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 "social brand reputation", a metric of how positively or negatively a "brand" is perceived 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.

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 is 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. We 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.

 Probability of R given S

Assume that there are m brands and n users. Then there are 2^(m+n) possible 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 could take O(mn * 2^(m+n)) time.

Markov Chain Monte Carlo

In this approach, we iteratively sample a positivity for every brand R and every user U, using the previous sample's positivities as well as the sentiments of the reviews. The inference equations can be seen below.

 Probability of MCMC

As we can see, each brand's probability of positivity is independent of all other brands. We must take the product of all reviews in S belonging to a particular brand in R, and we must sum up over both possible values of a brand reputation (positive or negative, so only 2 values). So, the computation of a single brand's reputation in a single iteration of MCMC should be O(n), for the possible number of users who reviewed the product. Since there are m products, the total time complexity should be O(mni), where i is the number of iterations.

Priors

 Probability of MCMC

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.
- "Gamma"=`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".

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). However, the paper provides no justificaiton for this assumption, so we ignored it in our analysis.

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).

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`.

Current Results

The below results are from a linear regression between the predicted probability that an arbitrary users would like an item, and the actual Amazon review score. This was done on 708 scraped products from the category "Office Supplies". Not all products in the dataset (which was from 2014) are still present on the Amazon website.

Below is the result after convergence (28 iterations).

delta = 0.1 alpha = 0.3 beta = 0.6

We had a correlation coefficient of `0.353`.

It takes 28 rounds of MCMC for "convergence", where convergence is defined as the point in time where the average absolute difference between consecutive computated marginal probabilities of R becomes less than 1%.

The model's predicted rating after 40 rounds of MCMC is poorly correlated with the true 5-star ratings from the Amazon website. There are multiple reasons why this could be the case. Our sentiment analysis model may have poorly predicted a review's sentiment. Our small test data sample may have been outliers in the population. Perhaps users factor in other confounding variables into their ratings that are independent of their enjoyment of the movie, such as the movie's prestige or its current Amazon rating.

Test Cases:

1. For delta = 0.1, alpha = 0.3, beta = 0.6
    - 10 rounds, correlation is 0.152
    - 20 rounds, correlation is 0.333
    - 30 rounds, correlation is 0.353
2. For delta = 0.1, alpha = 0.5, beta = 0.6
    - 10 rounds, correlation is 0.099
    - 20 rounds, correlation is 0.177
    - 30 rounds, correlation is 0.193
3. For delta = 0.05, alpha = 0.3, beta = 0.6
    - 10 rounds, correlation is 0.155
    - 20 rounds, correlation is 0.330
    - 30 rounds, correlation is 0.337
4. For delta = 0.1, alpha = 0.3, beta = 0.3
    - 10 rounds, correlation is 0.101
    - 20 rounds, correlation is 0.167
    - 30 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.


Speedup Results

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