README.md

Team

- Karan Shukla
- Ashwin Kumar
- Linh Truong Hoang
- Viswa Teja

Description of Project

Our project is based off the paper **"A Probabilistic Graphical Model for Brand Reputation Assessment in Social Networks"**, written by researchers at **Northwestern University** and presented at the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. **The paper proposes a probabilistic graphical model to measure "brand reputation" on a social network.** The paper uses a **Facebook dataset**, where "brands" include public pages, such as Barack Obama or Starbucks. Along with social brand reputation, the proposed model computes "user positivities", a metric of how positive or negative a user's posts are on the social network.

Both **social brand reputation and user positivities are modeled as hidden variables** whose values are inferred from the **sentiment of social media posts, which act as the observed variable**. The **inference is conducted via Markov Chain Monte Carlo sampling**.

We aim to produce a similar model to the one described in the paper on an Amazon product review dataset. This dataset is a good fit, because just like in a social network, Amazon has "brands" (products) with "reputations" (5-star ratings) determined by "posts" (reviews) from "users". We can evaluate our model's success by comparing each product's "social brand reputation" with their real Amazon rating.

This model computes `P(R|S)` and `P(U|S)` , where `R` is a brand's reputation or a product's rating, `U` is user positivity, and `S` is the sentiment of posts or reviews. The model does not compute `P(R|U)` , the probability of a particular user liking a product. In other words, this model only computes the "aggregate" quality of a product, given a corpus of reviews. This model is not a recommendation system. It would be interesting to explore the use of MCMC models in recommendation systems in the future.

Sentiment analysis is performed by the Textblob package, which wraps around the pattern package created by the CLiPS research center at the University of Antwerp. The sentiment analysis is performed by assigning a sentiment polarity to every adjective in the text and averaging all of the adjective sentiments. This is a very simple model; more advanced models could be trained and tested on the Amazon Reviews for Sentiment Analysis dataset. For simplicity, we assume that Textblob's false positive rate and false negative rate are approximately equal, so that the predicted rating for any given product is not impacted by errors in Textblob's sentiment prediction.

The original paper uses a much more involved **ensemble sentiment analysis algorithm** that we excluded in our model due to time constraints. The original paper also has an involved data cleaning phase that filters out brands and users with very few posts, as well as users with too many posts (spammers). We did not include this phase.

Challenges faced

1. There are many ways to represent the relation among Users and Social brand. (compares the model with several relations)
2. The accuracy of sentiment identification of social media is crucial.
3. Due to volume of data, data quality and efficiency of inference is another big challenge. (filter out noisy data and spam users)

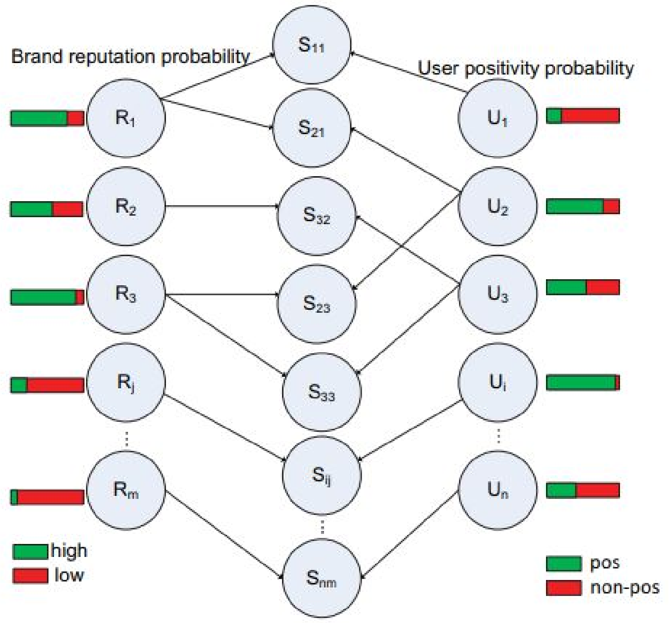


Fig. 1: A collective inference framework of our system. U_i: positivity of a i-th user; R_j: reputation of a j-th brand; S_{ij}: sentiment of comments made by U_i on brand R_j. Green means the high reputation for brands and more positive for users. Red has the opposite meaning. (Best viewed in color)

Data

We received our training data from the University of California San Diego's Amazon dataset. In particular, the dataset we used was `reviews_Office_Products_5.json.gz` , a set of reviews for office products on Amazon from 2014, but our technique should work on any of the Amazon datasets provided from the University of California San Diego. The dataset contains approximately **50,000 reviews**, with approximately 700 products. On average there are about 70 reviews per product.

Our testing data was **scraped from Amazon's website** using Python's `requests` library. Our scraping program searched for the same products that were in the training set; hence, the number of products in the training set and testing set are about the same. It should be noted that some of the products in the training set are no longer sold on Amazon; hence, the testing set of products is actually slightly smaller than the training set.

Mathematics and Time Complexity of the Model

Brute Force

A naive, brute force inference of reputation of brands (R) and positivity of users (U) given the sentiment of comments/reviews (S) can be seen below.

$$P(R_i \mid S_{11}, \cdots, S_{ij}, \cdots, S_{mn})$$

$$= \sum_{R-i,U} P(R_1, \cdots, R_m, U_1, \cdots, U_n \mid S_{11}, \cdots, S_{ij}, \cdots, S_{mn})$$

$$= \sum_{R-i,U} \frac{P(R_1, R_2, \cdots, R_m, U_1, U_2, \cdots, U_n, S_{11}, \cdots, S_{ij}, \cdots, S_{mn})}{\sum_{i,j} P(S_{ij})} \qquad (1)$$

Given that there are m brands and n users, each of which could be "positive" or "negative", we sum over `2^(m+n)` unique combinations of brand and user positivities. For each combination, we need to sum the probability of sentiment for each review. The number of reviews could be up to `mn` (one review per user-brand tuple). So, in total, the brute force computation would take `O(mn * 2^(m+n))` time.

As we can see, this approach takes superpolynomial time, and is not feasible. We reduce the state space that we compute over by instead adopting the Makrov Chain Monte Carlo method.

Markov Chain Monte Carlo

In this approach, we iteratively sample a positivity for every brand `R(i+1)` and every user `U(i+1)`, using the previous sample's positivities `R(i)` and `U(i)`, as well as the sentiments of the reviews `S`.

Our Markov Chain is the sequence of samples `R` and `U`, dependent on the previously-drawn values of `R` and `U`. By iteratively sampling via the Markov Chain, we eventually create a sample that approaches that of the true population distribution.

The inference equations (derived from Bayes Rule, applied on the structure of our Bayesian Net) can be seen below.

$$\begin{aligned}
&P(R_i \mid R_{-i}, U, S) \\
&= \frac{P(R,U,S)}{P(R_{-i},U,S)} \\
&= \frac{P(R,U,S)}{\sum_{R_i} P(R,U,S)} \\
&= \frac{P(R_1)\cdots P(R_m)\cdot P(U_1)\cdots P(U_n)\cdot \prod_{i,j} P(S_{ij}|U_i,R_j)}{\sum_{R_i} P(R_1)\cdots P(R_m)\cdot P(U_1)\cdots P(U_n)\cdot \prod_{i,j} P(S_{ij}|U_i,R_j)} \\
&= \frac{P(R_i)\prod_k P(S_{ki}|U_k,R_i)}{\sum_{R_i} P(R_i)\prod_k P(S_{ki}|U_k,R_i)} \qquad (2)
\end{aligned}$$

$$\begin{aligned}
&P(U_j \mid U_{-j}, R, S) \\
&= \frac{P(U,R,S)}{P(U_{-j},R,S)} \\
&= \frac{P(U,R,S)}{\sum_{U_j} P(U,R,S)} \\
&= \frac{P(R_1)\cdots P(R_m)\cdot P(U_1)\cdots P(U_n)\cdot \prod_{i,j} P(S_{ij}|U_i,R_j)}{\sum_{U_j} P(R_1)\cdots P(R_m)\cdot P(U_1)\cdots P(U_n)\cdot \prod_{i,j} P(S_{ij}|U_i,R_j)} \\
&= \frac{P(U_j)\prod_k P(S_{jk}|U_j,R_k,)}{\sum_{U_j} P(U_j)\prod_k P(S_{jk}|U_j,R_k)} \qquad (3)
\end{aligned}$$

In a single iteration, for a single brand, we sum over both reputation values `R_i` and multiply over each user in `U`. Thus, the computation takes `O(2n)=O(n)` time.

There are `m` products, each of which is updated every iteration, so the total time complexity of Markov Chain Monte Carlo sampling is `O(mni)`, where i is the number of iterations.

The same logic follows for sampling user positivies.

Priors

The inference equations above require the calculation of `P(S|R,U)`. The paper provides a prior distribution for this conditional, seen below.

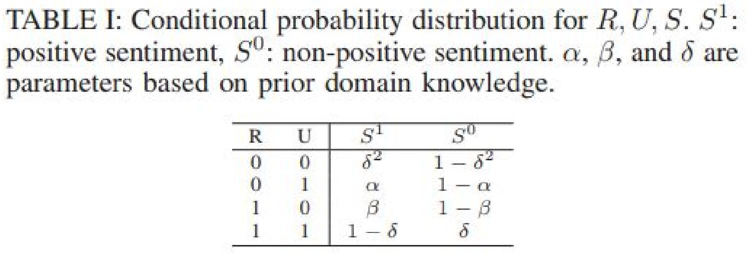


TABLE I: Conditional probability distribution for R, U, S. S^1: positive sentiment, S^0: non-positive sentiment. α, β, and δ are parameters based on prior domain knowledge.

R	U	S^1	S^0
0	0	δ^2	$1-\delta^2$
0	1	α	$1-\alpha$
1	0	β	$1-\beta$
1	1	$1-\delta$	δ

The probabilities P(S|U,R) are dependent on priors alpha, beta, and delta.

- alpha = `P(S=1 | U=1, R=0)` is the probability that a positive user rates a negative product positively. It can be interpreted as the strength of a user's positivity on a review's rating.
- beta = `P(S=1 | U=0, R=1)` is the probability that a negative user rates a positive product positively. It can be interpreted as the strength of a product's quality on a review's rating.

- delta = `P(S=0 | U=1, R=1)` is the probability that a positive user rates a positive product negatively. It can be interpreted as a "noise factor". Note that `P(S=0 | U=1, R=1) > P(S=1 | U=0, R=0)`; this assumption states that a positive user is more likely to write a negative review about a bad product than a negative user is to write a positive review about a bad product.

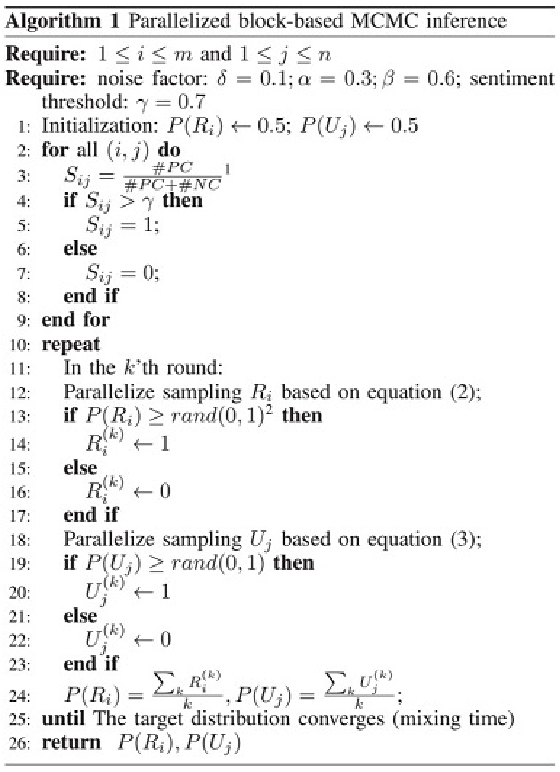
The paper makes the assumption that alpha < beta (the product's quality has a greater impact on a review's rating than the user's positivity). In other words, the authors assume that a negative user was more likely to give a positive score for a high-quality product than a positive user was to give a negative score for a low-quality product. Although, the paper provides no justification for this assumption, it intuitively made sense, so we adopted it in our work.

The values used in the paper are alpha=0.3, beta=0.6, and delta=0.1, which we used as a "control" or "baseline" for the model. We compared the performance of the model using three alternative parameter configurations (increasing alpha, decreasing beta, and increasing delta).

Parallelized Block-Based MCMC Inference

Notably, the inference computation for some product `i` is independent of all other product. Similarly, the inference computation for some user `j` is independent of all other users. This means that we can run these computations in parallel, which significantly reduces sampling time for MCMC.

The paper's algorithm alternates between sampling `P(R|S,U)` and `P(U|S,R)`, and refers to this as "Parallelized Block-Based MCMC Inference". The algorithm can be seen below.



Algorithm 1 Parallelized block-based MCMC inference

```
Require: $1 \le i \le m$ and $1 \le j \le n$
Require: noise factor: $\delta = 0.1; \alpha = 0.3; \beta = 0.6$; sentiment
    threshold: $\gamma = 0.7$
 1: Initialization: $P(R_i) \leftarrow 0.5$; $P(U_j) \leftarrow 0.5$
 2: for all $(i, j)$ do
 3:     $S_{ij} = \frac{\#PC}{\#PC + \#NC}$ [1]
 4:     if $S_{ij} > \gamma$ then
 5:         $S_{ij} = 1$;
 6:     else
 7:         $S_{ij} = 0$;
 8:     end if
 9: end for
10: repeat
11:     In the $k$'th round:
12:     Parallelize sampling $R_i$ based on equation (2);
13:     if $P(R_i) \ge rand(0, 1)$ [2] then
14:         $R_i^{(k)} \leftarrow 1$
15:     else
16:         $R_i^{(k)} \leftarrow 0$
17:     end if
18:     Parallelize sampling $U_j$ based on equation (3);
19:     if $P(U_j) \ge rand(0, 1)$ then
20:         $U_j^{(k)} \leftarrow 1$
21:     else
22:         $U_j^{(k)} \leftarrow 0$
23:     end if
24:     $P(R_i) = \frac{\sum_k R_i^{(k)}}{k}, P(U_j) = \frac{\sum_k U_j^{(k)}}{k}$;
25: until The target distribution converges (mixing time)
26: return $P(R_i), P(U_j)$
```

Because we now sample from all products simultaneously, the time complexity drops form `O(mni)` to `O(ni)`.

Code Execution

1. Run `python parse.py [-d=<data_file>] [-p=<product_pickle_file>] [-u=<user_pickle_file>] [-s=<sentiment_pickle_file>]` to parse the input dataset (which must be stored in a file of type `.json.gz`). The data is processed and stored in three Python dictionaries - `product_pickle_file` is a mapping between each product and its reviews, `user_pickle_file` is a mapping between each user and his or her reviews, and `sentiment_pickle_file` is a mapping between each user-product tuple and the sentiment of the corresponding review. These dictionaries are stored in pickle files. The defaults are "reviews_Amazon_Office_Products_5.json.gz", "products.pkl", "users.pkl", and

"sentiments.pkl". This step only needs to be run once per dataset.

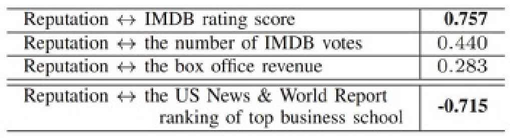
2. Run `python sample.py [-p=<product_pickle_file>] [-u=<user_pickle_file>] [-s=<sentiment_pickle_file>] [-r=<rounds>]` to run a MCMC sampling for the specified number of rounds to predict the probability that each product is "of quality". The default files are the same as parse.py. The default number of rounds is 100.
3. Run `python evaluate.py` to create a linear regression model measuring the strength of correlation between our model's output probabilities and the real Amazon 5-star score. This model could also be used as a rating-prediction score for any given user.

If you would like to run all the Python processes in a single pipeline, you can run `python pipeline.py`.

Model Evaluation

In the paper, authors compare results with two existing publicly available ranking systems: IMDB movie ranking and top business schools ranked by the US News & World Report and they find high correlations between them. They also compare reputation with number of IMDB votes and the box office revenue; however, they show weak correlation.

TABLE III: Correlations between social brand reputation and IMDB / top business school ranking.



Reputation $\leftrightarrow$ IMDB rating score	**0.757**
Reputation $\leftrightarrow$ the number of IMDB votes	0.440
Reputation $\leftrightarrow$ the box office revenue	0.283
Reputation $\leftrightarrow$ the US News & World Report ranking of top business school	**-0.715**

For us, because the dataset are so huge, we only did experiment on products of "Office Supplies" category. We compare the "reputation" with the **actual average rating**, **number of review** and **product ranking** from Amazon website. We first extract all product IDs from dataset and then build a "web scraper" to retrieve information from corresponding products.

Current Results

paramter setting: delta = 0.1 alpha = 0.3 beta = 0.6

- delta - probabilities that a positive user rates a positive product negatively; "noise factor"
- alpha - probabilities that a positive user rates a negative product positively
- beta - probabilities that a negative user rates a positive product positively

Features	Correlation	Round to converge*
Average Rating	0.0625	28
Number of review	0.0348	32
Product Ranking	-0.353	36

* Convergence is defined as the point in time where the average absolute difference between consecutive computed marginal probabilities of R becomes less than 1%

Observation and Interpretation:

1. From the result, we see that "Average Rating" and "Number of review" are not good indicators of reputation. This makes sense because many sellers in Amazon hire agency to boost their ratings and number of reviewers so they cannot represent brand reputation, this point is consistent with paper.
2. Generally, in comparison with performance of paper, our performance is quite low. This fact can be explained by following points

Paper	Our work

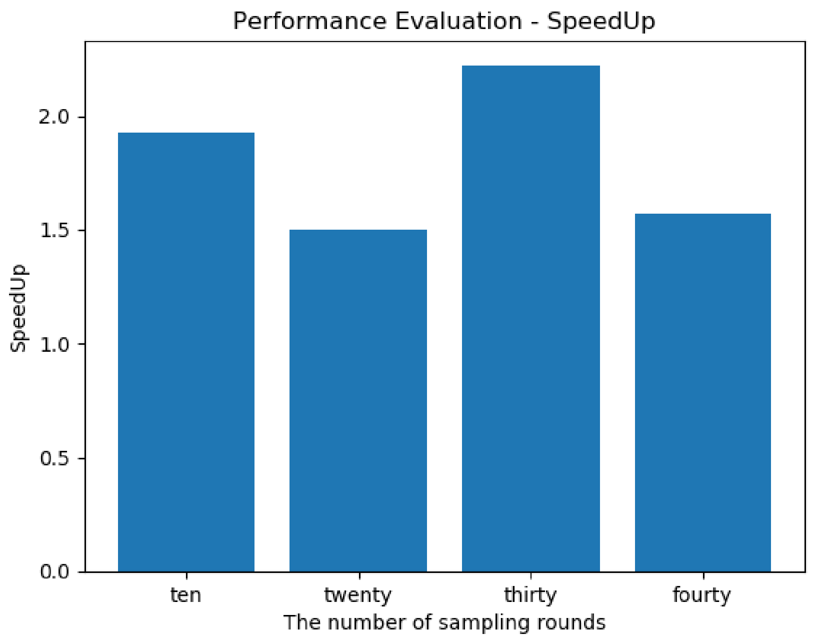
Paper	Our work
Put a lot of effort on data cleaning	No data cleaning
Manually check sentiment analysis	Use python library without manually checking
Up-to-date data	Data is collected from May 1996 to July 2014

Influence of parameters

1. For delta = 0.1, alpha = 0.3, beta = 0.6
   - 100 rounds, correlation is -0.353
2. For delta = 0.1, alpha = 0.5, beta = 0.6
   - 100 rounds, correlation is -0.193
3. For delta = 0.05, alpha = 0.3, beta = 0.6
   - 100 rounds, correlation is -0.337
4. For delta = 0.1, alpha = 0.3, beta = 0.3
   - 100 rounds, correlation is -0.201

Performance Evaluation

One of the common aspects related to MCMC- based inference algorithm with respect to performance evaluation is time complexity of sampling. To address this, we parallelize block-based MCMC due to conditional independency and compare to sequential version in terms of time taken. Speedup is a very common metric used in the field of parallelism. Speedup is determined as a ratio of time taken in a sequential algorithm to time taken in parallel algorithm with p processors. We are using 8 processors in parallel.



Usefulness and Applicability of the Model

Our application computes a "reputation" for every product given the text of reviews that users provide on products. In reality, Amazon already has a metric for this - 5-star ratings. Given the 5-star rating system with adequate spam detection, a reputation inference system may not be useful. Nevertheless, it is interesting to predict a product's reputation exclusively using text. This model could integrated into an ensemble model recommendation system, and we could expand the Markov field to include other variables to inference, such as the popularity of certain genres or the reputation of different retailers.

Checklist

Completed:

- [X] Sentiment Analysis
- [X] Implementation of Algorithm (for product ratings)
- [X] Evaluation (compare predicted 5-star rating to real Amazon rating by training a regression model)
- [X] Find Mixing Time
- [X] Add multiprocessing
- [X] Write up a report / demo (~4/18)
- [X] Create a visualization demonstrating our results
  - [X] Visualize convergence/mixing time and relate it to rounds of MCMC
  - [X] Evaluate the impact of priors (delta, alpha, and beta) on model performance ("performance" = speed of mixing time, or accuracy of model)
- [X] Bigger, better dataset
  - [X] Query real Amazon reviews to evaluate on from the Amazon API