$$

where n is the recommendation list size, $\#TP(n)$ is the number of true positives (the number of movies the user rated 5 out of the n movies presented), $\#FP(n)$ is

		Ground truth	
		true	false
Prediction	true	true positive	false positive
	false	false negative	true negative

Table 2.2: Confusion matrix. This helps in understanding the definition of true_positive, false_positive, false_negative and true_negative. For example, a prediction is called false_negative if it was predicted to be false by the model but was actually true.

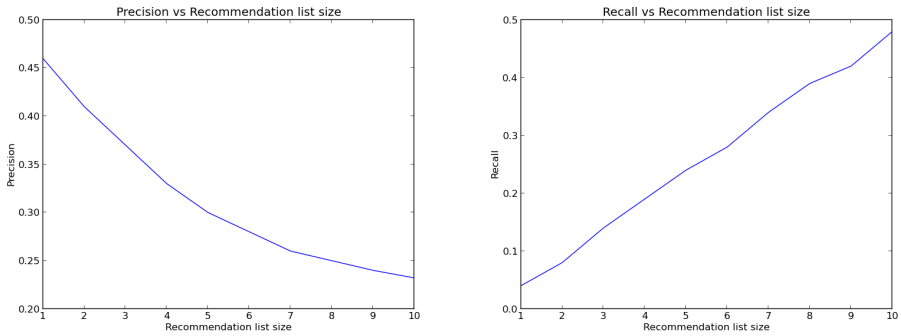
the number of false positives ($\#TP(n) + \#FP(n) = n$) , $\#FN(n)$ is the number of false negatives ($\#TP(n) + \#FN(n) =$ the number of items the user rated as 5) when the recommendation list size is n. True positives, false positives and false negatives can be understood from Table 2.2

We can observe that as we increase the value of n, recall at n would increase as the denominator in (2.8) is a constant and the $\#TP(n)$ would monotonically increase. On the other hand, precision at n would show a tendency to decrease. The recommendation list of increasing lengths are made by adding the successive elements from the ranked list. So the probability of the new item being liked by the user is less than any of the previously recommended items. This implies that with an increase in n, the new element would have a greater tendency to become a false positive rather than true positive. Therefore, $\#FP(n)$ would increase relatively quickly as compared to $\#TP(n)$ as we increase n. Hence, the precision at n would show a tendency to decrease.

Figure 2.2a shows a toy example of how the precision of a model varies as we change the recommendation list size. Figure 2.2b shows a toy example of the variation in recall of the model as we change the recommendation list size. The information is aggregated together by plotting the precision at n vs recall at n curve Figure 2.3.

2.3.2 Receiver operating characteristic curve

We also compare the models based on the Receiver Operating Characteristic or ROC curves [15]. ROC curve is a plot between the true positive and false positive



(a) Precision vs recommendation list size **(b)** Recall vs recommendation list size

Figure 2.2: Precision and recall as we change the size of the recommendation list.

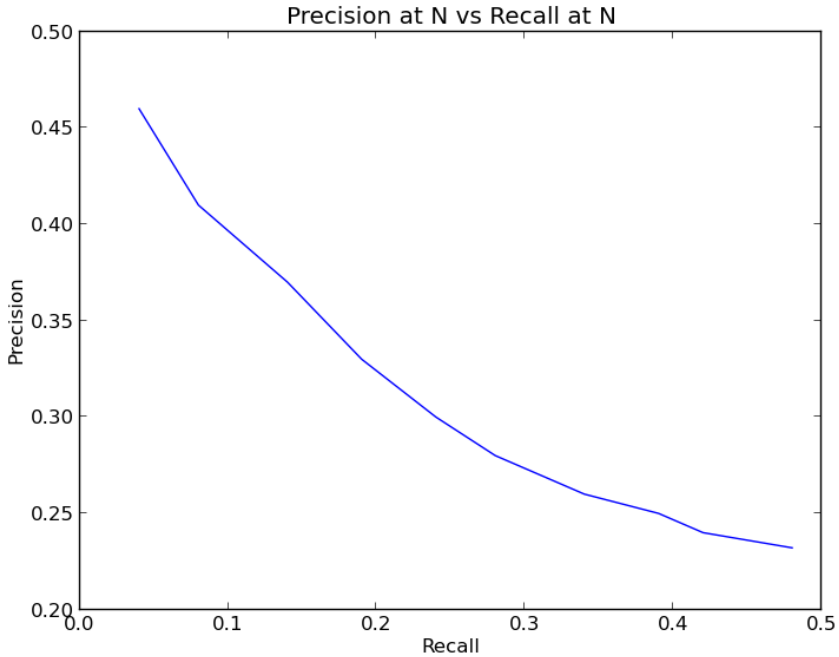


Figure 2.3: Precision at n vs recall at n

rate:

$$\text{True positive rate} = \frac{\#TP(n)}{\#TP(n) + \#FN(n)} \tag{2.9}$$

$$\text{False positive rate} = \frac{\#FP(n)}{\#FP(n) + \#TN(n)} \tag{2.10}$$

Different values for true positive and false positive rate are calculated from recommendation list of varying size and then plotted. Both precision at n vs recall at n and ROC curve gives us a representation of the proportion of items that would be consumed by the user from the recommendation list. However, precision at n vs recall at n curve emphasize more on the recommended items which are consumed by the user (precision) whereas ROC curve gives more emphasis to recommended items which are not consumed by the user (false positive) [32]. Due to this difference, we should chose between precision at n vs recall at n and ROC curve to do our analysis based on the context. For example, if we want our models to have a high precision, we should compare them on the basis of precision at n vs recall at n curve. On the other hand, if we want our models to have low false positive rate, we should compare them on the basis of ROC curves. In this work, we have plotted both curves and analyzed them independently.

2.3.3 Cost curves

Precision at n vs recall at n curve and ROC curves allow us to compare between two classifiers while we vary some model parameter to see how they work under different conditions. In this work, we have varied the recommendation list size to get different performance measures. However, these curves don't account for the probability distribution of the positive (or negative) samples in our test population. This is problematic especially when our test and train population probability distribution don't represent the real world accurately. In our experiments, there is no good way of knowing that the probability distribution of the real world is the same as the test set. Hence, we would like to have a tool which allows us to compare between classifiers at varying probability of positive samples.

Cost curve helps us in comparing between classifiers at varying probabilities of positive samples [10]. Cost curve plots the error rate at varying probability of

positive samples

$$E[Cost] = FN * p(+) * C(-|+) + FP * p(-) * C(+|-) \tag{2.11}$$

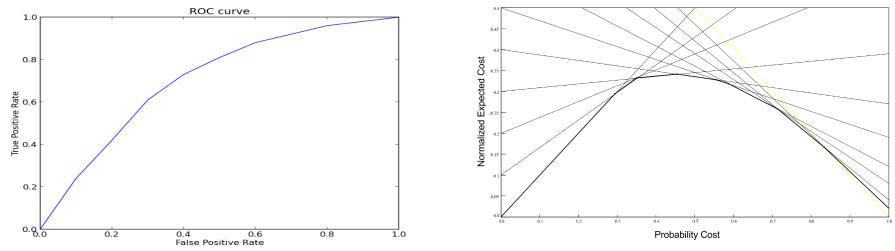
where FN is the false negative rate, p(+) represents the probability of positive sample, C(- |+) is cost of missclassifying positive sample as negative, FP is the false positive rate, p(-) represents the probability of negative sample and C(+|-) is cost of miss-classifying a negative sample as positive. Note that p(-) = 1 - p(+). We further assume that C(- |+) = C(+|-). Making these simplifications, we get:

$$\begin{aligned} Norm(E[Cost]) &= \frac{FN * p(+) * C(-|+) + FP * p(-) * C(+|-)}{p(+) * C(-|+) + p(-) * C(+|-)} \\ &= \frac{C(-|+)\Big(FN * p(+) + FP * p(-)\Big)}{C(-|+)\Big(p(-) + p(+)\Big)} \\ &= FN * p(+) + FP * \big(1 - p(+)\big) \\ &= (FN - FP) * p(+) + FP \end{aligned} \tag{2.12}$$

Note here that FN = 1-TP where TP is true positive rate. ROC curve is a curve between TP and FP. So, for every point in ROC curve, we get a line in cost curve by varying p(+) as seen in equation (2.12). As seen in Figure 2.4, for each point on the ROC curve, we plot a line in cost curve using the equation (2.12). We can see that these sets of intersecting lines form a lower envelope. This lower envelope represents the set of optimal operating points for a given classifier as the probability of positive samples is varied from 0 to 1. In our experiments, we plot cost curves for our three competing models and make inferences based on their lower envelopes.

2.3.4 Fraction of concordant pairs

The fraction of concordant pairs (FCPs) is another evaluation measure used to compare different classifiers. Precision at n vs recall at n, ROC and cost curve are basically meant to compare binary classifiers. We can extend them to recommendation system by asking a binary question like *Does the user like the movie recommended*



(a) ROC curve **(b)** Cost curve

Figure 2.4: ROC curve and corresponding set of lines in cost curve and the lower envelope.

by the recommendation model? Liking a movie is synonymous to giving a high rating to a movie. So a classifier which can predict movies which would receive high ratings, say 5, can do very well on these curves. However websites would often want to order movies over the entire rating scale. This is important for pricing the movies. So we would like to evaluate how our recommendation system does on the entire rating structure i.e., can it distinguish between a rating of 2 and 3 stars as well as it is able to distinguish between 2 and 5 stars. FCP allows us to do this. To define FCP, let's first define concordant pairs. Given a test set D, we define the number of concordant pairs for user u by counting those pair of items that are ranked correctly by the rating predictor:

$$n_c^u = |\{(i,j) : \hat{r}_{u,i} > \hat{r}_{u,j} \text{ and } r_{u,i} > r_{u,j}\}| \tag{2.13}$$

Similarly we can define discordant pairs for user u as the number of pair of items which are ranked incorrectly by the rating predictor:

$$n_d^u = |\{(i,j)\hat{r}_{u,i} > \hat{r}_{u,j} \text{ and } r_{u,i} < r_{u,j}\}| \tag{2.14}$$

Given the definition of n_c^u and n_d^u, we can define FCP as:

$$FCP = \frac{n_c}{n_c + n_d} \tag{2.15}$$

where $\mathrm{n}_c = \sum_u \mathrm{n}_c^u$ and $\mathrm{n}_d = \sum_u \mathrm{n}_d^u$. From the defintion, we can observe that in order

to do well on the FCP measure, the ranking predictor should be equally good at distinguishing *low quality* movies and *high quality* movies.

2.4 Constrained optimization

Constrained optimization is a large and well studied field of research. We are going to look at a type of constrained optimization problem which we encountered while modelling our recommendation system,

$$\text{Minimize f(x)},$$
$$\text{subject to } f_i(x) \leq 0$$

The problem is to minimize a function f(x) without violating the constraints $f_i(x) \leq 0 \; \forall i \in$ [1..m]. To solve the above minimization problem, a new function called a barrier function is introduced [17]:

$$\phi(x) = \begin{cases} \sum_{i=1}^{m} -log(-f_i(x)), & f_i \leq 0, i = 1, ..., m \\ \infty, & \text{otherwise} \end{cases} \tag{2.16}$$

You can imagine the barrier function $\phi(x)$ representing a wall separating the valid space of parameters from the invalid. The function attains a high positive value as $f_i \to 0$ (the wall) and a small value as we keep going away from zero. Therefore, this function motivates the parameters to not violate the constraints.

The barrier function is linearly combined with the original function which was to be minimized to create a new optimization problem but without any constraints this time,

$$h(x;t) = f(x) + \frac{1}{t}\phi(x) \tag{2.17}$$

The new optimization problem has a hyper-parameter t which is to be optimized. We can note here that higher the value of t, lower is the contribution of the $\phi(x)$ to h(x; t) and as t $\to \infty$, h(x; t) $\approx$ f(x).

In the literature, h(x;t) is minimized by minimizing a series of different h(x; $\hat{t}$) problems where $\hat{t}$ is pre-fixed [17, 33]. We start with an initial value for $\hat{t}$ and calculate x^* which minimizes h(x; $\hat{t}$). Then the value for $\hat{t}$ is increased and h(x; $\hat{t}$)

is again minimized. However, this time the minimization starts from x^* calculated in the previous minimization step and so on. As $t \to \infty$, x^* which minimizes $h(x;\hat{t})$ also minimizes $f(x)$.

Chapter 3

Proposed Models

We argued that most recommendation systems were treating movie ratings as numbers which was a misinterpretation of the data set. Movie ratings represents categories which have an ordered structure. Moreover, movie ratings are very subjective, in the sense that a movie which gets assigned a rating of 3 by a user might get a rating of 4 by another user even though they *equally enjoyed* the movie. Therefore, its important to not only learn what the user may or may not like, but an equal emphasis must be given to the way he assigns ratings to the movie. The second part is often ignored by recommendation systems. This is explicitly incorporated in our second recommendation system.

We are now going to talk about the two models. The two models vary in the way they interpret the data. The first model treats the data as nominal and the second one treats it to be ordinal. Due to their different way of interpreting the data, they have different ways of assigning probability, and hence cost functions to be optimized.

3.1 Logistic regression model

In this model, we interpret the data to be nominal. We learn independent logistic regression model (LRM) for each category[1]. While learning the LRM for a partic-

[1] We could also learn a multinomial representation, but this did not give different performance in our tests. We present the logisitic regression model as our testing was for rating = 5 and we used the logisitic regression model for this category

Rating	Percentage distribution
1	0.05616%
2	0.10753%
3	0.34889%
4	0.26114%
5	0.22626%

Table 3.1: Percentage distribution of the ratings in the 1 million MovieLens data set.

ular category, say j, we treat all data points with rating = j as positive samples and rest as negative. Due to this simplification step, the training data set for all LRM is heavily biased with lot of negative samples and relatively few positive samples.

Table 3.1 shows the distribution of ratings in the 1 million rating Movielens data set. For example, the most skewed data set is observed for rating = 1, where to train the LRM we would have approximately 5% positive samples and 95% negative samples. On the other hand, the least skewed data set is observed for rating = 4, where to train the LRM we would have approximately 35% positive samples and 65% negative samples.

The LRM is based on the BellKor model. Like the BellKor solution we assume that the users and items have latent traits which interact with each other, and their interaction decide the rating which a movie gets from a particular user. Since we are learning 5 independent LRMs, one for each category, we assume that user and movies have 5 sets of latent factors, one for each category. Then we define:

$$response_j = b_{u,j} + b_{i,j} + \sum_f p_{u,j,f} q_{f,j,i} \tag{3.1}$$

where j$\in [1,5]$ is an index over the 5 independent LRM models,$b_{u,j}$, $b_{i,j}$, $p_{u,j}$ and $q_{i,j}$ are the model parameters for j^{th} LRM. $b_{u,j}$ represents the user bias for user u, $b_{i,j}$ represents the item bias for item i, f are the latent factors, $p_{u,j,f}$ are values of the unobserved latent factors for user u and $q_{f,i,j}$ are values of the unobserved latent factors for item i.

This response is converted to a probability value by passing it through a sig-

moid function. So, we get:

$$P\big(r_{u,i} = j\big) = \frac{1}{1 + e^{-response_j}} \tag{3.2}$$

where P$\big(r_{u,i} = j\big)$ is the probability of user u assigning a rating j to movie i and $response_j$ is as defined in (3.1) Given the above definition of probability of a movie getting a rating of j, we can define the likelihood of a data set D with respect to parameters θ:

$$L_D(\theta) = \prod_{\langle u,i,r\rangle \in D}^{n} \prod_{j=1}^{5} \big(P(r_{u,i} = j)\big)^{\mathbb{1}(r_{u,i}=j)} \big(1 - P(r_{u,i} = j)\big)^{\mathbb{1}(r_{u,i}\neq j)} \tag{3.3}$$

where D is the dataset, u is the user, i is the movie and r is the rating given to the movie i by user u. We try to learn model parameters $\theta = \Big(b_{u,j}, b_{i,j}, p_{u,j,f}, q_{i,j,f}\Big)$ which maximize the likelihood of the data under our model. This is equivalent to minimizing the negative of the log likelihood. We have added a l_2 regularizer as well:

$$NLL(\theta) = -\sum_{u,i,r\in D}^{n} \sum_{j=1}^{5} log\Bigg(\big(P(r_{u,i} = j)\big)^{\mathbb{1}(r_{u,i}=j)} \big(1 - P(r_{u,i} = j)\big)^{\mathbb{1}(r_{u,i}\neq j)}\Bigg) + \lambda \|\theta\|_2^2 \tag{3.4}$$

where λ is the regularization constant. We minimize NLL(θ) by using stochastic gradient as described in Algorithm 2. The model parameters are initialized to random values using uniform random distribution. We use a constant step size [31]. We terminate when the improvement in NLL(θ) between consecutive iterations becomes less than 0.01%. The derivatives with respect to model parameters are:

$$\frac{\partial NLL(\theta)}{\partial b_{u,j}} = -\sum_{\langle u,i,r\rangle \in D} \left(\left(\mathbb{1}\left(r_{u,i} = j\right) - P(r_{u,i} = j) \right) + 2\lambda b_{u,j} \right)$$

$$\frac{\partial NLL(\theta)}{\partial b_{i,j}} = -\sum_{\langle u,i,r\rangle \in D} \left(\left(\mathbb{1}\left(r_{u,i} = j\right) - P(r_{u,i} = j) \right) + 2\lambda b_{i,j} \right)$$

$$\frac{\partial NLL(\theta)}{\partial p_{u,j,f}} = -\sum_{\langle u,i,r\rangle \in D} \left(\left(\mathbb{1}\left(r_{u,i} = j\right) - P(r_{u,i} = j) \right) * q_{f,j,i} + 2\lambda p_{u,j,f} \right)$$

$$\frac{\partial NLL(\theta)}{\partial q_{f,j,i}} = -\sum_{\langle u,i,r\rangle \in D} \left(\left(\mathbb{1}\left(r_{u,i} = j\right) - P(r_{u,i} = j) \right) * p_{u,j,f} + 2\lambda q_{f,j,i} \right)$$

Algorithm 2 Logistic regression model

```
INPUT: data set of ⟨UserID, MovieID, Rating⟩
OUTPUT: b_{u,j}, b_{i,j}, p_{u,j,f}, q_{i,j,f}
Initialize model parameters
while Termination criteria not met do
    for each data point: user u, movie i, rating r do
        for each category j do
            response_j = b_{u,j} + b_{i,j} + Σ_f p_{u,j,f} * q_{f,j,i}
            P(r_{u,i} = j) = 1/(1+e^{-response_j})          ▷ Make prediction
            θ^{new} = θ − step_size * ∂NLL(θ)/∂θ            ▷ Update model parameters
            θ = θ^{new}
        end for
    end for
end while
```

Once the model parameters and learned using the algorithm 2, we can make predictions using (3.2). Using (3.2) the model can answer questions like; what is probability of a movie i getting a rating j from user u. We are particularly interested in knowing what is the probability of a movie i getting a rating of 5 from user u which we use later in our experiments as a surrogate for *liking a movie*.

3.2 Cumulative model

The cumulative model is a recommendation model where we treat the ratings as ordinal. In models where the ratings are treated as numbers, the difference between

the ratings is assumed to be linear e.g., the difference between a rating of 1 and 2 is the same as difference between rating of 4 and 5. This is a very restrictive assumption.

The cumulative model is inspired by the regression models for ordinal data proposed in [26]. In this model, we model the ordinal ratings by postulating the existence of an unobserved latent variable $Z_{u,i}$ and a set of user specific thresholds. $Z_{u,i}$ captures the affection of user u towards an item i whereas the thresholds are used to define mapping between regions on $Z_{u,i}$ and ratings. By learning thresholds and $Z_{u,i}$ we arrive at a very flexible model in comparison to the models which treat the ratings as numbers. These additional sets of thresholds allow us to learn user rating pattern which could be non-linear in structure i.e., now with this model we can learn rating patterns where the difference between 1 and 2 stars could be very small as compared to difference between 4 and 5 stars.

Similar to the previous models, we define a response that takes into account user and item biases and the latent traits:

$$h_{u,i} = b_u + b_i + \sum_f p_{u,f} q_{f,i} \tag{3.5}$$

where b_u is the user bias, b_i is the item bias, f is the set of latent factors, $p_{u,f}$ is the value of latent factor for user u, $q_{f,i}$ is the value of latent factor for item i.

We then assume a noisy observation model, $Z_{u,i}$:

$$Z_{u,i} = h_{u,i} + \varepsilon_{u,i}, \tag{3.6}$$

where $\varepsilon_{u,i}$ is a random variable to model the noise. The noise could be because of numerous independent random processes. We assume that the $\varepsilon_{u,i}$ are independent for each u, i pair, with each one normally distributed: $\varepsilon_{u,i} \sim \mathcal{N}(0,1)$. Then we can say that

$$Z_{u,i} \sim \mathcal{N}(h_{u,i}, 1) \tag{3.7}$$

We map this unobserved latent variable $Z_{u,i}$ to categories by introducing thresholds or cutoffs. For data set with 5 categories, we introduce 6 cutoffs: $\gamma_{u,r}$ where r $\in \{0, 5\}$. We assume $\gamma_{u,0}$ = - ∞ and $\gamma_{u,5}$ = ∞. We mentioned earlier that movie rating is a subjective process. A movie can receive different ratings from different

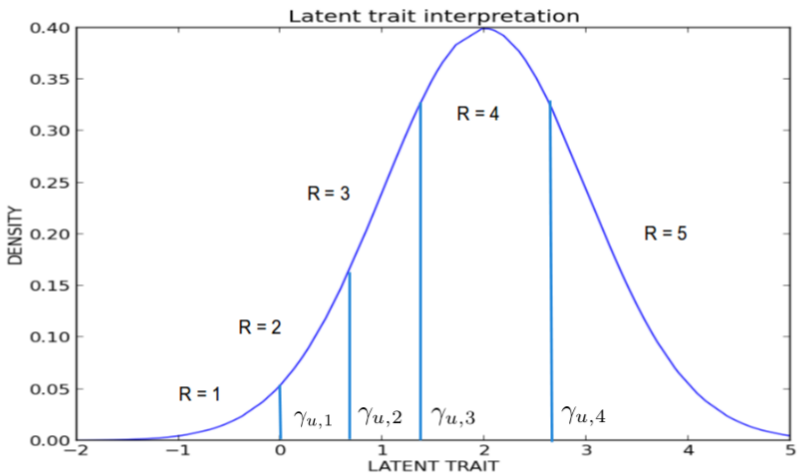


Figure 3.1: Plot of $Z_{u,i} \sim \mathcal{N}(h_{u,i}, 1)$ with cutoffs to map the latent trait to categories/ratings. If $Z_{u,i}$ falls between $\gamma_{u,r-1}$ and $\gamma_{u,r}$ it gets mapped to category r.

user even though they *equally enjoyed* (which is captured by $Z_{u,i}$) the movie. This motivates us to have user specific cutoffs.

Figure 3.1 shows $Z_{u,i}$ for a particular movie i and user u along with the user specific thresholds $\gamma_{u,r}$. $Z_{u,i}$ is distributed according to (3.7) and (3.5). The mapping using the thresholds is straightforward. If $Z_{u,i}$ lies between $\gamma_{u,r-1}$ and $\gamma_{u,r}$, we say that user u gave a rating of r to movie i.

Once we have $Z_{u,i}$ and user specific γ, its easy to come up with an expression for assigning the probability to the event of user u giving a rating r to item i:

$$\begin{aligned} P(r_{u,i} = r) &= P(\gamma_{u,r-1} < Z_{u,i} \leq \gamma_{u,r}) \\ &= P(Z_{u,i} \leq \gamma_{u,r}) - P(Z_{u,i} < \gamma_{u,r-1}) \\ &= F(\gamma_{u,r} - h_{u,i}) - F(\gamma_{u,r-1} - h_{u,i}) \end{aligned} \tag{3.8}$$

where $P(r_{u,i} = r)$ is the probability with which the user u gives a rating of r to movie i and F is the cumulative Gaussian probability distribution.

Given the definition of probability of the event of user u giving a rating r to

item i, we can define the likelihood of a data set D with respect to parameters θ:

$$L_D(\theta) = \prod_{\langle u,i,r\rangle \in D} F(\gamma_{u,r} - h_{u,i}) - F(\gamma_{u,r-1} - h_{u,i}) \tag{3.9}$$

We try to learn the model parameters $\theta = \left(\mathrm{b}_u, \mathrm{b}_i, \mathrm{p}_{u,f}, \mathrm{q}_{j,f}, \gamma_u\right)$ which maximize the likelihood of the data under our model. This is equivalent to minimizing the negative of the log likelihood. We add a l2 regularizer:

$$NLL(\theta) = -\sum_{\langle u,i,r\rangle \in D} log\Big(F(\gamma_{u,r} - h_{u,i}) - F(\gamma_{u,r-1} - h_{u,i})\Big) + \lambda \|\theta\|_2^2 \tag{3.10}$$

with constraints

$$\gamma_{u,r-1} \leq \gamma_{u,r} \forall r \in [0, 5]$$

where λ is the regularization constant and the second term is introduced to avoid over fitting. We minimize NLL(θ) by using stochastic gradient. The model parameters were initialized to random values using uniform random distribution. It is interesting to note here that NLL(θ) is not convex. In our empirical studies we realized that minimizing NLL(θ) in its current form was very difficult because of numerical instability. The constraints on γ were often violated, in the sense, we often encountered scenarios where $\gamma_{u,j} < \gamma_{u,j-1}$ for some user u and rating j. Due to this, the probability of a movie receiving a rating j from the user u becomes undefined. To tackle this problem, we made following adjustments in our solution:

1. We changed our cost function to explicitly include a term for the constraints. This was done using the barrier method discussed in Section 2.4. Then our new cost function is:

$$\begin{aligned} C(\theta;t) = &-\sum_{\langle u,i,r\rangle \in D} log\Big(F(\gamma_{u,r} - h_{u,i}) - F(\gamma_{u,r-1} - h_{u,i})\Big) \\ &+ \lambda \|\theta\|_2^2 + \frac{1}{t}\sum_{m=1}^{4} -log(\gamma_{u,m} - \gamma_{u,m-1}) \end{aligned} \tag{3.11}$$

2. Secondly, we made changes to the way γ_u were getting initialized. Rather than randomly initializing them, we trained our system assuming that all the

data points were coming from a single user. The final value for γ obtained in this pre-processing step was used as a seed. γ_u were initialized to γ and samples from zero mean Gaussian noise. This step can be interpreted as using some sort of global average to initialize the values of γ and then during the training step, we just learn deviations from the global average for each user. It can also be interpreted as imparting your prior knowledge about the distribution of γ.

Algorithm 3 Cumulative model

INPUT: data set of $\langle UserID, MovieID, Rating \rangle$
OUTPUT: $b_u, b_i, p_{u,f}, q_{i,f}, \gamma_u$
Initialize model parameters
while Termination criteria not met **do**
    for each data point: u, i, r **do**
        $P(r_{u,i} = r) = F\left(\gamma_{u,r} - h_{u,i}\right) - F\left(\gamma_{u,r-1} - h_{u,i}\right)$ ▷ Make prediction
        $\theta^{new} = \theta - step\_size \frac{\partial C(\theta)}{\partial \theta}$ ▷ Update model parameters
        $\theta = \theta^{new}$
    end for
end while

After making these two adjustments, the model parameters are obtained by minimizing C(θ;t) with respect to θ by using stochastic gradient. The algorithm for the same can be seen in Algorithm 3. The step size is set to a constant value [31]. We terminate when the improvement in C(θ) between consecutive iterations becomes less than 0.01%. The derivatives with respect to model parameters are:

$$\frac{\partial C(\theta)}{\partial b_u} = \sum_{\langle u,i,r\rangle \in D} \left(\frac{\big(f(\gamma_{u,r} - h_{u,r}) - f(\gamma_{u,r-1} - h_{u,r-1})\big)}{F\big(\gamma_{u,r} - h_{u,r}\big) - F\big(\gamma_{u,r-1} - h_{u,r-1}\big)} + 2\lambda b_{u,j} \right)$$

$$\frac{\partial C(\theta)}{\partial b_i} = \sum_{\langle u,i,r\rangle \in D} \left(\frac{\big(f(\gamma_{u,r} - h_{u,r}) - f(\gamma_{u,r-1} - h_{u,r-1})\big)}{F\big(\gamma_{u,r} - h_{u,r}\big) - F\big(\gamma_{u,r-1} - h_{u,r-1}\big)} + 2\lambda b_i \right)$$

$$\frac{\partial C(\theta)}{\partial p_{u,f}} = \sum_{\langle u,i,r\rangle \in D} \left(\frac{\big(f(\gamma_{u,r} - h_{u,r}) - f(\gamma_{u,r-1} - h_{u,r-1})\big) q_{u,f}}{F\big(\gamma_{u,r} - h_{u,r}\big) - F\big(\gamma_{u,r-1} - h_{u,r-1}\big)} + 2\lambda p_{u,f} \right)$$

$$\frac{\partial C(\theta)}{\partial q_{u,f}} = \sum_{\langle u,i,r\rangle \in D} \left(\frac{\big(f(\gamma_{u,r} - h_{u,r}) - f(\gamma_{u,r-1} - h_{u,r-1})\big) p_{u,f}}{F\big(\gamma_{u,r} - h_{u,r}\big) - F\big(\gamma_{u,r-1} - h_{u,r-1}\big)} + 2\lambda q_{u,f} \right)$$

$$\frac{\partial C(\theta)}{\partial \gamma_{u,r}} = \sum_{\langle u,i,r\rangle \in D} \left(\frac{\big(f(\gamma_{u,r} - h_{u,r}) - f(\gamma_{u,r-1} - h_{u,r-1})\big)}{F\big(\gamma_{u,r} - h_{u,r}\big) - F\big(\gamma_{u,r-1} - h_{u,r-1}\big)} - \frac{1}{t\big(\gamma_{u,r} - \gamma u, r-1\big)} + 2\lambda \gamma_{u,r} \right)$$

where F is the Gaussian cumulative distribution function and f is the Gaussian probability density function.

Once we have trained the system to learn all the model parameters, we can predict the probability with which user u will give a rating of r to a movie i using (3.8). We are in particular interested in knowing what is the probability of a movie i getting a rating of 5 from a user u which we use in our experiments as a surrogate for *liking a movie*.

Chapter 4

Experiments and Results

We evaluated the six models;

1. the logistic regression model presented in Section 3.1
2. the cumulative model presented in Section 3.2
3. BellKor solution as explained in Section 2.2.1
4. Ordec model as explained in Section 2.2.2
5. Most popular movies: In this case, the recommendation list was generated by sorting the movies on the basis of the number of 5 ratings they get in the training data set.
6. Random movies

The evaluation is done on two data sets; 1 million rating MovieLens data set and 10 million rating MovieLens data set. The 1 million rating data set has 5 categories whereas 10 million rating data set has 10 categories. For all experiments, the data set in question is divided into two parts chronologically: 80% training set and 20% test set.

4.1 Experiment 1

The first experiment was carried out using the 1 Million rating MovieLens data set collected as part of the Grouplens project[1] [16]. The data set has ratings from 4000 movies and over 6000 users. We want to evaluate if the recommendation models are able to generate personalized recommendations which the user would *like*. Since we didn't perform an actual user study, we used a surrogate for *liking a movie*. We say, a movie is *liked* by a user if he or she gives the movie a rating of 5. These movies form the positive samples. The data set was filtered to have only those users for which we had ≥ 20 positive samples. After the filtration step, we were left with 582 users.

The six models were evaluated by generating a list of recommendations for the users. In case of BellKor, the list was generated by sorting the movies on the basis of the rating they received by the user. For LRM, the list was generated by sorting the movies on the basis of the probability of them receiving a rating of 5. The same method was used for the cumulative and ordec model. We varied the size of the list of recommendation from 1 to 20 and calculated the precision and recall at each list size. The precision at n vs recall at n where n is the size of the list can be seen in Figure 4.1

As can be seen in Figure 4.1, the cumulative model is completely dominating the BellKor solution and LRM. The Ordec model and the most popular movie baseline has higher precision as compared to the cumulative model for small recommendation list sizes (the most popular couple of movies is easy to predict) but gets dominated in other region of the graph. The curve between precision and recommendation list size, and recall and recommendation list size, can be seen in Figure 4.2 and Figure 4.3 respectively. If we look at figure Figure 4.4 which is a plot between the average number of true positives per user vs the size of the recommendation list, we observe that all models achieve very low average with the highest around 1.4 attained by the cumulative model. This results in low precision and recall values as observed in Figure 4.2 and Figure 4.3.

The low precision and recall values can be explained. In our experiment design, we were making predictions over the set of all possible movies (4000). How-

[1] http://grouplens.org/datasets/movielens/

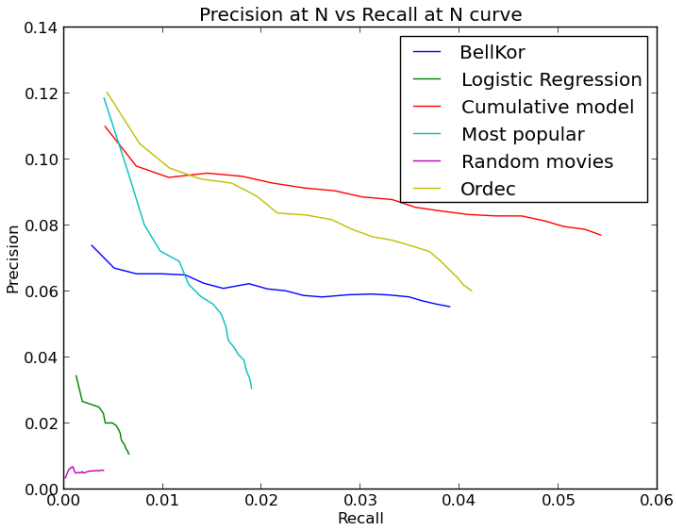


Figure 4.1: Experiment 1: Precision at n vs recall at n

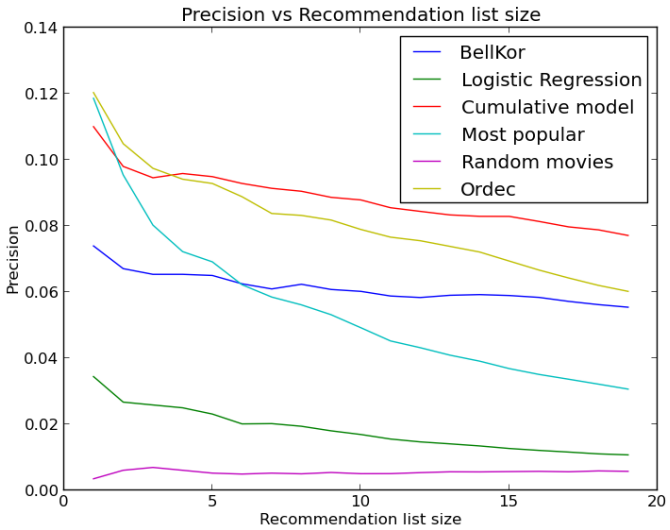


Figure 4.2: Experiment 1: Precision vs recommendation list size

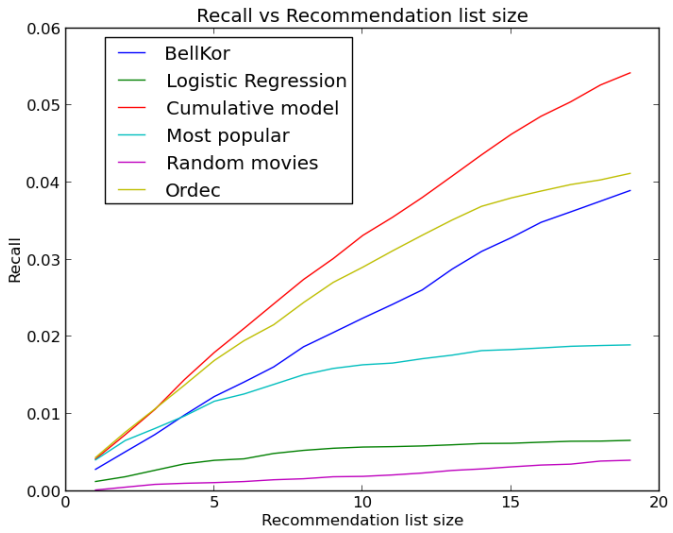


Figure 4.3: Experiment 1: Recall vs recommendation list size

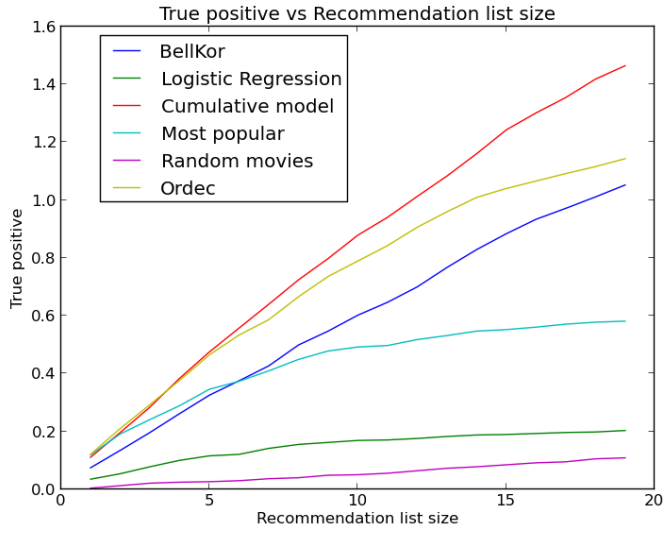


Figure 4.4: Experiment 1: Average number of true positives per user vs recommendation list size

ever, on an average only 100 movies are rated by the users. So a whole bunch of recommendations were being deemed negative, not because they were bad recommendations (not rated as 5) but because they were just never rated by the user. This is not a drawback of the recommendation system(s) being evaluated but of our experimental setup. This drawback is countered in the next experiment.

We have also plotted other metrics like average number of false positive vs recommendation list size, average number of false negative vs recommendation list size to help compare between the models. These plots can be seen in Section A.1

4.2 Experiment 2

To overcome the drawback of our experiment setup we observed in the previous experiment, we made the following adjustment. Instead of asking the recommendation models to come up with a list of recommendations from the list of all possible movies, we asked them to rank the movies in the test set of the user and return the recommendation list specific to the user test set. Now we are guaranteed that the recommendation being made by the model is already rated by the user and hence, we can fairly assign a positive or negative label to it depending on the rating it received by the user in question.

Figure 4.5 shows the precision at n vs recall at n curve for the six models. Figure 4.8 shows the ROC curve for the six models. We can observe that in both curves, the cumulative model is dominating the logistic regression model (LRM) and other baselines. LRM is doing even worse than the most popular movie baseline. As done in the previous experiment, the plot between precision at n vs recall at n has been broken down into two separate plots; precision vs recommendation list size and recall vs recommendation list size as shown in Figure 4.6 and Figure 4.7 respectively.

To further validate our results we have also plotted other metrics like average number of true positive vs recommendation list size, average number of false positive vs recommendation list size. These plots can be seen in Section A.2

We also plotted the cost curve for this experiment in Figure 4.9. As explained in section Section 2.3.3, cost curves allow us to compare classifiers at different values of probability of positive samples. Again we see that cumulative model is

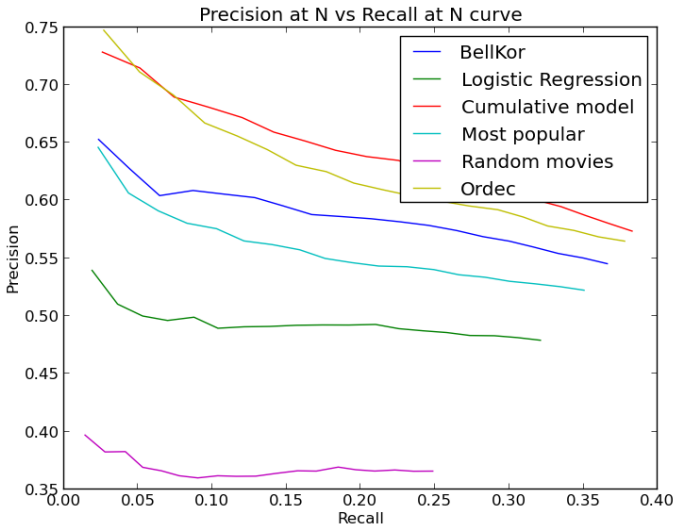


Figure 4.5: Experiment 2: Precision at n vs recall at n

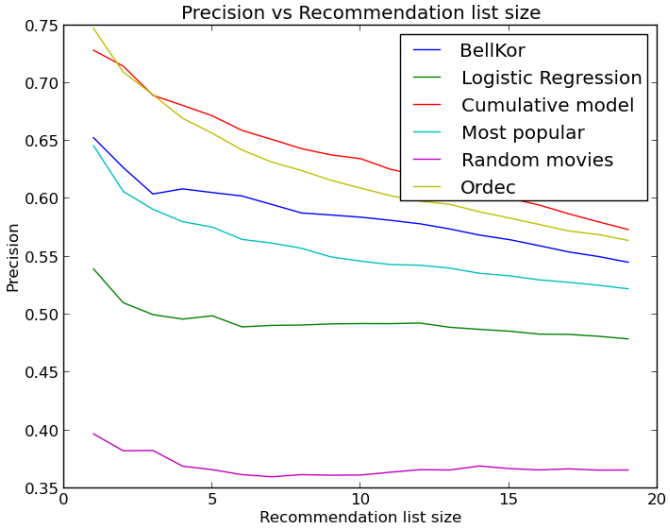


Figure 4.6: Experiment 2: Precision vs recommendation list size

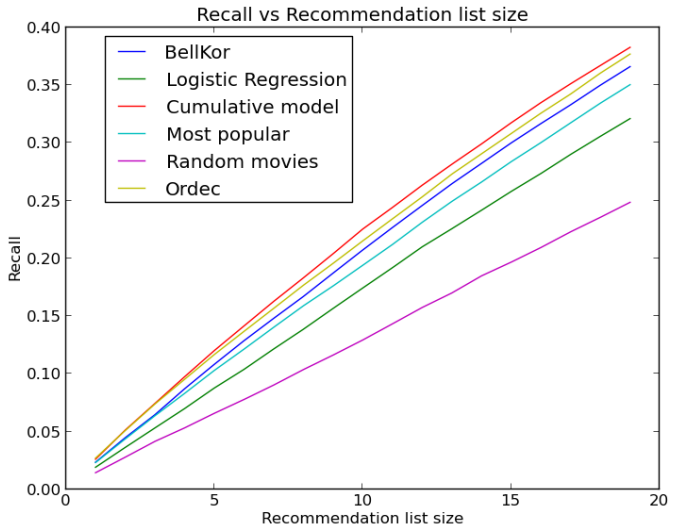


Figure 4.7: Experiment 2: Recall vs recommendation list size

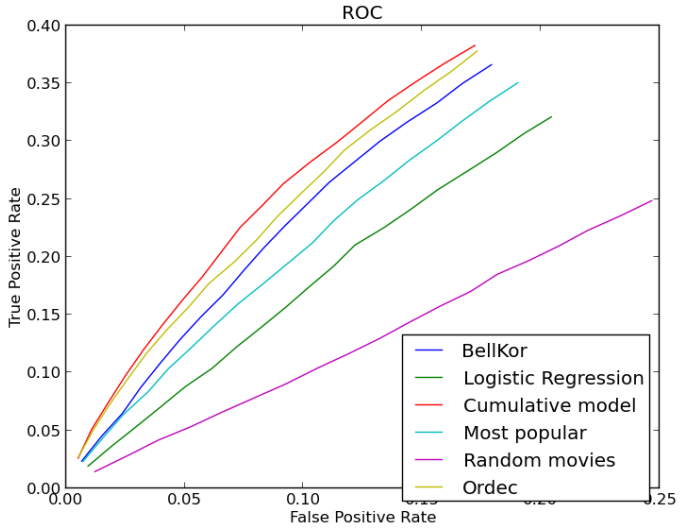


Figure 4.8: Experiment 2: Receiver operating characteristic curve

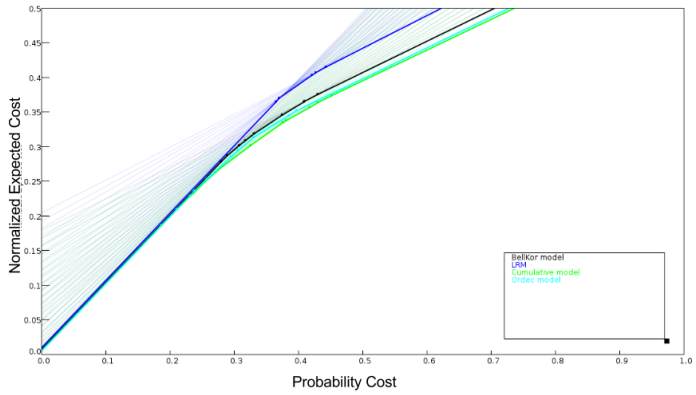


Figure 4.9: Experiment 2: Cost curve

Recommendation Model	FCP
BellKor model	0.6252
LRM	0.6259
Cumulative model	0.6766
Ordec model	0.6187

Table 4.1: Experiment 3: FCP values for the four models on the 1 million MovieLens data set. A higher value reflects better ranking accuracy.

doing better than LRM and Bellkor solution Figure 4.9.

4.3 Experiment 3

In the first two experiments, we have just evaluated the system on the basis of how well it is able to predict movies which are likely to be rated as 5 by the user. However, we also want to evaluate the system on the basis of how well they are able to distinguish other ratings from each other i.e., it is able to distinguish between a movie rated 2 and a movie rated 4. To evaluate the systems over the entire rating structure, we use the fraction of concordant pair measure as explained in Section 2.3.4. Table 4.1 shows the result. Again we see that the cumulative model is doing better than LRM and the BellKor solution.

Recommendation Model	FCP
BellKor	0.6330
LRM	0.6157
Cumulative Model	0.6621
Ordec model	0.6289

Table 4.2: Experiment 4: FCP values for the four models on the 10 million MovieLens data set

4.4 Experiment 4

The previous three experiments were carried out on the same data set. To ensure that our model was not over-fitting to the data set, we carried out the same set of experiments on another data set of a much larger size. We used the 10 million rating MovieLens data set [16] which has ratings for 17000 movies from 70000 users. Moreover, the ratings now have 10 categories between 0.5 to 5.

We carried out the same experiments. We filtered the data set to have only those users for which there are at least 20 training items in the test set. The models were asked to present personalized recommendation list for users of varying sizes (from 1 to 20). For the different recommendation list sizes, we calculated and plotted precision, recall and false positive rate. Figure 4.10 shows the precision at n vs recall at n curve, Figure 4.13 shows the ROC curve whereas Figure 4.14 shows the cost curve. We can observe that as the number of categories is increasing and the distance between them is decreasing, the cumulative model, Ordec model and BellKor model have similar performance on the different measures. Again to make it more clear, the plot between precision at n vs recall at n has been broken down into two separate plots; precision vs recommendation list size and recall vs recommendation list size as shown in Figure 4.11 and Figure 4.12 respectively.

We also calculated the fraction of concordant pairs for the four recommendation models on this data set. Table 4.2 contains the results. Again we observe that the cumulative model is beating the other models. To make it more evident, we have also plotted other metrics like average number of true positive vs recommendation list size, average number of false positive vs recommedation list size etc. These plots can be seen in Section A.3

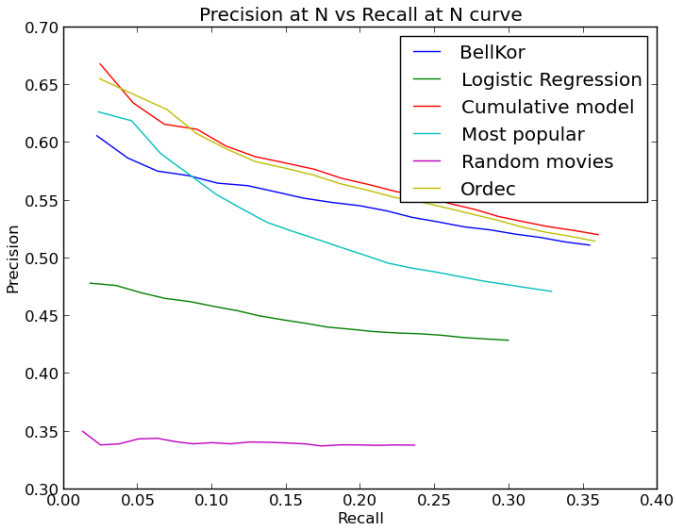


Figure 4.10: Experiment 4: Precision at n vs recall at n

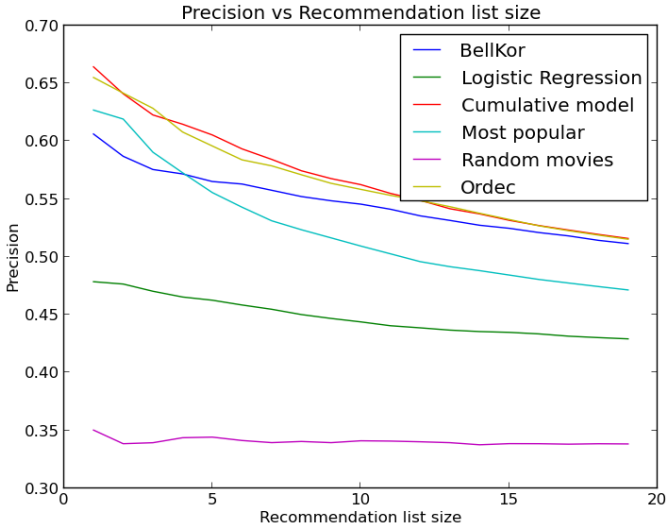


Figure 4.11: Experiment 4: Precision vs recommendation list size

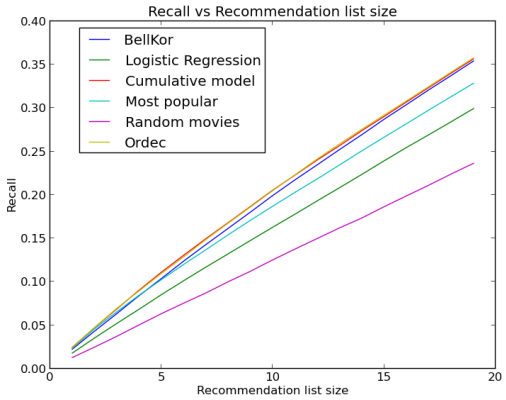


Figure 4.12: Experiment 4: Recall vs recommendation list size

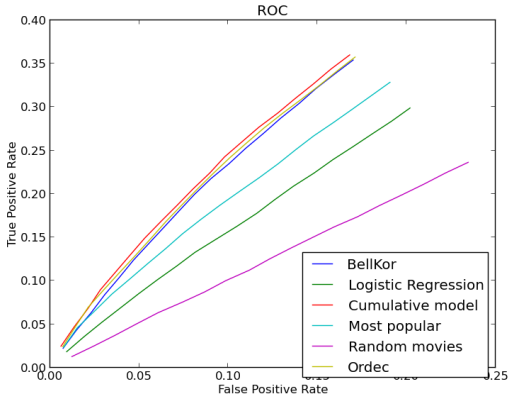


Figure 4.13: Experiment 4: Receiver operating characteristic curve

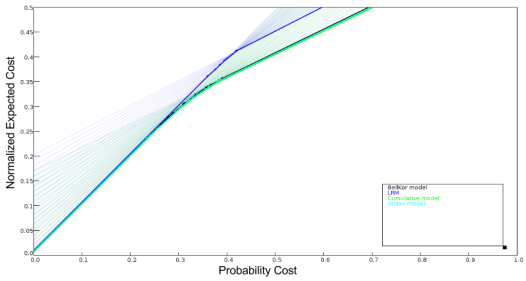


Figure 4.14: Experiment 4: Cost curve

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The experiment and results section allows us to make the following conclusions:

- The cumulative model does better than the LRM and other baselines including BellKor and Ordec model in predicting movies which would be rated 5 by the users. However, the precision and recall values are very small.
- The cumulative model does better than LRM and other baselines including BellKor and Ordec model in predicting movies which would be rated 5 by the user given the set of movies which are rated by the user.
- The cumulative model is able to do better than the LRM, BellKor and Ordec model on the FCP measure. This means the cumulative model is able to do better at learning the rating pattern of the user.

These results have been verified for data sets of varying sizes, categories and positive sample probability distributions. In all cases, we can say that the cumulative model does a good job at coming up with personalized recommendations.

The LRM is a very intuitive and straight-forward model. The idea behind it was very simple; if we want to predict movies with high ratings, we should simply learn a binary classifier for the same. However, in doing so we lost out on crucial information. In particular, when we train the LRM for rating 5, we treat data point

with other ratings as a negative sample. In doing so, we don't distinguish between a rating of 1 or 2 or 3 or 4 which is a major design flaw. 4 is clearly more close to 5 than 1 and this information should be utilized rather than making a binary decision of handing out positive and negative labels. This information is captured in our cumulative model. It is able to understand the ordering of the ratings and understand that 4 is closer to 5 as compared 3 and so on. Because of this basic difference in the way our two models treat the data, the cumulative model is able to do better than the LRM.

BellKor is a very simple and well principled method which tries to capture the latent characteristics of users and movies and makes predictions based on the interaction between these latent characteristics. The major drawback with BellKor is that it treats the ratings as numbers. In doing so, it implicitly assumes that the rating have a linear structure, i.e., the difference between a 2 stars and a 3 stars is the same as a 4 stars and a 5 stars. There is no valid reason to accept this assumption and it rather seems to be a very strong prior assumption on the user rating pattern. The cumulative model doesn't make this strong assumption. The cumulative model also tries to capture the latent characteristic of users and movies and assumes that the affection of a user towards a movie can be explained in terms of interaction between these latent factors. However, we learn an additional set of parameters for each user in our model. This additional set of parameters is used to map the ground truth reflecting the affection between a user and movie to the rating user would give to the movie. Since, we are learning an additional set of parameters, it gives our model a lot of flexibility to learn non-linear rating structure like 2 and 3 stars can be very far apart but 4 and 5 stars can be very close to each other. The only constraint is that the rating structure should be monotonic. Due to this additional flexibility provided in our model, the cumulative model is able to do better than the BellKor solution which is evident from our experiments and results.

5.2 Future work

Our experiments and results have shown that the cumulative model for recommendation is a very promising venture. There are various ways in which we would like to take this work forward. Some of them are:

1. We have used a very basic form of BellKor solution. Over the years, to gain marginal benefits over the existing model, new features have been added to the original BellKor model [23]. But we can extend our model very easily to wrap around these new features by changing the definition of $Z_{u,i}$ in (3.6). It would be interesting to note how our model would compare against these extensions of BellKor solution.

2. We made a modeling assumption in the cumulative model. We assumed that: $Z_{u,i} \sim \mathcal{N}(h_{u,i}, 1)$. We would like to experiment with different probability distributions and study the effect of choice of probability distribution on the recommendation system accuracy and scalability. One possible alternative is to use the extreme value distribution if we are mainly interested in high ratings.

3. We observed in our empirical studies that initializing the user specific cutoff randomly led to a very numerically unstable recommendation system. To overcome this problem, we made two adjustments:

   - We introduced the constraints on the cutoff explicitly into our cost function using the barrier function. In literature, there are various other ways [11, 14] of dealing with a constrained optimization problem. We would like to explore these different avenues and see how our system responds.

   - The other adjustment we made was to train the system assuming there is just one user and then use the final cutoffs as some sort of global average value for the cutoff which were used to initialize the cutoffs for other users by adding some random Gaussian noise. This step can be interpreted as using a hierarchical prior for the distribution of the cutoff where we define the prior for each user's cutoffs by using information available about other users. After this initialization we learn the deviation from the prior for each user. This motivates us to do come up with a Bayesian cumulative model. This goes hand in hand with choosing an alternative probability distribution because cumulative Gaussian distribution is known to not have a conjugate prior. On the other hand,

in [13] authors showed that all members of the exponential family distribution have a conjugate prior. Therefore, for Bayesian analysis of the problem, we need to change our choice of probability distribution as well.

4. A very common problem faced by recommendation systems is the cold start problem [28, 30]. In the cold start problem, our model have access to very little or no training data for a user. This is a recurring use case which occurs when a new user joins the website. In the literature, many solutions have been proposed to tackle this problem [1, 2, 28]. One common solution is to use user specific features like age, sex, etc., and movie specific features like genre, cast etc. in the recommendation system to make the predictions. We would also like to study how we can extend our existing model to incorporate these observed user and movie specific features, and how we compare against other recommendation systems specially designed to handle this problem.

Bibliography

[1] D. Agarwal and B.-C. Chen. flda: matrix factorization through latent dirichlet allocation. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 91–100. ACM, 2010. → pages

[2] H. J. Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51, 2008. → pages

[3] A. Albadvi and M. Shahbazi. A hybrid recommendation technique based on product category attributes. *Expert Systems with Applications*, 36(9): 11480–11488, 2009. → pages

[4] M. Balabanović and Y. Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997. → pages

[5] R. M. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the netflix prize, 2007. → pages

[6] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007. → pages

[7] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002. → pages

[8] I. Cantador, A. Bellogín, and D. Vallet. Content-based recommendation in social tagging systems. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 237–240. ACM, 2010. → pages

[9] P. Cremonesi, F. Garzotto, and R. Turrin. Investigating the persuasion potential of recommender systems from a quality perspective: An empirical

study. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2(2):11, 2012. → pages

[10] C. Drummond and R. C. Holte. Cost curves: An improved method for visualizing classifier performance. 2006. → pages

[11] M. A. Figueiredo, R. D. Nowak, and S. J. Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *Selected Topics in Signal Processing, IEEE Journal of*, 1(4): 586–597, 2007. → pages

[12] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer Series in Statistics New York, 2001. → pages

[13] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. CRC press, 2013. → pages

[14] P. E. Gill, W. Murray, and M. H. Wright. Practical optimization. 1981. → pages

[15] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982. → pages

[16] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 1999. → pages

[17] H. Hindi. A tutorial on convex optimization ii: duality and interior point methods. In *American Control Conference, 2006*, pages 11–pp. IEEE, 2006. → pages

[18] T. Hofmann. Collaborative filtering via gaussian probabilistic latent semantic analysis. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 259–266. ACM, 2003. → pages

[19] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254. ACM, 2001. → pages

[20] Y. Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81, 2009. → pages

[21] Y. Koren and J. Sill. Collaborative filtering on ordinal user feedback. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 3022–3026. AAAI Press, 2013. → pages

[22] O. Koyejo, S. Acharyya, and J. Ghosh. Retargeted matrix factorization for collaborative filtering. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 49–56. ACM, 2013. → pages

[23] R. Kumar, B. K Verma, and S. Sunder Rastogi. Social popularity based svd++ recommender system. *International Journal of Computer Applications*, 87(14):33–37, 2014. → pages

[24] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003. → pages

[25] L. Martínez, L. G. Pérez, and M. Barranco. A multigranular linguistic content-based recommendation model. *International Journal of Intelligent Systems*, 22(5):419–434, 2007. → pages

[26] P. McCullagh. Regression models for ordinal data. *Journal of the royal statistical society. Series B (Methodological)*, pages 109–142, 1980. → pages

[27] A. Y. Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004. → pages

[28] S.-T. Park, D. Pennock, O. Madani, N. Good, and D. DeCoste. Naïve filterbots for robust cold-start recommendations. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 699–705. ACM, 2006. → pages

[29] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001. → pages

[30] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002. → pages

[31] M. Schmidt. Convergence Rate of Stochastic Gradient with Constant Step Size. Technical report, University of British Columbia, Department of Computer Science, 09 2014. → pages

[32] G. Shani and A. Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011. → pages

[33] S. J. Wright. On the convergence of the newton/log-barrier method. *Mathematical programming*, 90(1):71–100, 2001. → pages

[34] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008. → pages

Appendix A

Additional Plots

In this appendix, we have presented some additional plots which should help the reader in understanding the results of our experiments and the conclusions drawn from them.

A.1 Additional plots for experiment 1

In this section, we are presenting some additional plots for experiment 1.

1. Figure A.1 shows the plot between the average number of false negative per user vs the size of recommendation list.

2. Figure A.2 shows the plot between the average number of false positive per user vs the size of the recommendation list.

We can see that cumulative model is dominating LRM and the other baselines in all the above mentioned plots.

A.2 Additional plots for experiment 2

In this section, we are presenting some additional plots for experiment 2.

1. Figure A.3 shows the plot between the average number of true positive per user vs the size of recommendation list.

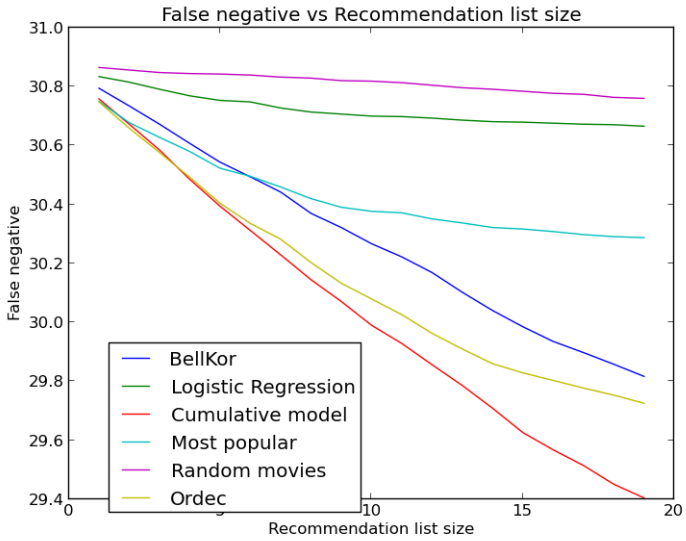


Figure A.1: Experiment 1: Average number of false negative per user vs recommendation list size

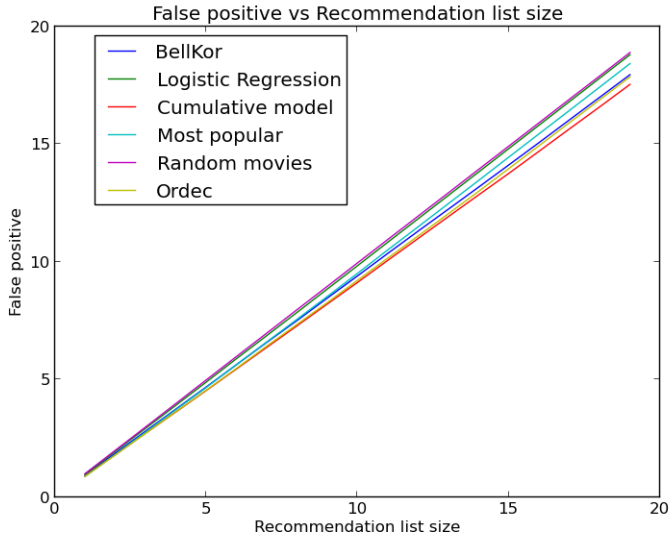


Figure A.2: Experiment 1: Average number of false positive per user vs recommendation list size

Figure A.3: Experiment 2: Average number of true positive per user vs recommendation list size

2. Figure A.4 shows the plot between the average number of false negative per user vs the size of recommendation list.

3. Figure A.5 shows the plot between the average number of true negative per user vs the size of the recommendation list.

4. Figure A.6 shows the plot between the average number of false positive per user vs the size of the recommendation list.

We can see that cumulative model is dominating LRM and other baselines in all the above mentioned plots.

A.3 Additional plots for experiment 4

In this section, we are presenting some additional plots for experiment 4.

1. Figure A.7 shows the plot between the average number of true positive per user vs the size of recommendation list.

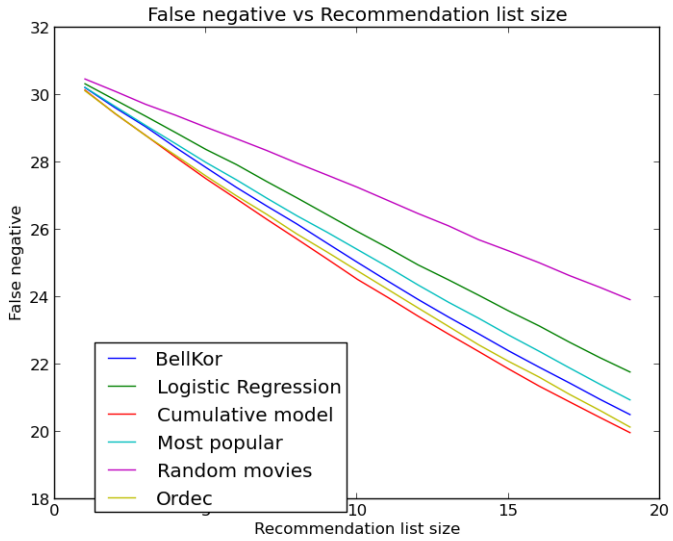


Figure A.4: Experiment 2: Average number of false negative per user vs recommendation list size

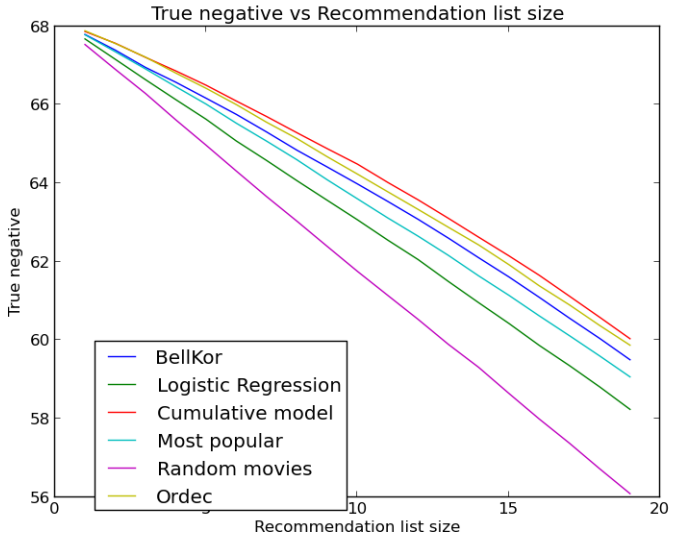


Figure A.5: Experiment 2: Average number of true negative per user vs recommendation list size

Figure A.6: Experiment 2: Average number of false positive per user vs recommendation list size

2. Figure A.8 shows the plot between the average number of false negative per user vs the size of recommendation list.

3. Figure A.9 shows the plot between the average number of true negative per user vs the size of the recommendation list.

4. Figure A.10 shows the plot between the average number of false positive per user vs the size of the recommendation list.

We can see that cumulative model is very similar in performance as BellKor and Ordec model in all the above mentioned plots.

Figure A.7: Experiment 4: Average number of true positive per user vs recommendation list size

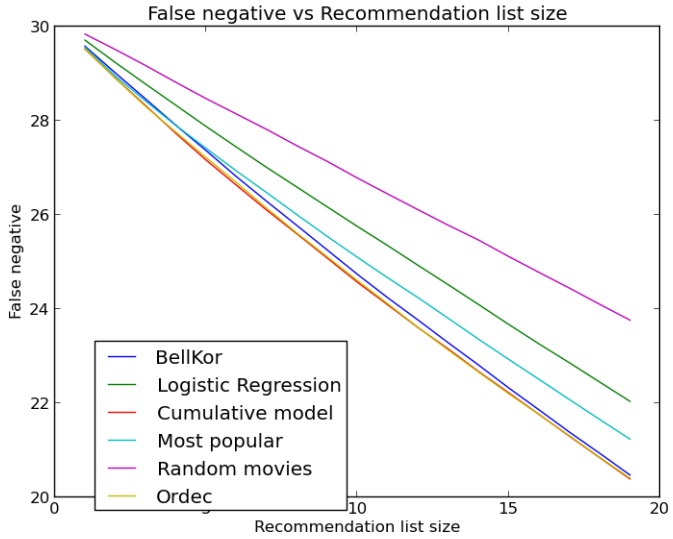


Figure A.8: Experiment 4: Average number of false negative per user vs recommendation list size

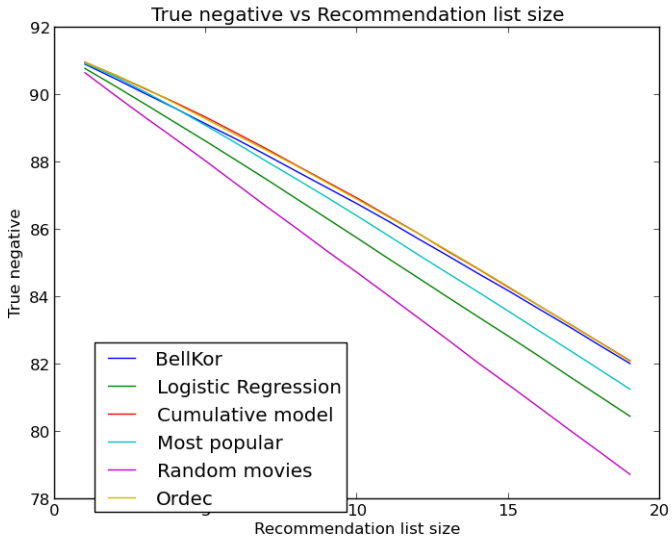


Figure A.9: Experiment 4: Average number of true negative per user vs recommendation list size

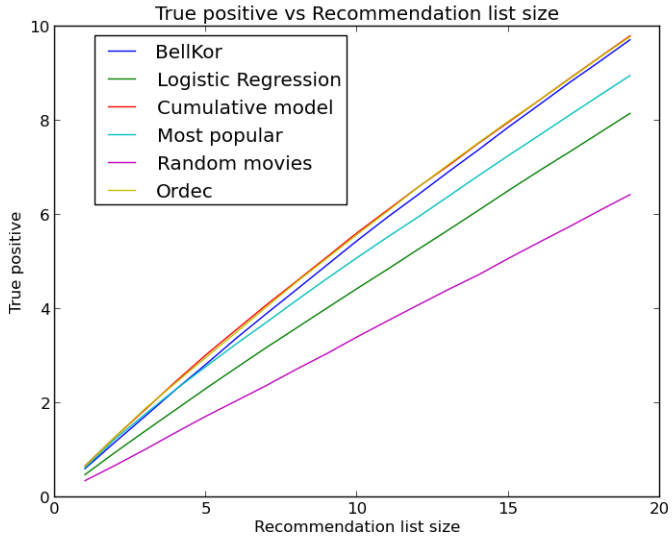


Figure A.10: Experiment 4: Average number of false positive per user vs recommendation list size