A Bayesian and Machine Learning approach to estimating Influence Model parameters for IM-RO

Trisha Lawrence
Department of Mathematics and Statistics
University of Saskatchewan
106 Wiggins Road
Saskatoon, SK S7N 5E6, CANADA

March 9, 2018

Abstract

The rise of Online Social Networks (OSNs) has caused an insurmountable amount of interest from advertisers and researchers seeking to monopolize on its features. Researchers aim to develop strategies for determining how information is propagated among users within an OSN that is captured by diffusion or influence models. We consider the influence models for the IM-RO problem, a novel formulation to the Influence Maximization (IM) problem based on implementing Stochastic Dynamic Programming (SDP). In contrast to existing approaches involving influence spread and the theory of submodular functions, the SDP method focuses on optimizing clicks and ultimately revenue to advertisers in OSNs. Existing approaches to influence maximization have been actively researched over the past decade, with applications to multiple fields, however, our approach is a more practical variant to the original IM problem. In this paper, we provide an analysis on the influence models of the IM-RO problem by conducting experiments on synthetic and real-world datasets. We propose a Bayesian and Machine Learning approach for estimating the parameters of the influence models for the (Influence Maximization- Revenue Optimization) IM-RO problem. We present a Bayesian hierarchical model and implement the well-known Naive Bayes classifier (NBC), Decision Trees classifier (DTC) and Random Forest classifier (RFC) on three real-world datasets. Compared to previous approaches to estimating influence model parameters, our strategy has the great advantage of being directly implementable in standard software packages such as WinBUGS/OpenBUGS/JAGS and Apache Spark. We demonstrate the efficiency and usability of our methods in terms of spreading information and generating revenue for advertisers in the context of OSNs.

1 Introduction

OSNs possess features that enable them to be an effective platform for spreading information and advertising products. Viral marketing through OSNs has become an effective means by which advertising companies monopolize their revenue. For example, in 2016, Twitter’s advertising revenue totaled $545 million, an increase in 60% year-over-year [47]. This phenomenon has led researchers and inventors to improve and develop advertising strategies which generate high revenue. The IM problem, formally defined in [19] as choosing a good initial set of nodes to target in the context of influence models, has been actively researched over the past decade with its emphasis on social networks and marketing products. In [17], Hosein and Lawrence introduced a SDP model for the IM problem and recently in [24], this approach was formally defined as the IM-RO problem. The SDP approach diverted from previous approaches to influence maximization that have been based on the theory of submodular functions and adopted a novel and practical decision-making perspective. In this SDP approach, an online user clicking on an impression or advertising link was equated to purchasing a product and thus the research focused on maximizing clicks and ultimately revenue to the advertiser [17, 24]. In [24], the SDP method for the IM-RO problem was demonstrated to generate lucrative gains to advertisers; causing over an 80% increase in the expected number of clicks when evaluated on various networks. In this paper, our interests lie in the influence models for the IM-RO problem and how their parameters affect revenue optimization.
Influence models are defined by node and edge probabilities that capture real-world propagations or the spread of information amongst users within a network. Although influence models for the IM problem have been proposed in [12, 41, 19, 14, 16, 5, 7], relatively few researchers have investigated methods for determining their parameters [12, 43, 5, 16]. Compared to the limited work that has been done our proposed methods have the great advantage to be easily implementable in the standard BUGS (Bayesian inference Using Gibbs Sampling) and Apache Spark software. Consequently, avoiding the burden of implementing specific algorithms and possible coding errors. The goal of this paper is to provide efficient and easily implementable methods for determining the parameters of the Graph Influence Model (GIM) and Negative Influence Model (NIM) mentioned in [24].

From the work in [16], three types of influence models were classified for the IM problem; static models, continuous models and discrete time models. Influence models have also be classified as dependent on model parameters or on some constants. For example, the Weighted Cascade model in [19] and Trivalency model in [8] estimated \( p_{uv} \), the parameter representing the edge probability between node \( u \) and node \( v \) by randomly selecting a probability from the following set \( \{0.1, 0.01, 0.001\} \) corresponding to low, high and medium probabilities of influence. In [43], the authors propose an EM algorithm to obtain \( \kappa_{uv} \), the diffusion probability through link \( (v, w) \) in the Independent Cascade model whilst the authors in [5] proposed a weighted sampling algorithm to determine \( \theta_{w} \), the set of threshold values under the Linear Threshold model.

The significance and novelty of this paper lies in a novel decision-making perspective towards influence maximization, defined as the IM-RO problem in [24]. This perspective is achieved through implementing SDP, a method primarily used in shortest paths and resource allocation problems [2, 27, 35, 39]. Because of the significant gains achieved from implementing the SDP method, we propose influence models to further leverage on this property. We provide an analysis on the influence models for the IM-RO problem namely, the GIM and NIM and explore how their parameters affect the optimal expected number of clicks generated under the SDP method and Lawrence Degree Heuristic (LDH) proposed in [24]. This analysis enables us to identify suitable priors for the parameter of interest \( \alpha \) in our Bayesian analysis.

Our work is a novel and practical variant of the original IM problem proposed by Kempe et al. in [19]. The IM problem uses diffusion or influence models and focuses on finding a good set of nodes in order to create the maximum cascade or spread over the entire network. Though an interesting concept, our framework captures a more realistic representation of how users influence each other within an online network.

Previous work has provided formal ways of modeling the probability of a user buying a product based on his/her friends buying the product [12, 41, 19, 16]. Similarly, we employ the GIM and NIM to capture these probabilities and adopt a Bayesian and Machine Learning analysis to determine their parameters. Our proposed methods have the advantage of being easily implementable in the standard BUGS (Bayesian Inference Using Gibbs Sampling) and Apache Spark softwares. We introduce a Bayesian hierarchical model to provide a point estimate for the parameter of interest, \( \alpha \), of the GIM by the mean of the posterior distribution. In addition, we present and compare the NBC, DTC and RFC to learn and predict the parameter, \( p_0 \), a user’s initial probability of purchasing a product in the absence of influence from friends.

2 Related Work

Because the IM-RO problem was recently defined, the only influence models for IM-RO problem to date are the GIM and NIM [24] . However, studies have been conducted on the diffusion or influence models for the IM problem in [12, 41, 16]. In [12], the authors used a non linear model that described the network as a Markov random field where the probability of the \( i \)th customer purchasing a product depended on the neighbours of the customer, the product itself and a marketing action offered to the customer. They showed that these probabilities could be obtained using a continuous relaxation labeling algorithm found in [36] and Gibbs sampling [15]. Our Bayesian analysis differs from the approach in [12] because it is easily implementable in the standard BUGS (Bayesian Inference Using Gibbs Sampling), consequently, avoiding the burden of implementing a specific Gibbs algorithm and possible coding errors. The Bayesian model also has the great advantage of directly providing an estimate for the uncertainty in the parameters such as credible intervals. In addition to this, the work in [12, 41] is restricted to collaborative filtering systems while our research is suited to users within any OSN.

The authors in [12, 41, 16] proposed a machine learning approach to learn the parameters of their influence models. In [41, 12], the authors assume a naive Bayes model [11] and determine a customer’s internal probability of purchasing a product by simply counting. Similarly, a the machine learning approach is
adopted in this paper and in [16]. In [16] the authors proposed several influence models and developed machine leaning algorithms for learning the model parameters and making predictions. Their algorithms generally took no more than two scans to learn the parameters of their influence model however our implementation of machine learning algorithms is achieved much faster through Apache Spark, a framework designed to fulfill the computational requirements of massive data analysis, and manage the required algorithms [42]. Apache Spark has another advantage of offering a single framework for processing data applications such as the machine learning algorithms used in this paper and can be used with applications in both static data and streaming data.

The remainder of this paper is organized as follows. We begin by presenting the GIM and NIM for the IM-RO problem in Section (3). We introduce the methods for estimating the parameters of the GIM in Section (4). Section (5) provides experimental results for our methods on synthetic and real-world OSNs. We conclude the paper in Section (6) by summarizing the main contributions and providing directions for future work.

3 Influence Model for IM-RO

3.1 Graph Influence Model

The Graph Influence model is inspired by the IC model in [19] and as its name suggest, is greatly affected by the graphical structure of the network. The model is given by:

\[
p_k[i] = \max \left[0, \min \left[1, p_{0k} + (1 - (1 - \alpha)^y)\right]\right]
\]  

(1)

where \( p_{0k} \) represents a user’s initial probability of clicking on an impression at the start of stage \( k \), with \( p_{0k} = p_0 \) when \( k = 0 \). \( \alpha \) is an influence constant and \( y \) represents the number of users given impressions and have clicked on them. The GIM’s reliance on the network structure stems from the parameter \( f \) which represents the number of friends of user \( i \) and is the value for which a user’s probability is being raised. In these experiments, we investigated a range of values for \( \alpha \) both less than 1 and greater than 1 to determine its effect on the optimal expected number of clicks.

3.2 Negative Influence Model

The NIM supports the same parameters as the GIM with the addition of the negative influence parameters \( n \) and \( \beta \).

\[
p_k[i] = \max \left[0, \min \left[1, p_{0k} + \alpha \frac{y}{f} - \beta \frac{n}{f}\right]\right]
\]  

(2)

Here, \( \beta \) generally takes on values between 0 and 1 and \( n \) represents the number of users given impressions in stage \( k \) that have not clicked on them. In reality it does not make sense to provide a user with negative information (friends who have not clicked on impressions) as the goal is to encourage users to make purchases. However, our aim is to understand the effect of different influence models for the IM-RO problem. Influence models incorporating the natural behavior of users having a negative influence on their friends have also been presented in [9, 3].
4 Methods

4.1 Bayesian Analysis

4.1.1 The Bayesian Hierarchical Model

Let $Y_i \in \mathcal{D}$ represent the responses (number of reposts) for a POSTID, $i = 1, \ldots, N$ and defined by the distribution of the data below. The probability model for reposting a post is represented by the parameters, $p_0$, the initial probability of reposting, $R_i$ the number of times POSTID $i$ is reposted, $F$, the average number of friends associated with a particular post and $\alpha$ the influence constant under the GIM model. Figure (1) depicts a graphical representation of the Bayesian hierarchical model following [29]. The model is as follows:

$$Y_i | p_i \sim Bernoulli(p_i)$$

with

$$p_i = \min(1, p_0 + (1 - (1 - \alpha \times R_i/F))$$

Figure 1: Graphical representation with plate of Bayesian Hierarchical Model for $i = 1, \ldots, N$ POSTIDs. Rectangular nodes denote known constants, round nodes denote deterministic relationships or stochastic quantities. Stochastic dependence is represented by single-edged arrows and deterministic dependence is denoted by double-edged arrows.

A suitable choice of a prior for $\alpha$ is determined from the results of the Performance Analysis conducted in Section 5 of [24]. Values of $\alpha = 5$ generated the optimal expected number of clicks on some networks while $\alpha = 10$ generated the optimal expected number of clicks on other networks. Thus, we deduce that the network structure also influences how $\alpha$ affects optimal expected click values. Therefore, we choose the following uniform priors for $\alpha$:

$$\alpha \sim Uniform(0, 5)$$

and

$$\alpha \sim Uniform(0, 15)$$

4.1.2 MCMC method

Monte Carlo Markov Chain methods are applied in very complicated situations when the data and the parameter of interest, say $\theta$ are very high dimensional. Combining the likelihood defined by the distribution of the data in Equation 3 and the prior gives the joint posterior distribution. Although no closed-form expressions exist for the posterior distributions, simulated values from the posterior can be obtained using
a Gibbs sampler. The method is described as follows:

Suppose \( \theta \) is our parameter of interest: \( \theta = (\theta_1, ..., \theta_d) \in \Theta \subseteq \mathbb{R} \). We know that

\[
\pi(\theta | X = x) \propto \pi(\theta)f(x; \theta)
\]

but there is no practical method of computing the normalizing constant to make this into a proper density function. Therefore, we generate a pseudo random sample of observations from \( \pi(\cdot | x) \), sampling from the distribution of \( \theta \), holding \( x \) fixed. Then we can easily approximate statistics and probabilities of interests. Because a posterior distribution is available for all of the parameters, a posterior distribution is also available for \( \alpha \). Hence, JAGS [37] a software using the BUGS syntax is used to specify the Bayesian model, by drawing random numbers to simulate a sample from the posterior to form the probability density. The results for this experiment are discussed in Section 5.

### 4.2 Machine Learning Algorithms

For the Machine Learning analysis, we provide description of classification algorithms implemented to learn the mapping from inputs or feature vectors, \( \vec{z} \) to the special feature set known as the class label, \( y \) where \( y \in \{1, 2, 3, ..., C\} \) and \( C \) represents the number of classes.

#### 4.2.1 Naive Bayes

For the Naive Bayes classifier (NBC), the model is derived from Baye’s theorem which states:

\[
P(Z = z|Y = y) = \frac{P(Z = z, Y = y)}{P(Y = y)}
= \frac{P(Z = z)P(Y = y|Z = z)}{\sum_{z'} P(Z = z')P(Y = y|Z = z')}
\]

where \( Y \) and \( Z \) are two random variables and the process is implemented in two steps.

For the first step, the process involves learning the classification from a training dataset \( D \) which comprises of features whose class labels are known. The classifier given by:

\[
p(\vec{z}|y = c, \theta) = \prod_{j=1}^{D} p(z_j|y = c, \theta_j)
\]

(4)

learns the class-conditional probabilities \( P(Z_i = z_i|y = c) \) of each feature \( z_i \) given the class label \( c \).

Equation (4) hinges on the Naive Bayes assumption that the features are conditionally independent given the class label [48]. After learning the classifier, the second step, the predicting of the posterior probability of the classes is given by the NBC prediction model:

\[
P(y = c|\vec{z}, D) \propto p(y = c|D) \prod_{j=1}^{D} p(z_j|y = c, D)
\]

(5)

An estimate \( \hat{\pi}(c) \), the MLE for class \( c \) is calculated by counting as:

\[
\hat{\pi}(c) = \frac{N_c}{N}
\]
where \( N_c \) is the total number of samples in class \( c \) and \( N \) is the total number of samples. The implementation for these experiments is executed through Spark Mlib [33] with the Scala version (2.1.0) which supports a multinomial Naive Bayes as its default model parameter.

### 4.2.2 Decision Trees

A DTC classifier comprises of a hierarchical structure of nodes and directed edges which achieves classification by asking a series of questions. Although they are easy to implement and are considered more informative since they can readily identify significant attributes for further analysis [40], they are prone to overfitting. Thus an ensemble of trees tend to generate more accurate results [1, 10]. The DTC algorithm can be summarized into the following two broad steps:

1. Let \( D_r = \{(z_i, y_i), \ldots\} \) be the set of training data belonging to node \( r \). At each internal node, predictions \( p(y = 1 | z_i) \) are made over class labels conditioned on features and the question is asked ‘is the feature \( z_i \leq t_i \)’, where \( t_i \) is a threshold value. The answer to this question is a binary variable and corresponds to a descendant node.

2. After the descendant nodes are created based on each outcome, the samples in \( D_r \) are then distributed to each appropriate descendant node based on the response outcome. The algorithm continues recursively for each descendant node until all of the data is classified.

The size of the decision tree is crucial to the decision tree model since too a large a decision tree results in over-fitting and too small a decision tree results in high misclassification rates. Upon implementing DTC, it is common to grow a tree large enough and prune the tree with a set of pruning rules found in [34]. However, for these experiments, the maximum depth of the tree was set to be 5 and N-fold cross validation was executed to select and evaluate the best decision tree model under a suitable metric.

### 4.2.3 Random Forests

RFC was first introduced in [4]. The method involves growing ensembles of decision tree predictors in which each node is split using the best among a subset of predictors, which are randomly chosen at that particular node. There are numerous advantages to implementing RFC algorithms as indicated in [18]. They are robust against over-fitting, less sensitive to outlier data and have high prediction. The basis steps of a RFC classification algorithm are summarized as follows:

1. Given a training set, \( \{(z_i, y_i), \ldots\} \), sample \( D_n \) a set of bootstrap samples where \( n \) corresponds to the number of trees.

2. For each of the samples, grow or train a decision classification tree \( f(\vec{z}, D_n) \) by randomly sampling \( m \) samples at each node and choosing the best split among the \( m \) sampled predictors.

3. Make predictions, for the test data based on an approximate value \( \hat{f}(\vec{z}, D_n) \), taking a majority of votes over the classifiers.

The bootstrapping and ensemble scheme adopted by RFCs enables them to be robust enough to avoid the problem of over-fitting and hence there is no need to prune the trees. For these experiments, the maximum depth of each tree was set to be 5 and comprised of a forest of 20 trees.
5 Experiments

5.1 Experiments for the GIM and NIM

Our influence models were evaluated using three synthetic networks, SYNT1, SYNT2 and SYNT3. SYNT1 was randomly drawn by hand and SYNT2 and SYNT3 were generated from a pseudo random number generator as in [31]. All methods were written from scratch and implemented using Python version 2.7 (64 bit) on a server with 8GB of RAM and i3 Processor and an average of ten runs were taken for each experiments. The goal of these experiments was to analyze the impact of the NIM and GIM parameters on the optimal solution obtained from implementing SDP.

5.1.1 Dataset Description

We executed our experiments on SYNT1, SYNT2 and SYNT3. SYNT1 consisted of 10 nodes, SYNT2 consisted of 2,000 nodes and SYNT3 consisted of 4,500 nodes. The SDP method was implemented on SYNT1 only, due to its complexity. For SYNT2 and SYNT3, the LDH was applied and values of $p_0$, $\alpha$ and $\beta$ were varied. $\beta$ was assigned values between 0 and 1 whilst for $\alpha$ we considered values both less than and greater than 1. The results are displayed in Figures (2- 11) for experiments involving 5 impressions over 3 stages.

5.1.2 Sensitivity analysis of the GIM and NIM

For the sensitivity analysis on the GIM and NIM their parameter values were varied. Figures(2- 5) display the effect that increasing $p_0$ has on the optimal expected number of clicks when its value is increased from 0.1 to 0.8 and $\alpha$ kept constant at 0.25 on SYNT1, SYNT2 and SYNT3.

The results indicate that $p_0$ has a significant effect on the optimal expected number of clicks and as $p_0$ increases so does the optimal expected number of clicks. This result is not surprising since for both the NIM and GIM the value for the parameter $p_0$ is additive. We also note that although the optimal expected number of clicks increases steadily in both the SDP method and its heuristic, expected click values for $p_0$ greater than 0.6 increases at a greater rate for the LDH than the SDP method on all three datasets. We believe that this is due primarily to the construction of the LDH algorithm and the structure of the synthetic networks. We note that when $p_0 = 0.8$, the SDP method generates almost 5 clicks under the GIM model which demonstrates significant gains that can be achieved by selecting ideal users and suitable influence models.

Figures (6-8), indicate the optimal expected number of clicks on datasets SYNT1 and SYNT3, as $\alpha$ increases from 0 to 0.9. Figure(9) displays the results when $\alpha \geq 1$ on SYHTN1. The optimal expected number of clicks under both the GIM and NIM increases as $\alpha$ increases. This is expected as $\alpha$ is the power in which the GIM is raised and is also additive under the NIM. For a problem involving 5 impressions in 3 stages, the results in Figure (9) ensures at least 2 clicks with $p_0 = 0.25$. That is, at least 75% more than the optimal expected number of clicks generated if all the impressions had been placed in one stage. As $\alpha$ increases beyond 5, under the NIM and beyond 2 with the GIM, the optimal expected clicks remains constant. This result is primarily due to the support for $p_k[i]: 0 \leq p_k[i] \leq 1$ in both the NIM and GIM.

Figure (10) and Figure (11) indicate the the optimal expected number of clicks as $\beta$ increases. The optimal expected number of clicks decreases as values of $\beta$ increases. This is expected as in the NIM, the term including $\beta$ is being subtracted. $(p_0 + \alpha \frac{7}{2} - \beta \frac{7}{2})$. However we note that at some point, the value of the optimal expected number of clicks remains constant even though values of $\beta$ continues to increase. This result is consistently true for graphs of all sizes. (The results illustrating the effect of $\beta$ on a graph of 2,000 and 4,500 nodes are similar and omitted).

Figures (4-9) indicate that the GIM consistently outperforms NIM in generating optimal expected number of clicks. These results provide insights into the choice of influence models and role that their parameters play in maximizing the expected number of clicks and generating revenue for the IM-RO problem.
5.2 Estimation of $\alpha$

The Bayesian Hierarchical model was fitted using jags, an R interface to JAGS (Just Another Gibbs Sampler) [37], which uses Gibbs Sampling to estimate the marginal posterior distribution for the parameter of interest, $\alpha$ in the GIM. The MCMC sampling process was allowed to simulate for 10,000 iterations with a burn-in of 1000 and 100,000 iterations for a burn-in of 10,000 iterations. The process involved three chains with the $5^{th}$ iteration in each chain stored (thinning). One limitation of the MCMC method is that it does not give a clear indication of whether it has converged [38], however, convergence was assessed from the trace plots and autocorrelation plots. The effective accuracy of the chain was measured by the Monte Carlo standard error (MCSE) [22]. To ensure the accuracy of the summary statistics we provided results in which the MCSE was 5% or less, than the posterior standard deviation [30]. The results are displayed in Tables (1-5).

5.2.1 Microblog Dataset for Bayesian Analysis

With five real-world datasets; MICRO0, MICRO1, MICRO2, MICRO3 and MICRO4 consisting of continuous variables and extracted from [28] a microblog website, we executed our simulations. MICRO0 consisted of 30,078 POSTIDs and an average number of 241 friends, MICRO1 consisted 20,090 POSTIDs and an average 84 friends, MICRO2 consisted of 10,099 POSTIDs and an average of 21 friends, MICRO3, 6,183 and 33 friends and MICRO4 consisted of 5,513 POSTIDs and an average number of 37 friends. We assumed that the average number of REPOST was a good indicator of the average number of friends in each dataset. The overall goal in analyzing these datasets was to determine an estimate for the parameter $\alpha$ in the GIM by utilizing the Bayesian hierarchical model.

5.2.2 Results for Bayesian Analysis

The results of the experiments are summarized in Tables (1-5) and Figure (12). As seen in the Tables, $\alpha$ was consistently found to be between 3.16 and 3.20 with a prior $\sim \text{uniform}(0,5)$ and between 8.15 and 8.22 with a prior $\sim \text{uniform}(0,10)$ for a burn-in of 10,000 iterations. From Table (1) and Table (3), a point estimate for $\alpha$ was found to be 3.19 with 95% CI (1.47, 4.91), 3.19 with 95% CI(1.46, 4.92), 3.16 with 95% CI(1.42, 4.91), 3.18 with 95% CI(1.43, 4.91), 3.18 with 95% CI(1.44, 4.91) for MICRO0, MICRO1, MICRO2, MICRO3 and MICRO4 respectively for a burn-in of 10,000 iterations. The top part of Figure (12) shows examples of autocorrelation plots at lag $k$, $ACF(k)$, for a burn-in of 1,000 iterations on MICRO1 and MICRO2 respectively and the bottom two parts show examples of the corresponding autocorrelation plots for a burn-in of 10,000 iterations. One can see in Figure (12) from observing the autocorrelation function, that the chains are non-autocorrelated, since the autocorrelations remain particularly close to zero for large lags. Not surprisingly, this result is further emphasized in the bottom two plots of Figure (12) when the burn-in is 10,000 iterations. Because the autocorrelation is an indicator of the amount of information contained in a given number of draws from the posterior, lower autocorrelation values are ideal. This is also an indication of a high level of efficiency or mixing of the chains. The remaining autocorrelation plots were similar to those in Figure (12) and therefore not included.

The MCSE is similar to the standard error of a sample mean and thus, as the sample size increases, the standard error should also decrease. Tables (1), (2) and (4) display the MCSE for our experiments. As seen in Table (2) and Table (4), the Time Series standard error is the smallest on MICRO0, the largest dataset, but for Table (1), the standard error is the smallest on MICRO2. We believe that this is due to the insufficient number of burn-in iterations and its effect on the autocorrelation. We note the higher autocorrelated values in the top two plots of Figure (12) hence causing an increase in the standard error.

In general, we find that the Bayesian method is efficient for predicting point estimates for $\alpha$, however its value significantly affected by the choice of priors. As seen in these experiments the point estimate for alpha varies greatly when the the distribution of the prior changes. In order for us to determine how accurate our point estimates are from the true value of $\alpha$, a dataset comprising of probabilities of reposting a POST is required.
Table 1: Shows the empirical mean and standard deviation for $\alpha$ with a 1,000 burn-in, 10,000 iterations and uniform prior (0,5)

| Dataset | Mean   | SD     | Naive SE | Time Series SE |
|---------|--------|--------|----------|----------------|
| MICRO0  | 3.20348| 1.05082| 0.01357  | 0.01357        |
| MICRO1  | 3.1820 | 1.05897| 0.01369  | 0.01420        |
| MICRO2  | 3.19057| 1.05277| 0.01359  | 0.01281        |
| MICRO3  | 3.16701| 1.04781| 0.01353  | 0.01353        |
| MICRO4  | 3.13730| 1.05210| 0.01358  | 0.01358        |

Table 2: Shows the empirical mean and standard deviation for $\alpha$ with 10,000 burn-in, 100,000 iterations with uniform prior (0,5)

| Dataset | Mean   | SD     | Naive SE | Time Series SE |
|---------|--------|--------|----------|----------------|
| MICRO0  | 3.193742| 1.044780| 0.004265 | 0.004265       |
| MICRO1  | 3.18722 | 1.0461  | 0.004271 | 0.004271       |
| MICRO2  | 3.162   | 1.0611  | 0.0043   | 0.0043         |
| MICRO3  | 3.18184 | 1.0518  | 0.00429  | 0.00429        |
| MICRO4  | 3.1758  | 1.0517  | 0.00429  | 0.004294       |

Table 3: Shows the quartiles for $\alpha$ with uniform prior (0,5)

| Dataset | Update | 2.5%   | 25% | 50% | 75%  | 97.5%     |
|---------|--------|--------|-----|-----|------|-----------|
| MICRO0  | 1000   | 1.470  | 2.283| 3.210| 4.121| 4.903     |
| MICRO1  | 10000  | 1.470  | 2.288| 3.193| 4.102| 4.908     |
| MICRO2  | 10000  | 1.437  | 2.255| 3.194| 4.075| 4.919     |
| MICRO3  | 10000  | 1.457  | 2.283| 3.184| 4.089| 4.908     |
| MICRO4  | 10000  | 1.437  | 2.271| 3.243| 4.092| 4.900     |
| MICRO5  | 10000  | 1.421  | 2.235| 3.155| 4.083| 4.912     |

Table 4: Shows the empirical mean and standard deviation for $\alpha$ with 10,000 burn-in, 100,000 iterations with uniform prior (0,10),

| Dataset | Mean   | SD     | Naive SE | Time Series SE |
|---------|--------|--------|----------|----------------|
| MICRO0  | 8.21104| 3.91881| 0.016    | 0.01549        |
| MICRO1  | 8.17812| 3.91684| 0.01599  | 0.01617        |
| MICRO2  | 8.15377| 3.93972| 0.01608  | 0.01582        |
| MICRO3  | 8.18965| 3.92468| 0.01602  | 0.01564        |
| MICRO4  | 8.19130| 3.91978| 0.01600  | 0.01585        |

Table 5: Shows the quartiles for $\alpha$ with uniform prior (0,10)

| Dataset | Update | 2.5%   | 25% | 50% | 75%  | 97.5%     |
|---------|--------|--------|-----|-----|------|-----------|
| MICRO0  | 10000  | 1.734  | 4.829| 8.198| 11.583| 14.658    |
| MICRO1  | 10000  | 1.694  | 4.791| 8.199| 11.528| 14.684    |
| MICRO2  | 10000  | 1.67   | 4.742| 8.154| 11.549| 14.675    |
| MICRO3  | 10000  | 1.681  | 4.821| 8.227| 11.552| 14.655    |
| MICRO4  | 10000  | 1.682  | 4.820| 8.242| 11.527| 14.666    |
5.3 Estimation of $p_0$

We conducted experiments on three datasets, two of which were extracted from the OSN, Twitter, and obtained in [26] and the third, a microblog dataset obtained from [28]. All methods were implemented through Apache Spark MLlib package [33] with Scala version 2.1.0 on a server with 8GB of RAM and i3 Processor. An average of ten runs were taken for each experiments. The objective of these experiments was to obtain the most efficient algorithm for classifying and predicting the data in order to obtain an accurate estimate for the parameter $p_0$, a user’s initial probability of clicking on an impression, in the absence of any influence from friends. Our approach is based on modeling $p_0$ as a function, $p_0 : \mathbb{N} \to [0,1]$ and implementing the DTC, NCB and RFC algorithms in order to learn this parameter based on features from three datasets.

5.3.1 Dataset Description

The three datasets, TWITT1, TWITT2 and MICRO5 entailed nominal and binary features and a class label consisting of two outcomes or classes corresponding to a user tweeting or not tweeting a phrase. TWITT1 consisted of 16 features and 447 Instances, MICRO5 consisted of 3 features and 142,369 instances while TWITT2 consisted on 10 features and 179 instances. TWITT1 and TWITT2 were made up of binary features whilst MICRO5 consisted of nominal features. A detailed description of the features for each dataset can be found at [25]. For further analysis we divided each dataset into ten disjoint training and test sets as follows:

- 10% training and 90% test data.
- 20% training and 80% test data.
- 30% training and 70% test data.
- 40% training and 60% test data.
- 50% training and 50% test data.
- 60% training and 40% test data.
- 70% training and 30% test data.
- 80% training and 20% test data.
- 90% training and 10% test data.
5.3.2 Performance Measure

For an analysis on the performance of the DTC, NBC and RFC algorithms, the receiver operating characteristics (ROC) was used. This is a plot of $p(\hat{y} = 1 | y = 1)$, known as the true positive rate (TPR) against $p(\hat{y} = 1 | y = 0)$, the false positive rate (FPR) in a function which is a fixed threshold for a parameter $\tau$ is used.

The quality of the ROC curve is summarized by a single number using the area under curve, AUC. The ROC sensitivity ranges from 0 to 1 with higher AUC scores being preferred. In general, a more accurate classifier has a AUC value closer to 1 and very low AUC values indicate that the classifier is possibly finding a relationship with the data that is exactly the opposite than what is expected.

Another metric used to evaluate the performance of the algorithm in these experiments was accuracy, in other words analyzing whether a prediction was correct or not. This metric however can be misleading since its prediction are based primarily on the datasets used. For example a predictive model can be evaluated as being the 90% accurate simply because 90% of the data used belonged to one class. Figures (13-15) was also used to determine the accuracy of each algorithm.
Table 6: A comparison of the various algorithms in terms of metrics

| Algorithm | AUC TWITT1 | MICRO5 TWITT1 | AUC TWITT2 | MICRO5 TWITT2 | Accuracy TWITT1 | MICRO5 TWITT2 |
|-----------|-----------|---------------|-----------|---------------|----------------|---------------|
| DTC       | 0.865     | 0.755         | 0.977     | 0.989         | 0.958          | 0.989         |
| NBC       | 0.613     | 0.794         | 0.621     | 0.973         | 0.057          | 0.966         |
| RFC       | 0.907     | 0.856         | 0.977     | 0.989         | 0.958          | 0.989         |

Table 7: Average time for running algorithms (msec) over entire datasets

| Dataset  | RFC | DTC | NBC |
|----------|-----|-----|-----|
| TWITT1   | 6000| 7000| 500 |
| MICRO5   | 11000| 8000| 3000|
| TWITT2   | 8000| 8000| 2000|

5.4 Results for Machine Learning Algorithms

The results in Figure (13), Figure (14) and Figure (15) confirm the original hypothesis and work done in [6] that the RFC outperforms the NBC and DTC algorithms in terms of accuracy. Its accuracy is the best in all three datasets and it is clear that the RFC learns faster than both the DTC and NBC as the prediction accuracy of the RFC is higher than NBC and DTC when a small percentage of the training data is used, only 10%.

Table (6) shows the results for the algorithms and their respective AUC and accuracy. It is worth noting that the NBC algorithm lags considerably behind the RFC and DTC based on evaluations on both metrics. We also note that for MICRO5 when evaluated by the accuracy metric, has smaller values. We believe that this result is due to the limited number of features used in proportion to the size of the dataset. Table (6) also shows that the RFC algorithm outperforms the NBC and DTC algorithm in terms of accuracy and AUC.

Table (7) displays the running times for the NBC, DTC and RFC algorithms on all three datasets. Despite having the worst performance in terms accuracy and AUC when compared to the DTC and RFC algorithms, the running times of the NBC algorithm is considerably less than the runningtimes for the DTC and RFC as the NBC converges towards asymptotic accuracy at a faster rate.

The experimental results displayed in Table (8), indicate the average probability of predicting class 1 (the probability of tweeting) for each classifier on each dataset. We conclude that the most accurate probability for TWITT1 is 0.01, predicted by the RFC, the DTC predicts a probability of 0.004 and we believe that this value is due primarily due to over-fitting of the DTC. For MICRO5, users had a much higher probability of predicting the phrase, as the best probability was selected as 0.99 determined by the RFC algorithm due to its AUC and accuracy value. The NBC algorithm performs considerably poorly for this case predicting a value of 0. Again, we attribute this result to the limited number of features used in the MICRO5 dataset and the performance of NBC algorithm. In general, we find that the probability of retweeting a character or phrase depended significantly on the dataset.

The results demonstrate that an estimate for $p_0$ can be easily obtained by using supervised learning algorithms through Apache Spark. They also implicitly provide additional insights for advertisers to achieve considerable gains by spreading information or advertising products.

Table 8: Probability of predicting class 1, based on 100 samples

| Dataset  | RFC | DTC | NBC |
|----------|-----|-----|-----|
| TWITT1   | 0.01| 0.004| 0.02|
| MICRO5   | 0.97| 0.99 | 0.00|
| TWITT2   | 0.02| 0.01 | 0.03|
5.5 Cross Validation

N fold Cross-Validation was implemented as a technique which determined the best model for each dataset by training and testing the model on different portions of the datasets. The idea behind the technique involves splitting the dataset into N folds then for each fold \( n \in 1, 2, 3, ..., N \), the model is trained on all but the \( n \)th fold and tested on the \( n \)th fold in a robin-robin fashion. Cross validation has proven to be an effective procedure for removing the bias out of the apparent error rate and has been implemented in numerous papers [23, 21, 44, 45, 20, 13, 46].

Table (9) displays the results for the best model determined by 5 fold cross validation. The technique computes the average error over all 5 folds and uses it as a representative for the error in the test data. The best model achieved through the cross validation process was then evaluated using the AUC metric.

| Dataset | DTC | RFC |
|---------|-----|-----|
| TWITT1  | 0.865 | 0.952 |
| MICRO5  | 0.755 | 0.874 |
| TWITT2  | 0.977 | 0.977 |

The results in Table (9) indicate an evaluation of the best model using the RFC and DTC and learning algorithms achieved through 5 fold cross validation and evaluated by the AUC metric. We can conclude that the RFC algorithm has the smallest error rate when evaluated using the AUC metric. For TWITT2, the DTC and RFC were both proven to be ideal for predicting \( p_0 \) however the RFC performed consistently well in all three datasets and can be considered as an effective method for obtaining \( p_0 \).

6 Conclusion

In this paper, we have presented a novel analysis on the influence models for the IM-RO problem. The IM-RO problem was first formally defined in [24] as a novel approach to the well-known IM problem which diverted from the theory of submodular functions and focused on maximizing expected gains for the advertiser. This approach is achieved through implementing SDP and is demonstrated to have lucrative gains when evaluated with the GIM and NIM on various real and synthetic networks. We have shown how the composition of the GIM and NIM and varying their parameters affect the optimal expected number of clicks generated. Our results show that the influence models as well as the structure of the OSN play an integral role in optimizing clicks and ultimately generating revenue to the advertiser. We have also introduced a Bayesian and Machine Learning approach for estimating the parameters \( \alpha \) and \( p_0 \) of the GIM which is easily implementable in the standard BUGS and Apache Spark softwares, respectively. Results indicate that the value for \( p_0 \) relies heavily on the particular character or phrase being retweeted and that the RFC is the most efficient algorithm for computing \( p_0 \).

There are several directions for future work. First, we would like to apply the methods to real datasets for which knowledge of a user’s probability of making a purchase at specific intervals in time, is available. That is, we would like to apply our machine learning algorithms to determine \( p_{0k} \), and a user’s probability of clicking on an impression with the knowledge of whether or not their friends have clicked on the impression at all stages, \( k \). This will enable us to further explore our Bayesian analysis. Our results indicate that the point estimates of \( \alpha \) are significantly affected by the choice of priors. Hence we will be able to determine how accurate our estimates of \( \alpha \) are from its true value and make more informed decisions about the choice of priors. Second, we would like to further explore influence models for the IM-RO problem in order to improve on the optimal expected number of clicks generated. Third, we would like to investigate alternative data science techniques for obtaining the parameters of these influence models. By defining a likelihood function on the parameters of an influence model, techniques such as the EM algorithm [32] can be implemented to obtain the optimal set of parameter values.
References

[1] Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, Boosting, and Variants. Mach Learn. (1999) doi: 10.1023/A:1007515423169.

[2] Bertsekas, D. P., & Tsitsiklis, J. N., "An Analysis of Stochastic Shortest Path Problems", Mathematics of Operations Research. 16 , 580-595 (1991)

[3] Bhagat, S., Goyal, A., Lakshmanan, L.: Maximizing product adoption in social networks. In Proceedings of the 5th ACM International Conference on Web search and Data Mining. ACM. 603-612 (2012)

[4] Breiman, L. Random Forests. Machine Learning. 45 , 5-32 (2001)

[5] Cao T, Wu X., Hu T.X., Wang S.: Active learning of model parameters for influence maximization. Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2011. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg 6911 280-295 (2011)

[6] Caruana, R. & Niculescu-Mizul, A.: In Proceeding ICML ’06 Proceedings of the 23rd international conference on Machine learning. 161-168, (2006)

[7] Chakrabarti, P., Dom, B., Indyk, P.: Enhanced hypertext categorization using hyperlinks. In Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, ACM. 307-318 (1998)

[8] Chen, W., Wang, C., Wang, Y.: Scalable influence maximization for prevalent viral marketing in large scale social networks. In Proceedings of the 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM. 1029-1038. (2010)

[9] Dietterich T: Applying the weak learning framework to understand and improve C4.5. Proc. 13th International Conference on Machine Learning. 96-104 (1996)

[10] Domingos, P. & Pazzani, M.: On the optimality of the simple Bayesian classifier under zero-one loss, Mach. Learn. 29 , 103-130 (1997)

[11] Domingos, P., Richardson, M.: Mining the network value of customers. In Proceedings of the 7th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM. 57-6 (2001)

[12] Dudoit, S.,& van der Laan, M.J.: Asymptotics of cross-validated risk estimation in estimator selection and performance assessment. Statistical Methods and Applications. 2, 131-154(2005)

[13] Galhotra S., Arora A., Shourya R.: Holistic Influence Maximization: Combining Scalability and Efficiency with Opinion-Aware Models. In Proceedings of the 2016 International Conference on Management of Data. SIGMOD. 743- 758 (2016)

[14] Geman, S. & Geman, D. Stochastic relaxation, Gibbs distribution and Bayesian restoration if images, IEE Transactions On Pattern Analysis and Machine Intelligence. 6, 721- 741 (1984)

[15] Goyal, A., Bonchi, M., Lakshmanan, L.: Learning influence probabilities in social networks. In Proceedings of the third ACM international conference on Web search and data mining. 241-250 (2010)

[16] Hosein, P. & Lawrence, T.: Stochastic dynamic model for revenue optimization in social networks. In Proceedings of the 11th International Conference On Wireless and Mobile Computing, Networking and Communications.IEEE. 378-383 (2015).

[17] Horning, N.: Introduction to Decision Trees and Random Forests. American Museum of Natural History’s (2016).

[18] Kempe, D., Kleinberg, J., Tardos, E.: Maximizing the spread of influence through a social network. In Proceedings of the 9th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM. 137-146 (2003)
[20] Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In IJCAI.1137- 1145 (1995)

[21] Krstajic, D., Buturovic, L., Eleahy, D., Thomas, S.: Cross-validation pitfalls when selecting and assessing regression and classification models. Journal of Cheminformatics 6 (2014)

[22] Kruschke, J.: Doing Bayesian Data Analysis: A Tutorial with R, JAGS, and Stan, Second Edition, 186. Elsevier Inc. Netherlands (2015)

[23] Lachenbruch, P.A.: An almost unbiased method of obtaining confidence intervals for the probability of misclassification in discriminant analysis. Biometrics 23, 639-645 (1967).

[24] Lawrence, T.: Stochastic Dynamic Programming Heuristics for Influence Maximization-Revenue Optimization. arXiv:1802.10515 [stat.ML]

[25] Leskovec, J., Krause, K., Geustrin, C.: Cost effective outbreak detection in networks. In Proceedings of the 13th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM. 420-429 (2007)

[26] Leskovec, J. & Krevl, A.: SNAP Datasets: Stanford Large Network Dataset Collection. (2014) http://snap.stanford.edu/data

[27] Levi, R., Roundy, R., Shmoys, D.B. Provably near-optimal sampling-based policies for stochastic inventory control models. Math. Oper. Res. 32 821-839. (2007)

[28] Lichman, M. (2013). UCI Machine Learning Repository Irvine, CA: University of California, School of Information and Computer Science. [http://archive.ics.uci.edu/ml].

[29] Lunn, D.J., Thomas, A., Best, N. et al. Statistics and Computing (2000) 10: 325. https://doi.org/10.1023/A:1008929526011

[30] Lunn, D., Jackson, C., Best, N., Thomas, A., Spiegelhalter, D.,: The BUGS Book: A Practical Introduction to Bayesian Analysis. CRC Press.(2012)

[31] Matsumoto, M. & Nishimura, T.: Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Transactions on Modeling and Computer Simulation. 8:1 3-30 (1998)

[32] McLachlan, G., Krishnan, T. The EM Algorithm and Extensions Volume 249 of Wiley Series in Probability and Statistics. Wiley. (1996)

[33] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M.J., Zadeh, R., Zaharia, M., Talwalkar, A.: MLlib: Machine learning in apache spark. Journal of Machine Learning Research, 17 1–7 (2016).

[34] Mitchell, T. Machine Learning. McGraw-Hill Science. (1997)

[35] Nascimento, J. & Powell, W.: An Optimal Approximate Dynamic Programming Algorithm for the Economic Dispatch Problem with Grid-Level Storage, IEEE Transactions on Automatic Control (2013)

[36] Pelkowitz, L.: A continuous relaxation labeling algorithm for Markov random Fields. IEEE Transactions on Systems, Man and Cybernet. 20 709-715 (1990)

[37] Plummer, Martyn.: Jags: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling. In Proceedings of the 3rd International Workshop on Distributed Statistical Computing. DSC. 20-22 (2003)

[38] Plummer, M., Best, N., Cowles, K. and Vines, K.: CODA: convergence diagnosis and output analysis for MCMC. R News, 6(1): 7–11. (2006)

[39] Powell, W. B. : Exploration Versus Exploitation, in Approximate Dynamic Programming: Solving the Curses of Dimensionality, Second Edition, John Wiley & Sons, Inc., Hoboken, NJ, USA. (2011) doi: 10.1002/9781118029176.ch12

[40] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Francisco. (1993)
[41] Richardson, M. & Domingos, R.: Mining knowledge sharing sites for viral marketing. In Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM. 61-70. (2002).

[42] Salloum, S., Dautov, R., Chen, X., Peng, P. X., Huang, Z., J.: Big data analytics on Apache Spark. International Journal Of Data Science and Analytics. 1 145-164 (2016)

[43] Saito K., Nakano R., Kimura M. Prediction of Information Diffusion Probabilities for Independent Cascade Model. In: Lovrek I., Howlett R.J., Jain L.C. (eds) Knowledge-Based Intelligent Information and Engineering Systems. KES 2008. Lecture Notes in Computer Science, vol 5179. Springer, Berlin, Heidelberg (2008)

[44] Simon, R., Subrahmanian, J., Li, M., Menezes, S.: Using cross-validation to evaluate predictive accuracy of survival risk classifiers based on high-dimensional data. Briefings In BioInformatics May; 12(3) 203-214,(2011) doi: 10.1093/bib/bbr001

[45] Stone, M.: Cross-alidatory choice and assessment of statistical predictions. Journal of the Royal Statistical Society 36(2), 111-14 (1974)

[46] Sylvain, A., Celisse, A. A survey of cross-validation procedures for model selection. Statist. Surveys. 4 40-79 (2010) doi:10.1214/09-SS054 https://projecteuclid.org/euclid.ss/1268143839

[47] Twitter Reports Third Quarter 2015 Results. http://files.shareholder.com/downloads/AMDA-2F526X/1043842696x0x856832/2812531C-1552-47D9-9FBB-ECAFEF5172AE/2015_Q3_Earnings_press_release_CS.pdf(2015). Accessed 29 November 2016

[48] Witten I. H. W & Eibe, F. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, (1999)