Project Report: User-Job Suitability Measurement

XIN LIN
Department of Computer Science
University of Texas at Austin
Austin, TX 78705
jimmylin@utexas.edu

Abstract

The essential part of job recommendation system is to evaluate suitability between job seekers and available positions. However, currently popular job search engines or job recommender systems online are weakly capable of providing recommendation in intelligent ways. This project formulates suitability evaluation problem as an complicated system that prototyped from application prediction and then transfer its power to suitability evaluation. For the the naive prototype, we apply recently emerged Inductive Matrix Completion on task of predicting job seekers' application behavior. This new approach significantly outshines other classic treatments. Our discussion section at the end aims at giving several possible directions of system extension in the future.

1 Introduction

Job recommender system aims at helping people figure out a list of suitable jobs and provide corresponding sugguestions if possible. However, exactly modelling the suitability between users and jobs are challenging.

Due to the limitation of datasets, we experiment on application prediction for initial setup and then find ways to reduce the problem to suitability evaluation.

2 Problem Formulation

If we assume that all job seekers are extremely knowledgeable (understand clearly and completely the profile and requirement of every job) and rational (never apply for the unsuitable jobs), we can directly makes use of the score obtained in the application prediction. However, such assumption receives little support from practical analysis, in the sense that people tend to apply for the job positions with higher salaries and correspongly much more capability seeking.

2.1 Standarded Matrix Completion

For startup, we first focus on binary suitability measurement problem. That is, we only investigate the binary association, suitable (1) or not suitable (0), for each pair of user and job. At first glance, this problem can be naively solved by traditional binary classifiers, e.g. logistic regression and support vector machine, or one-class learning solvers (or data description techniques). Nevertheless, more carefull examination reveals obvious disadvantage of such treatment. The main drawback of using traditional binary classifier is that.

2.1.1 Content-based Filtering

2.1.2 Collaborative Filtering

The developing motivation of *Collaborative Filtering* is that the profiles of items or users are not always available or poorly collect in some settings but instead, the past histories of user-item association, also called *explicit feedback*, are easy to obtain. Collaborative filtering put its focus directly on the target (user-job), inferring missing ones from observed one, rather than relationships between target variables and features. Nearest neighbour method and latent factor model are two major approaches for Collaborative Filtering.

The latent factor model has an underlying assumption: the real exact rating matrix are low-rank matrix. By modelling target rating variable as

$$R_{ij} = u_i^T v_j \tag{1}$$

we are able to use least square cost to guide the optimization procedure:

$$min \sum_{i,j} ||r_{ij} - u_i^T v_j||^2 \tag{2}$$

Note that $U \in R^{n \times k}$ and $V \in R^{m \times k}$ are respective matrices restoring hidden-topic measurements for users and items(jobs). Intuitively, k denotes the number of latent topics, with its quantity reflecting shared interest we want to measure over user and job entity. Also, regularized squared error is typically used, in order to avoid overfitting of this model. Hence, 2 can be further extended to be

$$min \sum_{i,j} ||r_{ij} - u_i^T v_j||^2 + \lambda(||u_i||^2 + ||v_j||^2) \tag{3}$$

One approach to get numerical solution is through *Stochastic Gradient Descent*, where each update take into account only one observed instance. The update rules with γ step size are presented as follows:

$$v_j \leftarrow v_j + \gamma\big((r_{ij} - u_i^T v_j) \cdot u_i + \lambda v_j\big) \tag{4}$$

$$u_i \leftarrow u_i + \gamma\big((r_{ij} - u_i^T v_j) \cdot v_j + \lambda u_i\big) \tag{5}$$

It can be easily figured out that equation 3 is not convex. But if we fix one variable (U or V), the remained problem becomes convex optimization and hence exact solution can be found. One alternative technique to solve 3, known as *Alternating Least Square*, takes advantage of this idea and turns out to be the one with both computational convenience and convergence to decent local minima.

In netflix prize, the latent factor model, combined with temporal dynamics and bias evaluation, winned as a single model with the greatest prediction power. Despite of that, collaborative filtering also has its limitation. It fails to provide supervision of missing values when only rare amount of previous association histories are provided, especially when the systems are on its early stage. This is commonly referred to as *cold startup* problem.

2.2 One-Class Matrix Completion

2.3 Inductive Matrix Completion

Comparing to collaborative filtering , the conventional approach for matrix completion, advantages of incorporating features are obvious in two aspects. One is lower sample complexity while predicting missing values for known users and jobs. The other is that precise prediction can be produced for new/unkonw users and jobs.

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).

One practical solution is to use trace-norm constraint as a proxy for the rank constraint and then solve the resulting non-smooth convex optimization problem. Notwithstanding, the resulting convex

optimization methods require computation of full SVD for matrices with potentially large rank and hence do not satisfy scalability to large problems.

Parameterization over W as $W = UV^T$ and then alternatingly optimize for U and V is also a considerable approach in practice. To our best knowledge, alternating minimization and its variants need to solve only least squares problems and hence are scalable in practice but might get stuck in a local minima.

A recent paper [1] provides theoretical supports for a variant of *alternating minimization* method to solve *Inductive Matrix Completion* problem, in which other than a small number of observations from the suitability matrix, the feature vectors for users and jobs are also available. According to [1], under standard set of assumptions, 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.

One possible enhancement for above inductive matrix completion is to extend our consideration from the linear association between features and latent factors to an version that accepts non-linear association. By this intuition, we can name this approach as *Kernel-based Inductive Matrix Completion*. By taking into account non-linear relations between designed features and hidden topics (shared latent factors), the space of latent factors can be largely expanded and then it would be more likely to automatically detect latent factors with higher quality.

2.4 Suitability Measurement with Prerequisites

1. simulate course recommendaiton (by adita)

3 Experiments: Application Prediction

Due to limitation of acquired data, our first experiment is oriented to problem of application prediction. Specifically, given a set of featured users and featured jobs, the designed system should predict whether one user will apply for one particular job.

3.1 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.

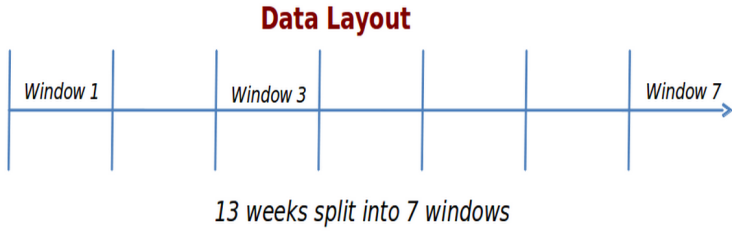

Figure 1: Illustrating Diagram for Temporal Separation of the Dataset

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.

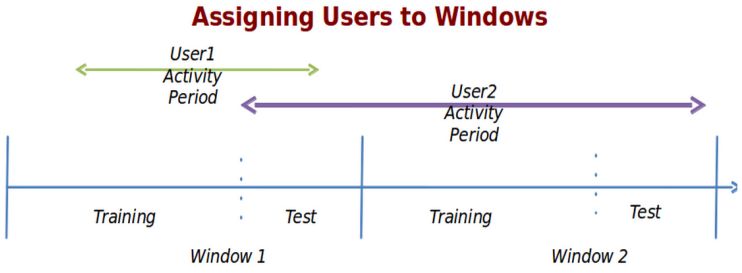

Figure 2: Explanation for user distribution

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.

3.2 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

3.3 Failure of Traditional Binary Classifiers

As we can observe from the dataset, all "not apply" behaviors of users are missing and hence the dataset cannot supervise the learning of "not apply" behavior of users. It is easy to conclude that all traditional binary classifiers, including logistic regression and classic support vector machine, fail to work in this setting.

And more intuitive failure occurs to nearest neighours model. Typically to predict application of one user and one job, we can look up the k most similar users or k most similar jobs in terms of their features. Nevertheless, under this one-class setting, no neibours of negative class can provide a negative support on the majority voting.

3.4 Standarded One-Class Matrix Completion

Due to the observation that only behaviors of "apply" are recorded in the dataset, one-class problem, where only instances of positive classes are supervised in the training, naturally come to our mind.

3.4.1 One-class Matrix Completion

3.4.2 One-class Matrix Completion with Bias

3.4.3 Weakness

3.5 Inductive Matrix Completion

As previously mentioned, one should formulate it as *inductive matrix completion* when side information of users and items (in this case, jobs) are available. Even through simple alternating minimization method, one should be able to recover back the exact underlying low-rank matrix and predict inductively on new users and jobs if provided conditions are satisfied by the set of measurements [1]. And intuitively, it is not exaggerated to claim that such formulation and corresponding solver (AltMin) take advantage of both classic approaches – content-based filtering and collaborative filtering in recommendation problem: less sample complexity and inductivity on new users/jobs.

3.5.1 Algorithmic Description

The algorithmic procedures are shown in Algorithm 1.

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

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.

Algorithm 1 Alternating Minimization for Least-Square Inductive Matrix Completion

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
Do
  $V_{(k+1)}$ = argmin $\alpha \sum_{(i,j)\in\Omega}(\boldsymbol{x}_i^T U_{(k)} V_{(k)}^T \boldsymbol{y}_j - 1)^2 + (1-\alpha)\sum_{(i,j)\notin\Omega}(\boldsymbol{x}_i^T U_{(k)} V_{(k)}^T \boldsymbol{y}_j - 0)^2$
  $U_{(k+1)}$ = argmin $\alpha \sum_{(i,j)\in\Omega}(\boldsymbol{x}_i^T U_{(k)} V_{(k+1)}^T \boldsymbol{y}_j - 1)^2 + (1-\alpha)\sum_{(i,j)\notin\Omega}(\boldsymbol{x}_i^T U_{(k)} V_{(k+1)}^T \boldsymbol{y}_j - 0)^2$
Until Convergence.
Predict values for missing entries: $\forall (i,j) \notin \Omega$, $R_{ij} = \boldsymbol{x}_i^T U_* V_*^T \boldsymbol{y}_j$

OUTPUT:
  a) Model Parameter U_* and V_*

3.5.2 Inductive One-Class Matrix Completion

3.5.3 Inductive One-Class Matrix Completion with Bias

3.5.4 Results

3.6 Conclusions

4 Experiments: Suitability Evaluation

We now start to divert our focus on investigating Application Potential to Suitability Evaluation.

5 Discussions

References

[1] Prateek Jain and Inderjit S Dhillon. Provable inductive matrix completion. *arXiv preprint arXiv:1306.0626*, 2013. 3, 6

[2] Nagarajan Natarajan and Inderjit S Dhillon. Inductive matrix completion for predicting gene–disease associations. *Bioinformatics*, 30(12):i60–i68, 2014. 3