Predicting Yelp User Ratings by Soft-Impute Matrix Completion

William Weimin Yoo

1 Introduction

The Yelp dataset consists of customer reviews of local businesses from the greater Phoenix, Arizona metropolitan area. The challenge is to come up with innovative ways to analyze and "mine" these data for interesting trends, and explore customer-business relationships. There are no specific goals and contestants are free to formulate research questions based on these data, where some possible questions are hinted in the dataset challenge website. In this paper, we will consider the problem of predicting potential user ratings of businesses based on observed ratings. A solution to this problem would enable personalized recommendation of businesses, and serve as a basis to design accurate recommender systems. This is reminiscent to the famous Netflix prize where rating predictions are made for movies.

2 Yelp Customer Reviews Dataset

There are four datasets in JavaScript Object Notation (json) format, conveniently named as business, checkin, review and user datasets. We used the statistical software package `R` for data preprocessing. Our first aim is to convert these json data into `R` data frames for easier viewing. For parsing json syntax, we used the `fromJSON` function of the `RJSONIO` package. This function will convert json to `R` list object. We then collect the converted list objects and coerced them into a data frame.

The business data frame consists of a directory listing of 11,537 local businesses in the greater Phoenix, Arizona metropolitan area. It contains basic information of businesses such as address, business categories, average business ratings and others. The checkin data frame consists of check-in counts by mobile devices at every hour of the day for a week. The user data frame consists of Yelp user information such as last names and user average ratings. We will mainly concentrate on the review data frame, which contains vital information for our problem. There are eight columns, the first is votes, which contains triplets of different vote counts made by other users for useful, cool, and funny reviews respectively. The second is encrypted user id's containing string of alphanumeric and special characters, the third is review id's with the same encryption scheme. The fourth is star ratings, with the scale of one to five, where higher value indicates more preferable. The fifth is date of review/rating stored in the format `yyyy-mm-dd`. The sixth column is user review text. The next column is an indicator denoting the type of dataset, which in this case is coded as `review`. The last column is business id's with the same encryption scheme used for user id's.

3 Data Preprocessing

At this stage, we were planning to use matrix completion techniques to do rating prediction. This then entails us to construct a matrix with users in the rows and businesses in the columns with entries populated by star ratings. Hence, we extracted the user id, star and business id columns into a new matrix (call this entry matrix) for this purpose. We first sort the business column in ascending order and map each unique alphanumeric encryption to a natural number. Utilizing this sorted matrix, we proceed to sort the user column in ascending order with the same mapping. Thus, each row of this entry matrix is a triplet containing the corresponding row and column number for the actual user-business matrix with the associated star rating value. We then used this entry matrix to construct the user-business matrix.

However, the construction is very time consuming and we received error saying that the maximum memory allocation (default is 3,892 MB for 64 bit `R`) is reached. We tried to increase the memory limit using `memory.limit` which partially alleviate the problem in a 64 bit Linux machine. The resulting user-business matrix has almost 2 GB of memory size. We found it very hard to work with such a large object in `R` and have to find alternatives. Closer inspection will reveal that there are 45,981 users, 11,537 businesses, and 229,907 star ratings. This implies that the user-business matrix will have around 530 million entries and only 0.04% are observed. Hence, this matrix is very sparse. After some discussion with

Professor Hua Zhou, we decided to implement two major changes. The first is to reduce the data size and the second is to do all computation in MATLAB.

Concerning the first, we reasoned that users or businesses with too few ratings will not give good predictions regardless of methods used. Hence, before creation of the entry matrix as detailed previously, we decided to exclude users who rated 20 or less businesses and also exclude businesses receiving 10 or less user ratings. These numbers were chosen empirically to increase the observed entries percentage from 0.04% to over 1%. After exclusion, we have 1,694 users, 4,025 businesses and 85,705 star ratings. Hence, the reduced user-business matrix has around 6.8 million entries with 1.26% observed entries.

Concerning the second, MATLAB has `sparse` function that will create sparse class object for sparse data. Using this class structure, MATLAB will store only the locations and values for nonzero entries, and hence significantly reducing memory requirement. Moreover, the sparse class object can be manipulated in the same manner as a dense matrix, for example we can perform matrix arithmetic and singular value decomposition on this class. However, instead of "porting" the preprocessing steps from `R` to MATLAB, we save the constructed entry matrix to a ASCII text file, which is the needed arguments for `sparse`, and import this entry matrix into MATLAB for actual matrix completion computations.

4 Matrix Completion using Spectral Regularization

By passing the entry matrix as an argument for `sparse` in MATLAB, we constructed the user-business matrix, which we will denote it as $\mathbf{X}$ with generic dimensions n (users) and m (businesses). In our case, n is 1694 and m is 4025. We note that the unobserved ratings were set to zero automatically. Our aim is now to predict these unobserved ratings given the relatively few observed ones. To this end, let us assume that $\mathbf{X}$ is generated from a latent matrix $\mathbf{Z}$ contaminated by noise through

$$\mathbf{X} = \mathbf{Z} + \boldsymbol{\varepsilon} \tag{1}$$

where $\boldsymbol{\varepsilon}$ is the mean zero error matrix. Our problem then boils down to learning $\mathbf{Z}$. Without any structural assumption on $\mathbf{Z}$, this is theoretically impossible since the missing values could be any rating between one to five. It is reasonable to assume that businesses can be grouped into different categories, and each user rates businesses in the same category similarly than between categories. Intuitively, this implies that ratings received by businesses are driven by a smaller set of factors based on these categories. Therefore, let us assume that $\mathbf{Z}$ lives in a lower dimensional manifold, and it can be represented as a lower rank matrix, i.e. $\mathbf{Z} \approx \mathbf{A}_{n\times r}\mathbf{B}_{r\times m}$, where the rank r is much smaller than $\min\{n, m\}$. Since $\mathbf{Z}_{ij} \approx \sum_{k=1}^{r} A_{ik}B_{kj}$, we can interpret A_{ik} as the affinity of user i to business category k and B_{kj} is the relative star rating received for business j in category k.

The model in (1) and the lower dimension representation lead us to consider the following optimization problem to learn $\mathbf{Z}$,

$$\begin{aligned} \text{minimize} \quad & \operatorname{rank}(\mathbf{Z}) \\ \text{subject to} \quad & \sum_{(i,j)\in\Omega} (X_{ij} - Z_{ij})^2 \leq \delta, \end{aligned} \tag{2}$$

where $\Omega \subset \{1, \ldots, n\} \times \{1, \ldots, m\}$, i.e. the first two columns in the entry matrix, and $\delta \geq 0$ is the regularization parameter. However, this problem is not convex and is NP hard due to the rank constraint with integer output. Now, $\operatorname{rank}(\mathbf{Z}) = \sum_i \mathbb{1}(\sigma_i(\mathbf{Z}) \neq 0)$ where $\sigma_i(\mathbf{Z})$ is the ith singular value of $\mathbf{Z}$ and $\mathbb{1}(\cdot)$ is the indicator function. Hence, this problem is analogous to ℓ_0 regression. In the latter context, the lasso with ℓ_1 penalty can be viewed as a convex relaxation of ℓ_0 regression. Adapting this idea to our problem, we solve the corresponding convex relaxation by replacing the rank constraint with the nuclear norm of $\mathbf{Z}$, which is the sum of the (absolute) singular values of $\mathbf{Z}$, to obtain:

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{Z}\|_* \\ \text{subject to} \quad & \sum_{(i,j)\in\Omega} (X_{ij} - Z_{ij})^2 \leq \delta, \end{aligned} \tag{3}$$

with $\|\cdot\|_*$ denoting the nuclear norm. To write (3) compactly, let P_Ω be a projection operator acting on a $n \times m$ matrix $\mathbf{Y}$ such that

$$P_\Omega(\mathbf{Y})_{ij} = \begin{cases} Y_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{if } (i,j) \notin \Omega, \end{cases} \tag{4}$$

i.e., P_Ω projects $\mathbf{Y}$ onto the observed entries with indices in Ω. Also, define the complementary projection $P_\Omega^\perp$ as $P_\Omega^\perp(\mathbf{Y}) = \mathbf{Y} - P_\Omega(\mathbf{Y})$. Using these operators, we can write the Lagrangian of (3) as,

$$\arg\min_{\mathbf{Z}} \frac{1}{2}\|P_\Omega(\mathbf{X}) - P_\Omega(\mathbf{Z})\|_F^2 + \lambda\|\mathbf{Z}\|_*, \tag{5}$$

where $\|\cdot\|_F$ is the Frobenius norm and $\lambda \geq 0$ is the penalty (Lagrangian) parameter. We note that by nature of the `sparse` function in MATLAB, $P_\Omega(\mathbf{X}) = \mathbf{X}$. There are two possible methods to solve this Lagrangian for $\mathbf{Z}$ at a given λ. The first is direct analytical solution (which is possible), and the second is through the use of Majorization-Minimization (MM) procedure. For the first method, direct solution were derived in papers by Cai et al. (2010), Ma et al. (2011), and Mazumder et al. (2010). It turns out that both methods will lead to the same algorithm for solving this Lagrangian at a given λ.

5 Details of Implementation

Partition $\Omega = \Omega_{\text{train}} \cup \Omega_{\text{valid}} \cup \Omega_{\text{test}}$ using simple random sampling without replacement, with set cardinalities 75,705, 5,000 and 5,000 elements respectively. Set the training matrix $\mathbf{X}_{\text{train}} = P_{\Omega_{\text{train}}}(\mathbf{X})$, the validation matrix to be $\mathbf{X}_{\text{valid}} = P_{\Omega_{\text{valid}}}(\mathbf{X})$, and the test matrix to be $\mathbf{X}_{\text{test}} = P_{\Omega_{\text{test}}}(\mathbf{X})$. The training matrix is used to solve (5) for a given lambda, the validation matrix is for determining the optimal lambda while the test matrix is used to access the performance of the algorithm. Define $S_\lambda(\cdot)$ to be the soft-thresholding operator such that for any matrix $\mathbf{W}$ of rank r, we have $S_\lambda(\mathbf{W}) = \mathbf{U}\mathbf{D}_\lambda\mathbf{V}^T$, where $\mathbf{D}_\lambda = \text{diag}[(d_1-\lambda)_+, \ldots, (d_r-\lambda)_+]$. Here, $\mathbf{U}\mathbf{D}\mathbf{V}^T$ is the singular value decomposition (SVD) of $\mathbf{W}$ with $\mathbf{D} = \text{diag}[d_1 \ldots, d_r]$ and $v_+ = \max\{v, 0\}$. We then used the following algorithm by Mazumder et al. (2010) to solve (5) at a grid of values for λ using warm starts, and choose the optimal solution using relative mean square error (MSE) by cross-validating (CV) the validation matrix.

Algorithm: SOFT-IMPUTE

```
1. Initialize by setting $\mathbf{Z}^{\text{old}} = \mathbf{0}$.
2. For each $\lambda_1 > \lambda_2 > \ldots > \lambda_K$:
   (a) Repeat:
       i. Soft threshold $\mathbf{Z}^{\text{new}} \leftarrow S_{\lambda_k}(\mathbf{X}_{\text{train}} + P^{\perp}_{\Omega_{\text{train}}}(\mathbf{Z}^{\text{old}}))$
       ii. If $\frac{\|\mathbf{Z}^{\text{new}} - \mathbf{Z}^{\text{old}}\|_F^2}{\|\mathbf{Z}^{\text{old}}\|_F^2} < \epsilon$, then break and go to (b);
       iii. Else, assign $\mathbf{Z}^{\text{old}} \leftarrow \mathbf{Z}^{\text{new}}$ as initial value (warm start) and go back to i.
   (b) Assign $\widehat{\mathbf{Z}}_{\lambda_k} \leftarrow \mathbf{Z}^{\text{new}}$
   (c) Compute relative CV $\text{MSE}_{\lambda_k} = \frac{\|P_{\Omega_{\text{valid}}}(\widehat{\mathbf{Z}}_{\lambda_k}) - \mathbf{X}_{\text{valid}}\|_F^2}{\|\mathbf{X}_{\text{valid}}\|_F^2}$
3. Output sequence of solution $\widehat{\mathbf{Z}}_{\lambda_1}, \ldots, \widehat{\mathbf{Z}}_{\lambda_K}$ and $\text{MSE}_{\lambda_1}, \ldots, \text{MSE}_{\lambda_K}$
4. Find the solution with minimum relative CV MSE, denote this solution as $\widehat{\mathbf{Z}}_{\text{optimal}}$.
```

The most computation intensive part is step 2(a)i. when we do matrix soft-thresholding. As noted previously, this involves computing the SVD of $\mathbf{X}_{\text{train}} + P^{\perp}_{\Omega_{\text{train}}}(\mathbf{Z}^{\text{old}})$. Using the Golub-Kahan-Reinsch algorithm will take approximately $4n^2m + 8nm^2 + 9m^3$, which for our problem is around 852 Giga FLOPS. Since we are starting from a large lambda, it follows that only the largest few singular values will remain after thresholding, while the smaller ones will be shrunk to zero. Coupled with the fact that $\mathbf{X}_{\text{train}}$ is sparse and $P^{\perp}_{\Omega_{\text{train}}}(\mathbf{Z}^{\text{old}})$ is of low rank, a Lanczos type algorithm would be more efficient and accurate. For our problem, we use the Lanczos bidiagonalization with partial reorthogonalization (LBPR) algorithm proposed by Larsen (1998).

Given a (sparse) matrix, this algorithm computes the r largest singular values and vectors. It can be shown that computing step 2(a)i using this algorithm will take about $O((n+m)r^2) + O(|\Omega_{\text{train}}|r)$ flops, with $|\cdot|$ denoting set cardinality. Hence the algorithm complexity in linear in the dimensions of $\mathbf{X}_{\text{train}}$. In actual implementation, we need to have another loop for step 2(a)i, i.e. use LBPR to get maximum singular value and compare it with given λ, if the difference is negative or below some threshold (say 1.0^{-6}), stop and set the rest singular values to zero and go to step 2(a)ii; else, do LBPR again to get second largest singular value and do the same comparison until the smallest computed singular value is smaller then λ. The original LBPR was written by Larsen (1998) in FORTRAN and bundled into the PROPACK library, which can be obtained from `http://soi.stanford.edu/~rmunk/PROPACK/`. Moreover, the same author also provides a MATLAB version in the same webpage containing the associated `mex` files and binaries, with well documented help pages. Mazumder et al. (2010) wrote the original SOFT-IMPUTE algorithm in MATLAB by calling on routines from the MATLAB PROPACK `mex` binaries (`http://www-stat.stanford.edu/~rahulm/software.html`). We then modified and expanded on this SOFT-IMPUTE algorithm, and adapted the programme for the present dataset.

For step 2, the largest lambda λ_1 is chosen to be $0.9\sigma_{\max}(\mathbf{X}_{\text{train}})$, where $\sigma_{\max}(\mathbf{X}_{\text{train}})$ is the spectral radius (largest singular value) of $\mathbf{X}_{\text{train}}$. The smallest λ_K is set to $\lambda_1/100$. Also, $K = 20$ and the lambda values are equally spaced. The tolerance ϵ in step 2(a)ii. is set to 1.0^{-4}, and to ensure faster computation, we put an upper limit of 100 iterates for the repeat statement in step 2(a).

6 Results and Statistical Analyses

To access the performance of SOFT-IMPUTE and gauge the variability of the computed performance measures, we created 20 randomly partitioned Ω and applied SOFT-IMPUTE to each resulting training and validation matrices, i.e.

1. For $j = 1, \ldots, 20$:
   (a) Partition $\Omega = \Omega_{\text{train}} \cup \Omega_{\text{valid}} \cup \Omega_{\text{test}}$ randomly
   (b) Construct $\mathbf{X}_{\text{train}}$, $\mathbf{X}_{\text{valid}}$, $\mathbf{X}_{\text{test}}$
   (c) Apply SOFT-IMPUTE to $\mathbf{X}_{\text{train}}$ and choose optimal lambda using $\mathbf{X}_{\text{valid}}$ to get $\hat{\mathbf{Z}}_{\text{optimal}}$

(d) Compute standardized training error: trainERROR $= \frac{\|P_{\Omega_{\text{train}}}(\widehat{\mathbf{Z}}_{\text{optimal}})-\mathbf{X}_{\text{train}}\|_F^2}{\|\mathbf{X}_{\text{train}}\|_F^2}$

(e) Compute standardized test error: testERROR $= \frac{\|P_{\Omega_{\text{test}}}(\widehat{\mathbf{Z}}_{\text{optimal}})-\mathbf{X}_{\text{test}}\|_F^2}{\|\mathbf{X}_{\text{test}}\|_F^2}$

(f) Compute rank of $\widehat{\mathbf{Z}}_{\text{optimal}}$ by counting number of nonzero singular values (by-product of step 2(a)i. SOFT-IMPUTE)

We average the computed performance measures (test and training errors) over the 20 Monte Carlo (MC) replicates and also computed the corresponding MC standard errors. The standardized test error is 0.3395 and the training error is 0.0670 where the MC standard errors for both are on the order of 1.0^{-4}. The mean rank recovered is 323 with standard error 0.211. This suggests that the 1694×4025 high-dimensional user-business matrix can be represented as a 323-rank matrix after denoising. The entire 20 iterations was run on the teaching server and the average computational time for the SOFT-IMPUTE algorithm at each iterate is around 51.22 minutes.

7 Conclusion

As noted in the previous analysis, we found that the high dimensional 1694×4025 user-business ratings matrix can be compressed into a 323-rank feature matrix through the SOFT-IMPUTE algorithm. As we did not try to optimize the hyperparameters in this study, we believe that further compression to a lower rank is possible after fine-tuning these parameters. Further investigation of this feature matrix will yield insights regarding the drivers and mechanism behind user ratings and the recommender system.

On the test error front, we could improve performance by taking weighted linear combination of predictions from SOFT-IMPUTE and other methods. This idea was used extensively during the famous Netflix prize where contestants “blend” an ensemble of predictions from different algorithms to get the prediction with the lowest mean square error. Models used for the ensemble predictor included factor models, nearest neighbour regression, restricted Boltzmann machines and others (Bell and Koren, 2007).

We conclude by saying that analyzing big datasets such as this one requires specialize skills, and standard methods for textbook examples are not adequate in dealing with data of this magnitude. Hence, in this era of “Big Data”, it is imperative that we learn the right techniques, use the appropriate software and implement the correct algorithm when solving any problem at hand.

References

Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.

Jian-Feng Cai, Emmanuel J. Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

Rasmus Munk Larsen. Lanczos bidiagonalization with partial reorthogonalization. *DAIMI Report Series*, 27(537), 1998.

Shiqian Ma, Donald Goldfarb, and Lifeng Chen. Fixed point and bregman iterative methods for matrix rank minimization. *Mathematical Programming*, 128(1–2):321–353, 2011.

Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of machine learning research*, 11(Aug): 2287–2322, 2010.