Reddit Recommendation System

Daniel Poon, Yu Wu, David (Qifan) Zhang
CS229, Stanford University
December 11th, 2011

1. Introduction

Reddit is one of the most popular online social news websites with millions of registered users. A user can submit content (identified by "links"), and each of these can be up-voted or down-voted by others. Reddit does not yet have a recommendation system so the goal of this project is to develop one using machine learning concepts. Our model can also be used to rank multiple recommendations since each recommendation has a real-value score.

2. Data and Statistics

The initial data set, obtained from the Reddit team, had 23,091,688 votes, 43,976 users and 3,436,063 links (within 11,675 sub-reddits). This data-set represents 17% of all votes in 2008.

We reduced the amount of data used to improve efficiency. For instance, the number of links was too large so we eliminated all links which had less than 500 votes. The reduced data set has 2,294,532 votes, 24,504 users and 3,212 links (within 29 sub-reddits). Some additional statistics follow:

Statistic	**Value**
links with more than 750 votes	987
links with more than 1,000 votes	295
average value of votes (sum of all vote values / number of votes)	0.8037
number of down -votes	225,165
number of up-votes	2,069,366
number of votes per link	714
number of votes per user	94
number of votes per sub-reddit	79,122
number of users per sub-reddit	845

3. Model Evaluation: RMSE

In order to evaluate our predictions, we use the widely acknowledged root mean square error (RMSE) defined as:

$$rmse = \sqrt{\frac{1}{|S_{test}|} \sum_{(u,l) \in S_{test}} (r_{ul} - p_{ul})^2}$$

At test phase, we evaluate on 10% of the data which had been held out during training.

4. Baseline: Naive Predictor

Since about 90% of votes are up-votes (value +1) our first attempt is to try a naive predictor that guesses +1 for all votes. By doing so, we obtain an RMSE of **0.6265**. Note that this is a decent result because the statistics show that average value of votes is 0.8037, which is close to 1.

5. k-Nearest-Neighbors (kNN)

In the kNN model, each user is represented as a vector of votes (-1, 0, or +1) and each vote is for a particular link. We find the nearest neighbor users based on Euclidean distance. The number of nearest neighbors, k, was adjusted manually.

In the first variant of the kNN model, we predict an unknown user vote as follows: if the link has been seen by at least one neighbor, we predict the average of his neighbor's votes for the link; otherwise the link is unseen and we predict the user's own vote average. Unfortunately, the best performance was seen at k=1 (RMSE = 0.52397) which suggests that, in general, user average vote on unseen links is a better predictor than neighbor average on seen links.

The model above was improved by predicting a weighted average of neighbor votes ($w_{neighbor} = 0.3$) and user average ($w_{user} = 0.7$) for seen links to achieve RMSE = **0.521522** at k = 50.

One variant that did not lead to improvement was using link average rating for all unseen links. The best performance with this approach yields RMSE = 0.572846 at k = 10.

5.1 Incorporating Sub-reddit Data

Since the data provided has sub-reddit information, where each link belongs to one sub-reddit, it was important to incorporate this natural clustering in our model.

We were able to further improve the kNN model by predicting each unseen link as a weighted average of average sub-reddit votes ($w_{sub\text{-}reddit} = 0.3$) and average user vote ($w_{user} = 0.7$). This gives us RMSE = **0.518808** at k = 50. The improvement we obtained confirms that the clustering inherent in sub-reddit data is useful for the recommender.

6. Singular Value Decomposition (SVD)

SVD is one of most widely used models for collaborative filtering problems. Data in recommendation systems usually comes with very high dimensionality. The algorithm assumes that the data actually lies in a lower dimensional subspace. Therefore, by compressing and reducing dimensionality, we try to discover hidden correlations and patterns in the data while reducing noisy and redundant features. In essence, the algorithm tries to decompose the original user-link matrix A into the product of three matrices:

$$A_{n\times m} = U_{n\times r}\Sigma_{r\times r}(V_{m\times r})^T$$

From this, we can construct diagonal matrix S so that $S_{ii} = \Sigma_{ii}$ if $i \leq k$ else $S_{ii} = 0$. Now we have:

$$A'_{n\times m} = U_{n\times r} S_{r\times r} (V_{m\times r})^T$$

Here, A' is the best rank-k approximation for the original matrix A, and we use the entries of A' as predictions.

Since users only provide ratings for a small set of links, we need to fill the missing values in A. We have tested two approaches: filling with the average vote for a particular link, and filling with average user vote. Empirically, we have determined that the first approach generates a better result so that is what we used in the final model. For the decomposition algorithm, we used a randomized algorithm implemented in the *redsvd* library. This algorithm very efficiently solves the decomposition with high accuracy. SVD by itself obtains an RMSE of **0.55948**, with r = 100 and k = 30.

7. Bayesian Probabilistic Matrix Factorization using Markov Chain Monte Carlo (BPMF-MCMC)

This model tries to tackle the matrix factorization problem through a Bayesian approach. The prior distributions over user and link vectors are assumed to be Gaussian:

$$p(U|\mu_U, \Lambda_U) = \prod_{i=1}^{N} N(U_i|\mu_U, \Lambda_U^{-1})$$

$$p(V|\mu_V, \Lambda_V) = \prod_{i=1}^{M} N(V_i|\mu_V, \Lambda_V^{-1})$$

In addition, the model also places Gaussian-Wishart priors on the user and link hyper parameters $\Theta_U = \{\mu_U, \Lambda_U\}$ and $\Theta_V = \{\mu_V, \Lambda_V\}$:

$$p(\Theta_U|\Theta_0) = N(\mu_U|\mu_0, (\beta_0\Lambda_U)^{-1}) W(\Lambda_U|W_0, v_0)$$

$$p(\Theta_V|\Theta_0) = N(\mu_V|\mu_0, (\beta_0\Lambda_V)^{-1}) W(\Lambda_V|W_0, v_0)$$

Here W is the Wishart distribution with v_0 degrees of freedom, and W_0 is a D×D matrix:

$$W(\Lambda|W_0, v_0) = \frac{1}{C}|\Lambda|^{(v_0-D-1)/2} exp(-\frac{1}{2}Tr(W_0^{-1}\Lambda))$$

The model is then trained through Gibbs sampling. BPMF-MCMC by itself achieves an RMSE of **0.5018**, which is a very decent performance.

8. Stochastic Gradient Descent (SGD)

We have also tested a simpler model based on stochastic gradient descent. Let q_l be the link vector, q_u be the user vector, and r_{ul} be the rating from user u for link l. The model tries to minimize the prediction error:

$$\min_{q*,p*} \sum_{u,l} (r_{ul} - q_l^T p_u)^2 + \lambda(||q_l||^2 + ||p_u||^2)$$

During the training process, we first calculate the error $e_{ul} = r_{ul} - q_l^T p_u$. Then we can update the parameters in the following way:

$$q_l \leftarrow q_l + \gamma(e_{ul} p_u - \lambda q_l)$$

$$p_u \leftarrow p_u + \gamma(e_{ul} q_l - \lambda p_u)$$

SGD is able to attain an RMSE of **0.596085**. Although SGD does not perform as well as BPMF-MCMC, we determined that the performance was still decent enough that SGD should be included in our final combined model.

9. K-means

Another model we tried involved using k-means. As with kNN, each user is represented as a vector of votes, one vote for each link. We used k-means to cluster similar users and make predictions for a test user-link pair by taking the average of votes made by other cluster members for that particular link. In the case where no cluster member has seen the link, we predict user average. This algorithm gives us RMSE of **0.536** for both 10 and 100 clusters. However, upon closer examination, it turns out that the majority of users (>90%) were assigned to one cluster – clustering had very little effect. Therefore, we decided to exclude k-means from the final model.

10. Linear Combination of Models

At this point, we had four models (kNN with sub-reddit data, SVD, BPMF-MCMC, and SGD) that we wanted to use in a linear combination for the final model. These will be indexed in order, 1 through 4. Given a test user-link pair, a model will make the prediction of $model_x(u, l)$, where u is the user and l is the link. We also added the +1 and -1 constant terms to the linear combination. Equivalently,

$$model_5(u, l) = 1$$

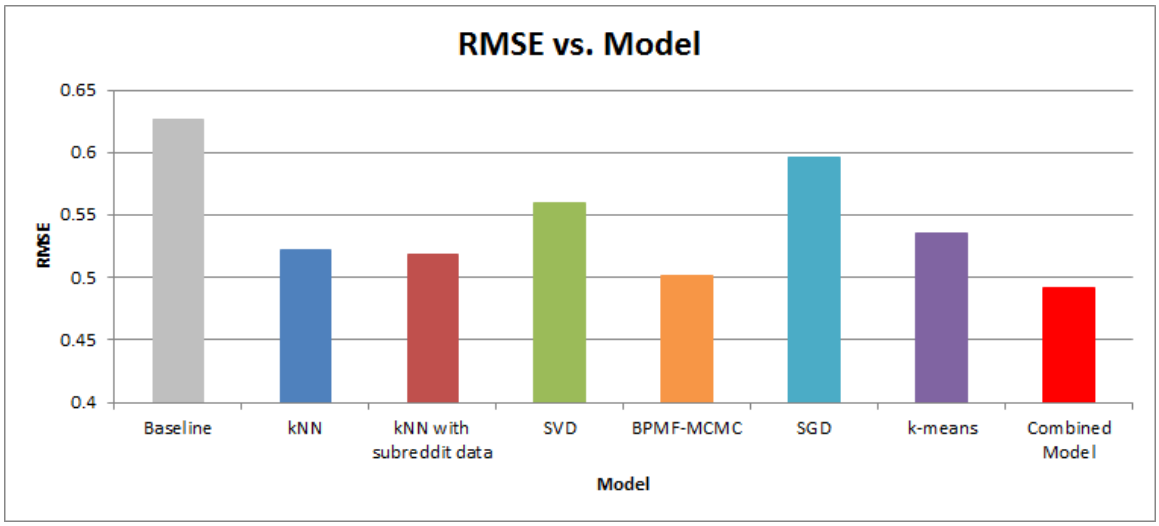
$$model_6(u, l) = -1$$

Our linear combination is:

$$prediction(u, l) = \sum_{x=1}^{6} w_x \, model_x(u, l) \ , \qquad 0 \le w_x \le 1$$

We implemented an automatic parameter tuner to efficiently adjust the weights for each model. Each weight was initialized to ⅙. At each iteration, the algorithm randomly chooses a weight and assigns it a random real value in the range [0, 1]. If the combined model performs better than the current best RMSE, the assignment is kept; if performance is the same as the best, the assignment is kept with 50% probability; otherwise, the assignment is rejected.

Several variations of this tuning process were attempted. We tried to normalize the weights so that they sum to 1 after a weight assignment at every iteration but the algorithm converged much slower than without normalization. We also tried the uniform distribution $U(0, 1)$ and Gaussian distribution $N(w_x, 0.1|w_x|)$ when generating a real number at each iteration. The uniform distribution was much faster at zoning in on a set of near-optimal weights, while the normal distribution was much more useful to fine-tune weights. Note that for the normal distribution, since variance approaches zero as w_x approaches zero, the weights can get stuck near zero. To address this, we ensure that variance > 0.001. Our final parameter tuner runs about 50 iterations with uniform distribution, followed by about 500 iterations with the normal distribution. The resulting weights were: w_1 = 0.0677075, w_2 = 0.0985695, w_3 = 0.934, w_4 = 0.0413389, w_5 = 0.0604689, and w_6 = 0.165892. This gives us a combined model RMSE of **0.491433**, which is better than any model individually, and the best performance overall. The following graph summarizes the performances of all models:



11. Conclusion and Future Work

Using data provided by the Reddit team, we were able to build a recommendation system which achieves an RMSE of 0.491433 using a linear combination of kNN with sub-reddit data, SVD, BPMF-MCMC, and SGD. However, the recommender can certainly be improved further.

The provided data has limited features. There are many powerful models that could be used if we were given more detailed data. For example, if we had time stamps associated with each vote, we would be able to test a few popular temporal models. Intuitively, more recent votes should be valued higher than older ones. Furthermore, if we were given the actual URLs and titles of the links rather than encrypted ids, we may use NLP techniques to add more features to the data. Another promising group of models that we should test are ones based on weighted ratings. Although the original data does not have any weight or confidence information, we can try to infer the weights based on the popularity of the links and the activeness of the users. These additional features all have the potential to decrease our RMSE score and significantly improve the recommendation system.