TOWARDS A GENERAL PLATFORM FOR EFFECTIVELY ANALYZING SOCIAL MEDIA

BY

RUI LI

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Doctoral Committee:

Associate Professor Kevin Chen-Chuan Chang, Chair
Professor Jiawei Han,
Professor ChengXiang Zhai
Associate Professor Lise Getoor, University of Maryland College Park
Abstract

Social media (e.g., Twitter) now become a popular information channel for general users to create and consume information. With many unique advantages, social media provide tremendous opportunities of analyzing what people are talking about in different domains, such as business intelligence (e.g., finding tablets mentioned by users in Christmas) and emergency management (e.g., finding places where users tweet about tornado) and political analysis (e.g., finding occupations of users who support President Obama).

My PhD thesis aims to design a general platform for supporting social media analysis to enable those opportunities. First, I propose a general platform, called BigSocial, which abstracts three essential functionalities, data monitoring, data argumentation and data analysis, for supporting different social media analytic tasks. Then, I study several research problems to realize BigSocial. In the data monitoring layer, I focus on how to efficiently collect relevant data for any given analytic task (e.g., emergency management) from social media (e.g., Twitter), and present the first automatic monitor that continuously collects most relevant tweets for a given task under cost budgets. In the data augmentation layer, I focus on how to accurately and completely profile users’ missing attributes in social media to enable advanced analysis. I begin with exploring how to profile users’ single value attributes (e.g., home location), and develop a probabilistic approach, which accurately profiles Twitter users’ home locations and improves the state of the art methods by 13%. Further, I look into how to profile users’ multiple value attributes (e.g., location, occupation), and design a probabilistic approach, which can discover their multiple locations completely. These studies pave the way for conducting advanced analysis in social media.
Table of Contents

List of Tables .......................................................... v
List of Figures .......................................................... vi

Chapter 1 Introduction ................................................... 1

Chapter 2 Platform ....................................................... 7
  2.1 BigSocial Platform Overview .................................... 7
  2.2 A Prototype System ................................................ 12
  2.3 Research Challenges .............................................. 13

Chapter 3 Social Media Monitoring: An Automatic Topic-focused Monitor for Twitter Stream ........................................... 14
  3.1 Introduction .......................................................... 14
  3.2 Related Work ........................................................ 18
  3.3 ATM Framework Overview ......................................... 20
  3.4 Tweet Sampling Problem ......................................... 25
    3.4.1 Motivation ..................................................... 25
    3.4.2 Tweet Sampling Algorithm .................................. 26
  3.5 Keyword Selection Problem ...................................... 32
    3.5.1 Challenge ...................................................... 32
    3.5.2 Keyword Selection Algorithm ............................... 33
  3.6 Extension: Predicting Usefulness and Costs Accurately .......... 37
  3.7 Experiments ........................................................ 39
    3.7.1 Experiment Setup ............................................. 39
    3.7.2 Experiments for Keyword Selection Algorithm ............... 41
    3.7.3 Experiments for Tweet Sampling Algorithm ................. 46
    3.7.4 Experiments for Prediction Algorithm ......................... 48
    3.7.5 Experiments for ATM framework ............................ 50
    3.7.6 Case Study .................................................. 51

Chapter 4 User Profiling in Social Media: A Probabilistic Approach for Profiling Users’ Home Locations ................................. 53
  4.1 Introduction ........................................................ 53
  4.2 Related Work ........................................................ 57
Chapter 4  Problem Abstraction  ............................................................ 59
  4.4 Discriminative Influence Model  ................................................. 60
    4.4.1 Motivation  ............................................................................. 62
    4.4.2 Model Formulation  ............................................................... 63
  4.5 Location Profiling Algorithms  ..................................................... 66
    4.5.1 Local Prediction Algorithm  .................................................. 66
    4.5.2 Global Prediction Algorithm  ................................................. 70
    4.5.3 Incorporating Constraints  ..................................................... 73
  4.6 Experiments  .................................................................................. 74
    4.6.1 Experiment Setup  ................................................................. 74
    4.6.2 Experiment Results  ............................................................... 76

Chapter 5  User Profiling in Social Media: A Probabilistic Approach for
  Profiling Users' Multiple Locations  ................................................... 80
  5.1 Introduction  ................................................................................. 80
  5.2 Related Work  .............................................................................. 84
  5.3 Problem Abstraction  ................................................................. 85
  5.4 Multiple Location Profiling  ......................................................... 87
    5.4.1 Location-based Generation  .................................................. 88
    5.4.2 Mixture of Observations  ....................................................... 93
    5.4.3 Partially Available Supervision  .............................................. 95
    5.4.4 Generative Model  ................................................................. 97
    5.4.5 Inference with Gibbs Sampling  .............................................. 100
  5.5 Experiments  .................................................................................. 103
    5.5.1 Experiments for Home Location Prediction  ......................... 104
    5.5.2 Experiments for Multiple Location Discovery  ..................... 107
    5.5.3 Experiments for Relationship Explanation  ......................... 110

Chapter 6  Conclusion and Future Agenda  .............................................. 112
  6.1 Conclusion  .................................................................................. 112
  6.2 Future Agenda  ............................................................................. 114

References  ......................................................................................... 117
List of Tables

4.1 Notations for UDI .................................................. 61
4.2 Prediction Results .................................................. 77
4.3 Local vs. Global with 80% Test Users ............................. 78
4.4 Case Studies ....................................................... 79

5.1 Notations ............................................................. 89
5.2 Home Location Prediction Results ................................ 105
5.3 Multiple Location Discovery Results .............................. 108
5.4 Case Study on Multiple Location Discovery ...................... 109
5.5 Case Studies on Relationship Explanation ........................ 111
# List of Figures

2.1 *BigSocial Platform Architecture* ........................................ 8  
2.2 Example of TEDAS ............................................................ 12  

3.1 Overview of ATM ................................................................. 22  
3.2 ATM vs. Baselines for Crime/Disaster .................................... 43  
3.3 ATM vs. Baselines for Crime/Disaster .................................... 43  
3.4 ATM vs. Baselines for Sport ................................................. 44  
3.5 ATM vs. Baselines for Different $f$ ........................................ 44  
3.6 ATM vs. Baselines for Different $M$ ....................................... 45  
3.7 ATM with Different $B$ and $M = 20$ ...................................... 45  
3.8 Efficiency of ATM and Baselines .......................................... 45  
3.9 Efficiency with Different $M$ and $B$ .................................... 46  
3.10 Efficiency on Different Sample Sizes ..................................... 46  
3.11 ATM with Different Sampling Algorithms ............................... 47  
3.12 Efficiency of Sampling Algorithms ....................................... 48  
3.13 Average Prediction Error ................................................... 49  
3.14 Coverage based on Different Prediction Algorithms .................. 49  
3.15 ATM with Different Iteration Lengths .................................... 50  
3.16 ATM vs. Baselines on Twitter Stream .................................... 51  
3.17 Examples of ATM and Baselines ......................................... 51  
3.18 Keyword Changes in Each Iteration ...................................... 51  

4.1 An Example of Twitter Graph ............................................... 59  
4.2 Numbers of Edges versus Distances ...................................... 61  
4.3 Numbers of Relations over the Geo Space ............................... 62  
4.4 Accumulative Accuracy at Various Distance .............................. 76  

5.1 Building Location Profiles for Users ...................................... 81  
5.2 Plate Diagram for MLoc ...................................................... 88  
5.3 Observations ............................................................... 90  
5.4 Accuracy Change in 14 Iterations ....................................... 106  
5.5 Precision and Recalls at Different Ranks ................................. 109  
5.6 Accuracy at Different Miles ................................................. 111
Chapter 1

Introduction

Recently, various social media services, such as Twitter, Facebook, Tumblr and Weibo, become popular information channels for general users to create and consume information online. In particular, Twitter, which we will focus on in this thesis, is the pioneer of such social media services. It now has nearly 140 million active users, who generate 340 million tweets everyday.

Compared to traditional information media (e.g., web pages), Twitter and other social media have several unique advantages.

- Broad Coverage: Tweets are generated by hundreds of millions of general users and cover nearly every aspect of our daily life from national news (e.g., president election), local events (e.g., car theft at 2nd st.), to personal expression (e.g., “I like iPad”).
- Fresh Content: With the wide use of mobile applications, tweets are generated instantly. For instance, we could detect a tweet related to a shooting crime minutes after shots fired, while the first news report appeared approximately 3 hours later.
- Rich Attributes: Tweets not only contain text but also are associated with many additional attributes, such as their timestamps and authors’ attributes (e.g., gender, location).

With these unique advantages, social media become information treasures for analyzing what people are talking about. Specifically, as we will examine below, social media provide tremendous opportunities for building novel analytic applications in various domains.

**Business Intelligence (BI).** As people often tweet about their daily experiences with business (e.g., ATT) and products (e.g., iPad), Social media enable market researchers to survey
their markets in a revolutionary way. On one hand, they can characterize demographics of users who are interested in a product. *E.g.*, what *age groups* of users are likely to talk about iPad. On the other hand, they can find popular products interested by a specific group. *E.g.*, which *tablets* are frequently mentioned by users in *Thanksgiving*?

**Emergency Management (EM).** As people tweet about events happening around them, social media become a timely source for first aid responders to detect and analyze emergency events. For example, Sakaki *et al.* [45] build a system to track earthquake related tweets from Twitter and show that earthquakes could be detected in seconds even faster than the traveling of earthquakes. In our own work [24], which is based on this thesis, we built the system TEDAS for detecting crimes and disasters and analyzing their spatial and temporal patterns. Particularly, our system can answer questions, such as where *Tornado* occur *last summer*, and what *crimes* are mentioned by users living in *New York*?

**Political Analysis (PA).** Similarly, political analysts can track and analyze voters’ opinions based on social media. They can first characterize voters who complain about a candidate from social media. *E.g.*, what are popular *locations* of voters who *tweet* about President Obama. Then, to win those voters, they can further obtain concerns of those voters from social media. *E.g.*, what are the issues frequently mentioned by voters in *Ohio*?. Thus, they can set campaigns in Ohio to address these concerns.

From the above motivating examples, we observe that social media enable many analytic applications (*e.g.*, what *age groups* of users are likely to talk about iPad, or which *tablets* are frequently mentioned by users in *Thanksgiving*), which looks for *fine-grained* information (*e.g.*, tablets mentioned in tweets or ages groups of documents’ authors) *aggregated* from multiple social media documents that satisfying some restrictions (*e.g.*, content about iPad or tweets created during Thanksgiving). Such tasks cannot directly support existing search engines, which is the major information system for accessing a text corpus, as search engines can only find individual documents relevant to keyword queries (*e.g.*, tweets about iPad).
While a few analytic applications (e.g., 45, 48) have been built, they are limited to their own analytic tasks specifically and it is difficult to extend their systems for other analytic applications.

This thesis aims to enable a general platform to support various social media analytic applications (e.g., BI, EM and PA), much like a database management system, which can support various kinds of database applications.

**BigSocial Platform** In Chapter 2, I first present the design of our platform, called BigSocial. It identifies and abstracts the common functionalities required by various social media analytic applications. Particularly, it contains three essential layers. At the bottom, there is a data monitoring layer, which continuously collects relevant social media data (e.g., tweets) from a social media stream (e.g., Twitter) for a given application (e.g., BI, PA, EM), since all applications need to collect relevant data for analysis (e.g., we need to collect crime and disasters related tweets in EM) and relevant data are being created all the time in social media. In the middle, there is a data augmentation layer, which enriches the unstructured data (e.g., relevant tweets) collected by the data monitoring layer into structured records with additional information, since all applications (e.g., BI, PA, EM) all require fine-grained information, such as entities mentioned in tweets (e.g., iPad ∈ tablet) and attributes (e.g., Ohio ∈ location, students ∈ occupation) of authors, for advanced analysis (e.g., what tablets are frequently tweeted by students?). At the top, there is data analysis layer (similar to the query processing modular in database systems), which processes analytical queries (similar to SQL queries) over the structured data. We built an initial prototype system for analyzing crimes and disasters based on the proposed platform [24]. The system clearly illustrates the usefulness of social media analytic applications and proves the feasibility of our proposed platform.

**Research Problems** Then, I study several research challenges motivated by BigSocial (particularly, the data monitoring layer and the data augmentation layer), and present cor-
responding solutions. In this thesis, I choose to focus on Twitter specifically to develop our solutions, as Twitter is one of the most popular social media services. Our solutions could be extended to other social media services (e.g., Tumblr and Weibo) as well.

**Social Media Monitoring** As we just discussed, the data monitoring layer needs to continuously monitor relevant tweets for any given application (e.g., EM) from the Twitter stream. It is a challenging problem for two reasons. First, since the Twitter stream is “big” (i.e., 340 million tweets per day) and “noisy” (e.g., 0.05% tweets are crime and disaster related tweets), it is inefficient (and often impossible) to monitor all the tweets. Second, the available Twitter APIs [52] cannot directly support monitoring tweets for a given application (e.g., BI, EM, PA).

In Chapter 3, I introduce an automatical topic-focus monitor (ATM) [26], which can collect relevant tweets for any given application in an optimal and continuous way. Specifically, the basic idea of ATM is to iteratively “samples” tweets from the stream, and “optimizes” keywords to use based on the samples. To enable ATM, I develop 1) a novel sampling algorithm, which samples sufficient number of random tweets with limited and biased Twitter APIs, 2) an accurate estimation algorithm, which estimates the usefulness of keywords in future based on the samples from the past, and 3) an efficient selection algorithm, which selects the optimal keywords based on their estimated usefulness in linear time. I conduct a large set of experiments to evaluate ATM and the results demonstrate that ATM is effective. E.g., it collects 90% of target tweets for a given application (e.g., EM).

**User Profiling in Social Media** The data augmentation layer needs to profile users’ attributes (e.g., locations, occupations), since only a few users provide their attributes in their online profiles (e.g., only 16% users provide city level locations). Thus, we need to profile users’ attributes from social media. While, in literature, there are some studies about user profiling based on user centric data, such as query-logs [43] and behaviors [55], user profiling in social media is new, because we could leverage not only user-centric data (e.g., their tweets) but also
their social connections. We emphasize that user profiling is not only useful for enabling our platform, but also is valuable for many other tasks, such as personalized search, targeted advertisement. In this thesis, I particularly focus on profiling users’ locations, since the location is a typical and important attribute for many systems.

In Chapter 4, I first introduce a unified discriminative influence model based approach (UDI) \[27\], which can accurately profile users’ single value attributes (i.e., home location) based on their social connections and tweets. Specifically, UDI formally captures how likely a user relates to a signal (e.g., a connected friend or a tweeted location) with respect to 1) the distance between the user and the signal, and 2) the influence scope of the signal. Our experiments on a large scale dataset show that UDI accurately places 67% users and improves the state-of-the-art methods by 13%.

In Chapter 5, I further introduce a multiple location profiling model based approach (MLoc) \[25\], which can completely profile users’ multiple value attribute (i.e., multiple locations). MLoc fundamentally captures that a user has multiple locations and his following relationships and tweeted venues can be related to any of his locations. As a result, MLoc can not only profile users’ home locations accurately but also discover users’ multiple location completely. In addition, MLoc can “explain” each following relationship by revealing users’ true locations in the relationship. Our experiments on a large-scale data set demonstrate those advantages. E.g., for discovering users’ multiple locations, MLoc improves the baseline methods by 16% in recall.

In summary, this thesis makes the following contributions:

- **System:** We propose a general platform BigSocial, which abstracts the essential functionalities for supporting various social media analytic application. A prototype system for analyzing crime and disaster events based on Twitter has been built (demonstrated in ICDE 2012 \[24\]), which not only illustrates the usefulness of social media analytic applications but also proves the feasibility of the proposed platform.

- **Techniques:** We develop several novel techniques to solve the research challenges in
realizing BigSocial. Particularly, 1) we design the ATM framework (published in VLDB 2013 [26]), which automatically monitors relevant tweets for any given topic; 2) I design the UDI approach (published in KDD 2012 [27]), which accurately profiles users’ home locations accurately in the context of social media; and 3) I design the MLoc approach (published in VLDB 2012 [25]), which discovers users’ multiple location completely.

• Evaluations: We collect a large scale of real-world Twitter data and evaluate our techniques through extensive experiments. The results show that our techniques achieve significant performance improvements over existing approaches. The datasets are publicly shared online [23, 22] to stimulate research in the area.

The rest of the thesis is organized as follows. Chapter 2 introduces the BigSocial platform and identifies new research challenges from there. Chapter 3 discusses how to automatically monitor relevant social media data for a given application. Chapter 4 explores how to accurately profile users’ home location in the context of Twitter. Chapter 5 looks into how to completely profile users’ multiple locations in the context of Twitter. Chapter 6 concludes the thesis.
Chapter 2

Platform

This chapter first introduces the BigSocial platform, which abstracts essential functionalities for supporting social media based analytic applications in various domains, and then presents a prototype analytic application built based on BigSocial, which illustrates the feasibility of the proposed platform. From there, we identify some research challenges in realizing BigSocial.

2.1 BigSocial Platform Overview

First, we introduce the BigSocial platform. Inspired by existing information systems, such as database systems and search engines, which support various kinds of applications or queries, we aim to design a general platform to support various social media applications. Particularly, as Fig. 2.1 shows, BigSocial abstracts the common key functionalities shared by various applications in a three layer architecture. As we mentioned, this thesis will be based on Twitter, as it is one of the most popular social media services.

Data Monitoring At the bottom of BigSocial, it is the data monitoring layer, where a monitor continuously collects relevant tweets from the Twitter stream for any given application, since all applications (e.g., BI, EM, PA) need to collect relevant tweets for analysis (e.g., we need to collect crime and disasters related tweets for EM), and tweets are being created by general users all the time.

Particularly, the monitor utilizes Twitter APIs and an application-specific classifier to output relevant tweets of a given application. To enable collecting tweets from the Twitter
stream continuously, Twitter provides three APIs, which represent standard programmatic ways of accessing a document stream. They are 1) the `firehose` API, which returns all tweets and their default metadata, including the timestamp of a tweet and the user of a tweet, 2) the `sample` API, which returns 1% of tweets (and their metadata), and 3) the `filter` API, which returns all the tweets (and their metadata) containing a keyword (e.g., “police”). Thus, our monitor should utilize those standard accessing methods to collect relevant tweets (and their default metadata) for a given analytic application. Here, we observe that, while different applications view relevant data differently, those applications always apply certain classification functions (e.g., simple rule-based classifier or complex model-based classifier) either explicitly or implicitly to automatically determine whether a tweet is relevant or not. Thus, our monitor could utilize the classifier of a given application to output only relevant tweets for the application.

**Data Augmentation** In the middle of BigSocial, it is the data augmentation layer, which enriches “unstructured text” into “structured records”, since different applications (e.g., BI, PA, EM) all require additional fine-grained information for advanced structured analysis.
(e.g., what tablets are tweeted by students, or what crimes are tweeted by users from Chicago during June 2013?).

Particularly, we choose to represent the enriched structured records in relational tables. The relational tables are a popular data model to represent structured records and are widely used in database systems, which support complex analysis with SQL queries. To find what the structured records should look like, we examine the fine-grained information required by different applications (e.g., BI, EM, PA). We find that, while different applications perform different kinds of analysis, they are all interested in two kinds of information: 1) entities inside tweets (e.g., iPad ∈ tablet, where tablet is the type of entities and iPad is an entity instance) and 2) attributes associated with tweets, which could be either the attributes of the tweets (e.g., Nov, 28th, 2013 ∈ timestamp, where timestamp is the attribute, Nov, 28th, 2013 is the value of the timestamp) or the attributes of the tweets’ authors (e.g., students ∈ occupation and Ohio ∈ location). Thus, we use three relational tables to represent the desired information.

- **Document Table**: a document tuple represents a tweet \( t_i \) and consists of a set of default attributes of a tweet, including docID, content, timestamp, and userID.

- **User Table**: a user tuple represents a user \( u_i \) and contains a set of attributes, including default attributes, i.e., userID, and optional attributes, such as gender, occupation, and location.

- **Document-Entity(DE) Table**: a document-entity tuple represents an occurrence of an entity instance of a certain type in a tweet, and consists of instance, type docID, and position of the instance in the tweet.

To enrich raw data (the tweets with their metadata) to relational records, this layer contains several functions. Specifically, it contains 1) a document analyzer, which extracts attributes for a tweet, a name entity recognizer (NER), which recognizes different types of entities from tweets, and a user profiler, which profiles users’ attributes (e.g., location) from
social media.

We note that, while different analytic applications may focus on different entities or attributes, they share the same computation process specified here. Thus, our platform is general to support them. Further, in practice, our platform could provide a list of default tools for an application to use, and further allows application developers to plug in their own tools into our platform. Thus, developers can configure their application easily based on our platform.

**Data Analysis Layer** At the top of BigSocial, it is the data analysis layer, where an analytic engine processes analytic tasks of different applications (e.g., answering what tablets are tweeted by students in BI, or answering what crimes are tweeted by users from Chicago during June 2013 in EM) over the relational records outputted by the previous layer.

To generally support different analytic applications in various domains, we propose to define our *structured analytic query language* via customizing SQL to express different analytic tasks. In database systems, SQL is the widely used query language to generally represent different complex analysis on relational tables. As our platform works a “fixed” relational tables and is text-oriented, we take a special form of `SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ...` with a few new constructs (e.g., for text matching and ranking) to express various kinds of analysis. Regardless of the syntax detail, our example analytical tasks, such as what tablets are tweeted by students in BI, or what crimes are tweeted by users from Chicago during June 2013, can be expressed as the following structured analytic queries.

\[ Q_{mr1} : \text{SELECT instance, count() as popularity FROM User, Document, DE} \]

\[
\begin{align*}
\text{WHERE occupation} & = \text{“students”}, \text{type=tablet,} \\
\text{Document.userID} & = \text{User.userID, DE.docID} = \text{Document.docID} \\
\text{GROUP BY} & \text{ instance ORDER BY} \text{ popularity}
\end{align*}
\]

\[ Q_{mr2} : \text{SELECT instance, count() as popularity FROM user, document, entity} \]
WHERE location=Chicago, timestamp > 2013/06/01, timestamp < 2013/06/31, type=crime, Document.userID = User.userID, DE.docID = Document.docID

GROUP BY instance ORDER BY popularity;

Here, we make two notes about our structured analytic queries. First, as social media documents are text, we add a text matching function (i.e., content relevant to “battery life”) in the WHERE clause to allow users to focus on relevant documents about certain content. Second, to generally support advanced models for analyzing social media, our platform can support different aggregation functions (e.g., count()) to aggregated and rank results. For example, a simple function simply counts how many occurrences match the WHERE clause for each entity instance, while a complex function incorporates implicit features (e.g., the relevance between documents and keywords) to aggregate occurrences of each entity instance. Users can plug in their aggregation functions as well.

Via representing different analytic tasks as structured analytic queries, our analytic engine can perform an analytic task easily like a database engine answering a SQL query. Particularly, it takes a structured analytic query as input, parses the queries into operations, executes those operations over relation tables, and outputs the results to users.

Summary BigSocial abstracts and supports the essential functionalities shared by different analytic applications. Based on BigSocial, users can easily build analytic applications in various domains. Particularly, users can configure an application via some easy steps. First, users can plug in a classification function for the data monitoring layer to collect relevant tweets. Then, users choose the interested entities (e.g., crime, disaster) and attributes (e.g., location, and timestamp) for the data augmentation layer. Now, our platform supports the analytic application, which continuously collect relevant data and enrich them with desired information. To conduct online analysis, users can formulate structured analytical queries to perform their analytic tasks, and our platform processes those queries and outputs the corresponding analysis results.
2.2 A Prototype System

To explore the feasibility of our proposed BigSocial platform, we built an analytic application based on BigSocial in the emergence management domain. Specifically, the application is called Twitter based Event Detection and Analysis System (TEDAS), which aims to analyze spatial and temporal patterns about crime and disaster related events (CDE) based on Twitter. It mainly supports two types of analysis: 1) finding popular locations and time periods about a specific CDE (e.g., “tornado”) and 2) finding popular CDEs for a location during a time period. The system was demonstrated in the ICDE 2012 conference [23].

Fig 2.2 shows an example results of TEDAS for analyzing popular locations where people tweet about a specific disaster “torndao” during a time period (from 2011/06/01 to 2011/07/14) and popular time periods when people tweet about “tornado” during the time period. The results are quite meaningful. The popular time periods clearly indicate that tornados happened on June 21, 2011, while the popular locations tell major tornado regions, such as Kentucky and Missouri.

Overall, TEDAS, as a specific type of analytic application based BigSocial, reveals tremendous opportunities for performing analysis based on social media, and also proves the feasibility of our proposed BigSocial platform.
2.3 Research Challenges

While TEDAS shows the promise of our BigSocial platform for analyzing social media, the experiences in implementing TEDAS helps us to identify two major research challenges for realizing BigSocial.

- **Social Media Monitoring** When implementing the monitor for TEDAS, we find that available Twitter APIs cannot support our task directly. First, it often impossible to use the firehose API to monitor, as it often requires a specific permission to use. Even if the API is available, it is inefficient and useless to monitor all the tweets, as the Twitter stream is “big” (*i.e.*, 340 million tweets per day) and “noisy” (*e.g.*, 0.05% tweets are crime and disaster related tweets). Second, the sample API, which only returns 1% of relevant tweets, is insufficient for many application. Third, the filter API, which returns all the tweets containing a keyword (*e.g.*, “police”), will miss many target tweets that do not contain the keyword. Thus, our BigSocial platform calls for a social monitor to continuously collect most relevant tweets for any given application.

- **User Profiling** When implement the profiler for TEDAS, we find that many users’ attributes are missing their online profiles (*e.g.*, only 16% users provide city level locations). Thus, we need to profile users’ attributes from social media. While, in literature, there are studies about user profiling based on user centric data, such as query-logs [43] and behaviors [55], user profiling in social media is new, because we could leverage not only user-centric data (*e.g.*, their tweets) to infer their attributes, but also their social connections to propagate attribute values among friends. This problem is not easy, as information on social media is extremely noisy (*e.g.*, users tweets different locations in their tweets and connects to friends who have different locations).

In the rest of the thesis, I will focus on solving these two research challenges.
Chapter 3

Social Media Monitoring: An Automatic Topic-focused Monitor for Twitter Stream

3.1 Introduction

In this chapter, we focus on designing a social media monitor, which can continuously collect relevant tweets from the Twitter stream for any given application, for the data monitoring layer of our BigSocial platform.

As we motivated in Chapter 1, social media enable many novel applications [13, 24, 48] in various domains (e.g., BI, EM, PA). While those applications conduct different kinds of analysis, they all need to collect relevant tweets from the Twitter stream for analysis. Particularly, those applications usually start with collecting potentially relevant tweets for an application (topic) (e.g., crime), apply a classifier $f$ to automatically determine whether a collected tweet is indeed relevant to the topic, and process the tweets that pass $f$ for application-specific analysis. Thus, we abstract the problem as monitoring target tweets from the Twitter stream with respect to a given classifier $f$. Everyday, while hundreds of millions of tweets are generated, only a small percentage of them may pass the classifier $f$ as target tweets for an application. It is difficult to collect most of them effectively.

To effectively support monitoring target tweets for any applications (e.g., EM, BI, PA), we aim to develop a social data monitor, which allows users to plug in any classifier $f$ as input and automatically collects target tweets for $f$ from the Twitter stream as output. Ideally, the platform should meet the following requirements.

- **Optimal**: it should collect, with optimal or near-optimal guarantees, as many target tweets as possible under given computation resources, since many applications need a
comprehensive coverage of target tweets to perform accurate analysis (e.g., reporting a car theft at 2nd street). As target tweets are sparsely scattered, they are hard to catch comprehensively.

- **Continuous**: it should collect target tweets from the Twitter stream *continuously*, since new tweets are being created all the time. As target tweets with new content (e.g., “Boston bomb”) may arise, it is challenging to capture the dynamics of the Twitter stream.

While the two requirements are crucial for all applications, current solutions of monitoring target tweets for a given classifier \( f (\text{topic}) \) cannot satisfy them.

Twitter provides a set of APIs [52], which represent standard programmatic ways of monitoring a document stream, but none of them can directly be used for monitoring target tweets for a topic. The *filter* API, which returns all the tweets containing a given keyword, will miss many target tweets that do not contain the keyword. The *sample* API, which returns 1% of all tweets (and thus 1% of target tweets), is insufficient for many applications mentioned above. The *firehose* API returns all tweets but requires a specific permission to use. Even if it is open to access, as it requires prohibitive processing costs (e.g., classifying all tweets), it is inefficient to use.

Given Twitter APIs, since target tweets for a topic (e.g., crime) may share some relevant keywords (e.g., “shoot”), many existing applications [15, 45] use the *filter* API with a set of manually selected keywords (e.g., \{“shoot”, “kill”\}). However, this manual approach has severe disadvantages. First, it is *laborious*, as it requires extensive human efforts to select keywords for each topic. Second, the selected keywords have *no guarantee* of optimality. People might miss useful keywords (e.g., “police”) and thus many target tweets. Third, the keywords may quickly become *outdated* as time goes by, since new target tweets, which have different contents from previous ones (e.g., “Boston bomb”), are emerging and will be missed by them.

**ATM Framework** Thus, to monitor target tweets for a given classifier (topic), we have to address how to enable *automatically* selecting “optimal” and “continuous” keywords.
As our first contribution, in Sec. 3.3, we design the Automatic Topic-focused Monitor (ATM) framework. Our basic intuition is that the “past” can predict the “near future”. Particularly, to select optimal and continuous keywords, we can estimate the “usefulness” of any set of keywords based on recent samples from the Twitter stream and select the most “useful” set to monitor. Thus, ATM takes a sampling, optimizing and tracking approach to monitor target tweets iteratively. In an iteration, ATM 1) samples tweets from the stream to enable estimation, 2) optimizes keywords to use based on their estimated coverage of target tweets, and 3) tracks target tweets with the selected keywords. To monitor target tweets continuously, ATM repeats the procedure in iterations (i.e., it tracks a new set of keywords every iteration). Since within a short iteration contents of tweets are similar and ATM selects keywords based on their coverage, the keywords are optimal. Further, in every iteration, ATM updates keywords according to recent samples from the Twitter stream, so the keywords are continuous.

**Tweet Sampling** To realize ATM, we need to sample a sufficient number of random tweets from the Twitter stream in each iteration for accurate estimation. It is challenging because the available APIs are limited or biased. As we will prove in Sec. 3.4, the “accuracy” of the estimated coverage of keywords is inversely related to the sample size, so the sample API, which only returns 1% of tweets in an iteration, is limited. Further, directly using the filter API with keywords is biased to the tweets containing the keywords.

As our second contribution, in Sec. 3.4, we develop a random sampling algorithm to collect a sufficient number of random tweets with the limited and biased APIs. To sample additional tweets, we effectively combine the available APIs to conduct random walk sampling on a carefully designed “tweet graph”. The method samples tweets according to a trial distribution defined by the graph. Further, we utilize rejection sampling to adjust the tweets from the trial distribution to the uniform distribution. As the main merit of our algorithm, we carefully design a tweet graph, which connects tweets with appropriate weights, to make sure that the random walk sampling 1) can be easily realized with the available APIs and
2) theoretically converges to a known trial distribution over tweets.

**Keyword Selection** To realize ATM, we further need to optimize keywords to use for a given classifier. When optimizing keywords, we have to consider “filtering costs” at Twitter and “post-processing costs” in ATM, since each selected keyword (or collected tweet) takes filtering (or post-processing) costs and both computation resources are limited. We formally model our problem as selecting a set of keywords that have the maximum coverage of target tweets under two cost constraints, 1) cardinality constraint, which limits the number of selected keywords below a threshold \( M \), and 2) budget constraint, which limits the total number of collected tweets below a budget \( B \). The problem is challenging, as we have to select keywords efficiently. As we will prove in Sec. 3.5, the problem is *NP-hard*, which prohibits finding the optimal solution in real time.

As our third contribution, in Sec. 3.5, we develop a keyword selection algorithm, which finds a near-optimal solution in polynomial time. Towards developing our algorithm, we observe and prove that our problem possesses a desirable “submodular” property. While optimizing a submodular function with one constraint is well known, our problem has two constraints and needs a new solution. Based on the fact that maximizing a submodular function with one constraint can be approximated with a greedy algorithm, we develop a new greedy algorithm for our problem, which first relaxes our problem to a simple problem with one constraint (i.e., budget) and then handles the other constraint (i.e., cardinality). We also give its approximation rate.

**Usefulness and Costs Prediction** Our keyword selection algorithm aims to select keywords that are optimal for the “near future” of the Twitter stream and the tweet sampling algorithm collects sufficient samples of the “current” of the recent stream. While it is intuitive to assume that the “current” samples are sufficiently good to estimate optimal keywords for the near future, they may not be the best, especially for some keywords (e.g., “traffic jam”) that have periodic patterns (e.g., rush hours of everyday).
As our forth contribution, in Sec. 3.6, we future extend our basic ATM framework using a machine learning approach. Specifically, we develop a machine learning based prediction algorithm, which learns models to predict the usefulness (and the post-processing costs) of a set of keywords $K'$ in near future based on not only the “current” samples but also “past” samples. Thus, the usefulness or post-processing costs can be more accurately predicted than using only the current samples. We note that, while machine learning approaches often require substantial training data to learn accurate models, our algorithm is fully automatic. It does not require any manual labels, since we can always use the past samples as labeled data to train our models.

**Experiments** As our fifth contribution, we implement our ATM based on the three algorithms and evaluate it with extensive experiments in Sec. 3.7. Our experiments show the following results. First, our sampling algorithm collects a large number of additional random tweets. Second, our selection algorithm 1) is effective, which collects 84% of target tweets for a topic with only 20 keywords and improves the best baseline by 19%, 2) is efficient, and 3) works for various topics and constraints. Third, our prediction algorithm can predict keywords usefulness and post-processing cost accurately. Forth, as an integrated framework, ATM greatly improves a manual (and static) approach by 49%.

### 3.2 Related Work

In this section, we discuss our related work. Our work is related to crawling web pages, monitoring social media, and retrieving relevant documents with keywords.

Web page crawling has been a fundamental task since the beginning of the Web. Many studies, which focus on different issues, have been done. A good survey of them can be found in [37]. Among them, Chakrabarti et al. [13] propose the concept of focused crawling, which crawls pages relevant to a predefined topic. Specifically, before crawling a URL, a topic-focused crawler analyzes the URL’s context and its link structure to determine whether it
is relevant. Our task is different, as we monitor tweets for a topic with keywords instead of
crawling pages via hyperlinks.

Social media monitoring becomes an important task due to the emergence of social media
based applications. While most applications [13, 18, 51] monitor data based on a manual
approach, some automatic monitors have been proposed. Hurst and Maykov [17] propose
an architecture for monitoring general blogs, but they select blog feeds using simple rules
(e.g., how often a new blog is posted) without considering topics. Our problem is different,
as we design a topic-focused monitor. Boanjak et al. [12] propose a focused crawler, which
crawls topic related tweets from heuristically selected users (i.e., the user who has the most
connections to the existing ones). However, heuristics based methods have two limitations:
1) they select users/keywords heuristically without any performance guarantees, and 2) they
do not consider cost constraints.

Selecting keywords to retrieve relevant documents have been studied [44, 18, 1, 15] in
other settings (instead of monitoring social media). For example, Robertson and Jones [44]
design a weighting function to find keywords to retrieve additional relevant documents for
a query. Agichtein and Gravano [1] utilize the function in [44] and other rules to find the
keyword queries that retrieve only the relevant documents for information extraction. As
these methods [44, 18, 1, 15] focus on different settings, they are not to find the keywords
that have the maximum coverage of target tweets for a topic under two cost constraints.
Thus, they 1) neither have guaranteed performance for our problem (e.g., keywords selected
in [1, 44] are too specific to cover many target tweets), 2) nor consider cost constraints (e.g.,
they [15] do not limit the total number of collected documents). Further, they do not address
how to sample sufficient tweets from the Twitter stream to update keywords iteratively.

Our ATM framework advances the above methods [12, 44, 18, 15] from two aspects.
First, ATM selects keywords in a constrained optimization approach, which 1) finds near-
optimal keywords with guarantees and 2) considers two types of costs. Second, ATM updates
keywords in iterations, which monitors the dynamic Twitter stream continuously. To en-
able updating keywords, we design a sampling algorithm to sample random tweets from the stream, and introduce a machine learning approach to predict usefulness and costs of keywords in near future.

3.3 ATM Framework Overview

In this section, we propose our ATM framework.

Twitter APIs To begin with, we introduce three Twitter APIs for monitoring public tweets from the Twitter stream. They represent three standard programmatic ways of accessing a corpus. Details of them can be found in [52].

- **Sample** returns a set of random samples (approximately 1%) of all public tweets.
- **Filter** returns the public tweets that match given filter predicates (e.g., a keyword “police”).
- **Firehose** returns all public tweets.

Given the three APIs, we choose the filter API to monitor target tweets for a topic for two reasons. First, we cannot use the sample API or the firehose API. The sample API only gives 1% of target tweets, which are not enough for many applications (e.g., detecting local crimes or predicting stock prices based on redundancies). We note that the sample API returns the same samples even if we call it from different machines. The firehose API requires a permission to use and is not available to general users. Even if it is available, we should not use it, as it requires prohibitive processing costs (e.g., processing all the tweets). Second, it is possible to collect most target tweets for a topic (e.g., crime) using the filter API with well selected keywords, since, intuitively, different target tweets may share similar keywords (e.g., “shoot”). We note that Twitter has other APIs, but they are not for monitoring public tweets. E.g., the user API requires a user’s authentication, and returns the tweets from his friends.
**ATM Overview** Based on Twitter APIs, we propose ATM to effectively monitor the Twitter stream for any given classifier (topic). As Fig. 3 illustrates, ATM generally takes any classifier \( f \) as input and outputs target tweets for \( f \) under certain cost constraints. Specifically, given a classifier \( f \) (e.g., crime), to collect its targets tweets from the Twitter stream in an optimal and continuous way, ATM iteratively selects optimal keywords to track.

At the \( i^{th} \) iteration, to monitor target tweets with “optimal” keywords, ATM works in three steps. First, to enable estimating the usefulness of any set of keywords, a *sampler* collects a large amount of tweets \( S_i \) (e.g., \( t_1: \) “police detained 17 people...”, \( t_2: \) “car theft...”, \( t_3: \) “enjoy my tea...”) from the stream. Then, given the samples \( S_i \) and the classifier \( f \), a *selector* selects a set of keywords \( K_i^* \) (e.g., \{“police”, “theft”\}) that have the maximum coverage of target tweets in \( S_i \) (e.g., \( t_1, t_2 \)) under cost constraints. Finally, a *tracker* calls the *filter* API with \( K_i^* \) to collect target tweets (e.g., \( t: \) “police arrested...”) from the stream for this iteration.

To monitor target tweets with new content “continuously”, ATM updates keywords iteratively. While the tracker monitors target tweets with keywords \( K_i^* \) for the \( i^{th} \) iteration, the *sampler* collects new samples \( S_{i+1} \) (e.g., \( t_1: \) “bomb in Marathon...”, \( t_2: \) “FBI came...”, \( t_3: \) “good food...”) for the \((i+1)^{th}\) iteration. When the \( i^{th} \) iteration finishes, the *selector* selects a new set of keywords \( K_{i+1}^* \) (\{“bomb”, “FBI”\}) based on \( S_{i+1} \), and the *tracker* uses \( K_{i+1}^* \) to collect target tweets for this new iteration.

Here, we explain that it is reasonable to take a classifier \( f \) as input. As we motivated in Sec. 3.1, our goal is to generally support monitoring target tweets for various social media based applications (e.g., EM, BI, PA), which have already used classifiers [48, 53] to automatically determine their target tweets. Thus, our framework only leverages the existing classifiers in those applications and does not add any extra burden. Further, while our focus is not the scenarios where classifiers do not already exist, it is possible to train classifiers and use ATM for these scenarios, since many classifiers have been studied in general or for Twitter [53, 40], and can accurately predict target tweets for a topic with advanced models and
novel features.

We further emphasize that the iteration length $l$ should be carefully set in ATM. For any topic, $l$ could not be too short or too long. On the one hand, $l$ cannot be too long (e.g., a day) since target tweets with new content may emerge and need to be captured with new keywords. On the other hand, $l$ cannot be too short (e.g., 5 mins), since we may not collect enough tweets in a short iteration to accurately estimate the usefulness of keywords. Further, since different topics require different numbers of samples for accurate estimation (e.g., a sparse topic like crime needs many samples) and their target tweets change at different rates (e.g., tweets about olympics news develop very fast), $l$ should be different for different topics. Thus, for a topic, we treat $l$ as an important parameter to tune. As ATM works for any given $l$, we can find a reasonable $l$ for a topic via testing the performance of ATM with different $l$.

ATM meets our requirements in Sec. 3.1. First, it is guaranteed to use optimal or near-optimal keywords, since we can intuitively assume that, within a short iteration, target tweets are similar to those in samples, and it selects keywords based on their usefulness on the samples. Second, it can continuously monitor target tweets, since every iteration it uses new keywords based on the most recent samples.
To realize ATM, in each iteration (i.e., a short time period), we have to 1) sample a sufficient amount of random tweets, which may be more than the samples returned by the sample API (i.e., 1% of tweets in an iteration) for accurate estimation, and 2) efficiently find the optimal keywords to use under cost constraints based on the samples.

Here, we treat them as two independent problems, tweet sampling and keyword selection, for two reasons. First, a separate sampler is “topic-independent” and can collect samples for serving different topics (e.g., crime or politic). Second, each problem is meaningful by itself with many applications. The solution for tweet sampling can be used as a general crawler to collect sufficient random tweets, as many applications (e.g., estimating prosperities of the Twitter stream in a day) require collecting more than 1% of tweets. The solution for keyword selection can also be applied to other scenarios (e.g., selecting experts for a community). We note that, for Twitter, which has access to all the tweets, the first problem might be easy, but how to solve the second problem is unclear. Further, as most social media based applications \cite{45, 48} only have access to the filter and sample APIs, both problems are challenging.

**Problem Abstraction** Next, we formally define the two problems. To begin with, we introduce some notations. We use 1) \( t \) as a tweet, 2) \( k \) as a keyword, 3) \( T \) as the set of all the tweets in an iteration, and 4) \( K \) as the set of all the keywords that can be used as filters. A keyword \( k \in K \) can be any single term (e.g., “police”). To cover all useful keywords, \( K \) should be complete (i.e., it covers all the keywords in \( T \)). We can construct \( K \) via enumerating all the unigrams in \( T \). We use \( K' \) to denote a subset of \( K \). We define the *match* of a tweet \( t \), denoted as \( M(t) \), as the set of keywords that \( t \) contains, and the *volume* of a keyword \( k \), denoted as \( V(k) \), as the set of the tweets containing \( k \).

First, we abstract the *tweet sampling* problem. We represent a set of samples of \( T \) as \( S \). To make unbiased estimation, \( S \) should be *uniformly* sampled from \( T \), which means that \( \forall t_i, t_j \in T \), the probability of \( t_i \) in \( S \), denoted as \( P(t_i \in S) \), is the same as \( P(t_j \in S) \). To make accurate estimation, \( S \) should contain a *sufficient* number of samples, which means
\(|S|\) should be larger than a threshold \(\gamma\). Thus, we formally state the tweet sampling problem as follows.

**Tweets Sampling Problem** Let \(T\) be all the tweets in an iteration. Given a threshold \(\gamma\), which is smaller than \(|T|\), the filter API, and the sample API, output a set of samples \(S \subset T\), s.t. \(\forall t_i, t_j \in T, P(t_i \in S) = P(t_j \in S)\) and \(|S| > \gamma\).

Next, we abstract the keyword selection problem. We denote the given classifier for a topic as a binary function \(f\). Given a tweet \(t\), \(f\) outputs 1 if \(t\) is revelent to the topic, and 0 otherwise. If \(f(t) = 1\), we call \(t\) a target tweet, and use \(R\) to represent all target tweets, \(R = \{t|f(t) = 1, t \in T\}\). Based on \(f\), we quantify the “usefulness” of keywords as follows. We define the cover of a keyword \(k\), denoted as \(C(k)\), as the set of the target tweets containing \(k\), \(C(k) = \{t_j|t_j \in V(k) \cap R\}\), and measure the usefulness of a set of keywords \(K' \subset K\), denoted as \(U(K')\), as the number of the target tweets covered by \(K'\), \(\sum_{k_i \in K'} |C(k_i)|\).

In this chapter, we use “usefulness” and “coverage” interchangeably. Further, we formally model two constraints.

- **Cardinality constraint** limits filtering costs of a solution \(K'\). It takes costs to filter incoming tweets for each keyword, but such computation resources are limited. E.g., the filter API only accepts up to 400 keywords as filters. Thus, we use \(K'\)’s cardinality \(|K'|\) to model its filtering costs, and limit \(|K'|\) below a threshold \(M\).

- **Budget constraint** limits post-processing costs of a solution \(K'\). It takes costs to process each collected tweet, but such computation resources are limited. We use the number of the tweets collected by \(K'\) to model its post-processing costs, denoted as \(P(K')\), and limit \(P(K')\) below a budget \(B\). \(P(K') = \sum_{k_i \in K'} |V(k_i)|\). We measure \(P(K')\) as the sum of the volumes of keywords in \(K'\) without considering that a tweet can be covered by multiple keywords, since each keyword filter is applied individually and we suffer from processing such redundancies.

Now, we formally abstract the keyword selection problem.
**Keywords Selection Problem.** Given a classifier \( f \), a set of tweets \( T \), a set of candidate keywords \( K \), a threshold \( M \) and a budget \( B \), output \( K' \subset K \), s.t. \( U(K') \) is maximized subject to \( |K'| \leq M \) and \( P(K') \leq B \).

### 3.4 Tweet Sampling Problem

We first focus on the tweet sampling problem. We aim to collect a sufficient number of random samples \( S \) from all the tweets \( T \) in an iteration with the available Twitter APIs (i.e., the filter and sample APIs) for estimating the usefulness \( U(K') \) and the post-processing cost \( P(K') \) for any set of keywords \( K' \).

#### 3.4.1 Motivation

First, we motivate the need of a sampling algorithm besides the sample API, which returns 1% of tweets. As we discussed in Sec. 3.3, to capture the dynamics of the Twitter stream, especially for fast developing topics (e.g., olympic news), ATM prefers a short iteration. Further, as we will show below, to enable accurate estimation, ATM needs a sufficient number of samples, which may be more than 1% of tweets in an iteration. Thus, it is desirable to have a sampling algorithm, which provides additional samples besides the sample API, to enable collecting sufficient samples in a short iteration or to speed up the sample API for capturing the dynamics of the stream.

Next, we develop a theorem to formally relate the estimation accuracy and the sample size. We focus on estimating \( U(K') \) for a set of keywords \( K' \), but our discussion can be applied to \( P(K') \). We denote the estimated value in \( S \) as \( \hat{U}(K') \) to differentiate it from the true value \( U(K') \) in \( T \).

We first show our intuition for the theorem. Here, we take a simple but realistic assumption. While the sample API samples tweets with replacement, we assume it samples without replacement, since \( T \) is very large and the chance of getting the same sample is negligible.
Intuitively, as $U(K')$ measures the number of the tweets that 1) are target tweets and 2) match any keyword $k \in K'$ in $T$. A random tweet from $T$ has a probability $U(K')/|T|$ to meet the two requirements. Since a set of random samples $S$ can be viewed as drawing tweets repeatedly for $|S|$ times, we can view $S$ as a Bernoulli Process with a success probability $U(K')/|T|$, and $\tilde{U}(K')$ as the number of successes in $|S|$ independent Bernoulli trials, which follows the binomial distribution with a success probability $U(K')/|T|$. Thus, our task becomes how accurately we can estimate the parameter $|U(K')|/|T|$ of the binomial distribution with $\tilde{U}(K')$ successes observed from $|S|$ samples. We directly obtain our theorem from existing results about the parameter estimation for the binomial distribution in statistics [56].

**Theorem 3.4.1.** Given random samples $S$ from the set $T$ and an error percentile $\alpha$, with $1 - \alpha$ confidence, $|U(K')/|T||$ is within $\tilde{U}(K')/|S|$ ± $z_{1-\alpha/2} \sqrt{\tilde{U}(K')/|S| - (\tilde{U}(K')/|S|)^2}$, where $z_{1-\alpha/2}$ is the $1 - \alpha/2$ percentile of the standard normal distribution.

The theorem is useful from several aspects. 1) It shows that, given a confidence level (e.g., 95%), we should increase the sample size $|S|$ to make our estimation $\tilde{U}(K')/|S|$ close to the true value $U(K')/|T|$. 2) It gives a formula to calculate the necessary number of samples for achieving a certain accuracy. 3) It shows that the required numbers of samples are different for different topics, since $U(K')$ is different.

### 3.4.2 Tweet Sampling Algorithm

Now, we develop our sampling algorithm with the available Twitter APIs. Since the sample API may not provide enough samples, we need to use the filter API. However, directly using it with a set of keywords is biased to the tweets containing those keywords. Thus, it is challenging to collect additional unbiased (or uniform) samples. Here, we clarify that we aim to collect additional samples besides those returned by the sample API instead of replacing them.
We develop our sampling algorithm based on a widely used sampling framework, which uniformly samples nodes from a graph via integrating two sampling methods, random walk sampling and rejection sampling. In the literature, specific algorithms have been developed based on the framework to sample pages from the Web graph [1] or users from a social network graph [19]. In this chapter, we adopt the framework to develop a new algorithm for sampling tweets with the available Twitter APIs. It is possible, because we can connect tweets as a “tweet graph” through the APIs. However, we cannot apply the existing algorithms, because they sample from different graphs (e.g., a social network graph) with different access functions (e.g., getting friends of a user). We must design our own “tweet graph” and sample with the available APIs.

**Preliminary** We first briefly describe random walk sampling and rejection sampling methods.

*Random Walk* on a graph $G(N, E)$, where $N$ denotes a set of nodes and $E$ denotes a set of weighed edges, is a markov chain on a finite state space $N$ [30]. It can be described as a surfer randomly walking among $G$. At a node $n_i$, the surfer visits a neighbor node $n_j$ randomly according to the weight of their edge $e_{ij}$. After several steps, the surfer reaches different nodes with different probabilities. In theory, if $G$ is “ergodic”, the probabilities of visiting different nodes are guaranteed to converge to a distribution $\phi$ over nodes $N$. $G$ is *ergodic*, if 1) $G$ is strongly connected, and 2) the greatest common denominator of all cycle lengths is 1. Thus, random walk sampling works in two steps. 1) It randomly walks for several steps, which are called as the *burning period*. 2) It generates the next visited node as a sample. If $G$ is ergodic, the samples are generated according to the distribution $\phi$ defined by $G$, which we call a *trial distribution*.

*Rejection Sampling* is a simulation method for generating samples according to a target distribution $\pi$ with samples generated from a trial distribution $\phi$. Intuitively, it uses “acceptance probabilities” to bridge the gap between $\phi$ and $\pi$. *E.g.*, when $\pi$ is the uniform
distribution and $\phi$ is another distribution, it assigns high acceptance probabilities to instances that have low probabilities in $\phi$. As rejection sampling only cares the relativity of $\phi$ and $\pi$, it is defined based on their un-normalized forms. We define an un-normalized form of a distribution $\pi$, denoted as $\hat{\pi}$, if $\exists Z_\pi$, s.t. $\forall n \in N$, $\hat{\pi}(n) = \pi(n) \times Z_\pi$. Given an un-normalized trial distribution $\hat{\phi}$ and an un-normalized target distribution $\hat{\pi}$ over the space $N$, the acceptance probability of an instance $n$ is defined as $\hat{\pi}(n)/(C\hat{\phi}(n))$, where $C$ is a constant that satisfies $C \geq \max_{n \in N} \hat{\pi}(n)/\hat{\phi}(n)$.

**Algorithm 1** TweetSample()

\[
\text{while true do}
\quad t = \text{RandomWalk}_\phi();
\quad \text{toss a coin with head probability } \frac{\hat{\pi}(t)}{C\hat{\phi}(t)};
\quad \text{If head return } t
\quad \text{end while}
\]

**Sampling Algorithm** Alg. 1 shows our sampling algorithm based on the framework. It first calls RandomWalk$_\phi$ to get a sample $t$. This function utilizes the available APIs to conduct random walk sampling on a “tweet graph”. We denote the tweet graph as $G(T,E)$, where the nodes are tweets $T$ and they are connected by weighed edges $E$. We will define $G$ and describe RandomWalk$_\phi$ in detail later. As $t$ follows the trial distribution $\phi$ defined by $G$ instead of the uniform distribution $\pi$, Alg. 1 then applies rejection sampling to decide whether $t$ is accepted with the acceptance probability $\hat{\pi}(t)/(C\hat{\phi}(t))$. As $\pi$ is the uniform distribution, $\hat{\pi}(t) = 1$. We show $\hat{\phi}(t)$ and $C$ after we define $G$.

**Challenges** To complete Alg. 1, we need to design $G(T,E)$, which connects tweets $T$ with weighed edges $E$. It is not easy, as $G$ has to meet two requirements.

- **Feasible**: We can realize random walk from $t_i$ to $t_j$ according to their edge weight $e_{ij}$ with the available APIs.
- **Ergodic**: $G$ must be ergodic so that random walk on $G$ converges to a unique probability distribution $\phi$. 
**Tweet Graph** As the key merit of our algorithm, we design a tweet graph $G$, which meets the two requirements. We clarify that our algorithm only needs to conduct random walk from a tweet to another tweet according to their weight in $G$ and does not need to build a complete $G$ explicitly. We use $P(t_i \rightarrow t_j)$ to denote the probability of walking from $t_i$ to $t_j$. According to random walk sampling, $P(t_i \rightarrow t_j) = \frac{e_{ij}}{D(t_i)}$, where $D(t_i)$ is the degree of $t_i$, $D(t_i) = \sum_{t_j \in T} e_{ij}$.

To make $G$ feasible, we use the *filter* API to randomly “walk” from $t_i$ to $t_j$. As the *filter* API uses a keyword to retrieve tweets, we can implement walking in two steps. First, we randomly pick a keyword $k$ from the set of keywords in $t_i$, which is $M(t_i)$. Second, we use the *filter* API with $k$ to get a random tweet $t_j$ from the set of the tweets containing $k$, which is $V(k)$. In this way, the probability of walking from $t_i$ to $t_j$ through a keyword $k$ is $\frac{1}{|M(t_i)|} \sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}$. For different $t_j$, $P(t_i \rightarrow t_j)$ is proportional to $\frac{1}{|M(t_i)|} \sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}$, as $|M(t_i)|$ is a constant at a specific $t_i$. According to the definition, $P(t_i \rightarrow t_j)$ is proportional to $e_{ij}$, so we directly set $e_{ij}$ as $\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}$.

However, $G$ with $e_{ij}$ defined above may not be ergodic, as $G$ may not be strongly connected.

To make $G$ ergodic, we add a small teleport weight to the edge of any pair of tweets. Thus, at a tweet $t_i$, we can “jump” to any tweet $t_j$ with a small probability. As any pair of tweets is connected, $G$ is ergodic. Specifically, we add a total weight $\lambda$ for jumping from $t_i$ to all the tweets in $T$, and a weight $\frac{\lambda}{|T|}$ for jumping from $t_i$ to $t_j$. Thus, we adjust $e_{ij}$ as $(\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}) + \frac{\lambda}{|T|}$. To implement jumping from $t_i$ to $t_j$, we use a sample returned by the *sample* API, as we can view the *sample* API as a uniform sampler, which returns a tweet $t_j$ with $\frac{1}{|T|}$ but can only be used for a limited number of times (i.e., $|T|/100$). Thus, $G$ is feasible.

According to the new weight, we need to determine how likely we do “walking” and “jumping” at a tweet $t_i$. We first calculate the new $D(t_i)$ based on the new weight $e_{ij}$, and
then derive \( P(t_i \rightarrow t_j) \) based on \( \frac{e_{ij}}{D(t_i)} \).

\[
D(t_i) = \sum_{t_j \in T} \left( \sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|} \right) + \frac{\lambda}{|T|} = |M(t_i)| + \lambda \tag{3.1}
\]

\[
P(t_i \rightarrow t_j) = \frac{|M(t_i)|}{|M(t_i)| + \lambda} \left( \sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)||M(t_i)|} \right) + \frac{\lambda}{|M(t_i)| + \lambda} \frac{1}{|T|} \tag{3.2}
\]

We can interpret \( P(t_i \rightarrow t_j) \) as a combination of “walking” \( (\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)||M(t_i)|}) \) based on the filter API with a probability \( \frac{|M(t_i)|}{|M(t_i)| + \lambda} \) and “jumping” \( (\frac{1}{|T|}) \) based on the sample API with a probability \( \frac{\lambda}{|M(t_i)| + \lambda} \). \( \lambda \) works as a parameter for choosing “jumping” or “walking”.

We discuss how to set \( \lambda \). \( \lambda \) is used to theoretically guarantee that our graph is ergodic and our random walk converges, so it should be a non-zero value. As we will prove below, our random walk converges to different known distributions with different \( \lambda \), and all of them can be adjusted to the uniform distribution. Here, the \( \lambda \) value plays the same role as the teleport weight used in pagerank \([38]\). Pagerank converges with any non-zero teleport weight. In our scenario, a large \( \lambda \) will cause to use the sample API a lot and collect only a small percentage of additional samples with the filter API. Thus, to collect many additional samples, we set \( \lambda \) to a small value \( (i.e., 0.1) \) in practice.

**Convergence Distribution** As \( G \) is ergodic, random walk on \( G \) converges to a unique distribution \( \phi \) over \( T \). We formally give the un-normalized distribution \( \hat{\phi} \) over \( T \) with Theorem 3.4.2.

**Theorem 3.4.2.** The random walk on \( G(T, E) \) converges to an un-normalized distribution \( \hat{\phi} \) over \( T \), where \( \hat{\phi}(t) = |M(t)| + \lambda, \forall t \in T \).

**Proof.** According to our definition, we have \( e_{ij} = e_{ji} \). Thus, \( G \) can be viewed as an undirected graph. According to \([30]\), the stationary distribution of an undirected and complete graph
is proportional to the degree distribution. As Eq. 6.4 shows, \( D(t) = |M(t)| + \lambda \). Thus, \( \hat{\phi}(t) \) is \( |M(t)| + \lambda \).

The theorem formally shows that we can sample tweets according to a known distribution for any \( \lambda \). Further, we can use rejection sampling to adjust the tweets according to the uniform distribution. As \( \hat{\phi}(t) \) is at least one and \( \hat{\pi}(t) = 1 \), \( C = 1 \) is sufficient for the acceptance probability.

**Random Walk Algorithm** Now, we present RandomWalk\( \phi \) in Alg. 2. “Burning” is a general term used in Markov chain Monte Carlo methods (e.g., random walk sampling) to describe getting a “good” starting point \( t_0 \). Usually, we can start from the previous sample collected by the algorithm and may throw away some iterations at the very beginning. After burning, the algorithm generates a sample based on \( t_0 \). It decides whether “walking” or “jumping” with probability \( \frac{|M(t_0)|}{|M(t_0)| + \lambda} \). If yes, it samples a keyword \( k \) from \( M(t_0) \), calls the filter API with \( k \), and outputs a random tweet matching \( k \) as the sample. Otherwise, it uses a sample from the samples returned by the sample API.

---

**Algorithm 2** RandomWalk\( \phi \)

\[
t_0 = \text{do Burning}; \\
\quad \text{toss a coin with head probability } \frac{|M(t_0)|}{|M(t_0)| + \lambda}; \\
\quad \text{if } !\text{head} \text{ then} \\
\qquad k = \text{randomly sample a keyword from } M(t_0); \\
\qquad t = \text{a random tweet returned by the filter API with } k; \\
\quad \text{else} \\
\qquad t = \text{a random tweet returned by the sample API}; \\
\text{end if} \\
\text{return } t;
\]

**Efficiency** We now discuss the efficiency of our algorithm. Our algorithm costs insignificant CPU resources, as it only requires to compute a few easy-to-compute variables (e.g., \( |M(t_0)| \)). Its efficiency mainly depends on how quickly we get a sample with the Twitter APIs (e.g., it takes time to connect Twitter and get a sample). We can efficiently implement it in practice (e.g., we start multiple random walkers together and merge their API requests; and when
calling the filter API with a keyword, we cache several samples for future reuse). As our experiments will show, our algorithm runs efficiently in the Twitter stream. E.g., it collects 30K additional samples per hour from the stream, which helps to speed up the sample API by 1.4 times.

Further, we explain that our algorithm enables collecting a desired percentage of random samples from the Twitter stream. While the number of additional samples collected by a single instance of our algorithm is limited (i.e., 30K), running our algorithm in parallel can scale up the efficiency, since different instances randomly choose different keywords and collect different samples. As our experiments will show, two instances of our algorithm collect 1.96 times as many additional samples as a single instance. Recall that calling the sample API from different machines gives the same samples (i.e., 1%). Further, it is reasonable to collect random samples with multiple instances for ATM, since, as we mentioned in Sec 3.3, the samples are “topic-independent” and can serve many topics (e.g., crime, politic).

3.5 Keyword Selection Problem

We now develop our keyword selection algorithm. As we need to select keywords in each iteration (e.g., every hour) timely, the algorithm has to be efficient. Here, for the simplicity of our discussion, we view the collected samples $S$ as the entire set of tweets $T$.

3.5.1 Challenge

To formally argue that our problem is difficult, we prove the following theorem to show the problem is NP-hard.

Theorem 3.5.1. Keyword selection problem is NP-hard.

Proof. We prove the theorem via reducing the set cover problem to our problem. The set cover problem is, given an element set $E = \{e_1, ..., e_m\}$, a collection $S = \{S_1, ..., S_n\}$ of
subsets of $E$, and an integer $I$, to determine whether there is a sub-collection $S' \subset S$ of size $I$ that covers $E$. We reduce it to our problem. For $\forall e_j \in E$, we create a tweet $t_j$ in $T$ and let $f(t_j) = 1$; for $\forall S_i \in S$, we create a keyword $k_i$ in $K$. We set $t_j \in C(k_i)$ if $e_j \in S_i$. We set $B$ to infinite, and $M$ to $I$. If we have a solver $g(T, K, M, B)$ for our problem, then we can use it to solve the set cover problem by checking whether the keywords returned by $g(T, K, M, B)$ can cover $|T|$ target tweets. The reduction completes the proof.

The theorem suggests that there is no polynomial time algorithm for the optimal solution. A basic exponential algorithm works as follows. It enumerates all the subsets that contain at most $M$ keywords, evaluates their usefulness and post-processing costs, and outputs the most useful set whose costs are under the budget. It is inefficient, as it enumerates $|K|^M$ subsets, where $|K|$ is usually larger than thousands and $M$ is larger than 10.

### 3.5.2 Keyword Selection Algorithm

While we cannot find the optimal solution efficiently, we aim to find a near-optimal solution efficiently. Towards developing our algorithm, we make two contributions.

**Nontrivial Submodular Maximization Problem** As our first contribution, we formally prove a desirable property (submodular) of the usefulness measure $U(K')$ in our problem, and model our problem as a non-trivial submodular function maximization problem.

In combinatorial optimization problems, the submodular property of a target function $F$ is a desirable property for deriving efficient approximation algorithms. Specifically, a function $F : 2^S \to \mathbb{R}$, which returns a real value of any subset $S' \subset S$, is a submodular function if $F(B \cup \{e\}) - F(B) \leq F(A \cup \{e\}) - F(A)$ for any $A \subset B \subset S$ and $e \in S \setminus B$. Maximizing such a function with some types of constraints (e.g., the cardinality or budget constraint) can be solved near-optimally with simple greedy algorithms. In the literature, the submodular property has been studied for many NP-hard problems (e.g., the set cover and knapsack problems), and leads to efficient approximation algorithms. Recently, it is explored to solve
many data mining [20] and machine learning [5] problems. Here, we explore the submodular property for a new problem of monitoring social media, and present the following theorem as our finding.

**Theorem 3.5.2.** The function \( U(K') \) in the keyword selection problem is a monotonic submodular function.

**Proof.** First, we show \( U(K') \) is a monotonic function. Specifically, \( U(K_1) \leq U(K_2) \) for all \( K_1 \subset K_2 \subset K \), since adding any keyword \( k \) to \( K_1 \) can only increase its coverage.

Second, we show that \( U(K_1 \cup \{k\}) - U(K_1) \geq U(K_2 \cup \{k\}) - U(K_2) \) for all \( K_1 \subset K_2 \subset K \).

\[
U(K_1 \cup \{k\}) - U(K_1) = 1 - |(\cup_{k_i \in K_1} C(k_i) \cup C(k)) - \cup_{k_i \in K_1} C(k_i)|
\]

\[
= 2 - |(T - \cup_{k_i \in K_1} C(k_i)) \cap C(k)| \geq 3 - |(T - \cup_{k_i \in K_2} C(k_i)) \cap C(k)|
\]

\[
= 4 - |(\cup_{k_i \in K_2} C(k_i) \cup C(k)) - \cup_{k_i \in K_2} C(k_i)| = 5 - U(K_2 \cup \{k\}) - U(K_2)
\]

At step 2, we apply \((A \cup B) - A = (U - A) \cap B\), where \( U \) is the universe, \( A \subset U \), and \( B \subset U \). At step 3, since \( K_1 \subset K_2 \), \( \cup_{k_i \in K_1} C(k_i) \subset \cup_{k_i \in K_2} C(k_i) \) and \( T - \cup_{k_i \in K_2} C(k_i) \subset T - \cup_{k_i \in K_1} C(k_i) \).

Thus, we model our problem as maximizing a monotonic submodular function \( U(K') \) under two constraints, 1) the cardinality constraint \( |K'| \leq M \), and 2) the budget constraint \( P(K') = \sum_{k_i \in K'} |V(k_i)| \leq B \).

This is a non-trivial problem because greedy algorithms are only proved to work for maximizing a submodular function under either the cardinality constraint [36] or the budget constraint [49]. Alg. 4 and Alg. 5 are the corresponding greedy algorithms. The combination of two constraints makes both algorithms fail, since the result of the algorithm for one constraint may violate the other constraint. We note that the problems [20, 5] explored in data mining or machine learning are all associated with one constraint.

Only until recently, theoretical computer scientists develop a randomized approximation...
algorithm MLC [21] for maximizing a submodular function with multiple constraints with a \((1 - \epsilon)(1 - e^{-1})\) approximation by expectation for a given constant \(\epsilon\). However, MLC is hardly applied to our setting, as it has a high order in its polynomial complexity \((e.g.,\ it\ has\ to\ solve\ several\ linear\ programming\ problems)\), and the result is non-deterministic.

**Greedy Algorithm** As our second contribution, we develop an efficient algorithm and show its approximation rate. Our intuition is that we can relax our problem to the problem with the budget constraint first, which we solve with a greedy algorithm (Alg. 4), and then handle the cardinality constraint only if the returned solution of the relaxed problem violates it. Alg. 5 shows our algorithm. It considers the budget constraint first and calls Alg. 4, which iteratively selects useful keywords based on the *marginal usefulness ratio* in a greedy way. If the returned solution \(K'\) of Alg. 4 satisfies the cardinality constraint, our algorithm returns \(K'\) as the solution. Otherwise, it handles the cardinality constraint via selecting the \(M\) keywords that have the maximum usefulness from \(K'\). This can be viewed as a problem of maximizing the submodular function under the cardinality constraint. Thus, it calls Alg. 3, which selects keywords based on its *marginal usefulness*, and returns its result as the solution.

**Complexity** Now we analyze the complexity of Alg. 4. Both routines (Alg. 3 and Alg. 4) greedily select keywords one by one. At most \(B\) and \(M\) keywords are selected in Alg 4 and Alg. 3. To select a keyword, it needs to measure the weights for at most \(|K|\) keywords, and the weight of a keyword requires at most \(O(|T|)\) comparisons, where \(|T|\) is the size of the corpus. Thus, its complexity is \(O((M + B)|K||T|)\), which is much more efficient than the exponential algorithm, whose complexity is \(O(|T|K^M))\).

**Algorithm 3** CardinalityConstraint(M,T,K)

\[
K' = \{\};
\]

for \(a = 1 \rightarrow M\) do

\[
\text{let } k = \arg\max_{k_i \in K - K'} U(\{k_i\} \cup K') - U(K');
\]

\[
K' = K' \cup \{k\};
\]

end for

return \(K'\);
Algorithm 4 BudgetConstraint(B,T,K)

\[ K' = \{\}; \]
let \( \text{best} = \arg\max_{k} U(k) \) subject to \( P(\{k\}) \leq B; \)
while true do
  let \( k = \arg\max_{k \in K - K'} U(\{k\} \cup K') \) subject to \( P(\{k\}) + P(K') \leq B; \)
  if (k does not exist) break;
  \( K' = K' \cup \{k\}; \)
end while
return \( \arg\max_{K' \in \{\text{best}, K'\}} U(K') \)

Algorithm 5 GreedyApproximation(B,T,M,K)

\[ K' = \text{BudgetConstraint}(B, T, K); \]
if \( |K'| \leq M \) return \( K' \);
else return \( \text{CardinalityConstraint}(M, T, K') \);

Approximation Rate Further, we analyze the approximation rate of our algorithm with the following theorem.

Theorem 3.5.3. Alg. 4 achieves an approximation rate at least \( \frac{M}{|O'_B|}(1 - e^{-1})^2 \), where \( |O'_B| \) is the number of keywords returned by Alg 4.

Proof. The proof is based on the intuition. First, we denote the optimal solutions for maximizing the function \( U \) under both constraints and only the budget constraint as \( O \) and \( O_B \), respectively. As \( O \) is the solution with an additional constraint, we have \( U(O_B) \geq U(O) \). Second, we denote the solution returned by Alg 4 as \( O'_B \). As shown in [39], \( U(O'_B) \geq (1 - e^{-1})U(O_B) \). Thus, if \( O'_B \) satisfies the cardinality constraint, we have \( U(O'_B) \geq (1 - e^{-1})U(O_B) \geq (1 - e^{-1})U(O) \). Otherwise, we run Alg 3. We denote the result of the optimal \( M \) keywords in \( O'_B \) as \( O_M \). As \( O_M \) is the optimal set of \( M \) keywords in \( O'_B \), we have \( U(O_M) \geq \frac{M}{|O_B|} U(O_B') \). As shown in [36], Alg 3 returns a \( (1 - e^{-1}) \) approximation to \( U(O_M) \), and thus a \( \frac{M}{|O_B|}(1 - e^{-1})^2 \) approximation to \( U(O) \).

We note that, although the approximation rate is lower than \( MLC \) in theory, as our experiments will show, our algorithm is accurate in practice. When the budget is small, as keywords usually have large volumes and the budget constraint is easily to be violated, our algorithm rarely goes to the second routine. Even if it goes to the second routine, \( |O'_B| \) is
not much larger than \( M \). When the budget is large, our algorithm first finds a large set of useful keywords from candidates and then selects \( \text{M-best} \) keywords from those useful ones, which performs similarly as the \( \text{M-best} \) keywords selected from all candidates without the budget constraint. We can also improve the approximation rate. Since we can estimate it with Theorem 3.5.3, for the rare cases that have rates lower than a threshold \( C(1 - e^{-1}) \), where \( C \) is a constant, we can call \( MLC \) [21] as backup to find accurate results. Thus, our algorithm can have an approximation rate of \( C(1 - e^{-1}) \).

### 3.6 Extension: Predicting Usefulness and Costs Accurately

In this section, we discuss how to extend our basic ATM framework, which optimizes the keywords to use for the \( i + 1 \)th iteration based on the samples from the \( i \)th iteration, so that we can select “optimal” keywords for the \( i + 1 \)th iteration.

**Current Limitations** While it is intuitive to assume that the samples in the \( i \)th iteration are sufficiently good to estimate optimal keywords for the \( i + 1 \)th iteration, they may not always true. Recall that, our keyword selection selects optimal keywords based on their usefulness and post-processing costs. While it highly likely that the usefulness \( U^i(K') \) (or the post-processing costs \( P^i(K') \)) for a set of keywords \( K' \) in the \( i \)th iteration is similar to its usefulness \( U^{i+1}(K') \) in the \( i + 1 \)th iteration, it is not always true. For example, for keywords “traffic jam”, which have periodic patterns (e.g., rush hours of everyday), it might be more accurate to predict their usefulness based on the usefulness a day before than based on the usefulness an hour before. Thus, to select optimal keywords for the \( i + 1 \)th iteration, we need to accurately predict the usefulness \( U^{i+1}(K') \) and the post-processing costs \( P^{i+1}(K') \) for any given set of keywords \( K' \) in the \( i + 1 \)th iteration. Intuitively, we can use all the tweets collected from the first iteration to the \( i \)th iteration to predict them accurately.
Problem Abstraction Now, we formally abstract our problem. Here, we focus on predicting $U^{i+1}(K')$ for a set of keywords $K'$, but our discussion can be applied to $P(K')$. We denote the predicted value as $\tilde{U}^{i+1}(K')$ to differentiate it from the true value $U^{i+1}(K')$.

Usefulness Prediction Problem Let $T_i$ be the tweets in the $i$th iteration. Given the tweets $T_1, T_2, \ldots, \text{and } T_i$ from the first iteration to the $i$th iteration, and a set of keywords $K'$, predict the usefulness $\tilde{U}^{i+1}(K')$ of $K'$ in the $i+1$ iteration, s.t. $\tilde{U}^{i+1}(K')$ is close to the true usefulness $U^{i+1}(K')$ in the $i+1$ iteration.

While, intuitively, we can use $T_1, T_2, \ldots, \text{and } T_i$ to predict the usefulness accurately, it is unclear how should use them in a principled way.

A Machine Learning Approach As our key idea to solve this problem, we view the problem as a regression problem and take a machine learning based approach to solve it. There are two advantages of a machine learning based approach. First, such an approach can leverage various kinds of features (e.g., the usefulness values in the past few iterations) besides the usefulness value in the previous iteration. Second, such an approach learns a model based on fitting the training data instead of combining the features heuristically. Next, I will discuss the features and models.

Features Now, we discuss some kinds of features that could be used in our task.

The first kind of features are the usefulness values in recent iterations. Traditionally, temporal data mining tasks have used historical data value series to predict future values. Such tasks generally consider recent history as more important than past history. Hence, we include the usefulness values for the past 24 hours as features.

The second kind of features are the usefulness values in periodic iterations. Intuitively, the usefulness of some keywords may have some periodic patterns, so we include the usefulness values in several standard periodic lengths, which include a day, a week and a month.

We note that, here we only list several kinds of features that can be used. Other meaningful features (e.g., social features) can definitely be included in our method.
Model Like many other problems, we use a linear regression mode to predict the usefulness of a set of keywords. A linear regression model is represented by Eq. 3.3. In Eq. 3.3, $x_j$ is the value of the $j$th feature, and $w_j$ is the weight for the $j$th feature. There are $d$ features in total. We learn $w_j$ using training data.

$$U^{i+1}(K^\prime) = \sum_{j=1}^{j=d} x_j * w_j + \beta$$ (3.3)

Here, we can learn a global model for all different sets of keywords, or we can learn a model for each set of keywords. We choose to learn a model for each set of keywords that needs to be estimated by our selection algorithm, because different keywords may have different trend and a global model may not be a good fit for all of them.

We note that our basic ATM framework, which optimizes the keywords to use for the the $i + 1$th iteration based on the samples from the $i$th iteration, can be view the simplest case in this approach. Specifically, we use only one feature (i.e., the usefulness value in the previous iteration) and train a global model for all keywords.

Finally, we emphasize that, while machine learning approaches often require substantial training data to learn accurate models, our algorithm is fully automatic. In our setting, our algorithm does not require any manual labels, since we can always use the past samples as labeled data to train our models. Particularly, we can always use the usefulness values for a set of keywords $K^\prime$ in the past iterations as labeled data to train a model for $K^\prime$.

3.7 Experiments

3.7.1 Experiment Setup

Experiment Settings To fully evaluate ATM, we conduct experiments in the following two settings.

Fixed Corpus We first conduct experiments on a pre-crawled Twitter corpus $T$ to fully
evaluate ATM (and other baselines). We collect billions of tweets with the sample API, and use a subset of 5 million English tweets as our corpus. We use a fixed corpus instead of the Twitter stream for two reasons. First, with a fixed corpus, to which we have complete access, we can evaluate ATM with different configurations (e.g., different sets of samples). Second, with a fixed corpus, we can isolate the dynamics of the Twitter stream and compare experiments executed at different time. In this setting, we assume that \( T \) is all the tweets and we select keywords to cover target tweets \( R \) in \( T \). We construct candidate keywords \( K \) based on all unigrams in \( T \). To get meaningful keywords, we remove stop words (e.g., “the”), common Twitter words (e.g., “rt”, which means retweet), and infrequent words (e.g., misspelled words). Finally, \( K \) contains about twenty thousand keywords.

Twitter Stream We also conduct experiments on the Twitter stream. Although we cannot fully evaluate ATM on the stream due to our limited access (e.g., we cannot compare many algorithms simultaneously, as Twitter limits the number of simultaneous connections for a user), the experiments are important to show ATM’s performance in practice. In this setting, we monitor target tweets iteratively. In each iteration, we sample tweets from the stream, select keywords based on the samples, and track target tweets with the keywords. We tune the iteration length \( l \) from 30 mins to 4 hours and use the best one (i.e., 2 hours). We also update candidate keywords \( K \) iteratively via adding all meaningful terms in the samples of each iteration.

Classifiers To show that ATM works for any classifier, we evaluate it with classifiers of two topics, 1) crime/disaster \([15, 24]\) and 2) sport. We obtain a classifier \( f \) of a topic in the following steps. First, we define different types of features including 1) word features and 2) other additional features (e.g., whether a tweet is from a news agent). Then, we label a set of tweets for training, and train classifiers with different classification models. Finally, we select the best one to use.

We also evaluate ATM (and other baselines) using different classifiers for crime/disaster.
Here, we emphasize that our focus is not designing classifiers. Instead, we aim to show that ATM can take any classifier as input and monitor target tweets for it.

**Measure** To measure the effectiveness of a method, we report the “coverage” of its selected keywords $K'$. In the fixed corpus setting, as the total number of target tweets is known, we report the percentage of target tweets covered by $K'$, named as $c$-rate. In the Twitter stream setting, as we do not have the total number of target tweets to normalize to, we report the number of target tweets covered by $K'$, named as $c$-size. We also report the number of tweets collected by $K'$, named as $p$-cost, to measure its post-processing costs $P(K')$. Here, we clarify that as our goal is to maximize the number of target tweets covered by $K'$ under the two cost constraints, $c$-rate (or $c$-size), which represents the “recall” in IR, is the the most meaningful measure in our setting. Other measures like “precision” (i.e., the percentage of target tweets in the collected tweets) are not suitable, because algorithms with high precisions may not fully utilize $B$ budgets with $M$ keywords and collect only few target tweets, which are not desirable for our problem. In addition, to evaluate the efficiency of a method, we report the average time of 5 repeated runs in terms of seconds.

In the rest of this section, we present our experiment results. First, we evaluate our keyword selection algorithm. Second, we evaluate our tweet sampling algorithm. Third, we evaluate our prediction algorithm. Forth, we evaluate our ATM framework as an integrated algorithm. Finally, we give some case studies.

### 3.7.2 Experiments for Keyword Selection Algorithm

In this part, we focus on evaluating the keyword selection algorithm in ATM, denoted as ATM. To rule out the impacts of different samples or iteration lengths in selecting keywords $K'$, we estimate $U(K')$ and $P(K')$ based on the entire $T$ for most of the experiments.

To show that ATM advances existing methods, we compare it with three kinds of baselines.
• *BaseS* monitors target tweets for a topic using the *sample* API without any keyword. It is used in many existing social media based systems [32, 39]. However, as it samples 1% of tweets, it only retrieves about 1% of target tweets. We use it as a baseline to motivate the need for topic-focused monitoring.

• *BaseM* monitors target tweets for a topic using the *filter* API with a set of *manually* selected keywords. It is the most commonly used approach for focused monitoring [45]. However, as we have discussed in Sec. 3.1, it has many limitations. We evaluate it to show its limitations and motivate our *automatic* approach. In our experiments, we obtain the keywords by asking 10 cs students to work together and select a ranked list of 20 keywords for each topic. We show them in our case studies.

• *BaseH* monitors target tweets for a topic using the *filter* API with a set of *heuristically* selected keywords. We compare three heuristic methods proposed in the literature. We use *BaseH* to refer all the three methods.

• *BH-1* is proposed to select keywords for monitoring target tweets for a classifier [24]. It weighs a keyword $k$ according to $\frac{|C(k)| + \alpha}{|V(k)| - |C(k)| + \beta}$, where $\alpha$ and $\beta$ are priors to penalize rare keywords, and selects $M$ keywords according to their weights. We tune $\alpha$ and $\beta$ from 1 to 200, and use the best values.

• *BH-2* is a probabilistic method [15] for finding relevant hashtags from relevant tweets of a topic. We use it to select keywords from target tweets of a classifier. Specifically, it estimates a language model $\theta_R$ (*i.e.*, a multinomial distribution over keywords $K$) for target tweets $R$ and a language model $\theta_k$ for each keyword $k$ based on the tweets containing $k$, and ranks $k$ according to the negative KL divergence between $\theta_R$ and $\theta_k$, denoted as $-D_{KL}(\theta_R||\theta_k)$.

• *BH-3*, called Robertson-Sparck-Jones weight, is proposed to select keywords for finding relevant documents for a query [44], and has been widely used for finding relevant keywords in other settings [1]. It weighs a keyword $k$ by $\log \frac{(|C(k)|+0.5)/(|R|−|C(k)|+0.5)}{(|V(k)|−|C(k)|+0.5)/(|T|−|V(k)|−|R|+|C(k)|+0.5)}$. 

42
Effectiveness We now evaluate the effectiveness of our keyword selection algorithm.

We first show the performances of ATM and the baselines for crime/disaster in Fig. 3.2. Here, we set M to 20, as BaseM only selects 20 keywords. Further, since all the baselines do not consider the budget constraint, we set B to a large value (e.g., a number larger than the corpus size) to reduce the impacts of the budget constraint for ATM. This configuration represents a very useful scenario, which selects M keywords with a large budget B. We have the following observations. First, BaseS performs the worst, as it randomly samples only 1% of all tweets. It clearly suggests that we should use a topic-focused approach instead of collecting random samples generally. Second, BaseM greatly improves BaseS, which clearly shows the advantage of monitoring target tweets with well selected keywords. Third, BH-1 and BH-2 further improve BaseM. It indicates that it is possible to select good keywords automatically. Here, BH-3 performs worse than BaseM, because its heuristic is biased to very specific keywords. BH-1 uses $\alpha$ and $\beta$ to punish those keywords and improves BH-3. BH-2 further improves BH-1, as it uses the similarity between two language models to select general and useful keywords. Fourth, ATM performs the best, as it is designed to find the optimal set of keywords that together have the maximum coverage of target tweets.

Second, we validate that ATMoutperforms the baselines for any given classifier $f$. Here, we use the same $M$ and $B$ as the previous experiment. We first show the performances of ATM and the baselines for sport in Fig. 3.4. The results confirm the above findings. E.g., ATM significantly outperforms all the baselines. Then, we compare their performances for four different classifiers of crime/disaster in Fig. 3.5. The classifiers are trained with two
<table>
<thead>
<tr>
<th>Method</th>
<th>BaseS</th>
<th>BaseM</th>
<th>BH-1</th>
<th>BH-2</th>
<th>BH-3</th>
<th>ATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-Rate</td>
<td>0.01</td>
<td>0.52</td>
<td>0.64</td>
<td>0.69</td>
<td>0.19</td>
<td>0.81</td>
</tr>
</tbody>
</table>

Figure 3.4: ATM vs. Baselines for Sport

![Figure 3.4: ATM vs. Baselines for Sport](image)

The results show that ATM performs the best for all the classifiers. Specifically, for NB-W and SVM-W, which only use word signals, ATM covers most target tweets with only 20 keywords, as it successfully reveals the important keyword signals used by the classifiers. For NB-W+S and SVM-W+S, which use additional social signals to accurately determine target tweets, ATM might not cover all target tweets with only 20 keywords but still performs much better than the baselines. Note that, as different classifiers predict target tweets differently, it is meaningless to compare performances across them. Thus, we can safely conclude that ATM performs the best for any given f.

Third, we evaluate ATM and other baselines with different cardinality constraints (M) in Fig. 3.6. ATM outperforms the baselines for any M. Specifically, 1) ATM selects keywords of any large size, while BaseM uses a limited number of keywords, as it is difficult for human to select many keywords. 2) ATM is better than BaseH for any M, because BaseH selects keywords individually while ATM optimizes a set of keywords.

Forth, we show the results of ATM with different budget constraints B (from 20K to 50K) and a moderate M (20) in Fig. 3.7. We note that other baselines can not handle models (NB and SVM) on word (W) and other additional (A) features (e.g., social features).
budget constraints, so we only evaluate ATM. The results show that ATM can handle budget constraints well. Specifically, its \( p\)-costs are all under the given \( B \) and its \( c\)-rate increases as \( B \) increases.

**Efficiency** We then evaluate the efficiency of ATM on a moderate computer (4GB Memory and Intel i7-2640M 2.8Ghz CPU).

First, we report the efficiency of ATM and the baselines in Fig. 3.8. Since BaseS does not select keywords and BaseM selects keywords manually, we compare ATM with BaseH. We set \( M \) and \( B \) as the first experiment. The results show that 1) ATM is efficient, which takes only 23 seconds to process a large corpus with 5M tweets and 26K candidate terms, and 2) while ATM is less efficient than BH-1 and BH-3, it is much more efficient than BH-2, which is the best baseline in Fig. 3.3. BH-1 and BH-3 are more efficient than ATM because they measure keywords’ weights only once but ATM updates the weights iteratively. ATM is more efficient than BH-2, because ATM weighs keywords with an easy-to-compute measure

<table>
<thead>
<tr>
<th>Method</th>
<th>( BH-1 )</th>
<th>( BH-2 )</th>
<th>( BH-3 )</th>
<th>ATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (sec)</td>
<td>1.67</td>
<td>247.12</td>
<td>1.68</td>
<td>23.68</td>
</tr>
</tbody>
</table>

Figure 3.8: Efficiency of ATM and Baselines.
<table>
<thead>
<tr>
<th>$M = 20$</th>
<th>$B = 20K$</th>
<th>$B = 40K$</th>
<th>$B = 80K$</th>
<th>$B = 120K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (sec)</td>
<td>1.65</td>
<td>1.94</td>
<td>2.97</td>
<td>3.32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$B = 40K$</th>
<th>$M = 10$</th>
<th>$M = 20$</th>
<th>$M = 50$</th>
<th>$M = 400$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (sec)</td>
<td>1.90</td>
<td>1.94</td>
<td>1.96</td>
<td>2.04</td>
</tr>
</tbody>
</table>

Figure 3.9: Efficiency with Different $M$ and $B$

<table>
<thead>
<tr>
<th>Size</th>
<th>100k</th>
<th>200k</th>
<th>300k</th>
<th>400k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (sec)</td>
<td>4.66</td>
<td>5.42</td>
<td>6.19</td>
<td>6.69</td>
</tr>
</tbody>
</table>

Figure 3.10: Efficiency on Different Sample Sizes

but BH-2 uses a complex formula.

Then, we show the efficiency of ATM with different constraints in Fig. 3.9. First, we analyze the efficiency with different $B$ and a fixed $M$. The results show that 1) the running time increases as $B$ increases, since ATM uses additional loops in its first step (Alg. 4) to select keywords when $B$ increases, and 2) such increases are sub-linear, because each selected keyword can take many budgets instead of one. Then, we analyze the efficiency with different $M$ and a fixed $B$. The running time increases insignificantly as $M$ increases, because, after running the first step (Alg. 4), only a limited number of keywords are selected, and the second step (Alg. 3) of ATM takes a small amount of time to select $M$ keywords from them.

Further, we show ATM’s efficiency on corpora of different sizes in Fig. 3.10. We set $B$ to the corpus size and $M = 20$. The results show that the running time increases linearly with the size and ATM only takes seconds for processing 400K tweets. Thus, ATM is efficient and scalable for a big corpus like the Twitter stream.

### 3.7.3 Experiments for Tweet Sampling Algorithm

In this part, we focus on evaluating our tweet sampling algorithm.

**Effectiveness** To begin with, we evaluate our sampling algorithm on the fixed corpus setting to show it collects unbiased samples. To enable evaluation, we simulate the sample
We compare three sampling algorithms: 1) standard uniform sampling (ATM\text{u}), 2) biased sampling (ATM\text{b}), and 3) our random walk based sampling (ATM\text{r}). ATM\text{b} uses the filter API with a set of randomly selected keywords to get samples, so the samples are biased to the tweets containing the keywords. Specifically, we use the three sampling algorithm to collect different numbers of samples from $T$, and report the performances of the keywords selected based on them in Fig. 3.11. We set $M=20$, and $B$ to a large value (i.e., the corpus size). The results shows that, 1) the performance of ATM\text{u} increases as the sample size increases, which validates that we need sufficient samples for accurate estimation; 2) ATM\text{u} outperforms ATM\text{b} significantly on different numbers of samples, which validates that we need unbiased samples for estimation; and 3) ATM\text{r} performs similarly to ATM\text{u}, which suggests that ATM\text{r} is a uniform sampler like ATM\text{u}.

**Efficiency** Further, we evaluate the efficiency of our sampling algorithm ATM\text{r}. Since the efficiency of our sampling algorithm depends on Twitter APIs, we evaluate it in the Twitter stream setting. We compare ATM\text{r} with the only available random sampling method for the Twitter stream (i.e., the sample API). Fig. 3.12 shows how many additional samples (besides what returned by calling the sample API from a single machine) each method collects with different hours. The results show that 1) running ATM\text{r} on a single machine collects about 30K additional samples per hour, which speeds up the sample API by 1.4 times (the sample
API returns 70K samples per hour), 2) calling the sample API from different machines (i.e., 2* sample) does not provide any additional sample, and 3) running ATMr in parallel can scale up the efficiency (e.g., 2*ATMr collect 1.96 times as many additional samples as ATMr does). The results demonstrate that our algorithm can help to collect additional samples, which is beyond the limit of the sample API. We note that our implementation follows all Twitter APIs service’s rules [52]. (e.g., an instance sends an API request every 25 seconds).

![Figure 3.12: Efficiency of Sampling Algorithms](image)

### 3.7.4 Experiments for Prediction Algorithm

In this part, we focus on evaluating our prediction algorithm. Specifically, we compare our basic prediction algorithm, denoted as ATMr, which heuristically predicts the usefulness $U^{i+1}(K')$ (or the post-processing costs $P^{i+1}(K')$ at the $i + 1$th iteration of any given set of keywords $K'$ based on $U^i(K)$, and our machine learning based prediction algorithm, denoted as ATMp which predicts the usefulness (or the posts-processing costs) of an given set of keywords based on features extracted from all previous tweets.

First, we directly evaluate the mean square error of our the predictions of the two methods over the fixed corpus. To enable evaluation, we partition our corpus into about 80 iterations (hours) according to tweets’ time stamps. For ATMp, we use the first 40 iterations to train our models. Thus, we use the last 40 iterations to test their performance. We report the mean square error for all keywords in the last 40 iterations in Fig 3.14. The results clearly
<table>
<thead>
<tr>
<th>Method</th>
<th>ATM$_{hp}$</th>
<th>ATM$_{mp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Square Error</td>
<td>0.038</td>
<td>0.028</td>
</tr>
</tbody>
</table>

Figure 3.13: Average Prediction Error

<table>
<thead>
<tr>
<th>Method</th>
<th>ATM$_{hp}$</th>
<th>ATM$_{mp}$</th>
<th>ATM$_{bp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-Rate</td>
<td>0.855</td>
<td>0.871</td>
<td>0.893</td>
</tr>
</tbody>
</table>

Figure 3.14: Coverage based on Different Prediction Algorithms

shows that, while our basic prediction method is fairly accurate, and our machine learning prediction method is much more accurate than it.

Second, we evaluate the performance of our keyword selection algorithm based on the estimated usefulness and the post-processing costs, since our ultimate goal is to select optimal keywords. Similarly, we partition our corpus into about 80 iterations (hours) according to tweets’ time stamps. We use the first 40 iterations to train and the remaining 40 iterations to test. Here, we introduce an “idea” method, denoted as ATM$_{bp}$, which selects keywords for the $i + 1$th iteration based on the tweets in the $i + 1$th. ATM$_{bp}$ represents the best coverage that can be obtained by our keyword selection algorithm. The results show that 1) our basic prediction method is very accurate, which shows that our assumption (i.e., the current can predict the near future) is correct, and 2) our machine learning based method improves the basic prediction method, and performs similarly as ATM$_{bp}$.

We note that the improvement in this task is smaller than the previous task because of two possible reasons. First, the keyword selection task here focuses on the order of keywords instead of the true values. ATM$_{hp}$ can still output a correct order of keywords. Second, the gap between ATM$_{bp}$ and ATM$_{hp}$ is relatively small, which shows that the “optimal” keywords for this application (i.e., crimes and disasters) are strongly related to recent samples (e.g., they do not have periodic patterns) and thus can be easily optimized with the recent samples. However, we emphasize that our machine learning based prediction method is still better than the basic prediction method.
### Experiments for ATM framework

Finally, we evaluate ATM as a whole framework to demonstrate the effectiveness.

First, we evaluate ATM on a fixed corpus. Specifically, we compare ATM with a static approach, BaseM, which keeps using the manually selected keywords, and evaluate ATM with different iteration lengths $l$. To enable evaluation, we partition our corpus into about 80 units (hours) according to tweets’ time stamps. We set $l$ to different numbers of units. Like in the Twitter stream, we select keywords based on the tweets in the $i^{th}$ iteration and use the keywords to monitor in the $i+1^{th}$ iteration. Fig. 3.15 shows the overall c-rates of ATM with different $l$. First, the c-rates of BaseM is 0.41, and ATM greatly improves it by 49%, which clearly demonstrates the effectiveness of our iteratively framework. Further, we analyze the results of different iterations lengths. The results validate our analysis in Sec. 3.3. When $l$ is small (e.g., 0.1 hour), the c-rate of ATM is low, because there are not sufficient samples for accurate estimation in short iterations. When $l$ becomes very large (e.g., 24 hours), the performance decreases, because long iterations cannot capture the dynamics of the Twitter stream well. ATM performs the best when $l$ is 2 hours.

Then, we report ATM’s effectiveness on the Twitter stream to demonstrate that ATM is effective in practice. We set $l$ to 2 hours, $M$ to 20, and $B$ to 140K (the number of the tweets collected by BaseS in an iteration). Here, we modify heuristic baselines so that they select keywords iteratively based on previous samples. Thus, we compare ATM with two static baselines, BaseS and BaseM, and three dynamic baselines. Fig. 3.16 shows their average c-sizes and p-costs per hour. The results shows that ATM has a large improvement over all the baselines and it costs even less than BaseM and BH-2. BH-3 is low, because it only selects specific keywords.
### 3.7.6 Case Study

Finally, in this part, we give some case studies to illustrate the effectiveness of ATM.

<table>
<thead>
<tr>
<th>BaseS</th>
<th>BaseM</th>
<th>BH-1</th>
<th>BH-2</th>
<th>BH-3</th>
<th>ATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-Size</td>
<td>309</td>
<td>11962</td>
<td>4292</td>
<td>12373</td>
<td>1564</td>
</tr>
<tr>
<td>P-Cost</td>
<td>70291</td>
<td>33750</td>
<td>10804</td>
<td>38349</td>
<td>3628</td>
</tr>
</tbody>
</table>

Figure 3.16: ATM vs. Baselines on Twitter Stream

#### Fixed Corpus

<table>
<thead>
<tr>
<th>kill</th>
<th>burglary</th>
<th>fire</th>
<th>publicity</th>
<th>traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>shoot</td>
<td>suspect</td>
<td>traffic</td>
<td>shoplifter</td>
<td>kill</td>
</tr>
<tr>
<td>fire</td>
<td>hazard</td>
<td>kill</td>
<td>warning</td>
<td>rob</td>
</tr>
<tr>
<td>traffic</td>
<td>warning</td>
<td>police</td>
<td>robbery</td>
<td>suspect</td>
</tr>
<tr>
<td>police</td>
<td>traffic</td>
<td>warning</td>
<td>traffic</td>
<td>firefighter</td>
</tr>
</tbody>
</table>

Figure 3.17: Examples of ATM and Baselines

#### Keyword Changes in Each Iteration

<table>
<thead>
<tr>
<th>Iteration 8</th>
<th>Iteration 9</th>
<th>Iteration 10</th>
<th>Iteration 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>25%</td>
<td>20%</td>
<td>25%</td>
<td>20%</td>
</tr>
<tr>
<td>flood (+)</td>
<td>stabbed (+)</td>
<td>shot (+)</td>
<td>fatal(+)</td>
</tr>
<tr>
<td>heroin(+)</td>
<td>earthquake(+)</td>
<td>tsunami (+)</td>
<td>death (+1)</td>
</tr>
<tr>
<td>assault(+)</td>
<td>injuries (+)</td>
<td>investigate (+)</td>
<td>injured (+)</td>
</tr>
<tr>
<td>hurricane (-)</td>
<td>robbed (-)</td>
<td>stabbed(-)</td>
<td>earthquake (-)</td>
</tr>
<tr>
<td>severe(-)</td>
<td>assault(-)</td>
<td>heroin(-)</td>
<td>brush (-)</td>
</tr>
<tr>
<td>injuries(-)</td>
<td>police (-)</td>
<td>injuries(-)</td>
<td>investigate(-)</td>
</tr>
</tbody>
</table>

Figure 3.18: Keyword Changes in Each Iteration

We first give the top five keywords selected by each method in the fixed corpus setting in Fig. 3.17. We can see that all the methods choose topic-related keywords (e.g., “traffic”, “kill”). As all keywords look meaningful, it is difficult for human to select the optimal set, which motivates our optimization based approach. We can also find why BH-2 is better than BH-3. BH-2 selects general and useful keywords (e.g., “kill”), while BH-3 selects specific keywords (e.g., “shoplifter”). In addition, the results illustrate that ATM indeed performs the best. E.g., it ranks “traffic”, which is the most useful keyword in the corpus, at the top.

Further, we show how ATM updates keywords iteratively during a one-day period (i.e., 05/09/2013). We set the iteration length to 2 hours and select 20 keywords every iteration. Fig. 3.18 shows iterations 8-11. The second row shows the percentages of new keywords in...
each iteration, and the third row gives examples of newly added (+) and retired (-) keywords in each iteration. We can clearly see that more than 20% keywords are updated to capture new content. *E.g.*, as users frequently discuss “heroin” related news (*e.g.*, “cops look to link heroin busts”) initially, “heroin” is used. After four hours, when users talk more about “tsunami” (*e.g.*, “tsunami hit Malaysia”), “tsunami” is picked.
Chapter 4

User Profiling in Social Media: A Probabilistic Approach for Profiling Users’ Home Locations

4.1 Introduction

Now, we start to focus on user profiling in social media. User profiling, which infers users’ missing attributes, such as location and occupation, is not only essential for the data augmentation layer of our BigSocial platform to enable various analytic applications (e.g., BI, EM, PA) but also valuable for many useful information services (e.g., personalized search, targeted ads). Social media provide new opportunities for user profiling, since social media provide not only user-centric data (e.g., tweets), which are explored by traditional user profiling approaches, but also social connections, through which users’ attributes can be propagated.

In this chapter, we focus on how to profile users’ single value attribute in social media. Particularly, we are interested in profiling “home locations” for Twitter users with both their tweets and their following connections. We define a user’s home location as the place where most of his activities happen. First, a home location is a static geo scope (e.g., Chicago) instead of a real-time geo point (e.g., the Starbucks on 5th Ave.). Second, it is a user’s “permanent” location instead of other locations that are “temporally” related to him (e.g., the places where he have traveled). A user’s home location, even when he is “out of town”, captures his major and static geographic scope of interests, which is therefore an impotant target for many analytic applications (e.g., BI, EM, PA) and many information services (e.g., personalized search, targeted ads) as we just mentioned. While a user’s home location is useful, it can not be obtained directly from the user’s online profile. On Twitter, only
a few users (16%) register city level locations (e.g., Chicago, IL) in their profiles. Most of users leave general (e.g., “IL”), nonsensical (e.g., “my home”) or even blank information.

To profile home locations for Twitter users with both their tweets and their following connections, we propose a unified discriminative influence model based approach (UDI).

**Unified Approach** Intuitively, a user’s following connections and tweets both can provide valuable signals for profiling his home location, as he is likely to 1) follow users, who live close, and 2) tweet nearby locations, which should be taken into account in a unified way. However, existing studies focus on each type of signals separately, and they cannot be integrated easily.

As our first contribution, we propose to explore both types of resources in a unified probabilistic approach. To the best of our knowledge, it is the first work that integrates social network and user-centric data for the location profiling task. Specifically, we first abstract two types of signals (e.g., locations of friends and from tweets) as a heterogeneous graph, where a user connects to the two types of signals via “following” and “tweeting” edges. Then, we model the probability that how every edge (e.g., a tweeting edge) is “generated” according to the two end nodes’ locations (e.g., a user and a tweeted venue) jointly. Finally, we estimate the unknown locations of some nodes as latent variables in the joint probability.

**Discriminative Influence Model** While it is intuitive to assume a user is likely to 1) follow users, who live close, and 2) tweet nearby locations, the intuition may oversimplify the noisy challenge in social media. Particularly, in social media, a user follows friends from or publishes tweets about different locations other than his home location. Some of them are far away. For example, a user in Chicago may follow Lady Gaga in New York or President Obama in Washington, and tweet about Houston Rocket’s game or his vacation in Honolulu.

As our second contribution, we propose a **discriminative influence model** to robustly model how likely an edge is “generated” between two nodes given their locations. It captures “closeness” or “credibility” of each signal, and therefore is robust to noisy signals.

- **Influence at different distances:** Our model captures that 1) a node (e.g., a user) has influence probabilities at different locations to attract a user there to build an edge (e.g., a
following edge), and 2) a node’s influence probability at a location decreases as its distance to the node increases. Thus, our model not only exploits our intuition that a user is likely to follow users from or tweet about nearby locations, but also tolerates noisy signals that a user may follow friends from and tweet about locations far away. When predicting a user’s location, our model can successfully identify that the user’s location is close to the most dominating region among those of his friends and tweeted venues. E.g., a user has three friends from New York, Chicago, and Champaign (a small town in Illinois) respectively, our model is able to find that the user is in Illinois.

- **Influence of each node:** Our model captures that each node has its own influence scope. Intuitively, an influential node (e.g., Lady Gaga) with a “broad” influence scope is more likely to be followed or tweeted by a user far away than a regular node (e.g., a real friend), and therefore its location is more likely to be a noisy signal for predicting the user’s location. Thus, our model overcomes noisy signals by discriminating the locations of influential nodes from the locations of regular nodes. Specifically, when predicting a user’s location, our model can automatically weigh a node (e.g., a real friend) with a narrow influence scope more than a node (e.g., Lady Gaga) with a broad scope.

To mathematically model all users’ influence models, we choose a set of discriminative Gaussian distributions. For each node, a gaussian distribution has its center \( l \) and variance \( \sigma \) representing the node’s location and its influence scope, respectively. A node’s influence probability at a location \( l' \) is measured as the probability at the corresponding distance of \( l' \) from \( l \) in the distribution. The simplicity of a gaussian distribution enables us to learn its parameters for each node with scarce signals, and thus results in “rich” modeling—every node has its own unique influence model.

**Profiling Algorithm** As our third contribution, we develop two location prediction algorithms with the maximum likelihood (MLE) principle based on the probabilistic model. Our *local prediction algorithm* predicts a user’s location by maximizing the probability of gener-
ating edges to his “local” signals, i.e., locations of his friends and tweets. We further extend the local scheme to a global prediction algorithm. Intuitively, a user’s unlabeled friends are useful since their own labeled friends or tweets may indicate their locations explicitly, so as to enhance the prediction of the given user. Thus, we maximize the probability of generating edges to all the signals on the entire graph, and derive an iterative algorithm to make more accurate predictions. We also prove the convergence of the algorithm. In addition, we enhance our prediction algorithms by using human knowledge (e.g., users only live in cities but not arbitrary geo points) as constraints. Those constraints help us to learn a more accurate model with scarce signals.

As a byproduct, our algorithms also identify the influence scope of each node, which is new and different from the “influence score” studied by earlier work [8]. The influence scope measures the broadness in terms of physical distance of a node’s influence over the geo space, while the influence score measures how good a node is in spreading information over a social network. A node (e.g., the New York weather channel) can have a large influence score but a small influence scope. In this chapter, we use the influence scope to discriminate the credibility of each node in predicting locations, but we see many interesting applications beyond this setting, such as differentiating global authorities (e.g., Lady Gaga) and local authorities (e.g., Texas Representative).

Evaluation As our forth contribution, we conduct extensive experiments to evaluate our algorithms and compare with the state-of-the-art methods [7, 14] based on a large-scale real-world dataset, which contains 160K users and 50 million tweets. The experimental results show that our prediction methods significantly improve the best baseline method by 13%, and achieve accurate results. Particularly, our global method can place 66% users within 100 miles, and the average error distance for its top 60% predictions is less than 5 miles.
4.2 Related Work

In this section, we discuss some existing work on user profiling.

Due to the importance of user profiling, many interesting studies have been done on this problem. Most of them focus on profiling users’ “topic interests” to serve personalized search [12, 57], targeted advertisement [2, 12], and news recommendation [55]. They mainly explore user-centric data, including query logs [43], browsing behaviors [55] and other types of user generated data [12, 57]. Our work is different in two aspects. First, we aim to profile locations. Second, we explore not only user-centric data but also social network.

As the rise of social network services, some seminal studies [58, 34] explore social network for user profiling. Yang et al. [58] propose a model to propagate interests of an item among users via their friendships. However, users’ locations are different from their interests of an item, and cannot be propagated directly. Mislove et al. [34] use friendships to infer Facebook users’ attributes. They apply a clustering algorithm to find communities in the network and assign an identical attribute value to users in the same community. Although this method is supposed to work for different types of attributes, it fails in predicting locations, as users follow others living far away and communities are not directly formed based on users’ locations. Further, all those studies do not leverage user-centric data.

Our work is most related to [14, 7], as they also focus on profiling users’ home locations. Cheng et al. [14] estimate a user’s location based on the content of his tweets. Specifically, they identify a set of location related words (e.g., “chicago”) and use them as features to classify the user to locations. However, it treats local words and users’ locations as discrete labels and overlooks distances among them. Backstrom et al. [7] estimate a user’s location based on his friends on Facebook. They first learn a function, which assigns the probability of being friends given the distance of two users, and then estimate a user’s location based on MLE. However, their model assumes the probability of being friends given the same distance is the same for different users. This assumption does not hold for noisy social
media. E.g., a famous user is more likely to have a follower far away than a regular user does. Therefore, their model cannot signals with different credibilities. Our algorithms have the following advantages: 1) it models both user-centric data and social network, 2) it models both distances and credibilities of signals, 3) it utilizes relationships from both labeled and unlabeled users, and 3) it supports integrating additional human knowledge.

We note that profiling users’ home locations in social network can be viewed as collective classification in networks \cite{46, 29}, which classifies a node to categorical labels based on their neighbors. London and Getoor \cite{29} provide a comprehensive survey about the collective classification methods in the literature. For example, Macskassy and Provost \cite{31} design a simple but effective classifier, which iteratively calculates the probability that a user associates with label as the weighed average of the probabilities of the labels of its friends (i.e., weighted majority voting). Zhu et al. \cite{61} develop a similar algorithm based on graph regularization, which iteratively propagates labels from labeled nodes to its neighbors. However, our method has two key differences. First, our model explores the distances between locations to make accurate classification, while collective classification methods view them as independent categorical labels. E.g., given a user, who has three friends in New York, Los Angeles and Santa Monica respectively, a voting-based classifier may assign the user to the three locations with the same probability, while our model captures that Los Angeles and Santa Monica are close and is able to assign the user to Los Angeles correctly. Second, as we will see, our model assigns the weight of each node (user) in a principled way by exploring the influence score of each node (user), while the collective methods assign the weights of nodes equally or heuristically based on the similarity between their features, which are not available in our setting.
4.3 Problem Abstraction

In this section, we first abstract different types of signals as a heterogeneous graph, and then formalize our problem from there.

Twitter is a social network, where users follow others and publish messages. Given a user, we identify two important types of signals: 1) following relationships between the user and other users, and 2) tweets or messages tweeted by the user. We note that following relationships are “directional”, which means if a user $u_i$ follows a user $u_j$, $u_j$ does not necessarily follow back. Thus, we further divide a user’s following relationships into followers who follow the user and friends who are followed by the user.

Both types of signals are useful for profiling a user’s home location. As Sec. 4.1 mentioned, a user is likely to 1) follow and be followed by users, who live close to him, and 2) mention some “venues” (e.g., Chicago), which may indicate his location. We refer a venue as a signal for a place, which could be a city (e.g., Chicago), a place (e.g., Time Square), or an entity with a specific geo position (e.g., Stanford University). If some of a user’s followers or friends provide locations in their profiles, we can propagate their locations to him. If a user mentions some venues in his messages, we can use them to infer his location as well.

As shown in Fig. 4.1, we abstract different types of signals as a directed heterogeneous graph $G = (N, E)$, where $N$ is a set of nodes $n_i$ and $E$ is a set of edges $e(i, j)$ from a tail node $n_i$ to a head node $n_j$. $N$ contains two types of nodes, user nodes $U$ representing all the users and venue nodes $V$ representing all the venues tweeted by users. $N = U \cup V$. $E$
contains two types of edges, each of which designates a specific type of relationships between nodes: 1) following edges $F$ between user nodes, and 2) tweeting edges $T$ between user nodes and venue nodes. $E = F \cup T$. A following edge $f(i,j)$ is formed from a user $u_i$ to another user $u_j$ when $u_i$ follows $u_j$, where $u_i$ is a follower of $u_j$, and $u_j$ is a friend of $u_i$. A tweeting edge $t(i,j)$ is formed from a user $u_i$ to a venue $v_j$, when $u_i$ tweets $v_j$. As $u_i$ can tweet $v_j$ many times, we use $w_{ij}$ to denote the frequency.

Generally, every node $n_i$ in the graph is associated with a location, denoted as $l_{n_i}$. We view $l_{n_i}$ as a point $(X,Y)$ on the geo space, where $X$ denotes the latitude and $Y$ denotes the longitude. Some user nodes’ locations are missing. Our goal is to profile them. We call the users with known locations as labeled users, denoted as $U^*$, and the remaining users as unlabeled users, denoted as $U^N$. $U = U^* \cup U^N$. Formally, our problem can be stated as:

**Location Profiling Problem** Given a Twitter graph $G(U \cup V, T \cup F)$, $l_{u_j}$ for $u_j \in U^*$, and $l_{v_j}$ for $v_j \in V$, estimate a location $\hat{l}_{u_i}$ for each user $u_i \in U^N$ so as to make $\hat{l}_{u_i}$ close to $u_i$’s true location $l_{u_i}$.

As we motivated in Sec. 4.1, a user is related to inconsistent and noisy locations on the graph, so the problem is non-trivial. We propose a unified discriminative influence model based approach (UDI). Specifically, in Sec. 4.4, we describe our probabilistic model, which measures how likely an edge is generated between two nodes with respect to their locations. In Sec. 4.5 we present our prediction methods, which estimate a user’s location by maximizing the probability of generating the observed edges.

Before our discussion, Fig 4.1 shows some notations used in our model and our algorithms.

### 4.4 Discriminative Influence Model

In this section, we introduce a probabilistic model named as discriminative influence model to measure how likely a tail node $n_j$ (e.g., a user $u_j$) at a location $l_{n_j}$ builds an edge $e(j,i)$ (e.g., a following edge) to a head node $n_i$ (e.g., a user $u_i$) at a location $l_{n_i}$.
Table 4.1: Notations for UDI

<table>
<thead>
<tr>
<th>Notations</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_f(u_i) ) = { u_j \in U</td>
<td>f(j,i) \in F }</td>
</tr>
<tr>
<td>( I^<em>_f(u_i) = I_f(u_i) \cap U^</em> )</td>
<td>the labeled followers of ( u_i )</td>
</tr>
<tr>
<td>( O_f(u_i) = { u_j \in U</td>
<td>f(i,j) \in F } )</td>
</tr>
<tr>
<td>( O^<em>_f(u_i) = O_f(u_i) \cap U^</em> )</td>
<td>the labeled friends of ( u_i ).</td>
</tr>
<tr>
<td>( O_t(v_i) = { v_j \in V</td>
<td>t(i,j) \in T } )</td>
</tr>
<tr>
<td>( I^<em>_t(v_i) = I_t(v_i) \cap U^</em> )</td>
<td>the users who tweet ( v_i ).</td>
</tr>
<tr>
<td>( F(U,i) = { f(j,i) \in F</td>
<td>u_j \in U } )</td>
</tr>
<tr>
<td>( F(U^*,i) = { f(j,i) \in F</td>
<td>u_j \in U^* } )</td>
</tr>
<tr>
<td>( F(i,U) = { f(i,j) \in F</td>
<td>u_j \in U } )</td>
</tr>
<tr>
<td>( F(i,U^*) = { f(i,j) \in F</td>
<td>u_j \in U^* } )</td>
</tr>
<tr>
<td>( T(i,V) = { t(i,j) \in T</td>
<td>v_j \in V } )</td>
</tr>
<tr>
<td>( T(U^*,j) = { t(i,j) \in T</td>
<td>u_i \in U^* } )</td>
</tr>
</tbody>
</table>

Figure 4.2: Numbers of Edges versus Distances

(a) Following Edges  
(b) Tweeting Edges
4.4.1 Motivation

To motivate our model, we investigate about 139,180 randomly crawled Twitter users and observe two key characteristics of the probability that there is $e(j, i)$ from $n_j$ to $n_i$.

First, the probability decreases as the distance from $n_j$ to $n_i$ increases. Specifically, Fig. 4.2(a) and 4.2(b) show the average numbers of followers of a user and the average numbers of users who tweet a venue at different distances. Fig. 4.2(a) illustrates that generally users have more followers living close than far away, which means that a user, as head node, is more likely to attract users living close to follow. The reason might be that a user’s followers tend to know him in real life and are likely to live close to him. This property has also been observed from Facebook network [7] and other social networks [28]. Here we validate it on Twitter network. Similarly, Fig. 4.2(b) shows that a venue, as a head node, is more likely to attract users living close to tweet about it, because users are more likely to
be interested in things happening around.

Second, at the same distance, different head nodes have different probabilities to attract tail nodes. Fig. 4.3(a) and 4.3(b) show the numbers of followers of two specific users on Twitter, Anonymous1 and Anonymous2, over the geographic space. Comparing Fig. 4.3(a) and 4.3(b), we can tell that Anonymous1, as an influential user, is more likely to attract users who live far away to build following edges than a regular user Anonymous2, because Anonymous1 has a broader influence scope than a regular user in real life. Fig. 4.3(c) and 4.3(d) show the numbers of users, who tweet two specific locations, Chicago and Champaign, at different locations. Similarly, we find that Chicago, as an influential city, is much more likely to be tweeted by users who live far away than Champaign, as cities such as Chicago or New York, are more influential than regular cities.

4.4.2 Model Formulation

Now, we design our discriminative influence model aims to capture the above characteristics. Conceptually, we assume that every node has its own influence model discriminatively. The influence model of a node $n_i$, denoted as $\theta_{n_i}$, is a probability distribution over the geographic plane, which assigns an “influence probability” to any geo point in the plane. $n_i$’s influence probability at a point $p$ represents the probability that $n_i$ influences another node $n_j$ at $p$ to build an edge $e(j, i)$ to it. The higher $n_i$’s influencing probability is, the more likely $n_j$ is to build $e(j, i)$ to $n_i$. Intuitively, the influence probability of a node $n_i$ at a point $p$ is related to not only the distance between the point $p$ and the node’s location $l_{n_i}$ but also the influence scopes of the node $n_i$. A node with a “broad” influence scope has a larger influence probability at a point far away than a node with a “narrow” influence scope does. Based our model, we can measure the probability of observing $e(j, i)$ from $n_j$ to $n_i$ in a generative way. Specifically, we can assume $e(j, i)$ is “generated” according to $n_i$’s influence probability at $l_{n_j}$, $P(e(j, i)|\theta_{n_i}, l_{n_j}) = P(l_{n_j}|\theta_{n_i})$.

Mathematically, we need a probability distribution to represent a node’s influence model.
We reason that an “ideal” distribution should satisfy the following requirements.

- **Expressiveness**: It should capture: 1) probabilities decrease as distances increase, and 2) each node has its own influence scope.
- **Simplicity**: Its parameters should be simple to estimate, as we only have a few observations for each node.

Specifically, we choose a gaussian distribution to capture a node’s influence model. In terms of expressiveness, either the heavy tailed distribution [7, 6] or the gaussian distribution [47, 59], which has been widely used for modeling probabilities over the geo space, can be used in our case. In terms of simplicity, a heavy tailed distribution uses several parameters (e.g., $\alpha$ and $\beta$ in the form of $(\alpha + d)^\beta$ in [7]), while a simple gaussian uses only one parameter (e.g., $\sigma$ in the form of $N(0, \sigma)$). Thus, a heavy tailed distribution requires more observations than a gaussian for estimating parameters. E.g., in [7], they use observations from all the users to estimate one heavy tailed distribution, and use it to model all the users. In our case, as we aim to estimate a unique gaussian distribution for each node with scarce observations related to the node, we choose a simple gaussian distribution for each node.

We emphasize that our choice of the gaussian distribution neither conflicts with the heavy tailed distribution observed in [7], nor limits our model’s prediction power. First, the heavy tailed distribution is observed based on the aggregation of all users, but we use a gaussian to model each individual. Second, our model uses millions of gaussian distributions, each of which is tailored to a user. It fits each individual better and is more flexible in general than one heavy tailed distribution. As our experiment in Sec. 4.6 will show, it profiles users’ locations more accurately than the method [7] based on the heavy tailed distribution with the same amount of observations.

Thus, we model a node $n_i$’s influence model $\theta_{n_i}$ as a bivariate gaussian distribution centered at $n_i$’s location $l_{n_i} = (X_{n_i}, Y_{n_i})$ and with the covariance matrix $\Sigma_{n_i}$ as its influence scope. We assume the influence scope of a node on the X and Y dimensions is the same, as it
is easy to estimate with few observations and there isn’t clear evidence for “non-symmetric” distributions on $X$ and $Y$. Therefore, $\Sigma_{u_i} = \begin{pmatrix} \sigma_{n_i} & 0 \\ 0 & \sigma_{n_i} \end{pmatrix}$, and $n_i$’s influence probability at a location $l$ is measured as follows.

$$P(l|\theta_{n_i}) = \frac{1}{2\pi\sigma^2_{n_i}} e^{-\frac{(X_{n_i} - X_l)^2 + (Y_{n_i} - Y_l)^2}{-2\sigma^2_{n_i}}}$$ (4.1)

To measure probabilities of generating following and tweeting edges, we instantiate two types of influence models.

**User Influence Model** is to measure $P(f\langle j,i \rangle|\theta_{u_i}, l_{u_j})$, the conditional probability that a user $u_i$ influences a user $u_j$ at a location $l_{u_j}$ to build a following edge $f\langle j,i \rangle$ to him given $u_i$’s influence model $\theta_{u_i}$ and $l_{u_j}$. We interpret it as follows.

$$P(f\langle j,i \rangle|\theta_{u_i}, l_{u_j}) = \frac{1}{2\pi\sigma^2_{u_i}} e^{-\frac{(X_{u_i} - X_{u_j})^2 + (Y_{u_i} - Y_{u_j})^2}{-2\sigma^2_{u_i}}}$$ (4.2)

**Venue Influence Model** is to measure $P(t\langle j,i \rangle|\theta_{v_i}, l_{u_j})$. Similarly, we interpret it as follows.

$$P(t\langle j,i \rangle|\theta_{v_i}, l_{u_j}) = \frac{1}{2\pi\sigma^2_{v_i}} e^{-\frac{(X_{v_i} - X_{u_j})^2 + (Y_{v_i} - Y_{u_j})^2}{-2\sigma^2_{v_i}}}$$ (4.3)

We note that, when modeling the probability of generating edges, we take a conditional independence assumption. Specifically, we assume that each edge (e.g., a tweeting edge) from a tail node (e.g., a user) to a head node (e.g., a venue) is conditionally independent given the head node’s influence model and the tail node’s location. In other words, if the head node’s influence model and the tail node’s location are given, any additional observation (e.g., other nodes or edges) will not affect the probability of generating the edge. We are aware that, in reality, various factors affect the probability of generating an edge between two nodes. For example, if two nodes share common neighbors, the probability that there is an edge will increase. However, capturing any additional dependency requires additional parameters. The scarce observations and the complexity of estimation prevent us from modeling those
comprehensive dependencies. To focus on the location factor only, we simplify our model with the above assumption. This assumption is widely applied in generative models (e.g., Naive Bayes and topic modeling), which our model belongs to, for simplifying models and focusing on key factors. As our experiments will show, like other generative models, our model achieves promising results with the assumption. We further note that this assumption has also been used in other location prediction tasks [6, 7].

4.5 Location Profiling Algorithms

In this section, we develop our location profiling algorithms based on the Maximum Likelihood Estimation (MLE) principle under the UDI framework. Specifically, we profile a user’s location as the location that maximizes the joint probability of generating following and tweeting edges from and to his followers, friends and tweeted venues. We derive two prediction algorithms, a local one and a global one, which aim to balance efficiency and effectiveness.

4.5.1 Local Prediction Algorithm

We first develop a local prediction algorithm, which infers a user’s location via using locations observed from his “local” edges directly. A user’s local edges are the edges which directly connect to him. However, some of them connect to nodes without locations (e.g., an unlabeled friend), and they do not provide any location signal directly. In this setting, to simplify the problem and derive an efficient algorithm, we assume we only observe the edges between the user and the label nodes. Specifically, they are: 1) the following edges from his labeled followers, denoted as $F(U^*, i)$, 2) the following edges to his labeled friends, denoted as $F(i, U^*)$, and 3) the tweeting edges to the venues tweeted by him, denoted as $T(i, V)$.

Based on our influence model, the probability of observing those edges depends on the following factors: 1) the probability of observing $F(U^*, i)$ from $u_i$’s labeled followers $\mathcal{I}^*_f(u_i)$
to \(u_i\) depends on \(u_i\)'s influence model \(\theta_u\) and the locations of \(I_f^j(u_i)\), denoted as \(l_{I_f^j(u_i)}\), 2) the probability of observing \(F(i, U^*)\) from \(u_i\) to his labeled friends \(O_f^*(u_i)\) depends on \(u_i\)'s location \(l_{u_i}\) and the influence models of \(O_f^*(u_i)\), denoted as \(\theta_o(u_i)\), and 3) the probability of observing \(T(i, V)\) from \(u_i\) to his tweeted venues \(O_t(u_i)\) depends on \(u_i\)'s location \(l_{u_i}\) and the influence models of \(O_t(u_i)\), denoted as \(\theta_o(u_i)\).

**Likelihood Function** Given parameters \(\theta_{u_i}, l_{u_i}, l_{I_f^j(u_i)}, \theta_o(u_i)\) and \(\theta_o(u_i)\), we write the joint conditional probability (the likelihood function) of observing \(F(U^*, i)\), \(F(i, U^*)\) and \(T(i, V)\) as Eq. (4.4). At step 1, we express the joint conditional probability as the product of \(P(e(j, i)|\theta_{u_i}, l_{u_i})\) based on the conditional independence assumption. \(t(i, j)\) is multiplied \(w_{ij}\) times, as each \(t(i, j)\) appears \(w_{ij}\) times in \(T(i, V)\). At step 2, we represent \(P(e(j, i)|\theta_{u_i}, l_{u_i})\) as \(n_i\)'s influence probability at \(l_{n_j}\) based on our influence model.

\[
P(F(U^*, i), F(i, U^*), T(i, V)|l_{u_i}, \theta_{u_i}, l_{I_f^j(u_i)}, \theta_o(u_i), \theta_o(u_i))
\]

\[
= \prod_{u_j \in I_f^j(u_i)} P(f(j, i)|\theta_{u_i}, l_{u_j}) \times \prod_{u_j \in O_f^*(u_i)} P(f(j, i)|\theta_{u_j}, l_{u_j}) \times \prod_{v_j \in O_t(u_i)} P(t(i, j)|l_{u_i}, \theta_{v_j}) w_{ij}
\]

\[
= \prod_{u_j \in I_f^j(u_i)} \frac{1}{2\pi \sigma_{u_i}^2} e^{\frac{(X_{u_i} - X_{u_j})^2 + (Y_{u_i} - Y_{u_j})^2}{-2\sigma_{u_i}^2}} \times \prod_{u_j \in O_f^*(u_i)} \frac{1}{2\pi \sigma_{u_j}^2} e^{\frac{(X_{u_i} - X_{u_j})^2 + (Y_{u_i} - Y_{u_j})^2}{-2\sigma_{u_j}^2}} \times \prod_{v_j \in O_t(u_i)} \left(\frac{1}{2\pi \sigma_{v_j}^2} e^{\frac{(X_{u_i} - X_{v_j})^2 + (Y_{u_i} - Y_{v_j})^2}{-2\sigma_{v_j}^2}}\right)^{w_{ij}}
\]

(4.4)

Based on MLE, we find parameters, \(u_i\)'s location \(l_{u_i}\) and \(u_i\)'s influence scope \(\sigma_{u_i}\), by maximizing the above equation, and use the estimated \(l_{u_i}\) as \(u_i\)'s location.

However, in Eq. (4.4), besides \(l_{u_i}\) and \(\sigma_{u_i}\), which we aim to estimate, there are other unknown parameters. Particularly, for each labeled friend \(u_j \in O_f^*(u_i)\) and each tweeted venue \(v_j \in O_t(u_i)\), their influence scopes \(\sigma_{u_j}\) and \(\sigma_{v_j}\) are unknown, as we only observe their locations. In our local prediction setting, we assume each labeled node’s influence scope can be accurately estimated with its labeled neighbors as well. Thus, we estimate them before
predicting the user’s location, and view them as the known parameters. Next, we discuss how to estimate them.

**Influence Scope of a Friend** To estimate $\sigma_{u_j}$ in a labeled friend $u_j$’s influence model $\theta_{u_j}$, we can use $u_j$’s following relationships from his labeled followers. Among $u_j$’s edges, only $u_j$’s following edges $F(U, j)$ from his followers depend on $\theta_{u_j}$. As those edges also depend on his followers’ locations, we use $u_j$’s following edges $F(U^*, j)$ from his labeled followers $T^*_f(u_j)$ as observations, and estimate $\theta_{u_j}$ by maximizing the joint conditional probability of observing $F(U^*, j)$ given $\theta_{u_j}$ and $l_{T_f(u_j)}$. We write the probability as Eq. (4.5).

$$P(F(U^*, j) | \theta_{u_j}, l_{T_f(u_j)}) = \prod_{u_k \in T^*_f(u_j)} P(f(k, j) | \theta_{u_j}, l_{u_k}) \prod_{u_k \in T^*_f(u_j)} \frac{1}{2\pi \sigma^2_{u_j}} e^{-\frac{(X_{u_j} - X_{u_k})^2 + (Y_{u_j} - Y_{u_k})^2}{2\sigma^2_{u_j}}}$$ (4.5)

In Eq. (4.5), $\sigma_{u_j}$ is the only unknown variable, as $u_j$ is a labeled user and $u_k$ is his labeled follower. We directly estimate $\sigma_{u_j}$ by maximizing Eq. (4.5). Technically, we get its closed-form solution by differentiating Eq. (4.5) with respect to $\sigma_{u_j}$ and setting the result to zero. Eq. (4.6) shows the solution.

$$\sigma^2_{u_j} = \sum_{u_k \in T^*_f(u_j)} \frac{(X_{u_j} - X_{u_k})^2 + (Y_{u_j} - Y_{u_k})^2}{2|T^*_f(u_j)|}$$ (4.6)

**Influence Scope of a Venue** Similarly, to estimate a venue $v_j$’s influence scope $\sigma_{v_j}$, we use the tweeting edges from $v_j$’s labeled twitters, denoted as $T(U^*, j) = \{t(i, j) \in T|u_i \in U^*\}$. We derive $\sigma_{v_j}$ by maximizing the conditional probability of generating $T(U^*, j)$ given $v_j$’s influence model $\theta_{v_j}$ and labeled twitter’s locations $L_{T^*_f(v_j)}$. We write the condition probability as Eq. (4.7), and derive $\sigma_{v_j}$ in Eq. (4.8).
\[
P(T(U^*, j) | \theta_{v_j}, \mathcal{L}_{I^*_t(U^*)}) = \prod_{u_i \in \mathcal{I}_t(v_j)} P(t(i, j) | \theta_{v_j}, \ell_{u_i})^{w_{ij}} \quad (4.7)
\]

\[
\sigma_{v_j}^2 = \sum_{u_i \in \mathcal{I}_t(v_j)} \frac{w_{ij}((X_{u_i} - X_{v_j})^2 + (Y_{u_i} - Y_{v_j})^2)}{2 \sum_{u_i \in \mathcal{I}_t(v_j)} w_{ij}}. \quad (4.8)
\]

**Solution** Now each tweeted venue \( v_j \)'s \( \sigma_{v_j} \) and \( \ell_{v_j} \), each labeled friend \( u_j \)'s \( \ell_{u_j} \) and \( \sigma_{u_j} \), and each labeled follower \( u_j \)'s \( \ell_{u_j} \) are known. \( \ell_{u_i} \) and \( \sigma_{u_i} \) are the unknown variables left. We estimate them by maximizing Eq. 4.4. We first differentiate Eq. 4.4 with regard to \( \ell_{u_i} \) and \( \sigma_{u_i} \), and obtain Eq. 4.9 and Eq. 4.10, which show \( \ell_{u_i} \) and \( \sigma_{u_i} \) depend on each other. We substitute Eq. 4.10 for \( \sigma_{u_i} \) in Eq. 4.9, and obtain a polynomial function of \( \ell_{u_i} \). We apply the Newton–Raphson method to find its solution, and derive \( \sigma_{u_i} \) accordingly. We note that because \( X_{u_i} \) and \( Y_{u_i} \) are symmetric in Eq. 4.4, the solutions for \( X_{u_i} \) and \( Y_{u_i} \) are in the same form. Due to the space limit, we only give the solution for \( X_{u_i} \).

\[
X_{u_i} = \frac{\sum_{u_j \in \mathcal{I}_f^*(u_i)} \frac{X_{u_j}}{\sigma_{u_j}^2} + \sum_{u_j \in \mathcal{O}_f^*(u_i)} \frac{X_{u_j}}{\sigma_{u_j}^2} + \sum_{v_j \in \mathcal{O}_t(u_i)} \frac{w_{ij}X_{v_j}}{\sigma_{v_j}^2}}{\sum_{u_j \in \mathcal{I}_f^*(u_i)} \frac{1}{\sigma_{u_j}^2} + \sum_{u_j \in \mathcal{O}_f^*(u_i)} \frac{1}{\sigma_{u_j}^2} + \sum_{v_j \in \mathcal{O}_t(u_i)} \frac{w_{ij}}{\sigma_{v_j}^2}} \quad (4.9)
\]

\[
\sigma_{u_i}^2 = \sum_{u_j \in \mathcal{I}_f^*(u_i)} \frac{(X_{u_j} - X_{u_i})^2 + (Y_{u_j} - Y_{u_i})^2}{2|\mathcal{I}_f^*(u_i)|} \quad (4.10)
\]

The above solution also works for the cases that only a subset of resources (e.g., tweets) is used, as we can simply view the unused resource as an empty set in our solution.

**Interpretation** The above solution can be interpreted meaningfully. As Eq. 4.10 shows, the influence scope of \( u_i \) will be large if \( u_i \)'s followers are far away from him. Celebrities (e.g., Lady Gaga) will get large influence scopes as their followers are distributed broadly. As Eq. 4.9 shows, when we estimate a user’s location, each node contributes differently, where
the weight of a node is inversely proportional to its influence scope. E.g., if we profile a user’s location using two friends of him, e.g., Lady Gaga and a regular user, the prediction is close to the regular user, as Lady Gaga has a broad influence scope, and her location is likely to be a noisy signal.

**Computation Complexity** The algorithm computes a user’s location in $O(K^2)$, where $K$ is the average number of edges associated with a user and is less than a hundred. Specifically, it first computes influence scopes for $K$ neighbors of the user, and each of them requires $O(K)$. Then, it uses $O(tK)$ to estimate the location with $K$ edges, where $t$ is the number of iterations in the Newton method. Theoretically, $t$ is $O(d \log^2(d))$ for $d$ digits precision, which is a small constant and can be ignored. In practice, we can precompute the influence scope for each labeled node, and the complexity is reduced to $O(K)$. The algorithm can be viewed as an online algorithm, which efficiently infers a user’s location at real-time.

### 4.5.2 Global Prediction Algorithm

We further develop a global prediction algorithm, which infers a user’s location via using all the edges in the graph, and profile users’ locations more accurately than the local one.

To motivate, unlabeled users are valuable as we can propagate locations of their tweets, followers and friends to them. Let us revisit the example in Fig. Although $u_6$ is unlabeled, we can tell $u_6$ is close to Chicago as he tweets Chicago. As a result, $u_6$ becomes an additional observation, which suggests that $u_1$ should be close to Chicago. However, unlabeled users can not be directly used, because we can not tell which unlabeled user we should predict first, say, $u_1$ or $u_6$, and how to propagate a user’s predicted location to others.

We develop our global algorithm to model all the edges in the graph and utilize all the observed locations. Specifically, it models the joint conditional probability of observing all the edges $F$ and $T$ given all the nodes’ locations and influence models, and estimates all unlabeled users’ locations together via maximizing the probability.
We write the probability as Eq. (4.11). Step 1 is based on the independence assumption, and step 2 is based on our influence model.

\[
P(F, T|\theta_U, \mathcal{L}_U, \theta_V, \mathcal{L}_V) = \prod_{f(i,j) \in F} P(f(i,j)|\theta_{u_i}, l_{u_i}) \prod_{t(i,j) \in T} p(t(i,j)|\theta_{v_j}, l_{u_i})^{w_{ij}}
\]

\[
= \prod_{f(i,j) \in F} \frac{1}{2\pi \sigma^2_{u_i}} e^{-\frac{(X_{u_j} - X_{u_i})^2 + (Y_{u_j} - Y_{u_i})^2}{2\sigma^2_{u_i}}} \times \prod_{t(i,j) \in T} \left( \frac{1}{2\pi \sigma^2_{v_j}} e^{-\frac{(X_{u_i} - X_{v_j})^2 + (Y_{u_i} - Y_{v_j})^2}{2\sigma^2_{v_j}}} \right)^{w_{ij}} (4.11)
\]

In the above equation, for \( u_i \in U^N \), both \( l_{u_i} \) and \( \sigma_{u_i} \) are unknown; for \( u_i \in U^* \) and \( v_j \in V \), \( \sigma_{u_i} \) and \( \sigma_{v_j} \) are unknown. We estimate their values by maximizing the probability. To derive them, we first differentiate Eq. (4.11) with regard to every unknown variable, and obtain the following equations.

\[
X_{u_i} = \frac{\sum_{u_j \in I_f(u_i)} \frac{X_{u_j}}{\sigma^2_{u_j}} + \sum_{u_j \in \partial_f(u_i)} \frac{X_{u_j}}{\sigma^2_{u_j}} + \sum_{v_j \in \partial_t(u_i)} \frac{w_{ij}X_{v_j}}{\sigma^2_{v_j}}}{\sum_{u_j \in I_f(u_i)} 1/\sigma^2_{u_j} + \sum_{u_j \in \partial_f(u_i)} 1/\sigma^2_{u_j} + \sum_{v_j \in \partial_t(u_i)} w_{ij}/\sigma^2_{v_j}} (4.12)
\]

\[
\sigma^2_{u_i} = \sum_{u_j \in I_f(u_i)} \frac{(X_{u_j} - X_{u_i})^2 + (Y_{u_j} - Y_{u_i})^2}{2|I_f(u_i)|} (4.13)
\]

\[
\sigma^2_{v_j} = \sum_{u_i \in I_t(v_j)} \frac{w_{ij}((X_{u_i} - X_{v_j})^2 + (Y_{u_i} - Y_{v_j})^2)}{2 \sum_{u_i \in I_t(v_j)} w_{ij}} (4.14)
\]

In these equations, the unknown variables are dependent on each other. Their closed-form solutions are not easy to get. However, if we assume \( \sigma_{u_i} \) and \( \sigma_{v_j} \) for each \( u_i \in U \) and each \( v_j \in V \) are known, \( X_{u_i} \) only depends on \( X_{u_j} \in U \) and \( X_{v_j} \in V \). In this case, Eq. (4.12) tries to find \( X_{u_i} \) for each \( u_i \in U^N \) such that \( \sum_{f(i,j) \in F} 1/\sigma^2_{u_j}(X_{u_i} - X_{u_j})^2 + \sum_{t(i,j) \in T} w_{ij}/\sigma^2_{v_j}(X_{u_i} - X_{v_j})^2 \) is minimized. An iterative algorithm, which updates each \( X_{u_i} \) based on other \( X_{u_j} \) iteratively, has been proposed to find \( X_{u_i} \) for this problem [51]. When \( X_{u_i} \) and \( Y_{u_i} \) are derived, \( \sigma_{u_i} \) and \( \sigma_{v_j} \) can be derived directly based on Eq. (4.13) and (4.14).

71
Therefore, we develop a two stage iterative algorithm based on the above intuition. The algorithm is shown in Algorithm 1. At step 1-2, it initializes all $u_i \in U^N$. At step 3-14, the algorithm does the iterative computation. There are two iterations. The outer iteration updates $\sigma_{u_i}$ and $\sigma_{v_j}$ according to $l_{u_i}$ based on Eq. 4.13 and 4.14, while the inner iteration (from step 8 to 11) takes a set of fixed $\sigma_{u_i}$ and $\sigma_{v_j}$ as inputs and iteratively computes $l_{u_i}$ based on Eq. 4.12. The newly obtained $l_{u_i}$ is then used to update $\sigma_{u_i}$ and $\sigma_{v_j}$ again. The algorithm stops until the likelihood converges. We can formally prove the convergence of

Algorithm 1: Global Prediction Algorithm

| Input: $G, l_{u_i}, \forall u_i \in U^*$ |
| Output: $l_{u_i}, \forall u_i \in U^N$ |

// Initialization
1. $\forall u_i \in U^N$
2. $X_{u_i} = \text{Random}$ and $Y_{u_i} = \text{Random}$
3. Repeat //Outer Iteration
4. $\forall u_i \in U$
5. Update $\sigma_{u_i}^2$ based on Eq. (4.13)
6. $\forall v_j \in V$
7. Update $\sigma_{v_j}^2$ based on Eq. (4.14)
8. Repeat //Inner Iteration
9. $\forall u_i \in U^N$
10. Update $X_{u_i}^{n+1}$ and $Y_{u_i}^{n+1}$ based on Eq. (4.12)
11. Until converge
12. $\forall u_i \in U^N$
13. $X_{u_i} = X_{u_i}^{n+1}$, $Y_{u_i} = Y_{u_i}^{n+1}$
14. Until converge

the algorithm based on the following theorem.

**Theorem** The global prediction algorithm converges.

The proof of the theorem is derived based on the intuition of the algorithm stated above. In the inner iteration, the method can converge and yield $l_{u_i}$ that maximizes the probability with fixed $\sigma_{u_i}$ and $\sigma_{v_j}$, as shown in [60]. Second, the outer iteration directly computes $\sigma_{u_i}$ and $\sigma_{v_j}$ that maximize the probability given fixed $l_{u_i}$ computed in the previous iteration, because Eq. 4.13 and 4.14 are the closed-form solutions for maximizing the probability when a set of $l_{u_i}$ is given. In summary, each iterative step monotonically increases the probability
and the probability has a maximum value, so the algorithm must converge.

The above algorithm, like many of other iterative algorithms (e.g., EM), may converge to a local maximum. To avoid that, we can initialize the unknown variables with the values obtained from the local prediction method. The above iterative algorithm will always generate a better solution than the local one as each iteration improves the likelihood monotonically.

As each inner iteration requires $O(|E|)$ to update every user’s location, the algorithm runs in $O(t|E|)$, where $t$ is the number of iterations and $|E|$ is the number of edges of the graph. It can be viewed as an offline algorithm, which effectively profiles all users’ locations.

### 4.5.3 Incorporating Constraints

To further improve our algorithms, we utilize human knowledge as constraints in our prediction methods. To motivate, let us revisit the example in Fig. 4.1. Most of $u_1$’s followers and friends are in or close to Chicago (e.g., $u_5$, $u_3$) except one ($u_4$) in New York. Our algorithms will estimate $u_1$’s location to be near but not exactly Chicago. If we ask a human to predict $u_1$’s location, he will definitely pick a city instead of an arbitrary geo point, and he is likely to choose one from Chicago, Urbana and New York, because he knows a user usually has some friends living in the same city.

We model such human knowledge as constraints in our prediction methods. A constraint specifies the set of candidate locations when we maximize a likelihood function. There are different choices of constraints, such as a candidate must be a city or within 30 miles of a city. Particularly, we apply the following assumption as the constraint in our implementation. We assume that a user’s location must be the same as one of his friends, followers or tweeted venues. The assumption is generally valid. In our data, an incomplete crawl of Twitter, there are about 92% of users whose locations appear in their followers, friends or tweets. We note that this constraint may not be the best one. We use it to illustrate how our methods can incorporate constraints.

The constraint version of the local prediction method becomes maximizing Eq. 4.4 subject
to \{l_{u_i} \in \mathcal{L}_{I_i}(u_i) \cup \mathcal{L}_{O_i}(u_i) \cup \mathcal{L}_{O}(u_i)\}. To solve it, we can rank each candidate location \(l_{u_i}\) according to Eq. 4.4, and use the top one as the prediction.

The constraint version of the global prediction method becomes maximizing Eq. 4.11 subject to \{l_{u_i} \in \mathcal{L}_{I_i}(u_i) \cup \mathcal{L}_{O_i}(u_i) \cup \mathcal{L}_{O}(u_i)\} for any \(u_i \in U^N\). If we rank all candidate solutions, which consist of all the combinations \(l_{u_i}\) for all \(u_i \in U^N\), the complexity of the algorithm is \(O(K^N)\), where \(K\) is the average number of candidate locations per user (it is usually larger than 2), and \(N\) is the number of unlabeled users (about millions). Instead, we propose an approximation algorithm based on the relax and round paradigm, which is widely used by approximation algorithms for optimization with constraints \[10\]. We first use the global algorithm to find \(L_{u_i}'\) for each \(u_i\) without any constraint, then find the closest location \(l_{u_i}\) that satisfies the constraint.

### 4.6 Experiments

#### 4.6.1 Experiment Setup

**Data Set** We constructed our data set by crawling Twitter. We randomly selected 100,000 users as seeds to crawl in May 2011. For each user, we crawled his profile, followers and friends. We obtained 3,980,061 users’ profiles and their social network. Then, we extracted their registered locations from their profiles based on the rules described in \[14\]. Specifically, we extracted locations with city-level labels in the form of “cityName, stateName” and “cityName, stateAbbreviation,” where we considered all cities listed in the Census 2000 U.S. Gazetteer. We found 630,187 users, who provided city level locations, and treated them as labeled users. Among them, we found 158,220 users, who had at least one labeled friend or follower. We further crawled their tweets and extracted venues from those tweets based on the same gazetteer. We crawled at most 600 tweets for each user. As we could not get some users’ tweets due to their privacy settings or lack of tweets, only 139,180 users’ tweets were crawled. We used the 139,180 users with their following relationships and tweets, as
our data set. There are 14.8 friends, 14.9 followers, and 29.0 venues per user. We took their registered locations as their home locations, and applied five fold validation, which means that we used 80% of users as labeled users and 20% of users as unlabeled users and reported our results based on the average of 5 runs.

**Methods** To fully evaluate our approach, we not only compare them with two state-of-the-art methods in [7] and [14], but also evaluate our prediction algorithms with different settings. Specifically, our experiments evaluate the following methods.

- **Base** is the method developed in [7], which predicts a user’s location based on his social network. Twitter is a directional network, so we treat both followers and friends of a user as his undirected connections (“friends”) in this method.
- **Base** is the method developed in [14]. It assigns a location to a user based on a set of local words identified from his tweets.
- **UDI** is our local prediction method, but only uses a user’s friends and followers.
- **UDI** is our local prediction method, but only uses venues identified from a user’s tweets.
- **UDI** is our local prediction method discussed in Sec. 4.5.1.
- **UDI** is our global prediction method discussed in Sec. 4.5.2.

**Measurement** We use *average error distance in miles (AED)* and *accuracy within 100 miles (ACC)* proposed in [14] as measures. Specifically, let $Err(u_i)$ be the error distance between a user’s home location and an estimated location. For a set of users $U$, $AED(U)$ is $\sum_{u_i \in U} \frac{Err(u_i)}{|U|}$, and $ACC(U)$ is $\frac{|\{u_i | u_i \in U \land Err(u_i) \leq 100\}|}{|U|}$.

However, as $AED$ is easily affected by outliers in results, we report $AED$ at different percentiles (60%, 80% and 100%) of users ranked by their error distances. *E.g.*, $AED@60\%$ is the average error distance of the top 60% of users ranked by their error distances.

We use T-test to conduct *significance tests* between our methods and baseline methods. If a method passes the significant test, we make it **boldface** in result tables.
4.6.2 Experiment Results

User-based Prediction We first compare $UDI_U$ with $Base_U$. Both of them profile a user’s location based on his social network.

Tab. 4.2 shows the performance of each method. The results demonstrate that generally our method performs better than $Base_U$. When using the same amount of information, $UDI_U$ improves $Base_U$ by 4% in terms of ACC. Such an improvement soundly proves our assumption that different users have different influence scopes and we should model them discriminatively. $AED@60\%$ tells that the average error distance of the top 60% of predictions of $UDI_U$ is 20 miles, which is fairly accurate. However, when comparing $AED@80\%$ and $AED@100\%$, we find that $AED$ dramatically increases from 159 to 525, because $AED$ is easily affected by a small set of users, who are not accurately predicted. Therefore, we
Table 4.2: Prediction Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Base_U</th>
<th>Base_C</th>
<th>UDI_U</th>
<th>UDI_C</th>
<th>UDI_L</th>
<th>UDI_G</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>52.4%</td>
<td>49.7%</td>
<td>56.0%</td>
<td>60.0%</td>
<td>64.4%</td>
<td>65.9%</td>
</tr>
<tr>
<td>AED@60%</td>
<td>33.7</td>
<td>21.8</td>
<td>20.6</td>
<td>9.5</td>
<td>6.6</td>
<td>4.4</td>
</tr>
<tr>
<td>AED@80%</td>
<td>200.0</td>
<td>161.5</td>
<td>159.6</td>
<td>123.6</td>
<td>97.0</td>
<td>75.0</td>
</tr>
<tr>
<td>AED@100%</td>
<td>616.9</td>
<td>542.5</td>
<td>524.5</td>
<td>483.6</td>
<td>440.4</td>
<td>421.3</td>
</tr>
</tbody>
</table>

should not only focus on AED@100%.

To illustrate our results in detail, we plot an *accumulative accuracy at distances* (AAD) curve for each method in Fig. 4.4(a). A point \((X,Y)\) in the curve means that \(Y\) percentages of users are accurate within \(X\) miles. From the figure, we can tell that \(UDI_U\) has higher accuracy than \(Base_U\) within different distances. E.g., \(UDI_U\) places about 47% of users within 25 miles, while \(Base_U\) only places 44% of users within that range.

**Content-based Prediction** In this experiment, we compare \(UDI_C\) with \(Base_C\). Both of them profile a user’s location with his tweets.

We show results and AAD curves of two methods in Tab. 4.2 and Fig. 4.4(b) respectively. From them, we can see that 1) \(UDI_C\) significantly improves \(Base_C\) by 10% in terms of ACC, 2) the improvement is consistent at any distance level, and 3) \(UDI_C\) achieves very good results by making good use of content. The average error distance for the top 60% of its prediction is less than 10 miles. From the results, we can safely conclude that our method is much better than \(Base_C\) as our model captures the relation between a user’ location and locations from his tweets in a meaningful way.

**Integrated vs. Non-Integrated** In this experiment, we evaluate whether our framework can take advantage of integrating more resources. Specifically, we compare \(UDI_L\) with \(Base_U\), \(Base_C\), \(UDI_C\) and \(UDI_U\). Tab. 4.2 shows the performance of each method. As expected, \(UDI_L\) gives a significant improvement (12%) over the best baseline method, and advances \(UDI_C\) and \(UDI_U\) by 4.4% and 8.4%. Fig. 4.4(c) shows that those improvements are consistent at any distance level. We can safely conclude that integrating different types of resources is useful for profiling locations. Meanwhile, we can find that \(UDI_L\) is very accurate.
Table 4.3: Local vs. Global with 80% Test Users

<table>
<thead>
<tr>
<th>Model</th>
<th>Base_U</th>
<th>Base_C</th>
<th>UDI_L</th>
<th>UDI_G</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>34.0%</td>
<td>42.4%</td>
<td>57.0%</td>
<td>66.0%</td>
</tr>
<tr>
<td>AED@60</td>
<td>116.9</td>
<td>60.9</td>
<td>11.7</td>
<td>4.3</td>
</tr>
<tr>
<td>AED@80</td>
<td>347.7</td>
<td>259.3</td>
<td>133.9</td>
<td>71.6</td>
</tr>
<tr>
<td>AED@100</td>
<td>897.4</td>
<td>679.9</td>
<td>514.1</td>
<td>415.3</td>
</tr>
</tbody>
</table>

It correctly places 57% of users within 25 miles. Its AED is only 6 miles for the top 60% of its predictions, and less than 100 miles for the top 80%.

**Global vs. Local** To investigate the usefulness of our global prediction method, we compare $UDI_G$ with $UDI_L$ and the two baselines.

We evaluate the methods on the data set used in the previous experiments, which includes 20% unlabeled and 80% labeled ones. The last column in Tab. 4.2 gives the results of $UDI_G$. We can see that, although $UDI_G$ improves $UDI_L$ slightly (1.5%) in terms of ACC, it reduces $AED@80%$ a lot, and Fig. 4.4(d) shows that the improvement is consistent at any level. We believe that the improvement here is limited because there is already enough information from the labeled users and the iterative based method can not add much help. We expect that $UDI_G$ improves $UDI_L$ in a more realistic scenario, where less users are labeled.

To test this conjecture, we evaluate those methods in another data set, where only 20% of users are labeled and 80% users are unlabeled. This scenario is more close to the real-world case, where only about 16% users have registered locations. Tab. 4.3 shows the results. We find that 1) $UDI_G$ significantly outperforms the other three methods, as it can utilize information from even unlabeled users, 2) compared to the preceding experiment, $UDI_G$ achieves nearly comparable results, but the other three methods perform much worse, as they make predictions with limited amount of information. We can conclude that $UDI_G$ utilizes both labeled and unlabeled information and achieves better profiling.

We evaluate $UDI_G$ for its convergence, and find it takes 3 outer iterations to converge. Due to space limit, the figure is omitted.

**Case Studies** We give some concrete examples of influence scopes derived by our methods to
Table 4.4: Case Studies

<table>
<thead>
<tr>
<th>Users</th>
<th>Follower No.</th>
<th>σ</th>
<th>Cities</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MythBusters Official</td>
<td>860688</td>
<td>1.127</td>
<td>Honolulu</td>
<td>0.970</td>
</tr>
<tr>
<td>Lady Gaga</td>
<td>18428360</td>
<td>0.633</td>
<td>San Francisco</td>
<td>0.582</td>
</tr>
<tr>
<td>National Geographic</td>
<td>162870</td>
<td>0.655</td>
<td>New York</td>
<td>0.551</td>
</tr>
<tr>
<td>NY Knicks</td>
<td>178297</td>
<td>0.172</td>
<td>Austin</td>
<td>0.11</td>
</tr>
<tr>
<td>Philadelphia 76ers</td>
<td>62210</td>
<td>0.161</td>
<td>Houston</td>
<td>0.12</td>
</tr>
<tr>
<td>timpawlenty</td>
<td>63896</td>
<td>0.239</td>
<td>Dallas</td>
<td>0.14</td>
</tr>
</tbody>
</table>

illustrate their correctness and usefulness. Tab. 4.4 shows influence scopes of some Twitter users and venues. For easy understanding, we only choose verified users (celebrities). In Tab. 4.4, we can clearly distinguish local authorities (e.g., “timpawlenty”, a former governor of Minnesota), and national celebrities (e.g., “Lady Gaga”). We note that we cannot easily tell the difference between “national graphic” and “NY Knicks” just by the numbers of their followers. Similarly, our methods identify that Honolulu, a famous vacation destination, has a broad influence scope and is likely to be a noisy signal.
Chapter 5

User Profiling in Social Media: A Probabilistic Approach for Profiling Users’ Multiple Locations

5.1 Introduction

In this chapter, we continue focusing on user profiling in social media. We aim to profile users’ multiple value attribute in social media completely. Particularly, we are interested in profiling “multiple locations” for Twitter users with both their tweets and their following connections.

While, in the literature, some methods \([14, 7]\) have been proposed to profile a user’s home locations, the single “permanent” resident location of the user, by exploring the users’ social network and tweets, these methods have the same shortcoming – they assume that a user has only a “home location”. In reality, as illustrated in Fig. 5.1, a user \((e.g., \text{Carol})\) is related to multiple locations, such as her home location \((e.g., \text{Los Angeles})\) and college location \((e.g., \text{Austin})\). She follows friends from and tweets “venues” \((i.e., \text{location words mentioned in her tweets})\) about all of them. \(E.g., \text{Carol performs her classmate Lucy in Austin and her co-worker Bob in Los Angeles. Thus, these methods not only profile a user’s locations incompletely, but also estimate her home location inaccurately, because signals related to her other locations are noises for profiling her home location.}\)

Thus, we aim to build complete “location profiles” for Twitter users with their following network and tweets. We define a user’s \((e.g., \text{Carol})\) location profile as a set of locations related to her \((e.g., \{\text{Los Angeles, Austin}\})\). It includes not only her home location \((e.g., \text{Los Angeles})\) but also her other related locations \((e.g., \text{Austin})\). Further, we clarify that each user related location is 1) a geo scope \((e.g., \text{Los Angeles})\) instead of a geo point \((e.g., \text{the}})
Starbucks on 5th Ave.), and 2) a long-term location instead of a temporarily related location (e.g., the places where he is traveling). Thus, a user’s location profile captures her multiple long-term geographic scopes of interests. We emphasize that we only use users’ following network and tweets, and do not use GPS tags because they are rarely available as we just mentioned. Thus, we avoid the need for private information (e.g., IP address) and enable third-party services (e.g., researchers) to profile users’ locations with Twitter APIs.

In addition, for each relationship (e.g., the following relationship from Carol to Lucy), we aim to profile users’ specific locations underlying the relationship (e.g., Carol follows Lucy as they studied in Austin), because a user has multiple locations of interests and each of her relationships can be a result of any of her locations. Profiling locations for each relationship not only helps us to discover users’ locations accurately and completely, but also enables interesting applications, such as understanding the true geo connection between two users and grouping a user’s friends into geo groups (e.g., Carol is in Lucy’s Austin group).

Thus, we propose a multiple location profiling model (MLoc) for users and their relationships. To the best of our knowledge, MLoc is the first model that 1) discovers users’ multiple locations and 2) profiles both users and their relationships.

Specifically, MLoc takes a generative probabilistic approach and models the joint probability of generating “following” and “tweeting” relationships based on users’ multiple loca-
tions. With the joint probability, we estimate users’ locations and locations of relationships as latent variables in the probability. However, when modeling the joint probability, MLoc must deal with the following challenges.

**Location-based Generation** To connect users’ locations with observed relationships, MLoc needs to formally model the probability that a relationship is generated based on users’ locations. Specifically, it should capture that a user at a specific location 1) follows her friends from different locations or tweets different venues, and 2) is likely to follow users living close to her or tweet her nearby venues.

We investigate the connections between the two types of relationships and users’ locations on a Twitter data and derive a location-based generative model for each type of relationships. For the “following probability” based on two user’s locations, we explore the probability based on their distance, and formally model the probabilities over distances as a power law distribution. For the “tweeting probability” based on one user’s location, we view locations and venues as discrete labels, and formally model the probabilities of tweeting different venues at each location as a multinomial distribution over a set of venues.

**Mixture of Observations** We can not straightforwardly use observed relationships to build a user’ location profile, because of two challenges: 1) the noisy-signal challenge, which means she may follow friends (e.g., Lady Gaga) and tweet venues (e.g., Honolulu) that are not based on her locations, 2) the mixed-signal challenge, which means she follows friends (e.g., Lucy and Bob) or tweets venues based on her multiple locations. We introduce two mixtures in MLoc to deal with the two challenges.

With respect to the noisy-signal challenge, we model relationships as a mixture of “noisy” and “location-based” relationships. Specifically, we introduce a random generative model to model how a noisy relationship is generated randomly, besides the location-based generative model introduced above. Each relationship is generated by either of the two models with a certain probability. Thus, MLoc explicitly captures noisy relationships, and automatically rules out them when profiling users’ locations.
With respect to the mixed-signal challenge, we extend the location-based generative models to generate relationships based on users’ multiple locations. Specifically, we view a user’s location profile as a multinomial distribution over a set of locations, and extend the models to generate a location-based relationship in two steps: 1) generate a location assignment from each related user’s location profile, and 2) generate the relationship based on the assignments. Thus, MLoc fundamentally captures that a user has multiple locations. It not only discovers her multiple locations completely, but also estimates her home location accurately. Further, MLoc reveals the true geo connection in a relationship with the location assignments for the relationship.

**Partially Available Supervision** As we mentioned that some users provide their home locations, those locations are the only observed locations and crucial for accurate profiling. However, they are difficult to use, because we can neither view them as users’ location profiles, as a profile should contain more than a home location, nor use them to generate relationships because of the mixed-signal challenge.

We incorporate the observed home locations as prior knowledge to generate users’ location profiles. Specifically, we assume that a user’s location profile is generated via a prior distribution with a hyper parameter, and use the observed locations to set the hyper parameter for each user. As a result, for a user with an observed location, her derived location profile has a large probability to generate the observed location, and her relationships are likely to be generated based on the location as well.

Based on MLoc, we profile users and their relationships as estimating the latent variables in the joint probability. However, as MLoc models the above new aspects and integrates discrete (multinomial) and continuous (power low) distributions, it does not allow exact inference. We derive an efficient sampling-based algorithm based on the Gibbs sampling framework to estimate the latent variables.

To evaluate MLoc, we conduct extensive experiments and compare MLoc with the state-of-the-art methods [7, 14] on a large-scale Twitter data containing about 160K users. The
results show that MLoc is effective. Specifically, 1) for predicting users’ home locations, MLoc largely improves the baseline methods by 10% and places 62% users accurately; 2) for discovering users’ multiple locations, MLoc captures users’ multiple locations accurately and completely, and improves the baseline methods by 11% and 14% in terms of “precision” and “recall”; 3) for explaining following relationships, MLoc achieves 57% accuracy.

5.2 Related Work

In this section, we discuss some related work. In terms of the problem, our work is related to user profiling. In terms of the technique, our work is related to collective classification and mixture models. As we have reviewed the location prediction related work in the previous chapter, we focus on the related techniques.

**Collective Classification** As we aim to assign users (nodes) in a social network to locations (labels), our work is related to collective classification techniques [10, 29], which classify nodes in a network setting. For example, in [31], the authors take a local consistent assumption that a node’s label is likely to be the same as its neighbors, and derive a voting-based neighborhood classifier. In [50], the authors apply a Markov dependency assumption that the label of one node depends on its neighbors’ labels, and develop a pairwise Markov random field model. However, those methods may fail in our setting because of two reasons. First, as Sec. 4.2 discussed, they fail to utilize distances between location labels to profile users location inaccurately. Second, most of collective classification methods (e.g., [50]) make implicit assumptions that 1) a node has one label (location), and 2) all of its relationships are related to the label (location). However, in reality, a user is often related to multiple labels (locations) and the relationships are only related to one of labels. Thus, they fail to address the mixed-signal challenge and profile users’ locations inaccurately and incompletely.

**Mixture Model** As we aim to model observations (i.e., relationships and tweeted venues) as generated by a mixture of hidden variables (i.e., locations), MLoc works in a similar way
as Latent Dirichlet Allocation (LDA) \cite{11} and Mixed Membership Stochastic Blockmodels (MMSB) \cite{3}.

LDA and its various extensions \cite{11,54} model a text collection as a mixture over a set of hidden topics. There are clear distinctions between MLoc and LDA. First, MLoc models locations instead of topics as the variables. Locations are predefined attributes, which can be observed from some users and have explicit correlation, while topics are loosely defined “clusters” of tokens, which are hidden in documents. In order to classify users into location labels, MLoc explores distances between locations and utilizes observed locations from some users as supervision. Second, MLoc models following relationships in addition to content (tweeted venues), as observations. We introduce a new generative process and a new probabilistic distribution (power law) to model them.

MMSB and its extensions \cite{35} explicitly model how relationships (e.g., citations) are generated based on a mixture of nodes’ communities (e.g., papers’ topics). As communities are also loosely defined clusters, MLoc is different from it by the first reason mentioned above. Furthermore, MLoc advances MMSB in modeling relationships as well. MMSB assumes that a relationship between two nodes is generated based on pairwise interactions of their communities, while MLoc explicitly explores the correlations between locations and introduces a power law distribution over distances to parameterize pairwise location interactions. As a result, we greatly eliminate the number of parameters and explicitly capture that users in a following relationship are likely to live close.

5.3 Problem Abstraction

In this section, we first introduce Twitter, and then abstract our problem from there.

As illustrated by Fig 5.1, a user $u_i$ (e.g., Carol) in Twitter connects to two types of resources, 1) her following network, which is a set of users (e.g., Bob and Lucy), who follow or are followed by the user, and 2) her tweeting content, which is a set of messages tweeted
Every \( u_i \) is related to a set of locations, which is \( u_i \)'s location profile, denoted as \( L_{u_i} \). \( L_{u_i} \) contains \( u_i \)'s home location (e.g., Bob's home location San Diego), denoted as \( l_{u_i} \), and other related locations. Our goal is to build the location profile for each user, and we are interested in profiling their city-level locations specifically. All possible city-level locations can be given by a gazetteer, which can be easily obtained from various online resources (e.g., Geographic Names Information System). We name them as candidate locations, and use \( L \) to denote them. Further, some users' home locations are observed. We call them as labeled users, denoted as \( U^* \), and the remaining users as unlabeled users, denoted as \( U^N \). We use \( U \) to denote all the users, where \( U = U^* \cup U^N \).

As mentioned in Sec. 5.1, both types of resources are useful for profiling a user’s locations, because a user (e.g., Carol) is likely to 1) follow and be followed by users (e.g., Mike and Bob), who live close to her, and 2) tweet some “venue names” (e.g., Los Angeles or Hollywood), which may indicate her locations. Here, we refer a venue name as the name for a geo signal, which could be a city (e.g., Los Angeles), a place (e.g., Time Square), or a local entity (e.g., Stanford University). In the rest of the chapter, we use “venue” for short. We note that a venue may refer different locations. E.g., there are 19 towns named as “Princeton” in the States.

We formally abstract the two types of resources as “following” and “tweeting” relationships. A following relationship, denoted as \( f(i, j) \), is formed from a user \( u_i \) to another user \( u_j \) when \( u_i \) follows \( u_j \). \( u_i \) is named as a follower of \( u_j \), and \( u_j \) is named as a friend of \( u_i \). We use \( f_{1:S} \) to represent all the following relationships, where \( S \) is the total number of the relationships. A tweeting relationship \( t(i, j) \) is formed from a user \( u_i \) to a venue \( v_j \), if \( u_i \) tweets \( v_j \). As \( u_i \) can tweet \( v_j \) many times, there could be many tweeting relationships between \( u_i \) and \( v_j \). We use \( t_{1:K} \) to represent all the tweeting relationships, where \( K \) is the total number of the relationships.

Further, we assume a relationship is associated with the location assignments that the relationship is based on. Specifically, for \( f(i, j) \), the location assignments \( x_i \) and \( y_j \) indicate
that $u_i$ follows $u_j$ as $u_i$ and $u_j$ are in $x_i$ and $y_j$, respectively. E.g., Austin is the location assignments for both Carol and Lucy for their following relationship, which indicates that Carol follows Lucy as they were classmates in Austin. Similarly, for $t\langle i, j \rangle$ (e.g., Carol tweets about “Hollywood”), the location assignment $z_i$ (Los Angeles) indicates $u_i$ (e.g., Carol) tweets $v_j$ (e.g., “Hollywood”) because $u_i$ is interested in $z_i$. However, as a user’s relationship could be related to any of her locations and its assignments are hidden to us, we need to profile its assignments.

Based on the above definitions, we formally abstract our problem as follows:

**User and Relationship Location Profiling** Given a set of users $U$, which contains both labeled users $U^*$ and unlabeled users $U^N$, the home location $l_{u_i}$ for $u_i \in U^*$, their following and tweeting relationships $f_{1:S}$ and $t_{1:K}$, and candidate locations $L$, estimate a set of locations $\hat{L}_{u_i} \subset L$ for $u_i \in U$, location assignments $\hat{x}_i \in \hat{L}_{u_i}$ and $\hat{y}_j \in \hat{L}_{u_j}$ for $f(i, j) \in f_{1:R}$, and a location assignment $\hat{z}_i \in \hat{L}_{u_i}$ for $t\langle i, j \rangle \in t_{1:K}$, so as to make $\hat{L}_{u_i}$, $\hat{x}_i$, $\hat{y}_j$ and $\hat{z}_i$ close to $u_i$’s location profile $L_{u_i}$ and the true assignments $x_i$, $y_j$ and $z_i$ respectively.

### 5.4 Multiple Location Profiling

In this section, we develop MLoc to profile locations for both users and their relationships with the following network and the tweeting content.

Our first goal is to connect the two types of relationships with users’ locations. Intuitively, we can assume that both of them are “generated” based on a same set of latent variables — users’ locations. Then, it naturally leads us to a probabilistic generative approach, which models the joint probability of generating the two types of relationships based on users’ locations. We can estimate users’ locations and location assignments for relationships as the latent variables in the probability.

However, as we have motivated in Sec. 5.1 and 5.2, to model the joint probability, we need to address the challenges of location-based generation, mixture of observations and partially
available supervision, which have not been studied by the existing generative models like LDA and MMSB.

We propose MLoc to model the joint probability and deal with those challenges. Fig. 5.2 shows its plate diagram and Tab. 5.1 gives notations. Generally, it illustrates how MLoc models the joint probability that 1) generates each user $u_i$’s location distribution $\theta_i$ based on a hyper distribution with a parameter $\gamma_i$, which is determined by the observed locations from the labeled users, 2) generates location assignments (e.g., $x_{s,i}$ and $z_{k,i}$) based on $\theta_i$, and 3) generates the associated following and tweeting relationships (e.g., $f_{s}(i,j)$ and $t_{k}(i,j)$) based on the location assignments. Thus, we can estimate $\theta_i$, $x_{s,i}$, $y_{s,j}$ and $z_{k,i}$ with the observed relationships and locations, and use $\theta_i$ as $u_i$’s location profile.

In the following parts, we first explain three key components of MLoc, which deals with the above challenges, and then present MLoc and its inference algorithm in detail.

5.4.1 Location-based Generation

We first present our location-based generative models, which formally measure the probability that a following or tweeting relationship (e.g., $f(i,j)$ or $t(i,j)$) is generated given users’
Table 5.1: Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Total number of users</td>
</tr>
<tr>
<td>$L$</td>
<td>All the candidate locations</td>
</tr>
<tr>
<td>$V$</td>
<td>All the venue names</td>
</tr>
<tr>
<td>$\vec{\eta}_i$</td>
<td>Observation vector for $u_i$</td>
</tr>
<tr>
<td>$\vec{\lambda}_i$</td>
<td>Candidacy vector for $u_i$</td>
</tr>
<tr>
<td>$b_o, b_c$</td>
<td>Bernoulli distributions that generate $\vec{\eta}_i$ and $\vec{\lambda}_i$</td>
</tr>
<tr>
<td>$\Lambda$</td>
<td>Boosting matrix</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Prior for candidate locations</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>Location profile of $u_i$</td>
</tr>
<tr>
<td>$\theta_{1:N}$</td>
<td>Location profiles for $N$ users</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>General prior distribution parameter for $\theta_i$</td>
</tr>
<tr>
<td>$\gamma_i$</td>
<td>Prior distribution parameter for $\theta_i$</td>
</tr>
<tr>
<td>$F_L, T_L$</td>
<td>Location-based following and tweeting models</td>
</tr>
<tr>
<td>$\alpha, \beta$</td>
<td>Parameters of $F_L$</td>
</tr>
<tr>
<td>$\psi_l$</td>
<td>Location-based tweeting model of $l$</td>
</tr>
<tr>
<td>$\psi_{1:L}$</td>
<td>Location-based tweeting models for $L$</td>
</tr>
<tr>
<td>$T_R, F_R$</td>
<td>Random tweeting and following models</td>
</tr>
<tr>
<td>$S$</td>
<td>Total number of following relationships</td>
</tr>
<tr>
<td>$f_{1:S}$</td>
<td>All the following relationships</td>
</tr>
<tr>
<td>$f_s(i,j)$</td>
<td>$s^{th}$ following relationship from $u_i$ to $u_j$</td>
</tr>
<tr>
<td>$\mu_s$</td>
<td>Model selector for $f_s(i,j)$</td>
</tr>
<tr>
<td>$\mu_{1:S}$</td>
<td>Model selectors for $f_{1:S}$</td>
</tr>
<tr>
<td>$x_{s,i}$</td>
<td>Location assignment for $u_i$ in $f_s(i,j)$</td>
</tr>
<tr>
<td>$y_{s,j}$</td>
<td>Location assignment for $u_j$ in $f_s(i,j)$</td>
</tr>
<tr>
<td>$x_{1:S}$</td>
<td>Location assignments for followers in $f_{1:S}$</td>
</tr>
<tr>
<td>$y_{1:S}$</td>
<td>Location assignments for friends in $f_{1:S}$</td>
</tr>
<tr>
<td>$K$</td>
<td>Total number of tweeting relationships</td>
</tr>
<tr>
<td>$t_{1:K}$</td>
<td>All the tweeting relationships</td>
</tr>
<tr>
<td>$t_k(i,j)$</td>
<td>$k^{th}$ tweeting relationship from $u_i$ to $u_j$</td>
</tr>
<tr>
<td>$\nu_k$</td>
<td>Model selector for $t_k(i,j)$</td>
</tr>
<tr>
<td>$\nu_{1:K}$</td>
<td>Model selectors for $t_{1:K}$</td>
</tr>
<tr>
<td>$z_{k,i}$</td>
<td>Location assignment for $u_i$ in $t_k(i,j)$</td>
</tr>
<tr>
<td>$z_{1:K}$</td>
<td>Location assignments for users in $t_{1:K}$</td>
</tr>
</tbody>
</table>
locations (e.g., $x_i$, $y_j$ or $z_i$). In Fig. 5.2, they are represented by $F_L$ and $T_L$.

The models should be carefully designed, as a user follows friends from different locations and tweets different venues. Fortunately, locations are predefined semantic attributes, and we observe locations and relationships of some users. Thus, we investigate a large-scale Twitter data (Sec. 5.3 gives the statistics of the data), and learn the models from there.

![Graph showing following probabilities versus distances.](image)

(a) Following Probabilities versus Distances

![Bar chart showing tweeting probabilities of 10 venues.](image)

(b) Tweeting Probabilities of 10 Venues at Austin and Los Angeles

![Table showing relationships as a mixture of a user's locations.](image)

(c) Relationships as a Mixture of a User’s Locations

**Figure 5.3: Observations**

**Location-based Following Model** We begin with investigating the following probability of observing a following relationship $f(i,j)$ from a user $u_i$ to a user $u_j$ given their locations $x_i$ and $y_j$. It involves two locations. If we view any pair of locations as simply two distinct categorical labels, we overlook the inherent relation between them. Thus, we explore the probability as a function of distance, since the distance is a natural and fine-grained measure for the relation between two locations.

Fig. 5.3(a) shows following probabilities over distances. We compute the distance between any pair of labeled users, resulting about $2.5 \times 10^{10}$ pairs. Then we bucket them by intervals of 1 mile and measure the probability of generating a following relationship at $d$ miles as the ratio of the number of pairs that have following relationships to the total number of pairs in the $d^{th}$ bucket. We plot the probabilities versus distances in the log-log scale.

The figure shows that 1) the following probability decreases as the distance increases, and 2) at the distances in a long range, the probabilities do not decay as sharply as those
at the distances in a short range. Such probabilities successfully capture our intuition that a user is likely to follow friends, who live close to him, but also may follow some users, who live far away. When he follows the users living far away, the following probabilities are less sensitive to his distances to them.

We can fit the probabilities in Fig. 5.3(a) with a power law distribution, as power laws are straight lines when they are plotted in the log-log scale. Mathematically, a power law distribution has two parameters, $\alpha$ and $\beta$, and the probability at a point $x$ is expressed as $P(x|\alpha, \beta) = \beta x^\alpha$. Given a set of observations, i.e., $x$ and $P(x|\alpha, \beta)$, we can learn $\alpha$ and $\beta$. In our case, $\alpha = -0.55$ and $\beta = 0.0045$.

Now, we formally describe our location-based following model. We model the following probabilities of whether there is $f\langle i, j \rangle$ from $u_i$ to $u_j$ given $x_i$ and $y_j$ as a Bernoulli distribution with a parameter $p$, and model $p$ at different distances $d(x_i, y_j)$ as a power law distribution with parameters $\alpha$ and $\beta$. Mathematically, we measure it as follows.

$$P(f\langle i, j \rangle|\alpha, \beta, x_i, y_j) = \beta d(x_i, y_j)^\alpha \quad (5.1)$$

We note that similar power law distributions have been observed in Facebook data [7] and other social networks [28], but this paper is the first study on Twitter and gives new observations. Specifically, the exponent is -0.55, which is different from -1 observed in the Facebook data [7]. It suggests that the following relationships on Twitter are less sensitive to users’ distances than the friendships in Facebook. Therefore, profiling locations for Twitter users is more difficult than for Facebook users studied in [7]. It requires us to utilize additional resources and build an advanced model.

**Location-based Tweeting Model** Next, we explore the tweeting probability that a tweeting relationship $t\langle i, j \rangle$ is generated from a user $u_i$ to a venue $v_j$ given $u_i$’s location $z_i$. As a venue name (e.g., Princeton) may refer different locations (e.g., Princeton, NJ or Princeton, WV), we cannot view it as a single location. Thus, we view venues as categorical labels and
explore tweeting probabilities at a specific location as a discrete distribution over venues \( V \).

Fig. 5.3(b) shows the tweeting probability of 10 venues by the users at Austin and Los Angeles. To generate Fig. 5.3(b), we first extract venues (city names) from users’ tweets. Then, for each location, say Austin, we count the relative frequencies of the venues, and thus the probabilities, that the venues are tweeted by those users at the location. Due to the space limit, we only select the top five venues with the largest probabilities from each location, and plot their probabilities in the log scale.

We obtain the following observations. The tweeting probabilities of different locations are different over the same venues. E.g., users in Los Angeles are more likely to tweet “los angeles” than those in Austin. For tweeting probabilities at a location (e.g., Austin), we see that 1) nearby venues (e.g., “austin”) have high probabilities to be tweeted, 2) faraway venues (e.g., “hollywood”) have small probabilities to be tweeted, and 3) the probability to tweet a venue is not a monotonic function of its distance to the location. E.g., “hollywood” and “round rock” have similar probabilities to be tweeted by users in Austin, but Round Rock city is much closer than Hollywood. The tweeting probabilities so observed do reflect that users are likely to tweet their local venues as well as far but popular venues.

We develop our location-based tweeting model to capture the above observations. Specifically, for a location \( l \), we use a multinomial distribution \( \psi_l \) over venues \( V \) to model the tweeting probabilities of \( l \). \( V \) can be defined based on a gazetteer. Each \( l \) is associated with its own \( \psi_l \), and there are totally \( |L| \) multinomial distributions, denoted as \( \psi_{1:L} \). We measure the tweeting probability that \( u_i \) builds \( t(i,j) \) to \( v_j \) given \( z_i \) as the probability of picking \( v_j \) from \( \psi_{z_i} \). Mathematically, it is measured as follows.

\[
P(t(i,j)|\psi_{1:L}, z_i) = P(v_j|\psi_{z_i}). \tag{5.2}
\]

The above distributions are obtained based on the locations provided by labeled users. The parameters (e.g., \( \alpha \) and \( \beta \)) in those distributions may not be precisely learned due to the
noisy-signal and mixed-signal challenges, which will be discussed next. However, we believe the observations are reliable for choosing proper distributions to model the two probabilities. We can further precisely estimate those parameters as we will show in Sec. 5.4.5.

5.4.2 Mixture of Observations

To fundamentally deal with the noisy-signal and mixed-signal challenges motivated in Sec. 5.1, we introduce two level mixture components in MLoc. The first level aims to capture that there are both “noisy” and “location-based” relationships, and the second level aims to address that the “location-based” relationships are related to users’ multiple locations.

The Noisy-signal Challenge First, we argue that some relationships are not generated based on locations, and therefore are noises for profiling users’ locations. E.g., Carol in Austin follows Gaga in New York. We call those relationships as noisy relationships, and the remaining ones as location-based relationships. The previous methods [14, 7] do not model noisy relationships explicitly, and can not profile users’ locations accurately.

We propose a mixture component to capture noisy and location-based relationships. Conceptually, we assume a relationship is generated based on either a location-based generative model, which is introduced above, or a random generative model, which we will introduce below. Technically, for each following relationship, we introduce a binary model selector $\mu$, where $\mu = 1$ means the random generative model is selected to generate the relationship, and 0 otherwise. We further assume that $\mu$ is generated based on a Bernoulli distribution with a parameter $\rho_f$, which models how likely a following relationship is generated based on the random generative model. Similarly, for a tweeting relationship, we introduce a model selector $\nu$ and a Bernoulli distribution with a parameter $\rho_t$ to generate $\nu$.

We now design the random generative models. Intuitively, we model the random following model, denoted as $F_R$, as a Bernoulli distribution to represent the probabilities of whether a following relationship is randomly built between two users. We model the random
tweeting model, denoted as $T_R$, as a multinomial distribution over venues $V$ to represent the probabilities that a tweeting relationship is randomly built to venues from a user.

Similar to existing work [33], we learn $F_R$ and $T_R$ empirically. Specifically, we model $F_R$, which measures the probability that $u_i$ randomly builds $f(i, j)$ to $u_j$, as $p(f(i, j) = 1 | F_R) = \frac{S}{N^2}$, where $S$ is the number of following relationships and $N^2$ is the total number of user pairs. We model $T_R$, which measures the probability that $u_i$ randomly builds $t(i, j)$ to $v_j$, as $p(t(i, j) | T_R) = \frac{\sum_{x \in U} t(x, j)}{K}$, where $\sum_{x \in U} t(x, j)$ is the number of tweeting relationships to $v_j$, and $K$ the total number of tweeting relationships.

The Mixed-signal Challenge Next, we argue that the location-based relationships are generated based on users’ multiple locations. To illustrate, we give an example of the user with id 13069282. From the user’s home page in her Twitter profile, we know that she used to study in Austin and now works in Los Angeles. Fig. 5.3(c) shows her friends’ locations, tweeted venues, as well as a map with her friends’ locations plotted. The figure clearly shows that her friends are in and her tweets are about the two regions, and suggests that a user follows friends from or tweet venues related to his multiple locations.

The previous methods [14, 7] haven’t addressed this issue. They not only profile a user’s locations incompletely, but also predict the home location incorrectly, because locations of the friends related to her other locations (e.g., Austin) are noisy information to profile her home location (e.g., Los Angeles). Although the our model can handle noises somehow, a lot of friends at great distances are “noisy” enough to make our model fail.

To fundamentally deal with the mixed-signal challenge, we first model a user $u_i$’s location profile as a multinomial distribution over candidate locations $L$, denoted as $\theta_i$. The probability of a location $l$ in $\theta_i$ represents how likely $u_i$ is at $l$. We aim to estimate $\theta_i$ for each $u_i$. We assume that a location-based relationship is generated based on a specific location assignment picked from each related user’s profile, rather than their home locations only.

Thus, we extend our location-based models into two stage generative processes. Specifically, the location-based following process models that a location-based following relationship
\( f(\langle i, j \rangle) \) from \( u_i \) to \( u_j \) is generated via the following two steps: 1) randomly select two location assignments \( x_i \) and \( y_j \) from \( \theta_i \) and \( \theta_j \), and 2) randomly generate \( f(\langle i, j \rangle) \) based on the location-based following model \( F_L \), specifically, \( P(f(\langle i, j \rangle) | x_i, y_j, \alpha, \beta) \). Similarly, the location-based tweeting process models that a tweeting relationship \( t(\langle i, j \rangle) \) from \( u_i \) to \( u_j \) is generated via the following two steps: 1) random select a location \( z_i \) from \( \theta_i \), and 2) randomly generate \( t(\langle i, j \rangle) \) based on the location-based tweeting model \( T_L \), specifically, \( P(t(\langle i, j \rangle) | z_i, \phi_{z_i}) \).

We note that the location assignments for a relationship explain the true geo connection in the relationship in terms of users’ hidden locations rather than users’ home locations only, and thus help us to fundamentally capture that a user’s relationships are generated based on her multiple locations.

5.4.3 Partially Available Supervision

To incorporate home locations from labeled users as supervision, we further model how a user’s location profile \( \theta_i \) is generated by a prior distribution with a particularly derived parameter, denoted as \( \gamma_i \) in the plate diagram.

First, we motivate the need for supervision. By far, our model runs in an “unsupervised” way as LDA and MMSB. It assumes that relationships are generated based on users’ location profiles, and can estimate them with the relationships. It neither models nor requires that locations of some users are observed. However, without an “anchoring” point, which is known somehow, the hidden clusters of “near locations” would be floating. For example, given a set of densely connected users, our model can tell that they are likely in a location, but can not identify which location (e.g., Los Angeles or Austin) they are in. In reality, 16% Twitter users provide their home locations. If our model captures some of the users in the example are in Los Angeles, it can accurately learn location profiles for all of them.

However, there is no obvious way of incorporating observed locations as supervision. First, we can not set a user’s \( \theta_i \) as observed, because we observe only his home location instead of his location profile. Second, we can not set the location assignments for his re-
relationships as the observed location, as it does not allow the relationships to be generated based on other locations and fails to address the mixed-signal challenge. The existing modifications of LDA incorporate supervision in different settings. For example, the supervised LDA model [11] assumes a document has a label and each label corresponds to a mixture of topics. Our setting is different. First, we view each hidden dimension (a topic in LDA) as a semantic label (location). Second, a user has multiple labels, but only one label is observed.

We choose to use the home locations of labeled users as prior knowledge to generate their location profiles. As LDA, we assume that a user’s location profile $\theta_i$ is generated from a Dirichlet distribution $\text{DIR}(\vec{\gamma})$ with a hyper parameter $\vec{\gamma}$. In $\text{DIR}(\vec{\gamma})$, the larger $\vec{\gamma}$’s $l$th dimension $\gamma_l$ is, the more likely $\theta_i$ with a large probability in the $l$th dimension is to be generated. However, in LDA, $\vec{\gamma}$ is set uniformly, as it does not have any preference on any topic, while we can set them differently to encode our prior knowledge for labeled users, as we observe their home locations.

Technically, we introduce an “observation vector” and a “boosting matrix” to set the prior for each user. For a user $u_i$, an observation vector is an $L$-length binary vector, denoted as $\vec{\eta}_i$, and its $j$th dimension $\eta_{i,j}$ represents whether the $j$th location is observed. We assume $\eta_{i,j}$ is generated via a Bernoulli process with a parameter $b_o$, but is observed. A boosting matrix is an $L \times L$ matrix, denoted $\Lambda$, and a cell $\Lambda_{ij}$ represents how much the prior of the $j$th location should be boosted when the $i$th location is observed. In our implementation, we assume $\Lambda$ is a diagonal matrix for simplicity, which means observing the $i$th location only boots its prior. Thus, the hyper parameter $\vec{\gamma}_i$ for $u_i$ is set by $\vec{\gamma}_i = \vec{\eta}_i \times \Lambda \times \vec{\gamma} + \vec{\gamma}$, where the first term encodes how much we boost the prior for an observed location, and the second term encodes our priors for candidate locations. With $\vec{\gamma}_i$, we will have a high probability to obtain $\theta_i$ that has a high probability to generate the observed location. We will see this clearly in Sec. 5.4.4.

Then, we motivate the need for limiting the number of candidate locations in a user’s location profile. There are three reasons. First, it is useless to consider every location for
a user, as some are definitely not related to him. E.g., if a user only follows users in and tweets about California, any location from the east coast is not related to him. Second, a user usually has a small number of locations due to relocation costs. Third, it is inefficient to consider every location for every user. We will show this clearly in Sec. 5.4.5.

This is a unique challenge in our setting and has not been addressed by LDA, because in LDA the number of topics can be adjusted (usually from 20 to 200) during the estimation, while in MLoc, a set of candidate locations \( L \) is given, which could be a very large number (5000 in our experiment).

To solve the challenge, we introduce a “candidacy vector” to represent the candidacy of locations for a user \( u_i \). For \( u_i \), his candidacy vector is an \( L \)-length binary vector, denoted as \( \vec{\lambda}_{i,j} \). \( \lambda_{i,j} \) is 1 if and only if the \( j^{th} \) location is a candidate location for \( u_i \). We can assume \( \lambda_{i,j} \) is generated via a Bernoulli process with a parameter \( b_c \), but is observed.

We utilize location observed from a user’s neighbors to set his candidacy vector. Specifically, we assume that \( \vec{\lambda}_{i,j} \) is 1, if and only if the \( j^{th} \) candidate location is observed from \( u_i \)’s following and tweeting relationships. The statistics from our data generally validate this assumption. In our incomplete crawl of Twitter, there are about 92% users whose locations appear in their relationships. We use \( \tau \) to represent the prior value for each candidate location. \( \tau \) is set to a small number (0.1 in our experiments), as previous studies show [11] that the values of hyper parameter below 1 prefer sparse distributions. Thus, we can use \( \tau \cdot \vec{\lambda}_{i} \) to represent priors of candidates locations for \( u_i \).

Thus, the prior \( \gamma_i \) for a user \( u_i \) can be set as follows,

\[
\vec{\gamma}_{i} = \vec{\eta}_{i} \times \Lambda \times \vec{\gamma} + \tau \cdot \vec{\lambda}_{i}.
\] (5.3)

### 5.4.4 Generative Model

We now present MLoc completely. As a generative model, it can be explained by an imaginary process that describes how following and tweeting relationships are generated.
**Generative Process** First, for each user $u_i$, we generate his prior distribution parameter $\gamma_i$ and location profile $\theta_i$.

- Generate $u_i$’s observation vector $\vec{\eta}_i$ via a Bernoulli distribution with a parameter $b_o$.
- Generate $u_i$’s candidacy vector $\vec{\lambda}_i$ via a Bernoulli distribution with a parameter $b_c$.
- Calculate $\vec{\gamma}_i$ based on Eq. 5.3.
- Generate $\theta_i$ from a Dirichlet distribution with $\vec{\gamma}_i$.

We note that since $\vec{\eta}_i$ and $\vec{\lambda}_i$ are observed, they block the influence of $b_o$ and $b_c$. We can ignore $b_o$ and $b_c$ in the joint probability. As $\vec{\gamma}_i$ can be computed from $\vec{\eta}_i$ and $\vec{\lambda}_i$, we will use the computed $\gamma_i$ in the joint probability directly.

Second, for each location $l$, its tweeting model $\psi_l$ is generated from a Dirichlet distribution $\text{DIR}(\vec{\delta})$.

Third, for each pair of users $u_i$ and $u_j$, whether $u_i$ builds a following relationship $f\langle i, j \rangle$ to $u_j$ is determined as follows.

- Generate a model selector $\mu$ according to a Bernoulli distribution with a parameter $\rho_f$.
- If $\mu = 1$, we choose the random following model $F_R$ to decide whether there is $f\langle i, j \rangle$.
- If $\mu = 0$, we choose the location-based following process, which contains three steps.
  - Choose a location assignment $x_i$ from $\theta_i$.
  - Choose a location assignment $y_j$ from $\theta_j$.
  - Decide whether there is $f\langle i, j \rangle$ based on the location-based following model in Eq. 5.1.

We note that the above process models any pair of users including pairs with or without a following relationship. However, we choose to use only the pairs with following relationships as our observations because of two reasons. First, it is more faithful to the underlying semantics of the data in our setting, as the absence of a following relationship from $u_i$ to $u_j$ does not necessarily mean that $u_i$ will not follow $u_j$. E.g., they may be real friends who are unaware of each other’s existence in the network. Second, it significantly decreases the
computational cost of inference, as the complexity of computation scales with the number of observed relationships rather than the number of user pairs.

Fourth, for each tweeting relationship $t_k\langle i, j \rangle$ from a user $u_i$ to a venue $v_j$, it is generated by the following steps.

- Generate a model selector $\nu_k$ according to a Bernoulli distribution with a parameter $\rho_t$.
- If $\nu_k = 1$, we choose the random tweeting model $T_R$ to generate $t_k\langle i, j \rangle$.
- If $\nu_k = 0$, we choose the location-based generation process, which contains two steps.
  - Choose a location assignment $z_{k,i}$ from $\theta_i$.
  - Generate $t_k\langle i, j \rangle$ based on the location-based tweeting model as shown in Eq. 5.2.

**Joint Probability** Based on the generative process, MLoc defines the joint probability of generating both the observed and hidden random variables given model parameters. Specifically, we assume the parameters, $\rho_f$, $\rho_t$, $\alpha$, $\beta$, $F_R$, $T_R$, $\bar{\gamma}_i$, and $\bar{\delta}$ are given. To simplify our notations, we use $\Omega$ to represent them. The joint distribution can be represented as follows.

$$
P(\theta_{1:N}, \psi_{1:L}, \mu_{1:S}, x_{1:S}, y_{1:S}, f_{1:S}, \nu_{1:K}, z_{1:K}, t_{1:K} | \Omega) = \prod_{i=1}^{N} P(\theta_i | \bar{\gamma}) \prod_{l=1}^{L} P(\psi_l | \bar{\delta}) \prod_{k=1}^{K} P(\nu_k | \rho_t) \prod_{s=1}^{S} P(\mu_s | \rho_s)$$

$$\prod_{s=1}^{S} (P(x_{s,i} | \theta_i)P(y_{s,j} | \psi_j)P(f_{s}\langle i, j \rangle | \alpha, \beta, x_{s,i}, y_{s,j}))^{1-\mu_s}$$

$$\prod_{k=1}^{K} (P(z_{k,i} | \theta_i)P(t_k\langle i, j \rangle | z_{k,i} = l, \psi_l))^{1-\nu_k} \prod_{s=1}^{S} P(f_{s}\langle i, j \rangle | F_R)^{\mu_s} \prod_{k=1}^{K} P(t_k\langle i, j \rangle | T_R)^{\nu_k}(5.4)$$

In the above equation, the following and tweeting relationships, i.e., $f_{1:S}$ and $t_{1:K}$, are observed, while users’ location profiles $\theta_{1:N}$, the locations’ tweeting models $\psi_{1:L}$, the model selectors (e.g., $\mu_s$, $\nu_k$) and the location assignments (e.g., $x_{s,i}$, $y_{s,j}$, and $z_{k,i}$) are hidden. The central computational problem for MLoc is to use the observed relationships and the given parameters to infer the hidden unknown variables.
Discussions Based on Fig. 5.2, we can show the difference between MLoc and MMSB mentioned in Sec. 5.2 in terms of generating “relationships” between nodes based on pairwise variable interactions. MMSB associates every pair of communities with an interaction parameter and uses $K^2$ parameters for $K$ communities, while MLoc uses a power law distribution with $\alpha$ and $\beta$ to parameterize pairwise location interactions based on the real-world observations in Sec. 5.4.1. MLoc has two advantages. First, it greatly reduces the number of parameters from $K^2$ to 2, and thus parameters can be estimated accurately with limited observations. Second, it explicitly constrains “interaction probabilities” and makes location profiling accurate. The interaction probabilities in MMSB could be any distribution, while the power law distribution explicitly constraints that the two users in a relationship are likely to be close.

5.4.5 Inference with Gibbs Sampling

MLoc models various aspects that haven’t been addressed by existing generative models, and combines discrete and continuous distributions in a non-trivial manner. It is complex and does not allow for exact inference. We derive our own approximate inference algorithm. We derive our inference algorithm via the following steps: 1) we integrate $\theta_1:N$ and $\psi_1:L$ in the joint probability, so we do not need to estimate $\theta_1:N$ and $\psi_1:L$ at the beginning, 2) we use the Gibbs sampling method, which is one of classical methods, to sample from the posterior distribution of the model selectors and the location assignments given the relationships and the model parameters, $P(\mu_1:S, x_1:S, y_1:S, \nu_1:K, z_1:S | f_1:S, t_1:K, \Omega)$, and 3) we estimate the location profile $\theta_i$ for user $u_i$ based on sampled $\mu_1:S, \nu_1:K, x_1:S, y_1:S$ and $z_1:K$.

To sample from $P(\mu_1:S, x_1:S, y_1:S, \nu_1:K, z_1:S | f_1:S, t_1:K, \Omega)$, a standard Gibbs sampling procedure requires to compute the following conditional posterior distributions.

- $P(\mu_s | \mu_{-s}, \nu_1:S, x_1:S, y_1:S, f_1:S, z_1:K, t_1:K, \Omega)$,
- $P(\nu_k | \nu_{-k}, \mu_1:S, x_1:S, y_1:S, f_1:S, z_1:K, t_1:K, \Omega)$,
\[
\begin{align*}
&\cdot P(x_{s,i}|\mu_{1:S}, \nu_{1:S}, x_{-s,i}, y_{1:S}, f_{1:S}, z_{1:K}, t_{1:K}, \Omega), \\
&\cdot P(y_{s,j}|\mu_{1:S}, \nu_{1:S}, x_{1:S}, y_{-s,j}, f_{1:S}, z_{1:K}, t_{1:K}, \Omega), \\
&\cdot P(z_{k,i}|\mu_{1:S}, \nu_{1:S}, x_{1:S}, y_{1:S}, f_{1:S}, z_{-k:i}, t_{1:K}, \Omega), \\
\end{align*}
\]

In the above probabilities, \(\mu_{-s}, \nu_{-k}, x_{-s,i}, y_{-s,j}\), or \(z_{-k,i}\) denote all the assignments except the \(s^{th}\) or \(k^{th}\) assignment. We derive those equations as below. The detailed derivation is omitted due to the space limitation.

\[
P(\mu_{s}|\mu_{-s}, \nu_{1:S}, x_{1:S}, y_{1:S}, f_{1:S}, z_{1:K}, t_{1:K}, \Omega) \\
\sim P(\mu_{s}|\rho_f)(P(f_s(i,j)|F_R))^{\mu_{s}} \times \left( \frac{\varphi_{i,l} + \gamma_{i,l} - 1}{\varphi_{i} + \sum_{l=1}^{L} \gamma_{i,l} - 1} \beta \times d(x_{s,i}, y_{s,j})^{\alpha} \right)^{1-\mu_{s}} (5.5)
\]

\(\varphi_{i,l}\) denotes the frequency that the \(l^{th}\) location is observed from \(u_i^{'}s\) locations. \(\varphi_{i}\) denotes the total number of \(u_i^{'}s\) location assignments. \(\gamma_{i,l}\) is the \(l^{th}\) dimension of the prior \(\vec{\gamma}_{i}\).

\[
P(\nu_{k}|\nu_{-k}, \mu_{1:S}, x_{1:S}, y_{1:S}, f_{1:S}, z_{1:K}, t_{1:K}, \Omega) \\
\sim P(\nu_{k}|\rho_f)(P(t_s(i,j)|T_R))^{\nu_{k}} \times \left( \frac{\varphi_{i,l} + \gamma_{i,l} - 1}{\varphi_{i} + \sum_{l=1}^{L} \gamma_{i,l} - 1} \sum_{v=1}^{V} (\phi_{l,v} + \delta_{v}) - 1 \right)^{1-\nu_{k}} (5.6)
\]

\(\phi_{l,v}\) is the frequency that \(v\) is tweeted by users at \(l\). \(\delta_{v}\) is the \(v^{th}\) dimension of the prior \(\vec{\delta}\).

The above two equations sample model selectors of relationships, which help us to identify noisy relationships. They can be interpreted intuitively. For example, in Eq. 5.5, the probability of \(\mu_{s} = 1\) is proportional to two factors: 1) the probability of \(\mu_{s} = 1\) encoded in \(\rho_f\), and 2) the probability of observing \(t_s(i,j)\) in the random model \(F_R\).
\begin{align*}
P(\mathbf{x}_{s,i} | \mathbf{\mu}_1, \mathbf{\nu}_1, \mathbf{x}_{-s,i}, \mathbf{f}_1, \mathbf{z}_1, \mathbf{t}_1, \Omega) & \sim \frac{\phi_{i,l} + \gamma_{i,l} - 1}{\phi_i + \sum_{l=1}^{L} \gamma_{i,l} - 1} (d(\mathbf{x}_{s,i}, \mathbf{y}_{s,j})^\alpha)^{1-\mu_s} \\
P(\mathbf{y}_{s,j} | \mathbf{\mu}_1, \mathbf{\nu}_1, \mathbf{x}_{s,i}, \mathbf{y}_{-s,j}, \mathbf{f}_1, \mathbf{z}_1, \mathbf{t}_1, \Omega) & \sim \frac{\phi_{j,l} + \gamma_{j,l} - 1}{\phi_j + \sum_{l=1}^{L} \gamma_{j,l} - 1} (d(\mathbf{x}_{s,i}, \mathbf{y}_{s,j})^\alpha)^{1-\mu_s} \\
P(\mathbf{z}_{k,i} | \mathbf{\mu}_1, \mathbf{\nu}_1, \mathbf{x}_{1}, \mathbf{y}_{1}, \mathbf{f}_1, \mathbf{z}_{-k,i}, \mathbf{t}_1, \Omega) & \sim \frac{\phi_{i,l} + \gamma_{i,l} - 1}{\phi_i + \sum_{l=1}^{L} \gamma_{i,l} - 1} \left(\phi_{l,v} + \delta_v - 1 \sum_{v=1}^V \left(\phi_{l,v} + \delta_v - 1\right)^{1-\nu_k}\right)^{1-\nu_k}
\end{align*}

The above three equations sample location assignments for relationships, which can be viewed the estimated location assignments that explain the true geo connections in the relationships. They can be interpreted intuitively. For example, Eq. 5.7 contains two parts. The first one suggests that the probability of \( \mathbf{x}_{s,i} = l \) should be proportional to the frequency of the \( l \)th location in the existing samples of \( u_i \) plus our prior belief \( \gamma_l \). The second one suggests that the probability should be negatively related to the distance from \( \mathbf{x}_{s,i} \) to \( \mathbf{y}_{s,j} \) (remind that \( \alpha \) is learned as \(-0.55 \) initially), but this part is active when the location-based model is used (\( \mu_s = 0 \)). When the random model is used (\( \mu_s = 1 \)), the probability is only proportional to the first part.

Our algorithm performs the above update equations for every following and tweeting relationship in one iteration. The algorithm runs a number of iterations until convergence.

From the above equations, we can clearly see that the supervision is encoded in our model. \( \gamma_{i,l} \) can be interpreted as pseudo counts for the \( l \)th location in \( \theta_i \). Remind that we set \( \gamma_{i,l} \) high when the \( l \)th location is observed from the \( i \)th user. Thus, we will have a high probability to generate the observed location for a labeled user.

From the above equations, we can also see that users’ candidacy vectors greatly improve the efficiency our algorithm. As Eq. 5.7, 5.8 and 5.9 estimate a probability for each candidate location for each assignment, the candidacy vector helps us to prune a large set of unrelated
locations, and we do not need to estimate their probabilities.

After obtaining the location assignments for relationships, we estimate the location distribution \( \theta_i \) for user \( u_i \) with the maximal likelihood estimation principle.

\[
p(l|\theta_i) = \frac{\varphi_{i,l} + \gamma_{i,l}}{\varphi_i + \sum_{l=1}^{L} \gamma_{i,l}}
\]  

(5.10)

Given the estimated \( \theta_i \), we can predict \( u_i \)'s the home location as the one with the largest probability in \( \theta_i \), and \( u_i \)'s location profile as the top K locations in \( \theta_i \) or the locations whose probabilities are larger than a threshold.

Furthermore, we can apply the Gibbs-EM principle \([1]\) to refine \( \alpha \) and \( \beta \) in our model. Specifically, at the E-step, we use the same Gibbs sampling algorithm to estimate \( x_{s,i} \) and \( y_{s,i} \)'s distribution and calculate the expected distance of each following relationship. At the M-step, we estimate \( \alpha \) and \( \beta \) based on the expected distance for each following relationship. Therefore, the new algorithm contains two iterations. In the inner iteration, it uses Eq. 5.7, 5.8 and 5.9 to estimate the location assignments iteratively. The outer iteration computes \( \alpha \) and \( \beta \) iteratively according to the results from the inner iteration.

5.5 Experiments

In this section, we conduct extensive experiments to demonstrate the effectiveness of our model. Specifically, we first evaluate our model on the home location prediction task, and demonstrate that our model predicts users’ home locations accurately and improves two state-of-the-art methods significantly. We further evaluate our model on discovering users’ multiple locations and explaining following relationships, and show our model discover users’ multiple locations completely and makes an accurate explanation for each relationship.

Data Collection We used the same data set used in the previous chapter. Specifically, our dataset contains 139,180 users with their following relationships and tweets. There are 14.8
friends, 14.9 followers, and 29.0 venues per user.

**Tasks** We evaluate our model’s performance on three tasks. Specifically, we apply our model to profile users’ 1) *home location prediction* and 2) *multiple locations discovery*. Then, we evaluate our model for *explaining following relationships*.

**Methods** To demonstrate the effectiveness of our model, we not only compare our model with two state-of-art methods in [7] and [14], but also evaluate our model with different types of resources. Specifically, we evaluate the following methods.

- **Base** \(_U\) is the location profiling method proposed in [7], which predicts a user’s location based on his social network using a distance-based probabilistic model. It is a state-of-the-art method for profiling user home location based on social network.
- **Base** \(_I\) is the collective classification method proposed in [31], which iteratively calculates the probability that a user associates with a location as the weighted average of the probabilities of its labeled friends (i.e., weighted majority voting). We note that this method can also be viewed as a popular graph-based semi-supervised learning (GSSL) method [61]. It is a widely applied method for network-based classification and performs surprisingly well in many settings.
- **Base** \(_C\) is the location profiling method proposed in [14], which classifies a user into locations based on local words identified from tweets using a standard classification method. It is a state-of-the-art method for content-based location profiling.
- **MLoc** \(_U\) is our location profiling method, but only uses users’ following relationships.
- **MLoc** \(_C\) is our location method, but only uses users’ tweeting relationships.
- **MLoc** is our complete location profiling method, which uses both following and tweeting relationships.

### 5.5.1 Experiments for Home Location Prediction

We first present our experiment results for predicting users’ home locations.
Ground Truth To get users’ home locations, we took their registered locations as their home locations, and applied five fold validation, which means that we used 80% of users as labeled users and 20% of users as unlabeled users. We note that we directly took users’ registered locations as their home locations, because we wanted to set up our experiments in the same way as the existing methods [14, 7]. We are aware that some registered locations are incorrect, but we believe they are rare, as leaving profiles empty is always an easy option. Therefore, our results are reliable overall.

Measures To evaluate performance, we applied Accuracy within m miles (ACC@m) used in [14] and [7] as our measure. Particularly, for a user \( u \), let \( l_u \) be \( u \)'s home location, \( \hat{l}_u \) be the predicted one, and \( d(l_u, \hat{l}_u) \) be their distance. For a set of test users \( U \), \( ACC@m = \frac{|\{u_i \mid u_i \in U \land d(l_i, \hat{l}_i) \leq m\}|}{|U|} \). By default, we set \( m \) to 100.

<table>
<thead>
<tr>
<th>Method</th>
<th>Base_U</th>
<th>Base_I</th>
<th>Base_C</th>
<th>MLoc_U</th>
<th>MLoc_C</th>
<th>MLoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC@100</td>
<td>52.44%</td>
<td>48%</td>
<td>49.67%</td>
<td>58.8%</td>
<td>55.3%</td>
<td>62.3%</td>
</tr>
</tbody>
</table>

User-based Performance First, we compare MLoc_U with Base_U and Base_I. All of them profile a user’s location based on his social network. Tab. 5.2 shows the results. Base_U performs the better than Base_I, because Base_U uses a probabilistic model to model the distances between locations while Base_I views the each location as independent categorical labels. The result validates that Base_U is a state-of-the-art method for profiling user home location and performs better than standard collective classification methods. MLoc_U further improves Base_U by 6%. We believe the improvement results from explicitly dealing with the noisy-signal and mixed-signal challenges. Although we predict the home location of a user, modeling multiple locations helps us to rule out “noisy” following relationships and make accurate predictions. MLoc_U improves Base_I by 10%, because it also explores the distances between locations.

Content-based Performance Next, we compare MLoc_C with Base_C. Both of them profile a user’s location with his tweets only. From the results in Tab. 5.2, we can clearly see that
ML$\text{Loc}_C$ significantly improves $\text{Base}_C$ by 5% in terms of $\text{ACC}@100$. Thus, we conclude that $\text{ML} \text{oc}_C$ is better than $\text{Base}_C$, and we believe the improvement is due to explicitly modeling users’ multiple locations and noisy venues.

We clarify that $\text{Base}_C$ requires human labeling to train a model to select local words, which are used as features for the classification model, and $\text{Base}_C$’s performance highly depends on the selected words. As the labeling is a subjective task, by no means could we get the same set of local words as in the original paper. We test performances of $\text{Base}_C$ with various local word sets, and we get $\text{ACC}@100$ ranging from 35.98% to 49.67%. We choose the highest one to report. Our method advances $\text{Base}_C$ in this aspect, as we do not require any labeling work, and only use venue names in an existing gazetteer.

**Overall** Then, we compare $\text{MLoc}$ with $\text{Base}_U$, $\text{Base}_I$, $\text{Base}_C$, $\text{MLoc}_U$, and $\text{MLoc}_C$. Tab. 5.2 shows that $\text{MLoc}$ improves the best baseline method $\text{Base}_U$ by 10%, and advances $\text{MLoc}_C$ and $\text{MLoc}_U$ by 7.0% and 3.6% respectively. We conclude that integrating different types of resources is useful, and our model can integrate them in a meaningful way. Meanwhile, we can say $\text{MLoc}$ is very accurate. It correctly places 62% users within 100 miles.

**Convergence** We evaluate the convergence of our model. Fig. 5.4 shows the convergence of $\text{MLoc}$. It converges quickly after about 14 rounds of iterations. We note that the number of iterations is much less than other cases where the Gibbs sampling algorithm is applied (e.g.,
hundreds iterations in LDA [11]). We believe that our model converges quickly because we initialize each user’s candidate locations based on our observations as discussed in Sec. 5.4.3.

5.5.2 Experiments for Multiple Location Discovery

We continue our evaluations to see whether our model can capture and discover users’ multiple locations.

**Ground Truth** To evaluate our model for discovering users’ multiple locations, we first got the ground truth. As a user’s profile does not contain multiple locations, we manually labeled locations for 1,000 users of the 139,180 users, and obtained 585 users, who clearly have multiple locations. We used those 585 users to evaluate our model and baseline methods. On average, a user has 2 locations.

To label users’ related locations, we explored different sources. The first one is user profiles. Some profiles explicitly state multiple locations (e.g., Augusta, GA/New London, CT), or contain external links (e.g., linkedin accounts), which provide detailed information. The second one is tweets. Some tweets clearly express the user’s related locations (e.g., “praying for my hometown. houston is wilding out.”), and some contain GPS tags. Our labeling requirements are very strict. We do not consider a location as a related location for a user, if it just appears several times in his tweets but does not indicate that the user lives or lived there (e.g., “watching houston game”).

**Measures** To evaluate the results, we introduce two new measures, distance-based precision (DP) and distance-based recall (DR). Specifically, we want to evaluate whether a set of discovered locations is close to a set of related locations of a user. In information retrieval, precision and recall evaluate whether retrieved results are relevant to a set of answers. However, they may underestimate performances in this task, because a predicted location (e.g., Santa Monica) may be different from but fairly close to a true location (e.g., Beverly Hills). Therefore, we propose DP and DR. Intuitively, DP is the fraction of predicted locations that
are close enough to true locations, while $DR$ is the fraction of true locations that are close enough to predicted ones. Formally, we define that a location $l$ is close enough to a set of locations $L$, denoted as $c(l, L) = true$, if and only if $\exists l' \in L$, s.t., $D(l, l') < m$, where $m$ is a threshold and is set to 100 miles. For a user $u$, let $L'(u)$ and $L(u)$ be predicted and true locations for $u$. $DP(u) = \frac{|\{l \in L'(u) \cap c(l, L(u))\}|}{|L'(u)|}$ and $DR(u) = \frac{|\{l \in L(u) \cap c(l, L'(u))\}|}{|L(u)|}$. To measure $DP$ and $DR$ for a set $U$ of users, we average $DP(u)$ and $DR(u)$ for $u \in U$. We use $DP@K$ or $DR@K$ to denote $DP$ or $DR$ of the top $K$ results. $K$ is set to 2 by default, as users have 2 locations on average. As $Base_U$ and $Base_C$ find only one location, we use their top $K$ predicted locations as the related locations.

<table>
<thead>
<tr>
<th>Method</th>
<th>$Base_U$</th>
<th>$Base_I$</th>
<th>$Base_C$</th>
<th>$MLoc_U$</th>
<th>$MLoc_C$</th>
<th>$MLoc$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP@2</td>
<td>33.8%</td>
<td>38.6%</td>
<td>39.3%</td>
<td>45.1%</td>
<td>48.3%</td>
<td>50.6%</td>
</tr>
<tr>
<td>DR@2</td>
<td>27.2%</td>
<td>32.6%</td>
<td>33.1%</td>
<td>42.3%</td>
<td>45.3%</td>
<td>47.0%</td>
</tr>
</tbody>
</table>

**Overall Performance** Tab. 5.3 shows the performance of each method. Generally, our methods, $MLoc_U$, $MLoc_C$ and $MLoc$, perform better than the baselines in both measures. In terms of $DP@2$, our methods predict more accurately than the baseline methods. In terms of $DR@2$, our methods discover users’ locations more completely than the baseline methods. We believe that such advantages are achieved because our model fundamentally captures that a user has multiple locations. For example, when a user has multiple locations from different areas, our methods discovers them completely, while the $Base_U$ and $Base_C$ retrieve only one location and its nearby cities, and $Base_I$ do not explore the distances between locations.

In addition, we plot $DP$ and $DR$ at different ranks in Fig. 5.5(a) and 5.5(b). From the figures, we obtain the following observations. First, our methods are better than the baseline methods at every $K$. It indicates that the baseline methods are not good at discovering multiple locations completely and accurately. Second, if we look at $DP@1$, baseline methods perform much worse than our methods. It is because when a user has multiple locations,
Figure 5.5: Precision and Recalls at Different Ranks

his relationships generated based on other locations are noisy information for the baseline methods. It again validates that a user’s multiple locations should be captured even for profiling his home location. We note that \textit{Base}_I performs better than \textit{Base}_U in this case, because \textit{Base}_U explicitly assumes that every user has only one location while \textit{Base}_I does not have such an explicit assumption. The results again validate that \textit{Base}_U cannot profile users’ multiple locations effectively.

\textbf{Case Studies} To illustrate the correctness of our model in discovering multiple locations, we give some examples in Tab. 5.4. It clearly shows that our model finds multiple locations completely and accurately, while the baseline methods find only one of true locations and its nearby locations. For instance, for user 13069282 in the 2nd row, who studied in Austin and works in Los Angeles, \textit{MLoc} discovers both locations, while the top 2 results returned by \textit{Base}_U are all around Los Angeles area.

Table 5.4: Case Study on Multiple Location Discovery

<table>
<thead>
<tr>
<th>UID</th>
<th>True Locations</th>
<th>\textit{MLoc}</th>
<th>\textit{Base}_U</th>
</tr>
</thead>
<tbody>
<tr>
<td>1178-4102</td>
<td>St. Louis, MO</td>
<td>St. Louis, MO</td>
<td>St. Louis, MO</td>
</tr>
<tr>
<td></td>
<td>Anaheim, CA</td>
<td>Los Angeles, CA</td>
<td>Chicago, IL</td>
</tr>
<tr>
<td>1306-9282</td>
<td>Los Angeles, CA</td>
<td>Los Angeles, CA</td>
<td>Los Angeles, CA</td>
</tr>
<tr>
<td></td>
<td>Austin, TX</td>
<td>Austin, TX</td>
<td>San Diego, CA</td>
</tr>
<tr>
<td>1501-3125</td>
<td>Nashville, TN</td>
<td>Murfreesboro, TN</td>
<td>New York, NY</td>
</tr>
<tr>
<td></td>
<td>Chicago, IL</td>
<td>Chicago, IL</td>
<td>Franklin, TN</td>
</tr>
</tbody>
</table>
5.5.3 Experiments for Relationship Explanation

We further evaluate our model to see whether relationships are correctly profiled.

**Ground Truth** To get the location assignments in following relationships, we manually labeled following relationships of the 585 users, whose multiple locations are known to us. In the labeling process, we only kept the following relationships in which users’ location assignments could be clearly identified by their shared “regions” (e.g., a user at Hollywood follows Los Angeles Weather Channel), and we obtained 4,426 relationships and the location assignments of them.

**Measure** We use Accuracy within $m$ miles ($ACC@m$) as our measure. We define that a relationship is accurately explained if and only if both users’ locations in the relationship are accurately assigned within $m$ miles.

As no previous work assigns locations for a relationship, we design a home location based explanation method to compare, denoted as *Base*. Specifically, for a following relationship, it directly assigns users’ home locations as their location assignments in the relationship. It is a strong baseline, as users are likely to follow others based on their home locations, and in most cases we do not know users’ home locations. However, this method will not work for the cases where users follow others based on their other locations.

**Overall Performance** Fig. 5.6 shows the $ACC@m$ of each method with different $m$. Generally, we see *MLoc* is significantly better than *Base*. Specifically, *Base* profiles only 40% relationships correctly. It again validates our assumption that a user’s following relationships are not necessarily generated based on his home location. *MLoc* significantly improves *Base* by 15%, which suggests that *MLoc* correctly profiles each relationship and so as to profile users’ locations accurately. The advantages are consistent with different distances. However, $ACC@50$ of *MLoc* is almost the same as $ACC@100$, which means most of the correctly profiled relationships are profiled within 50 miles.
Table 5.5: Case Studies on Relationship Explanation

<table>
<thead>
<tr>
<th>Follower’s ID and Follower’s Location</th>
<th>Location Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Location</td>
</tr>
<tr>
<td>101566144</td>
<td>Austin</td>
</tr>
<tr>
<td>14119630</td>
<td>Portland</td>
</tr>
<tr>
<td>15669188</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>53154473</td>
<td>Long Beach</td>
</tr>
</tbody>
</table>

**Case Study** To illustrate the correctness of our model, we continue our examples. Specifically, we show the location assignments for some following relationships of user 13069282 in Tab. 5.5. Due to the space limitation, we remove the state information for each city in the table. Our method correctly assigns different locations (e.g., Austin or Los Angeles) to her following relationships. Based on these assignments, MLoc can estimate the user’s multiple locations, i.e., Los Angeles and Austin, correctly. In addition, it allows us to group a user’s followers into different geo groups (e.g., Los Angeles and Austin). Geo groups can be further used to group followers into more meaningful groups (e.g., classmates in Austin).
Chapter 6
Conclusion and Future Agenda

In this chapter, I first summarize the contributions of this thesis and then discuss some future research directions.

6.1 Conclusion

Social media bring tremendous opportunities for building novel analytic applications in various domains. In this thesis, I discuss initial research efforts for realizing a general platform to enable those opportunities. Particularly, this thesis makes the following important steps towards a general platform that can support various social media analytic applications.

System At the beginning of this thesis, I propose the design of our BigSocial platform, which abstracts the essential functionalities required by different analytic applications. A prototype system has been built based on the proposed platform, which shows the great promise of social media based analysis and the feasibility of our proposed platform.

Techniques Further, I identify and solve some challenging research problems in realizing BigSocial. I make some important contributions to each problem specifically.

- First, I study how to continuously monitor/collect relevant tweets for any given analytic application with Twitter APIs. I design the first framework (i.e., ATM) that can monitor most relevant tweets for any given application in an continue and optimal way, and develop three important algorithms to enable the framework, including 1) a novel tweet sampling algorithm, which can sample sufficient of random tweets with the limited and biased
Twitter APIs, 2) a machine learning based prediction algorithm, which can accurately predict usefulness of keywords in future based on collected samples, and 3) an efficient keyword selection algorithm, which can select the optimal keywords under two types of constraints with guaranteed approximation rate in a linear time. Extensive experiments show that ATM is very effective in practice. E.g., it collects most of relevant tweets (90%) for a given application.

- Second, I study how to profile Twitter users’ home locations accurately in social media. In this study, I propose a unified discriminative influence model UDI based approach. To the best of our knowledge, our approach is the first approach that profiles users’ home location with both social network and user-centric data. Further, our approach novelly captures the influence scopes of location signals in social media, and thus is robust to noisy signals in social media. Extensive experiments on a large scale data set demonstrate our algorithms are accurate and improve the state of the art methods by 13%.

- Third, I study how to profile users’ multiple locations completely in social media. In this study, we propose a multiple location profiling model MLoc. To the best of our knowledge, MLoc is the the first model that 1) discovers users’ multiple locations and 2) profiles relationships. Specifically, for profiling users’ locations, MLoc advances the existing methods from the following aspects: 1) it profiles a user’s home location more accurately, as it fundamentally models that following relationships and tweeted venues are generated by users’ multiple locations, and 2) it profiles a user’s locations more completely, as it explicitly models that a user has multiple locations. In addition, MLoc is able to profile each following relationship in terms of users’ hidden locations. Extensive experiments on three different tasks demonstrate those advantages empirically.

**Evaluation** Finally, I want to emphasize that I collect a large scale real-world social media data during my research study, and I share those datasets online. I hope these datasets will be useful for future research in this direction.
In summary, this thesis presents my initial efforts of realizing a general platform to support different analytic applications. Based on the research in this thesis, many useful applications, such as finding popular tablets liked by female users in California, or surveying political issues concerned by the voters in Illinois, could be enabled in an effective way. Specifically, our social media monitor enables collecting relevant data for an analytic task efficiently, and our user profiler allows us to conduct complex analysis with users’ hidden dimensions (missing attributes) like locations and occupations. With these applications, we can turn the “big” but “noisy” social media to useful insights in various domains.

6.2 Future Agenda

While this thesis paves the way for enabling a general platform to support various social media applications effectively, there are many directions to further enhance and extend the platform. They could be explored as future research.

Social Media Monitoring For the social media monitoring problem, the automatic topic-focus monitor could be enhanced from two aspects.

First, we can study how to update classifiers used in the monitor dynamically. Currently, we assume that our monitor directly takes a classifier, which can automatically determine whether tweets are relevant or not, since our major focus is to design a general platform to collect relevant tweets for various applications, which have already applied classifiers to automatically determine whether tweets are relevant to their applications. However, in practice, as new tweets are being generated all the time, classifiers need to be updated as well. Thus, we could enhance the monitor via updating them. One possible way is that we could explore collected tweets to monitor the classifiers’ performances and update classifiers if there are dramatic changes of their performances. How to update those classifiers in a principled way is an undressed and interesting problem for future research.

Second, we can study how to collect relevant tweets if advanced APIs are provided for
monitoring tweets (e.g., searching relevant with keywords). Currently, the social media APIs can only filter tweets for given keywords, and the monitor collects target tweets based on those APIs. As searching relevant tweets with keywords is a popular method for retrieving relevant tweets, it is interesting to explore how to select multiple keyword queries, if the search function is provided for monitoring targeted tweets. Intuitively, as the keyword-document relationships change from binaries (i.e., whether a keyword matches documents or not) to real numbers (i.e., their relevance score) and we may be only interested in some documents that pass certain relevance scores, we need to redefine the usefulness and costs of keywords according to this new setting and extend our keyword selection algorithm accordingly.

**User Profiling** For the user profiling problem, we could further extend our current methods from the following directions.

The first direction is to add the temporal dimension into location profiling. The temporal information, such as when users connect to their friends and when users tweet venues, is useful for location profiling. For instance, utilizing temporal information can help us to develop a robust model for location profiling and to understand users’ movement from a place to another place over time. To enable such opportunities, we need to build a new model to fundamentally model how users tweet different locations and connect to different friends over time. For example, we can assume that, 1) at a time point, a user is associated with one location, where the user has possibilities to follow friends from this location and previous locations, and 2) the user has some probabilities to change his/her locations over time.

The second direction is to utilize advanced text processing techniques (e.g., recognizing entities from tweets, understanding semantics of tweets) to improve location profiling. Since those advanced techniques may have their own difficulties in processing short tweets, our current model only uses location terms extracted from tweets to avoid depending on those complex techniques. However, those techniques are definitely useful for locations profiling.
For example, if we can extract fine-grained entities/activities (e.g., restaurants, universities), we would be able to profile users’ locations more accurately. Further, if we can understand the semantics of the tweets (e.g., “terrible traffic at I74”, and “travel to Honolulu for vacation”), we would be able to distinguish true locations from noisy locations easily, and profile users’ locations accurately.

The third direction is to extend our profiling models for general user attributes. Besides locations, other attributes, such as gender and occupation, are important to applications as well. It is necessary and useful to extend our profiling models to profile those general attributes. We need to explore the specific models for those attributes.

**Analytic Tasks** Finally, some research problems, which are related to specific analytic tasks, are worth to explore as well. To provide complex analysis for a specific task, we need to develop a complex analysis model or utilize domain-specific features. For example, we need to develop a trend analysis model to find trends about an entity in social media or an abnormality analysis model to detect unexpected aspects about an entity in social media, and we may utilize the followers of their actors for predicting popularity of movies.
References


[59] Zhijun Yin, Liangliang Cao, Jiawei Han, Chengxiang Zhai, and Thomas Huang. Geographical topic discovery and comparison. In Proceedings of the 20th International Conference on World Wide Web, pages 247–256, 2011.

[60] Zhijun Yin, Rui Li, Qiaozhu Mei, and Jiawei Han. Exploring social tagging graph for web object classification. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 957–966, 2009.