IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 16, 2019, accepted February 21, 2019, date of publication February 25, 2019, date of current version March 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2901347

A Parallel Community Detection in Multi-Modal Social Network With Apache Spark

YOON-SIK CHO, (Member, IEEE)
Department of Data Science, Sejong University, Seoul 05006, South Korea
e-mail: yscho@sejong.ac.kr

This work was supported in part by the National Research Foundation of Korea through the Korea Government (MSIP) under grant 2018R1C1B5045931 and in part by the Sejong University.

ABSTRACT A constrained latent space model (CLSM) infers community membership based on two modalities in multi-modal social network data: network topology and node attributes. In this paper, we extend our previous model, CLSM in two ways. First, we introduce the Spark implementation of CLSM for parallel computation by fitting the inference algorithm into the map-reduce framework. Second, we consider user reputation besides the network homophily, which also affects social interactions between users. We test CLSM and its extension on two real-world problems: understanding link and user attributes in the location-based social network, and a review-trust network. Our proposed models in Spark can be easily deployed on commercial cloud services, such as Google cloud or Amazon web service and find latent community membership in large-scale datasets. We perform extensive experiments on real-world datasets and show how CLSM extension improves our previous model. We also share meaningful insights we discovered with the datasets.

INDEX TERMS Apache Spark, community detection, latent Dirichelet allocation, mixed-membership stochastic blockmodels.

I. INTRODUCTION

Many real-world phenomena can be interpreted as complex networks, and researchers from various fields have been studying diverse real-world systems through graphs. Often the relational data can be represented as a collection of edges in a graph, where the edge reflects the relationship between the vertices. In social network analysis, for instance, social network users can be represented as vertices and the friendship between the users can be represented as edges. Many applications, such as recommendation systems [1]–[4], personalized systems [5]–[7], targeted advertising [8], [9] is based on the studies in social graphs.

One of the most active topics in this field is finding the communities [10]–[15] among the nodes in a network graph, where the nodes within the same community probably share common properties leading the links between the nodes densely connected. Identifying the communities in a network graph is crucial not only because it well summarizes the whole graph, but also because it allows useful applications in machine learning literature such as classifications, recommendation or ranking systems.

The associate editor coordinating the review of this manuscript and approving it for publication was Feng Lin.

Until recently, researchers paid less attention to finding out what the inferred communities represent. This could be mainly due to the limitation of datasets, where often only network structures were available. For clustering purposes, or for predictive tasks such as link predictions [16]–[20], identifying and understanding the inferred community is unnecessary. Instead, the focus of previous studies are finding network features, compute the feature scores of the pairs and predict how likely the nodes within a given pair are to establish a link. While this can build practical applications, it might be limited when correlating the results to other studies or other information.

The recent proliferation of multi-modal social network datasets allows a better understanding of the communities inferred from network structures. The user attributes such as tweeted words, selected tags, or the checked-in places can be collected within a given cluster, and help interpretation of the communities. Moreover, combining the two modalities of network topology and node attributes leads to a more powerful model. It significantly achieves better predictions by fully using the information from multiple sources, unlike the conventional predictive models. It can also perform predictions without any information from the same source but solely using the information from the other source.

Motivated by the abundance of large-scale multi-modal social network data, and the popularity of cloud computing, we develop an algorithm that detects communities combining network structure and additional information of nodes in a distributed fashion. Specifically, we rely on Apache Spark cloud platform that is suitable for iterative machine learning and parallel computation. Apache Spark overcomes the limitations of Hadoop which lacks cyclic data flow and consequently loses efficiency for iterative processing. As Spark accesses data from RAM instead of disk, it significantly improves the execution time of iterative algorithms. Another advantage of using Apache Spark for distributed computation comes from its own data abstraction called resilient distributed dataset, RDD, which is a collection of data items that can be easily distributed over the nodes of the clusters.

In this paper, we introduce Spark implementation of Constrained Latent Space Model (CLSM) and its extension. CLSM has been introduced in our previous work [21]. CLSM augments the generative process by introducing a set of constraints that allows to account for stronger correlations between user attributes and social networks, which avoids the fragmentation of the joint latent space. Overall, CLSM achieves significant improvement in performance over the others in various learning scenarios as shown in our previous work. Comparing CLSM to other previous baselines is not the focus of this work; instead, we mainly focus on redesigning the implementation of CLSM for parallel, distributed computing. We also extend our previous model CLSM as we test CLSM on large-scale datasets. Our proposed extension introduced in this work further achieves superior results on prediction tasks. Our primary contributions can be summarized as follows:

- We design a general framework called Constrained Latent Space Model (CLSM) in a distributed fashion with Apache Spark for cloud computing.
- We perform experiments on large-scale datasets that augment social network information with various data modalities including texts, purchase reviews, and check-in data.
- We extend our previous model, CLSM, and show it achieves better performance in both prediction tasks.
- We conduct a case study on the location-based social network discussing the unique insights Spark-based CLSM and its extension yields.

The paper is organized as follows: The following section summarizes the related literature. Section III introduces the background on latent space models followed by Section IV which presents our model. A rigorous experimental evaluation of our model is provided in Section V with a case study using large-scale datasets on Apache Spark. Finally, Section VI summarizes our contribution and future plans.

II. RELATED WORK

With the increasing popularity of social network services, community detection for large-scale social network data has gained a lot of attention in recent years. Moreover, the availability of abundant data including user attributes and behaviors enables various applications based on the studies. However, few researchers have studied the two modalities simultaneously through a unified framework. Most previous community detection methods focused on finding communities based on network topology.

Mixed Membership Stochastic Blockmodel(MMSB) [22] is one of the successful community detection algorithms that performs soft clustering of the nodes based on the observed network. MMSB is a probabilistic model that allows each node of a network to exhibit a mixture of communities capturing the overlapping communities, which is the main significance over other models. As shown in Fig. 1, MMSB infers the communities for each node based on the edge list. In this example, nodes in a network graph correspond to movie actors from IMDb, and an edge is generated when two actors in a pair appear in a same movie. For visualization, each node has been colored in RGB based on their membership vectors of size 3. Some nodes such as Jackie Chan and Jean Reno exhibits multiple communities with *mixed* colors.

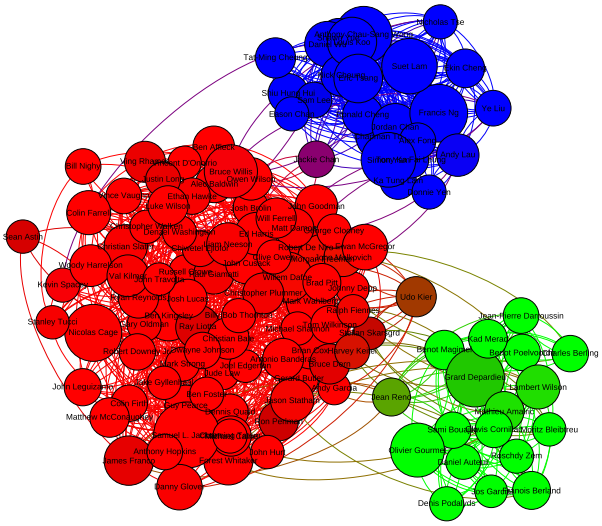


FIGURE 1. Three main inferred communities of co-starring movie actors. Colors represent the inferred communities through MMSB. Based on the name of the actors, community with red color, green color, and blue color corresponds to Hollywood, Europe, and Asia respectively.

Another line of work related to our's is the user behavior modeling, where probabilistic topic models such as LDA [23] have been applied to infer users' preferences. LDA has been initially proposed in text mining to discover the mixture of topics in a document by inferring the hidden topic structure from the observed documents using the posterior distribution. LDA can be easily adapted to other kinds of observations not limited to words. It can be adapted to many different types of data, including survey data, user preferences, audio, and social networks [24].

Since its introduction, LDA has been extended in many ways. One of the fields is the joint modeling of text and citations in the topic modeling framework. The Pairwise-Link-LDA [25] model combines the ideas of LDA and

Mixed Membership Block Stochastic Models and allows modeling the link structure between the documents. Unfortunately, Pairwise-Link-LDA often fails to exploit the synergies between different modalities, as it tends to fragment the latent space into non-overlapping (or weakly overlapping) regions corresponding exclusively to either words or to links. This fragmentation is due to the loose dependency between the words and links which are connected only through a common latent topic distribution. To address this shortcoming, the Relational Topic Model (RTM) [26] was designed to impose additional constraints on the generative model that would require stronger correlations between links and words.

Relational topic model (RTM) extends the idea of LDA by incorporating the links between the documents. RTM assumes that each document is modeled as in LDA and that the links between documents depend on the distance between their topic distributions. This model can also be adapted to other fields such as social network modeling, and perhaps RTM is the closest to our model that projects the users' mixed membership distribution on latent topic space fully using the two modalities. For instance, when used on social network analysis, RTM can infer user interest based on its attributes and interactions. However, the generative process for link generation in RTM is simplified, and lacks the flexibility to capture the multi-faceted interactions.

III. BACKGROUND

Let $y(n_1, n_2) \in \{0, 1\}$ be an indicator of a link from node n_1 to node n_2, where n_1, n_2 are from a set of N nodes. As many social networks based on friendship are undirected, we will focus on undirected networks where $y(n_1, n_2) = y(n_2, n_1)$. This relational data forms a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of vertices (or nodes), and $\mathcal{E}$ is a set of edges (or links). In addition to the network information, we also have the information of node attributes. These attributes can be the collection of the node behaviors or node characteristics. For instance, the node attribute we consider in the location-based social network is the places the users visited, and the node attribute in the review-trust network is the collection of selected interests in product categories.

Our proposed model in this paper extends the work of our previous research [21] in two ways. First and foremost, we propose a parallel community detection algorithm for multi-modal social network data, where the iterative update equations are implemented in the Spark framework. Furthermore, we extend our previous model by considering the reputation of each user that can affect social interactions. We defer the details of the model extension to the section IV-B.

A. CONSTRAINED LATENT SPACE MODEL

In this section, we review the Constrained Latent Space Model (CLSM) [21], which simultaneously describes social network information and user attribute data using a latent space representation. CLSM relies on variational method [27] and approximates the posterior in tractable structure. The update equations from variational inference are provided in Appendix. Following the same update equations, we redesign the implementation of CLSM for parallel processing in Apache Spark framework.

TABLE 1. Summary of notations.

	Notation	Description
dimension	N	total number of nodes
	K	total number of communities (topics)
	V	total number of attributes
observed	$y(n_1, n_2)$	link indicator from node n_1 to n_2
	$\mathcal{W}_n$	set of attributes of node n
latent	θ_n	community (topic) distribution

1) GENERATIVE PROCESS OF CLSM

The generative process of CLSM consists of two modules: network generation and attribute generation, where the network topology is generated based on MMSB framework, and the attributes are generated by reusing the membership indicator vectors. To elaborate, the topic indicator is sampled from the set of sampled membership indicator that has been used when forming links. Namely, when generating attributes for the given user, we reuse the indicator variables that were used to generate links for the same user. This sampling method differentiates the CSLM from the earlier work which samples the topic indicator from the topic distribution (or membership distribution). Chang *et al.* believe that the generative process of Pairwise-Link-LDA model ends to fragment the latent space into non-overlapping (or weakly overlapping) regions corresponding exclusively to either user attributes or to the social network. Below we summarize the generative process of CLSM.

- **Network Generation:**
  1) For each community k, let β_k be the community strength, which is the probability of two nodes from community k forming a link.
  2) For each node n, sample the $K \times 1$ membership vector $\boldsymbol{\theta}_n \sim \text{Dirichlet}(\boldsymbol{\alpha})$.
  3) For each node n, initialize an empty multiset of indicator variables, $\mathbb{Z}_n = \emptyset$.
  4) For each pair (n_1, n_2)
     a) Draw membership indicator vectors $\mathbf{z}_{n_1 \to n_2} \sim \text{Mult}(\boldsymbol{\theta}_{n_1})$ $\mathbf{z}_{n_1 \leftarrow n_2} \sim \text{Mult}(\boldsymbol{\theta}_{n_2})$.
     b) Sample the pair interaction $y(n_1, n_2) \sim \text{Bernoulli}(\mathbf{z}_{n_1 \to n_2}^{\top} \mathsf{B} \mathbf{z}_{n_1 \leftarrow n_2})$, where $\mathsf{B} = \text{diag}(\beta_1, \beta_2, \ldots, \beta_K) + \epsilon(\mathbb{1}\mathbb{1}^{\top} - I)$, and ϵ is a small regularization parameter for off-diagonal components.
     c) Augment the corresponding indicator multisets (allowing repetitions) $\mathbb{Z}_{n_1} \to \mathbb{Z}_{n_1} \cup \{\mathbf{z}_{n_1 \to n_2}\}$ $\mathbb{Z}_{n_2} \to \mathbb{Z}_{n_2} \cup \{\mathbf{z}_{n_1 \leftarrow n_2}\}$.

- **Behavior Generation:**
  Let M_n be the total number of selections[1] of user n from a (attributes) set $\mathcal{V} = \{1, ..., V\}$.
  1) For each topic k, sample the attribute distribution $\boldsymbol{\omega}_k \sim \text{Dirichlet}(\boldsymbol{\kappa})$, where κ is a hyperparameter.
  2) For each selection $\mathbf{w}_{n,m}$ of user n, where $m \in \{1, ..., M_n\}$,
     a) Sample an index $\mathbf{c}_n^m \sim \text{Unif}(\{1, ..., \text{size}(\mathbb{Z}_n)\})$, and let $\hat{\mathbf{z}}$ be the topic indicator vector corresponding to $\mathbf{c}_n^m$.
     b) Sample a selection $\mathbf{w}_{n,m} \sim \text{Mult}(\boldsymbol{\omega}_i)$, where i is the index of $\hat{\mathbf{z}}$'s non-zero component.

IV. PARALLEL CLSM

One of the main advantages of using Apache Spark is the parallel computation of large-scale datasets. Spark has its own data abstraction called Resilient Distributed Dataset (RDD). Data can be easily loaded into RDD, which is later partitioned and distributed over the nodes of a cluster. Implementing the CLSM, and its extension in Spark requires extensive use of RDDs. Fig. 2 shows an overview of the RDDs used throughout the process. It shows how the RDD is created from the raw data, transformed into other forms.

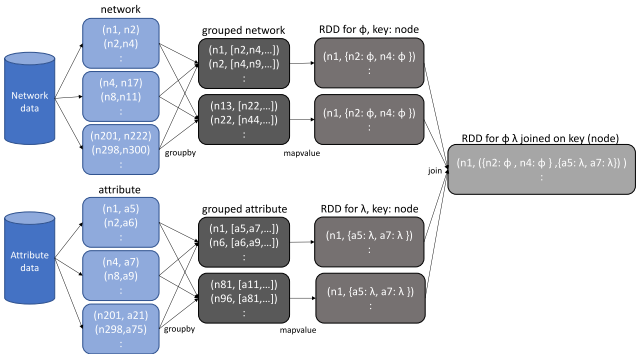


FIGURE 2. RDD concept diagram of CLSM.

Our variational inference updates will be using the final form of RDD (shown in the most right). Our update equations in Spark implementation follows the update equations in Appendix, and has been specifically redesigned in RDD format. The overall step is summarized in Algorithm 1.

A. PARALLEL PROCESS FOR VARIATIONAL INFERENCE

Spark provides map, filter join functions, and existing RDDs can be easily transformed to other form. Our snippet code provided in this section is written in PySpark[2]. Specifically, we introduce how to create RDD from data loading, to handle RDDs in an efficient manner, and to construct a new RDD from a previous one. One key factor that differentiates RDDs in Spark from other variables in conventional languages is the immutability. Hence, every iterative step requires constructing a new RDD from a previous one.

[1]For the purposes of data generation M_n can be sampled from, say, a Poisson distribution. This is not relevant for inference, however, where M_n is specified in the data.

[2]PySpark is the Python API for Spark

Algorithm 1 CLSM Parallel Updates Using Spark

```
Data: edge list, attribute list
Result: community distribution for each user
data loading and preprocessing;
create RDD_edge ;
create RDD_attribute;
join RDD_edge, RDD_attribute into RDD_multi;
while not converged do
    update {ϕ} in RDD_multi;
    update {γ} in RDD_multi and broadcast;
    update {λ} in RDD_multi;
    update model parameters with reduce;
end
obtain community distribution in {γ} in RDD_multi;
```

1) DATA LOADING AND PREPROCESSING

The provided code below assumes the dataset to follow the similar format from Gowalla dataset which can be obtained from [28]. For inference and learning, we only use user id and venue id as node id and attribute id; other information which is unique to the dataset such as time, the location with geo-coordinates has been ignored.

```
schema_attribute = StructType([
    StructField("node", IntegerType(), True),
    StructField("att_id", StringType(), True)])

df_attribute = sqlContext.read.csv(file_path ,sep='\t',schema=schema_attribute)

schema_network = StructType([
   StructField("nodefrom", IntegerType(), True),
   StructField("nodeto", IntegerType(), True)])

df_network = sqlContext.read.csv(file_path ,sep='\t',schema=schema_network)

groupedDF_attribute = df_attribute.groupby("node").agg(F.collect_list("att_id"))
groupedDF_network = df_network.groupby("nodefrom").agg(F.collect_list("nodeto"))
```

Spark also provides SQL and DataFrames as built-in libraries, and we rely on these libraries for data reading and pre-processings. Afterward, we aggregate all the selected attributes (or visited places for Gowalla dataset) per each node to form grouped DataFrame for user attributes and aggregate all the linked nodes per each node to form grouped DataFrame for edge list. Each row consists of a node and its selected attributes in a list format and a list of its friends. When a certain attribute is selected multiple times by the user, the attribute id will appear multiple times in the list accordingly. This aggregation respect to each node is required for creating an RDD,where the key (in key-value PairRdd) becomes the id of each node.

2) TRANSFERRING DATA INTO RDD AND CREATING RDDs FOR VARIATIONAL PARAMETERS

The DataFrame can be easily converted to RDD using the rdd and map function.

```
networkRdd = groupedDf_network.rdd \
                    .map(lambda x:((x['nodefrom'], x['collect_list(nodeto)'])) \
                    .repartition(partitionSize)
phiRdd = networkRdd \
     .mapValues(lambda x: dict(zip(x,np.random.dirichlet(np.ones(K)*10,len(x)))))
```

The networkRDD is an RDD of tuples consisting of key and value, where the key corresponds to the node, and the value corresponds to a list of friends of a given node. The phiRDD will be handling the update formula in Equation 7, and can be created from networkRDD using the mapValue function. mapValues is preferred over map function because it operates on the value only instead of the entire record. Besides, mapValues preserves any partitioner set on the RDD. The last line initializes the phiRDD and keeps it in a dictionary form, where the key of a dictionary is the id of its friend and value is the corresponding $\boldsymbol{\phi}_{n_1 \to n_2}$. In other words, n_1 and n_2 in $\boldsymbol{\phi}_{n_1 \to n_2, k}$ are key from the RDD and key from the dictionary in corresponding value respectively. This phiRDD will be used both for CLSM following the updates in Equation 7.

```
from pyspark.ml.feature import CountVectorizer

attributeSize = df_attribute.select('att_id').distinct().count()
cv = CountVectorizer(inputCol="collect_list(att_id)", \
                                outputCol="features", vocabSize=attributeSize)
cv_model = cv.fit(groupedDF_attribute)
co_occurrence = cv_model.transform(groupedDF_attribute)
attributeRdd = co_occurrence.rdd.map(lambda x: ( (x['node'], x['features'])) \
                            .repartition(partitionSize)
```

For CLSM, we create an RDD for user attributes besides the RDD that contains $\{\boldsymbol{\phi}\}$. As previously stated, in this example, we consider users ''check-ins'' in Gowalla as attributes. We rely on PySpark library for using CountVectorizer which converts a collection of unique location ID to a matrix of token counts. By using CountVectorizer, we obtain a sparse representation of the counts of check-ins, which is effective for computation efficiency. The co-occurrence matrix is then converted to RDD using the rdd.map function. The RddAttribute will be handling various kinds of attributes.

3) HANDLING VARIATIONAL LOCAL PARAMETERS

Previously, we introduced how to create an RDD that handles variational parameter $\boldsymbol{\phi}$. Besides $\boldsymbol{\phi}$, variational parameter $\boldsymbol{\lambda}$ needs to be taken care of. Each node has its own set of variational parameters $\{\boldsymbol{\phi}_{n\to.}\}$, and $\{\boldsymbol{\lambda}_{.}^{n}\}$. we collect these parameters and join over the key which is the node index, and form a new key-value pair RDD where the value is a tuple of a collection of $\{\boldsymbol{\phi}_{n\to.}\}$, and $\{\boldsymbol{\lambda}_{.}^{n}\}$. Instead of simply collecting $\{\boldsymbol{\phi}_{n\to.}\}$ and $\{\boldsymbol{\lambda}_{.}^{n}\}$ as a set per a given node n, we collect them as a dictionary, which allows efficient computation. When updating the variational variables for each neighbor of given node n, it's efficient to use the dictionary as each variational parameters for the linked node can be easily looked up.

```
totalRdd = phiRdd  \
          .join(attributeRdd,partitionSize) \
          .map(lambda x: (x[0], (x[1][0],   \
             dict( zip(zip(x[1][1].indices,x[1][1].values), \
               [(dict(zip(x[1][0].keys() ,np.ones(len(x[1][0].keys()))/K ) ),0)]\
                 *len(x[1][1].indices)) ) ,x[0])))\
          .repartition(partitionSize)
```

4) UPDATING VARIATIONAL LOCAL PARAMETERS

Two sets of variational parameters $\{\boldsymbol{\phi}_{n\to.}\}$, $\{\boldsymbol{\lambda}_{.}^{n}\}$ are combined into an RDD as a tuple shown previously. These two variational parameters are handled in single RDD. Due to the dependence between $\{\boldsymbol{\phi}_{n\to.}\}$, $\{\boldsymbol{\lambda}_{.}^{n}\}$ as shown in Equation 5, 6, and 11, it is efficient to group a set of variational parameters under the same user id. This allows each user id to keep its variational parameters in single element reducing unnecessary lookups. Our update equations implemented in Spark are based on this approach.

For CLSM on an undirected network, the update of variational multinomial parameters can be expressed as Equation 7. Equation 7 shows that the update equation also depends on the set of variational parameters of n_2. Our proposed implementation uses collectAsMap() function to create dictionary in master node. This dictionary is later broadcasted to worker nodes and variational parameters of n_2 can be easily referenced when updating the variational parameters of n_1. The attribute [0], and attribute [1] below keeps the index of attributes and number of repeats respectively.

```
temp = totalRdd.mapValues(lambda x: transformMap(x, roleAttribute, K))
phiAttributeMap = sc.broadcast(temp.collectAsMap())

def transformMap(total, RM,K):
    nl = total[2]
    neighbors = total[0]
    attributes = total[1]
    value_dict = {}
    for n2 in neighbors.keys():
        sum1 = 0
        for attribute,val in attributes.items():
            sum1+=val[0].get(n2)*attribute[1]*np.log(RM.value[:,attribute[0]])
        value_dict[n2] = sum1
    return value_dict
```

5) UPDATING VARIATIONAL GLOBAL PARAMETERS

Once all the local parameters are updated, global parameters need to be updated. In Equation 4, $\{\boldsymbol{\gamma}\}$ are updated by summing up the corresponding $\{\boldsymbol{\phi}\}$. We adopt an approximation technique [29] by assuming that the parameters $\boldsymbol{\phi}_{n\to.}$ for non-links of a given node n can be replaced by a single mean-field parameter. These mean-filed parameters for every network node can be computed using the lambda function below. The global parameters $\{\boldsymbol{\gamma}\}$ are being used in the variational local parameter updates in Equation 7 when computing $\mathbb{E}_q[\log p(\boldsymbol{\theta})]$. We found that by broadcasting the global parameters to worker nodes can significantly reduce the computation time.

```
phiMeanRdd = totalRdd \
          .mapValues(lambda x: ( sum(x[0].values())/sum(sum(x[0].values())) ) )
...
phiMeanMap = sc.broadcast(phiMeanRdd.collectAsMap())
```

6) UPDATING VARIATIONAL MODEL PARAMETERS

Each variational global parameter is updated after the local updates. This update requires the summation of every value from the RDD. In this section, we present the updates of $\boldsymbol{\tau}_1$, and $\boldsymbol{\tau}_0$, where the local variational parameters are summed over all pairs of interest.

```
(t_1,t_0) = updateBlock(totalRdd, phiMeanRdd)
tau_one = sc.broadcast(t_1+eta_one)
tau_zero = sc.broadcast(t_0+eta_zero)
...
omega= updateRoleVenue(phiAttriRdd,K,V,smoothing_ratio,pop_BC)
roleAttribute = sc.broadcast(omega)
...
def updateBlock(totalRdd, phiMeanRdd):
    t_1 = totalRdd.mapValues(lambda x: (sum(x[0].values()))).values().sum()
    phimeansqresum = phiMeanRdd.mapValues(lambda x: pow(x,2)).values().sum()
    phimeansumsqure = pow(phiMeanRdd.values().sum(),2)
    t_0 = (phimeansumsqure - phimeansqresum)/2 - t_1
    return (t_1, t_0)

def updateRoleAttribute(phiAttriRdd,K,V,smoothing_ratio,pop):
    omegaSum = phiAttriRdd.mapValues(lambda x: toRoleAtt(x,K,V)).values().sum()
    omegaNormalized = normalize(omegaSum, axis=1, norm='l1')
    updatedOmega = (omegaNormalized*(1-smoothing_ratio) +
                    np.array([np.array(pop.value)]*K)*(smoothing_ratio))
    return updatedOmega
```

The variational model parameter $\boldsymbol{\tau}_1, \boldsymbol{\tau}_0$ requires summation of $\{\boldsymbol{\phi}\}$ for existing links and summation of $\{\boldsymbol{\phi}\}$ for all the pairs with no links respectively. Since most of the graphs are sparse, obtaining $\boldsymbol{\tau}_0$ is relatively expensive in computation compared to obtaining $\boldsymbol{\tau}_1$. Following the similar approach from [30], we can efficiently obtain summation for non-links without accessing all of the non-link pairs, which is reflected on the code above.

B. CLSM WITH USER REPUTATION

In this section, we first elaborate on the intuition of user reputation and propose an approach that can capture user reputation by adding a generative process for link generation. Links in social networks can be generated by various factors. One of the element widely considered is the network homophily, which generative process of link generation in CLSM is based on. Another strong element that affects the creation of a link is the user reputation, which has been omitted in CLSM. Users with high social reputation usually attract more users and establish relationships accordingly. For instance, celebrities tend to have more followers on Twitter, Yelp elite users attracts more followers or readers than average reviewers. To this end, we incorporate the user reputation into our model.

Whenever a link exists between two users, our previous model, CLSM enforces the sampled membership vector to be the same between two users. This approach is limited when we consider the exogenous effects such as user reputations. By incorporating the parameters that measure how the given user attracts other users out of network homophily, we can capture the links generated from user reputation. Throughout this study, we name this model Reputation CLSM or RECLSM. The generative process for the network is formulated below:

- **Network Generation:**
  1) For a given node n, sample the 2×1 vector $\boldsymbol{\pi}_n \sim \text{Beta}(\boldsymbol{\alpha}_r)$.
  2) For all the possible pairs with node n, draw the pair interaction triggered by node reputation $r(m,n) \sim \text{Bernoulli}(\boldsymbol{\pi}_{n,1})$,
     a) If $r(m,n) = 1$, then $y(m,n) = 1$.
     b) When $r(m,n) = 0$, generate edge based on membership vectors from MMSB.

The rest of the generative process follows the generative process of CLSM. In a way, RECLSM is 2-staged model, where the first stage considers user reputation and the second stage only focuses on the community membership if no exogenous effect exists.

V. EXPERIMENTS

We evaluate our model from various angles. First, we examine the scalability of our model implemented in Spark. Second, we evaluate the predictive performances of CLSM and its extension; We compare the performances between CLSM and RECLSM in this study as we have already shown how CLSM outperforms previous models in our previous work [21]. Third, we perform a case study by visualizing our results and share meaningful insights we discovered from a selected dataset.

In the following, we present our real-world datasets used in the experiments. Two different types of multi-modal social network data have been used, where each dataset contains a social graph and the attributes of social actors. In one dataset from Gowalla we take the attributes as off-line user behaviors; in the other dataset, which is from Epinions, we take the attributes as user's selected skills (or interests). Another difference between the attributes in the two datasets is the repetition of attributes. Users in Gowalla can check-in to the same places multiple times, while users in Epinions can select different skills for their skillsets. The details of the datasets are presented in section V-A. We use these datasets, and evaluate our models.

A. DESCRIPTION OF THE DATASETS

We use two different types of multi-modal social network datasets obtained from two platforms: Gowalla and Epinions. Gowalla is a Location-Based Social Network (LBSN), where users can check-in to places sharing with online friends. Users can also make friendships with other users online. Epinions is a review website, where users can leave reviews on various consumer items. Epinions also allows users to establish a link with other users by expressing trust or distrust. The summary of each dataset is provided in Table 2.

TABLE 2. Summary of datasets.

	Gowalla	Epinions
Number of nodes (N)	196,591	31,322
Number of total attributes (V)	1,280,969	587
Number of total links	950,327	174,928 (positive)
Number of total selection	6,442,892	135,226

a: GOWALLA DATASET

One of the distinctive features of Gowalla dataset is the *check-in* functionality. While most social network services focus on online activities, LBSN allows users to share off-line activities. This significant feature attracted a lot of researchers from various domains. Gowalla dataset is

a multi-modal social network data consisting of two components: friendship and check-ins. Friendship data is a conventional social graph provided in the form of edge list that describes the friendship between the pairs. Check-in data contains detailed information of each check-in with the user id, location id, timestamp, and the GPS-coordinate. As we intend to illustrate the flexibility of CLSM and RECLSM that adapt well within various multi-modal social network datasets, we ignore the timestamp and the coordinate[3].

b: EPINIONS DATASET

Epinions was one of the most active consumer review site, which ended their service in 2014. Epinions dataset has been widely used in various areas including trust-network [31], [32] and recommender system [2], [33]–[35]. Users in Epinions can share their experiences on various consumer items, and can also trust /distrust other users. Users can also announce their skills in their profiles. The selected skills for each user become binary representations over the attributes, which capture their presence (0 or 1). Dataset from [36] contains these features, which can be incorporated in CLSM and RECLSM. We take the trust relationship as the link information and the skills for each user for the attributes.

B. EXPERIMENTAL RESULTS AND ANALYSIS

Throughout our experiments, n1-highmem-8 with 8 vCPUs and 52GB of memory Google Cloud platform instance is used as a worker node. The instance is configured with Spark 2.3.1. Our Spark implementation reads the edge list and the attribute list directly from Google Cloud Storage into RDD following the PySpark code in Section IV-A1. In our experiments, we evaluate the scalability of our model and its predictive performances using the two datasets. Specifically, when evaluating the scalability, we examine the execution time of each iteration with respect to the number of communities, the number of worker nodes in a cluster, and the size of V (number of total attributes). For evaluating the performance of our model, we show the performance of prediction tasks and also visualize the communities we found from the two datasets.

1) MODEL SCALABILITY

In this section, we examine the model scalability respect to various factors. We found the execution time of CLSM and RECLSM is similar each other, and present the results from CLSM; RECLSM is the extension of CLSM, which achieves better performance in prediction tasks.

a: NUMBER OF COMMUNITIES

We first examine the model scalability respect to the community size K. The experiments with real-world datasets have been performed on Spark cluster consisting of 5-nodes (1 master - 4 workers) with varying K. We measure the execution time in each iteration when updating the model until it converges. The average execution time is computed for each case when K is set to 4, 16, 32, 64, and 100. As shown in Figure 3, we observe the execution time for each iteration grows as the dimension K grows. This is expected considering the model complexity which is linear to the size of K. However, the growth rate is faster than linear. This can be mainly due to the computational overhead such as communication between the workers or the broadcasting of variables. We further investigate the overload issue in the following section.

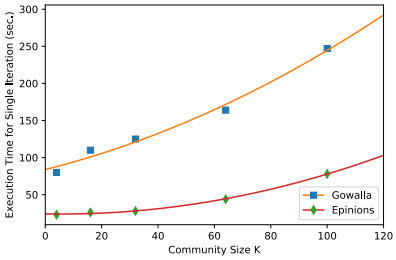


FIGURE 3. Execution time respect to the size of the community K. A cluster consists of 4 workers each with 8vCPUs (n1-highmem-8).

b: COMPUTATIONAL BOTTLENECK

To further study the computational overhead, we examine the case with the longest execution time in the previous experiment (Gowalla dataset with $K = 100$). We break-down the execution time into each process in a single iteration. Figure 4 shows that the 'model parameter update of ω' step is the bottleneck for overall update equations within an iteration. We believe this is due to the summation of the high dimensional matrices ($K \times V$) of the attributes, where the size V corresponds to the total number of unique places in Gowalla dataset.

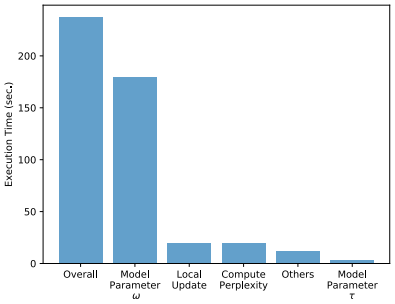


FIGURE 4. Break-down of average execution time of each iteration on Gowalla dataset with K fixed to 100.

To better understand the issue, we generate a set of synthetic datasets with various V selecting from 100, 500, 1000, 5000, 10000, and 50000 for 10000 nodes with K fixed to 100. The set of synthetic datasets allows a fair comparison with varying V, as the number of edges and the number of selected attributes can be controlled to be kept under the level throughout the experiment. Further

[3]coordinates will be used only for visualizing purposes.

inspection revealed that when the dimension V increases, the required computation time grows faster than linear. Note that the x-axis in Figure 5 is in logarithmic scale. We believe the high growth-rate is mainly due to the communication overhead between workers when summing the high dimensional matrices. The current implementation of model parameter ω update requires the summation of the full RDDs which contains selected attributes from all the nodes. One can achieve higher efficiency by only focusing on top-k attributes considering the dimension is unnecessary enormous with attributes seldom selected. Another approach to consider can be the stochastic variational inference [37], which we leave for future work.

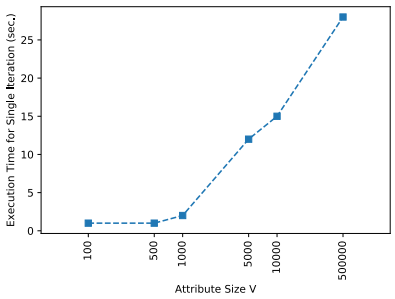


FIGURE 5. Average execution time for a single iteration with various size V. Synthetic datasets with $N = 10000$, and $K = 100$ have been used for the given experiments.

c: FAULT TOLERANCE

Many machine learning algorithms including our's requires iterative updates. As shown in Algorithm 1, each RDD depends on previous RDDs repeatedly until the model converges. In Spark, when a node fails, the involved RDD requires recomputing from the beginning by tracing all the previous RDDs. When dealing with iterative processes, this dramatically slows down the whole process. We can avoid this risk by caching the intermediate results in each iterative step using the cache function, where cache function allows RDD to be saved in memory. Figure 6 summarizes how we achieve a fault-tolerant system by caching RDDs in each iterative step. If the RDDs are not cached regularly, the execution time within a given step will grow as it proceeds. The execution time in Figure 6 is measured within each iterative steps up to 25 comparing two approaches: one with caching and the other without caching. Synthetic data has been used on a Spark cluster configured with 1 master and 4 worker nodes. Considering the execution time is measured within each step, the cumulative time will grow quadratic as the model proceeds. By caching RDDs regularly, the execution time can be remained stable across every step.

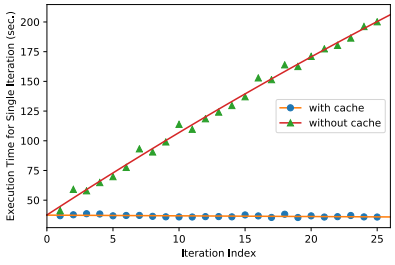


FIGURE 6. Average execution time comparison for two cases: with and without caching. The execution time is measured within each iterative step, which is not the cumulatively time from the start. Synthetic data with $N = 10000$, $V = 50000$, and $K = 100$ has been used on a cluster with 1 master and 4 workers.

d: NUMBER OF WORKERS

Figure 7 illustrates how the execution time can be reduced by increasing the number of worker nodes in a cluster. We only increase the number of workers while keeping the same hardware settings for each worker node(n1-highmem-8). We measure the execution time for two datasets by increasing the workers from 4 to 8, 16, and 32. As expected, more workers permit higher parallelism. However, the execution time is reduced slowly when the number of workers gets large. This can be due to the communications between workers, where more workers lead to higher network traffic.

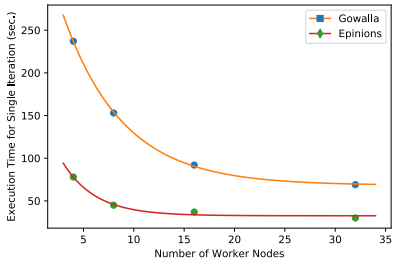


FIGURE 7. Execution time respect to the number of workers in a cluster when K is fixed to 100 on both datasets.

e: LOAD BALANCING

Our Spark implementation has been optimized by utilizing the partition and repartition function. When dealing with a skewed dataset, RDD should be partitioned well over executors through shuffling. Our implementation distributes the data well among the worker nodes following random partitioning. In another effort to achieve higher efficiency, we favor mapValue function over map function on PairRDDs. The mapValue function only transforms the value, whereas map function transforms both key and value.

2) QUANTITATIVE EVALUATION ON REAL-WORLD DATASETS

Following the similar approaches from the previous studies, we perform prediction tasks for evaluating the performance of our model. In our previous work, we showed how CLSM can perform challenging prediction on the given test user solely based on other domain information. Link prediction was performed solely based on the attributes; Attribute prediction was performed solely based on the link information. In this paper, we perform similar prediction tasks but based on a different setting. For performing link prediction, we hide 10% of existing links and train the model with 90% of link information and all the information about the attributes.

Similarly, for attribute prediction, we hide 10% of the selected attributes, and train the model with 90% of attribute information and all the link information. For both tasks, we perform 10-fold cross-validation approach.

a: LINK PREDICTION

The link prediction problem can be posed as a binary classification problem, where the label $y(n_1, n_2) \in \{0, 1\}$ reflects the existence of an edge between the given nodes within a pair. AUC-ROC (Area Under Curve - Receiver Operating Characteristic) which is widely used in various classification problems have been used for evaluation. As shown in Fig 8, AUC-ROC measures the area under the curve of ROC which plots the true positive rate against the false positive rate by adjusting the threshold to various value. For all the pairs in the test set, the existence of a link is hidden, and the probability of a link is computed using $\mathbb{E}_q[p(y(n_1, n_2)|\boldsymbol{\theta}_n)]$. By controlling the threshold, the true positive rate and the false positive rate can be computed with the pairs whose probability is above the threshold. For quantitative measure, we use AUC-ROC prediction tasks.

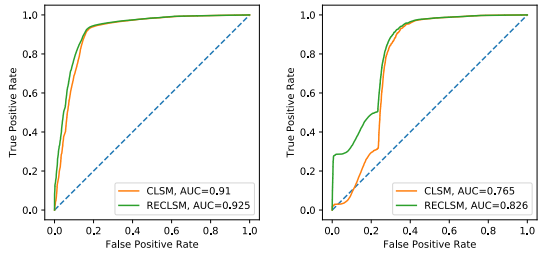


FIGURE 8. Figures show the ROC, and the AUC-ROC for two datasets, where K is set to 16. (a) Link prediction ROC for Gowalla. (b) Link prediction ROC for Epinions.

For each dataset, we perform link prediction with two models: CLSM and its extension: RECLSM. Each model repeats the same prediction task only changing the number of communities. The total number of communities (K) has been set to 4, 16, 32, 64, and 100. The AUC-ROC for all the experiments on the two datasets are presented in Table 3 which well summarizes the overall results.

TABLE 3. Link prediction performance.

		AUC-ROC for each K			
Dataset	Model	16	32	64	100
Gowalla	CLSM	0.91	0.93	0.94	0.942
	RECLSM	**0.925**	**0.937**	**0.943**	**0.946**
Epinions	CLSM	0.765	0.824	0.871	0.869
	RECLSM	**0.826**	**0.94**	**0.928**	**0.938**

Comparing the results of CLSM between the two datasets, we observe Gowalla dataset always tend to have higher AUC-ROC than Epinions'. We believe this is due to the unique characteristic of LBSN that allows users to check-in to their places and share it with their friends. The location information is often considered as personal information raising privacy concerns. To share their private information, users at LBSN make friends more carefully than other social platforms. Hence, the AUC-ROC is remarkably higher for Gowalla dataset. On the other hand, Epinions users make friends (or trust reviewers) more casually compared to Gowalla users. Predicting links in Epinions should be more challenging by nature, and this is exhibited in lower AUC-ROC compared to that of Gowalla's. RECLSM always performs better than CLSM for all across K on both datasets. The prediction tasks on Epinions show remarkable improvement with RECLSM. In the review-trust network, the trust relationship is usually established with reputable users. RECLSM considers this phenomenon in the generative process in the link generation, whereas CLSM always considers the similarity of the sampled communities. However, we want to stress that less significant improvement for Gowalla dataset doesn't mean RECLSM isn't suitable for LBSN. The AUC-ROC for Gowalla from CLSM was already too high allowing less room for improvement.

b: ATTRIBUTE PREDICTION

We follow the same setup for attribute prediction on the two datasets and evaluate our two models. Again, K has been set to 4, 16, 32, 64, and 100. We predict the 10% of the hidden attributes with all available information. Following the same logic in link prediction, the predictive probability can be computed using $\mathbb{E}_q[p(\mathsf{w}_{n,i}|\boldsymbol{\theta}_n)]$, where the membership distribution can be estimated through the network information and observed attributes. With the membership vectors and the model parameters obtained from the training set, we perform predictions on the places that will be visited or the skills that will be selected. The overall results for these predictions are provided in Table 4.

TABLE 4. Attribute prediction performance.

		AUC-ROC for each K			
Dataset	Model	16	32	64	100
Gowalla	CLSM	0.925	0.934	0.935	0.935
	RECLSM	**0.929**	0.934	**0.936**	**0.937**
Epinions	CLSM	0.886	0.891	0.882	0.878
	RECLSM	**0.905**	**0.900**	**0.893**	**0.881**

RECLSM always achieves higher AUC-ROC, except for one case when the two models exhibit the same performance. Compared to the link prediction, the attribute prediction has high AUC scores. The base model, CLSM already achieves high scores on both datasets; 0.925, 0.934, 0.935, and 0.935 AUCs are obtained for Gowalla; 0.886, 0.891, 0.882, 0.878 AUCs are obtained for Epinions. Considering these high scores from CLSM, the improvement from RECLSM is meaningful even the increase in number is smaller than that of the link prediction. The overall increase of AUC-ROC on attribute prediction using RECSLM reveal the effectiveness of modeling the user reputation. This is interesting as the original model has been extended in link generation specifically.

RECSLM differentiates the links between the link generated by reputation and the link generated by community structure. By only focusing on the links generated by the community structure and connecting them to the user attributes, the model can discover more refined community structure that directly reflects user attributes than CLSM.

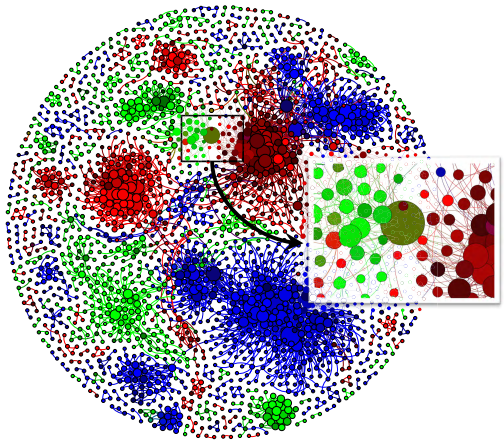
3) QUALITATIVE EVALUATION ON REAL-WORLD DATASET

For this particular evaluation, as location information is easy to visualize on the map, we focus on Gowalla dataset. Each node can be assigned to its community through the inference algorithm in RECLSM, and the frequent places for each community (or topic) can be learned. These places will be plotted on the map based on their geo-coordinates, which has never been used throughout the inference and learning process. Specifically, we set K to 64, where K is the total number of communities (or topics) of our interest. The topic size should be large enough considering the worldwide users, but it shouldn't be too large for manual inspection. We also control K to smaller values for other insights in the later part of the experiment.

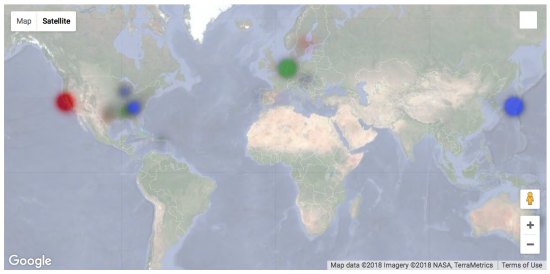
a: VISUALIZING GOWALLA DATASET

We apply our model on Gowalla dataset with K fixed to 64, infer the communities for each node, and find the favored places for each community (or topic). 3 topics from 64 are selected for visualization. Only 3 topics out of 64 are considered here as we can easily assign RGB values based on the membership vector of size 3. From the results, we would like to find: 1. whether RECLSM assigns membership vectors according to network topology. 2. where frequent places for each topic is located at. To find where the places are, we use Google Maps API with each place's latitude and longitude coordinate. The mixed-membership assumption which is incorporated in our model allows the node to be assigned to multiple memberships. Hence, we find the set of nodes that belong to each community of three by comparing the highest weight and the community of our interest. From Fig. 9a, we observe that nodes with the same color are densely connected while nodes of different colors are not. Fig. 9b shows that the top favored 500 places from the three communities form a city-level cluster, which indicates that the friendship at Gowalla rarely goes above the city level with the given community size of 64.

Another interesting observation we have in Fig. 9a is that the clusters of nodes under the same community color are not necessarily connected. For instance, three small red clusters in the left top area in Fig. 9a are disconnected, yet have the same color. In the conventional MMSB, when the size of the community is fixed small, the clusters of nodes are randomly assigned to the same community. However, RECLSM or CLSM examines the similarity in attributes even when the nodes are not connected, and it assigns the nodes under the same community when necessary. The colored regions in Fig. 9b supports our expectancy; If the assignments were random as in the conventional



(a)



(b)

FIGURE 9. Randomly selected 3 communities from 64 for visualization. Top (a) shows their network topology and bottom (b) shows their corresponding frequent places.

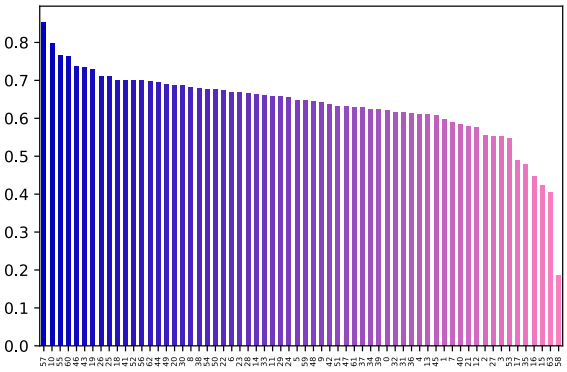
MMSB, these regions could have been scattered forming multiple clusters with the same color. This also shows how RECLSM or CLSM fully uses the two modalities simultaneously and finds the shared latent topics.

b: MIXED MEMBERSHIP

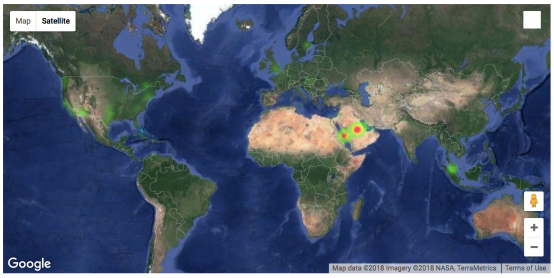
In Figure 9a, we see majority of nodes exhibit pure RGB color except some nodes. Some nodes have well-mixed membership vectors, and this has been reflected in colors. The node in the center from the enlarged window has well-mixed memberships of R and G. The value for RGB has been renormalized to a value between 0 and 255, where 0 is the darkest and 255 is the brightest. RGB color value of the focused node is [89.5, 110.1, 0]. This node happens to belong to other community out of the 3 of our interest, so the sum of the values is smaller than 255. In fact, all the nodes in dark color are affiliated to other communities besides the three. In most cases, non-pure colored nodes have high-degree, where the size of the node represents the degree. However, high-degree does not necessarily imply well-*mixed* membership.

We further inspect each community by examining the purity of nodes. Each node can be assigned to its major community by examining the index of the highest value in the membership vector, where the K has been set to 64. For every 64 community, we also count the number of nodes of which the membership vector value for the assigned community is

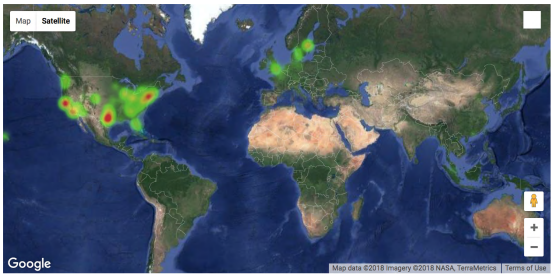
higher than 90%. These nodes above 90% are defined as *pure* nodes. We can compare these numbers with the total number of nodes that belong to the given community. Fig. 10a shows the ratio of pure nodes to the total nodes for each community. The users from the community with *pure* the most, tend to visit certain places in Saudi Arabia, and Kuala Lumpur in Malaysia. The users from these areas are less open to other users from different communities; 85.30% of the users only interact with themselves and their behavior is restricted compared to others'. On the other hand, community 58 has the least pure nodes, where the ratio is as low as 18.57%. As shown in Fig. 10c, the frequent places are scattered across major cities in the US and cities in Europe including London, Gothenburg, and Stockholm. Based on the scattered places, we believe nodes in this community have loose ties with the geography. Other factors could have strongly affected the community formation, such as location popularity.



(a)



(b)



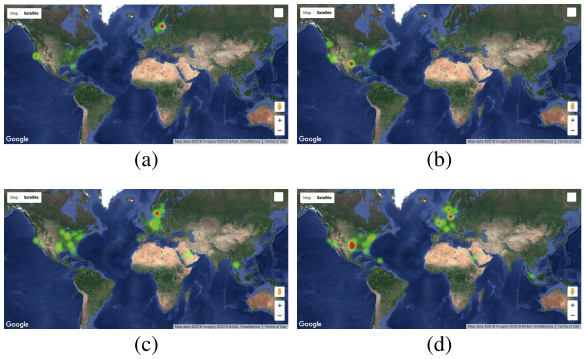
(c)

FIGURE 10. Proportion of *pure* nodes for each community and frequent places of communities of our interest. Note that the community index number itself does not have any meaning. Figure (a) shows community 57 is has the highest ratio of *pure*, and community 58 the lowest. Figure (b) shows the frequent places of community 57, and (c) shows the frequent places of community 58.

c: NUMBER OF COMMUNITIES IN GOWALLA

The previous two analyses were conducted when the number of topics K was fixed to 64. We control the number of topics to 4, 16, and 64. Note that we are not trying to find the optimal number of communities; instead, we are interested in analyzing the results from various granularity. Based on this approach, we can find frequent places from the communities for each granularity. Besides, we can track how the communities get split when moving to fine-scale inference.

We begin the analysis by setting the $K = 4$, which is the smallest of the three. Compared to the previous inference when K was 64, we expect the high-lighted places in the heat map to be broader as users are taking membership from the smaller set. Fig. 11 shows the frequent places from each community when K was set to 4. We can find that the users from Gowalla have been well separated into four groups. Fig. 11a shows that the users from one of the community mostly visited Stockholm, the San Francisco Bay Area, Orlando in Florida. Interestingly, with K set to 4, RECLSM merged communities from different places far apart through nodes that bridge two communities. From Fig. 11b, we can infer the second community from the four is based in Austin. In fact, Austin was one of the most active city in terms of check-in, where the company Gowalla is based. Fig. 11c shows a community of which the users are mostly from Gothenburg, multiple cities from Germany, and cities from the US. The last community from the four frequented Malmo, Oslo-Norway, London, Dallas, Oklahoma City, New York, Washington as shown in Fig. 11d. We find that the cities from Sweden form different communities as in Fig. 11a,11c,11d, which is opposite from the case of Germany in Fig. 11c. This is mainly because Sweden has the second highest number of users follwed by the US [38], and RECLSM or CLSM avoids any community to be skewed in the size of nodes. Hence, users from Sweden starts to form a new community across cities with enough amount of *seeds* rather than being absorbed into the other community.



(a) (b) (c) (d)

FIGURE 11. Heat map of frequent places on Google Map when topic size (or community size) is set to 4.

We further investigate the communities when the size of communities (or topics) gets increased. Specifically, we are interested to see how more fine-grained communities are formed. The size of communities has been increased

to 16 from 4, and the membership vectors of each node are inferred based on the size 16 without any information from the previous results when K was set to 4. In other words, the inferred 16 communities are not 4 sub-communities from the four previously found. Due to the space limit, we do not present all the 16 communities, but present visualizations of 4 selected communities that are highly related to the communities found previously when K was set to 4. Based on two series of inferences, each node has two main communities for two cases: when K was set to 4, and when K was set to 16. For each case in Fig. 11 we tie with the most relevant one from the sixteen. To find the most relevant community, we collect nodes from each 4 case and see how those nodes are assigned to its main clusters from 16 cases. Taking the first community from the four as an example, most nodes are assigned to community 15, we therefore choose community 15 for visualization (Fig. 12a). By examining the first and second largest communities from our histogram, we were able to confirm that two communities are based in San Francisco and Stockholm which were previously together in Fig. 11a.

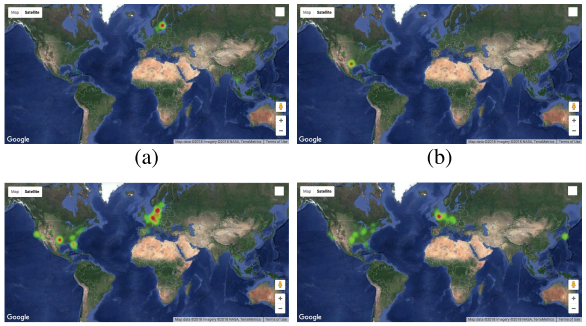


FIGURE 12. Heat map of frequent places on Google Map when topic size (or community size) is set to 16. Four majority matching communities from Fig. 11 are selected for visualization.

Following the same step, four communities obtained previously can be compared to its most matching communities from the sixteen respectively. Here the most match merely means the majority matching, and some can be split into multiple communities close to even. Fig. 12 has been arragned in order for comparison with Fig. 11. Community in Fig. 12b is now focused on Austin only removing smaller multiple clusters in Fig. 11b. Fig. 12c is more highlighted in Gothenburg and in Germany compared to the Fig. 11c. However, it still has multiple areas with high weights, which is expected to be reduced when the size of community gets larger. The community in Fig. 12d is mostly based in London with some users from Tokyo, where the users from two cities have been connected online in Gowalla. Tokyo wasn't visible when K was set to 4 because these minority users have been outnumbered by other users in the heat map.

VI. CONCLUSIONS AND FUTURE WORK

In our previous study [21], we showed how CLSM effectively predict links in social network and node attributes along multiple dimensions by fully using two modalities. This paper extends CLSM in two ways: *(i)* we introduce Spark implementation of CLSM and performs parallel community detection on multi-modal social dataset on a cloud; *(ii)* we extend CLSM by incorporating user reputation and describe links that have been missed in network homophily framework. The appealing feature of CLSM and RECLSM is that the inference algorithm can be easily fitted to Map-Reduce framework for distributed computation. Specifically, we present our code in Apache Spark, a widely used open-source for distributed computing. We deploy our models on a commercial cloud for evaluations on two predictive tasks using large-scale multi-modal social dataset. Our results show RECLSM improves our previous model both on link prediction and attribute prediction. We plan to apply our model to other types of multi-modal social dataset, where the dimension of modality is higher than two.

APPENDIX
VARIATIONAL INFERENCE

Given the observation of the interaction network Y and user behaviors $\mathbf{w}_{1:N} = \{\mathbf{w_k}\}_{k=1}^{N}$, we are interested in inferring the posterior distributions of the model's latent variables, $p(\boldsymbol{\theta}_{1:N}, \mathbb{Z}_{1:N}, \mathbf{C}|\mathsf{Y}, \mathbf{w}_{1:N})$ (where $\mathbf{C}$ is the collection of all $\mathbf{c}_m$ of all nodes), as well as estimating the hyperparameters $\boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\kappa}$. A number of approximate inference algorithms have been proposed in literature. We rely on variational method [27] that approximates the posterior by a computationally tractable variational distribution with some free variational parameters.

A factorized variational distribution over the latent variables $q(\boldsymbol{\theta}_{1:N}, \mathbb{Z}_{1:N}, \mathbf{C})$ is suggested as below:

$$\begin{aligned} &q(\boldsymbol{\theta}_{1:N}, \mathbb{Z}_{1:N}, \mathbf{C}) \\ &= \prod_{n=1}^{N} q_{\text{dir}}(\boldsymbol{\theta}_n|\boldsymbol{\gamma}_n) \prod_{m=1}^{M_n} q_{\text{mul}}(\mathbf{c}_m^n|\boldsymbol{\lambda}_m^n) \\ &\quad \times \prod_{n_1,n_2}^{N} q_{\text{mul}}(\mathbf{z}_{n_1\to n_2}|\boldsymbol{\phi}_{n_1\to n_2}) q_{\text{mul}}(\mathbf{z}_{n_1\leftarrow n_2}|\boldsymbol{\phi}_{n_1\leftarrow n_2}), \end{aligned} \tag{1}$$

where $\{\boldsymbol{\gamma}\}$, $\{\boldsymbol{\lambda}\}$, and $\{\boldsymbol{\phi}\}$ are the variational parameters. Similarly, for the distributions over the model's parameters, we also use factorized $q(\cdot)$ distributions, which are $q_{\text{beta}}(\beta_k|\tau_{k1}, \tau_{k0})$ and $q_{\text{dir}}(\boldsymbol{\omega}_i|\boldsymbol{\rho}_i)$.

Given the factorized variational distribution $q(\cdot)$, we next bound the log likelihood of the observed data using Jensen's inequality. Specifically, we consider the so called evidence lower bound (ELBO) defined as follows:

$$\begin{aligned} \log p(\mathsf{Y}, \mathbf{w}_{1:N}|\boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\kappa}) &\geq \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) \\ &\triangleq \mathbb{E}_q[\log p(\mathsf{Y}, \mathbf{w}_{1:N}, \boldsymbol{\theta}_{1:N}, \mathbb{Z}_{1:N}, \mathsf{B}, \mathbf{C}, \Omega|\boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\kappa})] \\ &\quad - \mathbb{E}_q[\log q(\boldsymbol{\theta}_{1:N}, \mathbb{Z}_{1:N}, \mathsf{B}, \mathbf{C}, \Omega)], \end{aligned} \tag{2}$$

where we defined Ω as a collection of $\boldsymbol{\omega}$.

The ELBO in Equation 2 is expanded as follows:

$$\begin{aligned}\mathcal{L} = &\sum_{n_1,n_2} \mathbb{E}_q[\log p(y(n_1,n_2)|\mathbf{z}_{n_1\to n_2}, \mathbf{z}_{n_1\leftarrow n_2}, \mathsf{B}] \\ &+ \sum_{n_1,n_2} \mathbb{E}_q[\log p(\mathbf{z}_{n_1\to n_2}|\boldsymbol{\theta}_{n_1}) + \log p(\mathbf{z}_{n_1\leftarrow n_2}|\boldsymbol{\theta}_{n_2})] \\ &+ \sum_n \sum_{m=1:M_n} \mathbb{E}_q[\log p(\mathbf{w}_{n,m})|\mathbf{c}_m^n), \Omega] \\ &+ \sum_n \sum_{m=1:M_n} \mathbb{E}_q[\log p(\mathbf{c}_m^n|\mathbf{z}_{n\to\cdot}, \mathbf{z}_{\cdot\leftarrow n})] \\ &+ \sum_n \mathbb{E}_q[\log p(\boldsymbol{\theta}_n|\boldsymbol{\alpha})] \\ &+ \sum_k \mathbb{E}_q[\log p(\beta_k|\boldsymbol{\eta})] + \sum_k \mathbb{E}_q[\log p(\boldsymbol{\omega}_k|\boldsymbol{\kappa})] \\ &- \sum_{n_1,n_2} \mathbb{E}_q[\log q(\mathbf{z}_{n_1\to n_2}|\boldsymbol{\phi}_{n_1\to n_2}) \\ &+ \log q(\mathbf{z}_{n_1\leftarrow n_2}|\boldsymbol{\phi}_{n_1\leftarrow n_2})] \\ &- \sum_n \sum_{m=1:M_n} \mathbb{E}_q[\log q(\mathbf{c}_m^n|\boldsymbol{\lambda}_m^n)] \\ &- \sum_n \mathbb{E}_q[\log q(\boldsymbol{\theta}_n|\boldsymbol{\gamma}_n)]. \end{aligned} \tag{3}$$

The lower bound can be maximized using the coordinate ascent algorithm. Toward that goal, we take the (partial) derivatives of $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda})$ with respect to the variational parameters and set them to zero.

For $\boldsymbol{\gamma}_n$, the update equations are as follows:

$$\boldsymbol{\gamma}_{n,k} \leftarrow \boldsymbol{\alpha}_k + \sum_{n'\neq n} \boldsymbol{\phi}_{n\to n',k} + \sum_{n'\neq n} \boldsymbol{\phi}_{n'\leftarrow n,k}. \tag{4}$$

Similar reasoning leads to update equations for the parameters $\{\boldsymbol{\phi}\}$ and $\{\boldsymbol{\lambda}\}$, with additional constraints that each component of those vectors sum to 1. The corresponding update equations are as follows:

$$\begin{aligned}\boldsymbol{\phi}_{n\to n',k} \propto \exp(&\mathbb{E}_q[\log p(\boldsymbol{\theta}_{n,k})] \\ &+ \mathbb{E}_q[\log p(y(n,n'))] + \mathbb{E}_q[\log \prod_{m=1}^{M_n} p(\mathbf{w}_{n,m})^{\mathbf{c}^n_{m,n'}}]),\end{aligned} \tag{5}$$

$$\boldsymbol{\lambda}^n_{m,n'} \propto \exp(\mathbb{E}_q[\log p(\mathbf{w}_{n,m}|\mathbf{c}^n_{m,n'} = 1)]), \tag{6}$$

where the term n' appearing in $\mathbf{c}^n_{m,n'}$ and $\boldsymbol{\lambda}^n_{m,n'}$ denotes the index of $\boldsymbol{\phi}_{n\to n'}$ in $\mathbb{Z}_n$. As with the indicator vector $\mathbf{z}_{n\to n'}$, $\mathbf{c}^n_m$ should have only one component equal to 1 setting all others to 0.

Note that the number of the parameters $\boldsymbol{\phi}_{n\to n'}$, and thus the computational complexity of the update equations for Eq. 5, is quadratic in the number of nodes, even when the network is sparse. This is because the parameters are defined both for links and non-links. To avoid this computational bottleneck, we adopt an approximation technique [29], by assuming that the parameters $\boldsymbol{\phi}_{n\to\cdot}$ for non-links can be replaced by a single mean-field parameter. Namely, let $\{\boldsymbol{\phi}_{n\to\cdot}\}^+$ and $\{\boldsymbol{\phi}_{n\to\cdot}\}^-$ be the set of parameters for links and non-links, respectively, and let $\bar{\boldsymbol{\phi}}_{n\to\cdot}$ be the average over the set $\{\boldsymbol{\phi}_{n\to\cdot}\}^+$. Within the above mean field approximation, each element of $\{\boldsymbol{\phi}_{n\to\cdot}\}^-$ is replaced by $\bar{\boldsymbol{\phi}}_{n\to\cdot}$. Thus, the time complexity of algorithm becomes linear in the number of existing links in the network.

Further gains in computational efficiency is achieved by limiting the number of components in set $\mathbb{Z}_n$ to the number of edges incident on node n, rather than having all the relations in the set. This corresponds to reusing indicator variables only when they have generated a link. Note also that now Equation 4 does not require all the parameters $\{\boldsymbol{\phi}\}$, as the parameters correspodning to non-links are replaced by $\bar{\boldsymbol{\phi}}_{n\to\cdot}$.

Finally, the update Equation 5 can be further simplified by making the best use of the assortative property in aMMSB, where only diagonal components of the block matrix are being considered. Instead of updating $K \times K$ combinations of $\boldsymbol{\phi}_{n_1\to n_2}$ and $\boldsymbol{\phi}_{n_2\to n_1}$ for a link between node n_1 and n_2, we can update only K parameters of by disregarding the inter-community links. By incorporating the assortative property for CLSM, we use the following update equation:

$$\begin{aligned}\phi_{n_1\to n_2,k} \propto \exp(&\mathbb{E}_q[\log p(\boldsymbol{\theta}_{n_1,k}) + \log p(\boldsymbol{\theta}_{n_2,k}) + \beta_k] \\ &+ \mathbb{E}_q[\log \prod_{m=1}^{M_{n_1}} p(\mathbf{w}_{n_1,m})^{\mathbf{c}^{n_1}_{m,n_2}} + \log \prod_{m=1}^{M_{n_2}} p(\mathbf{w}_{n_2,m})^{\mathbf{c}^{n_2}_{m,n_1}}]),\end{aligned} \tag{7}$$

where we further use $\mathbb{E}_q[\log p(\boldsymbol{\theta}_{n,k})] = \psi(\gamma_{n,k}) - \psi(\sum_r \gamma_{n,r})$, $\mathbb{E}_q[\beta_k] = \psi(\eta_{k,1}) - \psi(\eta_{k,2})$ employing the exponential family distribution property. As for the last terms in Equation 7, we use the following equation:

$$\begin{aligned}&\mathbb{E}_q[\log \prod_{m=1}^{M_{n_1}} p(\mathbf{w}_{n_1,m})^{\mathbf{c}^{n_1}_{m,n_2}}] \\ &= \sum_{m=1}^{M_{n_1}} \lambda_{m,n_2}(\psi(\sum_r \mathbf{1}(\mathbf{w}_{n_1,m} = i)\rho_{k,r}) - \psi(\sum_i \rho_{k,i})).\end{aligned} \tag{8}$$

The last terms in Eauation 7 show how the user attributes are infused in the update equation. Compared to the updates of $\{\boldsymbol{\phi}\}$ in CLSM, the updates of $\{\boldsymbol{\phi}\}$ in MMSB is more simple by omitting the last two terms.

$$\phi_{n_1\to n_2,k} \propto \exp(\mathbb{E}_q[\log p(\theta_{n_1,k}) + \log p(\theta_{n_2,k}) + \beta_k]. \tag{9}$$

Let us now focus on variational distributions over the model parameters β_k and $\boldsymbol{\omega}_k$, $k = 1, .., K$. We had previously defined a Beta distribution $q_{\text{beta}}(\beta_k|\tau_{1k}, \tau_{0k})$ with variational parameter $\boldsymbol{\tau}_1, \boldsymbol{\tau}_0$, and a Dirichlet distribution $q_{\text{dir}}(\boldsymbol{\omega}_i|\boldsymbol{\rho}_i)$ with variational parameter $\boldsymbol{\rho}_k$. Here we omit the derivation details and only present the final update equations for these parameters:

$$\tau_{1,k} \leftarrow \eta_1 + \sum_{(n_1,n_2)\in\text{link}} \phi_{n_1\to n_2,k}, \tag{10}$$

$$\tau_{0,k} \leftarrow \eta_0 + \sum_{(n_1,n_2)\in\text{non-link}} \phi_{n_1\to n_2,k}\phi_{n_2\to n_1,k},$$

$$\rho_{ij} \leftarrow \kappa_j + \sum_{n=1}^{N}\sum_{m=1}^{M_n} \mathbf{1}(\mathbf{w}_{n,m} = j) \sum_{n'\in\text{link}} \phi_{nn',i}\boldsymbol{\lambda}_{m,n'}. \tag{11}$$

Once the variational parameters are found, we can use them to estimate the model parameters themselves. We note that as an alternative approach, one can also derive explicit update equations for the parameter $\boldsymbol{\omega}_i$ directly, without using the variational parameter $\boldsymbol{\rho}_i$ in Equation 11. $\boldsymbol{\omega}_i$ can be optimized by introducing a Lagrange multiplier where we have the update equation as follows:

$$\boldsymbol{\omega}_{ij} \propto \sum_{n=1}^{N} \sum_{m=1}^{M_n} \mathbf{1}(\mathsf{w}_{n,m} = j) \sum_{n' \in \text{link}} \boldsymbol{\phi}_{nn',i} \boldsymbol{\lambda}_{m,n'}. \tag{12}$$

However, for this case, extra caution is needed that guarantees non-zero entities in any $\boldsymbol{\omega}_i$. This can be easily achieved by using smoothing techniques such as Laplace smoothing or pseudo-count smoothing [39].

REFERENCES

[1] H. Kautz, B. Selman, and M. Shah, "ReferralWeb: Combining social networks and collaborative filtering," *Commun. ACM*, vol. 40, no. 3, pp. 63–66, Mar. 1997.
[2] H. Ma, H. Yang, M. R. Lyu, and I. King, "Sorec: Social recommendation using probabilistic matrix factorization," in *Proc. 17th ACM Conf. Inf. Knowl. Manage.*, Oct. 2008, pp. 931–940.
[3] J. Chen, W. Geyer, C. Dugan, M. Muller, and I. Guy, "Make new friends, but keep the old: Recommending people on social networking sites," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, Apr. 2009, pp. 201–210.
[4] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *Proc. 4th ACM Int. Conf. Web Search Data Mining*, Feb. 2011, pp. 287–296.
[5] M. E. Hull, F. R. Farmer, and E. S. Perelman, "Method and system for customizing views of information associated with a social network user," US 7 269 590 B2, Sep. 11, 2007.
[6] A. Shepitsen, J. Gemmell, B. Mobasher, and R. Burke, "Personalized recommendation in social tagging systems using hierarchical clustering," in *Proc. ACM Conf. Recommender Syst.*, Oct. 2008, pp. 259–266.
[7] D. Carmel *et al.*, "Personalized social search based on the user's social Network," in *Proc. 18th ACM Conf. Inf. Knowl. Manage.*, Nov. 2009, pp. 1227–1236.
[8] D. G. Taylor, J. E. Lewin, and D. Strutton, "Friends, fans, and followers: Do Ads work on social Networks? How gender and age shape receptivity," *J. Advertising Res.*, vol. 51, no. 1, pp. 258–275, Mar. 2011.
[9] C. E. Tucker, "Social networks, personalized advertising, and privacy controls," *J. Marketing Res.*, vol. 51, no. 5, pp. 546–562, Oct. 2014.
[10] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Nat. Acad. Sci. United States Amer.*, vol. 99, no. 12, pp. 7821–7826, Apr. 2002.
[11] S. Fortunato, "Community detection in graphs," *Phys. Rep.*, vol. 486, nos. 3–5, pp. 75–174, 2010.
[12] M. E. J. Newman, "Modularity and community structure in networks," *Nat. Acad. Sci. United States Amer.*, vol. 103, no. 23, pp. 8577–8582, 2006.
[13] J. Duch and A. Arenas, "Community detection in complex networks using extremal optimization," *Phys. Rev. E*, vol. 72, p. 027104, Aug. 2005.
[14] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *Nat. Acad. Sci. USA*, vol. 101, no. 9, pp. 2658–2663, 2004.
[15] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, "Link communities reveal multiscale complexity in networks," *Nature*, vol. 466, no. 7307, pp. 761–764, Jun. 2010.
[16] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *J. Amer. Soc. Inf. Sci. Technol.*, vol. 58, no. 7, pp. 1019–1031, 2007.
[17] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Phys. A, Stat. Mech. Appl.*, vol. 390, no. 6, pp. 1150–1170, Mar. 2011.
[18] M. A. Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," in *Proc. SDM06 Workshop Link Anal., Counter-Terrorism Secur.*, Apr. 2006.
[19] B. Taskar, M.-F. Wong, P. Abbeel, and D. Koller, "Link prediction in relational data," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, pp. 659–666.
[20] R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla, "New perspectives and methods in link prediction," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2010, pp. 243–252.
[21] Y.-S. Cho, G. Ver Steeg, E. Ferrara, and A. Galstyan, "Latent space model for multi-modal social data," in *Proc. 25th Int. Conf. World Wide Web*. Geneva, Switzerland: Committee, Apr. 2016, pp. 447–458. doi: 10.1145/2872427.2883031.
[22] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, "Mixed membership stochastic blockmodels," *J. Mach. Learn. Res.*, vol. 9, pp. 1981–2014, Sep. 2008.
[23] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.
[24] D. M. Blei, "Probabilistic topic models," *Commun. ACM*, vol. 55, no. 4, pp. 77–84, Apr. 2012.
[25] R. M. Nallapati, A. Ahmed, E. P. Xing, and W. W. Cohen, "Joint latent topic models for text and citations," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2008, pp. 542–550.
[26] J. Chang and D. M. Blei, "Relational topic models for document networks," in *Proc. 12th Int. Conf. Artif. Intell. Statist.*, Aug. 2009, pp. 81–88.
[27] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul, "An introduction to variational methods for graphical models," *Mach. Learn.*, vol. 37, no. 2, pp. 183–233, Nov. 1999.
[28] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2011, pp. 1082–1090.
[29] P. Gopalan, D. M. Mimno, S. Gerrish, M. J. Freedman, and D. M. Blei, "Scalable inference of overlapping communities," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2249–2257.
[30] P. K. Gopalan and D. M. Blei, "Efficient discovery of overlapping communities in massive networks," *Nat. Acad. Sci.*, vol. 110, no. 36, pp. 14534–14539, Jul. 2013.
[31] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *Proc. Int. Semantic Web Conf.*, 2003, pp. 351–368.
[32] J. Tang and H. Liu, *Trust in Social Media*, 1st ed. San Rafael, CA, USA: Morgan Claypool Publishers, 2015.
[33] M. Jamali and M. Ester, "Trustwalker: A random walk model for combining trust-based and item-based recommendation," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: ACM, Jun. 2009, pp. 397–406. doi: 10.1145/1557019.1557067.
[34] J. Jamali and E. Martin, "A matrix factorization technique with trust propagation for recommendation in social networks," in *Proc. 4th ACM Conf. Recommender Syst.* New York, NY, USA: ACM, Sep. 2010, pp. 135–142. doi: 10.1145/1864708.1864736.
[35] S. Meyffret, E. Guillot, and L. Médini, and F. Laforest. (2012). *RED: A Rich Epinions Dataset for Recommender Systems*. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01010246
[36] M. Zerva. *Epinions Dataset*. Accessed: Nov. 1, 2018. [Online]. Available: http://cs.uoi.gr/~mzerva
[37] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *J. Mach. Learn. Res.*, vol. 14, pp. 1303–1347, 2013.
[38] F. Comunello, *Networked Sociability and Individualism: Technology for Personal and Professional Relationships*. Hershey, PA, USA: IGI Global, 2012.
[39] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition: United State (Prentice Hall Series in Artificial Intelligence)*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall, 2000.

YOON-SIK CHO received the B.S. degree in electrical engineering from Seoul National University, South Korea, in 2003, and the Ph.D. degree in electrical engineering from the University of Southern California, USA, in 2014. He was an Academic Mentor for RIPS program with the Institute for Pure and Applied Mathematics, University of California Los Angeles, and a Postdoctoral Scholar with the Information Sciences Institute, University of Southern California, before he joined Sejong University. He is currently an Assistant Professor with the Department of Data Science, Sejong University, South Korea. His research interests include large-scale data science, social network analysis, and cloud computing.