Recruitment Analytics: Predict Applicant Behavior from Sparsely Labeled Application History

1 Introduction

2 Formulation

In this project, the task of predicting" apply" behaviors of given users can be formulated as a recommendation problem. Such formulation naturally gives rise to the matrix completion techniques. The optimization subsection gives detailed illustration of the regular matrix completion problem and the recently emerging inductive matrix completion problem. Besides, common statistics for model evaluation in recommendation problem are given in the assessment section.

2.1 Optimization

We will start from the classic (binary) matrix completion problem. What follows is the one-class matrix completion problem, arised from practical needs. Then we will transit to the fashioned inductive matrix completion, which elegantly incorporate the features of users and jobs and address cold startup problem in the meanwhile.

2.1.1 Classic Matrix Completion

The latent factor model has an underlying assumption: the real exact rating matrix are low-rank matrix. That is, the derived low-rank matrix can exactly recover the real rating matrix. By modelling target rating variable as

$$R_{ij} = U_i^T V_j \tag{1}$$

Then we are able to employ least square cost to approximate the presumed low-rank matrix R to the real rating matrix A:

$$\min_{U,V} \quad \sum_{(i,j)} (A_{ij} - U_i^T V_j)^2 + \frac{\lambda}{2}(||U||_F^2 + ||V||_F^2) \tag{2}$$

where $||\cdot||_F$ denotes Frobenius norm, $A_{i,j}$ is the observed ground-truth entry and Ω is the index set of entries that are observed as positive $+1$ or negative 0.

2.1.2 One-Class Matrix Completion

$$\min_{U,V} \quad \sum_{(i,j)\in\Omega} (1 - U_i^T V_j)^2 + \frac{\lambda}{2}(||U||_F^2 + ||V||_F^2) \tag{3}$$

2.1.3 Inductive Matrix Completion

Formulate the problem as that of recovering a low-rank matrix W_* using observed entries $R_{ij} = x_i^T W_* y_j$ and the user/job feature vectors x_i, y_j. By factoring $W = UV^T$, we see that this scheme

constitutes a bi-linear prediction $(x^T U_*)(V_* y)$ for a new user/job pair (x, y).

$$\min_{U,V} \quad \sum_{(i,j)} (A_{ij} - x_i^T U V^T y_j)^2 + \frac{\lambda}{2}(||U||_F^2 + ||V||_F^2) \tag{4}$$

According to [1], under standard set of assumptions, the alternating minimization provably converges at a linear rate to the global optimum of two low-rank estimation problems: a) RIP measurements based general low-rank matrix sensing, and b) low-rank matrix completion. A more recent paper [2] in bioinformatics demonstrated successful application of such inductive matrix completion framework on gene-disease analytics.

2.1.4 One-Class Inductive Matrix Completion

$$\min_{U,V} \quad \sum_{(i,j)\in\Omega} (1 - x_i^T U V^T y_j)^2 + \frac{\lambda}{2}(||U||_F^2 + ||V||_F^2) \tag{5}$$

2.1.5 One-Class Inductive Matrix Completion with Biases

$$\min_{U,V} \quad (1-\alpha) \sum_{A_{ij}=1} (1 - x_i^T U V^T y_j)^2 + \alpha \sum_{A_{ij}=0} (0 - x_i^T U V^T y_j)^2 + \frac{\lambda}{2}(||U||_F^2 + ||V||_F^2) \tag{6}$$

2.2 Assessments

In this research project, we employ Recall-Precision curve and MAP@N curve to asess and visually present the accurate performance of a particular model.

2.2.1 Recall-Precision Curve

In pattern recognition and information retrieval with binary classification, precision, a.k.a. positive predictive value, is the fraction of retrieved instances that are relevant, while recall, a.k.a. sensitivity, is the fraction of relevant instances that are retrieved. According to the general definition, the precision and recall for one query in this recommendation problem can be formulated as

$$\text{precision} = \frac{\text{number of jobs one user should appply and also recommended}}{\text{number of jobs recommended to the user}} \tag{7}$$

$$\text{recall} = \frac{\text{number of jobs one user should appply and also recommended}}{\text{number of jobs one user should apply}} \tag{8}$$

Nevertheless, precision and recall are single-value metrics that take no ordering information into account. Thus, the recall-precision curve cannot assess performance of a model from the "order-does-matter" perspective.

2.2.2 MAP@N Curve

For job recommendation system, it is supposed to return a ranked sequence of suitable jobs for users to apply. Therefore, it is desirable to also consider the order in which the returned jobs are presented. The mean average precision (MAP) does vary in terms of the order of returned jobs and thus cumulative MAP@N curve can serve as an effective tool to visualize the quality of queried recommendations for both ordering and accuracy.

The prerequisite of computing mean average precision is to figure out average precision (AP) for each queried user. The average precision at certain position N is defined as

$$\text{AP@N} = \sum_{i=1}^{N} \text{precision}(i) \cdot \Delta\text{recall}(i) \tag{9}$$

Taking the mean of average precision at the corresponding position for all queried users derives the mean average precision .

$$\text{MAP@N} = \frac{1}{|\Psi|} \sum_{u \in \Psi} \text{AP}_u\text{@N} \tag{10}$$

where Ψ indicate the collection of queried users and $|\cdot|$ returns the cardinality of a set.

3 Implementation

3.1 Algorithm

The algorithmic description for *One-class Inductive Matrix Completion with biased weight* are shown in Algorithm 1.

Algorithm 1 Alternating Minimization for One-Class Inductive Matrix Completion with Biases

INPUT:
a) sparse matrices X and Y that denote features of users and jobs respectively.
b) matrix A that denotes partial observation of "apply" association with observed index set Ω

Initialize $U_{(0)}$ and V_0 by uniform randomization
Do
$V_{(k+1)}$ = argmin $(1-\alpha)\sum_{(i,j)\in\Omega}(1-\boldsymbol{x}_i^T U_{(k)} V_{(k)}^T \boldsymbol{y}_j)^2 + \alpha \sum_{(i,j)\notin\Omega}(0-\boldsymbol{x}_i^T U_{(k)} V_{(k)}^T \boldsymbol{y}_j)^2$
$U_{(k+1)}$ = argmin $(1-\alpha)\sum_{(i,j)\in\Omega}(1-\boldsymbol{x}_i^T U_{(k)} V_{(k+1)}^T \boldsymbol{y}_j)^2 + \alpha \sum_{(i,j)\notin\Omega}(0-\boldsymbol{x}_i^T U_{(k)} V_{(k+1)}^T \boldsymbol{y}_j)^2$
Until Convergence.

OUTPUT:
a) Model Parameter U_* and V_*

Algorithm 2 Newton Conjugate Descent Subroutine

INPUT:
a)
b)

Initialize $U_{(0)}$ and V_0 by uniform randomization

OUTPUT:
a) Model Parameter U_* and V_*

Algorithm 3 Prediction

INPUT:
a) sparse matrices X and Y denote features of users and jobs.
b) matrix A denotes partial observation of application association with observed index set Ω

Initialize $U_{(0)}$ and V_0 by uniform randomization

OUTPUT:
a) Model Parameter U_* and V_*

3.2 Practical Issues

Bias. Bias parameter $\alpha \in (0, 1)$ is the weight to balance optimization objective between observed and missing labels. Note that we can simply set $\alpha = 1$ to solve unbiased version of one-class inductive matrix completion.

Algorithm 4 Preclustering

INPUT:
  a) sparse matrices X and Y that denote features of users and jobs respectively.
  b) matrix A that denotes partial observation of "apply" association with observed index set Ω

Initialize $U_{(0)}$ and V_0 by uniform randomization
Do
  $V_{(k+1)}$ = argmin $(1-\alpha)\sum_{(i,j)\in\Omega}(1-\boldsymbol{x}_i^T U_{(k)} V_{(k)}^T \boldsymbol{y}_j)^2 + \alpha\sum_{(i,j)\notin\Omega}(0-\boldsymbol{x}_i^T U_{(k)} V_{(k)}^T \boldsymbol{y}_j)^2$
  $U_{(k+1)}$ = argmin $(1-\alpha)\sum_{(i,j)\in\Omega}(1-\boldsymbol{x}_i^T U_{(k)} V_{(k+1)}^T \boldsymbol{y}_j)^2 + \alpha\sum_{(i,j)\notin\Omega}(0-\boldsymbol{x}_i^T U_{(k)} V_{(k+1)}^T \boldsymbol{y}_j)^2$
Until Convergence.

OUTPUT:
  a) Model Parameter U_* and V_*

Initialization. Randomizing all components of $U_{(0)}$ and $V_{(0)}$ to uniform distribution over $(0, 1)$ works well in our experimentation.

Optimization. During each stage of alternating minimization, standarded conjugate gradient method is employed to solve least square problem.

4 Experiment I: Feature Extraction

4.1 Dataset

The dataset utilized in this experimental project comes from a Job Recommendation Challenge posted on Kaggle.com. The provider of this dataset is CareerBuilder.com, one of the biggest job recommendation service providers. This particular dataset, sized of several Gigabytes, contains 389,708 featured users, 1,092,096 characterized jobs and 1,603,111 application records that are divided into training and testing split. In this section, we will at first provide the fundamental introduction to these three basic elements: User, Job, Application. And then explanation are illustrated about temporal separation (concept of window) and designed distribution for user, job and application.

As the most essential component of job recommendation system, users are recorded by UserID, WindowID, Split, City, State, Country, ZipCode, DegreeType, Major, GraduationDate, WorkHistoryCount, TotalYearsExperience, CurrentlyEmployed, ManagedOthers, ManagedHowMany. *UserID* refers to the nubmering index of that indicated user. *WindowsID* is about the timing period in which this particular applicaiton about user happened. *City, State, Country, and ZipCode* are related to the living place of that user. What follows are the achievements in school. *DegreeType* shows the highest degree that user has gained from school, *Major* presents the field of his/her speciality and *GraduationDate* reveals when he/she gained the highest degree. Working and management experience comes next. *WorkHistoryCount* represents how many previous jobs one has had and *TotalYearsExperience* represents how many years one has been involved in occupation. *CurrentlyEmployed* and *ManagedOthers* are both binary values, indicating whether one was currently on his/her job and whether he has been in certain management position. *ManagedHowMany* implies his management power and capability, that is, the maximum number of people he/she has managed before.

When it comes to information of every individual job, fields like JobID, WindowID, Title, Description, Requirements, City, State, Country, Zip5, StartDate, EndDate are provided. *JobID* is the identifying number for each particular job. *WindowID* captures the same semantics as it is in a user record – involved timing period of one job. *Title* is the name of position sepcified by corresponding corporation, which can be significantly important since it reflects relative position and power in one company's hierarchy. Description and Requirements are both textual characterization over each particular job. *Description* provides a characteristic overview of one particular job. *Requirements* can be viewed the basic expectation of hiring company towards the job applicants. Similarly to the record of each individual user, every job also has location information, such as *City, State, Country, ZipCode*. Besides, *StartDate* and *EndDate* shows the timing information of one job, i.e. on which day it starts and ends.

As to each instanced record of application, the dataset contains information about the UserID, WindowID, Split, ApplicationDate, and JobID. *UserID* is indexing number of the user who applied for particular job, indexed by *JobID*. *WindowID* implies the timing period of that application event, while *ApplicationDate* presents the specific date of application event. And *Split* labelled in which division (training or testing set) that piece of application was placed.

Next presented was the overall explanation about timing design of dataset. In outline, all application instances span 13 weeks. All the job applications are split into 7 groups, with each group representing a 13-day window. Each 13-day window is split into two parts: The first 9 days are the training period, and the last 4 days are the test period. The graphical representation demonstrating such splits is illustrated below.

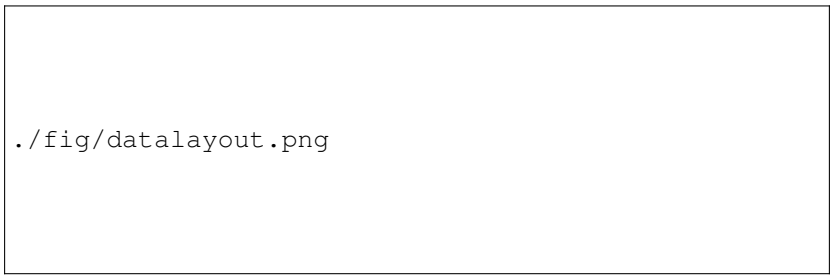

Figure 1: Illustrating Diagram for Temporal Separation of the Dataset

Note that each user and each job posting is randomly assigned to exactly one window. Each job is assigned to a window with probability proportional to the time it was live on the site in that window. Each user is assigned to a window with probabilty proportional to the number of applications they made to jobs in that window. As shown in the figure below, User1 only made submissions to jobs in Window 1, and so was assigned to Window 1 with probability 100%. User2, however, made submissions to jobs in both Window 1 and Window 2, and so may have been assigned to either Window1 or Window2.

Figure 2: Explanation for user distribution

In each window, all the job applications that users in that window made to jobs in that window during the 9-day training period. And with each window, users have been split into two groups, *Test group* and *Train group*. The Test users are those who made 5 or more applications in the 4-day test period, and the Train users are those who did not.

For each window, the task of prediction is which jobs in that window the Test users applied for during the window's test period. Note that users may have applied to jobs from other windows as well, but the only thing needed to be predicted is which jobs they applied to in their own windows.

4.2 Preprocessing

Preprocessing, as the earliest step for any machine learning experimentation, is extremely important in that it, as a role of filter, process the raw data and then generates what is later employed by human-design AI system. From this perspective, any inconsistent or invalid processing would cause devastating effects for the entire system.

Instance Subsetting. Since there are huge quantities of users and jobs, our experiment only exploits the first part of dataset (WindowsID = 1) for computational convenience. Another benefit for such instance subsetting is to temporarily put aside the temporal dynamics, that is, disregard the impact of timing variable for now. And the investigation over how to accommodate the designed system to the entire dataset can be deferred to later study.

Binary Encoding Scheme. For all nominal attributes, we represent every of them as a collection of binary variables. For example, DegreeType, indicating the highest degree obtained by one user, has five possible values: None, High School, Bachelor, Master, PhD. We have five binary variables indicating if one user has highest degree on None, High School, Bachelor, Master, PhD respectively. Other examples of such nominal attributes in this application prediction problem are Zip5, City, State and Country. One advantage of such processing lies in multi-valued features, like Major. Some people may be double-major or even three-major. In these cases, no conflict would occur if we apply take binary encoding scheme. However, the binary encoding scheme will significantly augment the scale of learning task since some nominal attributes have huge domain dimensionality. Hence, it is unavoidable to leave out redundant attributes.

Attributes Subsetting. To simplify initial setup of this machine learning experimentation, we manually left out some three categories of attributes. First category is semantically meaningless information. On top of that, those attributes that may contribute to application prediction (slightly higher accuracy) but overwhelmingly enlarge the scale of learning (significantly lower speed) are also eliminated for now. Zip5 is the most illustrative example for this type of attribute. Another set of attributes are disregarded because we are incapable of modelling these relevant factors. For example, temporal dynamics (i.e. *StartDate and EndDate* of job) are not planned to be captured in our current design.

Efficient Representation of Textual Data. The ways to process textual data are important because we hope to narrow the scale of numerical representation for these texts. The sequential procedures of processing raw text data are specified as follows:

(i) Convert all terms into lowercase.

(ii) Eliminate stop words.

(iii) Annihilate all meaningless terms with particular forms, i.e. timestamps, datestamps, IP addresses, phone numbers and webpage link

(iv) Remove all punctuations.

(v) Check validity of words.

(vi) Acquires linguistic stemmes for each tokens. This step also removes plurals and tenses.

As a consequence of applying above transformations, **12841** keywords (distinct vocabularies) are obtained.

Preprocessing Products. Through preprocessing, we generate three data files for later machine learning algorithm: *win1_Users.sparse, win1_Jobs.sparse and win1_Apps.sparse*. Within *win1_Users.sparse*, there are 77060 users with 1416 features per user. *win1_Jobs.sparse* maintained records of 285515 jobs, each of which has 12841 features for now. *win1_Apps.sparse* contains a sparse matrix with dimensionality 77060×285515.

Storage Format Note that all numerics generated by preprocessing procedure are restored as libSVM format in that the generated features for user and job have huge extent of sparsity. Specifically, for every job or user contained in win1_Users.sparse, win1_Jobs.sparse, the representation of each object is as follows:

feature_index:feature_value feature_index:feature_value ....

On the other hand, the representation for each application instance is slightly different, but still in standarded LibSVM format.

apply_or_not 0:user_index 1:job_index

5 Experiment II: Basic Parameters

5.1 Effects of Rank K

We trace the main term in training stage.

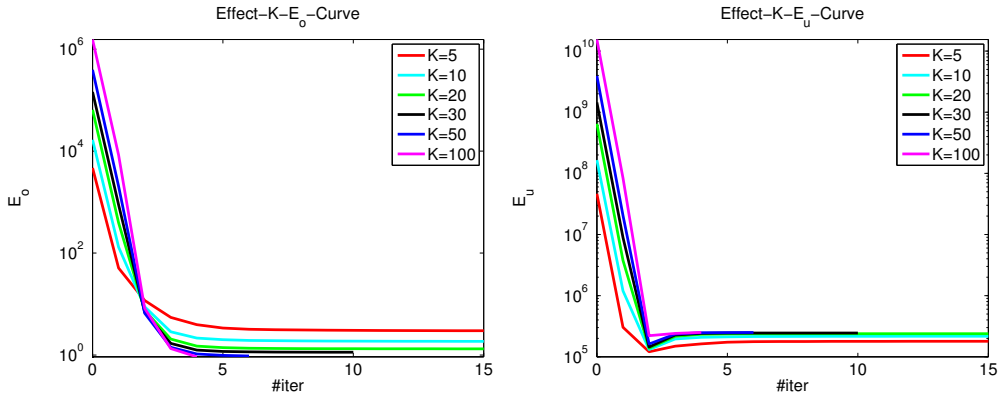

Figure 3: Trace E_o and E_u between the trained models with various $K = 5, 10, 20, 30, 50$. Left: E_o Curve. Right: E_u Curve.

The performance evaluation is as follows.

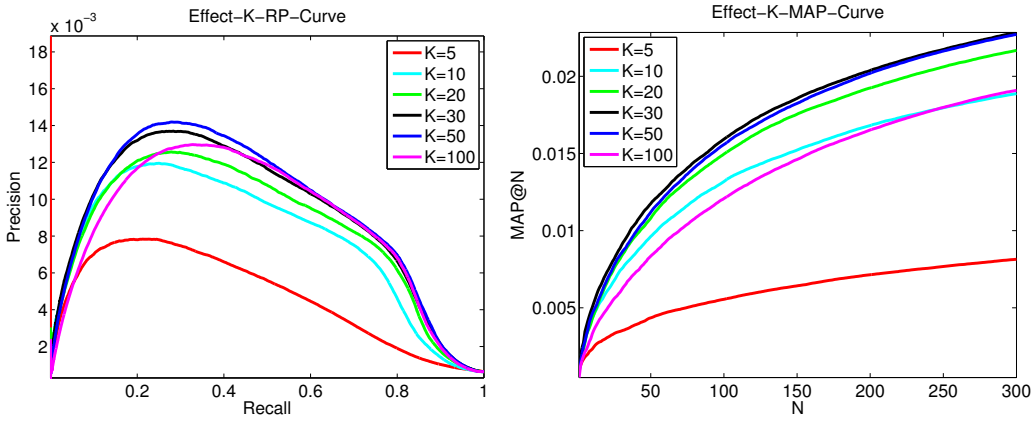

Figure 4: Performance Comparison between the trained models with various $K = 5, 10, 20, 30, 50$. Left: Precision-Recall Curve. Right: MAP@N Curve.

5.2 Effects of Bias Weight α

5.3 Effects of Regularization Parameter λ

5.4 Effects of Early Stopping

6 Experiment III: Revisits Feature Engineering

6.1 Best-Tuned Setting

6.2 Principal Component analysis

7 Experiment IV: Pre-clustered Matrix Completion

7.1 Results

Figure 5: Precision v.s. Recall Comparison between models with $\lambda = \lambda_0, \lambda_1, \lambda_2, \lambda_3$

Figure 6: Precision v.s. Recall Comparison between various models with different preprocessed data

8 Conclusions

Features do provide supervision for recognizing the pattern of new users.

Future work can be extended to modeling time-series recommendation problem by means of one-class inductive matrix completion.

References

[1] Prateek Jain and Inderjit S Dhillon. Provable inductive matrix completion. *arXiv preprint arXiv:1306.0626*, 2013. 2

[2] Nagarajan Natarajan and Inderjit S Dhillon. Inductive matrix completion for predicting gene–disease associations. *Bioinformatics*, 30(12):i60–i68, 2014. 2