Reviewer Rank Analysis

Brian Oh

Derek Nelson

Contents of Table

1. Abstract
2. Yelp
3. Problem
4. Reviewer Rank
5. Modification to each Review Score
6. Business Rank
7. Concerns
8. Our Goals
9. Yelp Data
10. Transition Matrix generation
11. Fake Review Simulation
12. Three Different Ranking Models
13. Testing & Evaluation
14. Comparison 1 (without modification)
15. Comparison 2 (with fake reviews - no useful vote)
16. Comparison 3 (with fake reviews and spam farm architecture)
17. Additional factors (time depreciation, number of item advantage)
18. What to do Next
19. Conclusion

Abstract

Yelp is having issues with fake reviews, and they reported approximately 25 percent of the reviews are fake reviews. To measure better ranking we suggested a reviewer rank model that uses a similar algorithm as pagerank. We compared two different models with reviewer rank. The first model is simply averaging review scores for a business. The second one calculates user weight using the following formula:

user weight = the number of user's useful vote/ total number of all user's useful vote

To test how all three models perform, we picked one restaurant, and simulated writing 100 fake reviews with 5 stars, then measured the ranking of the chosen restaurant. In this simulation, the result of higher rank means the model is easily manipulated by fake reviews. We repeated the same process with spam farm architecture with useful votes between fake reviewer accounts to test how those models perform against a spam farm.

Unexpectedly, the second model above out-performed the reviewer rank model. We want to study more about the reason for this result. As we expected, the reviewer rank model performed poorly with the spam farm architecture. We also added some factors to improve the performance; such as time depreciation of review score, and an additional advantage when a business has a higher number of reviews. We got slightly better results with those additional factors. We ended our project with some ideas to improve the performance of the models, by filtering unusual data such as too many reviews in a short amount of time, too many useful votes in a short amount of time, etc.

Yelp

Yelp is one of the biggest review web sites and was founded in 2004. They provide valuable local business info to users. They had a monthly average of 89 million unique visitors who visited Yelp via their mobile devices and more than 90 million reviews were submitted in Q3 2015.

Problem

Recently Yelp is having hard time fighting with fake reviews. People are writing fake reviews to raise their own business ranking or to lower competitor's business ranking. If you search Google with “yelp fake review”, there are lots of articles about their fake review problems. According their report, about 25 percent of submitted reviews are suspicious or not recommended. They have recently filed lawsuits against websites that sell fake reviews, and 20% of their total employees are manually filtering fake reviews. However, it is very hard to get rid of fake reviews.

In a paper by Arjun Mukherjee titled “Fake Review Detection: Classification and Analysis of Real and Pseudo Reviews” we find that in 2013 there were many problems with fake reviewing. Arjun calls these people opinion spammers. These opinion spammers are causing such a problem that Yelp even tried to produce an algorithm, that they have kept to themselves, to detect these spammers. The paper talks about how the main type of algorithms are supervised learning algorithms. But these techniques rely on gold-standard fake reviews,which are hard to determine, due to the writing style of people when they give reviews. This then turns to methods that are mostly based on ad-hoc labels. Some methods try to detect duplicate reviews, these duplicate reviews can be from the same user or different users, but they are assumed to be fake. Arjun gives a Yelp dataset of good reviews, taken from Yelp, and poor reviews, also given by Yelp, and adds 400 fake reviews made by Amazon Mechanical Turk. These were applied to a supervised learning algorithm to remove the effect of these reviews.

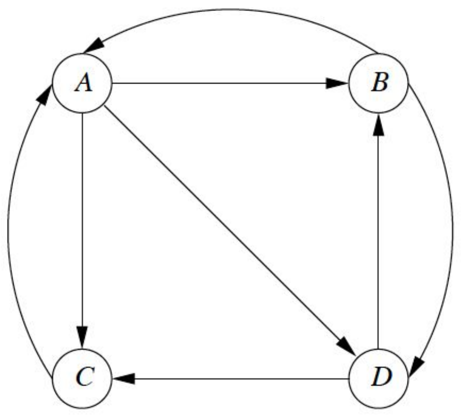
There was another paper written by Michael Luca in May of this year titled “Fake It Till You Make It: Reputation, Competition, and Yelp Review Fraud”. This paper goes through a similar study that gives additional light on the issue with filtering fake reviews. The algorithm developed by Yelp.com made lots of mistakes that were false positive and false negative. An example being that when a review that was not fake was filtered, or when the review was fake but not filtered. Another point that is made is that people that actually purchase a product from the business tend to leave good reviews, and people that do not purchase products tend to leave negative reviews. This gives a very wide range to the spread of data, in which is difficult to detect real reviews that are not fake.

In another paper by Susan M. Mudambi, titled “What Makes a Helpful Online Review? A Study of Customer Reviews on Amazon.com”. This paper is directly related to Amazon products, but the underlying message applies to all reviews. The paper talks about how helpful votes can help persuade a consumer to side with the author of the review. This influences the places that you as a consumer like and eventually purchase from. So to know what ‘helpful’ means can be challenging.

We decided to take a direction that presents itself in the middle of the two. We wanted to take these approaches, and put them together. Hopefully, presenting a method that does not use filtering, but reduces the effects of the fake reviewers, and also utilizes the useful votes of each review. We are not going to use a supervised learning algorithm because that type of algorithm is beyond the scope of the course.

Reviewer Rank

We came up with idea that we could use Google’s pagerank algorithm to solve the problem. In pagerank, if a webpage has more links to it, it gets a higher rank. Similar to this, when a user writes a review, other users can vote it as useful. If a reviewer gets many useful votes from other users, this reviewer should get higher ranking. With this basic idea, we generated a user to user transition matrix. This matrix represents user to user votes. Matrix *M* has *n* rows and columns, if there are *n* users. The element *mij* in row *i* and column *j* has value 1/*k* if user *j* has *k* arcs out, and one of them is to user *i*. Otherwise, *mij* = 0. For example, in figure 1, there are 4 users, A, B, C, and D. The arrow represents useful votes from one user to the other. Then, we can generate the transition matrix *M* see Figure 2.



$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

Figure 1. Figure 2.

Next, we need to generate the reviewer rank vector. This vector is a column vector, and it describes each user's weight. Initially, this vector starts with same value 1/*n* for all elements where *n* is number of all users. However, as we multiply this vector to transition matrix *M*, it starts to change and starts to represent reviewer rank. Each iteration of multiplication will generate new user weight vector *v'*. In other words, *v'* = *Mv*. And finally, if we continue this iteration, *v'* converges to a certain state, and does not change anymore, then becomes our final user weight vector. The sum of the elements should be 1, and each element represents how much percentage this user's opinion takes among the whole group's opinion.

However, this algorithm requires two conditions: First, the graph is strongly connected ; that is, it is possible to get from any node to any other node. Second, there are no dead ends: nodes that have no arcs out. Our situation doesn't meet these requirements, so we have to utilize taxation. To utilize taxation, each iteration need to be altered as follows:

$$\mathbf{v}' = \beta M \mathbf{v} + (1 - \beta)\mathbf{e}/n$$

where β is a chosen constant, usually in the range 0.8 to 0.9, e is a vector of all 1's with the appropriate number of components, and n is the number of users. The term βMv represents the normal useful vote part with probability β. The term (1−β)e/n is a vector each of whose components has value (1−β)/n and represents the introduction, with probability 1 − β, taxation part. For more detailed information please check Google's pagerank algorithm.

Modification to each Review Score

Now, with the reviewer rank we got, we need to modify each review score. First, review scores range from 1 to 5 stars, and we need to modify this from -2 to 2. It is because 1 to 2 stars represent negative opinion on the business, 3 star is neutral, and 4 to 5 stars represent positive opinion. Then we multiply reviewer rank to the review score that ranges from -2 to 2.
Modified Review Score = (Review Score -3) X Reviewer rank
As an example, Papa John's got 1 star from a high rank reviewer:

(1 - 3) x (0.02) = - 0.04.

Here, 0.02 is very high rank, and this review had a very negative impact on Papa John’s ranking. As another example, Olive Garden got 5 stars from low rank reviewer:

(5 - 3) x (0.0000013) = + 0.0000026

Here, 0.0000013 is a relatively lower rank, and the 5 star review has little positive impact on Olive Garden.

Business Rank
As our final step, summing all the modified score for a particular business will be the rank of the business.

Business Score = ∑ modified score / number of reviews

To find the best restaurant in a city, we just need to filter the list with restaurants in the city, then, sort the restaurants by business rank.

Concerns
We had two concerns on this project. First, is user to user useful vote data available? Unfortunately, we could not find any data set that included this data, but yelp data included the number of useful votes that each user got, so we generated the transition matrix based on this data. Second, is the transition matrix too sparse? Fortunately, Yelp user vote data was not that sparse. Approximately, 90% of users have multiple useful votes from other users.

Our Goal

Our goal of this project is to analyze effectiveness of the reviewer rank system. To accomplish this goal, we decided to do two things. First, simulate writing fake reviews, and analyze impact on business ranking, and second, compare to other rank models.

Yelp Data

We obtained Yelp data from http://www.yelp.com/dataset_challenge. The data is approximately 2 GB in size, and it has been divided into five groups. Each group of data is given in JSON format.

The first group of data is business data. The data given in the business data contains the business ID, full address, hours, categories, city, review count, name, state, stars, and different attributes. All of this information is too much so we decided to slim it down to:

Business data: (BusinessID, BusinessName, categories, city, state, price)

The number of individual businesses is 61184.

The second group of data is review data. The data given in the review data contains vote information (useful, funny, cool), user ID, review ID, stars, date, text, type, and business id for each review. The text of each review was too much for us so we removed this information. We slimmed it down to:

Review data: (review date, review score, userID, BusinessID)

The number of individual reviews is 1569264.

The third group of data is user data. The data given in user data contains how long the user has been on Yelp, vote information (funny, useful, cool), number of reviews, name, user ID, friends, number of fans, average stars, type, compliments, and years when elite. This was too much information for us so we slimmed this information to:

User data: (UserID, number of reviews, number of reviews helpful to other user)

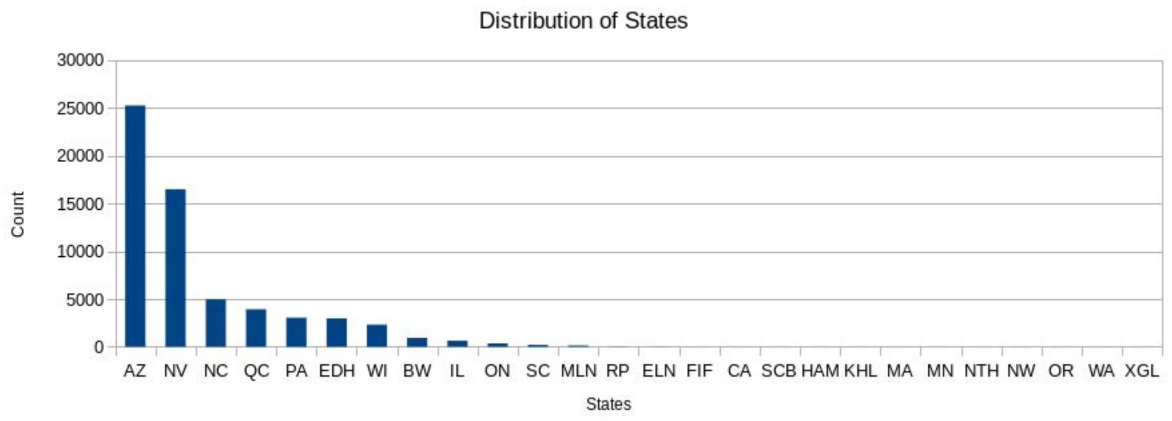
The number of individual users is 366715.

The fourth group of data is check-in data. The data given in check-in data contains check-in information, type, business ID. This information was not needed for our project so we didn't use this group.

The fifth group of data is tip data. The data given in tip data contains user ID, text, business id, likes, date, and type. This data was not needed for our project so we didn't use this group.

To make the calculations and problems simpler we made some assumptions about the data. First, we are assuming that all the data given by Yelp contains no farming. Second, there is a random distribution of the users. This allows us to create user to user data to do further analysis and to make our review ranking system work. Third, we assume all the data is complete, meaning the five groups all coincide with each other.

As we looked at the data we noticed that there were only a select number of states. The following figure shows that the most common states are Arizona and Nevada. There are a total of 26 states. There are 391 distinct cities. The maximum number of reviews for one business is 4578 reviews, and the minimum number of reviews for one business is 3.



Knowing the distribution of states allowed us to choose an individual business to determine if the method we are using makes a difference.

We were not able to get a larger set of data because we couldn’t find a larger set. We started looking for Amazon data but found that Amazon didn’t offer user to user information. This was a typical scenario with all the data we found. The data that we found that was even close to what we were looking for was the Yelp data. We did the best we could with what we got.

Transition Matrix generation

User to user vote data was not available. Fortunately, the number of useful votes that each user got was available, so we generated based on this data. For example, if user A had 100 useful votes from other users, we simply picked 100 other users, then removed duplication. This process will generate 100 of (A, x), where x is random 100 numbers. After that, we still need to count the number of out-going votes for each user which will be the degree of out-going votes.

Continuing above example, we need to count how many votes user A sent to other users which is in the form of (y, A). Let’s say it was 60 votes. Finally, we need to add degree of the vote, and it will end up with 60 of ((y, A), 1/60), where y is random 60 users. We need to do the same process for every user to generate the full transition matrix.

Fake Review Simulation

To Analyze the effectiveness of our reviewer rank model, we decided to simulate writing fake reviews. We chose one particular restaurant that has a low average star rating in Las Vegas. It has 19 reviews, and average star rate is 1. The data we picked was as follows:

- business_id: '-sV52FN-D-I808tyRPEvwg'
- name: "Papa John's Pizza"
- categories: ['Pizza', 'Restaurants']
- stars: 1.0
- city: 'Las Vegas'
- state: 'NV'
- review_count: 19

We will analyze how its ranking changes as we modify our data on this chosen restaurant.

Three Different Ranking Models

To analyze the effectiveness of our reviewer rank model, we decided to compare it with two different models.

Averaging

This model is simply averaging review scores. It doesn’t care who wrote the review, or who voted this user as a good reviewer.
We calculated this by:

$$ReviewScore = Star\ Count\ for\ Review$$
$$RC = Review\ Count\ for\ Business = \#\ of\ Reviews$$
$$Rank\ = \frac{\Sigma(ReviewScore)}{RC}$$

Each review has a score between 1 to 5. Then we average that score for each business.

Useful Weight

This model does care who wrote this review, and the reviews written by a user who has more useful votes gets heavier weight on the ranking of the business. However, it doesn’t care who voted this user, it simply counts how many useful votes this user got.
We calculated this by:

$$ReviewScore = Star\ Count\ for\ Review$$
$$UserWeight = \frac{User\ Useful\ Votes}{Total\ Sum\ of\ Useful\ Votes}$$
$$RC = Review\ Count\ for\ Business = \#\ of\ Reviews$$

$$MR = Modified\ Reveiw\ Rank = (ReviewScore\ -\ 3)\ *\ UserWeight$$
$$Rank\ = \frac{\Sigma(MR)}{RC}$$

1. Each review has a score on a scale of 1 to 5, but we don't just average these scores. Instead we will use a modified review score as follows:
   *ModifiedReviewScore = (ReviewScore - 3) * UserWeight*

Some users have written many reviews, and if a review was helpful to anyone, they can click helpful for the review. We wanted to amplify the opinion of the users who got many helpful votes. This is accomplished by scaling the review score from -2 to 2 instead of 1 to 5, then we multiply by the user weight.

2. User weight is calculated by taking that users useful votes and dividing it by the total count of all useful votes given to users.
3. Then we divide by the number of reviews that the business had to produce a rank for each business.

Reviewer Rank

This is our model, and it does care both who wrote this review, and who voted this user as a good reviewer. If a user gets a useful vote from a high rank user, it increases this user's rank greatly, and useful votes from a low rank user doesn't increase that much. The only difference from the Useful Weight model is that we use the result from pagerank as user weight instead of number of useful vote.

We calculated this by:

$$ReviewScore = Star\ Count\ for\ Review$$
$$UserWeight = Result\ from\ pagerank$$
$$RC = Review\ Count\ for\ Business = \#\ of\ Reviews$$
$$MR = Modified\ Reveiw\ Rank = (ReviewScore\ -\ 3)\ *\ UserWeight$$
$$Rank\ = \frac{\Sigma(MR)}{RC}$$

1. Each review has a score on a scale of 1 to 5, but we don't just average these scores. Instead we will use a modified review score as follows:
   *ModifiedReviewScore = (ReviewScore - 3) * UserWeight*

Some users have written many reviews, and if a review was helpful to anyone, they can click helpful for the review. We wanted to amplify the opinion of the users who got many helpful votes. This is accomplished by scaling the review score from -2 to 2 instead of 1 to 5, then we multiply by the user weight.

2. UserWeight is calculated in a similar fashion as the pagerank. We will have N x N transition matrix, row and column represent users, and the value is 1/degree. v starts as 1/(number of users). We will apply taxation as well.
3. Then we divide by the number of reviews that the business had to produce a rank for each business.

Testing & Evaluation

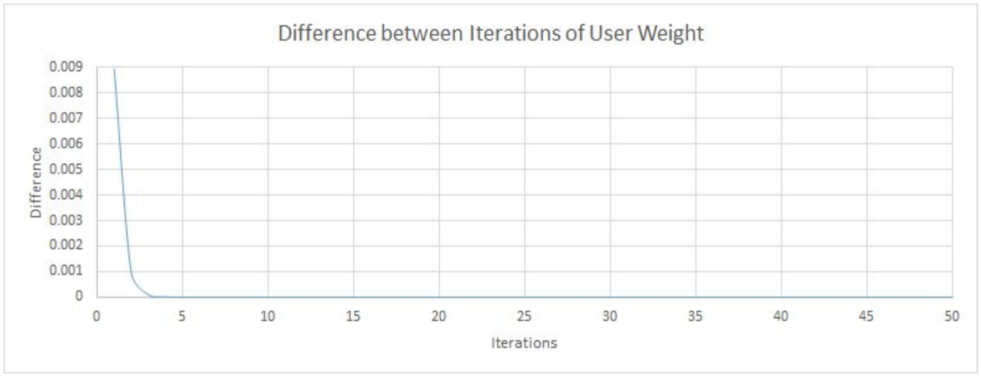
To test these methods we started with smaller sets of data that we created. These small sets were similar to the real data but it allowed us to ensure our code was working. The data from Yelp was large enough that it was hard to tell what we were getting because of the large business count. To narrow the search we took the information from how the states were distributed. There were no businesses from Utah so we chose the closest state, which was Nevada. We were familiar with Las Vegas so we decided to concentrate on that city. Also the most familiar category to both of us was restaurants. This gave us a large selection of businesses. We decided to pick the worst business out of that group. This meant we sorted the businesses by score of 1. This gave us a very small list. The business that was well known and had lots of reviews ended up being a Papa John's Pizza. The code below allowed us to search for that one business every time, while producing an order which is the score or rank given in the results following.

```
business = businessData \
    .filter(lambda x: filter_by_city('Las Vegas', x[4])) \
    .filter(lambda x: filter_by_state('NV', x[5])) \
    .filter(lambda x: filter_by_category('Restaurants', x[2])) \
    .map(lambda x: (x[0], x[1])) \
    .join(reviewScore) \
    .sortBy(lambda (businessID, data): data[1], False) \
    .map(lambda (businessID, data): (businessID, data[0])) \
    .zipWithIndex() \
    .filter(lambda (data, rank): data[0] == "-sV52FN-D-I808tyRPEvwg") \
    .collect()
```

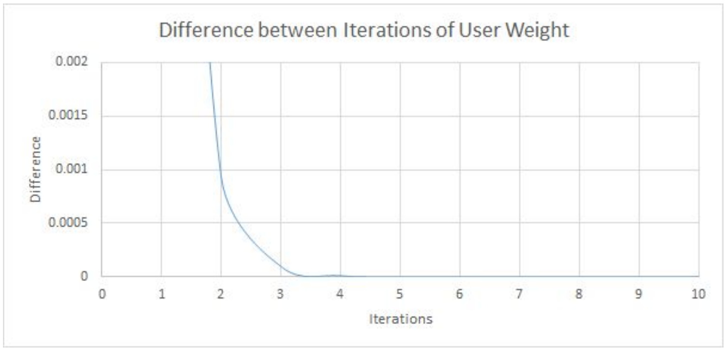
The code filters all the data by the city, state, and category before joining the review score. Then it zips an index according to the review score. After the score has been assigned we can filter the exact business we are looking for, allowing us to see how the index, or score, changes.

The user weight was not used with the averaging method. The user weight that was calculated with the useful weight was all the same. Then with the review ranking system the user weight was either one for each user or the calculated user weight from the pagerank calculation.

We found that the pagerank calculation converged with about 5 iterations.



If we zoom in to the area of interest, we can see that the method converges at 4 iterations.



Comparison 1 (without modification)

This is pure data and the results of three models follows:

Averaging: 4081

Useful Weight: 3700

Reviewer Rank: 3757

Comparison 2 (with fake reviews - no useful vote)

We added 100 farmer accounts. The purpose of these farmer accounts is to raise the ranking of the chosen restaurant. There is a Main account, and 99 supporter accounts. They all wrote 1 review for the chosen restaurant, and rated it 5 stars.

Averaging: 107

Useful Weight: 3301

Reviewer Rank: 3125

As we expected the averaging method skyrocketed, while both useful weight and reviewer rank method stayed lower. Surprisingly useful weight method out-performed the reviewer rank method.

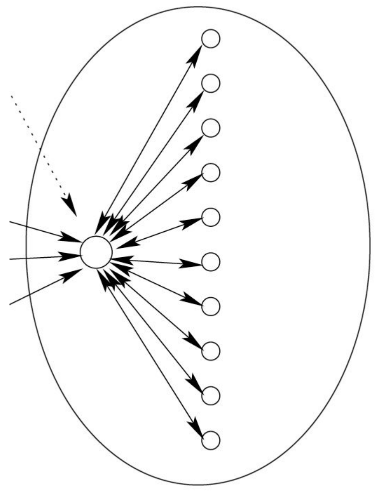
Comparison 3 (with fake reviews and spam farm architecture)

To test the performance against a spam farm architecture, we decide to simulate a spam farm architecture for our models. We added useful vote information to our transition matrix. The main account voted the reviews of all 99 supporters as "useful", and all supporters voted the review of the main account as "useful" just like the spam farm in the picture below.

Averaging: 107

Useful Weight: 3261

Reviewer Rank: 1702



As we expected reviewer rank suffered from the spam farm architecture. The spam farm is designed to fool the pagerank system, and it fools the reviewer rank system as well. In contrast, the useful weight system is still performing well.

Additional factors (time depreciation, number of item advantage)

Finally, we added two more aspects to our ranking in an attempt to get more accurate results.

1. Review score will be depreciated in time. Depreciation Rate is as follows:
   *DepreciationRate = 1 - (AgeOfReview * DepreciationConstant)*
   Depreciation Constant is not determined yet. We could decide one value, or we might find out some more meaningful way to come up this number. If the DepreciationRate become negative, it will be 0.

2. If a product has a high number of reviews, it should have a higher rank because it means that many people were interested in that product, and bought it. Then we just multiply [1+ (TotalNumberOfReviews * AdvantageConstant)] to the score.

Finally our review score for a product is as follows:

$$ReviewScore = Star\ Count\ for\ Review$$
$$UserWeight = Result\ from\ PageRank$$
$$RC = Review\ Count\ for\ Business = \#\ of\ Reviews$$
$$AgeOfReview = difference\ in\ days$$

$$MR = Modified\ Reveiw\ Rank = (ReviewScore\ -\ 3)\ *\ UserWeight$$
$$DR = Depreciation\ Rate = 1 - (AgeOfReview * 0.0001)$$
$$AR = Advantage\ Rate\ =\ 1 + log(RC))$$
$$Rank\ = \frac{\Sigma(MR*DR)}{RC} * AR$$

The following is the result of a modified reviewer rank model.

No Modification: 3770 (down from 3757)

With Fake Review: 3123 (down from 3125)

With Spam Farm Architecture: 1362 (down from 1702)

The results are a little bit better compared to the result for the plain reviewer rank model.

What to Do Next

Since we don't know why the useful weight model out-performed the reviewer rank model, we want to study more about this.

We also thought about how we can improve the result from all of the models, and we came up with the idea that filtering unusual data can be helpful, especially against the spam farm architecture. Some examples of unusual data would be following:

- A user writes too many reviews in a short amount of time.
- A user gives too many useful votes in a short amount of time.
- A user gets too many useful votes in a short amount of time.
- A user writes too many reviews in many different regions.

Conclusion

Unexpectedly, the user weight model out-performed reviewer rank. It turned out that the reviewer rank model is not very effective in terms of both cost of computation and performance. However, we did not have enough time to study the reason of the result. We want to study more about it. Adding some additional factors such as time depreciation, and advantage to number of reviews gave us better results, so we believe that adding additional factors can be helpful.

References

Luca, Michael, and Georgios Zervas. "Fake it till you make it: Reputation, competition, and Yelp review fraud." *Harvard Business School NOM Unit Working Paper* 14-006 (2013).

Mudambi, Susan M., and David Schuff. "What makes a helpful review? A study of customer reviews on Amazon. com." *MIS quarterly* 34.1 (2010): 185-200.

Mukherjee, Arjun, et al. *Fake review detection: Classification and analysis of real and pseudo reviews*. Technical Report UIC-CS-2013-03, University of Illinois at Chicago, 2013.