The Report Committee for Wenjing Zhan
Certifies that this is the approved version of the following report:

Clicks Prediction with L1 Regularized Logistic Regression and A Study on Poisson Factorization Recommender Evaluation

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: _____
Mingyuan Zhou

_____
Matthew Hersh

Clicks Prediction with L1 Regularized Logistic Regression and A Study on Poisson Factorization Recommender Evaluation

by

Wenjing Zhan, B.S.;M.A.;M.A.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Statistics

The University of Texas at Austin

August 2014

Abstract

Clicks Prediction with L1 Regularized Logistic Regression and A Study on Poisson Factorization Recommender Evaluation

Wenjing Zhan, M.S. Stat

The University of Texas at Austin, 2014

Supervisor: Mingyuan Zhou

The key task for a search engine advertising system is, for each query that the search engine receives, to choose what advertisement should be displayed, and in what order. This ranking order has a strong impact on the revenue the search engine receives from the ads. Meanwhile, showing the user an advertisement that they prefer to click on improves user satisfaction. Therefore, it is reasonable to set up click-through rate (CTR) as a weighted ranking criteria. In this project, we aim to develop a model capable of accurately predicting CTR of ads in the system. For ads that have been displayed repeatedly, this is empirically measurable, but for new ads, other means must be used. Combining logistic regression model with l1 regularization we are able to predict CTR for new ads based on self-defined features. The ultimate goal is to improve the convergence and performance of our advertising system, consequently increasing both revenue and user satisfaction.

Second part of this report is about a study on poisson factorization recommender evaluation. As recommender systems become more and more popular, many approaches have been suggested to evaluate the performance of traditional recommenders based on Gaussian distribution. However, few evaluating approaches were designed for poisson factorization recommender. This study checked some most common evaluation methods and discussed about their appropriateness in poisson factorization recommender evaluation.

Table of Contents

Chapter 1

Clicks Prediction

1.1 Introduction

Search engine advertising has become a significant element of the Web browsing experience. Choosing the right ads for the query and the order in which they are displayed greatly affects the probability that a user will see and click on each ad. Since the probability that a user clicks on an advertisement declines exponentially, it is most beneficial for the search engine to place best performing ads first. Then the question is, how to predict the probability of an ads being clicked?

Whenever an ad is displayed on the search results page, it has some chance of being viewed by the user. The farther down the page an add is displayed, the less likely it is to be viewed. As a simplification, we consider the probability that an ad is clicked on to be dependent on two factors:

- the probability that an ad is viewed.
- given it is viewed, the probability that the ad is clicked on.

Based on above assumption, given an ad being presented on the webpage and its ranking position, the probability distribution of users clicking on this ad

can be described as below:

$$p(click|ad, pos) = p(click|ad, pos, seen) \cdot p(seen|ad, pos)$$

Let us add two more assumptions. First, assume that given an ad was viewed, the probability of it being clicked is independent of its position. Second, given ranking position fixed, the probability an ad is viewed is independent of its quality, and independent of the other ads shown. Then the above formula can be rewritten as:

$$\begin{aligned} p(click|ad, pos) &= p(click|ad, seen) \cdot p(seen|pos) \\ &= CTR \cdot p(seen|pos) \end{aligned}$$

In this project we focus on predicting CTR but not $p(seen|pos)$. One reason is that estimating $p(seen|pos)$ is experimentally expensive, another reason is that intuitively CTR is much more essential in improving revenue and user satisfaction.

Our main goal is to determine and quantify CTR-related ad features, train the feature dataset for validated parameters that can be used for new CTR prediction. Logistic regression model is used for CTR prediction [1]. The logistic regression was initially trained using the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method [2]. A cross-entropy loss function was used with Gaussian weight priors with zero-mean and a standard-deviation of σ. According to Matthew Richardson, through cross-validation test $\sigma = 0.1$ was empirically best out of [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]. Here we simply employ $\sigma = 0.1$ yet a cross-validation test may worth being tested.

The self-defined feature set contains continuous and categorical features. For each continuous feature f_j, derivation $log(f_j + 1)$ and f_j^2 were added as well. To expedite optimization process and to simplify variable interpretation, all continuous features are normalized. KL-divergence was used as measure of performance.

A simulation test was originally conducted to check availability of this dataset. After simulation test, data were collected. The original model yielded error from 5000 to Infinity, mainly caused by noise in the dataset. Matthew Richardson et al. split data on an advertiser-level to prevent train-test contamination and filtered out any ads that had less than 100 views. After noise filtration the error dropped dramatically. Considering the 10:1 sample to feature rule, our sample is undersized. Meanwhile, adding binary unigram term features dramatically increase sparsity of the training data.

One possible explanation is that adding too many features without sufficient sample collected caused overfitting. Therefore, l1 regularized cross-entropy model was introduced for feature selection and optimization [3]. Eventually we implemented a more stable model and test error was reduced to fairly small value.

1.1.1 Logistic Regression

Logistic regression is ideally suited for probabilities as it always predicts a value between 0 and 1. Let $f_j(ad)$ represents the value of the j^{th} feature for the ad, let θ_i the learned weight for the j^{th} feature, then predicted CTR for

this ad can be expressed as:

$$C\tilde{T}R = \frac{1}{1 + e^{-\sum \theta_j f_j(ad)}}$$

For reading convenience we use y to represent collection of CTRs of n job advertisements in the training dataset. y is a $n \times 1$ vector. Similarly, $\tilde{y}$ is the collection of predicted CTR. θ is $p \times 1$ vector representing weights of the p features. X is an $n \times p$ data matrix, each row X_i represents features extracted from the i^{th} job advertisement, then rewrite above expression as:

$$\tilde{y} = \frac{1}{1 + e^{-X \cdot \theta}}$$

Use cross-entropy as loss function. For any given job, only two states are possible: $\{clicked, not - clicked\}$. y_i represents the observed possibility of being clicked, $\tilde{y_i}$ represents the estimated probability of being clicked, then the loss function can be constructed as

$$L(\theta) = \sum_{i=1}^{n} y_i ln(\tilde{y_i}) + (1 - y_i) ln(1 - \tilde{y_i})$$
$$\tilde{y_i} = \frac{1}{1 + e^{-X_i \cdot \theta}}$$

All we need is to find the very θ that minimize the loss function.

1.1.2 L-BFGS

Cross entropy loss function is a log linear model. It is convex and has unique global minimum. Based on efficiency performance we prefer Newton's method rather than gradient descent. The first (Gradient) and second (Hessian Matrix) derivative of the loss function are presented as below:

$$\frac{\partial}{\partial\theta}L(\theta) = X^T \cdot (\tilde{y} - y)$$
$$\frac{\partial^2}{\partial\theta^2}L(\theta) = X^T S X$$
$$S = diag\{\tilde{y}_i(1 - \tilde{y}_i)\}$$

Computing Gradient (usually notated as g) is easy but explicitly computing Hessian matrix (usually notated as H) for large-scale dataset is very expensive. Therefore, the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method[2], a derivative of Newton's algorithm, was applied. The key idea is, during the optimization process, for the k_{th} iteration, instead of computing the corresponding Hessian matrix explicitly, an approximation is computed based on the previous m iterations.

Denote H_k as the Hessian matrix approximation at the k^{th} iteration.

Given:

$$s_k = \theta_{k+1} - \theta_k$$

$$z_k = g_{k+1} - g_k$$

$$\rho_k = \frac{1}{z_k^T s_k}$$

$$V_k = I - \rho_k z_k s_k^T$$

We can compute H_k recursively as

$$H_k = V_{k-1}^T H_{k-1} V_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^T$$

Repeat this computation recursively to the previous m iterations

$$\begin{aligned} H_k = &(V_{k-1}^T \cdots V_{k-m}^T) H_k^0 (V_{k-m} \cdots V_{k-1}) \\ &+ \rho_{k-m}(V_{k-1}^T \cdots V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdots V_{k-1}) \\ &+ \rho_{k-m+1}(V_{k-1}^T \cdots V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdots V_{k-1}) \\ &\cdots \\ &+ \rho_{k-1} s_{k-1} s_{k-1}^T \end{aligned}$$

We choose $H_k^0 = \gamma_k I$. Heuristically this choice helps to ensure that the search direction p_k is well scaled and as a result the step length $\alpha_k = 1$ is accepted in most iterations.

$$\gamma_k = \frac{s_{k-1}^T z_{k-1}}{z_{k-1}^T z_{k-1}}$$

Algorithm 1 L-BFGS two-loop recursive formular to compute $H_k g_k$[4]$_{pp178}$

$q = g_k$
for $i = k-1, k-2, ..., k-m$ **do**
  $\alpha_i = \rho_i s_i^T q$
  $q = q - \alpha_i z_i$
$H_k^0 = \frac{s_{k-1}^T z_{k-1}}{z_{k-1}^T z_{k-1}} I$
$r = H_k^0 q$
for $i = k-m, k-m+1, ..., k-1$ **do**
  $\beta = \rho_i z_i^T r$
  $r = r + s_i(\alpha_i - \beta)$
STOP with $r = H_k g_k$

Algorithm 2 L-BFGS [4]$_{pp179}$

Choose starting point θ_0, integer m ¿ 0
$k = 0$
while true **do**
  Compute $p_k = -H_k g_k$ from Algorithm 1
  Compute $\theta_{k+1} = \theta_k + \alpha_k p_k$ where α_k is chosen to satisfy Wolfe conditions;
  if k greater than m **then**
    Discard the vector pair s_{k-m}, z_{k-m} from storage;
  Compute and save $s_k = \theta_{k+1} - \theta_k, z_k = g_{k+1} - g_k$;
  $k = k + 1$
untile converge

According to the reference [4], termination criterion $||g_k|| \leq 10^{-5}$ is used. In the real implementation of Wolfe conditions, to improve efficiency this routine checks only one of the Wolfe conditions. Since our loss function is convex, this method appropriate. If the objective is not convex it may lead to non-positive-definite Hessian approximations and non-descent search directions.

1.1.3 L-1 Regularization

One disadvantage of L-BFGS is, even though the loss function is convex and global minimum is guaranteed, because of line search and approximation, this method is strongly dependent on the initial guess. Another limitation of this method is, like many other algorithms, cannot handle overfitting problem with oversized feature set.

Therefore we modify the loss function with l1 regularization.

$$f(\theta) = L(\theta) + C \sum |\theta_j|$$

with C ¿ 0

This regularization is validate based on following observation: when restricted to any given orthant, the loss function is still differentiable and convex. Further, the second-order behavior of the regularized objective is not affected by the l-1 regularization part. This consideration suggests the following strategy: construct a quadratic approximation that is valid for some orthant containing the current point using the inverse Hessian estimated from the loss component alone, then search in the direction of the minimum of the quadratic, restricting the search to the orthant on which the approximation is valid.

The key change is to replace first-order gradient of $L(\theta)$ with pseudo-gradient of $f(\theta)$.

$$g_j^*(\theta) = \begin{cases} \partial_j^- f(\theta) & \text{if } \partial_j^- f(\theta) > 0 \\ \partial_j^+ f(\theta) & \text{if } \partial_j^+ f(\theta) < 0 \\ 0 & \text{otherwise} \end{cases}$$

where the left and right partial derivatives of f are given by

$$\partial_j^{\pm} f(\theta) = \frac{\partial}{\partial \theta_j} L(\theta) + \begin{cases} C * sgn(\theta_j) & \text{if } \theta_j \neq 0 \\ \pm C & \text{if } \theta_j = 0. \end{cases}$$

If $f(\theta)$ is convex, it can be prove that $\partial_i^- f(\theta) \leq \partial_i^+ f(\theta)$. The pseudo-gradient generlizes the gradient in that the directional derivative at θ is mnimized in the direction of $-g(\theta)$, and θ is a local minimum if and only if $g(\theta) = 0$.

The l1 regularized log-linear model is called Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) method. Compared to a normal L-BFGS, the only differences have been marked in the figure:

1. The pseudo-gradient $g^*(\theta)$ of the regularized objective is used in place of a pure corss-entropy gradient.

2. The resulting search direction is constrained to match the sign pattern of $-g^*(\theta)$. This is to make sure that orthant chosen to explore is the one containing θ and into which $-g^*(\theta)$ leads.

3. During line search, each search point is projected onto the orthant of the previous point.

4. The gradient of the un-regularized loss alone is used to construct the vector z_k used to approximate the Hessian matrix.

1.2 Data Collection

Data collected for this experiments are listed as below:

- jobs clicked
- jobs viewed
- job text descriptions

To reduce multicollinearity, features that are explicitly linearly correlated to each others are removed. Potential multicollinearity will be reduced by l1 regularization. Both raw features and processed features are listed in APPENDIX.

Raw data files were collected during one day. Only jobs with at least one click will be considered. This dramatically decrease sample size from 1,000,000 to around 10,000 jobs. How these jobs were viewed follows poisson distribution. Only 8,000 instances were selected after noise reduction. Cross validation was set as 80% training, 10% validation and 10% testing.

1.3 Feature Selection

Some features selected for this project worth being discussed. So we presented the reason of including such features below

1.3.1 Industry CTR

In Matthew Richardson paper they assume that CTR for different bid terms are different. It is hard for us to track the bid term for each job advertisement. So we use industry CTR as a predicting factor. We assume that all CTR for different industries follows a basic prior distribution as $p(y_{ind}) \sim N(\overline{y}, \sigma^2)$ where $\overline{y}$ is the mean CTR for all ads.

Given a specific job i from industry ind, we assume $y_{i,ind} \sim N(y_{ind}, \tau^2)$. If let n_{ind} represents the number of ads for industry ind, it can be proved that the posterior distribution of industrial level CTR, y_{ind}, is

$$
\begin{aligned}
y_{ind} &\sim N(m_{ind}, V_{ind}) \\
m_{ind} &= (\frac{1}{\sigma^2} + \frac{n_{ind}}{\tau^2})^{-1}(\frac{1}{\sigma^2}\overline{y} + \frac{1}{\tau^2}\sum_{i=1}^{n_{ind}} y_{i,ind}) \\
V_{ind} &= (\frac{1}{\sigma^2} + \frac{n_{ind}}{\tau^2})^{-1}
\end{aligned}
$$

If set $\tau^2 = 1, \frac{1}{\sigma^2} = \alpha$, we can rewrite m_i in the same format as:

$$
m_{ind} = \frac{\alpha\overline{y} + \sum_{i=1}^{n_{ind}} y_{i,ind}}{\alpha + n_{ind}}
$$

1.3.2 Term Expectation

As mentioned above, unlike a general advertisement, it is hard for us to track bid term associated with each job advertisement. Consequently we were not able to compute term CTR accurately. Instead, we computed a term expectation to reflect the user's expectation associated with a specific query.

Then by adding up each term's expectation, we can roughly derive a predicted expectation of the job.

1.3.3 Estimating AD Quality

The contents of the ads itself should be considered as important features since Web searchers consider the summary, the title, and the URL of an advertisement in deciding whether to click it. Therefore, we derived a number of features that we hoped would be indicative of the quality of the ad for that category.

- **Appearance:** How many words are in the title? In the body? Does it contain too many exclamation points, or other punctuation? Does it use short words or long words?

- **Attention Capture:** We can combine this feature with the unigram features. By asking, if this ads contain some most common and useful words? These features are intended as an automatic way to capture some of the same influences that our manual features do. For each of the most common 100 words in the ad title and 1,000 in ad body of the training set, we add a feature which takes the value 1 if the word exists in the ad and 0 otherwise.

landing page quality and relevance may be too expensive for us to collect so we neglect them at this moment.

1.3.4 Language Model and Term Frequency

Language model and tf-idf are also included in the feature set. Term frequencies of user's input query were counted from title and description. Based on a global term tf-idf library, we would be able to compute tf-idf and various language model probabilities as features as well.

1.3.5 Post Processing

Finally, the predicted ad CTR $\tilde{y}_{ij}$ for ad i be used as prior mean and constantly being adjusted by real data observations. Assume the real CTR for ad i $y_i \sim N(\tilde{y}_i, \gamma^2)$, then given number of clicks and views of this ad, based on the same Bayesian Theorem, the real CTR can be adjusted by

$$y_i = \frac{\alpha\tilde{y}_i + clicks}{\alpha + views}$$

α set the strength of the prior and is adjustable.

1.4 Experiments

Raw data files were pre-processed through C++ projects to collect all related features mentioned above. Then a python project was implemented for OWL-QN optimization. Cross validation was conducted based on 8:1:1 (training:validation:testing) ratio. The cross validation process was conducted multiple times and each iteration dataset was randomly reshuffled. Results of cross validation on KL Divergence can be found in Table 1.1. Once validation error became smaller than pre-decided threshold, corresponding θ was consid-

Table 1.1 : KL Divergence Cross Validation

iteration id	**valid error**	**test error**
1	14.88	15.79
2	16.14	16.14
3	15.96	14.91
4	16.53	17.20
5	15.75	17.64
6	16.42	15.58
7	15.51	16.27
8	16.63	14.88
9	14.78	16.80
10	15.05	18.13

ered to be valid and a test error will be computed. With L1 regularization and stringent penalty regulator, most parameters were filtered out to 0. Non zero features are considered to be important.

1.5 Discussion and Future Work

Based on above result we can see that this logistic regression model is indeed feasible in predicting click through rate and is yielding meaningful result. Except for industrial subgroup and state id, almost all other features were demonstrated, more or less, as important. The next step we hope to check the performance of this model and implement it into the search engine.

The currently dataset is collected only for one day. After filtrations the over-one-million sample set was reduced to 8,000 jobs, which has more than 5 views and have been clicked at least one time. This indicates a roughly

0.8% collection rate from raw data with 5 views. If we want to increase the threshold of job views to higher value and collect a fairly decent dataset, we need to collect data from multiple days instead of one day. To generate more valid samples, we need to extend the time of data collection. The unigram library and the user query expectation library will be re-generated as well. L-1 regularization was introduced because of the small sample size. With sufficient training data, we can lower the penalty regulator and more important features can be discovered.

Lastly, how much this model can be improved by the new dataset is, in fact, not clear. After all, based on cross validation of current test, the model has already became fairly stable with small errors. If collecting large data files is expensive, we should at least start prediction on new jobs and do a dynamic monitoring to check the realistic prediction accuracy of this model.

Chapter 2

Evaluation of Recommender Systems

2.1 Introduction

The goal of a recommender system, is to predict ratings or preferences of items for a given user. Recommender systems have become extremely common in recent years, and are applied in a variety of applications. The most popular ones are probably movies, music, news, books, research articles, and products in general. However, there are also recommender systems for experts, jokes, restaurants, online dating, and twitter followers.

Three methods were most often used for recommenders: collaborative filtering, content-based filtering and hybrid recommender systems. Collaborative filtering is based on the ratings that the user gives to some items. Content-based filtering is based on the profile of users or the description of the items. Collaborative filtering can make good predictions for the rating that hasn?t been rated but its corresponding user has rated other items and its corresponding item has been rated by other users. It can not predict anything for new users or new items. Content-based filtering, instead, can do such a prediction. But sometimes the description of the items or the profile of users is not reachable. And the accuracy of this approach is sometimes not so good

as collaborative filtering, because the information is not highly rating-related. Hybrid recommender is a mixture of these two. In this study we focus on collaborative filtering method.

The most successful collaborative filtering method is a latent factor model called matrix factorization, where users and items are in a shared low-dimension(K) feature space—user i is represented by u_i and item j by v_j. And the existing ratings, R_{ij}, that user i gives to item j should be as close to the prediction given by the model, $R_{ij} = u_i^T v_j$ as possible. So the matrix factorization problem can be written as the minimization problem of regularized squared error function,

$$\min_{U,V} \sum_{i,j} (R_{ij} - u_i^T v_j)^2 + \lambda_u \|u_i\|^2 + \lambda_v \|v_j\|^2 \tag{2.1}$$

The matrix factorization can also be interpreted as a probabilistic model. In probabilistic matrix factorization, usually we assume

$$\text{user's latent feature vector is drawn from } \theta_u \sim N(0, \sigma_{\theta_u}^2 I_K)$$
$$\text{article's latent feature vector is drawn from } \beta_i \sim N(0, \sigma_{\beta_i}^2 I_K)$$
$$\text{the rating is drawn from } R_{ij} \sim N(\theta_u^T \beta_i, \sigma_{ui}^2)$$

If we set $\sigma_{\theta_u}^2 = \lambda_u^{-1}$, $\sigma_{\beta_i}^2 = \lambda_v^{-1}$ and $\sigma_{ij}^2 = 1$, solving the maximum a posteriori(MAP) estimation of this probabilistic model will be exactly the optimization problem. Meanwhile, it is equivalent to minimize squared-loss. Such optimization approach treats zeros as evidence of user disliking items.

Recent years a new factorization method based on Poisson distribution was proposed by some researchers [6, 7]. When applying such poisson factorization method to recommender system, we assume that

$$\text{user's latent feature vector is drawn from } \theta_u \sim Gamma(a, b)$$
$$\text{article's latent feature vector is drawn from } \beta_i \sim Gamma(c, d)$$
$$\text{the rating is drawn from } R_{ui} \sim Poisson(\theta_u^T \beta_i)$$

Compared to Gaussian factorization method, advantages of such hierarchical poisson factorization models are [8]:

1. HPF captures sparse factors.
2. HPF models the long-tail of users and items.
3. HPF down-weights the effect of zeros.

In Gopalan's paper [7,8] they declared that the hierarchical Poisson factorization model out-performs Gaussian model in mean precision and recall. In this study we validate their evaluation method by comparing their Bayesian nonparametric poisson factorization recommender with a well-developed Gaussian factorization recommender [9] and discuss how to compare recommenders based on their own properties. Zhou's Beta-Negative Binomial Process [6], which also provides a hierarchal Poisson factorization recommender, was included as another comparison reference.

2.1.1 Root Mean Squared Error

Root Mean Squared Error (RMSE) is one of the most popular metric used in evaluating accuracy of predicted ratings. For any given user i, let e_i be the deviance between real rating and predicted rating, then the RMSE between the predicted and actual ratings is given by:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_i^2}$$

According to the computation formula, we see that RMSE penalizes large deviances in prediction values. Assume some given test data with rating as [3,4,3,5,4] (range 0-5), the corresponding prediction by Poisson Factorization is usually [0.5, 0.6, 0.2, 2, 0.5] with most predicted ratings smaller than 1. In comparison, Gaussian factorization method did much better by giving prediction values as [4, 2.4, 2.9, 3, 3]. We see the nature of RMSE favors Gaussian factorization recommender since it provided closer predictive values, no matter such prediction is accurate or not. This is the reason why Poisson factorization recommenders performed badly with RMSE. Another reason that RMSE naturally fits Gaussian factorization recommender is that, the optimization loss function of Gaussian Process is, in fact, sum of error squares.

However, closer prediction values does not guarantee accurate prediction. In fact, based on above example, the Poisson factorization recommender has predicted more accurate ranking orders than Gaussian recommender, even though the later's RMSE evaluation score will be much better.

2.1.2 Precision and Recall

Precision and recall were the evaluation methods Gopalan et al used to compare Gaussian and Poisson recommender performances [7, 8].

$$\text{recall} = \frac{|\{\text{relevant documents}\}| \cap |\{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

$$\text{precision} = \frac{|\{\text{relevant documents}\}| \cap |\{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

In most cases, researchers assume that unrated items would have not be used even if they had they been recommended. They are not interesting to the user. However, this assumption is highly possible to be false since for a large collection of items, it is impossible for many users to rate all items and the set of unrated items may contains some interesting items that the user did not saw or selected before. A user may not have rate or use an item because of his/her unawareness of its existance, but after the recommendation exposed that item the user can decide to select it. In this case the number of retrieved is over estimated.

To compute recall and precision, select relevant documents from observed ratings and retrieved documents from prediction ratings. Usually , relevant documents were picked with rating scores equal to a certain score. Retrieved documents were picked with top N rated films from all items.

According to the computation formula, we see that precision and recall use ranking order of predicted recommendations instead of the predicted values. With proper settings, Poisson factorization recommenders can indeed yield better precision and recall.

2.1.3 NDCG

Normalized Cumulative Discounted Gain (NDCG) is a measure often used in information retrieval to compare the rankings of website urls. The ranking position for each url was discounted logarithmically. NDCG is defined as follows [10]. The DCG (Discounted Cumulative Gain) for a given set of search results (for a given query) is

$$DCG(Q,k) = \sum_{i=1}^{T} \frac{2^{l_i} - 1}{log(1+i)}$$

where T is the truncation level (for example, if we only care about the first page of returned results), and l_i is the label of the i_{th} listed URL. We typically use five levels of relevance: $l_i \in \{0,1,2,3,4\}$. The NDCG is the normalized version of this:

$$NDCG(Q,k) = \frac{DCG}{maxDCG}$$

If assume T the truncation level be number of all rated films for each user, l_i equal to the observed rating value and ranking position i be decided by predicted ratings, above formula can be used to evaluate recommender performance. Unlike RMSE which evaluates performance from numerical values of predicted items, or prediction and recalls which only use ranking order of predicted items, we see the NDCG computation include both numerical values and the ranking order of predicted items.

2.2 Experiment

2.2.1 Precision and Recall

Dataset used here is MovieLens 1M data set contains 1 million ratings collected from 6040 users and 3952 movies. If picked relevant documents with rating scores equal to 5 (highest score) and picked Retrieved documents with top 100 rated films out of all 3952 items. HPF is the hierarchical poisson factorization proposed by Gopalan, Beta-NBP is the model proposed by Zhou and LIBPMF is a well-developed Gaussian factorization recommender. Precision and recall of each user were computed and mean precision and mean recall over all users are presented as below:

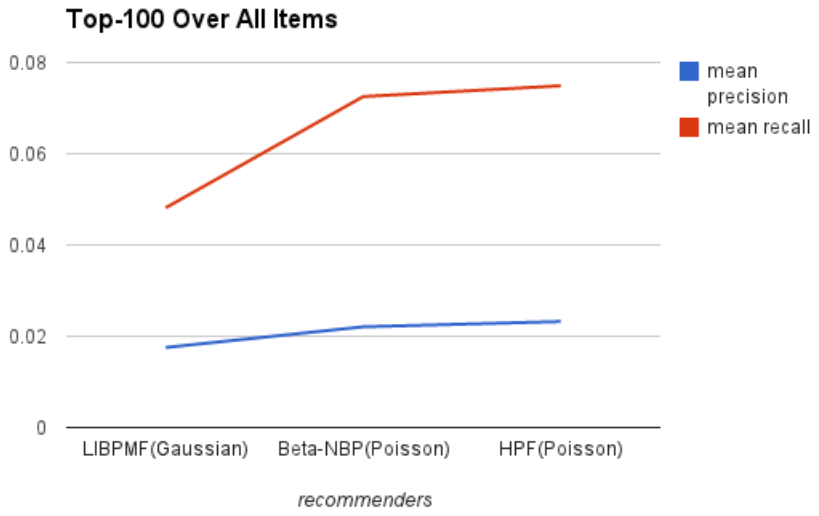

Figure 2.1: Mean Precision and Mean Recall with Top-100 Over All Items

Based on above graph we see that indeed both Poisson factorization

recommended performed better than Gaussian factorization method. However, if instead of picking retrieved documents from all 3952 items, but picking top-100 retrieved documents from only test data, the performance of Gaussian factorization recommender slightly out-perform Poisson methods, as shown below:

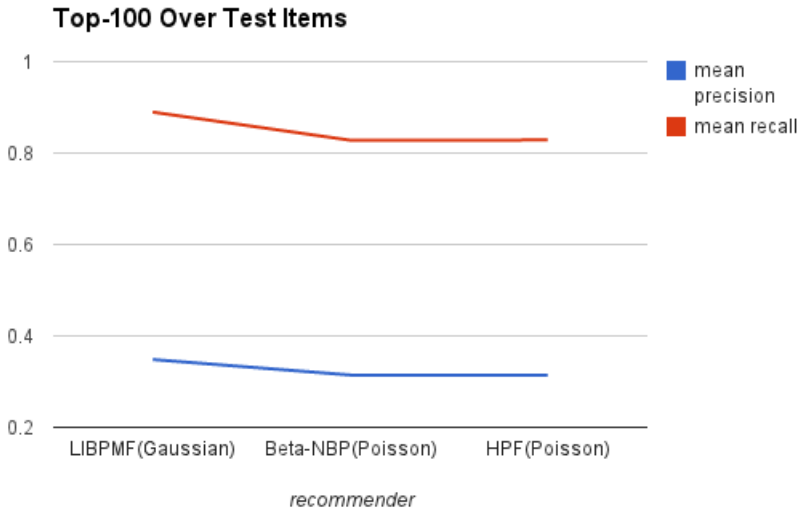

Figure 2.2: Mean Precision and Mean Recall with Top-100 Over Tested Items

2.2.2 NDCG

Compute NDCG of each user and compare the mean NDCG for all three recommenders. The result is shown as below:

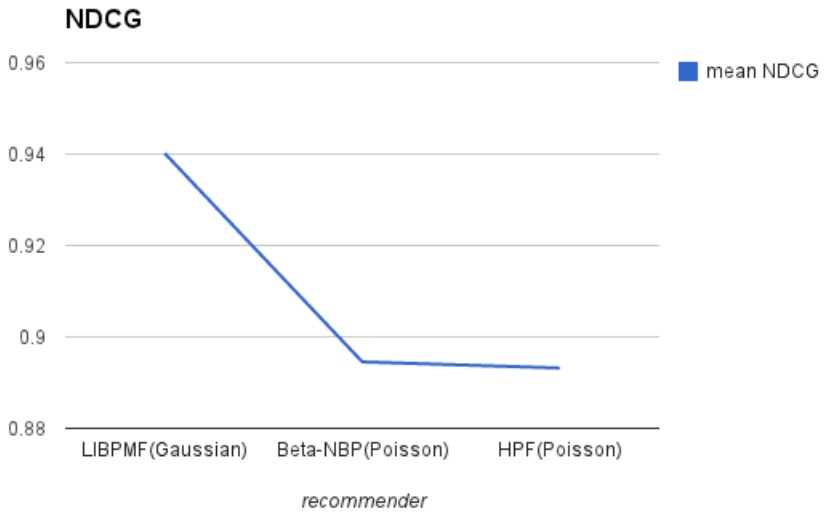

Figure 2.3: Mean NDCG Over Tested Items

Again we see that Gaussian factorization method outperformed the Poisson factorization method.

2.3 Discussion and Future Work

Based on above result we can see that, neither Poisson factorization recommender nor Gaussian factorization recommender can completely outperform the other. Gaussian factorization recommender is optimized to minimize mean square error, consequently it will give predicted ratings closer to the real values. Therefore, Gaussian recommender performed better if numerical values of predicted ratings are included in the evaluating formula. Poisson factorization recommenders applied in this study were both optimized

through sampling approaches like variational inference and MCMC. Poisson factorization recommenders performed better on evaluating large-scale recommendation order. Heuristically speaking, accuracy of prediction over all items may be more of our interests, therefore, Poisson factorization method may be a better choice.

Bibliography

[1] Matthew Richardson, Ewa Dominowska, Robert Ragno (2007), "Predicting Clicks: Estimating the Click-Through Rate for New Ads", 16th international conference on World Wide Web pp.521-530

[2] D. C. Liu and J. Nocedal (1989), On the limited memory BFGS method for large scale optimization, Mathematical Programming, vol. 45, no. 3, pp. 503528

[3] Galen Andrew, Jianfeng Gao (2007), "Scalable training of L1-regularized log-linear models", ICML '07 Proceedings of the 24th international conference on Machine learning, pp. 33-40

[4] Jorge Nocedal, Stephen Wright, "Numerical Optimization", 2nd Edition

[5] C Zhai, J Lafferty (2004), "A study of smoothing methods for language models applied to information retrieval", ACM Transactions on Information Systems (TOIS), Volume 22 Issue 2, pp. 179-214

[6] M. Zhou, L. Hannah, D. Dunson and L. Carin (2012), "Beta-Negative Binomial Process and Poisson Factor Analysis," International Conference on Artificial Intelligence and Statistics, JMLR W&CP, 22:1462-1471, La Palma, Canary Islands, Spain

[7] Prem Gopalan, Francisco J. R. Ruiz, Rajesh Ranganath, David M. Blei (2014) "Bayesian Nonparametric Poisson Factorization for Recommendation Systems", AISTATS

[8] Prem Gopalan, Jake M. Hofman, David M. Blei (2013) "Scalable Recommendation with Poisson Factorization", CoRR

[9] H. Yu, C. Hsieh, S. Si, I. Dhillon. (2013), "Parallel Matrix Factorization for Recommender Systems", Knowledge and Information Systems (KAIS)

[10] Christopher J.C. Burges (2010) "From RankNet to LambdaRank to LambdaMART: An Overview", Microsoft Research Technical Report MSR-TR-2010-82