LEARNING WITH SPARSITY FOR DETECTING INFLUENTIAL NODES IN IMPLICIT INFORMATION DIFFUSION NETWORKS

by

Yingze Wang

B.E., Xian Jiaotong University, 2007

M.Phil, The Chinese University of Hong Kong, 2009

Submitted to the Graduate Faculty of

The Kenneth P. Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2014

UNIVERSITY OF PITTSBURGH

THE KENNETH P. DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Yingze Wang

It was defended on

March 26th 2014

and approved by

Shi-Kuo Chang, Department of Computer Science

Milos Hauskrecht, Department of Computer Science

Jingtao Wang, Department of Computer Science

Ching-Chung Li, Department of Electrical & Computer Engineering

Dissertation Director: Shi-Kuo Chang, Department of Computer Science

LEARNING WITH SPARSITY FOR DETECTING INFLUENTIAL NODES IN IMPLICIT INFORMATION DIFFUSION NETWORKS

Yingze Wang, PhD

University of Pittsburgh, 2014

The diffusion of information and spreading influence are ubiquitous in social networks. How to model and extract useful information from diffusion networks especially in social media domain is still an open research area that requires significant attention. Many real applications pose new challenges in modeling information diffusion process. In particular, the first challenge comes from the fact that the underlying network structure over which the propagation spreads is unknown or unobserved. It is often the case that one can only observes that when nodes got infected by which contagion but without the knowledge about who infecting whom. The second challenge comes from the simultaneous transmissions of multiple correlated contagions through an implicit network. The third one comes from strong temporal effect in the diffusion process which needs to be carefully modeled.

In my thesis, we address two fundamental tasks, forecasting and influential-node detection, in an *implicit* diffusion network by a unified approach. In particular, we first proposed a sparse linear influence model (SLIM) which takes a nice form of a convex optimization problem. We further extended SLIM to multi-task sparse linear influence model (MSLIM), which could model diffusion networks with multiple correlated contagions. MSLIM, as a richer model than SLIM, not only improves prediction accuracy, but also allows to select influential nodes on a finer grid, i.e., select different sets of influential nodes for different contagions. For SLIM and MSLIM, we developed both deterministic and stochastic optimization algorithms for solving the corresponding problems and showed the fast theoretical convergence guarantees.

Another contribution of the thesis is the development of a general purpose system, called Slow Intelligent System (SIS), which is able to continuously learn and improve performance over time. We proposed the component-based SIS and developed the software with applications to face recognition task. Furthermore, we utilized the idea of the SIS to systematize the information diffusion process modeling and influential node detection and proposed SIS-based SLIM/MSLIM approaches, which further improve the flexibility and scalability of learning from implicit diffusion networks. We demonstrated the superiority of the proposed approaches on several real datasets from social media domains.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

First, I want to show my deep appreciation and gratefulness to my thesis advisor Professor Shi-Kuo Chang, who is always so patient of providing me with great support and suggestions. Without the freedom and encouragement he gives to me over the past five years, this thesis could not be done. I could still remember that he patiently listened to my practise presentation; he exchanged the emails with me at 2 am in the morning; he stand the surgical pain to discuss with me about my thesis proposal. In addition to all his help to my research, he is extremely nice and sometimes like the father figure in my life. I will never and ever forget that memorable wedding ceremony he host for me and Xi. Thanks again, Professor Chang!

I also would like to thank my other thesis committee members, Milos Hauskrecht, Jingtao Wang and Ching-Chung Li. They all provide many insightful suggestions to my thesis. In particular, I want to thank Prof. Milos who taught me a lot about the time-series analysis related works. I especially thank Prof. Wang who took tremendous efforts to read the preliminary draft of my thesis and gave great suggestions on how to modify and improve it. I deeply appreciate Prof. Li supported and encouraged my thesis work from the very beginning. I am also grateful to Prof. G. Elisabeta (Liz) Marai. She gave me great help for my thesis proposal. Her great course on CS3610 opens the door of the field of visualization to me, which is very helpful for my research. I also would like to thank many other faculty members in Computer Science Department: Daniel Mosse, Diane Litman, Kirk Pruhs and Youtao Zhang, George Novacky.

I am also indebted to many other colleagues: Wen-Hui Chen, Bin Gao, Xiang Guang, Liqun Kuang, Ting-Chun Peng, Chia-Chun Shih, Yao Sun and Emilio Zegarra. They all played important role during my Ph.D. through collaborations and discussions. During my

stay at University of Pittsburgh, I have pleasure to know great friends: Mehmud Abliz, Di Bao, Xiaolong Cui, Yu Du, Xiangmin Fan, Wen Gao, Yang Hu, Michael Lipschultz, Mengmeng Li, Youming Liu, Zitao Liu, Wencan Luo, Jiannan Ouyang, Boyu Sun, Lanfei Shi, Huichao Xue, Wen Xu, Wenting Xiong, Xiang Xiao, Fan Zhang, Miao Zhou and Xianwei Zhang. With them, my PhD study life became colorful and not alone. I also owe special thanks to Keena Walker. She had done a perfect administration job and provided assistance to make sure that my graduate experience went smoothly.

Most importantly, I want to thank my parents Heping Wang and Jiao Yu for their always companionship, love and support in all these years. I love you and miss you so much. Finally, I would like to thank my husband, Xi Chen for his unconditional love and endless support. Without you, I cannot enjoy the great life in the past few years.

1.0 INTRODUCTION

The modeling of diffusion and propagation of information has been an active research area recently. Information diffusion is a fundamental process taking place in many network applications. Examples include spreads of news and opinions [Adar et al., 2004, Gruhl et al., 2004, Leskovec et al., 2007b, 2009, Liben-Nowell, 2008], spreads of infectious diseases [Anderson and May, 2002, Hethcote, 2000], spreads of technology innovations [Rogers, 1995, Strang and Soule, 1998] and word of mouth effects in marketing [Domingos and Richardson, 2001, Kempe et al., 2003, Leskovec et al., 2007a].

In the study of information diffusion network, there are two fundamental tasks:

1. Forecasting or Prediction: the amount or volume of a contagion (e.g., the information, idea, disease) represents the importance of a certain type of information, especially in multi-contagion networks. In time-series social networks, by predicting the amount or volume of multiple contagions, one can capture the trend of information diffusion so that actions can be taken to either encourage the spreads (e.g, for product advertisement) or prevent the spreads (e.g,. virus or disease) of contagions. In a multi-contagion information diffusion network, it is of great importance to predict the volume for each type of contagions from historical observations.
2. Influential-node Detection: the successful identification of influential nodes is crucial in many applications. For example, in viral marketing, the company may give free samples of the product to those influential customers to trigger a large cascade of recommendations. In preventing the spread of infectious disease, once the sources of the infection have been detected, we could limit their interactions with the outside world.

There are many other research problems of great interest. For example, time-series link-

prediction problems where one studies how the links of network is changing over time in an information diffusion process.

In order to address the forecasting and influential-node detection problems, there are several challenges one has to address:

1. Many existing work makes the assumption that the complete network structure is given as *a priori* and information can only spread over the edges of the underlying diffusion network. However, in many scenarios, the diffusion network is *implicit* or *unknown*. In the other words, we can only observe that particular nodes get "infected" at certain time but do not know who infected them. For example, we only observe the fact that the people get sick without knowing who infected them. In social media, blogger writes new blogs usually without explicitly citing the source and thus we have no idea about where the blogger gets infected from. Moreover, in many situations, even though the network structure is available (e.g., friendship/followers relationship in social networks), the network itself cannot explain how a node gets infected by the contagion. For example, in viral marketing, a customer, who discovers and likes a new product, could have been influenced by many different types of information, e.g., recommendation from friends in the real world, media sites, blogs and forums.
2. In many real applications, there are multiple contagions diffused over information networks simultaneously. Many existing works either model the diffusion process for only one contagion or model multiple contagions independently without taking the correlation among contagions into account. When there are multiple related contagions in the network, we should utilize the similarities among contagions to make the forecasting as well as detecting the influential nodes. In other words, for two similar contagions, the estimated volumes and sets of influential nodes should also be close.
3. One needs to carefully model the temporal effect of the diffusion process and the change of the network over time. Some existing works assume a single static network over the entire process, which is not rich enough for modeling the information diffusion process.

The main goal of this thesis is to develop both statistically sound and computationally efficient approaches which can simultaneously conduct both forecasting and influential-node

detection in an implicit information diffusion network.

1.1 MAIN CONTRIBUTIONS

We summarize the main contributions of the thesis as follows:

1. We propose a unified framework, called sparse linear influence model (SLIM) [Wang. et al., 2012], to simultaneously address the problem of contagion-volume prediction and influential-node detection in an implicit information diffusion network. Our method takes the temporal effects into account when building a model and formulates the task as a convex optimization problem. We further develop an efficient method to solve the corresponding optimization problem, which achieves the *optimal* first-order convergence rate.
2. We extend the proposed SLIM model to better model multi-contagion information diffusion networks. In SLIM, we assumed a global influence function for each node without modeling different levels of influence for different contagions. To address this issue, we further propose the multi-task sparse linear influence model (MSLIM) by introducing a contagion-sensitive influence function so that we can express different sets of influential nodes for different contagions while taking into consideration the correlations among contagions [Wang et al., 2013]. We formulate this problem as a well-defined convex optimization problem and propose an efficient deterministic optimization method to solve it. Further, we propose stochastic optimization algorithm, which relies on the gradient approach applied to a random subset of data points instead of the entire dataset, to further improve the scalability and storage efficiency.
3. Slow intelligence system (SIS) proposed in [Chang, 2010] can be viewed as a general purpose system that is able to continuously learn and improve performance of the system over time. Since the seminal concept of slow intelligence has been proposed, it has been widely applied to many real-world applications, e.g., object tracing [Dong, 2010], topic trend detection [Kim et al., 2011], ontology filtering [Chang et al., 2010], network management [Colace et al., 2010], petcare system [Chang et al., 2013a]. In this thesis,

we propose a component-based slow intelligent system [Chang et al., 2011a], which can help SIS system developers to easily create SIS project templates. Further, we apply the component-based SIS to develop a new face recognition system [Chang et al., 2012].

4. We propose a SIS-based approach for sparse learning in the application of tumor classification, which is an iterative method combining the advantages of several sparse learning methods [Wang and Chang, 2011]. We demonstrate its empirical performance on two cancer datasets. Based on that, we further develop the SIS-based approaches for influential node detection in sparse linear influence model. As compared to SLIM and MSLIM models, SIS-based SLIM/MSLIM approaches are more flexible and scalable for dealing with large-scale complex network datasets.

1.2 ORGANIZATION

We organize the thesis as follows,

1. In Chapter 2, we review some background of information diffusion networks and their analysis, influential nodes detection, sparse learning and first-order optimization techniques, as well as a slow intelligence system. Since both information diffusion analysis and sparse learning contain a huge body of literature ranging from theory, computation to applications, we only present these material that is closely related to our work.
2. In Chapter 3, we introduce sparse linear influence model (SLIM) and an efficient optimization approach. We also demonstrate its application to influential user selection in an implicit social network. This chapter is based on the work published in [Wang. et al., 2012].
3. In Chapter 4, we extend SLIM to multi-task sparse linear influence model (MSLIM). We further propose both deterministic and stochastic optimization techniques to solve SLIM and MSLIM. This chapter is based on the work published in [Wang et al., 2013].
4. In Chapter 5, we present our work on the component-based slow intelligence system and its application to face recognition system. This chapter is based on a series of publications, [Chang et al., 2011a,b, 2013b, 2012].

5. In Chapter 6, we propose a SIS-based approach for high-dimensional sparse learning and demonstrate its application to tumor classification. We further develop the SIS-based approach for influential node detection in single-task and multi-task sparse linear influence models. This chapter is based on the work published in [Wang and Chang, 2011]
6. In Chapter 7, we summarize and conclude the thesis. We also present some possible future directions excluding the work in this thesis.

2.0 BACKGROUND

2.1 AUTOREGRESSIVE MODEL FOR INFORMATION DIFFUSION NETWORKS

In time series analysis, Autoregressive (AR) model [Brockwell and Davis, 1991] is one of the most popular models. Given a time series data $V(t)$ with the time index t, AR(L) model (autoregressive model of order L) can be written as

$$V(t) = \sum_{l=1}^{L} a_l V(t-l) + \epsilon_t,$$

where $a_1, \ldots, a_L$ are parameters to be learned and the random variable ϵ_t is white noise. In the information diffusion network application considered in this thesis, $V(t)$ usually represents the total volume of the contagion between $t-1$ and t, i.e., the total times that entire nodes get infected between $t-1$ and t. With the learned parameter $\widehat{a}_1, \ldots, \widehat{a}_L$, one can predict the total volume at time t by,

$$\widehat{V}(t) = \sum_{l=1}^{L} \widehat{a}_l V(t-l).$$

In addition to AR model, more complex models such as autoregressive moving-average model (ARMA) can be used for the prediction of total volume.

In practice, in addition to the total volume $V(t)$, we also have more refined observations. In particular, for each node u, we can observe the number of times that the node u gets infected by the contagion between $t-1$ and t, denoted by $M_u(t)$. Assuming there are in

total N nodes, let $M(t)$ be the length N vector. The VAR(L) model (vector autoregressive model of order L) takes the following form,

$$M(t) = \sum_{l=1}^{L} A_l M(t-l) + \epsilon_t,$$

where $A_1, \ldots, A_L$ are parameters to be learned with each $A_l \in \mathbb{R}^{N \times N}$ being a matrix. In such a model, we have in total $N^2 L$ parameters to be estimated, which is not only computationally too expensive, but also leads to overfitting and thus poor prediction performance when the total number of nodes N is large.

To design a good predictive model, we need to first reduce the number of parameters in the model. It is known that many nodes have little impact on the diffusion of the information while only a few nodes are the "hub nodes". Therefore, an effective way of learning a parsimonious model to utilize the sparse learning techniques to conduct subset selection of nodes. In particular, we identify a subset of influential nodes and estimate the parameter only for these selected nodes. This motivates the main work of the thesis. For more details, please refer to sparse linear influence model in Section 3.3.

2.2 INFORMATION DIFFUSION AND CASCADING IN NETWORKS

Information cascades are the phenomena in which an action or idea becomes widely adopted due to influence by others [Bikhchandani et al., 1992]. Information cascading and diffusion in social network studies how new opinions, technologies, behaviors or conventions spread from person to person through a social network. The earliest study of this problem dates back to the 20th century in the field of sociology [Coleman et al., 1966, Rogers, 1995, Strang and Soule, 1998]. In recent years, due to the rise of online social media, a lot of research has been devoted to this problem.

Pioneer research in modeling the flow of information through the network has been done in the field of epidemiology. The susceptible-infected-susceptible (SIS) and susceptible-infected-recovered (SIR) are two popular approaches for studying the spreads of diseases or viruses over a network [Hethcote, 2000]. These two classical models are based on the

stages of a disease in a host. The process is like this: when a person who is susceptible to a disease is exposed to an infectious people, she/he becomes infected by this disease. By some medical treatment, the person recovers and the disease goes away. The person is immune for some time and then she/he can become susceptible again. In particular, let $S(t)$ represents the number of individuals not yet infected with the disease at time t; $I(t)$ denotes the number of individuals who have been infected with the disease and are capable of spreading the disease; β is the transmission rate and α is the removal or recovery rate. The susceptible-infected-susceptible cycle is described by a set of differential equations:

$$\begin{aligned} \frac{dS}{dt} &= -\beta SI + \alpha I \\ \frac{dI}{dt} &= \beta SI - \alpha I \end{aligned}$$

Susceptible-infected-recovered model further introduces $R(t)$ to present those individuals who have been infected and then recovered from the disease. In addition, it assumes that recovered individuals are not able to be infected again or to transmit the infection to others. The susceptible-infected-recovered model takes the form of:

$$\begin{aligned} \frac{dS}{dt} &= -\beta SI \\ \frac{dI}{dt} &= \beta SI - \alpha I \\ \frac{dR}{dt} &= \alpha I \end{aligned}$$

However, these disease propagation models are overly simplified representations of a reality. More specifically, they cannot model the observed data/information at each time stamp and they do not take multi-contagions spreading into consideration.

Diffusion models [Domingos and Richardson, 2001, Goldenberg et al., 2001, Newman, 2002, Kumar et al., 2003, Granovetter, 1978, Leskovec et al., 2007b, Rogers, 1995, Song et al., 2007] which model the procedure of adoption and spread of information can be generally classified into two groups:

- *Threshold model* [Granovetter, 1978] where each node has an initiated threshold $t \in [0, 1]$ drawn from some distributions. Each edge of the network has a connection weight $w_{u,v}$. A node u adopts the information if $t \leq \sum_{\text{adopters}(u)} w_{u,v}$, that is the sum of the connection weights from the neighbours that adopted the information is larger than the threshold t.
- *Independent cascade model* [Goldenberg et al., 2001] where node u adopts the information with probability $p_{u,v}$ if its neighbour v adopts the information.

However these two models have several assumptions when modeling the spreads of influence in a network. They assume that the information can only spread over the edges of the diffusion networks. In other words, one can only propagate information to its neighbours. Thus these models need the complete network structure as a priori knowledge.

There are three main challenges of these models: 1) Parameter estimation of these models is challenging due to the heterogeneity of the nodes and scarcity of data which is difficult to obtain; 2) In some cases, the network structure is implicit or even unknown. We can only observe when the nodes get "infected" by the certain information but not know who infect them. For example, in information propagation, we can only observe when media sites mention certain information without explicitly acknowledging the source; 3) The structure of network itself cannot completely explain the observed diffusion process. In many cases, an activation of a node is not only based on the social network structure but also depends on some other factors. For example, in viral marketing, a customer, who discovered and liked a new product, could have been influenced by many different types of information, e.g., recommendation from friends in real life, media sites, blogs and forums. In virus propagation, although we can construct the network structure by using the friend relationship, it still cannot imply the entire diffusion process. A person got sick but one can not make the conclusion that he/she was infected by his/her friends. He may get the disease in some public places. Thus, even though traditional information diffusion process is modeled over underlying social network structure, existing methods may be too constrained to capture the complexity of the underlying phenomena. Recently, Yang and Leskovec (2010) proposed a new model called *linear influence model (LIM)*, which can effectively forecast the global influence through an implicit network and has been demonstrated to be a superior method in several applications. This model has two main advantages: 1) it does not require the

complete network structure. No explicit knowledge of the network is necessary. 2) it models not only the influence each node has on the diffusion but also how the diffusion unfolds over time. We will build our models based on LIM and the details of LIM will be presented in Section 3.2.

2.3 IDENTIFYING INFLUENTIAL NODES IN INFORMATION DIFFUSION NETWORKS

The problem of modeling the diffusion of information arises in a wide spectrum of domains, including social media analysis, infectious disease spread and viral marketing. One important research question is to detect the most influential nodes in an information diffusion network. The successful identification of influential nodes is crucial in many applications. For example, in viral marketing, customers can share their experiences and opinions regarding to a product with everyone. The company may give free samples of the product to those influential customers to trigger a large cascade of recommendations.

Identifying influential nodes in social network analysis is first formulated into a discrete optimization task by Kempe et al. (2003, 2005). After that, many works over the past ten years have been devoted to this research topic [Ma et al., 2008, Weng et al., 2010, Chen et al., 2010, Cha et al., 2010, Kimura et al., 2010, Ilyas and Radha, 2011, Biessmann et al., 2012]. However, most of these works suffer from the following two problems:

1. Many existing works need the complete network structure. However, in many scenarios, the diffusion network is implicit or unknown. Moreover, in many situations, even though the network structure is available (e.g., friendship/followers relationship in social networks), the network itself cannot fully explain how a node gets infected by the contagion (Note that "contagion" means a certain kind of information, e.g. topic). Therefore, when analyzing social network data, it is not proper to directly model the fact that "a node gets infected" as a result of influence from its neighbors. The question is: can we identify the influential nodes without knowing the network structure?
2. Many existing works strive to detect influential nodes for either only one contagion

Method	Implicit Network	Multi-Contagion	Temporal
[Kempe et al., 2003]	N	N	Y
[Ma et al., 2008]	N	N	Y
[Weng et al., 2010]	N	Y	X
[Cha et al., 2010]	N	N	Y
[Chen et al., 2010]	N	N	Y
[Bakshy et al., 2011]	N	N	Y
[Biessmann et al., 2012]	Y	N	Y

Table 1: Summary of some representative works for modeling influence of nodes.

(i.e., information, topic, disease) or across all contagions. When there are multiple related contagions in the network, we should detect different sets of influential nodes for different contagions, yet utilizing the similarities among contagions. In other words, for two similar contagions, the estimated sets of influential nodes should also be close. How can we detect the most influential nodes for multiple contagions given the fact that the network structure is unknown?

We compare some relevant works in Table 1 according to the following three metrics:

1. Implicit Network: whether the method allows the absence of the network structure.
2. Multiple contagions: whether the method can select different influential nodes for different contagions and utilize the relatedness of contagions to guide the selection.
3. Temporal: whether the method incorporates the temporal information into the modeling process or it treats each time-stamp independently.

As seen from Table 1, almost of all these works require the network structure to model the influence of nodes. Only one recent work was proposed to predict the canonical trend in an implicit network based on a canonical correlation analysis [Biessmann et al., 2012]. However, this work mainly focuses on detecting a *single* trend setter (i.e., earliest source of a canonical trend) but not a set of the most influential nodes. In this thesis, we address

the aforementioned two problems by developing a new methodology for detecting the most influential nodes for multiple contagions in the implicit information diffusion network. Our models take the three above metrics into account and can simultaneously predict contagion volume, which does not require explicit knowledge of the network structure.

2.4 SPARSE LEARNING

In high-dimensional learning problems, sparsity is one of the most important concepts, which often takes the form of feature selection in regression/classification. Exploring sparsity in learning problems gives many benefits: (1) it will make the model more interpretable and computationally cheaper in applications. Therefore, even though underlying problem does not admit sparse solutions, one might still prefer a sparse approximation of the solution; (2) When the underlying model is indeed sparse as in many high-dimensional applications (e.g., predicting the output y from a high-dimensional input vector $\mathbf{x} \in \mathbb{R}^J$), utilizing sparsity in learning methods will provide much improved prediction performance. Given a dataset of N input/output pairs: $\{\mathbf{x}_i, y_i\}$ for $i = 1, \ldots, N$, let $\mathbf{y}$ denotes the vector of outputs and $\mathbf{X}$ denotes the matrix of inputs of N samples. The vector pairs are assumed to be independent and identically distributed.

When $\mathbf{y}$ are real numbers for regression problem, we assume a linear model: $y_i = \boldsymbol{\beta}^T \mathbf{x}_i + \epsilon, (1 \leq i \leq N)$, where ϵ is the noise following a zero-mean Gaussian distribution. Then we estimate the regression coefficient $\boldsymbol{\beta}$ by minimizing the following squared loss function:

$$\ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) = \frac{1}{2}(y_i - \boldsymbol{\beta}^T \mathbf{x}_i)^2. \tag{2.1}$$

When $y \in \{-1, +1\}$ for the classification problem, we assume a logistic model: $Pr(y_i = 1|\mathbf{x}_i) = \frac{exp(\boldsymbol{\beta}^T \mathbf{x}_i)}{1+exp(\boldsymbol{\beta}^T \mathbf{x}_i)}, (1 \leq i \leq N)$. We estimate the regression coefficient $\boldsymbol{\beta}$ by minimizing the following logistic loss function:

$$\ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) = \log(1 + \exp(-y_i \boldsymbol{\beta}^T \mathbf{x}_i)). \tag{2.2}$$

One of the most widely used variable selection methods is the ℓ_1-regularized regression [Tibshirani, 1996, Chen et al., 1998]. The ℓ_1-regularized estimator can be obtained by solving the following convex optimization problem:

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^J} \sum_{i=1}^{N} \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) + \lambda \|\boldsymbol{\beta}\|_1, \tag{2.3}$$

where the loss function $\ell : \mathbb{R}^J \to \mathbb{R}$ is assumed to be a convex differentiable function with Lipschitz continuous gradient and the convex non-smooth ℓ_1-norm penalty $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{J} |\beta_j|$ is adopted to enforce sparsity among $\boldsymbol{\beta}$.

In the last decade, numerous works have been proposed to study ℓ_1-regularization regression problems and demonstrated superior statistical properties in terms of consistency and sparsistency [Zhao and Yu, 2006, Wainwright, 2009, Bickel et al., 2009, Zhang, 2009]. Based on ℓ_1-regularized approach, a more robust feature selection algorithm is proposed by using the elastic-net regularization [Zou and Hastie, 2005] which minimizes:

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^J} \sum_{i=1}^{N} \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) + \lambda \left(\alpha \|\boldsymbol{\beta}\|_1 + \frac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2 \right), \tag{2.4}$$

where α is the weight for $\|\boldsymbol{\beta}\|_1$. As we can see when $\alpha = 1$, it reduces to the standard ℓ_1-norm regularized approach.

The standard ℓ_1-norm penalty does not assume any structure among the input variables, which limits its applicability to complex high-dimensional scenarios in many applied problems. In some situations, the variables are partitioned into groups and it is desirable to jointly include or exclude all the variables within a group. To achieve group level variable selection, one can adopt the ℓ_1/ℓ_2 mixed-norm based group Lasso penalty [Yuan and Lin, 2006]:

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^J} \sum_{i=1}^{N} \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) + \gamma \sum_{g\in\mathcal{G}} w_g \|\boldsymbol{\beta}_g\|_2, \tag{2.5}$$

where $\mathcal{G}$ denotes a partition of $\{1, \ldots, J\}$, $\boldsymbol{\beta}_g \in \mathbb{R}^{|g|}$ is the subvector of $\boldsymbol{\beta}$ for the variables in group g; w_g is the predefined weight for group g; and $\|\cdot\|_2$ is the vector ℓ_2-norm . This ℓ_1/ℓ_2 mixed-norm penalty plays the role of jointly setting all of the coefficients within each group to zero or non-zero values. Theoretically, it has been demonstrated that if the group

structure is consistent with the true sparsity pattern, ℓ_1/ℓ_2 group Lasso penalty has the potential to improve the accuracy of the estimator [Huang and Zhang, 2010].

One can easily extend the previous group Lasso procedure to multi-task learning setting, where we learn multiple related tasks jointly by analyzing data from all of the tasks simultaneously instead of considering each task individually [Thrum and Pratt, 1998, Caruana, 1997, Yu et al., 2005, Zhang et al., 2008, Obozinski et al., 2009]. When the data are scarce, utilizing the information from other related tasks can make learning each task more effectively. More specifically, in the multi-task sparse regression problem, the goal is to learn for each task a functional mapping from a high-dimensional input space to a continuous-valued output space and such that only a small number of inputs are relevant to the output. In multi-task regression, it is often assumed that parameters for different tasks share the same sparsity pattern [Argyriou et al., 2008, Obozinski et al., 2010]. One can learn the joint sparsity pattern of parameters to achieve variable selection. One popular approach is to enforce group-wise sparsity across multiple tasks by adopting a joint sparsity regularization. In particular, assuming there are K related tasks, let $\boldsymbol{\beta}^k = (\beta_1^k, \ldots, \beta_J^k)^T$ be the regression coefficient vector for the k-th output. We denote by $\mathbf{B} = (\boldsymbol{\beta}^1, \ldots, \boldsymbol{\beta}^K)$ the matrix of regression coefficients for all of the K outputs. To learn the joint sparsity pattern, one can adopt the l_1/l_2 mixed-norm penalty [Argyriou et al., 2008, Obozinski et al., 2010, Negahban and Wainwright, 2011]:

$$\lambda \|\mathbf{B}\|_{2,1} = \gamma \sum_{j=1}^{J} \|\boldsymbol{\beta}_j\|_2, \tag{2.6}$$

where $\boldsymbol{\beta}_j = (\beta_j^1, \beta_j^2, \ldots, \beta_j^K) \in \mathbb{R}^K$ is the j-th row of $\mathbf{B}$ and γ is a positive regularization parameter.

2.5 FIRST-ORDER OPTIMIZATION

2.5.1 Accelerated Gradient Method

The aforementioned learning problems can always be formulated as a composite convex optimization problem:

$$\min_{\boldsymbol{\beta}} \Psi(\boldsymbol{\beta}) = f(\boldsymbol{\beta}) + P(\boldsymbol{\beta}), \tag{2.7}$$

where $f(\boldsymbol{\beta})$ is a smooth convex loss function with Lipschitz continuous gradient. $P(\boldsymbol{\beta})$ is a general non-smooth convex penalty function. As shown in the previous section, it can be the ℓ_1-norm $\lambda\|\boldsymbol{\beta}\|_1$ or the group Lasso penalty.

The traditional generic solvers include interior point method (IPM) [Dantzig and Thapa, 2003], block coordinate descent [Tseng and Yun, 2009, Tseng, 2001] and subgradient descent method [Bertsekas, 2004]. The optimization problem we considered here can be formulated as the second order cone programming, so it can be solved by the classical IPM. However IPM is computationally heavy and has very poor scalability because it needs to solve a large linear system for each iteration. Another possible method is block coordination descent, where each time it only optimizes on a small block of variables hoping that the optimization with respect to a small block will be very cheap. However for some complex structured penalties, each small block appears different places, so the optimization with respect to each block is very difficult. Another possible method is subgradient descent method, which is very scalable since it only utilizes the gradient information. However it has very slow convergence rate of $O(\frac{1}{\sqrt{t}})$, where t is the number of iterations. Another class of optimization methods [Nesterov, 2007, Beck and Teboulle, 2009], proximal methods have become increasingly popular recently. Similar to subgradient descent method, they only utilize the gradient information. Thus they can be scalable to the very large dataset. They achieves the fast convergence rate of $O(\frac{1}{t^2})$. However, when we apply proximal methods, we need to address the main challenge to guarantee the fast convergence rate. In particular, we require the exact solution of the proximal mapping step (proximal operator) at each iteration. More specifically, the proximal mapping, which is based on the linearizing of the smooth loss function f at the current

estimate $\mathbf{w}$, takes the following form:

$$\arg\min_{\boldsymbol{\beta}} Q_L(\boldsymbol{\beta}, \mathbf{w}) \equiv f(\mathbf{w}) + \langle \nabla f(\mathbf{w}), \boldsymbol{\beta} - \mathbf{w} \rangle + \frac{L}{2}\|\boldsymbol{\beta} - \mathbf{w}\|_2^2 + P(\boldsymbol{\beta}), \tag{2.8}$$

where L is the Lipschitz constant of ∇f and hence the problem in Eq. (2.8) is a quadratic upper bound of the original problem in Eq. (2.7). We also note that the term $\frac{L}{2}\|\boldsymbol{\beta} - \mathbf{w}\|_2^2$ can be replaced by any Bregman divergence between $\boldsymbol{\beta}$ and $\mathbf{w}$, $V(\boldsymbol{\beta}, \mathbf{w})$, which is defined as

$$V(\boldsymbol{\beta}, \mathbf{w}) := \omega(\boldsymbol{\beta}) - \omega(\mathbf{w}) - \langle \nabla \omega(\mathbf{w}), \boldsymbol{\beta} - \mathbf{w} \rangle, \tag{2.9}$$

where $\omega(\cdot)$ is a strongly convex and differentiable function.

When $\omega(\boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2$ so that $V(\boldsymbol{\beta}, \mathbf{w}) = \frac{1}{2}\|\boldsymbol{\beta} - \mathbf{w}\|_2^2$. The proximal operator can be written as:

$$\arg\min_{\boldsymbol{\beta}} \frac{1}{2}\|\boldsymbol{\beta} - (\mathbf{w} - \frac{1}{L}\nabla f(\mathbf{w}))\|_2^2 + \frac{1}{L}P(\boldsymbol{\beta}) \tag{2.10}$$

Note that $O(1/t^2)$ has already been optimal for solving any smooth convex function under the first-order black-box model [Nesterov, 2003]. It is important that the proximal operator can be computed *exactly*; otherwise, error created in each proximal operator will be accumulated over iterations.

Let $\mathbf{v} = (\mathbf{w} - \frac{1}{L}\nabla f(\mathbf{w}))$, when $P(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|_1$, then the proximal operator is simply component-wise soft-thresholding operation [Friedman et al., 2007]:

$$\boldsymbol{\beta}_j = \text{sign}(v_j) \max\left(0, |v_j| - \frac{\lambda}{L}\right). \tag{2.11}$$

For more complicated penalty, to apply any accelerated first-order method, one needs to first explore structure of penalty and derive the exact solution of the corresponding proximal mapping step (if it has one).

2.5.2 Stochastic First-order Optimization

The loss function $f(\boldsymbol{\beta})$ is defined based on a data set of finite input/output pairs: $\{\mathbf{x}_i, y_i\}$ for $i = 1, \ldots, N$. In fact, it is an approximation of the population loss function defined on the underlying distribution of the data points. More specifically, the loss function $f(\boldsymbol{\beta})$ can take the following form:

$$f(\boldsymbol{\beta}) := \mathbb{E}_\xi(F(\boldsymbol{\beta}, \xi)) = \int F(\boldsymbol{\beta}, \xi) P(d\xi), \tag{2.12}$$

where ξ is a random vector with the distribution P. In a typical learning problem setting, ξ represent the input/output pairs $(\mathbf{x}, y)$. We assume that for every random vector ξ, $F(x, \xi)$ is a convex and continuous function in x. Therefore, $f(x)$ is also convex.

The fact that the gradient $\nabla f(\boldsymbol{\beta}) = \mathbb{E}_\xi \nabla F(\boldsymbol{\beta}, \xi) = \int \nabla F(\boldsymbol{\beta}, \xi) P(d\xi)$ becomes computationally intractable for a high-dimensional P is one of the challenge of applying first-order method to solve sparse learning problems with $f(\boldsymbol{\beta})$ given by (2.12). To deal with this difficulty, a *stochastic gradient* $G(\boldsymbol{\beta}, \xi)$ is constructed to approximate $\nabla f(\boldsymbol{\beta})$. For example, we can simply choose $G(\boldsymbol{\beta}, \xi)$ to be $\nabla F(\boldsymbol{\beta}, \xi)$ or its mini-batch version $\frac{1}{m}\sum_{i=1}^m \nabla F(\boldsymbol{\beta}, \xi_i)$ where $\{\xi_1, \ldots, \xi_m\}$ are drawn from P independently. The algorithms utilizing these stochastic gradients are called stochastic first-order methods.

Many stochastic first-order methods [Duchi and Singer, 2009, Duchi et al., 2010, Hu et al., 2009, Langford et al., 2009, Nemirovski et al., 2009, Ghadimi and Lan, 2012, Lan and Ghadimi, 2010, Hazan and Kale, 2011] have been applied to different stochastic optimization problems, which have low per-iteration complexity and good capability of scaling to very large data sets. However, since a stochastic gradient unavoidably contains a certain level of noise, it needs more iterations for a stochastic first-order method to achieve the same optimality gap than a deterministic method. To find a solution $\widehat{\boldsymbol{\beta}}$ with $\mathbb{E}\Psi(\widehat{\boldsymbol{\beta}}) - \Psi(\boldsymbol{\beta}^\star) \leq \epsilon$ where $\Psi(\boldsymbol{\beta}) = f(\boldsymbol{\beta}) + P(\boldsymbol{\beta})$, a stochastic first-order method typically requires $O(1/\sqrt{t})$ iterations for convex loss function and $O(1/t)$ iterations for strongly convex loss function. [Nemirovski and Yudin, 1983] showed that both of these two complexities cannot be improved.

2.6 SLOW INTELLIGENCE SYSTEM

Chang (2010) proposed a new intelligence system called slow intelligence system (SIS), which is a general-purpose system that (i) solves problems by trying different solutions, (ii) is context-aware to adapt to different situations and to propagate knowledge, and (iii) may not perform well in the short run but continuously learns to improve its performance over time. A SIS system is not really slow, but continuously learns, searches for new solutions evolutionarily and propagates and shares its experience with other peers. The traditional expert system uses a knowledge base of human expertise for problem solving and learning in expert system is explicit. However, learning in SIS system is implicit and not always obvious. In [Chang, 2010], the author indicated that a slow intelligence system holds six typical characteristics, including Enumeration, Propagation, Adaptation, Elimination, Concentration, two complementary decision cycles.

- **Enumeration**: Given a problem, the system firstly enumerates different solutions until the appropriate one can be found.
- **Propagation**: The system analyzes the environmental changes and constantly exchanges information with the environment.
- **Adaptation**: Solutions are enumerated and adapted according to the environment. Sometimes the adapted solutions are mutations that exceed the previous ones.
- **Elimination**: Unsuitable solutions are eliminated, so that only the appropriate ones are further considered. Information acquired from the environment as well as learned experiences are used.
- **Concentration**: Among the suitable solutions left, one or most a few solutions are selected. Those solutions are the best ones for the problem.
- **Slow decision cycles to complement quick decision cycles**: Slow intelligence system at least has two decision cycles: quick and slow decision cycles. The quick decision cycles provide an instant response or solution to the environment and problem while the slow decision cycles evolutionarily select and adjust the solutions by following the gradual environmental changes and analyzing the information acquired by experts and

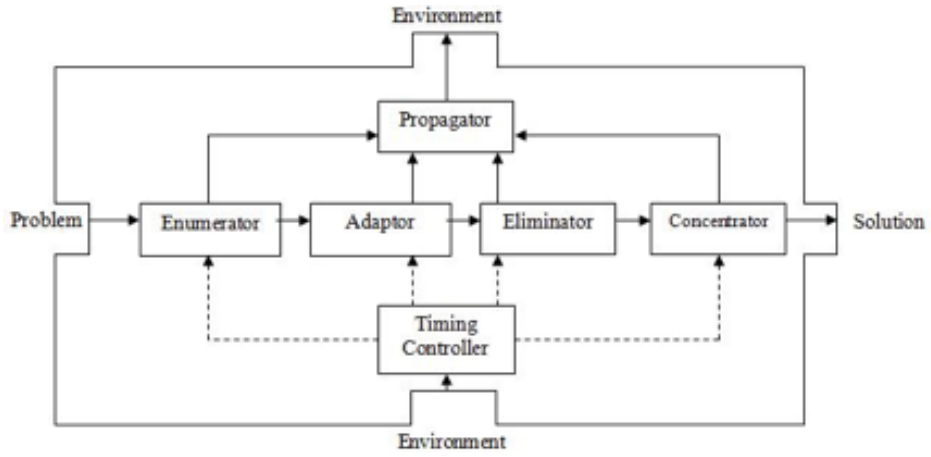


Figure 1: The basic building block (BBB) for SIS. This figure is adopted from [Chang, 2010].

past experiences. Usually, slow decision cycles may perform more poorly in the short run but better in the long run than quick decision cycles.

From the structural point of view, a Slow Intelligence System is a system with multiple decision cycles such that actions of slow decision cycle(s) may override actions of quick decision cycle(s), resulting in poorer performance in the short run but better performance in the long run. In [Chang, 2010], the author considered the structure of SIS by introducing the basic building block and advanced building block. SIS Basic Building Block (BBB) illustrated in Fig. 1 incorporates the above first five characteristics in its problem solving activities. The timing controller controls logical operations for the slow decision cycle and quick decision cycle. Depending on the level of abstraction, the BBB can be viewed differently: At the problem solving level the BBB is a pattern incorporating the above five described phases in its problem solving activities. At the implementation level the BBB is a software system consisting of interacting software components. An Advanced Building Block can be a stand-alone system as shown in Fig. 2. The major difference between an ABB and a BBB is the inclusion of a knowledge base, further improving the SIS problem solving abilities.

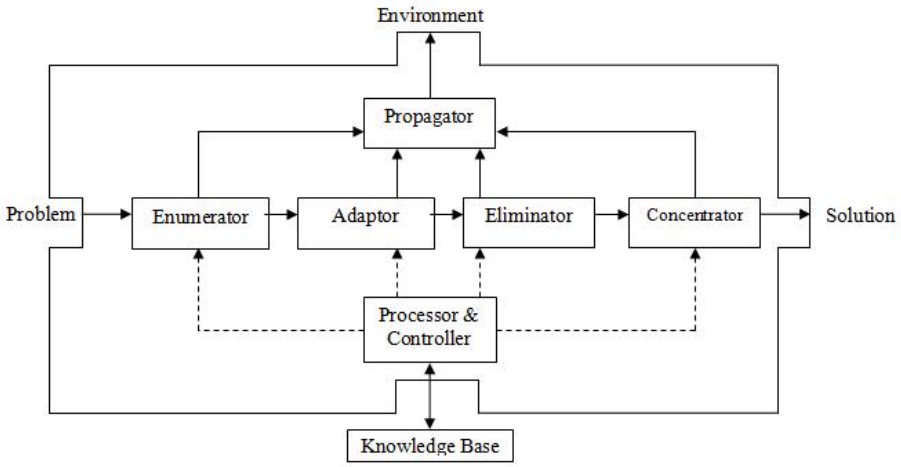


Figure 2: The advanced building block (ABB) for SIS. This figure is adopted from [Chang, 2010].

3.0 SPARSE LINEAR INFLUENCE MODEL

As we discussed in the introduction part, how to identify the influential nodes and forcast the contagion volume in the implicit information diffusion networks remains a challenging problem. In this chapter, we develop the sparse linear influence model (SLIM) to automatically identify global influential nodes in an implicit social network. Our model is based on the linear influence model with two main advantages: 1) we don't need to know the underlying network topology or structure as prior knowledge (e.g., connections among users); 2) it can simultaneoursly conduct the contagion volume prediction and influential node detection. To solve SLIM, which is a non-smooth optimization, we adopt the accelerated gradient descent (AGM) framework. We show that there is a closed-form solution for the key step at each iteration in AGM. Therefore, the optimization procedure achieves the optimal convergence.

3.1 INTRODUCTION AND MOTIVATION

As outlined in Chapter 2, modeling the diffusion of information has been one of the core research areas for analyzing social network data. A central question in modeling information diffusion is how to find the influential nodes in the network. In the past a few years, a great effort has been devoted to identifying the influential nodes [Ilyas and Radha, 2011, Ma et al., 2008, Weng et al., 2010, Kimura et al., 2010, Kempe et al., 2005]. Most of these algorithms rely on a strong assumption that the entire topology of the network is available as prior knowledge and the information can only disseminate over the given network structure. For example, one of the state-of-the-art methods [Cha et al., 2010] ranks users based on the number of followers and PageRank, which depends on the network structure. However,

in many problems, the network structure is unavailable or hard defined. Throughout this thesis, we make a basic assumption: the information on multiple topics spreads through an implicit network but we can observe the volume (the number of nodes infected) for each topic over time.

Very recently, linear influence model (LIM) [Yang and Leskovec, 2010] was proposed to analyze the social network with an unknown network structure. LIM models the global influence of each node through an implicit network. In particular, LIM captures the global temporal effect by modeling the number of newly infected nodes as a linear function in all other nodes infected in the past, where infected nodes mean the nodes mentioned or get certain contagion or information. Although LIM can roughly model the influence for each node under our basic assumption, it cannot tell which nodes are central for the information diffusion process.

In this chapter, we extend the LIM model so that we can identify the global influential nodes in an implicit information diffusion network. Our approach combines the LIM model with sparse learning method. In recent years, sparse learning has become a popular tool in machine learning and statistics. By jointly minimizing a smooth convex loss and a sparsity-inducing regularizer (e.g. ℓ_1-norm), sparse learning method can select the most relevant features from high-dimensional data. Utilizing the sparse learning framework, we introduce a special sparsity-inducing regularizer, group Lasso penalty [Yuan and Lin, 2006], in the LIM model and propose the corresponding SLIM model (Sparse Linear Influence Model). Our SLIM will automatically set the influence factors for those non-influential nodes to zero and hence select the remaining nodes as influential ones. Due to the non-smoothness of the penalty, the optimization problem for SLIM becomes quite challenging. To solve this optimization problem, we adopt the accelerated gradient descent framework (AGM) [Beck and Teboulle, 2009, Chen et al., 2009]. We show that for the proximal operator, which is the key step in AGM, there is a closed-form solution. Then, we directly apply fast-iterative shrinkage-thresholding algorithm (FISTA) to solve this optimization problem, which achieves an optimal convergence rate in $O(1/t^2)$, where t is the total number of iterations.

The proposed SLIM model can be applied to analysis of various types of social media. In this chapter, we specifically apply our SLIM model on the three social network datasets.

Using the SLIM model, we can efficiently predict the topics volume and select the global influential users simultaneously, which can provide a lot of useful information for companies and make it possible to develop more effective advertisement strategies.

3.2 LINEAR INFLUENCE MODEL

In this section, we introduce the *linear influence model (LIM)* proposed by Yang and Leskovec (2010), based on which we develop our SLIM model. Assume that there are N nodes and K different contagions diffused among these nodes over time, where each contagion can infect a subset of nodes at any time. We discretize the entire time span into T units: $\{0, 1, \ldots, T\}$. Let $V_k(t)$ be the total *volume* of the contagion k between $t-1$ and t, i.e., the total number of times that all nodes get infected by the contagion k between $t-1$ and t; and $M_{uk}(t)$ be the number of times that the node u gets infected by the contagion k in $[t-1, t]$. The LIM assumes a linear model between $V_k(t)$ and $M_{uk}(t)$:

$$V_k(t+1) = \sum_{u=1}^{N} \sum_{l=0}^{L-1} M_{uk}(t-l) I_u(l+1) + \epsilon_k(t+1), \tag{3.1}$$

where $\mathbf{I}_u = (I_u(1), \ldots, I_u(L)) \in \mathbb{R}^{L \times 1}$ is the *non-negative influence function* for the node u, the term we want to estimate. The length L of the vector $\mathbf{I}_u$ means that the influence of a node is assumed to drop to zero after L time units. $\epsilon_k(t)$ is the i.i.d. zero-mean Gaussian noise. Following [Yang and Leskovec, 2010], $V_k(t)$ and $M_{uk}(t)$ can be organized into the matrix form as shown in Figure 3. We further define the node influence function $\mathbf{I} \in \mathbb{R}^{L \cdot N \times 1}$ to be the concatenation of $\mathbf{I}_1, \ldots, \mathbf{I}_N$ and matrix $\mathbf{M}_k = (\mathbf{M}_{1k}, \ldots, \mathbf{M}_{Nk}) \in \mathbb{R}^{T \times L \cdot N}$. Given the matrix form of $\mathbf{V}_k$, $\mathbf{M}_k$ and $\mathbf{I}$, Eq. (3.1) can be written into a more compact form as follows:

$$\mathbf{V}_k = \mathbf{M}_k \mathbf{I} + \boldsymbol{\epsilon}. \tag{3.2}$$

Based on (3.2), LIM estimates the non-negative influence function by solving a non-negative

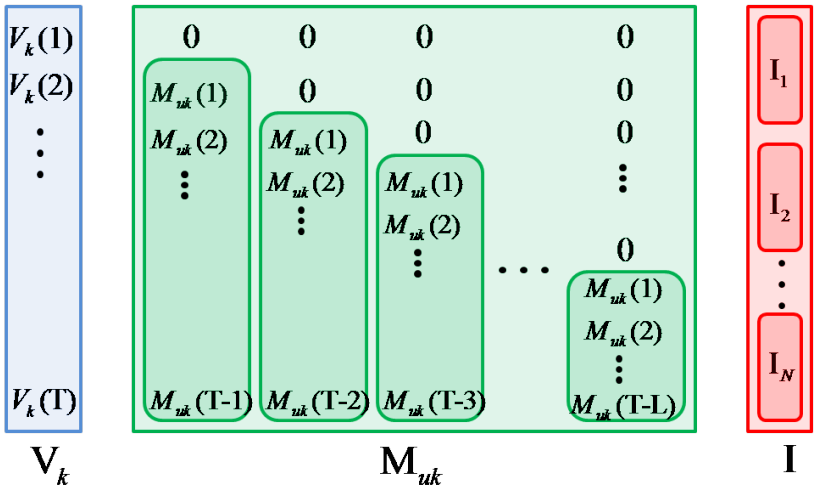


Figure 3: Volume vector $\mathbf{V}_k \in \mathbb{R}^{T\times 1}$, lower-triangular matrix $\mathbf{M}_{u,k} \in \mathbb{R}^{T\times L}$ and influence vector $\mathbf{I} \in \mathbb{R}^{L\cdot N\times 1}$. This figure is adopted from [Yang and Leskovec, 2010].

least square problem:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\textstyle\sum_{k=1}^{K} \|\mathbf{V}_k - \mathbf{M}_k \cdot \mathbf{I}\|_2^2, \\ \text{subject to} \quad & \mathbf{I} \geq 0 \end{aligned} \tag{3.3}$$

where $\|\cdot\|_2$ is the vector l_2-norm. By plugging the estimated influence function into (3.1), one can predict the total volume at the future time $T+1$ for each contagion k.

3.3 SPARSE LINEAR INFLUENCE MODEL

Although LIM can roughly model the influence for each node, it cannot tell which nodes are central in the network. In this section, we propose to extend the vanilla LIM to identify hub nodes (or the most influential users in social networks) in an implicit information diffusion network. We introduce a special sparsity-inducing regularizer, group Lasso penalty [Yuan and Lin, 2006], in the LIM model and propose the corresponding SLIM model (Sparse Linear

Influence Model). Our SLIM will automatically set the influence factors to zero for those non-influential nodes and hence select the remaining nodes as influential ones. In particular, SLIM is formulated as follows:

$$\begin{aligned} \text{minimize} \quad & \Psi(\mathbf{I}) \doteq \tfrac{1}{2}\|\mathbf{V} - \mathbf{M} \cdot \mathbf{I}\|_2^2 + \lambda \textstyle\sum_{u=1}^{N} \|\mathbf{I}_u\|_2 \\ \text{subject to} \quad & \mathbf{I} \geq 0. \end{aligned} \tag{3.4}$$

Where each $\mathbf{I}_u = (I_u(1), \ldots, I_u(L))$ is the influence vector for the u-th node. We take the l_2-norm on each $\mathbf{I}_u$ so that it will encourage all the elements in $\mathbf{I}_u$ to go to zero together.

With the estimated $\widehat{\mathbf{I}}$, we can directly identify those most influential nodes. Let us denote the set of hub nodes by $\widehat{\mathcal{U}}$:

$$\widehat{\mathcal{U}} = \{u : \|\widehat{\mathbf{I}}_u\|_2 > 0\}. \tag{3.5}$$

The cardinality of $\widehat{\mathcal{U}}$ (i.e., the number of hub nodes) is controlled by the regularization parameter λ. When $\lambda \to \infty$, then all $\widehat{\mathbf{I}}_u$ will become zero and none of the nodes will be selected as hub nodes (i.e., $\widehat{\mathcal{U}} = \emptyset$). On the other hand, if $\lambda \to 0$, $\widehat{\mathcal{U}} = \{1, \ldots, N\}$. Put another way, the smaller λ is, the more nodes will be selected as hub nodes. In practice, we could tune the regularization parameter to achieve the desirable number of hub nodes. With the estimated $\widehat{\mathbf{I}}$, we can predict the total volume of contagion k at $T+1$ by $\widehat{V}_k(T+1) = \sum_{u=1}^{N} \sum_{l=0}^{L-1} M_{uk}(T-l)\widehat{I}_u(l+1)$.

Although the formulation for SLIM is a convex optimization problem, the non-smoothness arising from $\|\mathbf{I}_u\|_2$ and the additional non-negativity constraint make the computation quite challenging. In the next section, we present an efficient and scalable solver for SLIM which could be applied to solve large-scale problems.

3.4 OPTIMIZATION TECHNIQUE

In this section, we present an efficient optimization algorithm for solving SLIM in (3.4). We first introduce some necessary notations. Let $f(\mathbf{I}) = \frac{1}{2}\|\mathbf{V} - \mathbf{M} \cdot \mathbf{I}\|_2^2$ be the smooth part of the objective in (3.4). The gradient of $f(\mathbf{I})$ over $\mathbf{I}$ takes the following form:

$$\nabla f(\mathbf{I}) = \mathbf{M}^T(\mathbf{M}\mathbf{I} - \mathbf{V}) \tag{3.6}$$

In addition, $\nabla f(\mathbf{I})$ is Lipschitz continuous with the constant $L = \lambda_{\max}(\mathbf{M}^T\mathbf{M})$, which is the maximum eigenvalue of $\mathbf{M}^T\mathbf{M}$. In other word, we have for any $\mathbf{I}_1$,$\mathbf{I}_2$,

$$\|\nabla f(\mathbf{I}_1) - \nabla f(\mathbf{I}_2)\|_2 \leq L\|\mathbf{I}_1 - \mathbf{I}_2\|_2$$

With these notations in place, we use the accelerated gradient descent method framework. In particular, we adopt the fast iterative shrinkage-thresholding algorithm (FISTA) [Beck and Teboulle, 2009]. It is known that this algorithm has the *optimal* convergence rate of $O(1/t^2)$.

To apply this algorithm, the main difficulty is how to compute the first step, which is the so-called *proximal mapping* step . Firstly, we observe Step 1 can be written as:

$$\mathbf{I}^t = \text{argmin}_{\mathbf{I}\geq 0}\frac{1}{2}\|\mathbf{I} - \mathbf{W}\|_2^2 + \frac{\lambda}{L}\sum_{u=1}^{N}\|\mathbf{I}\|_2 \tag{3.7}$$

where $\mathbf{W} = \mathbf{J}^t - \frac{1}{L}\nabla f(\mathbf{J}^t)$.

Since (3.7) is separable in terms of $\mathbf{I}_u$, we solve it by each $\mathbf{I}_u$ independently, i.e.,

$$\mathbf{I}_u^t = \text{argmin}_{\mathbf{I}_u\geq 0}\frac{1}{2}\|\mathbf{I}_u - \mathbf{W}_u\|_2^2 + \frac{\lambda}{L}\|\mathbf{I}_u\|_2 \tag{3.8}$$

For simplicity, we use $\mathbf{x}$, $\mathbf{w}$, λ to denote $\mathbf{I}_u$, $\mathbf{W}_u$ and $\frac{\lambda}{L}$ respectively. The closed form solution of the above minimization problem can be characterized by the following Theorem.

Theorem 1. *The optimal solution for the following optimization can be represented as following,*

$$\text{argmin}_{\mathbf{x}\geq 0}\frac{1}{2}\|\mathbf{x} - \mathbf{w}\|_2^2 + \lambda\|\mathbf{x}\|_2 \tag{3.9}$$

Algorithm 1 FISTA Algorithm for SLIM (i.e., Minimize $\Psi(\mathbf{I})$ in (3.4))

Input Parameters: $\mathbf{V}$ and $\mathbf{M}$ and the Lipschitz constant $L = \lambda_{\max}(\mathbf{M}^T\mathbf{M})$

Initialization: Set $\mathbf{J}^0 = \mathbf{0}$ and the step-size parameter $\theta_0 = 1$.

Iterate for $t = 0, 1, 2, \ldots,$ until convergence

1. $\mathbf{I}^t = \arg\min_{\mathbf{I}\geq 0}\langle I, \nabla f(\mathbf{J}^t)\rangle + \frac{L}{2}\|\mathbf{I} - \mathbf{J}^t\|_2^2 + \lambda\sum_{u=1}^{N}\|\mathbf{I}_u\|_2$
2. $\theta_{t+1} = \frac{1+\sqrt{1+4\theta_t^2}}{2}$
3. $\mathbf{J}^{t+1} = \mathbf{I}^t + \frac{\theta_t - 1}{\theta_{t+1}}(\mathbf{I}^t - \mathbf{I}^{t-1})$

Output: $\mathbf{I}^t$

Let $\mathcal{P} = \{i : w_i > 0\}$ be indices for the positive values in $\mathbf{w}$ and let $\mathcal{P}^C$ be the complement of $\mathcal{P}$. Then the optimal $\mathbf{x}$ is,

$$\begin{cases} \mathbf{x}_{\mathcal{P}} = \max(1 - \frac{\lambda}{\|\mathbf{w}_{\mathcal{P}}\|_2}, 0)\mathbf{w}_{\mathcal{P}} \\ \mathbf{x}_{\mathcal{P}^c} = 0 \end{cases}$$

Proof. Firstly, we utilize the fact that the dual norm of l_2-norm is l_2-norm, so that we could present $\|\mathbf{x}\|_2$ as

$$\|\mathbf{x}\|_2 = \max_{\|\mathbf{y}\|_2\leq\lambda}\mathbf{y}^T\mathbf{x}$$

Therefore, (3.9) can be reformulated as:

$$\min_{\mathbf{x}\geq 0}\max_{\|\mathbf{y}\|_2\leq\lambda}\frac{1}{2}\|\mathbf{x} - \mathbf{w}\|_2 + \mathbf{y}^T\mathbf{x}$$

Interchange the min and max, we have,

$$\max_{\|\mathbf{y}\|_2\leq\lambda}\left(\min_{\mathbf{x}\geq 0}\frac{1}{2}\|\mathbf{x} - \mathbf{w}\|_2 + \mathbf{y}^T\mathbf{x}\right) \tag{3.10}$$

Now we assume $\mathbf{y}$ is given, we first present the optimal $\mathbf{x}$ as a function of $\mathbf{y}$ using the KKT condition.

In particular, $\min_{\mathbf{x}\geq 0}\frac{1}{2}\|\mathbf{x} - \mathbf{w}\|_2 + \mathbf{y}^T\mathbf{x}$ can be decomposed into each component of $\mathbf{x}$:

$$\min_{x_i \geq 0} \frac{1}{2}(x_i - (w_i - y_i))^2$$

To derive the optimal solution, we introduce the Lagrange multiplier $\mu \geq 0$ for the constraint $x_i \geq 0$. The Lagrange function takes the form

$$L(x_i, \mu) = \frac{1}{2}(x_i - (w_i - y_i))^2 - \mu x_i$$

According to stationary condition, we have:

$$x_i = w_i - y_i + \mu$$

The complementary slackness condition implies that:

$$\mu x_i = \mu(w_i - y_i + \mu) = 0$$

If $w_i > y_i$, since $\mu \geq 0$, $w_i - y_i + \mu > 0$ and the complementary slackness implies that $\mu = 0$ and hence $x_i = w_i - y_i$. On the other hand, if $w_i \leq y_i$, we have $\mu = y_i - w_i$ and hence $x_i = 0$. In sum, we have:

$$x_i = \max(w_i - y_i, 0) \tag{3.11}$$

Now we plug the above relation back into (3.10) and obtain that

$$\max_{\|\mathbf{y}\|_2 \leq \lambda} \sum_{i=1}^{p} (-\frac{1}{2}y_i^2 + y_i w_i)\mathbb{I}(w_i > y_i) + \frac{1}{2}w_i^2 \mathbb{I}(w_i \leq y_i) \tag{3.12}$$

where $\mathbb{I}(\cdot)$ is the indicator function. Let

$$\phi(y_i) = (-\frac{1}{2}y_i^2 + y_i w_i)\mathbb{I}(w_i > y_i) + \frac{1}{2}w_i^2 \mathbb{I}(w_i \leq y_i)$$

We present its graphical illustration in Fig 4.

To maximize (3.12) over $\mathbf{y}$ under the constraint $\|\mathbf{y}\|_2 \leq \lambda$, we discuss the cases when $w_i \leq 0$ and $w_i > 0$ separately:

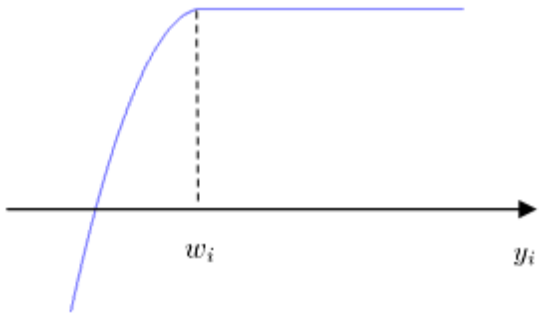


Figure 4: Graphical illustration for the function $\phi(y_i) = (-\frac{1}{2}y_i^2 + y_i w_i)\mathbb{I}(w_i > y_i) + \frac{1}{2}w_i^2\mathbb{I}(w_i \leq y_i)$

- When $w_i \leq 0$, we could simply set $y_i = 0$ so that $\phi(y_i)$ achieves its maximum value. Then according to (3.11), we have

$$x_i = 0$$

- When $w_i > 0$, let $\mathcal{P} = \{i : w_i > 0\}$ be the set of their indices. If $\|\mathbf{w}_\mathcal{P}\|_2 \leq \lambda$, we could simply set $\mathbf{y}_\mathcal{P} = \mathbf{w}_\mathcal{P}$ so that $\sum_{i\in\mathcal{P}} \phi(y_i)$ achieves its maximum. According to (3.11), $\mathbf{x}_\mathcal{P} = 0$. On the other hand, if $\|\mathbf{w}_\mathcal{P}\|_2 > \lambda$, we obtain the optimal $\mathbf{y}_\mathcal{P}$ by shrinking $\mathbf{w}_\mathcal{P}$ to the ball $\|\cdot\|_2 \leq \lambda$, i.e.,

$$\mathbf{y}_\mathcal{P} = \frac{\lambda}{\|\mathbf{w}_\mathcal{P}\|_2}\mathbf{w}_\mathcal{P}$$

Then

$$\mathbf{x}_\mathcal{P} = (1 - \frac{\lambda}{\|\mathbf{w}_\mathcal{P}\|_2})\mathbf{w}_\mathcal{P}$$

By summarizing above two cases, we obtain the results in Theorem 1, which completes our proof. □

Since the proximal mapping step (Step 1) can be solved in an analytical form, thus according to Theorem 4.4. in [Beck and Teboulle, 2009] , we can immediately obtain the convergence rate of Algorithm 1 as stated in the next Theorem.

Theorem 2. *Let* $\mathbf{I}^*$ *be the minimizer of* $\Psi(\mathbf{I})$ *defined in* (3.4) *and* $\Psi(\mathbf{I}^*)$ *be the optimal objective value. Then if we run Algorithm 1 for* t *iterations, the gap between the objective value at the* t*-th iteration and the optimal objective value is upper bounded by:*

$$\Psi(\mathbf{I}^t) - \Psi(\mathbf{I}^*) \leq \frac{2L\|\mathbf{I}^0 - \mathbf{I}^*\|_2^2}{(t+1)^2}$$

One may refer to [Beck and Teboulle, 2009] for the proof and this rate is known to be optimal under the first-order black-box model [Nesterov, 2003].

3.5 EXPERIMENTS

In this section, we evaluate the performance of SLIM model on three social network datasets: Twitter, Facebook, Plurk (the latter two are traditional chinese version). Here, we first describe the data collection and topic (contagion) extraction procedures, and then compare SLIM with several competitors on the prediction task. In addition, we present some qualitative analysis results on the detected global influential users(nodes) for twitter dataset.

3.5.1 Data collection

For twitter dataset, we use the tweets from Twitter, which are text-based messages of up to 140 characters. Prior to crawling a corpus of tweets, we need to collect a set of twitterers who composed those tweets. Following the work in [Xiang et al., 2012b], we resort to the public company profiles on TechCrunch [1], and first extract a list of 1000 Twitter usernames. TechCrunch is one of the leading technology media sites, dedicated to profiling startups, new products, and breaking finance and tech news. Using the Twitter search API [2], we crawl all the tweets of these twitter users between January 2009 and November 2011, which include the full text, the author, and the time-stamp for each tweet. In addition, we also collect the profile for each individual user, including followers count, the location, a short biography, etc.

[1] http://techcrunch.com/

[2] http://apiwiki.twitter.com/Twitter-API-Document

We conduct a standard Twitter-data cleaning procedure as used in many existing literatures [Yang and Leskovec, 2010, Xiang et al., 2012a, Williams and Katz, 2012]: 1) remove the URLs or shortened URLs and the linked webpage from tweets; 2) remove words in the form of "@username" from the tweets; 3) remove all the stopwords which refer to words that occur very often yet bear little actual meaning such as "the", etc. Also we remove those special symbols, e.g., # (refers to a hashtag), *rt* (which is the sign for retweets), dollar sign $; 4) remove non-English characters, numbers, punctuations. 5) develop the heuristic to condense 3 or more than 3 repetitive letters into a single letter. For example, "yyeeaahh" will be reduced to "yeah", while "committee" remains intact. This heuristic by no means covers all stylistic variations. However, given the high frequency of such repetitive letters, this heuristic is effective in helping reduce the volume of the vocabulary. After the pre-processing, we convert each tweet into a bag-of-words representation. In our dataset, on average, each user has 2,611.825 tweets and the maximum number of tweets for a specific user is 10,986. Following [Yang and Leskovec, 2010], we further select $N = 786$ users out of 1000 users with at least 1,000 tweets during these three years and use them to construct the so-called active node set for modeling the total volume of contagions to construct our implicit network.

For facebook and plurk datasets, we collaborate with the researchers in Institute for Information Industry in Taiwan to collect the traditional chinese version social network data through public APIs. First we manually label search terms as the input query for each API and collect the posts and responses between Jan. 1st, 2011 to May. 15th, 2011 on each social network platform. Top 1000 users are selected to construct the active node set, based on the number of posts created by all the participants in historical data.

3.5.2 Topic modeling

When applying SLIM, the first thing is to define semantically meaningful contagions (corresponding to topics). A straightforward way of defining topics is to directly use word as topic (e.g., LinkedIn). However, a single word may not be rich enough to represent a broad topic (e.g., social network sites). Another possible way is to use the hashtag for twitter dataset

Apple	apple	iphone	ios	ipad	ipod	verizon	siri	sprint	itunes	jailbreak
Samsung	phone	android	samsung	tablet	galaxy	phones	nexus	smartphone	tablets	dual
Startup	startup	funding	entrepreneurs	startups	raises	clients	investors	techcrunch	partners	founders
Finance	nytimes	wsj	nyc	gutschein	stock	bloomberg	tax	finance	bonus	euro

Table 2: Top 10 words from four interesting topics learned by LDA for twitter dataset.

(e.g., #SouthAfrica). However, the frequency of "#hashtag" is low in our twitter corpus, rendering the use of them in defining topics inappropriate. In our experiment for twitter dataset, we construct the topics using Latent Dirichlet Allocation (LDA) [Blei et al., 2003] , which is a renowned generative probabilistic model for topic discovery. To distill the topics that twitter users are interested in, documents should naturally correspond to tweets. We take the LDA implementation [Phan and Nguyen, 2007] , which uses the Gibbs sampling for parameter estimation and inference [Griffiths and Steyvers, 2004]. In particular, LDA is conditioned on three parameters, i.e. hyperparameters α, β and topic number K. Here they are set as $K = 50$, $\alpha = 50/K$ and $\beta = 0.1$. We set the number of Gibbs sampling iterations to be 5000 to guarantee that the sample procedure fully converges. For each topic, we only keep the top 100 words since probability on these 100 words has already achieved a significant portion. Table 2 lists the top 10 words from four example topics learned by LDA for twitter dataset. For each tweet $\mathbf{w}$, LDA will make inference on $\Pr(\text{topic of } \mathbf{w} = k|\mathbf{w})$. Thus, we could simply set the topic for each tweet $\mathbf{w}$ using the maximum a posterior rule, i.e, the topic for $\mathbf{w}$ is $\max_{1\leq k\leq K} \Pr(\text{topic of } \mathbf{w} = k|\mathbf{w})$. Given the topic for each tweet, we can directly construct the matrix $\mathbf{M}_k$, where $M_{u,k}(t)$ is the number of times that the user u mentioned the topic k in $[t-1, t]$. And the volume $V_k(t)$ is defined by the total number of tweets that the topic k appears in $[t-1, t]$ over the entire 1,000 users.

For facebook and plurk datasets, since they are the traditional chinese versions, it is not easy to apply LDA directly on these datasets. Instead, we choose the manually labeled search terms which are used in data collection stage as the interesting topics. There are 118 interesting topics for plurk and 61 ones for facebook. Each interesting topic is a single key-

Facebook	士林夜市 shilin night market	海嘯+地震+震災 tsunami+earthquake+disaster	新店+龍捲風 xindian district+tornado	輻射+核電 radiation+nuclear power
Plurk	奢侈稅 luxury tax	威廉+凱特 William+Kate	全聯+省錢+電話 pxmart+economize+phone	危險小吃 dangerous snacks

Figure 5: Examples of interesting topics in Facebook and Plurk datasets.

word or a combination of some related keywords. Figure 5 lists some examples of interesting topics in facebook and plurk datasets. Since these two social network datasets are traditional chinese version, we provide the translation for each interesting topic in Figure 5. For each post $\mathbf{w}$, we can simply set the topic k for $\mathbf{w}$ if it contains the keywords of this topic. Given the topic for each post, we can follow the same procedure as discussed previously in twitter dataset to construct the matrix $\mathbf{M}_k$ and $V_k(t)$.

3.5.3 Quantitative analysis

We evaluate the prediction performance of SLIM. Since the quadratic loss is adopted in our modeling, the prediction mean-squared error (MSE) is the most suitable evaluation metric. The MSE is the second moment of the error, which can be decomposed into the squared of bias plus the variance term (known as the bias-variance decomposition [Hastie et al., 2009]). The relationship between MSE(Total Error), the squared of bias, variance with respect to the *model complexity* can be illustrated in Figure 6 [3]. In this thesis, we evaluate SLIM on a time series prediction task using the prediction mean-squared error (MSE):

$$\text{MSE} := \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{T}\left(\sum_{t=1}^{T}(\widehat{V}_k(t) - V_k(t))^2\right)\right), \tag{3.13}$$

where $\widehat{V}_k(t)$ is the predicted total volume for the contagion k at the time t using the data from previous time. Morevoer we define *prediction error* for each topic k as $\frac{1}{T}\left(\sum_{t=1}^{T}(\widehat{V}_k(t) - V_k(t))^2\right)$.

[3] http://scott.fortmann-roe.com/docs/BiasVariance.html

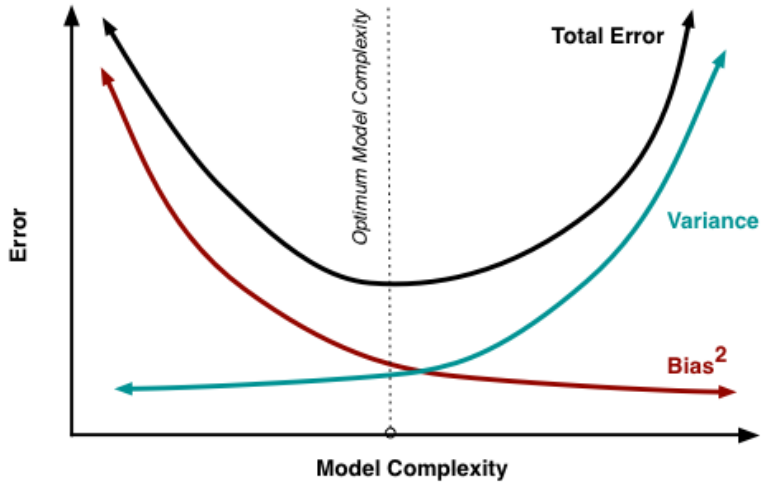


Figure 6: MSE v.s. Model Complexity and Bias-Variance Tradeoff.

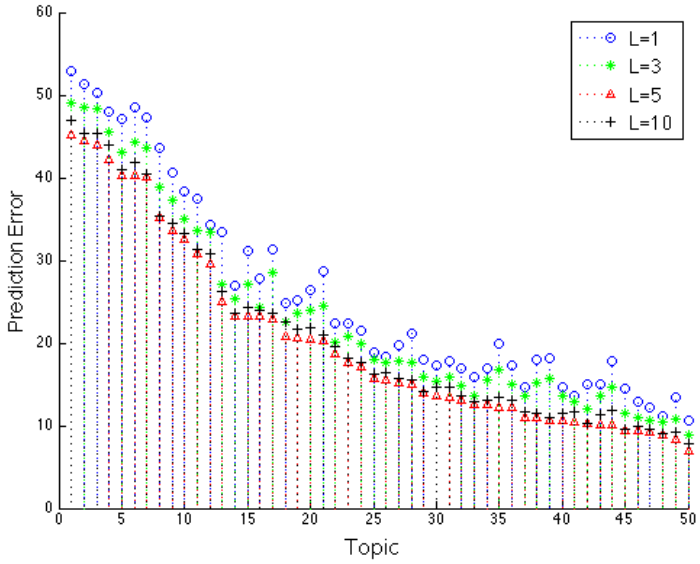


Figure 7: Comparison of prediction error among different influence decaying times on each of 50 topics.

To apply the model, some parameters (e.g.,time-tag L, regularization parameters λ) need to be determined since they control the model complexity. The model complexity of SLIM increases as the parameter L increases and/or λ decreases. For exmaple, parameter L denotes how long it takes the influence of a user to decay to zero. In Figure 7, we vary the parameter L and fix the parameter λ and report prediction error $\frac{1}{T}\left(\sum_{t=1}^{T}(\widehat{V}_k(t) - V_k(t))^2\right)$ of each topic k in an descending order. As we can see, the prediction error decreases as L goes larger from 1 to 5 and achieves the local minimum and then increases again when $L = 10$. This exactly corresponds to the phenomena explained in Figure 6. Thus in the experiment, we set $L = 5$. In our time-series data analysis setting, it is hard to apply cross validation since if the validation set is in the middle of the entire time sequence, the training set will not be consecutive in time. Therefore, a common way to tune parameters is to split the data into two portions: a training set and a validation set. In particular, we split the first 60% tweets in time as the training set and the last 40% as the validation set. We choose the best parameter that lead to the minimum prediction MSE on the validation set. In our experiment, we set one day as the time unit for twitter dataset and one hour for facebook and plurk datasets. For each contagion, we set the start ($t = 0$) of $V_k(t)$ to be the first time a node has been contaminated for contagion k and the length of the volume time series $T = 450$. The size of matrix $\mathbf{M}$ is $NL \times KT$, which is 8.8425×10^7 for twitter dataset. In our experiment, we use the validation set to tune the parameter λ and vary $\lambda \in \{1, 5, 10, 50, 100, 200, 300, 400, 500, 1000, 1500, 2000\}$ by grid search, where λ determines the number of selected influential users. The best $\lambda = 400$ for the twitter dataset. After determining the parameters L and λ, we combine the training and validation sets to re-train the model with the best selected parameters and estimate the influence function for each user. The final prediction MSE is reported on the entire time span. In our SLIM model, the degree of freedom is the dimension of influential function $\mathbf{I}$, which equals to $N \times L$ (i.e. 3930 for twitter data, 5000 for facebook and plurk datasets).

Baseline methods: We firstly compare the performance of our SLIM model with two baseline time series prediction methods:

- **1-time lag predictor:** it simply takes the volume at the current time as the prediction for the volume at the next time, $\widehat{V}_k(t+1) = V_k(t)$.

	1-time lag	L-time average	L-order autoregressive	SLIM
Twitter	48.17	43.49	35.75	20.24
Facebook	56.22	49.73	40.28	28.42
Plurk	43.56	39.13	31.74	16.17

Table 3: Comparison of the prediction MSE with three baseline methods.

- **L-time average predictor:** it predicts the volume at the next time as the average volume over the previous L time units, $\widehat{V}_k(t+1) = \frac{1}{L}\sum_{l=0}^{L-1} V_k(t-l)$.
- **L-order autoregressive predictor:** it predicts the volume at the next time as the weighted sum over the previous L times units, $\widehat{V}_k(t+1) = \sum_{l=0}^{L-1} \widehat{a}_l V_k(t-l)$, where $\widehat{a}_0, \widehat{a}_1, \ldots, \widehat{a}_{L-1}$ are the learned parameters of the model.

The comparison of the prediction MSE on the three datasets is shown in Table 3. Note that we set $L = 5$ in L-time average predictor and L-order autoregressive predictor as SLIM for a fair comparison. 1-time lag predictor is the worst, followed by L-time average predictor and then L-order autoregressive predictor. Note that the first two predictors are the special cases of L-order autoregressive predictor. Our SLIM model outperforms three baseline methods. Our results suggest that: 1) We obtain a substantial benefit from introducing the matrix $\mathbf{M}$ which considers more refined observations of each node; 2) The nodes' different levels of influence function $\mathbf{I}$ can efficiently models the volume $\mathbf{V}$; 3) The information diffusion process (i.e. time series prediction for volume) can be modeled well enough by a small subset of users' influence function $\mathbf{I}$. In other words, it is necessary to select the influential users for better predicting the volume.

Influential nodes selection methods: Next, we compare our SLIM algorithm to several highly relevant competitors on detecting influential nodes on three datasets.

- **In-degree selection:** select users based on their total number of followers (Twitter dataset) or total number of friends (Facebook and Plurk datasets) (i.e.,in-degree) [Cha

et al., 2010].

- **PageRank:** select users with only link structure of the network (Twitter based on following relationship, Facebook and Plurk based on friendship relationship) [Brin and Page, 1998].
- **TwitterRank:** one of the state-of-the-art methods for selecting topic-sensitive influential users of micro-blogging services. [Weng et al., 2010].
- **Random selection:** randomly select influential users

The purpose of this comparison is not to build a perfect time series predictor. Rather, we aim to evaluate whether the influential nodes selected by different methods are reasonable and to what degree the prediction accuracy can be attributed to the selection methods. All the first three methods require the network structure given as the prior knowledge. In our twitter dataset, we use the standard "following relationship" to construct the network structure. In facebook and plurk datasets, we use the "friendship relationship" instead. The first two methods, which are the most classical methods for hub node detection, select a common set of users across all different topics; while the TwitterRank is a topic-sensitive selection method. With the selected users from each method, we adopt the LIM for the prediction task. Here, SLIM is listed as the last competitor to investigate the benefit of introducing sparsity-inducing regularizer in LIM.

The comparison results are presented in Figure 8 and Table 4. More specifically, in Figure 8, we report the prediction error $\frac{1}{T}\left(\sum_{t=1}^{T}(\widehat{V}_k(t) - V_k(t))^2\right)$ of each topic k in an descending order ranked by SLIM for the Twitter dataset. In Table 4, we further report the prediction MSE over all topics on the three datasets. Note that we select the same number of influential users as SLIM for a fair comparison. Since Twitterrank is specific for Twitter, where a directed graph is formed with the twitterers and the following relationships among them. There is an edge between two twitter users if there is following relationship between them, and the edge is directed from follower to friend. However, for the other two datasets facebook and plurk, there are no such "following" relationship, thus we use friendship graph where two friends have two directed edge from each other. As shown from Figure 8 and Table 4, the random selection performs the worst, followed by In-degree selection, and then pagerank for all of three datasets. For Facebook dataset, our SLIM model per-

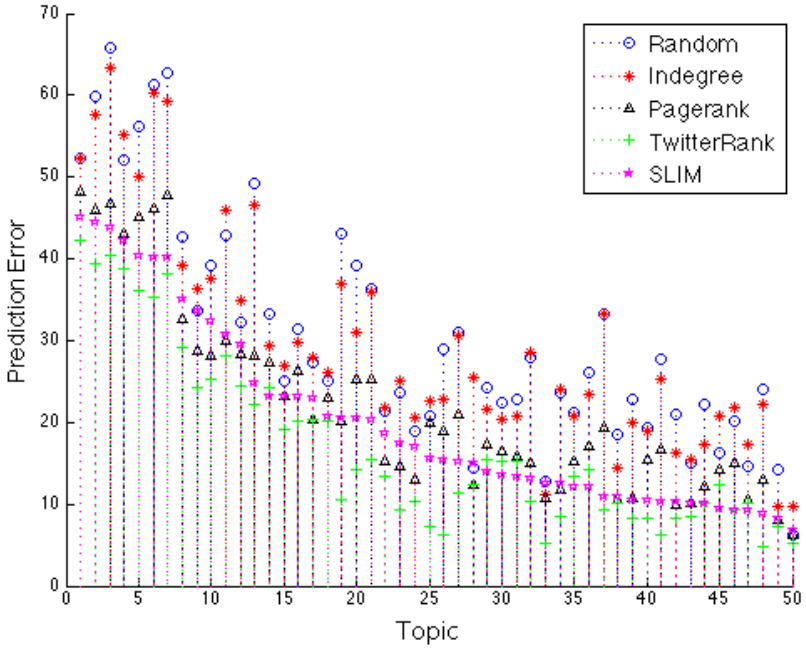


Figure 8: Comparison SLIM with other competitors of prediction error for each of 50 topics.

	Random	Indegree	Pagerank	Twitterrank	SLIM
Twitter	32.43	29.98	22.95	16.21	20.24
Facebook	41.28	38.52	30.96	29.34	28.42
Plurk	31.25	27.64	22.35	15.43	16.17

Table 4: Comparison of the prediction MSE with other selection methods.

forms best among all the competitors. TwitterRank performs better than SLIM on Twitter dataset since it considers the contagion information but our SLIM model doesn't consider the contagion-sensitive selection. We assume that each node has the same influence across all the contagions. Clearly, this assumption could be too restrictive for many applications. For different types of contagions, the set of the most influential nodes could be very different. Thus, we will further develop multi-task sparse linear influential model to select the topic-sensitive influential nodes, which will be described in the next Chapter 4. However, we find that TwitterRank doesn't perform much better on Facebook and Plurk datasets. The reason might be that "following relationship" is more related to information diffusion process than "friend relationship". User's followers are more interested in his posted tweets but friends may not. Again, we note that TwitterRank uses the information about network, which is unavailable in some practical situations. In contrast, our SLIM does not require the network structure and thus has wider applications.

3.5.4 Qualitative analysis

SLIM can detect global influential users for all of contagions. Since Twitter offers the good API to crawl the user profiles including full name, location, biography and etc, thus we analyze the selected influential users' profiles on Twitter dataset. Here, we tune the regularization parameter to select the top 35 most influential users on our twitter dataset.

- *The most influential users spread throughout the world*: We plot the geographical colorpleth map [Slocum et al., 2009] of total 1000 twitter users in Fig. 9 and colorpleth map of selected global influential users in Fig. 10 to show that which part of the world has more influential users. We also plot the more detailed locations of the selected influential users in Fig. 11. It is shown that the most global influential users spread all over the world, including Europe, Middle East, India, Indonesia, New Zealand, South America and North America. Most of the hot users are located in North America (16 out of 35), followed by the Europe (10 out of 35). There is no influential user in Africa, Australia and Antarctica. Since some countries in Asia like China doesn't use twitter frequently and some countries like Japan uses a different language twitter version, thus there are

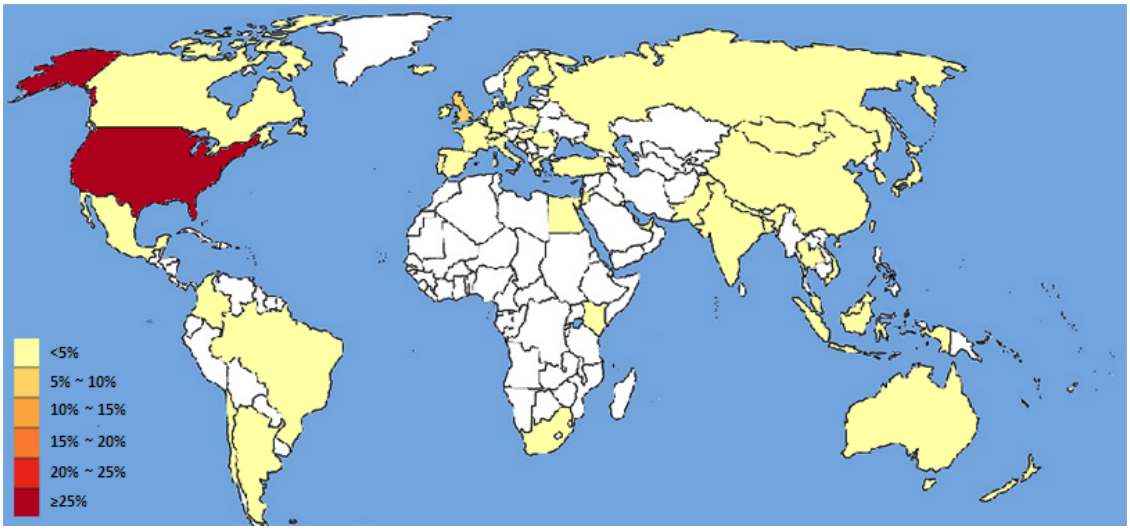


Figure 9: Colorpleth map of total 1000 users

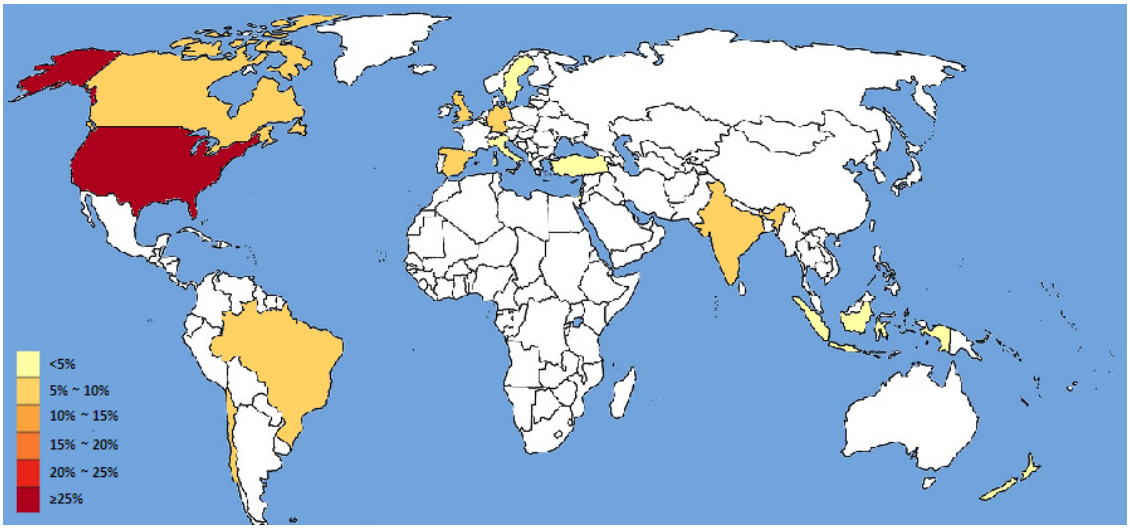


Figure 10: Colorpleth map of selected global influential users

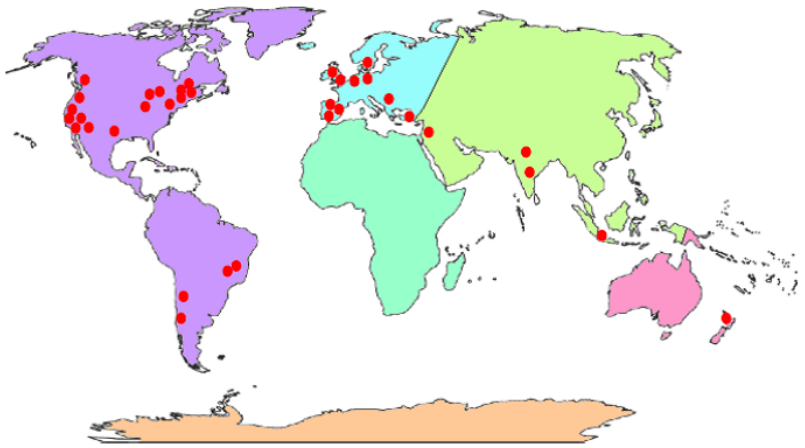


Figure 11: Detailed locations of selected global influential users

fewer selected hot users in Asia. Moreover, since most of twitter users are people or companies related to IT media and high technology, Africa has no hot users due to its low level technology development. From this result, we can conclude that, except for some countries where the twitter is blocked, North America and Europe are the world center for IT high technology development.

- *The followers count for influential users*:

  In the Fig. 12, we plot the number of followers for the total twitter users in a descending order with blue points. The red points represent the followers count for the selected global influential users. From the Fig. 12, one can observe that Twitter users with the intermediate number of followers have much higher influence than the highest in-degree users. Our results are consistent with the recent finding [Cha et al., 2010, Bakshy et al., 2011], which suggests that twitter users who have the most followers are not the most influential in terms of information diffusion.

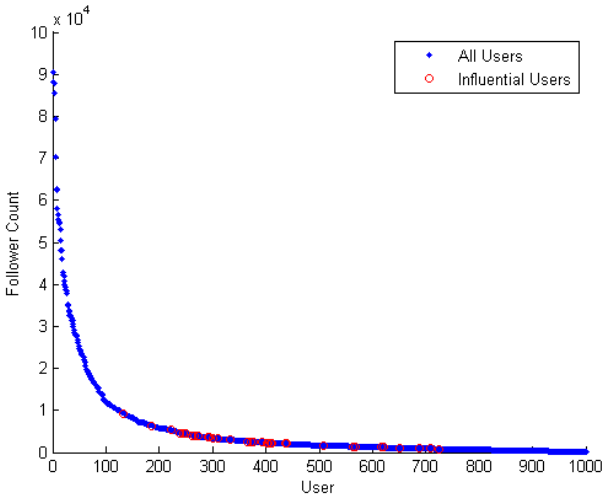


Figure 12: Follwers count of all twitter users and the selected global influential users

4.0 MULTI-TASK SPARSE LINEAR INFLUENCE MODEL

As we discussed in Section 2.3 in background part, most of the existing works for influential node detection suffer from two main problems: 1) it does require the network structure as a priori; 2) it can only detect influential nodes for either only one contagion or across all contagions. In the previous chapter 3, we solve the first issue: how to identify the global influential nodes without knowing the social network structures. However, LIM & SLIM assume a global influence function for each node without modeling the different levels of influence for different contagions. Thus SLIM model can not identify the contagion-sensitive influential nodes. In this chapter, we address the second issue by developing a new methodology which can predict the volume for each contagion and also automatically identify sets of the most influential nodes for different contagions without knowing the explicit network structure. In this work, we extend LIM by introducing a contagion-sensitive influence function for each node; and formulate the problem into a *convex optimization problem*, which jointly minimizes the prediction loss and a sparsity-inducing penalty function. Since our sparsity-inducing penalty is designed by extending the penalty in multi-task sparse Lasso [Obozinski et al., 2009] into its tensor form, we name our method as *multi-task sparse linear influential model (MSLIM)*. Our method is based on the linear influence model with two main advantages: 1) it does not require the network structure; 2) it can detect different sets of the most influential nodes for different contagions.

4.1 INTRODUCTION AND MOTIVATION

Although LIM can model the influence for each node and has been proven to be effective for predicting the future volume for each contagion, it cannot detect the most influential nodes in an implicit network. Thus, we extend the vanilla LIM and develop SLIM in chapter 3 to identify the most global influential users across all of contagions. However, both LIM and SLIM use a single influence function $\mathbf{I}_u$ for each node u as in (3.2), which is based on an underlying assumption that each node has the same influence across all the contagions. Clearly, this assumption could be too restrictive for many applications. For different types of contagions, the set of the most influential nodes could be very different. Some users might be active on some topics but are less influential on the other topics. As shown in the previous experimental results in section 3.5.3, TwitterRank performs better than SLIM on Twitter dataset since it considers the different levels of information on different contagions. To achieve contagion-sensitive node selection in an implicit network, we extend our SLIM model to the multitask sparse learning setting and propose the *multitask sparse linear influential model (MSLIM)* in Section 4.2, which can automatically select the influential nodes for different topics in an implicit network. Due to the non-smoothness and high-complexity of the penalty, the optimization problem for MSLIM becomes quite challenging. We firstly adopt the accelerated gradient method (AGM) [Nesterov, 2003, Beck and Teboulle, 2009] and prove that for the proximal mapping step there is an exact closed-form solution in Section 4.3. Therefore, the corresponding AGM achieves the optimal convergence rate in $O(1/t^2)$, where t is the total number of iterations. In Section 4.4, we apply the stochastic optimization method, regularized dual averaging, to solve both SLIM and MSLIM in a unified framework in order to better improve the effectiveness scalability.

In Section 4.5, we present experimental results on three social network datasets. We show that MSLIM can efficiently select the most influential users for specific contagions and achieve better prediction results. We also show the advantages of stochastic optimization (i.e. RDA) over deterministic optimization (i.e. FISTA) by comparing the CPU running time on these datasets.

Our new method so called MSLIM for detecting the contagion-sensitive influential nodes

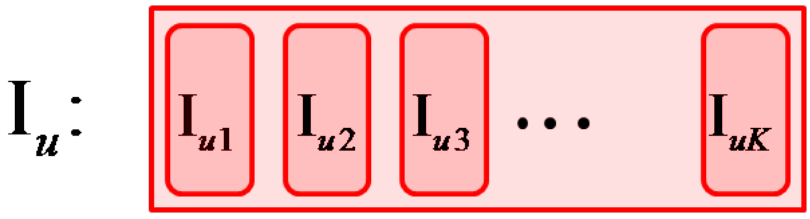


Figure 13: Influence function $\mathbf{I}_u \in \mathbb{R}^{L \times K}$ for the node u.

in an implicit information diffusion network, has the following three main advantages over state-of-the-art works:

1. MSLIM does not require the prior knowledge of the network structure.
2. MSLIM is a contagion-sensitive node selection method, which can detect different sets of influential nodes for different contagions.
3. MSLIM simultaneously conducts the diffusion prediction and influential node detection in a unified framework.

4.2 MULTI-TASK SPARSE LINEAR INFLUENCE MODEL

We first define the influence function by extending $\mathbf{I}_u$ in LIM into the so-called contagion-sensitive $\mathbf{I}_{uk} \in \mathbb{R}^{L \times 1}$, which is a L-length vector representing the influence of the node u on the contagion k. For each contagion k, let $\mathbf{I}^k \in \mathbb{R}^{L \cdot N \times 1}$ be the vector obtained by concatenating $\mathbf{I}_{1k}, \ldots, \mathbf{I}_{Nk}$, which corresponds to $\mathbf{I}$ in LIM. For each node u, we further define the influence matrix for the node u: $\mathbf{I}_u = (\mathbf{I}_{u1}, \ldots, \mathbf{I}_{uK}) \in \mathbb{R}^{L \times K}$, which is shown in Figure 13. Given the contagion-sensitive influence function, we assume a linear relationship between $\mathbf{V}_k$ and $\mathbf{M}_k$ as in (3.2) but replace $\mathbf{I}$ in (3.2) by $\mathbf{I}_k$ (see Figure 14):

$$\mathbf{V}_k = \mathbf{M}_k \mathbf{I}^k + \boldsymbol{\epsilon}. \tag{4.1}$$

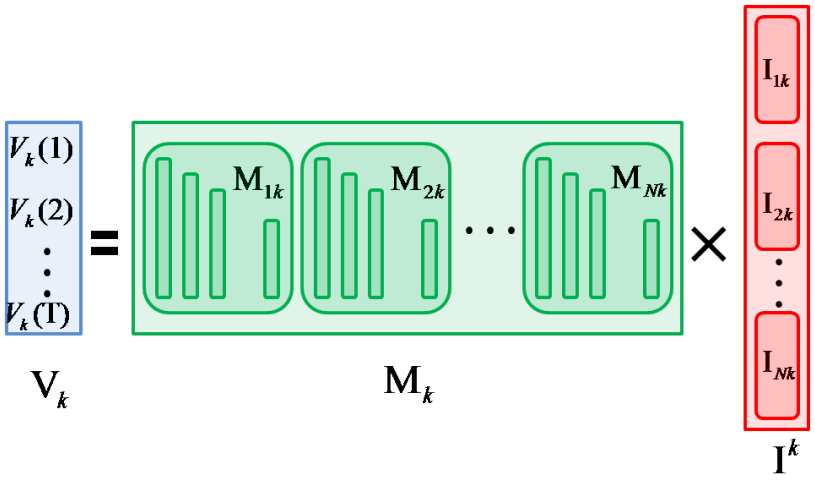


Figure 14: Linear relationship between $\mathbf{V}_k$ and $\mathbf{M}_k$.

We formulate the problem of contagion-sensitive influential node selection into a convex optimization problem, which jointly minimizes the square loss and a non-smooth sparsity-inducing penalty inspired by multi-task sparse learning [Obozinski et al., 2009]. In particular, we estimate the non-negative vector $\{\mathbf{I}_{uk}\}$ for all nodes and contagions by:

$$\begin{aligned} \text{minimize} \quad & \Psi(\mathbf{I}) \doteq \tfrac{1}{2}\sum_{k=1}^{K}\|\mathbf{V}_k - \mathbf{M}_k \cdot \mathbf{I}^k\|_2^2 + \lambda\sum_{u=1}^{N}\|\mathbf{I}_u\|_F + \gamma\sum_{u=1}^{N}\sum_{k=1}^{K}\|\mathbf{I}_{uk}\|_2 \quad (4.2) \\ \text{s.t.} \quad & \mathbf{I}_{uk} \geq 0, \qquad 1 \leq u \leq N, \quad 1 \leq k \leq K, \end{aligned}$$

where $\|\cdot\|_F$ is the matrix Frobenius norm. The penalty term $\|\mathbf{I}_u\|_F$ encourages the entire matrix $\mathbf{I}_u$ to be zero altogether, which means that the node u is non-influential for all different contagions. If the estimated $\|\mathbf{I}_u\|_F > 0$ (i.e., the matrix $\mathbf{I}_u$ is non-zero), a fine-grained selection is performed by the penalty $\sum_{u=1}^{N}\sum_{k=1}^{K}\|\mathbf{I}_{uk}\|_2$, which is essentially a group-Lasso penalty [Yuan and Lin, 2006] and can encourage the sparsity of vectors $\{\mathbf{I}_{uk}\}$. The sparsity-level (i.e., the number of selected nodes for each contagion) is controlled by the regularization parameters λ and γ. In particular, for a specific contagion k, one can identify the most influential nodes $\mathcal{U}_k$ with respect to this contagion as:

$$\mathcal{U}_k = \{u : \|\widehat{\mathbf{I}}_{uk}\|_2 > 0\}, \quad (4.3)$$

where $\widehat{\mathbf{I}}_{uk}$ is the optimal solution of (4.2).

With the estimated $\widehat{\mathbf{I}}_{uk}$, one can predict the total volume of the contagion k at $T+1$ by $\widehat{V}_k(T+1) = \sum_{u=1}^{N}\sum_{l=0}^{L-1} M_{uk}(T-l)\widehat{I}_{uk}(l+1)$. Therefore, our MSLIM can simultaneously conduct contagion-sensitive volume prediction and influential node detection in a unified framework.

We further note that as compared to the traditional multi-task sparse learning with the l_1/l_2 mixed-norm penalty [Obozinski et al., 2009] (l_1 corresponds to the summation and l_2 corresponds to the vector l_2-norm), the formulation of MSLIM is much more complicated, which makes the optimization (4.2) quite difficult. Firstly, the traditional multi-task sparse learning only has one l_1/l_2 mixed-norm penalty on the regression coefficients. In contrast, MSLIM in (4.2) has both l_1/l_F penalty $\sum_{u=1}^{N}\|\mathbf{I}_u\|_F$ and l_1/l_2 penalty $\sum_{u=1}^{N}\sum_{k=1}^{K}\|\mathbf{I}_{uk}\|_2$; and these two penalties are intertwined together in the sense that $\mathbf{I}_{uk}$ is a sub-matrix of $\mathbf{I}_u$. In addition, since the influence is always "positive" in the basic LIM model (3.1), MSLIM requires the non-negativity of $\{\mathbf{I}_{uk}\}$, which is generally not required by multi-task sparse learning. The complicated structure of the penalty function in MSLIM makes the corresponding optimization very challenging. Thus traditional methods for multi-task sparse learning (e.g., [Chen et al., 2009, Liu et al., 2009]) cannot be directly applied. In the next section, we will show how to adopt the FISTA algorithm [Beck and Teboulle, 2009] to solve MSLIM in (4.2) with a fast convergence rate of $O(1/t^2)$.

4.3 OPTIMIZATION TECHNIQUE

In this section, we present an efficient optimization algorithm to solve MSLIM in (4.2). Similar to SLIM, we adopt the accelerated gradient method, in particular, the popular fast iterative shrinkage-thresholding algorithm (FISTA) [Beck and Teboulle, 2009] in Algorithm 2. Since it only utilizes the gradient information of the squared loss, it is efficient and can be scaled to large problems.

To guarantee the optimal first-order convergence rate of an accelerated gradient method, it requires that the key step called *proximal mapping* has an exact analytical solution. For

Algorithm 2 FISTA Algorithm for MSLIM (i.e.. Minimize $\Psi(\mathbf{I})$ in (4.2))

Input Parameters: $\mathbf{V}$ and $\mathbf{M}$ and the Lipschitz constant $L = \lambda_{\max}(\mathbf{M}^T\mathbf{M})$

Initialization: Set $\mathbf{J}^0 = \mathbf{0}$ and the step-size parameter $\theta_0 = 1$.

Iterate for $t = 0, 1, 2, \ldots,$ until convergence

1. $$\mathbf{I}^t = \arg\min_{\mathbf{I}\geq 0} \sum_{u=1}^{N} \left(\langle I_u, \nabla f(\mathbf{J}_u^t)\rangle + \frac{L}{2}\|\mathbf{I}_u - \mathbf{J}_u^t\|_2^2 \right) + \lambda \sum_{u=1}^{N} \|\mathbf{I}_u\|_F + \gamma \sum_{u=1}^{N}\sum_{k=1}^{K} \|\mathbf{I}_{uk}\|_2$$
2. $\theta_{t+1} = \frac{1+\sqrt{1+4\theta_t^2}}{2}$
3. $\mathbf{J}^{t+1} = \mathbf{I}^t + \frac{\theta_t - 1}{\theta_{t+1}}(\mathbf{I}^t - \mathbf{I}^{t-1})$

Output: $\mathbf{I}^t$

our problem, the proximal mapping takes the form:

$$\mathbf{I}^t = \text{argmin}_{\mathbf{I}\geq 0} \sum_{u=1}^{N} \frac{1}{2}\|\mathbf{I}_u - \mathbf{W}_u\|_F^2 + \frac{\lambda}{L}\sum_{u=1}^{N}\|\mathbf{I}_u\|_F + \frac{\gamma}{L}\sum_{u=1}^{N}\sum_{k=1}^{K}\|\mathbf{I}_{uk}\|_2, \tag{4.4}$$

where $\mathbf{W}_u = \mathbf{J}_u^t - \frac{1}{L}\nabla f(\mathbf{J}_u^t)$ is a given matrix with the same size as $\mathbf{I}_u$ and L is the Lipschitz constant for the gradient of the squared loss. In theorem 3, we show that there is a close-form solution of the proximal mapping step in (4.4). Then we could directly apply FISTA [Beck and Teboulle, 2009] to solve MSLIM in (4.2), which achieves the optimal first-order convergence rate as in Theorem 2.

Theorem 3. *There is an analytical solution for proximal mapping step for MSLIM (i.e., first step in Algorithm 2), which takes the form in* (4.4).

Proof. We present the analytical solution for the proximal mapping in (4.4). As we can see, Eq. (4.4) can be decomposed into N independent problems where each problem only involves $\mathbf{I}_u$:

$$\mathbf{I}_u^t = \text{argmin}_{\mathbf{I}_u\geq 0}\frac{1}{2}\|\mathbf{I}_u - \mathbf{W}_u\|_F^2 + \frac{\lambda}{L}\|\mathbf{I}_u\|_F + \frac{\gamma}{L}\sum_{k=1}^{K}\|\mathbf{I}_{uk}\|_2 \tag{4.5}$$

For simplicity, we use $\mathbf{X}$, $\mathbf{W}$, λ, γ to denote $\mathbf{I}_u$,$\mathbf{W}_u$, $\frac{\lambda}{L}$ and $\frac{\gamma}{L}$ respectively so that Eq. (4.5) can be written as:

$$\text{argmin}_{\mathbf{X}\geq 0}\frac{1}{2}\|\mathbf{X}-\mathbf{W}\|_F^2+\lambda\|\mathbf{X}\|_F+\gamma\sum_{k=1}^{K}\|\mathbf{X}_k\|_2, \tag{4.6}$$

where $\mathbf{X}\in\mathbb{R}^{L\times K}$ and $\mathbf{X}_k$ is the k-th column of $\mathbf{X}$. Now we only need to find the analytical solution of (4.6), i.e., the solution not obtained by another optimization procedure.

Utilizing the dual-norm, $\|\mathbf{X}\|_F=\max_{\|\mathbf{Y}\|_F\leq\lambda}\langle\mathbf{X},\mathbf{Y}\rangle$ and $\|\mathbf{X}_k\|_2=\max_{\|\mathbf{Z}_k\|_2\leq\gamma}\langle\mathbf{Z}_k,\mathbf{X}_k\rangle$; and hence (4.6) can be reformulated as:

$$\max_{\|\mathbf{Y}\|_F\leq\lambda}\max_{\|\mathbf{Z}_k\|_2\leq\gamma}\left(\min_{\mathbf{X}\geq 0}\frac{1}{2}\|\mathbf{X}-\mathbf{W}\|_F^2+\langle\mathbf{X},\mathbf{Y}\rangle+\sum_{k=1}^{K}\langle\mathbf{Z}_k,\mathbf{X}_k\rangle\right) \tag{4.7}$$

Assuming that $\mathbf{Y}$ and $\{\mathbf{Z}_k\}_{k=1}^K$ are given, using the KKT condition, we can obtain the optimal $\mathbf{X}$ by solving the inner minimization problem in (4.7):

$$x_{lk}=\max\left(w_{lk}-y_{lk}-z_{lk},0\right). \tag{4.8}$$

We plug (4.8) back into (4.7). Now our goal is to solve $\mathbf{Y}$ and $\{\mathbf{Z}_k\}_{k=1}^K$ in the following problem:

$$\begin{aligned}\max_{\|\mathbf{Y}\|_F\leq\lambda}\max_{\|\mathbf{Z}_k\|_2\leq\gamma}\sum_{l,k}\Bigg[&\left(-\frac{1}{2}(y_{lk}+z_{lk})^2+(y_{lk}+z_{lk})w_{lk}\right)\mathbb{I}(w_{lk}>y_{lk}+z_{lk})\\&+\frac{1}{2}w_{lk}^2\mathbb{I}(w_{lk}\leq y_{lk}+z_{lk})\Bigg],\end{aligned} \tag{4.9}$$

where $\mathbb{I}(\cdot)$ is the indicator function.

Let $s_{lk}=y_{lk}+z_{lk}$ and define

$$\phi(s_{lk})=(-\frac{1}{2}s_{lk}^2+s_{lk}w_{lk})\mathbb{I}(w_{ik}>s_{lk})+\frac{1}{2}w_{lk}^2\mathbb{I}(w_{lk}\leq s_{lk}) \tag{4.10}$$

To maximize (4.9) over $\mathbf{Y},\mathbf{Z}_k$, we discuss it case by case:

1. When $w_{lk}\leq 0$, the solution $s_{lk}=0$ has already achieved the maximum value of $\phi(s_{lk})$. Therefore, we could simply set $y_{lk}=z_{lk}=0$ and obtain $x_{lk}=0$ according to (4.8).
2. When $w_{lk}>0$, we define $\mathcal{P}=\{\{l,k\}:w_{lk}>0\}$ to be the set of indices such that $w_{lk}>0$. Let $\text{vec}(\mathbf{W}(\mathcal{P}))$ be the vectorization of w_{lk} with $(l,k)\in\mathcal{P}$.

a. If $\|\text{vec}(\mathbf{W}(\mathcal{P}))\|_2 \leq \lambda$, we could set $\mathbf{Y}(\mathcal{P}) = \mathbf{W}(\mathcal{P})$ and all other elements in $\mathbf{Y}$ to zero so that $\|\mathbf{Y}\|_F \leq \lambda$; and set $\mathbf{Z} = 0$. Then, Eq. (4.7), i.e., $\sum_{(l,k)\in\mathcal{P}} \phi(s_{lk})$, achieves its maximum. According to (4.8), we have $\mathbf{X}(\mathcal{P}) = 0$.

b. If $\|\text{vec}(\mathbf{W}(\mathcal{P}))\|_2 > \lambda$, for each column of $\mathbf{W}_k$, let $\text{vec}(\mathbf{W}_k(\mathcal{P}))$ be the vectorization of elements w_{lk} with $(l,k) \in \mathcal{P}$ for a fixed k.

i. If $\|\text{vec}(\mathbf{W}_k(\mathcal{P}))\|_2 \leq \gamma$, we could set $\mathbf{Z}_k(\mathcal{P}) = \mathbf{W}_k(\mathcal{P})$ and other elements in $\mathbf{Z}_k$ to zero so that $\|\text{vec}(\mathbf{Z}_k(\mathcal{P}))\|_2 \leq \gamma$; and set $\mathbf{Y}_k(\mathcal{P}) = 0$. Then, $\sum_{l:(l,k)\in\mathcal{P}} \phi(s_{lk})$ achieves the maximum for a fixed k. According to (4.8), we have $\mathbf{X}_k(\mathcal{P}) = 0$.

ii. If $\|\text{vec}(\mathbf{W}_k(\mathcal{P}))\|_2 > \gamma$, we can imply that both $\mathbf{Y}_k(\mathcal{P})$ and $\mathbf{Z}_k(\mathcal{P})$ as the same direction as $\mathbf{W}_k(\mathcal{P})$. According to the constraint $\|\mathbf{Z}_k\|_2 \leq \gamma$:

$$\mathbf{Z}_k(\mathcal{P}) = \frac{\gamma}{\|\text{vec}\left(\mathbf{W}_k(\mathcal{P})\right)\|_2}\mathbf{W}_k(\mathcal{P}). \tag{4.11}$$

Now define $\overline{w}_{lk} = w_{lk} - \frac{\gamma}{\|\text{vec}(\mathbf{W}_k(\mathcal{P})\|_2)} w_{lk}$ and $\overline{\mathbf{W}}$ to be the matrix of $\overline{w}_{lk}$. Let $\Theta = \{k : \|\text{vec}\left(\mathbf{W}_k(\mathcal{P})\right)\|_2 > \gamma\}$, $\Omega = \{(l,k) : (l,k) \in \mathcal{P} \wedge k \in \Theta\}$. By plugging the solution of $\mathbf{Z}_k(\mathcal{P})$ into (4.9), Eq. (4.9) can be written as the following optimization problem:

$$\max_{\|\mathbf{Y}\|_F \leq \lambda, \mathbf{Y}(\Omega^C)=0} \sum_{(l,k)\in\Omega} \left[-\frac{1}{2}(y_{lk} - \overline{w}_{lk})^2 \mathbb{I}(\overline{w}_{lk} > y_{lk}) + \frac{1}{2} w_{lk}^2 \mathbb{I}(\overline{w}_{lk} \leq y_{lk}) \right] \tag{4.12}$$

Since $\|\text{vec}\left(\mathbf{W}_k(\Omega)\right)\|_2 > \gamma$, we have $\overline{w}_{lk} > 0$ for all $(l,k) \in \Omega$. Now

A. $\|\text{vec}\left(\overline{\mathbf{W}}(\Omega)\right)\|_2 \leq \lambda$, we set $y_{lk} = \bar{w}_{lk}$ for $(l,k) \in \Omega$ and $y_{lk} = 0$ for $k \in \Theta$ but $(l,k) \notin \Omega$. Therefore, we have $x_{lk} = 0$ for $(l,k) \in \Omega$. Combining with the previous discussions, we can further infer that the entire $\mathbf{X} = 0$.

B. $\|\text{vec}\left(\overline{\mathbf{W}}(\Omega)\right)\|_2 > \lambda$, we have

$$\mathbf{Y}(\Omega) = \frac{\lambda}{\|\text{vec}(\overline{\mathbf{W}}(\Omega))\|_2}\overline{\mathbf{W}}(\Omega) \tag{4.13}$$

Therefore, according to (4.8), we have $\mathbf{X}(\Omega^C) = 0$ and for each $(l,k) \in \Omega$:

$$\begin{aligned} x_{lk} &= w_{lk} - z_{lk} - y_{lk} \\ &= w_{lk} - \frac{\gamma}{\|\text{vec}\left(\mathbf{W}_k(\mathcal{P})\right)\|_2} w_{lk} - \frac{\lambda}{\|\text{vec}\left(\overline{\mathbf{W}}(\Omega)\right)\|_2}\overline{w}_{lk} \\ &= w_{lk}\left(1 - \frac{\gamma}{\|\text{vec}\left(\mathbf{W}_k(\mathcal{P})\right)\|_2}\right)\left(1 - \frac{\lambda}{\|\text{vec}\left(\overline{\mathbf{W}}(\Omega)\right)\|_2}\right) \end{aligned} \tag{4.14}$$

By combining the case 1, 2(a), 2(b)i, 2(b)iiA and 2(b)iiB, we obtain the analytical solution of $\mathbf{X}$. In other words, we obtain the closed-form solution for the proximal mapping in (4.4) stated in Theorem 3, which completes our proof. □

4.4 STOCHASTIC OPTIMIZATION TECHNIQUE

Although the optimization methods in Algorithm 1 and Algorithm 2 are very efficient and much more scalable than the traditional Newton methods or interior-point methods, for web-scale datasets, these methods cannot be easily scaled up. For example, in SLIM (3.4), at each iteration, the gradient of the loss function (3.6) can be written as the summation of K gradients, where K is the number of topics:

$$\nabla f(\mathbf{I}) = \sum_{k=1}^{K} \mathbf{M}_k^T (\mathbf{M}_k \cdot \mathbf{I} - \mathbf{V}_k) \tag{4.15}$$

Therefore, for each iteration, the total computational complexity is $O(K \times N \times T \times L)$. Furthermore, for each iteration, it requires to store the entire data matrices $\mathbf{M}_{uk}$ for all $1 \leq u \leq N$ and $1 \leq k \leq K$, which could also be very expensive in terms the storage. An effective way of improving the effectiveness scalability for solving SLIM and MSLIM is to use the stochastic optimization methods. In this section, we apply the popular stochastic optimization method, regularized dual averaging, to solve both SLIM and MSLIM in a unified framework.

Let $\mathbf{D}_k = \{\mathbf{V}_k, \mathbf{M}_k\}$ be the observed data for the k-th topic and we use $k \sim [K]$ to denote that the random quantity k, which is uniformly sampling from $\{1, \ldots, K\}$. The corresponding optimization formulation for SLIM and MSLIM can be written as:

$$\begin{aligned} \text{minimize} \quad & \Psi(\mathbf{I}) \doteq \underbrace{\mathbb{E}_{k \sim [K]}(F(\mathbf{I}, \mathbf{D}_k))}_{f(\mathbf{I})} + \text{Pen}(\mathbf{I}) \\ \text{subject to} \quad & \mathbf{I} \geq 0., \end{aligned} \tag{4.17}$$

Algorithm 3 Regularized Dual Averaging Method for Solving SLIM & MSLIM

Input Parameters: Starting point $\mathbf{I}_0$, stepsize $\tau > 0$.

Initialization: Set $\mathbf{I}_0 = \mathbf{0}$ and the averaged stochastic gradient $\overline{G}_0 = \mathbf{0}$

Iterate for $t = 1, 2, \ldots,$ until convergence

1. Sample a topic k uniformly from $\{1, \ldots, K\}$ and compute the stochastic gradient $G_t(\mathbf{I}, \mathbf{D}_k)$ according (4.18).
2. Update the average stochastic gradient:

$$\overline{G}_t = \frac{t-1}{t}\overline{G}_{t-1} + \frac{1}{t}G_t(\mathbf{I}, \mathbf{D}_k)$$

3. Update the parameter $\mathbf{I}$:

$$\mathbf{I}^{t+1} = \arg\min_{\mathbf{I}} \left\{ \langle \overline{G}_t, \mathbf{I} \rangle + \text{Pen}(\mathbf{I}) + \frac{\tau}{2\sqrt{t}} \|\mathbf{I}\|_2^2 \right\} \tag{4.16}$$

Output: $\mathbf{I}^t$

where $F(\mathbf{I}, \mathbf{D}_k)$ and $\text{Pen}(\mathbf{I})$ are defined as follows in SLIM and MSLIM, respectively:

$$\text{SLIM} \quad F(\mathbf{I}, \mathbf{D}_k) = \tfrac{1}{2}\|\mathbf{V}_k - \mathbf{M}_k \cdot \mathbf{I}\|_2^2, \quad \text{Pen}(\mathbf{I}) = \lambda \sum_{u=1}^{N} \|\mathbf{I}_u\|_2$$

$$\text{MSLIM} \quad F(\mathbf{I}, \mathbf{D}_k) = \tfrac{1}{2}\|\mathbf{V}_k - \mathbf{M}_k \cdot \mathbf{I}^k\|_2^2, \quad \text{Pen}(\mathbf{I}) = \lambda \sum_{u=1}^{N} \|\mathbf{I}_u\|_F + \gamma \sum_{u=1}^{N}\sum_{k=1}^{K} \|\mathbf{I}_{uk}\|_2$$

To apply any stochastic optimization method, we first need to construct the stochastic gradient, which is an unbiased estimator of $\nabla_{\mathbf{I}} \mathbb{E}_{k\sim[K]}(F(\mathbf{I}, \mathbf{D}_k))$. In our problem, we first sample a topic k uniformly from $\{1, \ldots, K\}$ and then construct the stochastic gradient as follows:

$$G(\mathbf{I}, \mathbf{D}_k) = \nabla_I F(\mathbf{I}, \mathbf{D}_k) = \begin{cases} \mathbf{M}_k^T(\mathbf{M}_k\mathbf{I} - \mathbf{V}_k) & \text{SLIM} \\ \mathbf{M}_k^T(\mathbf{M}_k\mathbf{I}^k - \mathbf{V}_k) & \text{MSLIM} \end{cases} \tag{4.18}$$

From the computation of stochastic gradient, the advantage of using the stochastic optimization approach is obvious. Taking SLIM as an example, when using stochastic optimization approach, the per-iteration time complexity of computing stochastic gradient is only

$O(N \times T \times L)$. As compared to the deterministic optimization with the per-iteration complexity $O(K \times N \times T \times L)$, the stochastic optimization reduces the per-iteration complexity by a factor of K. The stochastic gradient is not only computationally more efficient, it can also be applied to web-scale datasets where the entire $\mathbf{M}$ cannot be stored in the memory. For each iteration, the stochastic optimization method only requires $O(N \times T \times L)$ memory to store $\mathbf{D}_k$.

Among many stochastic optimization methods [Duchi and Singer, 2009, Duchi et al., 2010, Hu et al., 2009, Langford et al., 2009, Nemirovski et al., 2009, Ghadimi and Lan, 2012, Lan and Ghadimi, 2010, Xiao, 2010, Hazan and Kale, 2011], we choose to apply the recently proposed regularized dual averaging (RDA) [Xiao, 2010] method to our problem. Instead of only utilizing a single stochastic gradient of the current iteration as many classical stochastic gradient methods, RDA updates the parameter vector $\mathbf{I}$ using the *average* of all past stochastic gradients. This averaging step not only leads to the improved empirical performance but also has the theoretical guarantee that the final solution $\mathbf{I}$ achieves the desirable sparsity pattern (i.e., manifold identification property) [Lee and Wright, 2011]. The manifold identification property is of critical importance in our application since our main goal is to detect influential users. The regularized dual averaging method for solving our problem is presented in Algorithm 3. We note that for MSLIM, we will first vectorize $\mathbf{I}$ and then apply the key step in (4.16) which can be re-written as follows,

$$\mathbf{I}^{t+1} = \arg\min_{\mathbf{I}} \left\{ \frac{1}{2} \|\mathbf{I} - \frac{\sqrt{t}}{\gamma} \overline{G}_t\|_2^2 + \frac{\sqrt{t}}{\gamma} \text{Pen}(\mathbf{I}) \right\} \tag{4.19}$$

From (4.19), it is easy to see that it is exactly the *proximal mapping* step. The closed-form solutions of (4.19) when Pen($\mathbf{I}$) in SLIM and MSLIM have been provided in Theorem 1 and 3 respectively. By Theorem 3 from [Xiao, 2010], Algorithm 3 has the convergence rate in $O\left(1/\sqrt{t}\right)$.

4.5 EXPERIMENTS

Similar to SLIM, we evaluate the performance of MSLIM on the same three social network datasets in this section. The data collection, topic (contagion) extraction procedures have been described in Section 3.5.1 and Section 3.5.2. We compare MSLIM with SLIM on the prediction task, which is presented in quantitative analysis part. In addition, we also present some qualitative analysis results on the detected contagion-sensitive influential users(nodes) for different topics.

4.5.1 Quantitative analysis

As same as SLIM explained in section 3.5.3, we evaluate the prediction performance of MSLIM according to time series prediction task using the prediction mean-squared error (MSE) as the evaluation metric.

$$\text{MSE} := \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{T}\left(\sum_{t=1}^{T}(\widehat{V}_k(t) - V_k(t))^2\right)\right), \tag{4.20}$$

where $\widehat{V}_k(t)$ is the predicted total volume for the contagion k at the time t using the data from previous time.

To apply the model, some parameters (e.g.,time-tag L, regularization parameters λ, γ) need to be determined. Similar to SLIM, we split the first 60% tweets in time as the training set and the last 40% as the validation set. We choose the best set of parameters that lead to the minimum prediction MSE on the validation set. Parameter L denotes how long it takes the influence of a user to decay to zero and we set $L = 5$ as in SLIM. In our experiment, we first use the validation set to tune the parameters γ and λ, which determine the number of selected influential users for each topic. According to [Yuan and Lin, 2006], we set $\lambda = \sqrt{K}\gamma$, and vary $\lambda \in \{1, 5, 10, 50, 100, 200, 300, 400, 500, 1000, 1500, 2000\}$ by grid search. The best $\lambda = 300, \gamma = 42.43$ for the twitter dataset. Then we combine the training and validation sets to re-train the model with the best selected parameters and estimate the influence function for each user and topic. The final MSE is reported on the entire time span. In our MSLIM model, the degree of freedom is the dimension of influential function $\mathbf{I}$, which equals to

	Testing MSE		Training MSE	
	SLIM	MSLIM	SLIM	MSLIM
Twitter	20.24	13.03	14.82	10.15
Facebook	28.42	22.54	25.05	18.28
Plurk	16.17	11.75	12.75	9.48

Table 5: Comparison between MSLIM and SLIM in terms of the prediction MSE

$N \times K \times L$ (i.e. 1.965×10^5 for twitter data, 3.05×10^5 for facebook data and 5.9×10^5 for plurk dataset).

Here, we compare the MSLIM algorithm to the previously proposed SLIM model in terms of prediction MSE over total 50 topics; and the result is presented in Table 5 for the three datasets, where testing MSE is reported on the entire time span and training MSE is reported on the training sets. Furthermore, we present the comparison of testing MSE over different number of topics $K \in \{5, 10, 20, 30, 40, 50\}$ in Twitter dataset as shown in Figure 15. The procedure is that we randomly pick 5 topics' data among 50 and run SLIM/MSLIM and consecutively randomly pick more topics to run experiment. As shown from Table 5 and Figure 15, MSLIM outperforms SLIM. Moreover, we have better predicted MSE when using more topics' data. As explained in Chapter 3, our SLIM doesn't consider the contagion-sensitive user selection. We assume that each node has the same influence across all the contagions in SLIM model. From table 5, we can clearly see that the model performs better when considering the topic-sensitive selection. For different types of contagions, the sets of most influential nodes are different. When combing Table 5 and Table 4, we can see that MSLIM also outperforms TwitterRank, which is the best algorithm for the twitter dataset in Table 4. Again, we note that TwitterRank uses the information about network, which is unavailable in some practical situations. In contrast, our MSLIM does not require the network structure and thus has wider applications.

For the running time, we show the advantage of stochastic optimization (i.e. RDA) over

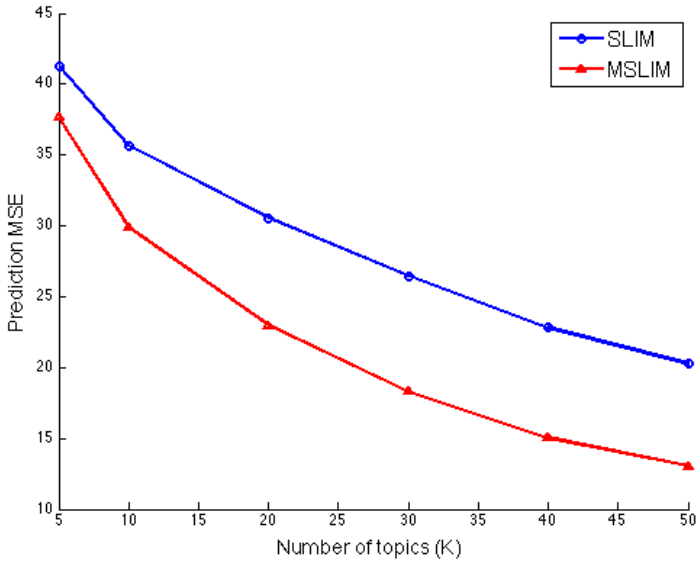


Figure 15: Comparison between SLIM and MSLIM of prediction MSE over different number of topics K

	SLIM		MSLIM	
	FISTA	RDA	FISTA	RDA
Twitter	118.12	45.37	170.62	62.08
Facebook	217.63	57.12	291.24	80.23
Plurk	423.75	130.47	650.21	225.18

Table 6: Comparison of CPU time (in seconds).

deterministic optimization (i.e. FISTA). The experiments are carried out on a standard PC with 4GB RAM in MATLAB. In particular, for deterministic optimization, we stop algorithms when the relative change in the objective is below 10^{-6}. For the stochastic optimization, since it will take a very long time to achieve the same objective value as deterministic optimization, we stop stochastic optimization when the objective value is within 1.001 times

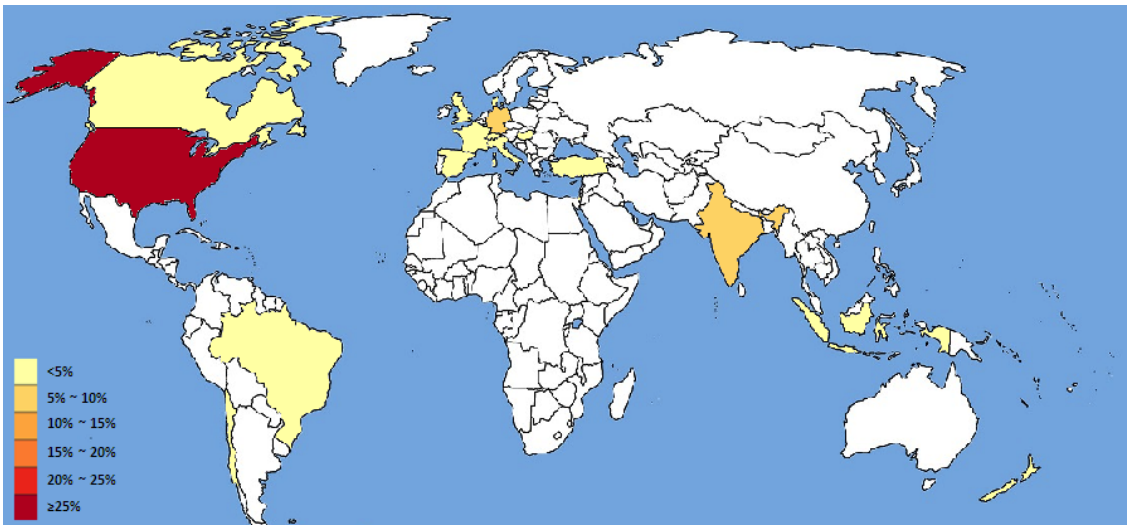


Figure 16: Colorpleth map of selected influential users for aggregation of 50 topics

the deterministic objective value. As we can see from Table 6, the stochastic optimization is much more efficient than the deterministic counterpart. Although the stochastic optimization leads to a less accurate objective function, its computation time is about one third of its deterministic counterpart. In addition, we also observe that the runtime of MSLIM is slightly longer than that of SLIM due to the complicated penalty function.

4.5.2 Qualitative analysis

In this section, we present some interesting patterns of the most influential users selected from MSLIM. For each topic k, we detect a set of the most influential users $\mathcal{U}_k$ in (4.3). The union set of the selected users (i.e., $\mathcal{U} = \{u : \|\widehat{\mathbf{I}}_u\|_F > 0\}$) contains 86 users while some users are influential on a variety of topics. We plot the geographical colorpleth map and the detailed locations of these 86 users as shown in Figure 16 and Figure 17. We omit four of them since they do not provide their locations in their profiles. We can observe that most of the influential users (42 out of 82) are located in North America, followed by the Europe (22 out of 82). There is no influential user selected in Africa, Australia or Antarctica. Such

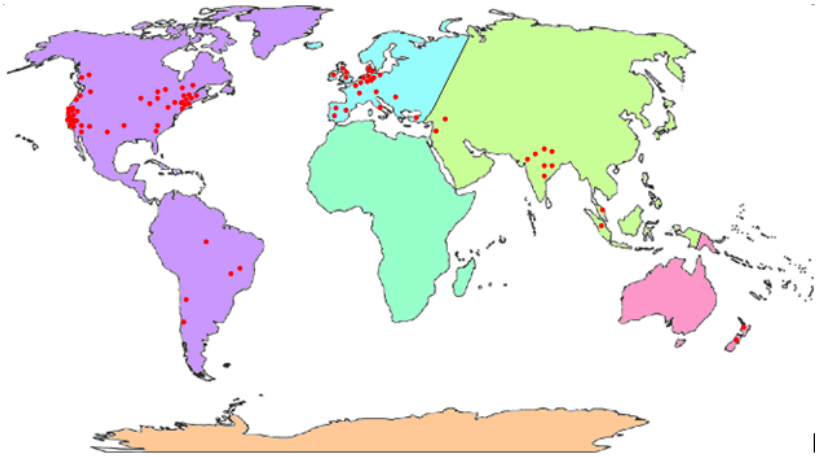
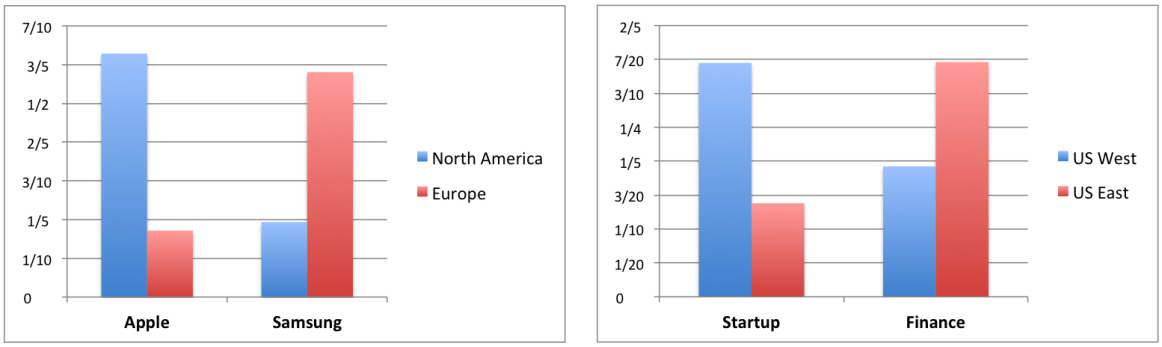


Figure 17: Detailed locations of selected influential users for aggregation of 50 topics



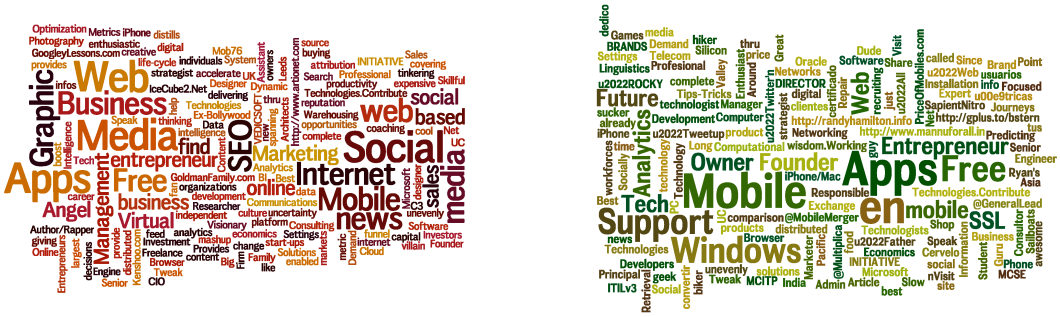
(a) Location distribution of influential users for topics "Apple" and "Samsung"

(b) Location distribution of influential users for topics "Startup" and "Finance"

Figure 18: Comparison of Location distributions of topic-sensitive influential users.

a result is quite expectable because most of 1000 users are closely related to internet media and high technology and North America and Europe are centers for high-technology.

By analyzing the contagion-sensitive influential users selected for each topic, we show some interesting findings for the four topics in Table 2. We omit those selected users without location information. As shown in figure 18(a), the topic "Apple" has more influential users in North America while "Samsung" is more popular in Europe. In figure 18(b), we compare location distribution for the other two topics. For the topic "Startup", nearly 30%



(a) Short biography of influential users for topic "Apple".

(b) Short biography of influential users for topic "Samsung".

Figure 19: Comparison of biography of topic-sensitive influential users presented by Wordle.

topic 18	apps	try	php	fun	developer	rss	programming	script	integration	google
topic 34	wsj	innovative	firms	technology	discusses	bloomberg	businessweek	aid	economist	bubble
topic 44	global	reuters	announces	consumer	aim	launches	firms	inc	users	aid

Table 7: Top 10 words for three topics learned by LDA.

of influential users are located in the west coast of USA. This is because the Silicon Valley in the bay area has many startups in the high-technology sector. In contrast, most influential users for the topic "Finance" are located in the east coast of the US, which is the world center of finance.

We also analyze the contagion-sensitive influential users' short biography for different topics. It is interesting to use wordle to present the description of users. We find that some influential users for "Apple" are marketing managers, graphic designers and news sources while all of influential users for "Samsung" are related to high-technology, as shown in Figure 19(a) and 19(b). Moveover, influential users for "Startup" include more IT related users but less news sources and financial media related users.

Since we utilize the correlations among topics to select the topic-sensitive influential users in MSLIM, thus for two similar topics, the sets of influential nodes should be close. We can use the Jaccard index (Jaccard coefficient) to measure similarity between two sample sets, which is defined as the size of the intersection divided by the size of the union of the sample sets: $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. For example, we compute the Jaccard coefficients for three sample sets $\mathcal{U}_{18}$,$\mathcal{U}_{34}$, $\mathcal{U}_{44}$, which correspond to the influential user sets for topic 18, 34, 44 respectively. Table 7 lists the top 10 words from these three topics learned by LDA. We get the result that $J(\mathcal{U}_{34}, \mathcal{U}_{44}) = 0.42$, $J(\mathcal{U}_{34}, \mathcal{U}_{18}) = 0.23$, $J(\mathcal{U}_{44}, \mathcal{U}_{18}) = 0.26$. We find that Jaccard index of $\mathcal{U}_{34}$, $\mathcal{U}_{44}$ is larger than the other two, since topic 34 and 44 are similar.

5.0 COMPONENT-BASED SLOW INTELLIGENCE SYSTEM

In this Chapter, we propose a generic intelligence system, called component-based slow intelligence system. It has important application in influential node detection in information diffusion network, which will be demonstrated in the next Chapter. In this Chapter, we first present the principle, architecture and example of applications of this component-based slow intelligence system.

5.1 INTRODUCTION AND MOTIVATION

People always strive to develop an intelligence system which is able to improve performance over time. In practice, one might face the difficulty of choosing the appropriate method among the different possible solutions for a given problem. We hope to design a system which can continuously learn, search for the suitable solutions and propagate and improve its performance over time.

A blogger who calls himself "Master Luke" proposed a new term "Slow Intelligence", which is the ability to step back and calculate possibilities without any sort of outside constraints (like time). "Slow intelligence" seems indispensable to our evolutionary history. He mentioned about an interesting story : "I just recently finished reading Robert Pirsig's Zen and the Art of Motorcycle Maintenance. There's a point toward the end of the book where Pirsig relates his experiences of a few philosophy graduate seminars at the University of Chicago. At one point, the professor asks him his opinion on, I believe, a Socratic dialogue. Pirsig is unable to respond, not because he's dumb, but because, after the question is asked, he begins to replay in his head possible answers and their logical consequences and

contradictions–over and over, until everyone has left the classroom except Pirsig." According to Master Luke, Slow Intelligence began to develop more and more as our cities and societies became more and more distant from Nature. In other words Slow Intelligence is distinctively human, although it must be recognized that in our evolution history.

Recently, Chang (2010) developed the basic slow intelligence principles. He defined the fundamental framework and characteristics of slow intelligence system. In section 5.2, we develop a theoretical model of slow intelligence system. We further present the architecture design of a component-based SIS system in section 5.3. An innovative software platform to create and implement the component-based SIS system is described in section 5.4. We also show an example of applications of SlS system in section 5.5.

5.2 A THEORETICAL MODEL OF SLOW INTELLIGENCE SYSTEM

In this section, a theoretical model of slow intelligence system is provided using an abstract machine. Through the use of abstract machine, it is possible to compute and simulate the problem-solving process for a specific problem.

An abstract machine $\mathrm{M_{sis}}$ for slow intelligence system is defined as:

$$\mathrm{M_{sis}} = \{\mathcal{P}, \mathcal{S}, P_0, \mathrm{cycle_1}, \mathrm{cycle_2}, \cdots, \mathrm{cycle_n}\} \tag{5.1}$$

where $\mathcal{P}$ is the problem space, $\mathcal{S}$ the solution space, P_0 the initial problem set and $\mathrm{cycle_1}, \mathrm{cycle_2}, \cdots, \mathrm{cycle_n}$ the computation cycles. The *problem space* $\mathcal{P}$ is a nonempty, enumerable set of *problem elements* $p_1, p_2, \cdots, p_m$. A *problem set* P_k is a finite subset of $\mathcal{P}$. At least some of the problem elements in the problem space are also the *solution elements*. Therefore, the *solution space* $\mathcal{S}$ is a nonempty subset of the problem space $\mathcal{P}$. A *solution set* S is a finite subset of $\mathcal{S}$.

Starting from an initial problem set P_0, the objective of the abstract machine $\mathrm{M_{sis}}$ for the slow intelligence system is to derive a problem set P_j that is also a solution set, i.e., it contains only solution elements, by applying one or more of the computation cycles: $\mathrm{cycle_1}, \mathrm{cycle_2}, \cdots, \mathrm{cycle_n}$.

A *computation cycle* is a sequence of slow intelligence operators to transform problem sets. The following five *slow intelligence operators* are defined for problem sets.

1. **Enumerator:** P_1 **-*enum*<** P_2 is the *enumerator* that takes each problem element of P_1 to generate a number of new elements, which are put into P_2.
2. **Eliminator:** P_1 **>*elim*-** P_2 is the *eliminator* that eliminates non-solution elements of P_1 and put the rest into P_2.
3. **Concentrator:** P_1 **>*conc*=** P_2 is the *concentrator* that selects some problem elements of P_1, which are put into P_2.
4. **Adaptor:** P_1 **+*adap*=** P_2 is the adaptor that inputs information from the environment and modifies elements of P_1 to produce P_2 according to the adaptation rule.
5. **Propagator:** P_1 **=*prop*+** P_2 is the propagator that outputs information to the environment and modifies elements of P_1 to produce P_2 according to the propagation rule.

As an example, the abstract machine M_{sis} may have the following simple computation cycle:

cycle$_1$: P_0 **-*enum*<** P_1 **>*elim*-** P_2

This simple computation cycle works as follows. First, we enumerate the potential solution elements to generate the problem set P_1 from the initial problem set P_0. Then, we eliminate those that are not solution elements to generate another problem set P_2 containing only the solution elements. The abstract machine M_{sis} may have another simple computation cycle:

cycle$_2$: P_0 **-*enum*<** P_3 **-*enum*<** P_4 **-*enum*<** P_5 **>*elim*-** P_6

In this computation cycle, three enumeration operators are used in sequence to enumerate potential solution elements, thus generating a larger problem set P_5. Then, we apply the elimination operator to derive the final solution set P_6.

A slow intelligence system can have multiple computation cycles. A control mechanism with evaluation rules in the system decides the transfer criteria and the switching mode among the cycles. To provide an appropriate control mechanism, we can add guards to the two computation cycles:

cycle$_1$: [guard$_{1,2}$] P_0 **-*enum*<** P_1 **>*elim*-** P_2

cycle$_2$: [guard$_{2,1}$] P_0 ***-enum<*** P_3 ***-enum<*** P_4 ***-enum<*** P_5 ***>elim-*** P_6

A *guard* for cycle$_i$ is a predicate of the form guard$_{i,j}$ defined on problem sets and evaluated whenever a problem set is generated. If the predicate is evaluated to be **true** then M$_{sis}$ transfers to cycle$_j$. If the predicate is evaluated to be **false** then M$_{sis}$ remains in cycle$_i$. If this is the last problem set then the machine halts.

For example, guard$_{1,2}$ may specify if the problem set P_2 is empty then M$_{sis}$ transfers to cycle$_2$ (a slow computation cycle). In other words, if cycle$_1$ produce no solutions then M$_{sis}$ should switch to cycle$_2$ even though cycle$_2$ is computationally more expensive. If P_2 is non-empty then M$_{sis}$ halts, i.e., a solution is found. Conversely, guard$_{2,1}$ may specify if the problem set P_6 is non-empty then M$_{sis}$ transfers to cycle$_1$ (a fast computation cycle). In other words, once cycle$_2$ produces a solution then M$_{sis}$ should switch back to cycle$_1$. If P_6 is empty then M$_{sis}$ halts, i.e., no solution can be found.

5.3 DESIGN OF COMPONENT-BASED SLOW INTELLIGENCE SYSTEM

In this section, we present the architecture and operational design of slow intelligence system by using component-based software development approach.

5.3.1 System Architecture

Fig 20 illustrates the SIS system architecture comprising the key system components including Enumerator, Eliminator, Concentrator and Time Controller. In addition, a customized Tester and a Transformer are included to provide more functionalities.

The system works as follows. The Enumerator reads in the specification of functional blocks and creates multiple candidate components for each functional block. The Tester tests all functional blocks and records their performance in the DB. The Eliminator selects the best candidate component based upon its performance. The Concentrator packs the selected candidate components based on dependency specifications and generates a generic software package. When there are terminological and/or conceptual differences within the

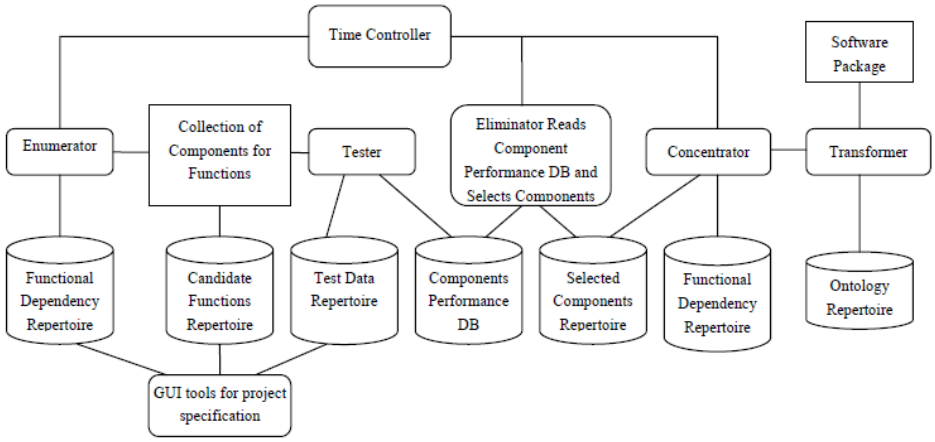


Figure 20: System architecture of Component-based Slow Intelligence System

software package, Transformer is used to transform the Concentrator-generated software package to target software package that serves specific purpose. The Time Controller is like an automatic system manager, which controls the operational sequence of the other system components.

5.3.2 Specification

Specification is the first phase of the system design. A GUI program is used to specify a certain real system. The GUI can be divided into three different parts, one for specifying functional blocks' dependencies, one for specifying pools of candidate components in each functional block, and one for specifying test data for each functional block.

All these three specifications are stored in three different repertoires. Functional blocks dependency repertoire stores the dependencies among different functional blocks. For example, the output of functional block i is the input of functional block j. Candidate functions repertoire stores all the candidate components for each functional block, and test data repertoire stores all the test data used for testing candidate components in each functional block. The detailed implementation of GUI will be presented in section 5.4.1.

5.3.3 Initialization

The next phase is the initialization of system. Enumerator reads the functional blocks' dependencies from the corresponding repertoire and creates multiple candidate components for each functional block. The candidate components are stored in candidate functions repertoire. In Fig. 21, we show the visual definition of initial run of the system. All candidate components are classified into different functional blocks. These are different algorithms or one algorithm with different parameters. The candidate components within the same functional block so called super component [Chang et al., 2011b] do the same job. We use the double-edged notation to denote the super component in Fig. 21. We will discuss how the candidate components are generated in section 5.3.5.1. Tester starts running in this phase and keeps running until the system halts. Tester tests each candidate component performance using test data stored in repertoire and records the performance in components performance DB.

5.3.4 Execution

In this phase, Time Controller manages the whole system's operation. The system goes into its main cycle of life. The system may execute multiple runs of Enumeration and Elimination. Time Controller has multiple functions: 1) it controls Enumerator to create new candidate components from candidate function repertoire; 2) it manages Eliminator to eliminate some worse candidate components based upon their performances recorded in components performance DB; 3) it notifies the Concentrator to store the best component in the selected components repertoire.

5.3.5 Operating Procedures for Slow Intelligence System Components

Here, we describe the operation procedures of the key system components in detail.

5.3.5.1 Enumerator The inputs to the Enumerator are functional blocks dependencies and candidate components. The outputs from Enumerator are multiple instances of candi-

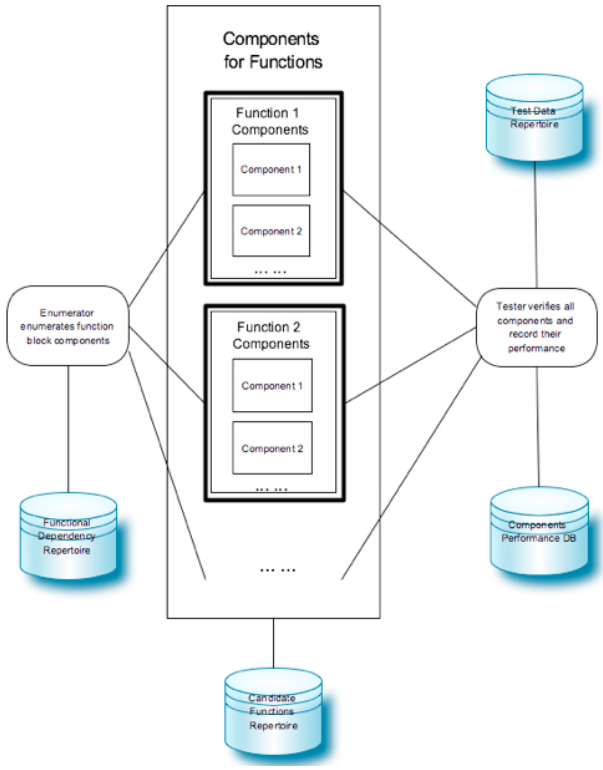


Figure 21: Initialization of Component-based Slow Intelligence System.

date components for each functional block.

At initialization phase, Enumerator firstly generates a super component for each functional block by retaining the dependencies among these blocks. These super components will be filled with instances of corresponding candidate components/algorithms in the next step. Enumerator reads in information stored in candidate components repertoire. It checks each functional block how many candidate component instances it needs. For example, it requires only one candidate component instance, it directly instantiates the default candidate component and fills in the corresponding super component. There are two ways to instantiate a candidate component. If a candidate component is generic, we may pass in different parameters to generate multiple instances of that candidate component. If a candidate component is specialized, only one determined instance is created for that candidate component.

5.3.5.2 Tester The input to Tester is the testing data stored in the test data repertoire. The output of the Tester is the performance result of each candidate component instances which will be stored in the components performance DB.

At initialization phase, Tester reads in the testing data for each functional block (super component). Then the corresponding candidate components in this functional block read these data as input and send back the result to Tester after computation. Tester compares the results with the correct or desired outputs and computes errors. Finally Tester records each corresponding candidate component's performance including running time and accuracy rate in the components performance DB. It also maintains each candidate component's overall performance on all the testing data.

5.3.5.3 Eliminator The input to Eliminator is candidate components performance record stored in components performance DB and the output of Eliminator is the appropriate candidate components and best candidate component for each functional block (super component). The output is stored in selected components repertoire.

At execution phase, Eliminator reads in all candidate components performance records of each functional block. Then it eliminates unqualified candidate components by the defined criterion, e.g. eliminating the components whose accuracy rate is below a certain threshold.

The selected qualified candidate components are stored in selected components repertoire for the usage in the next round of Enumeration and Elimination. In addition, Eliminator selects the best candidate component with the highest accuracy rate and shortest running time for each corresponding functional block. The best candidate component information is also stored in selected components repertoire.

5.3.5.4 Concentrator The input to the Concentrator is the best candidate component information in each functional block (supper component) and the functional blocks dependencies information. The output is a generic software package.

At execution phase, the Concentrator reads in the functional blocks dependencies information from the repertoire and constructs the dependency graph where the functional blocks (super components) as nodes. Then Concentrator reads the best candidate component for each functional block and replaces this block with its best candidate component. Finally the Concentrator packs all best candidate components and generates a generic software package.

5.4 IMPLEMENTATION FOR COMPONENT-BASED SLOW INTELLIGENCE SYSTEM

In this section, we describe an innovative software platform to create and implement the component-based slow intelligence system. The realization of this system contains three stages: graphical interface, project code generation and SIS system simulation, as shown in Fig. 22.

Since slow intelligence system is a general-purpose system, user can apply it to any real applications. In the Appendix, we show how to apply the component-based slow intelligence system to a simple example. Afterward, we will illustrate a complex application in Section 5.5.

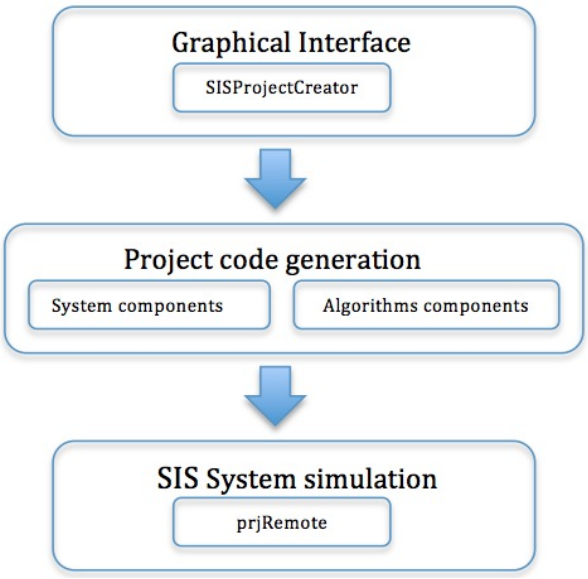


Figure 22: Three stages of SIS system implementation

5.4.1 Graphical interface

We develop a so-called SISProjectCreator which is a tool to help SIS system developers create a SIS project template. A SIS project template created by this tool basically includes template codes of SIS system components and candidate components, together with related scripts and XMLs. With SISProjectCreator, SIS developers could further write the code to implement the specific algorithms or methods without worrying the codes which handle low level system communications and executions within the SIS system.

The main purpose of the SIS system is to test multiple algorithms with multiple parameters for complex task and improve the performance over time. After some cycles of running and testing, the system could pick the algorithms with best performance based on the user-defined evaluation criteria and discard those with worse results; it could also pick the best candidate algorithm with appropriate parameters. Algorithms are classified into different functional blocks called cycles. Algorithms within the same cycle do the same job by the identical testing data. One algorithm could be instantiated multiple instances with

different parameters. The goal of SIS system is to pick best algorithms with proper parameters for each cycle and improve the overall performance over multiple cycles. A SIS system typically includes the following five system components: Enumerator, DataSender, Verifier, Eliminator and TimeController. Enumerator enumerates candidate components/algorithms with different parameters on cycle basis. Components could be instantiated sequentially or parallelled. DataSender sends testing data to algorithm instances created by Enumerator. Verifier evaluates algorithm instance results and sends back performance record to Eliminator. Thus Data Sender and Verifier together realize the functionality of Tester as described in section 5.3.5.2. Eliminator eliminates algorithm instances based on their performances on cycle basis. TimeController controls the executions order of different cycles.

Message communicated within SIS system is such a process: when five system components execute, TimeController firstly sends a message to Enumerator notifying which cycle(s) to execute. Enumerator instantiates algorithms and sends messages to Eliminator about algorithm instances information. When Enumerator completes, it sends another message to notify Eliminator that enumeration phase completes. Enumerator then sends a message to DataSender to query data message. DataSender sends data message to algorithm instances for processing these testing data and receives the computation result message. Verifier processes this result message to compute the accuracy rate and sends it to Eliminator to evaluate. Then Eliminator sends message to DataSender to query the new data message if all the algorithm instances send the results. DataSender can continue to send testing data or to notify Eliminator that there is no new testing data need to be processed. Then Eliminator discards the worse algorithms and sends back the result to Enumerator. If Enumerator has more algorithms to instantiate, it continues to next round. If Enumerator has no more algorithms to instantiate, it works as Concentrator to select one algorithm instance for each cycle and notify TimeController the cycle is completed. TimeController checks whether all the cycles are complete or not. Different SIS system for different application might go through a slightly different process; SISProjectCreator could generate a quite sophisticated specific SIS system template.

The GUI includes three sub interfaces: cycle specification, Time Controller specification, and other system behavior specification.

- *Cycle specification*

  User can add, delete or update cycle information. When a cycle is selected, user can add, delete or update candidate algorithm information. When a certain algorithm is selected, the parameters list is shown. SIS developers could directly edit on the parameters list table. For more complicated applications, user can define multiple cycles where each cycle conducts a certain task. We will show such kind of system in Section 5.5. Moreover, in each cycle, all the different algorithms utilize the same test data and expected result. User can click on "Test Data" tab to specify the testing data sources.

- *TimeController specification*

  The second sub interface for specifying the SIS system is the time controller specification where user can define the execution orders among different cycles. Each switching rule represents a order between two different cycles. User can add multiple switching rules in the SIS system if there are multiple cycles. TimeController uses these rules to determine running sequence. The SIS system finishes when it executes all the switching rules.

- *Other system specification*

  In this interface, all the other SIS system parameters can be defined. SIS developers could change the values to manage SIS system's behavior.

After defining the specific SIS system for the certain problem or application, a SIS project template folder is created by SISProjectCreator. It basically includes runnable java codes for SIS system components (e.g. Eliminator, Enumerator, DataSender, etc.) and template java codes for algorithm components and related running scripts. We use XML formatted messages in SIS system to communicate among components. Each message has a unique MsgID and is constituted by Key/Value pairs. All the communication information (e.g. data, result, etc.) are encoded as this format message. The created SIS project template also contains initialization XMLs and basic pre-defined system XMLs. With SISProjectCreator, SIS developers could concentrate on the code editing for implementing the candidate algorithm without writing any system components codes which handle low level message communications within the SIS system.

5.4.2 Project code generation

The next step is to generate the codes to implement the specific SIS system. Since SISProjectCreator has automatically generated the runnable codes for system components according to the specification in Section 5.4.1, the user only needs to edit a portion of codes for candidate components to implement the real algorithms.

Component-based SIS provides a mechanism to enumerate algorithms for a problem and selects the most appropriate algorithm based on evaluation criteria which is defined by user and should be different for the different applications. In a specific SIS system, user needs to implement the evaluation method in Verifier to define the particular metrics. In the Appendix, we show the edited codes of candidate components and verifier for a simple example.

5.4.3 SIS system simulation

The final step is the system simulation. After clicking all the .bat files in "script" folder, the SIS system will automatically execute the candidate algorithms on testing data and evaluate their performance by verifier and further to select the best algorithm. A widget *prjRemote* is developed to provide administrative view of all the messages routing through the SIS system. It can display the results of each algorithm and provide a way for user to send testing messages to SIS system, e.g. initialization message, testing data, etc. After all the algorithms complete, SIS automatically records their performance and selects the best one based on evaluation criteria defined by user. The output result is a plain text file in "Result" folder. In the Appendix, we show the system simulation results for a simple example.

In summary, the innovative software platform can help user to create and implement the component-based slow intelligence system for the specific problem. In the next section, we will show a more complex application using this component-based SIS system.

5.5 APPLICATION OF COMPONENT-BASED SLOW INTELLIGENCE SYSTEM

In this section, we demonstrate how to apply this Component-based SIS software platform to a real-world application: face recognition. In order to improve the accuracy and adaptation ability of different face recognition methods in accordance with surroundings, we develop a system with different combinations of various parameterized algorithms by component-based SIS platform. An experiment is conducted on four face databases. We split each face database to several parts, depending on some characteristics of the facial images (light variation, expression variation, angle variation, and normal) and train different face recognition algorithms in each part. The overall recognition performance is improved, and the experimental results show that our method is superior to individual image analysis method.

5.5.1 Design and Implementation of Face Recognition System

Face recognition is a hot research topic in pattern recognition area. It analyzes human face images and extracts the important features to identify status. There are a wide range of applications of face recognition, e.g. identity verification, criminal identification, scene monitoring, human-computer interaction, visual communication, etc. There are a lot of state-of-the-art face recognition algorithms available. Most of them perform image matching as a two-step process of subspace projection followed by classification in the space of compressed images. In a simple scenario, face matching may be implemented as subspace projection followed by a SVM classifier. The first key step is how to extract important feature vector exactly, which further effects the classification performance. It, therefore, is necessary for us to closely look at the feature extractor/subspace projection algorithms. One subspace projection technique performs totally differently on the different datasets and face image variation. The end user does not know which method should be the best to ensure the super recognition performance according to the different image variations like illumination, facial expressions and gestures.

We apply the component-based slow intelligence system to the face recognition task and

propose a new face recognition system, which utilizes three popular subspace projection algorithms as the candidate methods. Specifically, we use principle component analysis (PCA), independent component analysis (ICA) [Draper et al., 2003] and wavelet transformation PCA (WTPCA) [Zhang and Liu, 2009]. WTPCA is the method using the wavelet transformation to extract the low frequency coefficient, then applying PCA on the low frequency coefficient. For classification step, we apply identical method: Radial Basis Function SVMs. We use the SISProjectCreator in Section 5.4.1 to specify and create our face recognition template project. Firstly, the Enumerator invokes one functional block (super component) containing these three candidate algorithms (PCA, ICA, WTPCA). All the candidate components are the specified algorithms or the generic algorithm with different parameters (e.g. the reduced dimensionality). Each component is doing the same job, that is, subspace projection and RBF-SVMs. Then the DataSender sends the testing data to each candidate component. Time Controller restarts the face recognition cycle with a different super component. The Eliminator eliminates the inferior parameterized algorithms, and finally selects the best parameterized algorithm for a given dataset.Testing datasets defined by SISProjectCreator are in the form of images and are stored under a specific folder. All candidate components use the same testing data in each cycle. In our face recognition system, we define two Cycles that Cycle 1 precedes to Cycle 2.

5.5.2 Experiments

In this section, we describe the experimental steps by the proposed SIS-based face recognition system on the four different face datasets and analyze the recognition results.

5.5.2.1 Experimental Data Description Four face databases of FERET, Yale, ORL and Grimace are used in this experiment. The sample images of each database are shown in Figure 23-26 respectively. In FERET face database [Phillips et al., 2000], the face images were gathered independently from the FERET program developers. There were some minor variations in images collected on different dates. FERET database contains 1564 sets of images for a total of 14,126 images, in particular 1199 individuals and 365 duplicate sets of

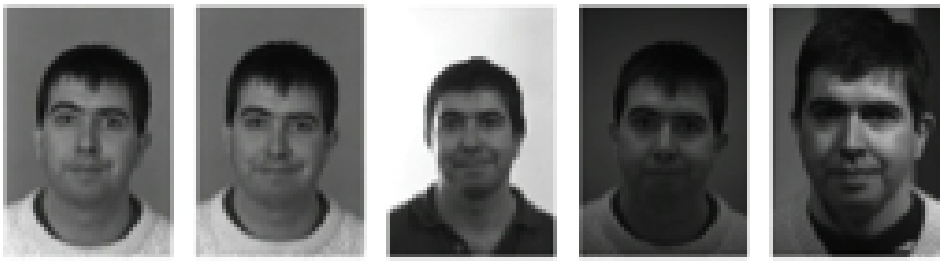


Figure 23: FERET face database sample images.

Figure 24: Yale face database sample images.

Figure 25: ORL face database sample images.

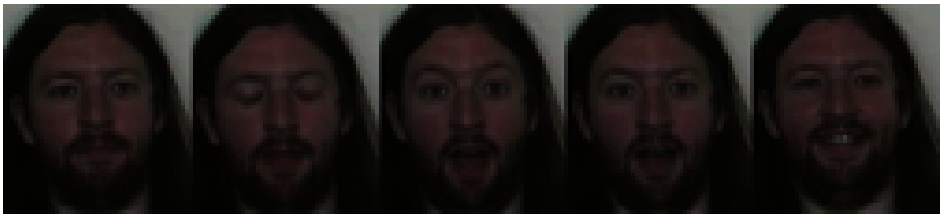


Figure 26: Grimace face database sample images.

images. For some individuals, they were photographed multiple times over two years. This time period was important because it enabled researchers to study the changes of a subject's appearance over years. We utilize the subset of this database which contains 82 individuals.

The second database is the Yale face database [Belhumeur et al., 1997]. It contains 165 grayscaling images in GIF format of 15 individuals. There are 11 images per subject, one per different facial expression or configuration: center-light, with glasses, happy, left-light, without glasses, normal, right-light, sad, sleepy, surprised, and wink.

ORL face database [Samaria and Harter, 1994] contains a set of face images taken between April 1992 and April 1994 at AT&T Laboratories Cambridge. There are ten different images for each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

In Grimace face database [1], a sequence of 20 images per subject was taken, using a fixed camera. The subject moves his/her head and makes grimaces from slight level to severe level. There is about 0.5 seconds between the successive frames in the sequence.

5.5.2.2 Experimental Results We have three subspace projection algorithms for face recognition named PCA, ICA and WTPCA. The users would be confused as they do not

[1] http://cswww.essex.ac.uk/mv/allfaces/grimace.html

Original Face Database	Light Variation	Expression Variation	Angle Variation	Normal
FERET	40	15	10	17
Yale	3	7	3	2
ORL	8	6	23	3
Grimace	3	10	3	2

Table 8: Subjects number in reorganized face databases

know which algorithm is the best or most appropriate in recognizing a given subject. Users may reorganize those face databases according to their experiences. As we know, the image variation is very important to any face recognition method. Users can reorganize those four face databases to another four face databases according to the variation degree of light, facial expression and face angle. The default database is normal if there is no obvious variation of face images. The subjects number in reorganized face databases is shown in Table 8.

In cycle one, the SIS system runs all three algorithms in super component for those four reorganized face databases to find out which parameterized algorithm is most appropriate to face image variations. For each subject, the first half images are used as training samples and the remains for testing. A feature matrix is obtained by each face recognition algorithm for each subject and is further used to verify the testing samples. After running all three algorithms by trying 20 appropriate parameter values in our experiment, the highest testing accuracy rate of each parameterized algorithm for all reorganized face databases are reported in Figure 27. The algorithm with highest accuracy rate in each reorganized face database is chosen to match the particular variation of image. For example, if ICA with reduced dimensionality 100 achieve the best accuracy rate for light variation reorganized dataset, it is chosen as the most appropriate algorithm for processing face images of light variation. The most suitable algorithm for the variation of light, expression and angle is ICA, PCA and PCA respectively. WTPCA is the best algorithm for the Normal reorganized database.

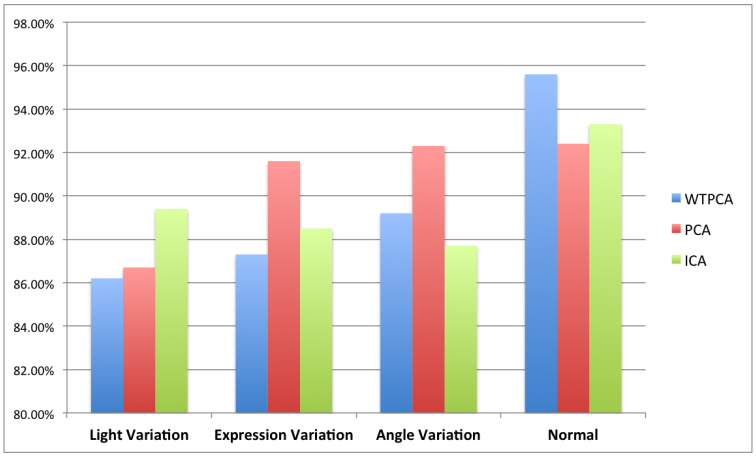


Figure 27: The accuracy rate of algorithms for recognized face databases.

	Light Variation	Expression Variation	Angle Variation	Normal
Best Algorithm	ICA	PCA	PCA	WTPCA

Table 9: The match of algorithms with images characteristics in reorganized face databases

The match of appropriate algorithms with image characteristics is shown in Table 9 and users are supposed to memorize them. But users don't need to remember the algorithm's best parameter sets which can be automatically retrieved in SIS system. In fact, the more reorganization ways for original face database the users try, the more proficient they will be. It is helpful for the users as they learn which method (including its parameter value) is most suitable for specified variation of images.

In cycle two, we compare the accuracy rate of the SIS synthesized method with that of the three individual face recognition methods on original four face databases. For each image in original database, SIS synthesized algorithm is to choose the best algorithm learned from cycle one according to the image variation. As we can see from Figure 28, the SIS synthesized method greatly outperforms any individual face recognition algorithm. Grimace achieves better recognition rate as compared to ORL, Yale and FERET. It is because the

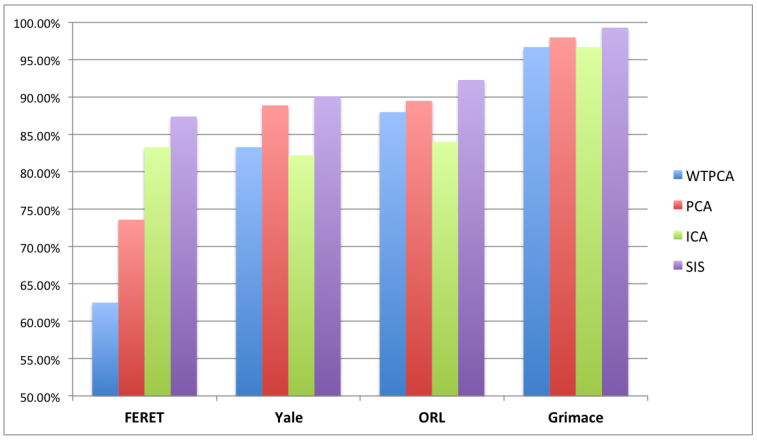


Figure 28: The accuracy rate of algorithms for original face databases.

face variation of each group in Grimace is minor. However, it is hard to say which algorithm is best for all databases. The SIS synthesized algorithm utilizing all these three methods in SIS system is superior to any individual one.

In conclusion, we design and implement a new face recognition system using the component-based SIS. It iteratively selects the best individual algorithm and propagates the learned knowledge over iterations. It leads a superior accuracy to individual face recognition method. We test our system on four face databases. The experimental results show that our system is helpful to improve the recognition accuracy in face recognition task.

6.0 SIS-BASED SPARSE LINEAR INFLUENCE MODEL

In this Chapter, we utilize the idea of the proposed component-based slow intelligent system to develop a new iterative algorithm for modeling diffusion networks and detecting influential nodes. To achieve this goal, we first develop a generic algorithm for high-dimensional sparse learning using slow intelligence principle and demonstrate its application to tumor classification. Based on that, we further develop new algorithms for influential node detection in both single-task and multi-task settings (SIS-SLIM and SIS-MSLIM). As compared to our original SLIM and MSLIM models, SIS-based approaches are much more scalable and flexible and can efficiently learn from social networks with tens of thousands of users.

6.1 INTRODUCTION

In the previous chapter, we demonstrated that our proposed component-based slow intelligence system is a very powerful learning framework, which can continuously learn, search for solutions evolutionarily and propagate its information with environment. Thus, in this chapter, we adopt the idea of SIS in our influential node detection problem in web-scale social networks. In Section 6.2, we start from a simpler setting on high-dimensional sparse learning and describe our work on how to adapt SIS for the corresponding tumor classification task. In Section 6.3, we further develop the SIS-based approaches for influential node selection based on our previous proposed sparse linear influence models (SLIM & MSLIM). We conduct investigation on large scale twitter datasets with tens of thousands of nodes/users. The experimental results show that our SIS-based approaches are much more scalable and flexible compared to the original SLIM and MSLIM.

6.2 METHODOLOGY: HIGH-DIMENSIONAL SPARSE LEARNING VIA A SLOW INTELLIGENCE APPROACH

Sparse learning in high dimensional space is the hot topic in contemporary machine learning area. In the past decade, a lot of effort has been devoted in developing various feature selection algorithms. However, each feature selection method has its own advantage for different datasets or applications. Therefore, one might face the difficulty of choosing the most suitable feature selection method in practice. In this section, we adopt slow intelligence idea in the application of high dimensional feature selection and propose a new framework which could dynamically choose feature selection algorithm from a pool of methods according to the status quo. We apply our method to cancer datasets with the expression levels of thousands of genes as features. The experimental result shows that our method is superior to individual state-of-the-art feature selection methods.

6.2.1 Motivation

High-dimensional sparse learning has many important applications in supervised learning. For example, in gene selection problems, we have gene expression levels of tens of thousands of genes as features for a number of patients as samples. A typical classification task is to separate cancer patients from the healthy ones or to distinguish different types of patients for helping regimens and treatment. There are two important issues of this application that motivate the efficient and scalable feature selection algorithm. On one hand, we may only have fewer than 100 patients (samples), it is important to eliminate those irrelevant genes (features) to obtain a robust classifier. On the other hand, from biological point of view, one wants to know which genes are the most critical factors to the cancer.

In the past decade, a lot of feature selection algorithms have been proposed. These feature selection algorithms can be roughly classified into two categories: convex regularization approach and stepwise greedy pursuit approach. The first approach regularizes the generalized linear models by adding a sparsity-inducing penalty function. The most representative algorithms include ℓ_1-norm regularized generalized linear model, e.g. lasso for

linear model [Tibshirani, 1996], elastic-net regularized generalized linear model [Mol et al., 2009], etc, which have been reviewed in Section 2.4. The second approach, the stepwise greedy pursuit approach, iteratively selects the current optimal feature according to some criteria, leading to the method as forward stepwise regression, or orthogonal matching pursuit. However, for ultra-high dimensional problem with tens of thousands features, the first approach cannot be easily applied since it leads to very slow optimization procedure. In contrast to the convex regularization method, the forward greedy pursuit approach does not suffer from such problems. Instead of trying to formulate the whole learning task to be a global convex optimization, the forward stepwise regression adopts iterative algorithm with a local view: during each iteration, it selects the best feature given a small set of selected features. Therefore, in the whole process, only a small number of variables are actually involved in the model fitting so that the whole inference only involves low dimension models. However, the main drawback of forward stepwise regression is that once an irrelevant feature is selected, it can never be removed later. In practice, it is often hard to say which individual sparse learning algorithm is better than the others for high dimensional setting. One method may be superior to another for different datasets or in different applications. In fact, a clever synthesis of the features obtained from several algorithms may lead to the best result. But currently, there is no method can do that. This drawback motivates our work [Wang and Chang, 2011] by using Slow Intelligence idea that can automatically choose the right algorithm over time and the final selected features are the results taking the advantages of multiple learning algorithms. Thus we propose a new feature selection system, which uses different feature selection algorithms as the candidate methods. Our system selects the feature using different methods, searches for the best method according to the status quo and propagates the learned features over iterations. As we show, it gives the superior feature selection performance and can handle ultra-high dimensional data.

6.2.2 Sparse learning system via SIS

In our high dimensional sparse learning process, we use the ℓ_1-regularized regression, elastic-net penalty regularized regression and forward stepwise regression as the candidate algo-

rithms, which are stored in a knowledge base. While in principle, any existing feature algorithm can be put into the knowledge base. The whole process for our system contains one main SIS procedure and a sub SIS procedure. The main system contains five phases of slow intelligence system: enumeration, elimination, adaptation, propagation and concentration. The sub system contains two phases of SIS with the knowledge base: enumeration, concentration. The main system has a particular time controller. Next we illustrate the high dimensional sparse learning process step by step.

Main Enumerator: Enumerate p features $X_1, \ldots, X_p$. Among these features, some are relevant to the responses while others not.

Main Eliminator: Compute pearson correlation between each feature X_j and response Y using:

$$R_j = \frac{\sum_{i=1}^{n}(x_{ij} - \bar{x_j})(y_i - \bar{y})}{(n-1)S_{X_j}S_Y}$$

where $\bar{x_j}$ and S_{X_j} are the sample mean and correlation for X_j and $\bar{y}$ and S_Y are the sample mean and correlation for Y . We rank the absolute value of R_j from high to low and eliminate the lowest $(p - C)$ features where C is a pre-defined constant. We denote the selected top C features as $\mathcal{A}_1$.

As we can see, the **Main Eliminator** procedure enables our system to analyze the ultra-high dimensional data. For an ultra-high dimensional data where convex optimization sparse learning algorithm cannot be directly applied, we first eliminate $(p - C)$ irrelevant features and keep only a small amount of features, so that the convex optimization approach can then be applied. After that, we iterate the following process from $k = 1$ to a certain number of iterations which is defined by the time controller.

Next, we start our Sub SIS procedure that uses the sparse learning algorithms in the knowledge base as the candidate methods. It automatically chooses the best method and selects the relevant feature set $\mathcal{S}_k$ from $\mathcal{A}_k$.

Sub Enumerator: Inquire the knowledge base that stores the existing sparse learning algorithms, e.g., ℓ_1-regularized regression, elastic-net penalty regularized regression and forward stepwise regression, etc. We enumerate all these learning algorithms by applying them to our data with the feature set $\mathcal{A}_k$. Suppose for the l-th algorithm, we denote the selected

top Q feature set as $\mathcal{S}_k^l$. Note that we always have $\mathcal{S}_k^l \subset \mathcal{A}_k$.

Sub Concentrator: For each selected feature set $\mathcal{S}_k^l$, we compute the loss function that is regressed on $\mathcal{S}_k^l$ as defined equation (2.1) or (2.2) :

$$L^l = \sum_{i=1}^{n} L(y_i, \beta_0 + \sum_{j \in \mathcal{S}_k^l} \beta_j x^{ij})$$

and choose the best algorithm l^* with the minimum loss:

$$l^* = \text{argmin}_l L^l$$

Now the Sub system selects Q features $\mathcal{S}_k^{l^*}$ from $\mathcal{A}_k$ by the best algorithm l^*. For simplicity, we denote the selected feature set as $\mathcal{G}_k \equiv \mathcal{S}_k^{l^*}$ with $|\mathcal{G}_k| = Q$. Then the process goes back to main process adaptation.

Main Adaptator: For all other features in the total p features, $X_h \in \{X_1, \ldots, X_p\} - \mathcal{G}_k$, we add each one to $\mathcal{G}_k$ and compute the loss function:

$$L_h = \sum_{i=1}^{n} L(y_i, \beta_0 + \sum_{j \in \mathcal{G}_k} \beta_j x_{ij} + \beta_h x_{ih})$$

Main Concentrator: rank all L_h with $X_h \in \{X_1, \ldots, X_p\} - \mathcal{G}_k$ from low to high, and select the top features $C - Q$ with the smallest L_h.

Main Propagator: add these features to $\mathcal{G}_k$ to form the new feature set $\mathcal{A}_{k+1}$. Note that the size of $\mathcal{A}_{k+1}$ is C.

In this entire process, the time controller controls the termination of whole process. It sets a threshold K , then checks whether the total number of cycles is equal to or larger than threshold. If yes, then it stops the process and outputs the feature selection result. That is, if $k > K$, it stops after sub concentration process and outputs the selected Q features; or if $k < K$, the process continues to main adaptation. The comprehensive algorithm for high dimensional sparse learning system via SIS is shown in Algorithm 4. Sub enumeration phrase inquires knowledge base and uses those stored algorithms to compute the top Q feature set $\mathcal{S}_k^l$, where l denotes the number of different algorithms. Time controller is the decision phrase which controls the termination of whole process. Thus, knowledge base and time controller design are important issues for whole system architecture.

Algorithm 4 High-dimensional sparse learning via SIS

Enumerate p features $X_1, \ldots, X_p$

Compute pearson correlation R_j and eliminate $p - C$ features with the lowest R_j and denote the selected feature set as $\mathcal{A}_1$ with $|\mathcal{A}_1| = C$.

for $k = 1$ to K **do**

  for each algorithm l in the knowledge base **do**

    Select the top Q features $\mathcal{S}_k^l \subset \mathcal{A}_k$ and compute the corresponding loss function L^l.

  end for

end for

Choose the best algorithm l^* which achieves the minimum loss and set $\mathcal{G}_k = \mathcal{S}_k^{l^*}$.

for each feature $X_h \in [p] \backslash \mathcal{G}_k$ **do**

  Add the feature X_h to $\mathcal{G}_k$ and compute the corresponding loss function L_h.

end for

Rank the loss L_h and select the top $C - Q$ features with the smallest L_h.

Add these features to $\mathcal{G}_k$ and form the new set $\mathcal{A}_{k+1}$.

Output: The selected feature set $\mathcal{A}_{K+1}$.

In [Chang, 2010], the author pointed out that problem and solution are both functions of times in SIS system. The time controller is also a time function to control the two decision cycles. In our proposed feature selection system, we use a time controller by defining a threshold K to check the loop condition in whole process. If $K = 1$, the whole process does not contain main adaptation, main concentration and propagation phrases. It just outputs sub concentrator's results of the initial run. If $K > 1$, the system iteratively selects features and outputs the results after K cycles. The larger the K is, the more accurate feature selection solution is, which will be shown in the experiment. Thus, we can identify that our feature selection system possesses the main characteristics of SIS system: slow decision cycles complement quick decision cycles. When K is smaller, the quick decision cycle outputs less accurate result. When K is larger, the slow decision cycle can result in better performance. Also, the quick decision cycle's result can be used and adapted into the slow cycle to enable the system having great long-term goals. That is one of the important reasons that why our

		True Class	
		Positive	Negative
Predicted Class	Positive	a	b
	Negative	c	d

Table 10: Confusion matrix for the test data where $a+b+c+d$ are the number of testing samples

system can outperform individual state-of-the-art feature selection method.

We introduce a knowledge base in the sub enumeration phrase, which stores the existing feature selection algorithms e.g., ℓ_1-regularized regression, elastic-net penalty regularized regression and forward stepwise regression, etc. When we design such knowledge base, we can add any individual feature selection method which is generic or specialized candidate algorithm. Generic one means a method template with different parameters to generate different candidate algorithms. Specialized one means the particular algorithm with no parameters. For example, we note that in (2.4), elastic-net regularized regression reduces to the standard ℓ_1-norm regularized regression when $\alpha = 1$. It means that elastic-net regularized regression (glmnet) can be a generic candidate algorithm with parameter α . ℓ_1-norm regularized approach is one candidate elastic-net method with parameter $\alpha = 1$. In the experiment, we also use another candidate elastic-net method with parameter $\alpha = 0.5$. But forward stepwise regression is a specialized candidate algorithm.

6.2.3 Experiments

We evaluate the performance of our SIS-based sparse learning system for a simple task: binary classification problem where responses $Y \in \{0,1\}^p$ on two cancer datasets [Chang and Lin, 2011]. In next section 6.3, we will apply this similar system on the more complex task: regression problem for the influential nodes detection on large scale diffusion network. Firstly, we use the leukemia data set [Golub et al., 1999]. Leukemia is a type of cancer

of the blood or bone marrow characterized by an abnormal increase of white blood cells. This dataset consists of 72 samples including 47 acute myeloid leukemia (ALL) ($Y_i = 1$) and 25 patients with lymphoblastic leukemia (AML) ($Y_i = 0$). For each sample, the dataset contains the expression levels of 7129 human genes (X) measured by Affymetrix high-density gene chips. The data is separated to training set with 38 samples (27 ALL and 11 AML) and testing set with 34 samples (20 ALL and 14 AML). The second dataset we use is the colon-cancer dataset [Alon et al., 1999]. It consists of 62 samples including 40 tumor colon tissues ($Y_i = 1$) and 22 normal colon tissue ($Y_i = 0$). For each sample, the dataset contains intensities of 2,000 genes (X) analyzed with an Affymetrix oligonucleotide array. Similarly, the data is separated into 32 training cases and 30 testing cases. As we can see, both datasets are ultra-high dimensional in the sense that the number of features are hundred times of the sample size.

We compare our sparse learning system via SIS with the three individual sparse learning algorithms in our knowledge base, namely, forward regression, ℓ_1-regularized logistic regression and elastic-net regularized logistic regression with $\alpha = 0.5$. In our system, we set the constant $C = 500$ in the main elimination phase and report the results by varying the threshold K for the time controller and the number of selected features Q in the sub system. More specifically, given the confusion matrix as in Table 10, where a, b, c and d represent the number of examples falling into each possible outcome, we report the number errors out of the total testing samples: $(b + c)/(a + b + c + d)$ and the balanced error rate: $0.5(\frac{b}{a+b} + \frac{c}{c+d})$. The results for leukemia and colon-cancer datasets are shown in Table 11 and 12. In addition, we list the selected genes descriptions of leukemia data by SIS ($K = 5$) and SIS ($K = 10$) in Table 13.

As we can see from Table 11 and Table 12, for both datasets, our method greatly outperforms individual feature selection algorithm. For individual feature selection algorithm, ℓ_1-regularzied Logistic regression and glmnet achieve better classification results as compared to forward stepwise regression. It might because the forward stepwise regression is "too" greedy. However, it is hard to say which one is better between ℓ_1-regularized logistic regression and glmnet. Our system utilizing all these three methods is superior to any individual one. In addition, as we can see, when we increase K , the number of cycles defined by time

Methods	$Q = 10$ features		$Q = 20$ features		$Q = 30$ features	
Forward Regression	8/34	0.2857	8/34	0.2857	8/34	0.2750
L1 Logistic Regression	5/34	0.1786	2/34	0.0714	4/34	0.1429
Glmnet with $\alpha = 0.5$	6/34	0.2143	3/34	0.1071	1/34	0.0357
SIS ($K = 5$)	2/34	0.0714	2/34	0.0714	1/34	0.0357
SIS ($K = 10$)	1/34	0.0357	1/34	0.0357	1/34	0.0357

Table 11: Performance of the Leukemia dataset: number of errors and the balance of error rate

Methods	$Q = 10$ features		$Q = 20$ features		$Q = 30$ features	
Forward Regression	8/30	0.3175	8/30	0.3175	11/30	0.4206
L1 Logistic Regression	5/30	0.1825	8/30	0.3175	8/30	0.2857
Glmnet with $\alpha = 0.5$	5/30	0.2143	6/30	0.2381	10/30	0.4206
SIS ($K = 5$)	4/30	0.1587	8/30	0.3175	8/30	0.2857
SIS ($K = 10$)	4/30	0.1587	5/30	0.1825	7/30	0.2619

Table 12: Performance of the Colon cancer dataset: number of errors and the balance of error rate

controller, the accuracy of our system improves. In fact, using a larger K needs more time to run, but can provide us a more accurate solution. In other words, slow cycles result in better performance in long run. It is a natural tradeoff between the running time and the performance of our feature selection system.

In addition, we present the selected genes of our system by SIS ($K = 5$) and ($K = 10$) on leukemia dataset. It is known from biological background [Fang and Grzymala-Busse,

leukaemia SIS gene selection ($K = 5$)
INDUCED MYELOID LEUKEMIA CELL DIFFERENTIATION PROTEIN MCL1
LYN V-yes-1 Yamaguchi sarcoma viral related oncogene homolog
CD33 CD33 antigen (differentiation antigen)
FAH Fumarylacetoacetate
PEPTIDYL-PROLYL CIS-TRANS ISOMERASE, MITOCHONDRIAL PRECURSOR
DF D component of complement (adipsin)
Leukotriene C4 synthase (LTC4S) gene
PRG1 Proteoglycan 1, secretory granule
Zyxin
LEPR Leptin receptor
leukaemia SIS gene selection ($K = 10$)
CYSTATIN A
LYN V-yes-1 Yamaguchi sarcoma viral related oncogene homolog
CST3 Cystatin C (amyloid angiopathy and cerebral hemorrhage)
FAH Fumarylacetoacetate
Leukotriene C4 synthase (LTC4S) gene
RETINOBLASTOMA BINDING PROTEIN P48
Zyxin
C-myb gene extracted from Human (c-myb) gene, complete primary cds, and five complete alternatively spliced cds
ELA2 Elastatse 2, neutrophil
TCF3 Transcription factor 3 (E2A immunoglobulin enhancer binding factors E12/E47)

Table 13: The selected genes description of leukemia data by SIS ($K = 5$) and SIS ($K = 10$)

2006] that these genes are critical for the leukemia disease. For example, Zyxin selected by both SIS ($K = 5$) and SIS ($K = 10$) processes LIM domain which is known to interact with leukemogenic bHLH proteins [Wadman et al., 1994]. For SIS ($K = 10$), it finds very important two genes that are not discovered by SIS ($K = 5$). The first one is Cystatin C (CST3), measured by reverse transcription polymerase chain reaction (RTPCR), and confirmed that this gene is significantly increased in AML patients [Sakhinia et al., 2005]. The other one is Cystatin A, a natural inhibitor of cysteine proteinases, which has been

suggested that an inverse correlation exists between cystatin A and malignant progression [Kuopio et al., 1998]. Therefore, SIS ($K = 10$) is leading to more meaningful feature selection results. All other methods miss these three important genes and the selected genes are less relevant to leukemia.

In conclusion, we propose a new sparse learning system using the framework of slow intelligence principle. It iteratively selects the best individual feature selection algorithm and propagates the learned features over iterations. It leads a superior feature selection performance and can handle ultra high dimensional data. The experimental results show that our method greatly outperforms individual feature selection algorithm, e.g., ℓ_1-regularzied Logistic regression, glmnet and forward stepwise regression.

6.3 SIS-BASED METHOD FOR INFLUENTIAL NODE DETECTION IN SPARSE LINEAR INFLUENCE MODEL

Based on our framework of SIS-based sparse learning in Section 6.2, we develop the SIS-based methods for our sparse linear influence models for influential node detection. We describe the main components of our SIS-based methods as follows.

6.3.1 SIS-based SLIM

In this section, we provide the details of the SIS-based algorithm for single-task SLIM model. The algorithm for multi-task MSLIM will be presented in the next section.

Enumerator: Enumerate N users, where each user $u \in \{1, \ldots, N\}$ is associated with data matrices $\mathbf{M}_u \triangleq (\mathbf{M}_{u1}, \ldots, \mathbf{M}_{uK})$. Here, M_{uk} is presented in Figure 3 in section 3.2. Among these users, some are influential users and some are not.

Eliminator: Due to the complicated structure of M_{uk} matrix, it is hard to construct the main eliminator component simply using the pearson correlation in section 6.2. Instead, we propose to construct the eliminator using the marginal regression approach where we regress

the volume vector $\mathbf{V}_k$ on each $\mathbf{M}_u$. In particular, we compute:

$$\begin{aligned} R_u = \text{minimize} \quad & \tfrac{1}{2}\sum_{k=1}^{K} \|\mathbf{V}_k - \mathbf{M}_{uk} \cdot \mathbf{I}_u\|_2^2, \\ \text{subject to} \quad & \mathbf{I}_u \geq 0 \end{aligned} \tag{6.1}$$

We rank R_u for $1 \leq u \leq N$ from low to high and eliminate the highest $(N - C)$ users where C is a pre-defined constant. We denote the selected top C users as $\mathcal{A}_1$.

As we can see, the **Eliminator** procedure greatly enhances the scalability for influential node detection in sparse linear influence model. Instead of running the optimization on the entire data matrix $\mathbf{M}$ of the size $N \times K \times T \times L$, which may not be able to be stored in memory, we first leave $N - C$ users out of our model and thus the size of the data matrix can be reduced to $C \times K \times T \times L$.

In our original framework in section 6.2, we construct the knowledge base with three different approaches. Since we only develop the SLIM for influential node detection in this thesis, we skip the sub-enumerator and sub-concentrator steps. In fact, it is possible to construct the elastic-net regularized SLIM by introducing another regularization term $\alpha\|I\|_2^2$. However, the extra tuning parameter will lead to a more painful tuning process in large scale datasets. One can also construct the forward stepwise regression for SLIM. However, for multi-task MSLIM, in each greedy step, one needs to search for the next candidate user for all K topics, which is also computationally expensive. Therefore, in our SIS-based method, we only focus on SLIM and we iterate the following process from $r = 1$ to a certain number of iterations which is defined by the time controller.

Concentrator: We apply the SLIM only to the selected user set $\mathcal{A}_r$ by solving the following optimization algorithm:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\sum_{k=1}^{K} \|\mathbf{V}_k - \textstyle\sum_{u \in \mathcal{A}_r} \mathbf{M}_{uk} \cdot \mathbf{I}_u\|_2^2 + \lambda \sum_{u \in \mathcal{A}_r} \|\mathbf{I}_u\|_2 \\ \text{subject to} \quad & \mathbf{I}_u \geq 0 \quad \forall\, u \in \mathcal{A}_r \end{aligned}$$

We denote by Q the number of selected influential nodes from the previous optimization procedure, i.e., $Q = |\{u \in \mathcal{A}_r : \|\mathbf{I}_u\|_2 > 0\}|$.

Adaptator: We denote by $\mathcal{G}_r \subseteq \mathcal{A}_r$ the selected users from the above optimization procedure. For all other users in $h \in \{1, \ldots, N\} \backslash \mathcal{G}_r$, we add user h to $\mathcal{G}_r$ and compute the following regression loss function:

$$\begin{aligned} L_h = \text{minimize} \quad & \tfrac{1}{2} \textstyle\sum_{k=1}^{K} \| \mathbf{V}_k - \sum_{u \in \mathcal{G}_r \bigcup \{h\}} \mathbf{M}_{uk} \cdot \mathbf{I}_u \|_2^2 \\ \text{subject to} \quad & \mathbf{I}_u \geq 0 \quad \forall\, u \in \mathcal{G}_r \bigcup \{h\} \end{aligned}$$

We note that here, since $\mathcal{G}_r \bigcup \{h\}$ has a very small cardinality, the corresponding regression problem is no longer a ultra high-dimensional one; and therefore, we do not need a sparsity-inducing regularization.

Propagator: rank all L_h for each $h \in \{1, \ldots, N\} \backslash \mathcal{G}_r$ from low to high, and select the top features $C - Q$ with the smallest L_h. Then add these users to $\mathcal{G}_r$ to form the new feature set $\mathcal{A}_{r+1}$. Note that the size of $\mathcal{A}_{r+1}$ is our pre-defined C.

In this entire process, the time controller controls the termination of whole process. It sets a threshold R , then checks whether the total number of cycles is equal to or larger than this threshold. If yes, then it stops at the Concentrator stage and outputs the selected influential user set $\mathcal{G}_R$.

6.3.2 SIS-based MSLIM

The SIS-based method for Multi-task MSLIM is similar to that for single-task SLIM. Therefore, we briefly describe the method and only highlight the main differences.

Enumerator: For each topic $1 \leq k \leq K$, we enumerate N users, where each user $u \in \{1, \ldots, N\}$ is associated with data matrices $\mathbf{M}_{uk}$ (see Figure 3 in section 3.2). Among these users, some are influential users for the topic k and some are not.

Eliminator: For each topic k, we use the marginal regression approach to first eliminate $N - C_k$ users and only keep C_k users in the model. We note that in contrast to the single-task scenario, for different topic k, the set of C_k users could be different. In particular, we

compute:

$$\begin{aligned} R_u^k = \text{minimize} \quad & \tfrac{1}{2}\|\mathbf{V}_k - \mathbf{M}_{uk} \cdot \mathbf{I}_u^k\|_2^2, \\ \text{subject to} \quad & \mathbf{I}_u^k \geq 0 \end{aligned} \tag{6.2}$$

We rank R_u^k for $1 \leq u \leq N$ from low to high and eliminate the highest $(N - C_k)$ users where C_k is a pre-defined constant. We denote the selected top C_k users for the topic k as $\mathcal{A}_1^k$.

Then, we iterate the following process from $r = 1$ to a certain number of iterations R which is defined by the time controller.

Concentrator: We apply the MSLIM only to the selected user sets for K topics $\{\mathcal{A}_r^k\}_{k=1}^K$ by solving the following optimization algorithm:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\sum_{k=1}^K \|\mathbf{V}_k - \textstyle\sum_{u \in \mathcal{A}_r^k} \mathbf{M}_{uk} \cdot \mathbf{I}_u^k\|_2^2 + \lambda \sum_{u=1}^N \sqrt{\sum_{k:\mathcal{A}_r^k \ni u} \|\mathbf{I}_u^k\|_2^2} + \gamma \sum_{k=1}^K \sum_{u \in \mathcal{A}_r^k} \|\mathbf{I}_u^k\|_2 \\ \text{subject to} \quad & \mathbf{I}_u^k \geq 0 \quad \forall\, 1 \leq k \leq K, u \in \mathcal{A}_r^k \end{aligned}$$

We denote by Q_k the number of selected influential nodes from the previous optimization procedure, i.e., $Q_k = |\{u \in \mathcal{A}_r^k : \|\mathbf{I}_u^k\|_2 > 0\}|$.

Adaptator: For each topic k, we denote by $\mathcal{G}_r^k \subseteq \mathcal{A}_r^k$ the selected users for the topic k from the above optimization procedure. For all other users in $h \in \{1, \ldots, N\} \backslash \mathcal{G}_r^k$, we add user h to $\mathcal{G}_r^k$ and compute the following regression loss function:

$$\begin{aligned} L_h^k = \text{minimize} \quad & \tfrac{1}{2}\|\mathbf{V}_k - \textstyle\sum_{u \in \mathcal{G}_r^k \bigcup \{h\}} \mathbf{M}_{uk} \cdot \mathbf{I}_u\|_2^2 \\ \text{subject to} \quad & \mathbf{I}_u^k \geq 0 \quad \forall\, u \in \mathcal{G}_r^k \bigcup \{h\} \end{aligned}$$

Propagator: for each topic k, rank all L_h^k for each $h \in \{1, \ldots, N\} \backslash \mathcal{G}_r^k$ from low to high, and select the top features $C_k - Q_k$ with the smallest L_h^k. Then add these users to $\mathcal{G}_r^k$ to form the new feature set $\mathcal{A}_{r+1}^k$. Note that the size of $\mathcal{A}_{r+1}^k$ is our pre-defined C_k.

6.3.3 Experiments

In this section, we evaluate the performance of SIS-based SLIM/MSLIM on a new larger twitter dataset. We first describe the dataset and then compare our proposed new models with original SLIM/MSLIM models on prediction MSE and CPU running time on this dataset.

6.3.3.1 Data Collection In the previous experiments in chapter 3 and chapter 4, we conduct investigations on the twitter dataset containing 1000 twitter users. Here, in order to show the scalability of our new SIS-based SLIM/MSLIM models, we collect a new larger twitter dataset to evaluate the models. The new dataset contains 9,860 users including those 1000 twitter users in the previous twitter dataset. We follow the same procedure described in section 3.5.1 to crawl all the tweets of these 9860 twitter users between January 2009 and November 2011, which include the full text, the author and the time-stamp for each tweet. In addition, we collect the profile of each twitter user. In our new dataset, on average, each user has 585.2 tweets and the maximum number of tweets for a specific user is 13,520. We conduct the standard Twitter-data cleaning procedure and topic modeling described in section 3.5 on this new data corpus.

6.3.3.2 Experimental results analysis We conduct the experiments on the new collected larger twitter dataset to show the performance of SIS-based method for SLIM, denoted by SIS-SLIM; and SIS-based method for Multi-task MSLIM, denoted by SIS-MSLIM. Here, to show the superior scalability of the SIS-SLIM and SIS-MSLIM, we consider three different subsets of the entire dataset: (1) 1000 users (original dataset used in previous experiments); (2) 5000 users (uniformly sampled dataset from the entire dataset); (3) the entire dataset with 9,860 users.

The experiments are carried out on a standard PC with 4GB RAM in Matlab. The results for original twitter dataset containing 1000 users are presented in Table 14. We set $C(\text{or } C_k) = 250$ in SIS-based approaches and tune the regularization parameters (λ, γ) to select the same number of influential users as SLIM (or MSLIM) for fair comparison. In SLIM

	Time	Testing MSE	Training MSE
SLIM	118.12	20.24	14.82
MSLIM	170.62	13.03	10.15
SIS-SLIM	141.74	17.91	15.63
SIS-MSLIM	238.87	11.88	10.72

Table 14: Comparison between SIS-SLIM/MSLIM and SLIM/MSIM in terms of MSE and CPU Time in seconds on original Twitter dataset

and MSLIM models, we use the best parameters explained in Section 3.5.3 and 4.5.1 (i.e. $L = 5$, $\lambda = 400$ for SLIM, $\lambda = 300, \gamma = 42.43$ for MSLIM). We set the number of iterations $R = 10$ for SIS-based approaches here and we will analyze the system performance based on different values of R and C on the larger datasets later. As we can see, for this smaller dataset, SIS-based approaches perform slightly worse than one-stage methods SLIM/MSLIM on training sets but achieve smaller MSE on the entire time span dataset. However SIS-based methods need a longer computational time since they are iterative approaches.

For larger datasets with 5,000 users or 9,860 users, SLIM and MSLIM cannot be applied due to the ultra large scale of the data, which exceeds the memory limit when applying optimization method. In contrast, since SIS-based approaches only maintain a small set of users in the memory at each iteration, they greatly improve the computation scalability. Our SIS-based methods contain two main important system parameters C and R. Both of them are pre-defined constant in our SIS-based approaches. We will analyze how these two parameters affect our system performance.

- Discussion on C (or C_k) in SIS-SLIM/MSLIM:

  In this experiment, we vary the parameter $C \in \{250, 500, 1000\}$ while fix the total number of iterations $R = 10$ and run the proposed SIS-SLIM on the new larger twitter dataset. For SIS-MSLIM model, we simply set the same C_k for all the topics $k \in \{1, 2, \ldots, K\}$

	$C = 250$		$C = 500$		$C = 1000$	
Total No. Users	Time	MSE	Time	MSE	Time	MSE
5,000	850.46	15.56	1200.78	11.85	2000.14	9.68
9,860	1000.45	17.15	1476.5	13.63	2600.08	10.73

Table 15: Comparison of different values of C for SIS-SLIM in terms of MSE and CPU Time in seconds on the new larger Twitter datasets

	$C_k = 250$		$C_k = 500$		$C_k = 1000$	
Total No. Users	Time	MSE	Time	MSE	Time	MSE
5,000	1399.08	9.71	2035.45	7.87	3125.17	6.88
9,860	1635.47	11.78	2405.74	8.14	3965.5	7.34

Table 16: Comparison of different values of C_k for SIS-MSLIM in terms of MSE and CPU Time in seconds on the new larger Twitter datasets

	$R = 10$		$R = 20$		$R = 30$	
Total No. Users	Time	MSE	Time	MSE	Time	MSE
5,000	1200.78	11.85	2393.56	11.74	3595.24	11.56
9,860	1476.5	13.63	2944.45	13.38	4418.5	13.06

Table 17: Comparison of different values of R for SIS-SLIM in terms of MSE and CPU Time in seconds on the new larger Twitter datasets

and vary this value similarly as C in SIS-SLIM. From Table 15 and 16, we can see that the larger C (or C_k) is, the better predicted MSE achieved. We have more accurate

	$R = 10$		$R = 20$		$R = 30$	
Total No. Users	Time	MSE	Time	MSE	Time	MSE
5,000	2035.45	7.87	4059.48	7.53	6089.14	7.37
9,860	2405.74	8.14	4798.75	7.93	7178.22	7.79

Table 18: Comparison of different values of R for SIS-MSLIM in terms of MSE and CPU Time in seconds on the new larger Twitter datasets

predictions when using larger C while the computation time is longer because the model complexity is increasing. Also we find that even we increase the C from 500 to 1000, the model's performance increase slightly.

- Discussion on R in SIS-SLIM/MSLIM:

  Here, we discuss the experimental results depending on different values of parameter R, the number of iterations. We set C (or C_k) as 500 and the number of selected influential users as 200 and vary the parameter $R \in \{10, 20, 30\}$. From the Table 17 and 18, the system performance only slightly improved for the larger R. It means that our approaches become stable after a certain number of iterations. However, using a larger R will bring more computational burden since the CPU time linearly increases with R.

  In summary, the experimental results show that our SIS-based approaches are much more scalable and flexible compared to the original SLIM and MSLIM models and can efficiently learn from large scale social networks with tens of thousands of users.

7.0 CONCLUSIONS AND FUTURE WORKS

7.1 CONCLUSIONS

In summary, this thesis makes some attempts to address the following challenges arising in forecasting contagions volume and influential node detection problems in information diffusion network:

1. **Implicit Network Structures:** Traditional models for information diffusion network analysis makes several assumptions: 1) complete network structure data is available, 2) contagion can only spread over the edges of the underlying network. However, in many scenarios, the network over which diffusion takes place is in fact implicit or even unknown. Most of the works for influential node detection are based on these strong assumptions. This thesis adapts and extends the existing Linear Influence Model by introducing a sparsity-inducing regularizer and proposes a new model so called Sparse Linear Influence Model (SLIM). Our SLIM model can simultaneously address the problems of contagion volume prediction and global influential node detection in an implicit information diffusion network. We further develop an efficient optimization scheme to solve the corresponding optimization problem and show that there is a closed-form solution for the key step, which achieves an optimal convergence rate. Our model can be broadly applicable to general diffusion process on social platform, as they do not require knowledge of the underlying network topology.
2. **Multiple contagions:** There are always multiple contagions propagating over the networks in the information diffusion process. Considering the correlations among different contagions in the model is important and necessary. One person should have different

influence on the different contagions. For different types of contagions, the set of the most influential nodes could be very different. Most of works strive to detect influential nodes for either single contagion or across all contagions. Our SLIM model can detect the global influential nodes for all contagions but it doesn't consider the different levels of influence for different contagions. In this thesis, we extend the proposed SLIM model to Multi-task sparse linear influence model (MSLIM) to conduct the contagion-sensitive node selection on the implicit information diffusion network. To solve the MSLIM model, which is formulated as a convex optimization problem, we propose an efficient deterministic optimization method which achieves the optimal first-order convergence rate. In addition, to further improve the scalability and storage efficiency, we propose a stochastic optimization algorithm for both SLIM and MSLIM.

3. **Scalability and Large-Scale:** The web social network datasets are often large-scale. In order to improve the model's scalability and flexibility, this thesis firstly proposes a novel and powerful component-based slow intelligence system which is a general-purpose system characterized by being able to improve performance over time through a process involving enumeration, propagations, adaptation, elimination and concentration. Furthermore, using the idea of SIS system, this thesis develops the SIS-based approaches named SIS-based SLIM and MSLIM for influential node detection in large-scale social networks.

In addition to the proposed single-task and multi-task sparse linear influence models, another contribution of this thesis is that we propose the fundamental architecture design and implementation of component-based slow intelligence system. We demonstrate that it has wide applications to many real-world problems. In this thesis, we present two important applications as follows:

1. **Face Recognition:** We propose a new image analysis system using the general component-based SIS for face recognition task. It iteratively selects the best individual image analysis algorithm and propagates the learned knowledge over iterations. It leads a superior accuracy of face recognition to individual method. Our system is beneficial to improve the human's learning skills on applying the appropriate approach for different image

variations.

2. **Tumor Classification:** Tumor classification from microarray data is an important issue in computational biology, which could potentially help detecting cancer at an earlier stage. One of the challenging problems is to choose the appropriate feature selection algorithm to facilitate the classification task. Another challenge is that for some approaches, when the dimension of genes is ultra-high, it cannot provide the solution due to the computational limitations. In this thesis, we use the SIS principle and specialize it to the gene expression levels selection task. Our new system can select the features using different methods, search for the best method according to the status quo and propagate the learned features over iterations. Our results on the leukemia and colon cancer datasets show that we not only achieve a better classification accuracy as compared to the individual approach, but also identify some important genes.

All of the aforementioned challenges are main challenges for influential node detection in information diffusion network analysis. This thesis takes attempts to address these challenges and opens up many possible future works. Moreover, slow intelligence approach has a much wider spectrum of applications in addition to the applications studied in this thesis, e.g., object tracing, ontology filtering, pet care, etc. In the next section, we will discuss a few promising future directions.

7.2 FUTURE DIRECTIONS

There are several future directions as follows:

1. **Extensions for influential node detection models:** In this thesis, we study how to identify the influential nodes for the different contagions in the large scale implicit social networks. Our models do not need the structure of the network because in many scenarios, the network over which information diffusion takes place is unknown. We assume that the node is influenced by other nodes in the social networks when it adapts the information. Sometimes an activation of a node is not just a function of the social

network but also depends on many other factors like imitation and recency. For example, so far, we have assumed that a node has the same influence function regardless of how early or late in the diffusion it appears. However, nodes are more likely to adopt novel and recent information while ignoring old and obsolete information. How to account this effect of recency and novelty in the model needs to be further explored. In addition, people discover new information or make decisions by using many other means like the search engines, news sites, TV, etc. How to capture these complex factors in the model is one of major challenges in future.

In this thesis, we utilize three social network datasets to evaluate our models. Our model can be broadly applicable to general information diffusion networks when the network structure is unknown, like virus propagation network, viral marketing settings, blogosphere domain, etc. In the future, it will be very fruitful to apply our models on more domain datasets and explore new findings from real applications.

2. **Computation:** When the information diffusion datasets are ultra-large, computation will always be one of the major bottlenecks for learning. In this thesis, we have discussed the efficient deterministic and stochastic optimization schemes. However there is still much more room for improvement. To improve the scalability, we can develop parallel/distributed algorithms on multi-core machines or connected clusters to learn from ultra-large scale data. The key problem is to develop the appropriate different locking schemes in distributed optimization.

3. **Applications of Slow Intelligence System:** We hope the component-based slow intelligence system proposed in this thesis could have more applications in various domains. For example, we proposed a web-based pet care system based on slow intelligence system recently [Chang et al., 2013a]. This system can help the pet owners monitor the pet behaviors when they are away from home and it can be further applied to senior patient care application. In the future, it will be very meaningful to collaborate with domain experts to explore new challenges from real applications and propose SIS-based system to address these challenges.

   Moreover, we have developed the efficient models for influential node detection and forecasting contagion volumes in this thesis. In the future, we would like to further add

human interaction components to the current component-based SIS system. For example, human can control the parameters in the cycles or use their domain knowledge to pre-select the features(e.g. influential nodes) in elimination stage. Eventually, the ultimate goal is to provide user-friendly software or the real system for general large-scale social influence analysis.

8.0 BIBLIOGRAPHY

E. Adar, J. Zhang, L. A. Adamic, and R. M. Lukose. Implicit structure and the dynamics of blogspace. In *Workshop on the Weblogging Ecosystem*, 2004.

U. Alon, N. Barkai, D. A. Notterman, K. Gish, S.Ybarra, D.Mack, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Cell Biology*, 96:6745–6750, 1999.

R. M. Anderson and R. M. May. *Infectious diseases of humans: Dynamics and control*. Oxford Press., 2002.

A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Machine Learning*, 73:243–272, 2008.

E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts. Everyone's an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, 2011.

A. Beck and M. Teboulle. A fast iterative shrinkage thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Science*, 2:183–202, 2009.

P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 1997.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2 edition, 2004.

P. Bickel, Y. Ritov, and A. Tsybakov. Simultaneous analysis of lasso and dantzig selector. *Annals of Statistics*, 37(4):1705–1732, 2009.

F. Biessmann, J.-M. Papaioannou, M. Braun, and A. Harth. Canonical trends: Detecting trend setters in web data. In *International Conference on Machine Learning*, 2012.

S. Bikhchandani, D. Hirshleifer, and I. Welch. A theory of fads, fashion, custom, and cultural change in informational cascades. *Journal of Political Economy*, 100(5):992–1026, 1992.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Network ISDN Systems*, 30:107–117, 1998.

P. J. Brockwell and R. Davis. *Time Series: Theory and Methods*. Springer, 1991.

R. Caruana. Multitask learning. *Machine Learning Journal*, 28:41–75, 1997.

M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi. Measuring user influence in twitter: The million follower fallacy. In *ICWSM 10: Proceedings of international AAAI Conference on Weblogs and Social*, 2010.

C.-C. Chang and C.-J. Lin. Libsvm:a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

S. Chang. A general framework for slow intelligence systems. *International Journal of Software Engineering and Knowledge Engineering*, 20(1):1–16, 2010.

S.-K. Chang, E. Zegarra, C. C. Shih, T.-C. Peng, F. Colace, and M. D. Santo. Ontological filters for slow intelligence systems. In *Proceedings of the 16th International Conference on Distributed Multimedia Systems*, 2010.

S.-K. Chang, Y. Sun, Y. Wang, C.-C. Shih, and T.-C. Peng. Design of component-based

slow intelligence systems and application to social influence analysis. In *Proceedings of 23rd International Conference on Software Engineering and Knowledge Engineering*, 2011a.

S. K. Chang, Y. Wang, and Yao. Visualization specification of component-based slow intelligence systems. In *Proceedings of 23rd International Conference on Software Engineering and Knowledge Engineering*, 2011b.

S.-K. Chang, L.-Q. Kuang, Y. Sun, and Y. Wang. Design and implementation of image analysis system by applying component-based slow intelligence system. In *Proceedings of The 18th International Conference on Distributed Multimedia Systems*, 2012.

S.-K. Chang, B. Gao, L. Kuang, and Y. Wang. A petcare system designed by slow intelligence principles. In *Proceedings of International Conference on Distributed Multimedia Systems*, 2013a.

S. K. Chang, Y. Wang, and Y. Sun. Component-based slow intelligence system. *Journal of Internet Technology*, 14, 2013b.

S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific and Statistical Computing*, 20:33–61, 1998.

W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010.

X. Chen, W. Pan, J. T. Kwok, and J. G. Carbonell. Accelerated gradient method for multi-task sparse learning problem. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, 2009.

F. Colace, S.-K. Chang, and M. D. Santo. Sinms: A slow intelligence network manager based on snmp protocol. In *Proceedings of the 16th International Conference on Distributed Multimedia Systems*, 2010.

J. S. Coleman, E. Katz, and H. Menzel. *Medical Innovation: A Diffusion Study*. Bobbs-Merrill Co, 1966.

G. Dantzig and M. Thapa. *Linear Programming 2: Theory and Extensions*. Springer-Verlag, 2003.

P. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001.

T. Dong. Modeling human intelligence as a slow intelligence system. In *Proceedings of the 16th International Conference on Distributed Multimedia Systems*, 2010.

B. A. Draper, K. Baek, M. S. Bartlett, and J. R. Beveridge. Recognizing faces with pca and ica. *Computer Vision and Image Understanding*, 91:115–137, 2003.

J. Duchi and Y. Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009.

J. Duchi, S. Shalev-Shwartz, Y. Singer, and A. Tewari. Composite objective mirror descent. In *Conference on Learning Theory (COLT)*, 2010.

J. Fang and J. W. Grzymala-Busse. Leukemia prediction from gene expression data: A rough set approach, artificial intelligence and soft computing. *ICAISC 2006 Lecture Notes in Computer Science,*, 4029:899–908, 2006.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1:302–332, 2007.

S. Ghadimi and G. Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization, i: a generic algorithmic framework. *SIAM Journal on Optimization*, 22:1469–1492, 2012.

J. Goldenberg, B. Libai, and E. Muller. Using complex systems analysis to advance marketing theory development. *Academy of Marketing Science Review*, 2001(9), 2001.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

M. S. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420–1443, 1978.

T. L. Griffiths and M. Steyvers. Finding scientific topics. In *Proc. of the National Academy of Sciences*, 2004.

D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. Information diffusion through blogspace. In *Proceedings of the 13th international conference on World Wide Web.*, 2004.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2nd edition, 2009.

E. Hazan and S. Kale. Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. In *Conference on Learning Theory (COLT)*, 2011.

H. W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):599–653, 2000.

C. Hu, J. T. Kwok, and W. Pan. Accelerated gradient methods for stochastic optimization and online learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

J. Huang and T. Zhang. The benefit of group sparsity. *Annals of Statistics*, 38(4):1978–2004, 2010.

M. Ilyas and H. Radha. Identifying influential nodes in online social networks using principal component centrality. In *IEEE International Conference on Communications*, 2011.

D. Kempe, J. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003.

D. Kempe, J. Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *Proceedings of the 32nd international conference on Automata, Languages and Programming*, 2005.

J. E. Kim, Y. Hu, S.-K. Chang, C.-C. Shih, and T.-C. Peng. Design and modeling of topic/trend detection system by applying slow intelligence system principles. In *Proceedings of the 17th International Conference on Distributed Multimedia Systems*, 2011.

M. Kimura, K. Saito, R. Nakano, and H. Motoda. Extracting influential nodes on a social network for information diffusion. *Data Mining and Knowledge Discovery*, 20(1):70–97, 2010.

R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. In *12th Annual World Wide Web Conference*, 2003.

T. Kuopio, A. Kankaanranta, P. Jalava, P. Kronqvist, T. Kotkansalo, E. Weber, and Y. Collan. Cysteine proteinase inhibitor cystatin a in breast cancer. *Cancer Research*, 58:432–436, 1998.

G. Lan and S. Ghadimi. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization, part ii: shrinking procedures and optimal algorithms. Technical report, University of Florida, 2010.

J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.

S. Lee and S. J. Wright. Manifold identification of dual averaging methods for regularized stochastic online learning. In *International Conference on Machine Learning (ICML)*, 2011.

J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *Journal ACM Transactions on the Web*, 1, 2007a.

J. Leskovec, M. Mcglohon, C. Faloutsos, N. Glance, and M. Hurst. Cascading behavior in large blog graphs. In *SIAM International Conference on Data Mining (SDM)*, 2007b.

J. Leskovec, L. Backstrom, and Kl. Meme-tracking ad the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.*, 2009.

D. a. K. Liben-Nowell. Tracing the flow of information on a global scale using internet chain-letter data. *Proceedings of the National Academy of Sciences.*, 105(12):4633–4638, 2008.

J. Liu, S. Ji, and J. Ye. Multi-task feature learning via efficient $\ell_{2,1}$-norm minimization. In *UAI*, 2009.

H. Ma, H. Yang, M. R. Lyu, and I. King. Mining social networks using heat diffusion processes for marketing candidates selection. In *Proceedings of the 17th ACM conference on Information and knowledge management*, 2008.

D. Mol, D. Vito, and L. Rosasco. Elastic net regularization in learning theory. *Journal of Complexity*, 25:201–230, 2009.

S. Negahban and M. J. Wainwright. Simultaneous support recovery in high dimensions: Benefits and perils of block ℓ_1/ℓ_∞-regularization. *IEEE Transactions on Information Theory*, 57 (6):3841–3863, 2011.

A. Nemirovski and D. Yudin. *Problem complexity and method efficiency in optimization*. John Wiley New York, 1983.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Y. Nesterov. *Introductory lectures on convex optimization: a basic course*. Kluwer Academic Pub, 2003.

Y. Nesterov. Gradient methods for minimizing composite objective function. ECORE Discussion Paper 2007, 2007.

M. Newman. he spread of epidemic disease on networks. *Phys. Rev. E*, 66(1):016128, 2002.

G. Obozinski, B. Taskar, and M. I. Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, 20:231–252, 2010.

G. R. Obozinski, M. J. Wainwright, and M. I. Jordan. High-dimensional union support recovery in multivariate regression. In *The 21st International Conference on Neural Information Processing Systems (NIPS)*, 2009.

X.-H. Phan and C.-T. Nguyen. GibbsLDA++: A C/C++ implementation of Latent Dirichlet Allocation (LDA), 2007.

P. J. Phillips, H. Moon, P. J. Rauss, and S. Rizvi. The feret evaluation methomethod for face recognition algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1090–1104, 2000.

E. M. Rogers. *Diffusion of Innovations, Fourth Ed.* Free Press, New York., 1995.

E. Sakhinia, M. Faranghpour, J. A. L. Yin, G. Brady, J. A. Hoyland, and R. J. Byers. Routine expression profiling of microarray gene signatures in acute leukaemia by real-time pcr of human bone marrow. *British Journal of Haematology*, 130:233–248, 2005.

F. S. Samaria and A. C. Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of 2nd IEEE workshop on Applications of Computer Vision*, pages 138–142, 1994.

T. Slocum, R. McMaster, F. Kessler, and H. Howard. *Thematic Cartography and Geovisualization (3rd Edition)*. Pearson Prentice Hall, 2009.

X. Song, Y. Chi, H. K, and B. Tseng. Information flow modeling based on diffusion rate for prediction and ranking. In *Proceedings of the 16th International Conference on World Wide Web*, 2007.

D. Strang and S. A. Soule. Diffusion in organizations and social movements: From hybrid corn to poison pills. *Annual Review of Sociology*, 24:265–290, 1998.

S. Thrum and L. Pratt. *Learning to Learn*. Kluwer Academic Publishers, 1998.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109 (3):475–494, 2001.

P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2009.

I. Wadman, J. X. Li, R. O. Bash, A. Forster, H. Osada, T. H. Rabbitts, and R. Baer. Specific in-vivo association between the bhlh and lim proteins implicated in human t-cell leukemia. *EMBO Journal*, 13:4831–4839, 1994.

M. J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programs. *IEEE Transactions on Information Theory*, 55:2183–2202, 2009.

Y. Wang and S.-K. Chang. High dimensional feature selection via a slow intelligence approach. In *Proceedings of the 17th International Conference on Distributed Multimedia Systems*, 2011.

Y. Wang., G. Xiang, and S. K. Chang. Sparse linear influence model for hot user selection on mining a social network. In *Proceeding of the 24th International Conference on Software Engineering and Knowledge Engineering*, 2012.

Y. Wang, G. Xiang, and S.-K. Chang. Sparse multi-task learning for detecting influential nodes in an implicit diffusion network. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13)*, 2013.

J. Weng, E.-P. Lim, J. Jiang, and Q. He. Twitterrank: finding topic-sensitive influential twitterers. In *Proceedings of the third ACM international conference on Web search and data mining*, 2010.

J. Williams and G. Katz. A new twitter verb lexicon for natural language processing. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, 2012.

G. Xiang, B. Fan, L. Wang, C. Rose, and J. Hong. Detecting offensive tweets via topical feature discovery over a large scale twitter corpus. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2012a.

G. Xiang, Z. Zheng, M. Wen, J. Hong, C. Rose, and C. Liu. A supervised approach to predict company acquisition with factual and topic features using profiles and news articles on techcrunch. In *Proceedings of the sixth international AAAI conference on weblogs and social media*, 2012b.

L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

J. Yang and J. Leskovec. Modeling information diffusion in implicit network. In *IEEE 10th International Conference on Data Mining*, 2010.

K. Yu, V. Tresp, and A. Schwaighofer. Learning gaussian processes from multiple tasks. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B*, 68:49–67, 2006.

J. Zhang and H. Liu. Simulation of face recognition algorithm based on wtpca and 3-neigbor classification. *Computer Engineering and Applications*, 45(11):175–177, 2009.

J. Zhang, Z. Ghahramani, and Y. Yang. Flexible latent variable models for multi-task learning. *Machine Learning*, 73(3):221–242, 2008.

T. Zhang. Some sharp performance bounds for least squares regression with l1 regularization. *Annals of Statistics*, 37:2109–2114, 2009.

P. Zhao and B. Yu. On model selection consistency of lasso. *Journal of Machine Learning Research*, 7:2541–2563, 2006.

H. Zou and T. Hastie. Regularization and variable selection via the elasticnet. *Journal of the Royal Statistical Society B*, 67:301–320, 2005.

APPENDIX

A SIMPLE EXAMPLE OF COMPONENT-BASED SLOW INTELLIGENCE SYSTEM

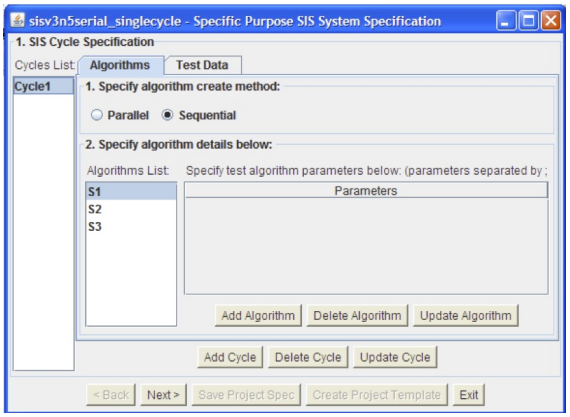
In order to illustrate how developers create specific purpose slow intelligence system by our software platform, we present a thorough explanation by a simple example.

A Simple Example: Given a sequence of numbers $\mathbf{I} = (I_1, I_2, \cdots, I_N)$ as input and another sequence of numbers $\mathbf{O} = (O_1, O_2, \cdots, O_N)$ as the expected output, we hope to find a suitable method to generate output from input. There are three algorithms S1, S2, S3, each one performs a specific method to generate following numbers. In particular, algorithm S1 uses Fibonacci sequence to generate the following numbers; algorithm S2 uses Random method; algorithm S3 generates consecutive numbers. All of these algorithms have the same input testing data and expected result.

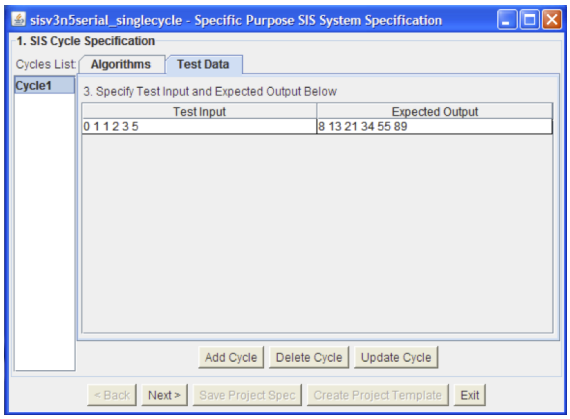
A.1 GUI FOR SYSTEM SPECIFICATION

- *Cycle specification*

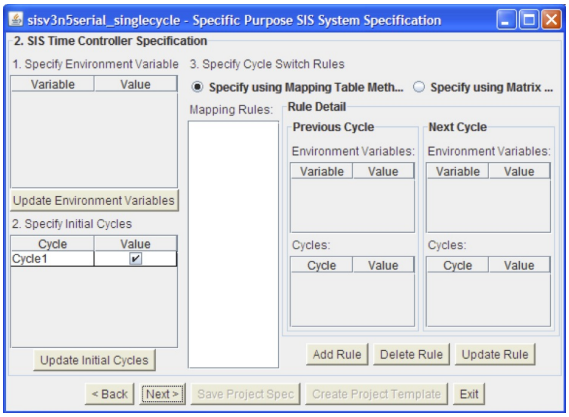
  Fig 29(a) shows how to configure cycles in SIS system for our simple example where there is only one cycle including three algorithms S1, S2 and S3. User can click on "Test Data" tab to specify the testing data sources. In our simple example, the input sequence of numbers and expected output sequence of numbers are listed as pairs, as shown in Fig. 29(b).



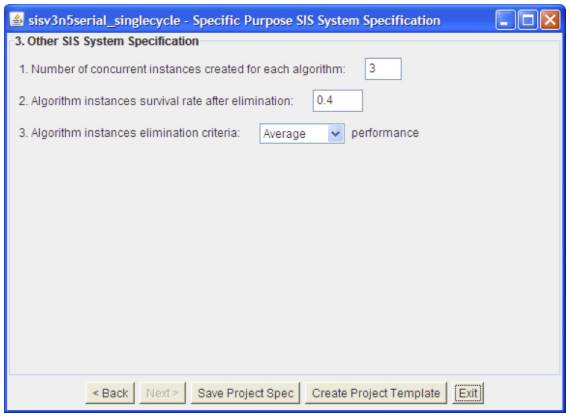
(a) The interface of cycle specification



(b) The interface of test data specification



(c) The interface of TimeController specification



(d) The interface of other system parameters specification.

Figure 29: Graphical interfaces of SIS system specification for the simple example.

- *TimeController specification*

Since there is only one cycle in our simple example, thus the specification is very simple as shown in Fig. 29(c). In the left top panel 1, all the environmental variables with their initial values are listed. SIS developers could edit the table directly. The value of an environment variable could be empty. In the left bottom panel 2, SIS developers could specify which cycle should be executed firstly. In the right panel, cycles switching rules in TimeController can be defined by mapping table. People can define the cycle number executed in the previous round and next round respectively in the sub-tables.

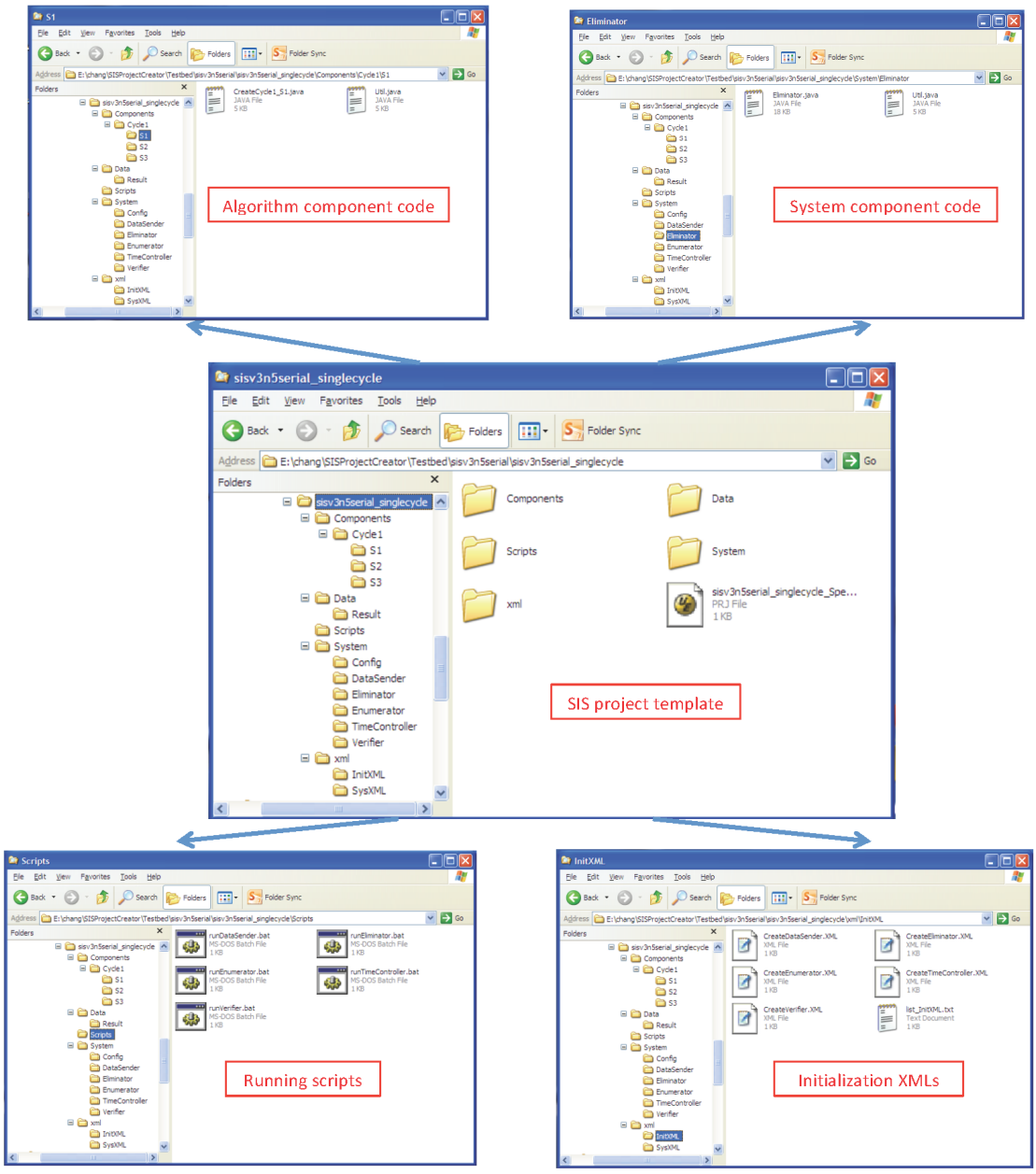


Figure 30: The SIS project template folder created by SISProjectCreator for the simple example.

In addition, the environment variable values in each round can also be configured.

- *Other system specification*

  Fig. 29(d) is the third stage of the Specific Purpose SIS System design.

After defining the specific SIS system components and parameters, a SIS project template folder is created by SISProjectCreator as shown in Fig. 30 .

A.2 PROJECT CODE GENERATION

Figure 31 shows the edited codes for algorithm S1 in our simple example. The part marked by yellow is edited by user while the other part of codes, that deals with the communication and operation in SIS system, is automatically created by SISProjectCreator. Similarly, user should implement the codes for specific algorithms S2 and S3.

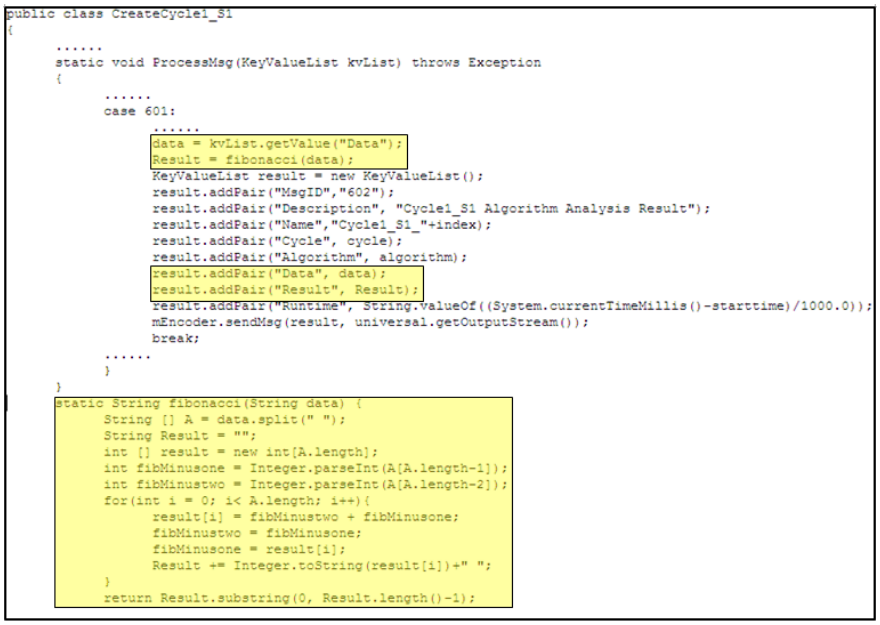
In our simple example, we use Euclidean distance as the evaluation rule to measure the difference between algorithm's output results $\widehat{\mathbf{O}}$ and the expected output sequence. The best algorithm selected by SIS system is the one with minimum distance.

$$\|\widehat{\mathbf{O}} - \mathbf{O}\|_2 = \sqrt{\sum_{i=1}^{N} (\widehat{O}_i - O_i)^2}$$

As shown in Figure 32, we only need to edit the *evaluatePerformance* function in Verifier.java and the other portions of codes has automatically generated by SISProjectorCreator.

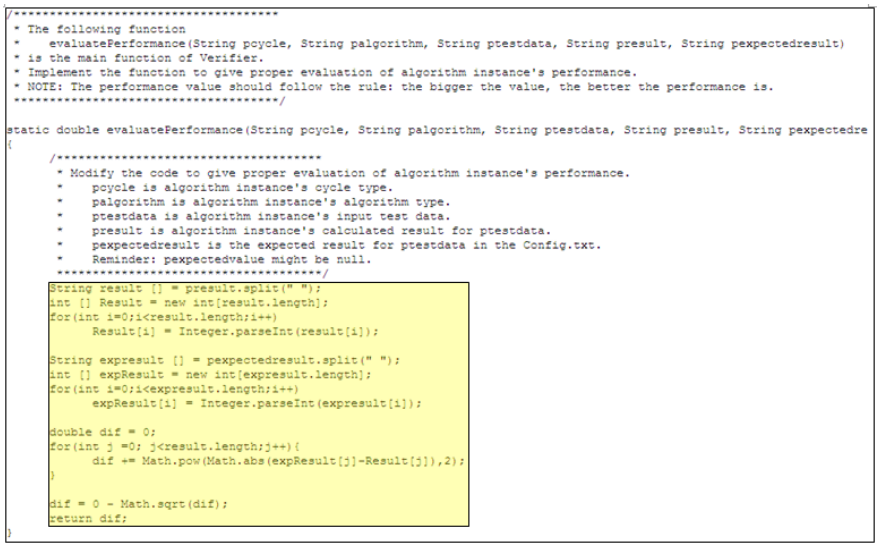
A.3 SIS SYSTEM SIMULATION

prjRemote in Figure 33 displays the results from each algorithm on the right panel. After all the algorithms complete, The output result is a plain text file as shown in Figure 34. Because the Euclidean distance between algorithm's output result and expected output sequence for the testing data is 0 by algorithm S1, thus which is selected by SIS system as the best algorithm.



```
public class CreateCycle1_S1
{
      ......
      static void ProcessMsg(KeyValueList kvList) throws Exception
      {
            ......
            case 601:
                  ......
                  data = kvList.getValue("Data");
                  Result = fibonacci(data);
                  KeyValueList result = new KeyValueList();
                  result.addPair("MsgID","602");
                  result.addPair("Description", "Cycle1_S1 Algorithm Analysis Result");
                  result.addPair("Name","Cycle1_S1_"+index);
                  result.addPair("Cycle", cycle);
                  result.addPair("Algorithm", algorithm);
                  result.addPair("Data", data);
                  result.addPair("Result", Result);
                  result.addPair("Runtime", String.valueOf((System.currentTimeMillis()-starttime)/1000.0));
                  mEncoder.sendMsg(result, universal.getOutputStream());
                  break;
            ......
            }
      }
      static String fibonacci(String data) {
            String [] A = data.split(" ");
            String Result = "";
            int [] result = new int[A.length];
            int fibMinusone = Integer.parseInt(A[A.length-1]);
            int fibMinustwo = Integer.parseInt(A[A.length-2]);
            for(int i = 0; i< A.length; i++){
                  result[i] = fibMinustwo + fibMinusone;
                  fibMinustwo = fibMinusone;
                  fibMinusone = result[i];
                  Result += Integer.toString(result[i])+" ";
            }
            return Result.substring(0, Result.length()-1);
```

Figure 31: The codes for candidate algorithm S1 in SIS system for the simple example.



```
/*************************************
 * The following function
 *    evaluatePerformance(String pcycle, String palgorithm, String ptestdata, String presult, String pexpectedresult)
 * is the main function of Verifier.
 * Implement the function to give proper evaluation of algorithm instance's performance.
 * NOTE: The performance value should follow the rule: the bigger the value, the better the performance is.
 *************************************/

static double evaluatePerformance(String pcycle, String palgorithm, String ptestdata, String presult, String pexpectedre
{
      /*************************************
       * Modify the code to give proper evaluation of algorithm instance's performance.
       *    pcycle is algorithm instance's cycle type.
       *    palgorithm is algorithm instance's algorithm type.
       *    ptestdata is algorithm instance's input test data.
       *    presult is algorithm instance's calculated result for ptestdata.
       *    pexpectedresult is the expected result for ptestdata in the Config.txt.
       *    Reminder: pexpectedvalue might be null.
       *************************************/
      String result [] = presult.split(" ");
      int [] Result = new int[result.length];
      for(int i=0;i<result.length;i++)
            Result[i] = Integer.parseInt(result[i]);

      String expresult [] = pexpectedresult.split(" ");
      int [] expResult = new int[expresult.length];
      for(int i=0;i<expresult.length;i++)
            expResult[i] = Integer.parseInt(expresult[i]);

      double dif = 0;
      for(int j =0; j<result.length;j++){
            dif += Math.pow(Math.abs(expResult[j]-Result[j]),2);
      }

      dif = 0 - Math.sqrt(dif);
      return dif;
}
```

Figure 32: The codes for Verifier in SIS system for the simple example.

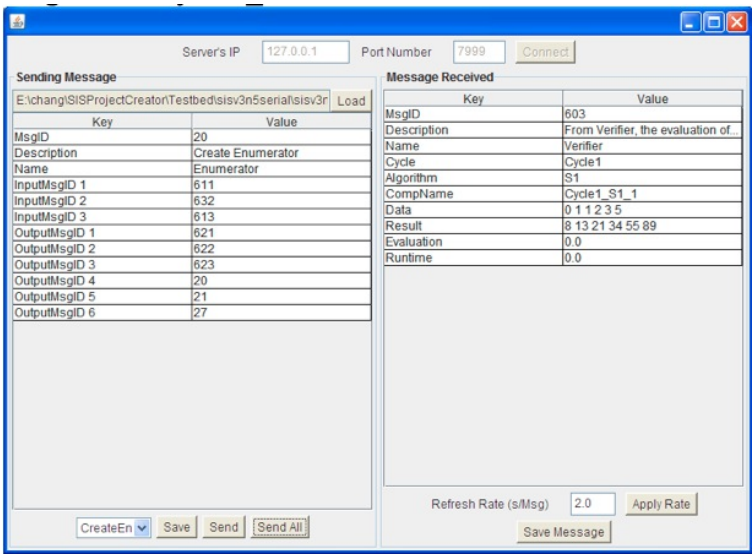


Figure 33: The simulation result of algorithm S1 shown in prjRemote

```
Cycle Cycle1 Algorithm S1 running completed. Best algorithm instance is:
Instance name: Cycle1_S1_1
Parameters: null
Input Test Data: 0 1 1 2 3 5
Output Result: 8 13 21 34 55 89
Result Evaluation: 0.0
Runtime: 0.0
Cycle Cycle1 Algorithm S2 running completed. Best algorithm instance is:
Instance name: Cycle1_S2_1
Parameters: null
Input Test Data: 0 1 1 2 3 5
Output Result: 50 33 57 29 3 36
Result Evaluation: -94.85778829384543
Runtime: 0.0
Cycle Cycle1 Algorithm S3 running completed. Best algorithm instance is:
Instance name: Cycle1_S3_1
Parameters: null
Input Test Data: 0 1 1 2 3 5
Output Result: 6 7 8 9 10 11
Result Evaluation: -94.56743625582752
Runtime: 0.0
Cycle Cycle1 running completed. Best algorithm is:
Algorithm type: S1
Instance name: Cycle1_S1_1
Parameters: null
Input Test Data: 0 1 1 2 3 5
Output Result: 8 13 21 34 55 89
Result Evaluation: 0.0
Runtime: 0.0
```

Figure 34: The simulation results of the entire SIS system.