Location Inference for Non-geotagged Tweets in User Timelines

Li, Pengfei; Lu, Hua; Kanhabua, Nattiya; Zhao, Sha; Pan, Gang

Published in:
I E E E Transactions on Knowledge & Data Engineering

DOI (link to publication from Publisher):
10.1109/TKDE.2018.2852764

Publication date:
2019

Document Version
Accepted author manuscript, peer reviewed version

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2852764, IEEE Transactions on Knowledge and Data Engineering

Location Inference for Non-geotagged Tweets in User Timelines

Pengfei Li, Hua Lu, *Senior Member, IEEE,* Nattiya Kanhabua, Sha Zhao, and Gang Pan

Abstract—Social media like Twitter have become globally popular in the past decade. Thanks to the high penetration of smartphones, social media users are increasingly going mobile. This trend has contributed to foster various location based services deployed on social media, the success of which heavily depends on the availability and accuracy of users' location information. However, only a very small fraction of tweets in Twitter are geo-tagged. Therefore, it is necessary to infer locations for tweets in order to attain the purpose of those location based services. In this paper, we tackle this problem by scrutinizing Twitter user timelines in a novel fashion. First of all, we split each user's tweet timeline temporally into a number of clusters, each tending to imply a distinct location. Subsequently, we adapt two machine learning models to our setting and design classifiers that classify each tweet cluster into one of the pre-defined location classes at the city level. The Bayes based model focuses on the information gain of words with location implications in the user-generated contents. The convolutional LSTM model treats user-generated contents and their associated locations as sequences and employs bidirectional LSTM and convolution operation to make location inferences. The two models are evaluated on a large set of real Twitter data. The experimental results suggest that our models are effective at inferring locations for non-geotagged tweets and the models outperform the state-of-the-art and alternative approaches significantly in terms of inference accuracy.

Index Terms—Twitter, Location Inference, Bayes, LSTM

1 Introduction

THANKS to the high penetration of smartphones, social media users are increasingly going mobile. For example, 56.5% Facebook users login only from mobile devices by the end of July, 2016[1], and Twitter has approximately 257 million mobile active users monthly as per the first quarter in 2016[2]. This big trend has fostered various location based services deployed on social media. The success of such location based services heavily depends on the availability and accuracy of users' location information that a social media platform can get access to. Knowing the locations of individual tweets of users enables a wide variety of applications, e.g., location-based summarization [1], location-aware recommender system [2], [3], friends notification [4], influential users recommendation [5], [6], [7] place advertisements [8] and business information spreading [9], city-scale collective attention analytics [10] and even disaster detection [11].

However, only a very small fraction of tweets are geo-tagged, i.e., being sent with GPS coordinates or place names. Cheng et al. [12] found that only 26% of Twitter users in a random sample of over 1 million users reported their location in their profiles and only 0.42% of the tweets in the sample were geo-tagged. Therefore, it is necessary to infer locations for Twitter users in order to attain the purpose of and to improve the quality of the location based services offered to the users. Most related works so far have focused on inferring only the home location for a Twitter user's timeline. This is apparently insufficient for location based services in general.

Location inference for tweets are challenged by two major issues. First, Twitter limits the length of each tweet content to 140 characters, and thus a tweet only contains a small number of words and conveys limited information. Second, Twitter users often use non-standard and shorthand terms, and tweets are often unclear and noisy. Consequently, finding location clues from short, noisy tweets is indubitably difficult.

In this study, we investigate how to infer the locations of non-geotagged tweets at the city level by scrutinizing Twitter users' timelines using a novel approach. Our approach combines analysis on the contents of tweet short texts and that on the user timelines with temporal information. Along the temporal dimension, each user timeline is split into a number of tweet clusters; each cluster implies a distinct user location. This process is called temporal clustering of tweets.

Subsequently, two machine learning models are carefully adapted to our problem setting and classifiers are designed to classify each tweet cluster from a user's timeline into one of the pre-defined location classes at the city level. The Bayes based model focuses on the information gain of words with location implications in the user-generated contents, whereas the LSTM based model treats user-generated contents and their associated locations as sequences and employs a bidirectional LSTM [13] and convolution operation to make location inferences. Our models are trained using offline data, but they can be used to infer locations for historical tweets and online (incoming) tweets.

The two models are experimentally evaluated on a large real dataset, in comparison with alternative approaches. The experimental results suggest that the proposed models are effective at inferring locations for tweets and they outperform alternatives significantly in terms of inference accuracy.

- *P. Li, S. Zhzo, and G. Pan are with Department of Computer Science, Zhejiang University, China.*
  E-mail: {pfl, szhao, gpan}@zju.edu.cn
- *H. Lu is with the Department of Computer Science, Aalborg University, Denmark.*
  E-mail: luhua@cs.aau.dk
- *N. Kanhabua is a senior data scientist at NTENT, Barcelona.*
  E-mail: nattiya@gmail.com

1. http://expandedramblings.com/index.php/facebook-mobile-app-statistics/
2. http://expandedramblings.com/index.php/twitter-mobile-statistics/

Our contributions in this study are summarized as follows.

- We design temporal clustering methods that split a user's tweet timeline into a set of clusters each of which contains tweets that are likely sent from the same city.
- We design a Bayes' theorem based model for location inference for tweet clusters. The model measures words' geographical scopes by computing words' information gains across all locations of interest.
- We build a novel neural network that combines convolution operation and long short-term memory unit when extracting features from the contents of tweet clusters. It is able to exploit spatially-local correlation [14], [15], [16] when inferring locations for tweet clusters.
- We evaluate the performance of our proposed approach and models using real-world Twitter data. The results show that our approach with the models outperforms state-of-the-art alternatives.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 formulates the research problem and introduces our solution framework. Section 4 describes the temporal clustering methods for training and testing data. Sections 5 and 6 detail our two models for tweet location inference, respectively. Section 7 reports on the experimental results. Finally, Section 8 concludes the paper and points to future work directions.

2 Related Work

Existing techniques for inferring locations in social networks fall in two categories: those inferring a Twitter user's location and those inferring locations for tweets. Most of them infer locations at the city level.

2.1 Location Inference for Social Network Users

Several existing techniques make use of social networks and friends' locations to infer locations for Twitter users. Davis Jr. et al. [17] and Jurgens [18] consider the locations of a user's friends and take the friend's location with the majority votes as the user's location. Rout et al. [19] use an SVM classifier and features extracted from Twitter user networks to predict home locations for Twitter users. Backstrom et al. [20] study the relationship between friendship and spatial distance for Facebook users, and exploit the relationship to compute a Facebook user's home location. McGee et al. [21] enhance that method with social tie strength for Twitter users. Rodrigues et al. [22] combine user-posted texts and user friendship network to infer the locations of Twitter users. Li et al. [23] propose a unified discriminative influence model that utilizes both user contents and social network to infer user locations. Jurgens et al. [24] report on a comparative study of user location inference approaches based on social networks.

As social networks are dynamic and their topology changes, predicting users' locations based on social networks may face difficulties. Therefore, more approaches make the location inferences only using user-generated contents.

Most content-based approaches aim to build a probabilistic system and determine user location by maximum likelihood. Wing et al. [25] use language models and information retrieval techniques for location prediction. Dividing the space into grids, the approach compares the distribution of words in a given user's tweets to those in each grid cell using Kullback-Leibler (KL) divergence to identify the user's most likely location. The approach has a serious data skewness problem—grid cells in rural areas tend to contain very few tweets while those in urban contain too many. To alleviate the problem, Roller et al. [26] use a k-d-tree-based adaptive grid in which cells contain approximately the same amount of data.

Topic models are also often used to infer locations based on contents. Eisenstein et al. [27] build geographic topic models to infer the home locations of Twitter users in terms of regions and states. The follow-up works [28], [29] use sparse additive models to combine region-specific, user-specific and non-informative topics. Chen et al. [30] build a topic model to mine user interest from short texts and then establish a mapping between locations and user interests.

Some probabilistic approaches use the idea of naive Bayes to estimate the probabilities that users are located in given locations. Cheng et al. [12] propose a probabilistic framework that applies a variant of naive Bayes to estimate users' city-level locations purely based on tweet contents. Ryoo and Moon [31] apply a similar idea to Korean Twitter users. Hecht et al. [32] analyze the user location field in user profiles and use multinomial naive Bayes to estimate user locations at country and state levels.

Recently, Qian et al. [33] propose a probabilistic model based on factor graphs for Twitter user's location inference at the country level. Chang et al. [34] use Gaussian Mixture Model (GMM) and an unsupervised approach to infer city locations of users. Liu et al. [35] propose a Hidden Markov Model framework that integrates tweet content and user movement to infer home locations at the city level.

There are also non-probabilistic approaches focusing on other aspects, e.g., feature selection. Han et al. [36], [37] emphasize the role of feature selection to identify location indicative words for the task of location inference for users. Huang et al. [38] pay particular attention to features from users' profiles to infer locations for Twitter users. Mahmud et al. [39], [40] use an ensemble of statistical and heuristic classifiers to predict locations at the city level. Ikawa et al. [41] use a rule-based approach to predict a user's current location based on their previous tweets. Krishnamurthy et al. [42] propose a knowledge-based approach utilizing Wikipedia as a source of knowledge base to predict a Twitter user's location. Yamaguchi et al. [43] propose an online location inference method over social streams that exploits the spatiotemporal correlation to infer the locations of users.

2.2 Location Inference for Individual Tweets

Only a handful of existing approaches infer locations for individual tweets [44], [45], [46], [47], [48], [49], [50], [51]. Their location inferences are all based on tweet contents. Table 1 summarizes these studies.

TABLE 1: Studies on location inference for tweets

	Location Granularity	Key Idea
Ref. [44]	City/Zip Code	Language model
Ref. [45]	Neighborhood	Language model
Ref. [46]	City	Gaussian mixture model
Ref. [47]	Country	Features combination
Ref. [49]	Geo-coordinate	Spatiotemporal topic model
Ref. [50]	Country/City	Using time as a feature
Ref. [48] Ref. [51]	City/Neighborhood	Tweet similarity comparison

Kinsella et al. [44] create language models of locations using geo-tagged tweets. They measure the difference between a tweet

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2852764, IEEE Transactions on Knowledge and Data Engineering

and the language models using Kullback-Leibler (KL) divergence and infer tweet locations at the city and zip code levels. Doran et al. [45] build smoothed language models to estimate tweet locations at the neighborhood level. Priedhorsky et al. [46] propose a two-dimensional Gaussian Mixture Model (GMM) to infer the city location of a tweet. Zubiaga et al. [47] extract features from user profiles and tweet contents, and then use a weighted maximum entropy classifier to determine the country in which a tweet was posted in a real-time scenario.

Some studies consider not only content and metadata in user profiles but also temporal information. Yuan et al. [49] propose a probabilistic topic model to exploit micro-blogging data to detect spatio-temporal topics, and then they use the discovered topics to model user mobility behaviors and infer tweet locations in the geo-coordinate level. Dredze et al. [50] also consider the impact of time on tweet locations. The authors take time as a feature and use a linear classifier trained on geo-tagged tweets to infer tweet locations at the country and city levels. Aiming at locations at city and neighborhood levels, studies [48], [51] exploit the similarities in the contents between a tweet and a set of geo-tagged tweets posted at the same time. However, a single tweet may be too short to contain sufficient features similar to other tweets that are even posted in the same location. In addition, people often post similar tweets at different times and locations.

Compared with existing approaches, our approach exploits the temporal information differently. A temporal clustering technique is used to split each Twitter user's timelines into clusters each of which is expected to contain tweets posted at the same location. Unlike existing approaches, ours adapts a deep learning model which helps achieve high accuracy when inferring locations for individual tweets. To the best of our knowledge, this is the first work on applying a deep learning model to tweet location inference. As Paraskevopoulos and Palpanas's approach [51] is the most recent study to estimate tweet locations at the city level, we compare our approach with it in the experimental study. The results show that ours achieves significantly better location inference results.

3 Research Problem and Solution Framework

This section formulates the research problem and gives our solution framework. The main notations used throughout this paper are given in Table 2.

TABLE 2: Notations

L	The location set of interest
V	The vocabulary that contains every word across L
V_l	The vocabulary of location l that contains every word appearing in l
W	Word sequences of the content of all tweets in a cluster
Δt	The time difference between two consecutive tweets
τ	The temporal period of a tweet cluster
w, w_i	A word in a word sequence W
TL_{train}	All timelines of users in training data
TL_{test}	All timelines of users in testing data
C_{train}	All clusters built for the timelines in TL_{train}
C_{test}	All clusters built for the timelines in TL_{test}

3.1 Problem Definitions

A tweet contains a unique id tid, a user id uid, a timestamp, a short content and some flags that indicate if the update is "original", a reply, or a retweet. The short content of a tweet may contain hashtags. Besides, some tweets are sent with GPS coordinates or place names. In such a case, we can know the actual location where the tweet was sent. With the help of an appropriate geographic information service like Google Map API, the GPS coordinates in a tweet can be translated into a hierarchical string in the format of "Country-State-City-District-Street". In this paper, we infer tweet locations at the city level as this granularity enables a wide range of applications for which locations at other levels are unsuitable. For example, a social network like Twitter can recommend points of interest [2] to a user who is known to be on a visit to a city other than her/his hometown. It is also of interest to recommend to social media users other users who share the same interest and appear in the same city such that local communities [5] may be established in an online-to-offline fashion. A coarser location at the country or state level disables many applications or cannot guarantee the application quality, whereas a finer location at the district and street level tends to cost considerably extra inference cost without improving the application quality. As a result, we regard all GPS coordinates in the same city as the same location and give every city-level location a unique id.

Below, we give some definitions about a Twitter user's data.

Definition 1 (Location). A location l is a 2-tuple l = (*lid*, *city_name*) where *lid* is an id that identifies the city with the name *city_name*.

We use L to denote the set of locations of interest throughout this paper. For simplicity, we sometimes use *lid* only to indicate a location when the context is clear. In other words, we may write $lid_i \in L$ if lid_i is a location id.

Definition 2 (Tweet). A tweet *t* is a 4-tuple *t* = (*uid, ts, content, lid*), where *uid* identifies the user who sent *t*, *ts* is the timestamp when *t* was sent, *content* is the content of this tweet, and *lid* implies the location where *t* was sent.

If a tweet t was sent without a geo-tag, its lid is set to -1. Otherwise, its lid is not -1 and we use *geo-tweets* to refer to such tweets.

Definition 3 (Timeline). A timeline *tl* of a user is a sequence of the user's original tweets, i.e., *tl* contains all the tweets a user has ever sent that are not replies or retweets.

Note that every tweet in a given timeline tl has the same user id. All tweets in *tl* are sorted by timestamps in chronological order. In other words, agelong tweets are firstly pushed to tl and the recent are appended to the end.

We define our research problem as follows.

Location Inference for Non-geotagged Tweets in a Timeline: Given a timeline tl and a location set $L = \{l_1, l_2, ..., l_n\}$ in which the items stand for n cities with unique identifiers, for every tweet t in tl, estimate the probability $p(l|t)$ that tweet t was sent at a location l in L, such that the location with maximum probability $p(l|t)$, i.e., $\hat{l} = \text{maxarg}_{l \in L} p(l|t)$, is the actual location where tweet t was sent.

3.2 Solution Framework

3.2.1 Temporal Clusters of Tweets

Due to the 140-character limit and possible noises, the content of an individual tweet hardly contains sufficient clue for location inference. Nevertheless, more tweets sent from the same location

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2852764, IEEE Transactions on Knowledge and Data Engineering

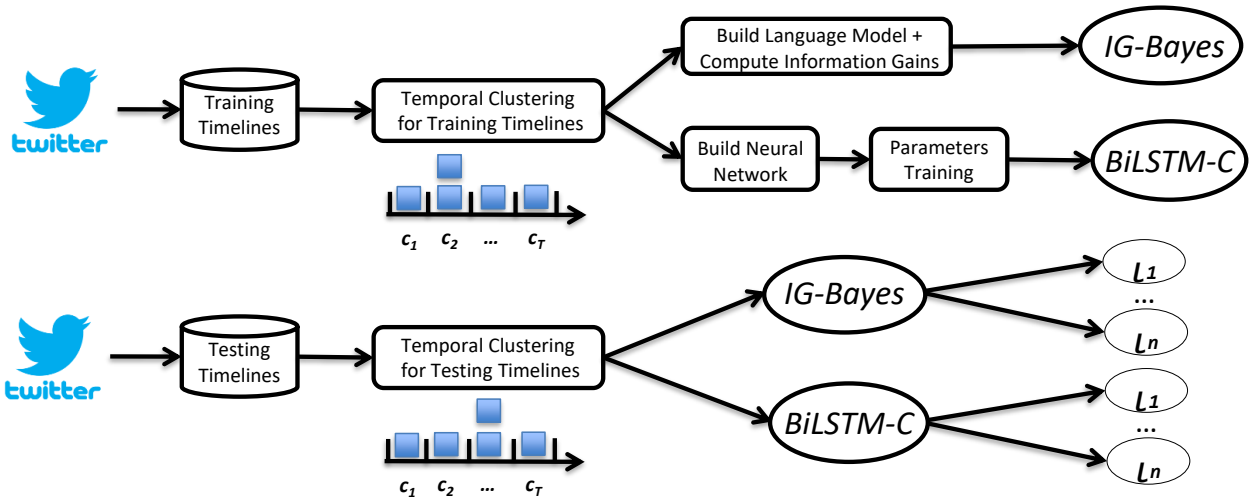


Fig. 1: The proposed framework

may imply more geographical information. A reasonable assumption is that the tweets sent by a user in a short period of time are likely from the same location. If we cluster such tweets together and make use of them collectively, the location inference task can become easier. Motivated as such, we define the concept of tweet clusters as follows.

Definition 4 (Tweet Cluster). A tweet cluster c is a 4-tuple c = (*otl, lid,* Δt, τ), where *otl* is a list of original tweets whose *uid* are the same. All tweets in *otl* are sorted by their timestamps in ascending order. The time distance[3] between any pair of consecutive tweets is less than Δt and the time distance between the first and last tweet in *otl* is less than τ.

In Section 4, we will give the details for finding temporal clusters of tweets based on this definition. We call τ the *temporal period* of cluster c. Ideally, every tweet t in $c.otl$ has the same *lid* except for those tweets whose lid are -1. If $c.lid = -1$, it means that every tweet in $c.otl$ was sent without geo-tag. In the rest of this paper, we focus on inferring the location for a tweet cluster instead of for individual tweets. We then use the inferred location for cluster c as the location for all the tweets in cluster c.

3.2.2 System Overview

Fig. 1 shows the framework of the solution to location inference for tweets. The upper part illustrates how to construct our *information gain based Bayes model* and *bidirectional LSTM convolution model* from training data. The two models are called $IG\text{-}Bayes$ and $BiLSTM\text{-}C$ for short, respectively. After that, we use the two models to solve the tweets location inference problem on real world data (or testing data). The application or testing of the two models is illustrated in the bottom half in Fig. 1.

The first step splits a Twitter user's entire timeline into clusters. We call this step *temporal clustering*, which is different for training and testing since different sets of information are available for training and testing. For training, we make use of the possible GPS coordinates and/or other geo-tags in tweets, whereas in the testing we do not need such implications.

We propose two probabilistic models for location inference that purely rely on tweet content. They both regard the content of a tweet cluster c as a bag-of-words $W = (w_1, ..., w_n)$ and estimate the probability $p(l|W)$ for every location l in L. One model compares the information gain value of a word w that appeared in location l and the average information gain of words in l, measures how closely word w is related to location l, and utilizes this measurement to infer the most possible location for a cluster according to Bayes' theorem. The other model builds a neural network to learn features ϕ of W and computes the probability $p(l|\phi)$. It contains a bidirectional LSTM layer to learn a long dependency of W and a convolution layer to learn spatial local features of phrases in W. Experimental results show that the two models are effective at inferring tweet locations.

3. *Time distance* is the difference between the timestamps of two tweets.

3.2.3 Evaluation Metrics

We compare the location inferred $l_{infer}(t)$ returned by our approach with the actual location $l_{act}(t)$ for each geo-tweet in a user's timeline tl in the testing data TL_{test}. We use $tl.geo\text{-}tweets$ to refer to all the geo-tweets in tl. The more locations correctly inferred by a model across all geo-tweets in TL_{test}, the better the model is. Therefore, we use the metric $accuracy$ (Acc for short) to evaluate a location inference model. Specifically, accuracy in our setting is the percentage of geo-tweets' locations that are inferred correctly.

$$Acc = \frac{\sum_{tl \in TL_{test}} |\{t \in tl.geo\text{-}tweets | l_{infer}(t) = l_{act}(t)\}|}{\sum_{tl \in TL_{test}} |tl.geo\text{-}tweets|}$$

4 Temporal Clustering

The key point of a temporal clustering method is to determine Δt and τ for a user's timeline. The simplest way is to let Δt be a fixed time distance, for example, the shortest flight time that enables a user to fly from one city to another. However, users have different frequencies of updating status on Twitter. Therefore, using a fixed threshold Δt or τ for all users can be too rigid to cope with the real situations. Instead, we use more sophisticated ways to decide Δt and τ and make them adaptive to different users and timelines.

We proceed to present the temporal clustering methods for training data and testing data respectively. We make use of the geo-tweets for training but not for testing.

4.1 Temporal Clustering for Training Data

The key idea of temporal clustering for training data is to combine a geo-tweet and its neighbor tweets to form a cluster. We want to find a proper temporal period τ for every timeline in the training data. For every geo-tweet t_{geo}, we put all the tweets in its $\frac{\tau}{2}$-neighborhood into t_{geo}'s cluster. According to our observation on the collected tweet data, the few tweets preceding or succeeding a geo-tweet are very likely to be sent in the same location as the geo-tweet. Intuitively, a user does not go to a third city during the period of his sending two tweets t_i and t_j if they are located in two cities and the time in-between is not long. However, a user may stay in a city for a long time without sending a (geo-)tweet. Alternatively, a user may send two geo-tweets in the same location in a short period of time. For example, tourists often do so when they visit a city for the first time.

To handle different situations, we define a minimum and maximum temporal period τ_{min} and τ_{max} to bound τ, i.e., $\tau_{min} \leq \tau \leq \tau_{max}$. Let Δt_{min} be the shortest temporal distance between two consecutive geo-tweets with different locations in the timeline. We set τ to be $min(\tau_{max}, max(\tau_{min}, \frac{2}{3}\Delta t_{min}))$. We use a multiplier of $\frac{2}{3}$ to get rid of those tweets with timestamps close to the midpoint of the timestamps of two geo-tweets sent in different cities. Essentially, it is impossible that a training tweet belongs to two adjacent, overlapping clusters. This requires $\tau_{min} < \frac{2}{3}\Delta t_{min}$, which is very rare according to our statistics. We exclude such overlapping clusters to avoid disturbance. On the other hand, there are tweets without geo-tags and not included in any clusters. We have little confidence to judge which city they belong to. They cannot help the training and thus are also excluded from the clustering results. In our implementation, we set τ_{min} and τ_{max} to 1 hour and 6 hours, respectively.

After deciding τ, the temporal clustering for a user's timeline tl for training data is done according to Algorithm 1. All geo-tweets in tl are pushed to a FIFO queue $geo\text{-}tl$ in the temporal order of their timestamps. A list $clusters$ is created to hold the clusters of tl. Each time a geo-tweet t_{geo} is picked up from $geo\text{-}tl$, a new cluster candidate c_{candi} is created. It accepts all tweets in tl whose timestamps fall in $(t_{geo}.ts - \frac{\tau}{2}, t_{geo}.ts + \frac{\tau}{2})$ into c_{candi}. If c_{candi} contains only one geo-tweet or every geo-tweet in it has the same lid, c_{candi} is a legal cluster and it is added into $clusters$. Otherwise, we regard c_{candi} as illegal and increase $error\text{-}count$, i.e., the number of illegal clusters.

The temporal clustering may be disturbed by company accounts shared by colleagues in different locations. For business or branding purpose, those colleagues may post tweets almost simultaneously that are geo-tagged with different locations. This kind of behavior results in tweets sent within τ_{min} but located in different locations. The variable $error\text{-}count$ in Algorithm 1 keeps track of how many cluster candidates contain tweets sent in more than one locations. The returned $error\text{-}count$ should be larger than a threshold value[4] for those timelines of company accounts. In this way, the algorithm is able to remove those timelines of company accounts.

Fig. 2 shows an example of temporal clustering for a timeline in training dataset, where t_1 and t_4 are two geo-tweets. The time distance between t_0 and t_1 and that between t_1 and t_2 are less than $\frac{\tau}{2}$. So the distance between the first tweet (t_0) and last tweet (t_2) in cluster c_0 is less than τ. In contrast, tweet t_3's timestamp is not within the $\frac{\tau}{2}$-neighborhood of any geo-tweet, it belongs to no cluster. Also, tweet t_4 forms a singleton cluster as no tweets fall in its neighborhood.

4. In our implementation, we set the threshold to $\frac{|clusters|}{4}$.

Algorithm 1 Temporal Clustering for Timelines in Training Data

Input:
tl: a user's timeline
$geo\text{-}tl$: an FILO queue of geo-tweets in tl, sorted by timestamps in descending order
τ: temporal period for the clusters to be generated
Output:
$clusters$: a list of clusters for tl
$error\text{-}count$: an integer recording illegal cluster candidates

```
set clusters an empty list
error-count ← 0
while geo-tl is not empty do
    Set c_candi an empty cluster
    t_geo ← geo-tl.pop()
    c_candi.lid ← t_geo.lid
    for all t ∈ tl such that |t.ts − t_geo.ts| < τ/2 do
        c_candi.otl.add(t)
    if ∀t ∈ c_candi.otl : t.lid = t_geo.lid or t.lid = −1 then
        clusters.add(c_candi)
    else
        error-count ← error-count + 1
return clusters, error-count
```

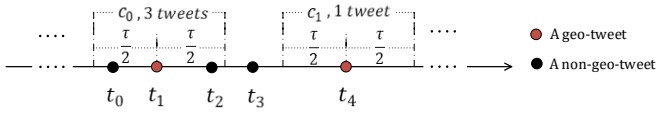


Fig. 2: Converting a timeline (training data) to clusters

4.2 Temporal Clustering for Testing Data

When deciding the temporal clusters for testing data, we treat all tweets in the way as if none of them is geo-tagged. We find the minimum and maximum time distances (Δt_{min} and Δt_{max}, respectively) between any two consecutive tweets in a timeline tl of a user, and split the interval $(\Delta t_{min}, \Delta t_{max})$ into several smaller intervals $(\Delta t_0, \Delta t_1), \ldots, (\Delta t_{n-1}, \Delta t_n)$ where $\Delta t_{min} = \Delta t_0$ and $\Delta t_{max} = \Delta t_n$. The length of every smaller interval is half an hour. For every such half-hour interval, we count the pairs of consecutive tweets whose time distance is within it. Suppose that we get the interval $(\Delta t_{i-1}, \Delta t_i)$ of which the count is the largest. Accordingly, we set $\Delta t = \Delta t_i$. We would like to find users' most likely frequency of sending tweets. By our observation, a user is unlikely to move if she/he sends two tweets within the period of Δt. Also, our statistics show that only a very small fraction (less than 1%) of the clusters in the testing data C_{test} contain more than one geo-tweet with different locations. This verifies the rationality of our temporal clustering for testing data.

Unlike the temporal clustering for training data, the temporal clustering for testing data here does not need the parameter τ. Here, it is possible to achieve the ideal circumstance where every tweet *t* in *tl* has the same *lid* except for $t.lid = -1$. As the location information in those geo-tweets is not needed by the temporal clustering for testing data, the clustering here starts with the time distance between two consecutive tweets instead of the overall temporal period that is indicated by the parameter τ. Algorithm 2 briefly formalizes the procedure of temporal clustering for a user's timeline tl in testing data.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2852764, IEEE Transactions on Knowledge and Data Engineering

Algorithm 2 Temporal Clustering for Timelines in Testing Data

Input:
tl: an FILO sequence of user's timeline sorted by timestamps in descending order
Δt: time distance for the clusters to be generated
Output:
$clusters$: a list of clusters for tl
$error\text{-}count$: an integer recording illegal cluster candidates

set $clusters$ an empty list
$error\text{-}count \leftarrow 0$
$t_{last} \leftarrow tl.pop()$
Set c_{candi} a new empty cluster
$c_{candi}.otl.add(t_{last})$
while tl is not empty **do**
  $t \leftarrow tl.pop()$
  if $t_{last}.ts - t.ts < \Delta t$ **then**
    $c_{candi}.otl.add(t)$
  else
    $clusters.add(c_{candi})$
    Set c_{candi} a new empty cluster
    $c_{candi}.otl.add(t)$
  $t_{last} \leftarrow t$
$clusters.add(c_{candi})$
return $clusters$

The temporal clustering converts the timelines in training data into clusters from which we build our inference models, whereas the temporal clustering converts the timelines in testing data into clusters for which we infer locations. In training data, as one cluster corresponds to one geo-tweet, the number of clusters is the same as that of geo-tweets that do not repeat within a short period of time except those illegal cluster candidates. In testing data, the clusters that contain at least one geo-tweet are used as ground truth in our model evaluation. The temporal clustering for testing timelines ensures that no geo-tweet in a timeline appears in two clusters.

Fig. 3 gives an example of temporal clustering for testing data. The time distance between tweets t_0 and t_1 and that between t_1 and t_2 are less than Δt. Thus, $\{t_0, t_1, t_2\}$ satisfies the conditions of a cluster and the set forms cluster c_0. In contrast, the time distance between t_2 and t_3 is larger than Δt, so t_3 does not belong to c_0 but forms a singleton cluster c_1. A tweet in a testing timeline cannot belong to two clusters, as a testing tweet always starts a new cluster if it is later than the previous cluster's last tweet by at least Δt.

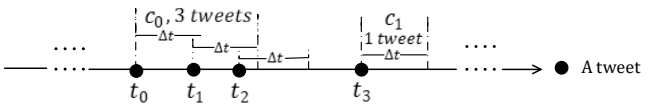


Fig. 3: Converting a timeline (testing data) to clusters

From the presentation above and the illustration in Fig 3, it can be seen that our temporal clustering method for testing data can be applied to an online setting. Suppose we have gotten a user's timeline tl up to now. When he/she posts a new tweet t, we just need to append t to tl, update Δt, and do the temporal clustering again using Algorithm 2. All these operations can be done in a very short time. As a result, given a model that is able to infer locations for clusters in a very short time, individual tweets' locations can be inferred in an online manner.

5 Information Gain Based Bayes Model

In this section, we first give a baseline probabilistic model for location inference that is based only on Bayes' theorem. Subsequently, we explain the drawbacks of the baseline model and present an improved Bayes model for location inference.

5.1 Baseline Bayes Model for Location Inference

First, we consider a basic probabilistic language model that regards a tweet cluster as a word sequence W. Our aim is to estimate the probability that a new word sequence W was issued from a given location by sampling from the word distribution for that location. Let D_l be the distribution of words associated with location l, and $p(l)$ the prior probability of location l. According to Bayes' theorem, we have

$$p(l|W) = \frac{p(W|D_l)p(l)}{p(W)} \tag{1}$$

Assuming a uniform distribution for $p(l)$ is unrealistic as some locations are associated with more tweets. We use an estimation for $p(l)$:

$$\hat{p}(l) = \frac{|\{c \in C_{train} | c.lid = l\}|}{|C_{train}|}$$

Note that $\hat{p}(l)$ gets larger with the increase of tweet density in l. We ignore $p(W)$ from Equation 1, since it is the same for all locations.

For simplicity, we assume all words are independent from each other and we have:

$$p(W|D_l) = \prod_{w_i \in W} p(w_i|D_l)$$

The smoothing *maximum likelihood* estimate of the probability that a word w is "generated" by location l is as follows:

$$\hat{p}_{ml}(w|D_l) = \begin{cases} \frac{wf_{(w,l)}}{|D_l|} & \text{if } wf_{(w,l)} > 0, \\ \frac{TF_w}{TN} & \text{otherwise.} \end{cases}$$

Above, $wf_{(w,l)}$ is the word frequency of w in D_l, and $|D_l|$ is defined as the total number of words in location l, i.e., $|D_l| = \sum_w wf_{(w,l)}$. Furthermore, $TF_w = \sum_l wf_{(w,l)}$ is the total word frequency of w across all locations, and $TN = \sum_w TF_w = \sum_l |D_l|$ is the total number of words in all locations. When estimating the probability given a sequence W, we ignore those words in W that are absent from all locations word distributions. Altogether, we have:

$$\hat{p}_{ml}(W|D_l) = \prod_{w_i \in W} \hat{p}(w_i|D_l)$$
$$\hat{p}_{ml}(l|W) \propto \hat{p}_{ml}(W|D_l) \times \hat{p}(l)$$

5.2 The Improved Bayes Model

There are two problems of the maximum likelihood estimator mentioned above. On the one hand, we cannot get an arbitrary sized sequence of data from the actual words distribution D_l^{actual} over locations. As a result, we cannot be confident in the maximum likelihood estimator [52], [53]. On the other hand, the maximum likelihood estimator treats all words equally, ignoring those words that have more compact geographical scopes and higher implications of locations. To address these two problems, we improve the maximum likelihood estimator and propose an information gain based Bayes model for location inference. We call this improved model $IG\text{-}Bayes$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2852764, IEEE Transactions on Knowledge and Data Engineering

5.2.1 A Robust Probability Estimator

To mitigate the first problem, we adapt the approach described by Song and Croft [52] to our problem. In particular, we define an average estimate of likelihood for a word w over all the locations whose words distribution contains w:

$$\hat{p}_{avg}(w) = \frac{\sum_{l_{(w \in D_l)}} \hat{p}_{ml}(w|D_l)}{|L_w|}$$

Above, L_w denotes the locations set in which every location's words distribution contains w. We call $|L_w|$ the location frequency of word w. This is a robust statistic as the average probability is calculated using considerably more data. As word sequences in different locations were generated by different distributions, there would be no distinction if we use $\hat{p}_{avg}(w)$ in every location and the mean probability would become riskier to use as an estimate. In order to make use of the robustness of average estimate above and to keep the pattern of word distributions, we model the risk for a word w generated by D_l using the geometric distribution [52], [53]:

$$\hat{R}(w,l) = \left(\frac{1.0}{1.0+\bar{f}_w}\right) \times \left(\frac{\bar{f}_w}{1.0+\bar{f}_w}\right)^{wf(w,l)}$$

Above, $\bar{f}_w$ is the average word frequency of word w in locations where w appears: $\bar{f}_w = \hat{p}_{avg}(w) \times |D_l|$.

The estimate of the probability that a word w is "generated" by location l is then as follows [53]:

$$\hat{p}_r(w|D_l) = \begin{cases} \hat{p}_{ml}(w|D_l)^{(1-\hat{R}(w,l))} \times \hat{p}_{avg}(w)^{\hat{R}(w,l)} & \text{if } wf(w,l) > 0, \\ \frac{TF_w}{TN} & \text{otherwise.} \end{cases}$$

Finally, we use Dirichlet smoothing [54] on $\hat{p}_r$ to get a smooth estimate $\hat{p}_\mu$:

$$\hat{p}_\mu(w|D_l) = \frac{wf_{(w,l)} + \mu \hat{p}_r(w|D_l)}{|D_l| + \mu}$$

In this estimate, we set $\mu = 0.5|D_l|$.

5.2.2 The Model Based on Local Measure of Words

It is still insufficient to only replace $\hat{p}_{ml}$ by $\hat{p}_\mu$. It is unreasonable to treat all words equally because some words may only appear at some particular locations. For example, "Louvre" can give the model a clear hint that the sequence was very likely to be in Paris. In contrast, "morning" cannot offer any local information. "Louvre" may appear at the same times as "morning" in Paris; but in other cities, "morning" may appear much more often than "Louvre". As a result, $\hat{p}_\mu(w|D_l)$ will be the same for "Louvre" and "morning" if the location l is Paris. Moreover, if $\bar{f}_{\text{"morning"}} = wf(\text{"Louvre"}, \text{Paris})$, the robust estimate $\hat{p}_\mu(w|D_l)$ will be useless.

When we estimate the probability of a sequence generated by the words distribution in a location, we want to emphasize the effect of the words that have close ties with one or more locations. In a previous work [12], a word like "Louvre" is called a "local" word. However, other words like "museum" seem to be related to some locations but it may be impossible to infer the location of a sequence by only using such a word. Instead of merely determining if a word is "local", we want to estimate to what extent a word is related to a location l. To this end, we propose a metric that calculates the normalized information gain of a word w with respect to a location l.

A measure of chaos in a set, information entropy shows the uncertainty of relationship between data in the set and one of the data properties. Related to that, information gain is a measure of how much uncertainty a feature of the data can help to reduce. Applying these concepts to our problem, we regard that "a cluster c is located in a location l" as a property and every word in c's content as a feature.

Given a set C_{train} of training clusters and a set L of locations, we define a random variable X_l to capture if a cluster c in C_{train} satisfies $c.lid = l$. The possible values of X_l are only 1 or 0. Consequently, the information entropy of the set C_{train} for a location $l \in L$ is as follows:

$$\begin{aligned} H_l(C_{train}) &= -\sum_{i \in \{0,1\}} p(X_l = i) \log p(X_l = i) \\ p(X_l = 1) &= \frac{|\{c \in C_{train} | c.lid = l\}|}{|C_{train}|} \\ p(X_l = 0) &= 1 - p(X_l = 1) \end{aligned}$$

Above, $H_l(C_{train}) = 0$ means every cluster's location is either l or not l.

We regard the word w's appearance in a cluster c's content W as an attribute c in C_{train} and we use A_w to represent it. If $A_w = 1$, w appears in c's content; otherwise, $A_w = 0$. The information gain for an attribute A_w is defined in terms of entropy $H(C_{train})$ as follows:

$$\begin{aligned} &IG(w,l) = H_l(C_{train}) - H_l(C_{train}|A_w) \\ &= H(C_{train}) - \\ &\sum_{i \in \{0,1\}} \frac{|\{c \in C_{train} | A_w = i\}|}{|C_{train}|} H(\{c \in C_{train} | A_w = i\}) \end{aligned}$$

As the information gain is used to measure how an attribute can decrease the degree of "chaos" in a set, if a word w has a compact geographical scope, $IG(w,l)$ will be larger for location l with which w has close ties.

Last, we normalize the information gain of a word w over the location set L to estimate the extent that w can be associated with locations. The estimation $\epsilon(w,l)$ is defined as:

$$\epsilon(w,l) = \frac{\bar{IG}_{exp}(l)}{e^{IG(w,l)}}, \text{where } \bar{IG}_{exp}(l) = \frac{\sum_{w \in V_l} e^{IG(w,l)}}{|D_l|}$$

Above, V_l contains all words that have appeared in location l.

We use $\epsilon(w,l)$ as a influence factor and the final probability estimator becomes:

$$\hat{p}(w|D_l) = \hat{p}_\mu(w|D_l)^{\epsilon(w,l)}$$

As $\hat{p}_\mu(w|D_l)$ is in the range $(0,1)$, when w is closely related to l, $\epsilon(w,l)$ will be smaller than 1 and $\hat{p}(w|D_l)$ will be larger than $\hat{p}_u(w|D_l)$. Otherwise, $\hat{p}(w|D_l)$ will get smaller. Naturally, the following holds:

$$\hat{p}(l|W) \propto \left(\prod_{w_i \in W} \hat{p}\left(w|D_l\right)\right) \times \hat{p}(l)$$

Therefore, we select the location l as the inferred location for a given cluster c such that $\hat{p}(l|W)$ is the largest among L.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2852764, IEEE Transactions on Knowledge and Data Engineering

6 BIDIRECTIONAL-LSTM CONVOLUTION MODEL

Although easy to train, the model based on Bayes' theorem is not sufficiently sophisticated to extract geographical features from tweet clusters. In contrast, Deep Neural Networks (DNNs) are extremely powerful models to solve many difficult problems because DNNs can approximate arbitrarily complex functions. Among many kinds of DNNs, Recurrent Neural Network (RNN) [55], [56] is specialized for handling sequential data, which is the case for our tweet data. In addition, by taking advantage of local feature extraction of the convolution operation, we are able to build a network consisting of an RNN layer and a convolution layer.

In this section, we first briefly describe the architecture of a standard RNN and one popular RNN called Long Short-Term Memory (LSTM) [57]. Then we present the architecture of our Bidirectional LSTM Convolution model ($BiLSTM\text{-}C$) that is suitable for our tweet location inference problem.

6.1 Background: RNN and LSTM

The RNN is a natural extension of feedforward neural network on modeling sequence. Given an input sequence $X = (x_1, x_2, ..., x_T)$ where each x_i is a fixed-dimensional vector, a standard RNN maps X to a sequence of hidden states $\mathcal{H} = (h_1, h_2, ..., h_T)$, and $\mathcal{H}$ to an output sequence $Y = (y_1, y_2, ..., y_T)$. The two mappings are done by iterating the following formulas from time $t = 1$ to time $t = T$:

$$h_t = \boldsymbol{tanh}(W_{hx}x_t + W_{hh}h_{t-1})$$
$$y_t = W_{yh}h_t$$

Although standard RNNs can learn complex temporal dynamics, it is difficult to learn long-term dynamics. Compared with a standard RNN, an LSTM has more complicated dynamics that enable it to "memorize" history information observed up to that time step. In this sense, an LSTM is capable of learning long-term dependencies. The core of an LSTM is the cell c_t that stores "long term" memory. An LSTM is regulated by components called gates that overwrite c_t, retrieve it, or keep it for the next time step. Gates are composed out of a sigmoid or tanh layer and a point-wise multiplication operation. The value of a sigmoid function lies within the range $(0, 1)$. So, gates can optionally let information through.

Fig. 4 (from [58]) illustrates how an LSTM computes $\mathcal{H}$. Specifically,

$$i_t = \boldsymbol{sigm}(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$f_t = \boldsymbol{sigm}(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$o_t = \boldsymbol{sigm}(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

$$g_t = \boldsymbol{tanh}(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \boldsymbol{tanh}(c_t)$$

Here, i, f, o and g are the *input gate*, *forget gate*, *output gate*, and *input modulation gate*, respectively; c is *cell activation vector*. Their size is identical to that of the hidden vector h_t.

The memory cell c_t is a summation of two parts: the previous memory cell c_{t-1} modulated by forget gate f_t, and the current input and previous hidden state i_t modulated by input modulation gate g_t. The layers i_t and f_t output numbers between 0 and 1, describing how much of each component should be let through.

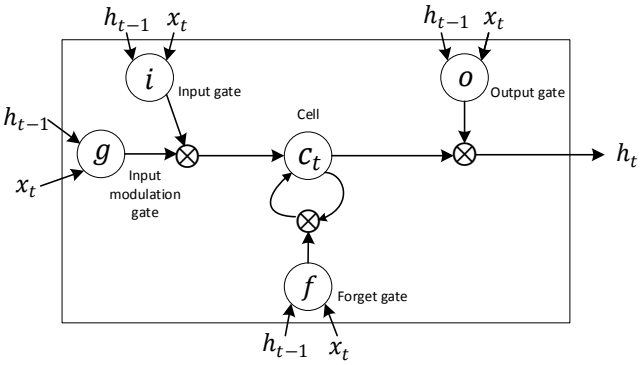


Fig. 4: A graphical representation of LSTM [58]

Therefore, an LSTM is able to learn to selectively forget its previous memory or to consider its current input. Likewise, o_t helps an LSTM learn how much of the memory cell to transfer to the hidden state. If a network stacks more than one LSTMs, the hidden state in layer l at time step t is usually written as h_t^l. Then the input x_t for a higher LSTM layer is h_t^{l-1}, as shown in Fig. 4.

One shortcoming of conventional LSTMs is that they cannot exploit future contexts. To this end, Bidirectional RNNs (BRNNs) are designed to process data in two directions with two separate hidden layers, as illustrated in Fig. 5. Combining BRNNs and LSTMs gives Bidirectional LSTMs (B-LSTM) [13].

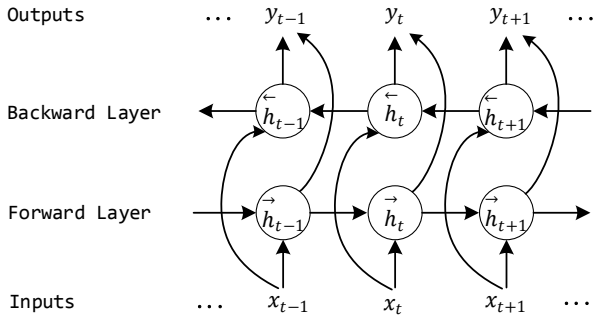


Fig. 5: Bidirectional RNN [13]

6.2 The BiLSTM-C Model

As in Section 5, we also regard a tweet cluster c as a word sequence W. By converting words into real number vectors, our problem can be translated into a sequence classification problem that an RNN-based neural network is suitable to solve. The goal of a network for sequence classification is to estimate the conditional probability $p(label|X = (x_1, x_2, ...x_T))$ where $label$ is the category to which X belongs. In practice, every sequence's length may be different. It is crucial to extract fixed dimensional features that express the whole sequence X. Using RNN-based networks can solve such classification problems about variable sequences.

An RNN calculates the X's corresponding hidden states of the final LSTM layer $\mathcal{H} = (h_1, h_2, ..., h_T)$, builds a feature function F that requires as input the sequence $(h_1, h_2, ..., h_T)$ and returns a fixed-dimensional feature vector ϕ, and computes the probability of $label$ given ϕ:

$$p(label|x_1, x_2, ...x_T) = p(label|\phi = F(h_1, h_2, ..., h_T))$$

Generally, we use some simple layers such as fully connected and relu layers followed by a softmax layer to handle ϕ and estimate $p(label|\phi)$.

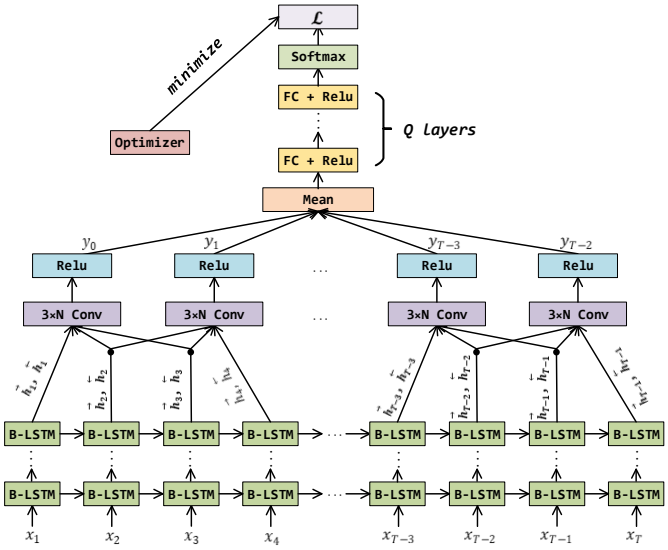


Fig. 6: $BiLSTM$-C Model for Location Inference

We adopt the idea mentioned above to solve our problem. Fig. 6 shows the architecture of our $BiLSTM$-C model. The key point is still how to extract features for the sequence W. First of all, we need to convert a sequence of words W into a sequence of fixed-dimensional vectors. In particular, we extract the content of all tweets of each timeline in our training data C_{train} and use the skip-gram algorithm [59] to train these word vectors. Consequently, each word is expressed as an M-dimensional vector of float numbers, where M is an empirical value that has little effect on the overall model performance. It is set to 512 in our experiments. Subsequently, we express the word sequence W of a cluster as a word vector sequence $\mathcal{V} = (v_1, v_2, ..., v_T)$. This $\mathcal{V}$ serves as the vectorization for the word sequence W in a given cluster. Each v_t in $\mathcal{V}$ is the word w_t's M-dimensional vector and the length of the sequence $\mathcal{V}$, i.e., $|\mathcal{V}|$, is T. Now we have the input $X = \mathcal{V}$.

We use a deep bidirectional LSTM layer to compute the N-dimensional hidden states sequences $\overrightarrow{\mathcal{H}}$ and $\overleftarrow{\mathcal{H}}$ of $\mathcal{V}$, where N is the number of hidden units used in our model. Therefore, $\overleftarrow{h_t}$ and $\overrightarrow{h_t}$ are v_t's two hidden states and they are also the word w_t's hidden states. Sometimes, an individual word cannot give clear location clues but word groups or phrases have close ties with some particular locations. For example, "statue" or "liberty" can be used everywhere while "Statue of Liberty" is the landmark of New York City. In this sense, if we can combine word groups together by considering such local features, more powerful features would be extracted for location inference.

Convolution Neural Networks (CNNs) consider spatially-local correlation in an image by enforcing a local connectivity pattern between neurons of adjacent layers by a convolution operation [14], [15], [16]. Inspired by this, we add a convolution layer above the LSTM layer to exploit the local features. Concatenating every vector in $\overrightarrow{\mathcal{H}}$ and $\overleftarrow{\mathcal{H}}$, we get two matrices $\overrightarrow{\boldsymbol{H}}_{T\times N}$ and $\overleftarrow{\boldsymbol{H}}_{T\times N}$, respectively. Subsequently, we convert the combination of $\overrightarrow{\boldsymbol{H}}_{T\times N}$ and $\overleftarrow{\boldsymbol{H}}_{T\times N}$ into a $T\times N\times 2$-dimensional tensor $\boldsymbol{H}$ that can be viewed as a 2-channel image with height T and width N. Using one filter $\boldsymbol{K} \in \mathbb{R}^{3\times N}$ to convolute $\boldsymbol{H}$, followed by a nonlinear rectified linear unit (relu) operation, we get a $(T\text{-}2)\times N$-dimensional output "feature map" Y. Different from traditional bidirectional RNNs which generate output sequence Y by directly concatenating $\overrightarrow{\mathcal{H}}$ and $\overleftarrow{\mathcal{H}}$, we use a convolution layer and a relu layer to map the forward hidden states $\overrightarrow{\mathcal{H}}$ and backward hidden states $\overleftarrow{\mathcal{H}}$ into output sequence Y. By computing the mean of elements in Y across the first dimension, we get the fixed N-dimensional feature ϕ:

$$\boldsymbol{H}_{::0} = \begin{pmatrix} \overrightarrow{h}_0 \\ \overrightarrow{h}_1 \\ \vdots \\ \overrightarrow{h}_T \end{pmatrix} \qquad \boldsymbol{H}_{::1} = \begin{pmatrix} \overleftarrow{h}_0 \\ \overleftarrow{h}_1 \\ \vdots \\ \overleftarrow{h}_T \end{pmatrix}$$

$$\phi = \boldsymbol{Mean}\big(\boldsymbol{Relu}\left(\boldsymbol{K} * \boldsymbol{H}\right)\big)$$

To make it easy to understand the architecture, we unroll the convolution and a following relu layer in Fig. 6. As a matter of fact, all the $3\times N$ convolution layers use the same filter $\boldsymbol{K}$.

The following fully connected layers and nonlinear rectified linear units ($Relu(x) = max(0, x)$) are used to handle the feature ϕ. On the top of those layers, the softmax layer outputs the probability estimates that cluster c with content W was posted at every location in L. The output $\boldsymbol{p_l}$ falls in $[0, 1]^{|L|}$. In particular, we have

$$h^q(\mathbf{x}) = \boldsymbol{Relu}(\boldsymbol{W}^q h^{q-1}(\mathbf{x}) + \boldsymbol{b}^q)$$

$$\boldsymbol{p_l}(W) = softmax\Big(h^Q\big(...h^2\big(h^1(\phi)\big)\big)\Big)$$

Above, h^q is a fully connected layer followed by relu units and Q is the total number of such layers. Finally, we construct the commonly used cross entropy as the supervised loss $\mathcal{L}$ for the softmax outputs of location inference,

$$\mathcal{L} = -\sum_{c\in C_{test}} log(\boldsymbol{p_l}[c.lid])$$

Above, $\boldsymbol{p_l}[c.lid]$ is the probability estimate that cluster c is located at the location identified by $c.lid$.

The aim of our $BiLSTM$-C model is to minimize the loss $\mathcal{L}$ defined above. Neural networks are often trained using an optimizer by performing SGD (stochastic gradient descent) with mini-batch. We first get the sets C_{train} according to Algorithm 1. Subsequently, we sample batches from C_{train}, feeding them to our $BiLSTM$-C model. Furthermore, we take gradient decent steps to optimize the supervised loss $\mathcal{L}$. We repeat this procedure for a number of iterations until the loss $\mathcal{L}$ converges. After that, we can apply it on testing data, for which the clusters are generated by Algorithm 2.

6.3 Alternative LSTM

An existing variant of LSTM called $ConvLSTM$ [60] also combines convolution and neural network. It uses convolution operations instead of fully-connected layers inside gates i, f, o and g. This model is suitable for those problems whose data input formats are sequences of 3D tensors such as videos. Formally, the input $\boldsymbol{\mathcal{X}}$ is in the form of $\boldsymbol{\mathcal{X}} = (\mathcal{X}_1, .., \mathcal{X}_T)$ where every $\mathcal{X}_t \in \mathbb{R}^{C\times H\times W}$. When $\mathcal{X}_t$ is a frame in a video, C, H and W are the number of channels, height and width of this picture,

respectively. $ConvLSTM$ aims to find local features inside the 3D tensor $\mathcal{X}_t$ at every timestep t. If we use $ConvLSTM$ to replace our $BiLSTM\text{-}C$ model for tweet location inference, as the input $\mathcal{V}$ is a sequence of M-dimensional vectors, we have to expand the dimensions of every item v_t in $\mathcal{V}$ to be an $M \times 1 \times 1$ tensor. In this case, the convolution operation will degenerate into a fully-connected layer with most items being 0 in the weights matrix. In addition, compared to $ConvLSTM$, our $BiLSTM\text{-}C$ takes the continuous word vectors of phrases such as "liberty of statue" into consideration. Our experiments disclose that $BiLSTM\text{-}C$ achieves better performance on the problem of tweets location inference than $ConvLSTM$. Details are given in Section 7.

7 Experiments

In this section, we report on an experimental study to compare models $Baseline$ (Section 5.1), $IG\text{-}Bayes$ (Section 5.2) and $BiLSTM\text{-}C$ (Section 6.2). We also compare our approaches with existing methods. First, we include a probabilistic model [12] that applies two optimization (Local Filtering and Neighborhood smoothing) on a baseline maximum likelihood estimation. We call it $BLFN$, short for *baseline with local filtering and neighborhood smoothing*. Next, to study the efficiency of the convolution layer of the model $BiLSTM\text{-}C$, we build another neural network model, named $B\text{-}LSTM$, that only omits the convolution layer from $BiLSTM\text{-}C$. Besides, we include the Logistic Regression model (LR) in the comparison since it is often used as a baseline for deep neural network models. Different from an RNN-based model, LR requires that the dimension of its input X_{LR} must be fixed. So we average all word vectors of an input word sequence W, i.e., $X_{LR} = \frac{1}{T}\sum_{t=1}^{T} v_t$ where v_t is w_t's word vector and the length of W is T. In addition, we implement $ConvLSTM$ [60] as discussed in Section 6.3. It uses convolutional structures instead of fully-connected layers in both the input-to-state and state-to-state transitions. Last, we include the state-of-the-art approach called $TG\text{-}TI\text{-}C$ [51] that infers tweet locations using similarity comparison between a tweet and a set of geo-tagged tweets. All models are trained and evaluated on some Linux servers with Nvidia Tesla M40 graphics cards.

7.1 Experimental Setting

The open-source library *twitter4j* was used to access Twitter's open API to crawl data. In total, 1,203,467 user timelines from January 2016 to March 2016 were collected, involving 140,345,512 tweets totally. Only 1.9 percent of the tweets were geo-tagged. We did not use all the data but focused on the cities with the most geo-tweets. Two scenarios were considered in our experiments:

- It is of interest to investigate the performance of the models when there are large amounts of data for every city in a small location set L. We selected 10 cities worldwide associated with the most geo-tweets to form L, ignoring other locations and corresponding tweets. The accuracy of location inference for the 10 cities was computed. The top 10 cities are *San Francisco, Los Angeles, London, New York, Seattle, Santa Clara, Vancouver, Portland, Toronto* and *San Diego*.
- Alternatively, we selected top 100 cities in the United States with the most geo-tweets to form a large L. All of the top 100 cities in US have large populations.

In order to build training and testing datasets, we picked up all user timelines that contain at least one geo-tweet from the top-10 worldwide cities or top-100 US cities. We removed those timelines from company accounts, which was done by the temporal clustering approach for training data (Section 4.1). In each remaining timeline, we only keep those tweets in English, by using a language detection tool[5]. After the preprocessing, we divided the remaining timelines into training and testing datasets by a ratio of $4:1$. The training and testing datasets were then fixed for all of the models. After temporal clustering for training (or testing) data, we removed those clusters without any geo-tweet as they cannot be used for training (or evaluation) purpose. Detailed statistics about the datasets used in our experiments are given in Table 3.

TABLE 3: Dataset statistics

10 cities	number of training clusters	615,607
	average number of tweets in a training cluster	4.08
	number of testing clusters	89,268
	average number of tweets in a testing cluster	3.69
	number of total geo-tweets in testing clusters	105,247
100 cities	number of training clusters	1,098,824
	average number of tweets in a training cluster	4.09
	number of testing clusters	163,078
	average number of tweets in a testing cluster	3.71
	number of total geo-tweets in testing clusters	186,391

In the top-10 worldwide cities, there are 548 testing clusters that contain more than one geo-tweets of different locations. The number is 957 in the top-100 US cities. Such bad clusters account for only about 0.61% and 0.59% for the two datasets, respectively. It implies that our temporal clustering approach is reasonable to a great extent.

7.2 Descriptions of Training

In order to train $IG\text{-}Bayes$, we removed stopwords[6] in each word sequence. In contrast, we simply replaced each stopword with a </s> symbol when training $BiLSTM\text{-}C$, since stopwords are often retained when an RNN is used.

Considering that a neural network with deeper layers works better in general, we used a stacked dynamic bidirectional LSTM with 4 layers when training $BiLSTM\text{-}C$. In particular, 512 hidden units at each layer and 512-dimensional word embeddings were used, with an input vocabulary of 545,576. A dynamic LSTM was selected because the sequence lengths are different and the numbers of cells may differ in training iterations. The sequence length varies from 3 to 200. The sequence longer than 200 or shorter than 3 were removed. Thus, 1,536 to 102,400 real numbers were used to represent a sequence in the network. More details are as follows:

- We initialized the parameters of the bidirectional LSTM and all the fully connected layers' parameters with Gaussian noise with mean being 0 and standard deviation set to 0.01.
- We initialized the initial state of the bidirectional LSTM with zero.
- If a sequence's length cannot be divided by 5 with no remainder, we complemented the sequence by </s>. We used batches of 128 sequences for back propagation and

5. http://polyglot.readthedocs.io/en/latest/Detection.html
6. https://www.ranks.nl/stopwords

every sequence in a mini batch has the same length. So there are $\frac{(200-5)}{5} + 1 = 40$ types of batches of different lengths.

- As LSTMs tend to suffer from exploding gradient problem, we enforced a hard constraint on the norm of gradient [0, 5] by scaling it when its norm exceeded a threshold.
- We used a momentum of 0.9 [61], and a dropout keep probability of 0.9 [62] when training $BiLSTM\text{-}C$. These are not involved when applying the model on testing data.
- To avoid overfitting, we added an $l2$-regularization term $\mathcal{L}_{reg}$ on the original supervised loss $\mathcal{L}$. The coefficient λ of $\mathcal{L}_{reg}$ was set to 0.0005 in the beginning and was divided by 10 every time when 48,000 training iterations were finished.
- We performed SGD with mini-batch RMSProp [63], started with a learning rate of 0.01 and divided it by 10 every time when 48,000 training iterations were finished. The objective loss this optimizer minimize is $\mathcal{L} + \lambda\mathcal{L}_{reg}$.
- All neural network-based approaches were trained with 4 different initializations and the accuracies of them are the averages over the corresponding 4 runs.

7.3 Experimental Results

7.3.1 Overall Results

Table 4 reports the accuracies of the approaches in comparison using the top-10 worldwide cities and the top-100 US cities ($Acc@10$ and $Acc@100$, respectively). All approaches but $TG\text{-}TI\text{-}C$ adopts the temporal clustering method introduced in Section 4 to convert group individual tweets into clusters.

TABLE 4: The performance of different models

Approach	Acc@10 (%)	Acc@100 (%)
$TG\text{-}TI\text{-}C$	48.20	33.61
$Baseline$	56.48	32.21
$BLFN$	59.23	38.76
$\boldsymbol{IG\text{-}Bayes}$	**63.66**	**42.84**
LR	64.86	43.85
$ConvLSTM$	68.27	46.84
$B\text{-}LSTM$	73.39	50.99
$\boldsymbol{BiLSTM\text{-}C}$	**76.87**	**54.38**

Referring to Table 4, $TG\text{-}TI\text{-}C$ and $Baseline$ get the worst overall performance and $BLFN$ performs a little better than both of them. Compared to $BLFN$, $IG\text{-}Bayes$ clearly improves the accuracy of location inference: from 59.23% to 63.66% on the top-10 worldwide cities and from 38.76% to 42.84% on the top-100 US cities. This indicates that our information gain based design in $IG\text{-}Bayes$ is able to make considerably better probability estimates when inferring tweet locations.

Although $IG\text{-}Bayes$ outperforms $Baseline$, $TG\text{-}TI\text{-}C$ and $BLFN$, it cannot match LR or any of the three RNN-based models ($B\text{-}LSTM$, $ConvLSTM$ and $BiLSTM\text{-}C$). Furthermore, the RNN-based models are considerably better than LR on both datasets: LR's Acc@10 is 64.86% whereas that of $B\text{-}LSTM$, $ConvLSTM$and $BiLSTM\text{-}C$ is 73.39%, 68.27% and 76.87%, respectively; LR's Acc@100 is 43.85% whereas that of $B\text{-}LSTM$, $ConvLSTM$ and $BiLSTM\text{-}C$ is 50.99%, 46.84% and 54.38%, respectively. These performance differences clearly demonstrate the power of LSTM used in all RNN-based models and that of the convolution layer used in $BiLSTM\text{-}C$. In addition, the fact that $ConvLSTM$ performs worst among the three RNN-based models indicates $ConvLSTM$ is not very suitable to solve this problem.

Besides, the accuracy achieved by each model on top-10 worldwide cities is much higher than its accuracy on top-100 US cities. This is attributed to the fewer location classes using the top-10 worldwide cities, which makes the classification relatively easier.

7.3.2 The Influence of τ and Δt in Temporal Clustering

It is evident from Section 4 that τ and Δt are self-adaptive parameters in our algorithms, i.e., their values may vary with different user timelines. This is reasonable as different users have different frequencies of sending tweets. Nevertheless, we vary the values for τ and Δt, obtain different training and testing clusters for each fixed value pairs, and run the $IG\text{-}Bayes$ and $BiLSTM\text{-}C$ classifications on the top-10 worldwide cities and the top-100 US cities. Table 5 shows the performance associated with different values of τ and Δt.

TABLE 5: Accuracies of $BiLSTM\text{-}C$ and $IG\text{-}Bayes$ in different settings of τ and Δt

$IG\text{-}Bayes$	$Acc@10$ (%)	τ=1h	τ=2h	τ=4h	τ=6h
	Δt=1h	57.84	58.35	59.78	59.14
	Δt=2h	58.42	58.95	60.23	59.57
	Δt=4h	58.78	59.41	60.71	60.33
	Δt=6h	59.03	59.73	60.38	59.91
	$Acc@100$ (%)	τ=1h	τ=2h	τ=4h	τ=6h
	Δt=1h	38.64	39.32	39.03	38.82
	Δt=2h	39.15	40.43	39.98	39.61
	Δt=4h	38.00	40.48	40.01	38.86
	Δt=6h	37.46	39.78	38.83	38.10
$BiLSTM\text{-}C$	$Acc@10$ (%)	τ=1h	τ=2h	τ=4h	τ=6h
	Δt=1h	67.32	67.52	68.26	68.64
	Δt=2h	68.93	69.38	69.58	71.20
	Δt=4h	70.54	70.81	72.20	71.91
	Δt=6h	68.61	70.37	70.58	69.09
	$Acc@100$ (%)	τ=1h	τ=2h	τ=4h	τ=6h
	Δt=1h	44.96	45.47	46.07	45.88
	Δt=2h	46.16	46.72	47.38	47.25
	Δt=4h	48.36	49.28	51.65	50.84
	Δt=6h	47.89	48.32	49.28	49.61

Referring to Table 4, with adaptive τ and Δt, $IG\text{-}Bayes$'s $Acc@10$ and $Acc@100$ are 63.66% and 42.84%, respectively, clearly better than the counterparts reported in Table 5. For $BiLSTM\text{-}C$, its $Acc@10$ and $Acc@100$ are 76.87% and 54.38%, respectively, when τ and Δt are adaptive, also clearly better than the counterparts in Table 5. These comparative experimental results indicate that both $IG\text{-}Bayes$ and $BiLSTM\text{-}C$ with self-adaptive τ and Δt outperform the corresponding variants with fixed τ and Δt.

7.3.3 Features Generated by BiLSTM-C

In order to understand why RNN-based models such as $BiLSTM\text{-}C$ work well, we conducted experiments to observe whether an RNN-based model can extract good features from data. We used $BiLSTM\text{-}C$ to capture the 512-dimensional features ϕ for every testing cluster in top-5 cities worldwide with most geo-tweets. Due to the high number of dimensions, we used t-SNE [64] transformation to visualize ϕ in Fig. 7.

As we can see from Fig. 7, only a very small part of the data in the center contains chaos. According to our observation, the contents in the clusters displayed in the center either have no tie

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2852764, IEEE Transactions on Knowledge and Data Engineering

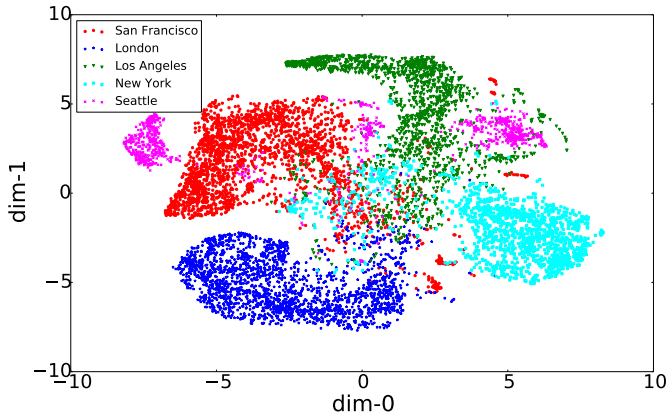


Fig. 7: 2-dimensional t-SNE projection of features generated by $BiLSTM\text{-}C$ for the testing clusters in top-5 cities

with any locations or are noises. Any local clue can be hardly found in the clusters. Except for the slight chaos, most clusters that come from the same cities are adjacent to each other and form clusters as shown in Fig. 7. Therefore, it is reasonable to say that our $BiLSTM\text{-}C$ model is able to extract good features of a sequence.

On the other hand, we discovered that $BiLSTM\text{-}C$ can extract more powerful features than $B\text{-}LSTM$ by finding some phrases with geographical clues. Three examples in the testing clusters are given in Table 6. For these tweet contents, $B\text{-}LSTM$ makes wrong location inferences but $BiLSTM\text{-}C$ works correctly. Note that "Stone Brewing World Bistro & Gardens", "Statue Liberty" and "Staples Center" in these tweets have very compact geographical scopes and strong ties to cities. The additional convolution layer in $BiLSTM\text{-}C$ is able to process them appropriately.

TABLE 6: The content of some clusters inferred correctly by $BiLSTM\text{-}C$ but wrongly by $B\text{-}LSTM$

City	Tweets
San Diego	1. We have arrived Stone Brewing World Bistro & Gardens http://t.co/8lsfx3ev
NYC	1. I'm at statue liberty - @designbyikea. 2. @ andaz wall street http://t.co/av3t8uq3kl
LA	1. Showtime!!! #summerslam #wwe #cenawinsweriot @ Staples Center http://t.co/a3fgss6erx 2. He's alive! ultimate warrior is back!!!#wwe

7.3.4 The Influence of Training Dataset Size

In order to investigate how the amount of training data affects the performance of location inference, we designed an experiment as follows. We randomly picked up 10%, 20%, .., 90% and 100% of training clusters to train our $IG\text{-}Bayes$ and $BiLSTM\text{-}C$. The comparative results of the two models with varying amount of training data are reported in Fig. 8.

Clearly, both $BiLSTM\text{-}C$ and $IG\text{-}Bayes$ work better as the amount of training data increases. With more training data, the models are able to extract the features better and then make probability estimates more accurately. Note that, $BiLSTM\text{-}C$ is slightly more sensitive to the amount of training data, as the more layers in it collectively are able to make better use of more data.

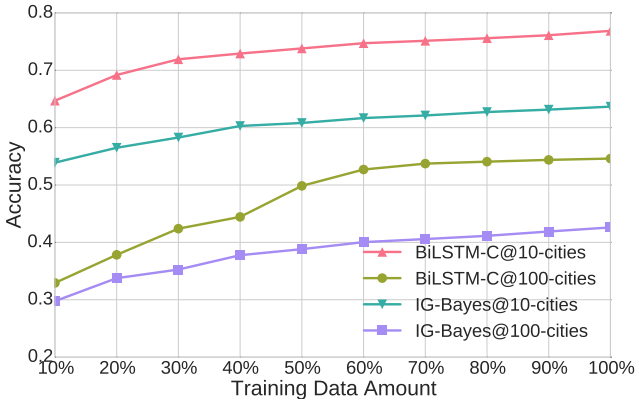


Fig. 8: Accuracies of $BiLSTM\text{-}C$ and $IG\text{-}Bayes$ on two kinds of dataset with varying amount of training dataset

7.3.5 The Influence of Word Sequence Length

We also conducted experiments to analyze how the sequence length affects the performance of $BiLSTM\text{-}C$ and $IG\text{-}Bayes$. Based on the sequence length, we divided all the testing clusters of top-10 world cities and top-100 US cities into two lists of small parts, each consisting of sequences with length in a specific range. The accuracy results are shown in Fig. 9.

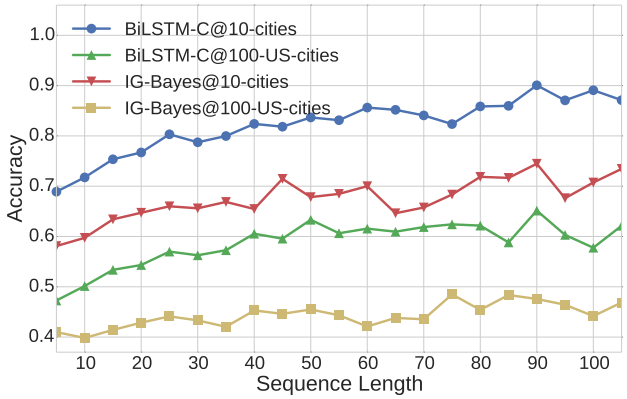


Fig. 9: Accuracies of $BiLSTM\text{-}C$ and $IG\text{-}Bayes$ on two kinds of dataset with varying sequence length

It is evident from Fig. 9 that both $BiLSTM\text{-}C$ and $IG\text{-}Bayes$ work better in general as the sequences become longer. For short sequences, especially those whose length is shorter than 10, $BiLSTM\text{-}C$ performs much better than $IG\text{-}Bayes$, which indicates that $BiLSTM\text{-}C$ can learn good representations even for very short sequences.

7.3.6 The Improvement by Temporal Clustering

From Fig. 9, it is also figured out that temporal clustering makes a significant contribution to the location inference. Due to the 140 characters limit in Twitter, we cannot obtain long word sequences without applying the temporal clustering methods. Table 7 shows the substantial performance improvement enabled by temporal clustering. Here, "$w\ TC$" means that temporal clustering was used in the model while "$w/o\ TC$" means the opposite.

7.3.7 Discussion on Model Choosing

Although $BiLSTM\text{-}C$ outperforms $IG\text{-}Bayes$ in terms of location inference accuracies, the performance gain is not free. In

TABLE 7: Effect of temporal clustering on performance

Approach		Acc@10(%)	Acc@100(%)
$IG\text{-}Bayes$	$w/o\ TC$	54.51	37.65
	$w\ TC$	63.66	42.84
$BiLSTM\text{-}C$	$w/o\ TC$	63.93	45.54
	$w\ TC$	76.87	54.38

our experiments, the training time used for $BiLSTM\text{-}C$ is much longer than that for $IG\text{-}Bayes$. It took approximately 40 hours (60 hours) for $BiLSTM\text{-}C$ to converge in the dataset of top-10 worldwide cities (top-100 US cities). In contrast, the LR used 3 and 5 hours to converge in the two datasets, respectively. Besides, obtaining word vectors cost approximately another 20 hours for $BiLSTM\text{-}C$. For $IG\text{-}Bayes$, it took less than quarter an hour to calculate the information gains and build indexes. On the other hand, $BiLSTM\text{-}C$ needs large space to store the word vectors; it is more sensitive to larger datasets compared with $IG\text{-}Bayes$. Both models can infer the location of a cluster in less than 10 ms on average in our experiments. Besides, the average computation time of temporal clustering is less than 1 ms for both training and testing datasets. This indicates that our models can work in online scenarios.

Taking all these issues discussed above into consideration, it is suggested that one choose $IG\text{-}Bayes$ to infer tweet locations if there is no sufficient data or space to use or if the overall time (from training to inference) is a sensitive factor. Otherwise, the $BiLSTM\text{-}C$ model can be trained to solve the problem with considerably higher inference accuracy if sufficient time and resources are available.

8 Conclusion and Future Work

In this paper, we propose a novel approach to infer city-level locations for tweets without any geo-tags. Our approach first employs a temporal clustering method to split each Twitter user's timeline into a set of clusters. Each of these clusters contains tweets that are likely sent from the same location within a short period of time. Subsequently, our approach adapts two probabilistic models to infer locations for tweet clusters. The Information Gain Bayes model ($IG\text{-}Bayes$) exploits the information gain of words with location implications in the user-generated contents. The bidirectional LSTM convolutional model ($BiLSTM\text{-}C$) treats user-generated contents and their associated locations as sequences and augments a bidirectional LSTM with convolution operation to make better location inferences. We conduct extensive experiments using large real datasets collected from Twitter. The experimental results demonstrate that $IG\text{-}Bayes$ and $BiLSTM\text{-}C$ achieve high location inference accuracy in multiple settings and clearly outperform the state-of-the-art and alternative approaches.

The proposed models in this paper use tweet contents only. For future work, it is interesting to consider other information such as social relationship among users and frequent patterns shared by users. When combined with tweet contents, such information may be utilized to make even better location inferences. Also, it is possible to make explicit use of the few geo-tagged tweets in a user's timeline, e.g., through spatio-temporal constraints, in the hope of improving or easing location inference for non-geotagged tweets.

References

[1] S. Yardi and D. Boyd, "Tweeting from the town square: Measuring geographic local networks," in *ICWSM*, 2010.

[2] H. Yin, W. Wang, H. Wang, L. Chen, and X. Zhou, "Spatial-aware hierarchical collaborative deep learning for POI recommendation," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 11, pp. 2537–2551, 2017.

[3] M. Sarwat, J. J. Levandoski, A. Eldawy, and M. F. Mokbel, "Lars*: An efficient and scalable location-aware recommender system," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 6, pp. 1384–1399, 2014.

[4] J. Li, M. L. Yiu, and N. Mamoulis, "Efficient notification of meeting points for moving groups via independent safe regions," in *ICDE*, 2013.

[5] J. Jiang, H. Lu, B. Yang, and B. Cui, "Finding top-k local users in geo-tagged social media data," in *ICDE*, 2015.

[6] J. Li, T. Sellis, J. S. Culpepper, Z. He, C. Liu, and J. Wang, "Geo-social influence spanning maximization," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 8, pp. 1653–1666, 2017.

[7] X. Wang, Y. Zhang, W. Zhang, and X. Lin, "Efficient distance-aware influence maximization in geo-social networks," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 3, pp. 599–612, 2017.

[8] J. Fiorillo, "Twitter advertising: Pay-per-tweet," *URL: http://www.wikinomics.com/blog/index.php/2009/04/22/twitter-advertising-pay-per-tweet/, Access at*, vol. 4, p. 12, 2010.

[9] M. J. Culnan, P. J. McHugh, and J. I. Zubillaga, "How large us companies can use twitter and other social media to gain business value." *MIS Quarterly Executive*, vol. 9, no. 4, 2010.

[10] C. Zhang, L. Liu, D. Lei, Q. Yuan, H. Zhuang, T. Hanratty, and J. Han, "Triovecevent: Embedding-based online local event detection in geo-tagged tweet streams," in *KDD*, 2017.

[11] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes twitter users: real-time event detection by social sensors," in *WWW*, 2010.

[12] Z. Cheng, J. Caverlee, and K. Lee, "You are where you tweet: A content-based approach to geo-locating twitter users," in *CIKM*, 2010.

[13] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *ICASSP*, 2013.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[17] C. A. Davis Jr, G. L. Pappa, D. R. R. de Oliveira, and F. de L Arcanjo, "Inferring the location of twitter messages based on user relationships," *Transactions in GIS*, vol. 15, no. 6, pp. 735–751, 2011.

[18] D. Jurgens, "That's what friends are for: Inferring location in online social media platforms based on social relationships." *ICWSM*, vol. 13, no. 13, pp. 273–282, 2013.

[19] D. Rout, K. Bontcheva, D. Preoţiuc-Pietro, and T. Cohn, "Where's@ wally?: a classification approach to geolocating users based on their social ties," in *HT*, 2013.

[20] L. Backstrom, E. Sun, and C. Marlow, "Find me if you can: improving geographical prediction with social and spatial proximity," in *WWW*, 2010.

[21] J. McGee, J. Caverlee, and Z. Cheng, "Location prediction in social media based on tie strength," in *CIKM*, 2013.

[22] E. Rodrigues, R. Assunção, G. L. Pappa, R. Miranda, and W. Meira, "Uncovering the location of twitter users," in *BRACIS*, 2013.

[23] R. Li, S. Wang, H. Deng, R. Wang, and K. C.-C. Chang, "Towards social user profiling: unified and discriminative influence model for inferring home locations," in *SIGKDD*, 2012.

[24] D. Jurgens, T. Finethy, J. McCorriston, Y. T. Xu, and D. Ruths, "Geolocation prediction in twitter using social networks: A critical analysis and review of current practice." *ICWSM*, vol. 15, pp. 188–197, 2015.

[25] B. P. Wing and J. Baldridge, "Simple supervised document geolocation with geodesic grids," in *ACL*, 2011.

[26] S. Roller, M. Speriosu, S. Rallapalli, B. Wing, and J. Baldridge, "Supervised text-based geolocation using language models on an adaptive grid," in *EMNLP*, 2012.

[27] J. Eisenstein, B. O'Connor, N. A. Smith, and E. P. Xing, "A latent variable model for geographic lexical variation," in *EMNLP*, 2010.

[28] J. Eisenstein, A. Ahmed, and E. P. Xing, "Sparse additive generative models of text," 2011.

[29] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis, "Discovering geographical topics in the twitter stream," in *WWW*, 2012.

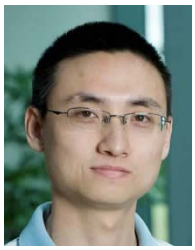
[30] Y. Chen, J. Zhao, X. Hu, X. Zhang, Z. Li, and T.-S. Chua, "From interest to function: Location estimation in social media." in *AAAI*, 2013.

[31] K. Ryoo and S. Moon, "Inferring twitter user locations with 10 km accuracy," in *WWW*, 2014.

[32] B. Hecht, L. Hong, B. Suh, and E. H. Chi, "Tweets from justin bieber's heart: the dynamics of the location field in user profiles," in *CHI*, 2011.

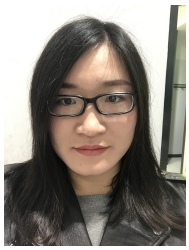
[33] Y. Qian, J. Tang, Z. Yang, B. Huang, W. Wei, and K. M. Carley, "A probabilistic framework for location inference from social media," *arXiv:1702.07281*, 2017.
[34] H.-w. Chang, D. Lee, M. Eltaher, and J. Lee, "@ phillies tweeting from philly? predicting twitter user locations with spatial word usage," in *ASONAM*, 2012.
[35] Z. Liu and Y. Huang, "Where are you tweeting?: A context and user movement based approach," in *CIKM*, 2016.
[36] H. Bo, P. Cook, and T. Baldwin, "Geolocation prediction in social media data by finding location indicative words," in *COLING*, 2012.
[37] B. Han, P. Cook, and T. Baldwin, "Text-based twitter user geolocation prediction," *JAIR*, vol. 49, pp. 451–500, 2014.
[38] W. Huang, I. Weber, and S. Vieweg, "Inferring nationalities of twitter users and studying inter-national linking," in *HT*, 2014.
[39] J. Mahmud, J. Nichols, and C. Drews, "Where is this tweet from? inferring home locations of twitter users." *ICWSM*, 2012.
[40] ——, "Home location identification of twitter users," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 5, no. 3, p. 47, 2014.
[41] Y. Ikawa, M. Enoki, and M. Tatsubori, "Location inference using microblog messages," in *WWW*, 2012.
[42] R. Krishnamurthy, P. Kapanipathi, A. P. Sheth, and K. Thirunarayan, "Knowledge enabled approach to predict the location of twitter users," in *ESWC*, 2015.
[43] Y. Yamaguchi, T. Amagasa, H. Kitagawa, and Y. Ikawa, "Online user location inference exploiting spatiotemporal correlations in social streams," in *CIKM*, 2014.
[44] S. Kinsella, V. Murdock, and N. O'Hare, "I'm eating a sandwich in glasgow: modeling locations with tweets," in *SMUC*, 2011.
[45] D. Doran, S. Gokhale, and A. Dagnino, "Accurate local estimation of geo-coordinates for social media posts," *arXiv:1410.4616*, 2014.
[46] R. Priedhorsky, A. Culotta, and S. Y. Del Valle, "Inferring the origin locations of tweets with quantitative confidence," in *CSCW*, 2014.
[47] A. Zubiaga, A. Voss, R. Procter, M. Liakata, B. Wang, and A. Tsakalidis, "Towards real-time, country-level location classification of worldwide tweets," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 9, pp. 2053–2066, 2017.
[48] T. Palpanas and P. Paraskevopoulos, "Fine-grained geolocalisation of non-geotagged tweets," in *ASONAM*, 2015.
[49] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann, "Who, where, when and what: discover spatio-temporal topics for twitter users," in *SIGKDD*, 2013.
[50] M. Dredze, M. Osborne, and P. Kambadur, "Geolocation for twitter: Timing matters." in *HLT-NAACL*, 2016.
[51] P. Paraskevopoulos and T. Palpanas, "Where has this tweet come from? fast and fine-grained geolocalization of non-geotagged tweets," *Social Network Analysis and Mining*, vol. 6, no. 1, p. 89, 2016.
[52] F. Song and W. B. Croft, "A general language model for information retrieval," in *CIKM*, 1999.
[53] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *SIGIR*, 1998.
[54] C. Zhai and J. Lafferty, "A study of smoothing methods for language models applied to ad hoc information retrieval," in *SIGIR*, 2001.
[55] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.
[56] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
[57] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[58] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv:1409.2329*, 2014.
[59] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013.
[60] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *NIPS*, 2015, pp. 802–810.
[61] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning." *ICML (3)*, vol. 28, pp. 1139–1147, 2013.
[62] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[63] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop, coursera: Neural networks for machine learning," *University of Toronto, Tech. Rep*, 2012.
[64] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

Pengfei Li is a PhD student in computer science with Zhejiang University, China. He received the BSc degree in computer science from Zhejiang University, in 2016. His research interests include machine learning and data mining.



Hua Lu is an associate professor in the Department of Computer Science, Aalborg University, Denmark. He received the BSc and MSc degrees from Peking University, China, and the PhD degree in computer science from National University of Singapore. His research interests include database and data management, geographic information systems, and mobile computing. He has served as PC cochair or vice chair for ISA 2011, MUE 2011 and MDM 2012, demo chair for SSDBM 2014, and PhD forum cochair for MDM 2016. He has served on the program committees for conferences such as ICDE, CIKM, DASFAA, ACM SIGSPATIAL, SSTD, MDM, PAKDD, APWeb, and WAIM. He is a senior member of the IEEE.

Nattiya Kanhabua is a senior data scientist at NTENT, Barcelona, Spain. Previously, she was an assistant professor at the Department of Computer Science, Aalborg University, Denmark. In addition, she was a postdoctoral researcher at the L3S Research Center. Her research interests are information retrieval, Web and social media mining, and Web archiving. In particular, she focus on the impacts of Web dynamics on search, e.g., terminology evolution, time-sensitive queries, and time-aware retrieval and ranking. She has served on the program committees for conferences such as WWW, CIKM, WSDM, SIGIR, JCDL, TPDL, and ECIR.



Sha Zhao is currently a Postdoctoral Research Fellow of the College of Computer Science and Technology, Zhejiang University. She received the Best Paper Award of ACM UbiComp16. Zhao visited the Human-Computer Interaction Institute at Carnegie Mellon University as a visiting PhD student from September 2015 to September 2016. She received the Ph.D. degree from Zhejiang University, Hangzhou, China, in 2017. Her research interests include pervasive computing, mobile sensing, data mining, and machine learning. For more information about her, please visit at http://www.shazhao.net.

Gang Pan received B.Sc. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently a Professor with the College of Computer Science and Technology, Zhejiang University. He was with the University of California, Los Angeles, Los Angeles, CA, USA, as a Visiting Scholar, from 2007 to 2008. He has published more than 100 refereed papers. His research interests include pervasive computing, computer vision, and pattern recognition.