Progress in Artificial Intelligence (2020) 9:195–207
https://doi.org/10.1007/s13748-020-00206-2

REGULAR PAPER

GDTM: Graph-based Dynamic Topic Models

Kambiz Ghoorchian[1] · **Magnus Sahlgren**[2]

Received: 21 August 2019 / Accepted: 9 April 2020 / Published online: 15 May 2020

Abstract
Dynamic Topic Modeling (DTM) is the ultimate solution for extracting topics from short texts generated in Online Social Networks (OSNs) like Twitter. It requires to be scalable and to be able to account for sparsity and dynamicity of short texts. Current solutions combine probabilistic mixture models like Dirichlet Multinomial or Pitman-Yor Process with approximate inference approaches like Gibbs Sampling and Stochastic Variational Inference to, respectively, account for dynamicity and scalability of DTM. However, these methods basically rely on weak probabilistic language models, which do not account for sparsity in short texts. In addition, their inference is based on iterative optimizations, which have scalability issues when it comes to DTM. We present GDTM, a single-pass graph-based DTM algorithm, to solve the problem. GDTM combines a context-rich and incremental feature representation method with graph partitioning to address scalability and dynamicity and uses a rich language model to account for sparsity. We run multiple experiments over a large-scale Twitter dataset to analyze the accuracy and scalability of GDTM and compare the results with four state-of-the-art models. In result, GDTM outperforms the best model by 11% on accuracy and performs by an order of magnitude faster while creating four times better topic quality over standard evaluation metrics.

Keywords Topic modeling · Dimensionality reduction · Distributional semantics · Language modeling · Graph partitioning

1 Introduction

Motivation topic modeling [1] is the problem of automatic classification of words, which form the context of documents, into similarity groups, known as topics. More specifically, it is a dimensionality reduction problem [2] where the goal is to reduce the high-dimensional space of words into a significantly low-dimensional and semantically rich space of topics. Traditional solutions considered topic modeling as a batch processing problem where a fixed number of documents were read into a system and iteratively analyzed, following an optimization function. However, documents generated in today's social media (like Twitter or Facebook) are (i) fast (large scale and continuous), (ii) sparse (short length) and (iii) dynamic (with constant emergent of newly generated phrases or context structures). This is a problem known as *Dynamic Topic Modeling (DTM)*. A legitimate solution to DTM should constantly receive a large number of short texts, extract their topics and adapt to the changes in the topics.

Current solutions Latent Dirichlet Allocation (LDA) [3] is one of the most well-known solutions to topic modeling that proposes a probabilistic modeling of the feature space and employs a deterministic inference approach, known as *Expectation Maximization (EM)*, to extract the topics. LDA is an online topic modeling approach by nature. However, for three reasons, it cannot be considered as a solution for DTM: first, the iterative structure of the EM algorithm that limits the scalability of LDA; second, the naive probabilistic model and the strong assumption on having a fixed number of topics that limit the adaptation of the model to the dynamicity in DTM; and third, the simplistic language model, known as *Bag Of Words (BOW)*, that does not account for sparsity in short texts.

To address sparsity, BTM [4] creates stronger context representations using a more complex language model, called bigram, and to cope with dynamicity, DTM [5], CDTM [6] and DCT [7] use stronger probabilistic modeling approaches by interconnecting the transition probabilities. However, all these solutions still use a fixed number of partitions that

✉ Kambiz Ghoorchian
ghoorian@kth.se

Magnus Sahlgren
magnus.sahlgren@ri.se

[1] Electrical Engineering and Computer Science (EECS) Royal Institute of Technology (KTH), Stockholm, Sweden

[2] Research Institute of Sweden (RISE), Stockholm, Sweden

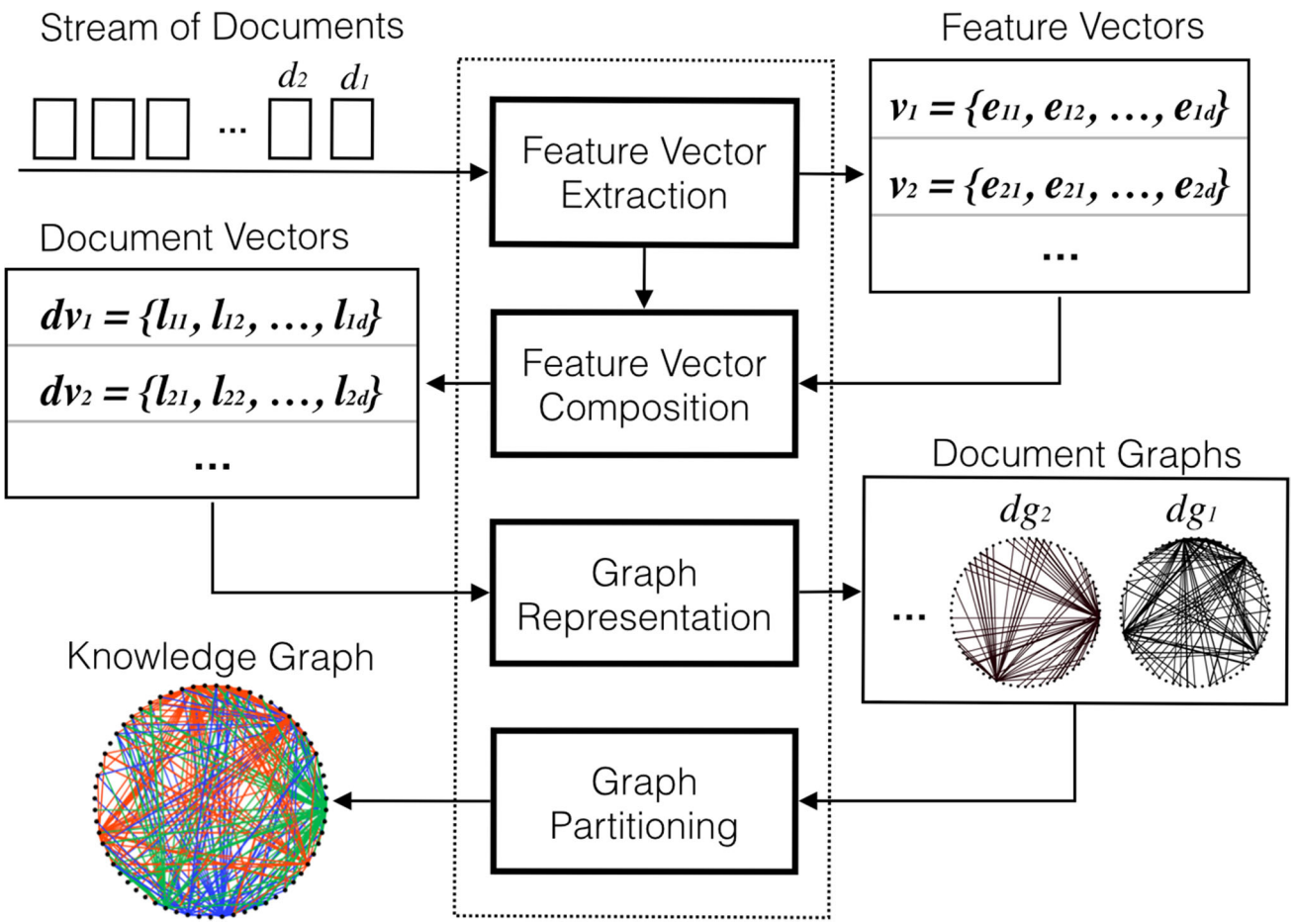

Fig. 1 The protocol of GDTM, a graph-based algorithm for dynamic topic modeling. A stream of documents pass through a set of four components to extract and dynamically maintain their topics

limits their power to account for dynamicity. GSDMM [8], FGSDMM [9] and PYPM [10] propose to solve this problem using stochastic optimization approaches like *Dirichlet Multinomial Mixture (DMM)* [11] models or *Pitman-Yor* [10] processes to consider infinite number of topics and allow the algorithm to dynamically adapt the number of partitions. In addition, all these approaches strive to achieve scalability by reducing the sample size using approximate optimization algorithms such as *Gibbs Sampling* [12] and *Stochastic variational inference* [13]. However, they are still relying on the same iterative optimization mechanisms and therefore sensitive to scalability issues when it comes to DTM.

Our approach This paper presents *GDTM*, a *Graph-based Dynamic Topic Modeling* algorithm designed to overcome those limitations by taking all the above-mentioned aspects into consideration. The solution combines a dimensionality reduction technique, called *Random Indexing (RI)* [14], to overcome the scalability, an advanced language modeling approach based on the *Skip-Gram* [15] technique, used in natural language modeling and speech recognition, to address the sparsity and an innovative graph modeling together with a single-pass graph partitioning algorithm to account for dynamicity. Figure 1 shows the overall protocol of the algorithm that is constructed in a pipeline approach.

A stream of documents pass through the pipeline of four components where each document gets processed by each component until the topic is assigned to it. First, the *Feature Vector Extraction* component reads and tokenizes the document and extracts a vector representation for each word in the document using RI. Then, the *Feature Vector Composition* combines the corresponding feature vectors to construct the document representation vector using the skip-gram model. After that, the *Graph Representation* component converts each document vector into a graph representation called document graph. Finally, the *Graph Partitioning* component extracts the topics by aggregating the document graphs into a single, weighted and highly dense graph representation called *Knowledge Graph (KG)*. The algorithm uses the KG for two reasons: first, to assign topics to new documents based on the overlap between their corresponding document graph and the

KG and second, to maintain the dynamics of the topics following a deterministic optimization function.

Main contribution The key element of success in our algorithm is to distinguish between the two main components, namely (i) feature representation and (ii) topic extraction. This allows us to develop a single-pass algorithm where each document only passes once through the entire process. Moreover, the two main characteristics that play a significant role in this scenario are (i) the incremental nature of the RI technique that allows us to extract semantically rich and highly low-dimensional feature representation vectors without the need to access the entire dataset and (ii) the single-pass streaming graph partitioning that enables the extraction of high-quality topics encoded in the graph representation using the rich language representation model.

Summary of experiments and results We run two sets of experiments to analyze the (i) *accuracy* and (ii) *scalability* of GDTM. To show the accuracy, we define a topic modeling task on a tagged Twitter dataset and compare GDTM with four state-of-the-art approaches on performing the task using a standard evaluation metric, called *B-Cubed* [16]. For scalability, we run a set of experiments on a large-scale Twitter dataset and compare the execution time and the quality of the extracted topics, using the evaluation method called *Coherency* [17]. The results show that GDTM outperforms all the state-of-the-art approaches in both accuracy and scalability. In particular, GDTM provides more than 11% improvement on accuracy compared to the best results over the state-of-the-art approaches. In addition, we show that GDTM is by an order of magnitude faster than the best approach over scalability, while the extracted partitions exhibit significantly higher quality in terms of coherency.

2 Related work

Classical solutions for topic modeling on text, such as PLSI [18] and LDA [3], proposed to model the co-occurrence patterns as a probability distribution over a batch of long documents and infer the topics using statistical techniques such as variational inference and Gibbs Sampling [12]. However, with the emergence of the online social networks and the appearance of short texts, like tweets, these solutions faced various challenges related to the size, the number and the dynamics of the documents in such new environments.

Yan et al. [4] and ghoorchian et al. [26] presented a method to solve the sparsity in short texts by applying a more complex language model, known as bigram. The authors used the bigram model to overcome the sparsity by constructing richer context representations from short texts. However, they did not consider the dynamicity as their model still requires to know the number of topics in advanced and therefore lacks flexibility when it comes to fast dynamic changes in the documents.

Blei et al. [5] proposed the first solution specifically designed to alleviate the dynamicity in DTM. They developed a family of probabilistic time-series models to analyze and extract the evolution of topics in a stream of documents. The authors tried to solve the dynamicity by discretizing the stream of documents and developing a stream of batches by interrelating the consequent models through variational approximation methods based on *Kalman Filters* [19]. Their model was limited in scalability when the discretization of the topics went to infinity. Wang et al. [6] proposed another solution, called Continuous-time Dynamic Topic Model (CDTM), to overcome the discretization problem in DTM using a continuous generalization approach. DTM and CDTM are basically designed for topic modeling on large documents and do not account for sparsity and scalability in short texts.

Liang et al. [7] proposed another solution based on short-term and long-term inter-dependency between the mean of the distributions across multiple time stamps to solve the discretization problem and also account for the sparsity in short texts. However, their model, similar to DTM and CDTM, requires to know the number of topics, which limits its power to account for the dynamicity. In addition, their inference approach is based on the same iterative Gibbs Sampling optimization mechanism that limits the scalability.

To solve the problem of the fixed number of topics, Yin et al. proposed solutions, GSDMM [8] and FGSDMM [9], based on Dirichlet Multinomial Mixture (DMM) Processes. Qiang et al. [10] improved Yin's solution using a new clustering with probabilities derived from a Pitman-Yor Mixture Process [20]. These approaches have significantly improved the accuracy of the extracted topics. However, they are basically designed for application on batch processing problems and therefore, face scalability issues when it comes to DTM.

Multiple solutions are developed to overcome different challenges in DTM but to our knowledge, a single approach that can tackle all the challenges at once is missing. Thus, we present GDTM as a universal model that is designed to meet all the challenges in DTM.

3 Solution

In this section, we will explain the details of our single-pass graphs-based dynamic topic modeling algorithm. The algorithm is designed using a pipeline approach that receives a stream of documents. The documents pass through four components: Feature Vector Extraction, Feature Vector Composition, Graph Representation and Graph Partitioning. In the following sections, we will explain each of these compo-

nents and the way they interact with each other to extract the topics.

3.1 Feature vector extraction

We consider words as the atomic features and use a vector representation model to construct the feature vectors as the building blocks of the document representation model. GDTM requires a representation model that (i) is low-dimensional to account for scalability, (ii) is incremental to be useful in streaming setting, and (iii) creates relatively rich representations that contributes to efficiency in a single-pass optimization approach. RI is a reliable [21] dimensionality reduction technique that perfectly satisfies all the above requirements.

RI follows the famous statement *"you shall know a word by the company it keeps"* [22] based on distributional semantics [23]. The algorithm iterates through the document and constructs a low-dimensional vector representation for each word as follows. First, for each new word, RI creates a new vector WV of a fixed dimension d and randomly initializes an arbitrary number of its elements ζ to 1 and the rest $d - \zeta$ to 0. Then, the algorithm updates the WV of each word by looking into a window of an arbitrary size ω around the corresponding word and aggregating their corresponding WVs. The dimension of the vectors d is fixed and is significantly lower than the original feature space n (e.g., the total number of words) $d << n$. To avoid redundancy, we maintain a list of previously seen words together with their corresponding feature vectors and update the feature vectors only upon the observation of new context structures. This mechanism allows each feature vector to contain a rich representation of the context structure around the corresponding word without any clue on the significance of those structures. This is the requirement that the algorithm will address in graph partitioning component.

Neural Language Models [15], are another group of vector representation models that create low-dimensional and rich feature vector representations. However, they use classification based on iterative back-propagation algorithm, which does not suit the dynamic ecosystem of the GDTM.

3.2 Feature vector composition

The next step is to compose the extracted feature vectors to construct a document representation vector. A valid composition method should satisfy two properties: (i) preserving the complexity of the original feature vectors without losing any information (ii) accounting for sparsity in the documents.

Mitchell et al. [24] proposed a variety of vector composition methods such as *pairwise multiplication*, *pairwise addition*, *weighted pairwise multiplication*, etc., that satisfy the lossless property. However, these simple composition methods do not account for sparsity. For example, pairwise addition is similar to the BOW [3] approach used in LDA method that does not address the sparsity. Therefore, a more complex composition method is required. The choice of the composition depends on the language model used in the analysis. We use a well-known technique called *Skip-gram* [25] for this purpose. Skip-gram drives the probability of a feature given the history of its surrounding context. (This provides a more complex model compared to its descendant model called N-gram, which only considers the history of the previous context). More specifically, we use a $m-skip-bigram$ model where m is a parameter to be specified by the user. This model drives the context structure of a word w in a given context by looking at the bigrams with $M = [0, m]$ step(s) before and after the w. Let us explain the composition model and the weighting mechanism with an example.

Table 1 List of skip-grams corresponding to each word using a m-skip-bigram model with $m = 1$

w_1	$\{w_1w_2, w_1w_3\}$
w_2	$\{w_2w_1, w_2w_3, w_2w_4\}$
w_3	$\{w_3w_1, w_3w_2, w_3w_4\}$
w_4	$\{w_4w_2, w_4w_3\}$

Assume we are given a document D containing four consequent words $W_D = \{w_1, w_2, w_3, w_4\}$ and we set $m = 1$. To construct the document vector, we iterate through the document and for each word first, we extract the set of 1-skip-bigrams that contains all the bigrams with skip value between 0 and 1. For example, w_2 has three bigrams including two 0-skip-bigrams w_2w_1, w_2w_3 and one 1-skip-bigram w_2w_4. Table 1 shows the list of all 1-skip-bigrams extracted for all the words in D. Afterward, for each bigram w_iw_j, we create a bigram vector by *weighted pairwise multiplication* of its corresponding feature vectors $v_i = \{e_{i1}, \ldots, e_{id}\}$ and $v_j = \{e_{j1}, \ldots, e_{jd}\}$, constructed in the previous step:

$$BV_{w_iw_j} = \alpha_i \times \alpha_j \times \{e_{i1} \times e_{j1}, \ldots, e_{id} \times e_{jd}\} \quad (1)$$

α_i and α_j are the weights, respectively, related to the words w_i and w_j, which are calculated using a Sigmoid function as follows:

$$\alpha_l = \frac{1}{1 + \exp\left(\delta \times \frac{|w_l|}{\sum_{\forall l} |w_l|}\right)} \quad (2)$$

The weight α_l is inversely proportional to the frequency of the corresponding word w_l and is used to reduce the negative effect of highly frequent words in the dataset. We use an adjustment parameter δ to indicate the significance of the ratio and a threshold parameter γ that indicates the words to remove from the document representation. In particular, if $\alpha_l < \gamma$, then we set $\alpha_l = 0$ that eliminate the bigram vector from the construction of the document vector.

The final step is to combine all valid bigram vectors to construct the corresponding document vector. We use a *normalized pairwise addition* as the composition method in this step. In the above example, given that none of the weights are zero and all bigrams are valid, we will have ten bigram vectors corresponding to the skip-grams presented in Table 1. Now assuming that each bigram vector is a vector containing d elements $bv_i = \{l_{i1}, \ldots, l_{id}\}$, then the document vector is created as follows:

$$DV = \frac{1}{10} \times \left\{ \sum_i l_{i1}, \ldots, \sum_i l_{id} \right\} \tag{3}$$

3.3 Graph construction

In previous steps, the algorithm encoded topics as unique structures in the form of document vector representations. The goal, in this step, is to project those structures into a graph representation model to be used for extracting topics by graph partitioning component. A graph $G < V, E >$ is a set of vertices V connected with a set of edges E. Graphs provide a simple representation model to present a set of concepts and the relation between those concepts (e.g., users and their friendship relation in Facebook). GDTM considers elements in document vectors as the concepts and extracts the patterns as the relation between those elements. Let us explain it with an example.

Assume we are given a document together with a d-dimensional document vector $DV = \{l_1, \ldots, l_d\}$. The corresponding graph representation of the document DG is defined as a set of d vertices $V = \{v_1, \ldots, v_d\}$, which are connected using a set of $\frac{d \times (d-1)}{2}$ edges $E = \{e_{0,1}, \ldots, e_{d-1,d}\}$. Each vertex v_i corresponds to one element l_i in the DV, and each edge e_{ij} represents the relation between its incident vertices v_i and v_j in the graph representation. The edges are weighted, and the weight w_{ij} of a given edge e_{ij} is calculated as the multiplication of the values of the corresponding elements in the DV, $w_{ij} = l_i \times l_j$. The construction methods suggests that the created graph is a mesh. However, this is not the case since the document vectors are often highly sparse with most of their elements being zero. Therefore, the created document graph will also be sparse.

After converting each DV into a DG, which is representative of the topical structure of the corresponding documents in the stream, the next step is to combine the DGs and extract the topics using graph partitioning.

3.4 Graph partitioning

Let us first present a set of definitions required for understanding the mechanism of the graph partitioning algorithm, before explaining the details:

Knowledge graph (KG): is a graph with the same number of vertices d and the same number of edges $\frac{d \times (d-1)}{2}$ as DGs. GDTM uses KG as the universal model in the algorithm to aggregate the DGs, keep track of the topics and assign topics to the documents.

Partition given a graph $G < V, E >$, a partition is defined as a sub-graph $G' < V', E' >$ of G such that $V' \subseteq V$ and $E' \subseteq E$.

Density 4: is a metric to measure the degree of connectedness of the nodes in a graph. We define the density d of a given graph $G < V, E >$ as the average weight over the total possible edges in the graph:

$$d(G) = \frac{1}{\frac{1}{2}|V|^2} \sum_{e \in E} w_e. \tag{4}$$

Therefore, the higher the weights of the edges, the higher the density will be. Consequently, a nonexisting edge makes zero contribution to the density.

Average density 5: is a measure to show the average total density over a graph and is calculated as the average of the densities of all partitions in that graph. Given a graph $G(V, E)$ and a set of n partitions $P = \{p_1, \ldots, p_n\}$, the average density is calculated as follows:

$$ad(G) = \frac{1}{n} \sum_{i=0}^{n} d(P_i). \tag{5}$$

Now, let us move forward to explain the graph partitioning algorithm. The main assumption is that each DG is representative of the unique topical structure of its corresponding document. Thus, the goal is to aggregate the DGs into a single graph representation, called KG and extract the topics by partitioning the KG following an optimization mechanism. GDTM is an online approach that requires to apply partitioning upon receiving every single document. Thus, the algorithm is designed in two steps (i) topic assignment and (ii) optimization. The first step assigns a topic to each document by comparing its DG with the KG. The second step aggregates the corresponding DG into the KG and applies an optimization mechanism such that the partitioning of the KG gets continuously updated over aggregating every single edge from the DG. The next sections will explain the details of these two steps and how they interact with each other to extract the topics.

3.4.1 Topic assignment

Before aggregating each document into the KG, we need to know the topic of the document in order to apply the correct optimization. The basic idea is to extract the distribution of the topics over that document and choose the

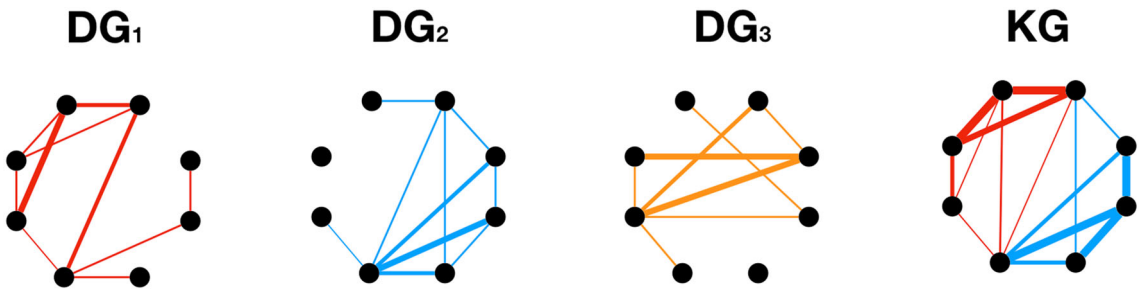

Fig. 2 Example topic assignments before aggregating the document graphs (e.g., DG_i) into the knowledge graph (KG). The figure shows three sample document graphs DG_1, DG_2 and DG_3 which are assigned different topics based on their overlap with the KG

topic with the largest probability as the representative topic of the document. Assume a partitioned KG with a set of edges $E = \{e_1, \ldots, e_n\}$ and a DG containing its own set of edges $E' = \{e'_1, \ldots, e'_m\}$. Also, assume that every edge e_i in the KG has a weight w_i and a partition p_i assigned to it. To extract the topic distribution, first we create a list containing all the topics and their weights initialized to zero. Then, we iterate through all the edges in the DG and for each edge e'_i GDTM finds the related edge e_i in the KG and adds its weight w_i to the corresponding partition p_i in the list. Finally, we select the topic with the highest probability as the representative topic of the document. Figure 2 shows an example of this operation. As we can see, DG_1 and DG_2 show examples of documents that are assigned to different topics, *Red* and *Blue*, based on their highest overlap with the KG.

The only challenge in this step happens when one or more edges in the DG have no overlapping edges in the KG and therefore cannot be assigned a topic. This condition occurs when the document under operation belongs to a new topic other than those currently presented in the KG (e.g., note the *Orange* topic on DG_3 in Fig. 2 that does not exist in the KG before aggregating DG_3). In this situation, GDTM creates a new topic and assigns it to the corresponding edge(s) in the DG. The new topic will then be added to the KG upon aggregating the corresponding DG. This is one of the key advantages of the GDTM that enables the model to account for an infinite number of partitions, in contrast to the approaches with fixed partition count. Following the same argument, it is important to note that the first document in the stream will always be assigned a new topic as the KG is initially null and there are no topics to be assigned. After assigning a topic to the DG, the next step is to aggregate the DG with the KG and update the KG following an optimization mechanism.

3.4.2 Optimization

Optimization is an online process to extract high-quality topics encoded as dense weighted partitions in the KG. We consider the quality of partitioning in terms of average density. More specifically, the higher the average density, the better the partitioning. Thus, the goal, in this step, is to define an optimization problem to maximize the average density of the partitioning over the KG and develop an accurate algorithm to solve it. Next comes a formal definition of the problem followed by the detailed explanation of the algorithm.

Problem definition Given a partitioned KG and a DG with a dominant partition assigned to it, how can we aggregate the DG to the KG and update the partitioning of the KG such that the average density of the partitioning is maximized.

Solution GDTM develops a local deterministic optimization algorithm to solve this problem. The algorithm establishes and applies a set of policies upon aggregating each DG to the KG. The policies ensure maximization of the local density of the partitions, which in turn guarantee the monotonic optimization of the global average density. Let us present and prove the basic proposition that ensures the monotonically increasing behavior of the algorithm before explaining the conditions and their corresponding policies.

Proposition 1 *Assume a set of real numbers* $R = \{r_1, \ldots, r_n\}$ *with mean* μ. *Consider the set* $R \backslash \{r_j\} = \{r_1, \ldots, r_{j-1}, r_{j+1}, \ldots, r_n\}$ *with mean denoted by* $\mu_n(j)$. *We have that* $\mu_n(j) \geq \mu$, *for all* j *such that* $r_j \leq \mu$.

Proof We have

$$\mu = \frac{1}{n} \sum_{i=1}^{n} r_i \tag{6}$$

$$\mu_n(j) = \frac{1}{n-1} \left[\sum_{i=1}^{j-1} r_i + \sum_{i=j+1}^{n} r_i \right] \tag{7}$$

We start by simple rearrangements:

$$r_j \leq \mu$$
$$= \frac{1}{n} \sum_{i=1}^{n} r_i$$

$$= \frac{1}{n} r_j + \frac{1}{n}(r_1 + \cdots + r_{j-1} + r_{j+1} + \cdots + r_n) \quad (8)$$

Deducting $\frac{1}{n} r_j$ from both sides and using (7) yields:

$$\frac{n-1}{n} r_j \leq \frac{n-1}{n} \mu_n(j)$$

Adding $\frac{1}{n}\mu_n(j)$ to both sides results in:

$$\frac{n-1}{n} r_j + \frac{1}{n}\mu_n(j) \leq \frac{n-1}{n}\mu_n(j) + \frac{1}{n}\mu_n(j)$$
$$\mu \leq \mu_n(j).$$

The two main intuitions behind the above preposition, relative to our definitions of **partition** and **density**, are as follows. Given a partition p with the density μ, (i) removing an edge e with the weight $w_e \leq \mu$ will not decrease the density, (ii) adding an edge e with the weight $w_e \geq \mu$ always increases the density with a positive value. Now let us present the details of the algorithm and the way it applies the above intuitions in the aggregation process to appeal optimization.

Given a partitioned KG and a DG with a dominant partition assigned to it, the algorithm iterates through all the edges in the DG and for each edge e' with the corresponding weight w' and dominant partition p', it applies an optimization upon aggregating e' with the matching edge e having the weight w and the partition p in the KG. Different conditions can happen depending on the type of p' and the weights of the edges w and w'. GDTM requires to apply an appropriate policy upon aggregation in each condition in order to ensure the optimization requirements. Two types of partitions can be assigned to a DG, as explained in Sect. 3.4.1. First, a *New Partition (NP)* when majority of edges in DG do not match any edge in the KG, and Second, an *Old Partition (OP)* that currently exists in the KG and the edges in the DG has the highest overlap with edges of this partition in the KG. Also, there are three different conditions depending on the current status of the edges and partitions in the KG and the DG (i) $e = \phi$ meaning that the edge does not exist in the KG (ii) $e \neq \phi$ and $p \neq p'$, meaning that the edge e exists in the KG but it has a different partition than e' and (iii) $e \neq \phi$ and $p = p'$ indicating that e exists and belongs to the same partition as e'. Table 2 shows a summary of all conditions labeled as $\{c_1, \ldots, c_5\}$. Note that the condition $e \neq \phi$ and $p = p'$ is not a valid condition when p' is new (NP), which is clear by definition.

Next, we will present different conditions and explain the appropriate policy applied in each condition. Algorithm 1 shows the overall process of the optimization mechanism and the corresponding policy applied depending on the condition. Each condition is numbered according to the numbers in Table 2. We use e, w and p to refer to the elements in the KG and the e', w' and p' for elements in the DG. In addition, we use a function called *density(p)* that is used to retrieve the density of a given partition p. For performance reasons, GDTM creates and maintains a key-value storage to retrieve the partition densities in the KG.

Table 2 Possible conditions that can happen during the aggregation process of an edge from a Document Graph (DG) to the corresponding edge in the Knowledge Graph (KG)

	New partition (NP)	Old partition (OP)
$e = \phi$	C1	C3
$p \neq p'$	C2	C4
$p = p'$	NA	C5

Algorithm 1 Optimization

```
KG ← Knowledge Graph
lp ← last partition
density[P, V]
procedure AGGREGATE(DG, P_DG)
    for all e ∈ DG do
        if p' > lp then                                   ▷ New Partition (NP)
            if p = φ then                                 ▷ c1
                p = p'
                w = w'
            if p ≠ p' then                                ▷ c2
                if w ≤ density(p) then
                    p = p'
                    w = w + w'
        if p' ≤ lp then                                   ▷ Old Partition (OP)
            if p = φ then                                 ▷ c3
                if Internal(e) then
                    p = p'
                    w = w'
                else
                    if w' > density(p) then
                        p = p'
                        w = w'
            if p ≠ p' then                                ▷ c4
                if w < density(p) then
                    if Internal(e') then
                        p = p'
                        w = w + w'
                    else
                        if (w + w') > density(p') then
                            p = p'
                            w = w + w'
            if p = P' then                                ▷ c5
                e = e + e'
```

C1: In this condition, we can simply add the new edge e' to the KG and assign p' as its corresponding partition. This will result in the increasing of the average density for two reasons. First, it will not affect the average density of any other partitions in the KG. Second, it will always increase the average density of the new partitions p'_i as it did not previously exist in the KG.

C2: In this condition, we can only aggregate if the weight of the current edge w is less than the density of its corresponding partition, $w \leq density(p)$. The reason is that according to Proposition 1 removing e will not reduce the density of p. We call it an *expansion* condition, where a partition tries

to expand its territory around the borders and take over the other partition. Proposition 1 ensures that no partition P can completely take over another partition P' unless the weight of the largest edge in P' is smaller than the density of P.

C3: This is when a nonexisting edge is going to be added to an existing partition p. There are two possible scenarios depending on either the newly created edge e is going to be an internal edge related to the partition p or not. An edge E is called *internal* with respect to a specific partition P if both vertices incident to E are connected to other edges with the same partition P. Based on this, if e will become an internal edge, then the algorithm aggregates e' without further consideration because the aggregation always increases the density of p and does not affect the density of any other partitions in the KG. On the other hand, if e is not an internal edge, then it can only be aggregated if $w \geq density(p)$ according to Proposition 1.

C4: In this condition, the aggregation will change the partition p of an existing edge e to another existing partition p' and moving its weight w to the p'. Since we are dealing with two existing partitions p and p', we need to check the optimization conditions on both partitions. In particular, we have to make sure that removing an edge with weight w from p and adding an edge with weight $w + w'$ to p' do not reduce their corresponding densities. Following Proposition 1, removing is allowed if $w \leq density(p)$. However, aggregation to the p' depends on whether the new edge is internal or not. If it is an internal edge, then we can apply the aggregation following the same reasoning in C3. However, for noninternal edge, the aggregation is only allowed if the weight of the new edge is larger than the density $w + w' \geq density(p')$. This is another example of the expansion condition similar to C3.

C5: The last condition is aggregating an edge with partition p' from DG to an existing edge e with the same partition p in the KG. We call this a *reinforcement* condition, where only the weight of an edge in a specific partition will increase. It is explicitly clear that this operation always results in the increase in the density of the corresponding partition p and does not affect any other partitions in the KG. Thus, the algorithm aggregates w' with w on e in the KG.

4 Experiments and result

In this section, we demonstrate the *accuracy* and *scalability* of GDTM by running the algorithm over two sets of experiments. To measure the accuracy, we run a set of supervised experiments on a tagged Twitter dataset and report the B-Cubed [16] score, and for scalability, we use a large-scale Twitter dataset and report the execution time and the coherence score [17] of the extracted topics. B-Cubed is an standard evaluation metric that measures the accuracy of a classification task. Each experiment is repeated 100 times, and the average is reported. We compare the results with four state-of-the-art approaches and show that GDTM significantly outperforms the others on both accuracy and scalability. All experiments are executed on a machine with 48 cores of $2GHz$ CPUs and $20GBs$ of RAM.

4.1 Datasets

In our experiments, we use a Twitter dataset collected during 2014 over the geographic area of London. The dataset contains 9.8 million tweets. We extracted the data related to 3 months of March, April and May from the original dataset to use in our scalability experiments. The dataset contained $1.8M$ tweets. We cleaned the dataset by removing *URLs, Punctuation Marks, Hashtags*, and *Mentions* and keep the tweets containing more than three words. The resulting dataset was reduced to $1.2M$ tweets. Next, we created a tagged dataset from the cleaned dataset for the experiments on accuracy. To create the tagged dataset first, we extracted a list of trending topics, during the corresponding timespan (Mar–May 2014), from *Twitter's Official Blog*[1] and the English Wikipedia page for *Reporting the Events from 2014 in the United Kingdom*.[2] Then, we hand-tagged the tweets in the clean dataset using the extracted topics and removed the topics with less than 100 occurrences. The remaining contained $26K$ tweets from 22 different topics. Figure 3 shows the titles and the overall distribution of the topics. As we can see, the topics cover a wide range of events from domestic (e.g., *London Marathon*) to international (e.g., *EuroVision*, *WorldCup* and *Oscar's Award*) and contain subjects from overlapping categories (e.g., *WorldCup*, *FACup* and *London-Marathon* from the sports category) (Table 3).

4.2 Evaluation metrics

B-Cubed [16] is a statistical metric to measure the accuracy of a classification compared to the ground truth. It is calculated as the average F-score over all documents. Given a dataset D with n documents, tagged with k hand labels, $L = \{l_1, \ldots, l_k\}$ and a classification of the documents into k class labels, $C = \{c_1, \ldots, c_k\}$, the B-Cubed of a document d with hand label l_d and class label c_d is calculated as:

$$B(d) = 2 \times \frac{P(d) \times R(d)}{P(d) + R(d)}, \tag{9}$$

[1] https://blog.twitter.com/official/en_gb/a/en-gb/2014/2014-the-year-on-twitter.html.

[2] https://en.wikipedia.org/wiki/2014_in_the_United_Kingdom.

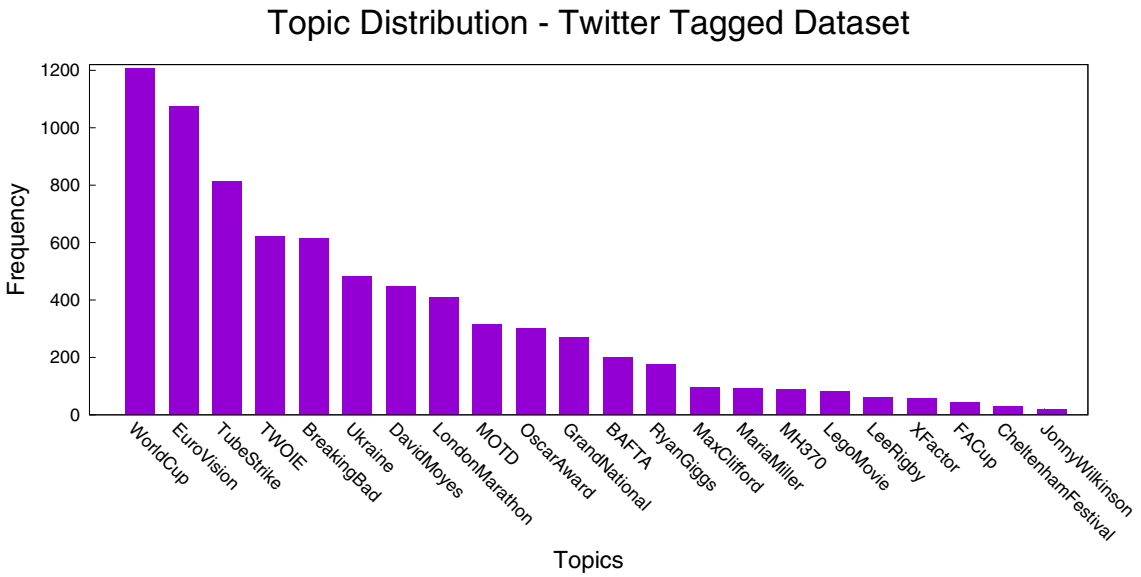

Fig. 3 Topic distribution in the Twitter tagged dataset for analyzing the accuracy of GDTM. A list of most frequent topics published in Twitter from March to April 2014 in London

Table 3 Comparison between different approaches and GDTM with respect to support for the three main properties in DTM, namely (i) sparsity, (ii) scalability and (iii) dynamicity

	Sparsity	Scalability	Dynamicity
LDA	×	×	×
BTM	✓	×	×
GDTM	✓	✓	✓
CDTM	×	×	✓
PYPM	×	×	✓

where P and R stand for *precision* and *recall*, respectively, and are calculated as follows:

$$P(d) = \frac{|d'|_{\{\forall d' \in D: c_{d'} = c_d, l_{d'} = l_d\}}}{|d'|_{\{\forall d' \in D: c_{d'} = c_d\}}} \tag{10}$$

$$R(d) = \frac{|d'|_{\{\forall d' \in D: c_{d'} = c_d, l_{d'} = l_d\}}}{|d'|_{\{\forall d' \in D: l_{d'} = l_d\}}}. \tag{11}$$

Precision shows the likelihood of documents correctly classified in a specific class c, with respect to the total number of documents in that class, whereas the recall represents the likelihood with respect to the total number of documents in a specific label l. The total B-Cubed score is calculated as the average over all documents in the dataset:

$$B_{total} = \frac{1}{n} \times \sum_{i=1}^{n} B(d_i). \tag{12}$$

Note that precision and recall measure the quality of the classifications with respect to the tagged labels for individual categories of the problem (e.g., individual topics), and therefore, they provide more accurate evaluation compared to more general methods like Coherency that provide an average over all instances. That is the main reason that we decided to use precision and recall in our supervised classification task.

Coherency [17] is an evaluation metric for measuring the quality of extracted topics in a topic classification problem. It assumes that the most frequent words in each class tend to have higher co-occurrence among the documents in that class rather than the documents across multiple classes. .Thus, given a set of documents classified into k topics, $T = \{t_1, \ldots, t_k\}$, first, the coherency of each topic, z, with top m probable words, $W^z = \{w_1, \ldots, w_m\}$, is calculated as,

$$C(z, W^z) = \sum_{i=2}^{m} \sum_{j=1}^{i-1} \log \frac{D(w_i{}^z, w_j{}^z)}{D(w_j{}^z)}, \tag{13}$$

where $D(w_i{}^z, w_j{}^z)$ is the co-occurrence frequency of the words v_i and v_j among documents in z and $D(w_j{}^z)$ is the total frequency of w_j in z. Then, the total coherency of partitioning is calculated as:

$$C(T) = \frac{1}{k} \times \sum_{z \in T} C(z, W^z). \tag{14}$$

4.3 Baseline and experimental settings

We compare GDTM with four state-of-the-art approaches, namely LDA [3], BTM [4], CDTM [6] and PYPM [10]. The source codes are available and downloaded from their corresponding URLs (LDA[3], BTM[4], CDTM[5] and PYPM[6]). Table 2 shows a summary of all approaches with respect to their support for sparsity, scalability and dynamicity, in comparison with GDTM. As we can see, GDTM is the only approach that satisfies all three properties, which we will analyze and show in the coming sections.

Accuracy. In this experiment, we use the tagged Twitter dataset to compare the accuracy of different algorithms on dealing with sparsity in short texts. We run each approach 100 times and report the average B-Cubed results. To set the parameter values, we run a set of validation experiments and use the values with the best performance during the main experiments. For GDTM, we set RI parameters as: $d = 1000$, $\omega = 2$ and $\zeta = 8$, and we use $m = 1$ for the skip-gram parameter and set the rest of the parameters as: $\delta = 60$ and $\gamma = 0.01$. For LDA, BTM, CDTM and PYPM, the parameter values are set to $\alpha = 0.01$, $\beta = 0.05$ and the number of Gibbs sampling iterations is set to 10. For LDA, BTM and CDTM, which require to know the number of topics k in advanced, we set the correct number of topics in the dataset $k = 22$. In addition, the value of the CDTM's specific parameter, called *top_chain_variance*, is set to 0.005, as suggested by the authors.

Scalability. This experiment is designed and run in an online structure to show the power of the GDTM to account for dynamicity and scalability in comparison with the state-of-the-art approaches. Since none of the four baseline approaches are real streaming solutions, to comply with our streaming model we developed a mini-batch streaming mechanism as follows. First, we sorted the documents by date and considered the cardinal number of the documents as the lowest possible discretization value for the streaming. Then, we used a snapshot period to extract the results for each algorithm. The snapshot was set to $10K$, and we run each algorithm over the entire dataset. More specifically, for every $10k$ documents, we calculate and report the coherence score for the extracted partitioning for each algorithm. Our initial experiments show that CDTM and PYPM are not tractable using the available resources. In particular, they required more than $20GBs$ of memory after processing around $350K$ and $80K$ number of documents, respectively. Therefore, we had to exclude those two approaches from the scalability experiments and already confirm their limitation to scale. Based on that, in this experiments we only compare GDTM with LDA and BTM and we set all the parameters to the same values as the experiments on accuracy. We only changed the number of topics in LDA and BTM to 100, which is the most commonly used value in conditions with unknown number of topics. The topic qualities are measured over each snapshot state using coherence score for three different values 5, 10 and 20 as the number of top words per partition in each evaluation step.

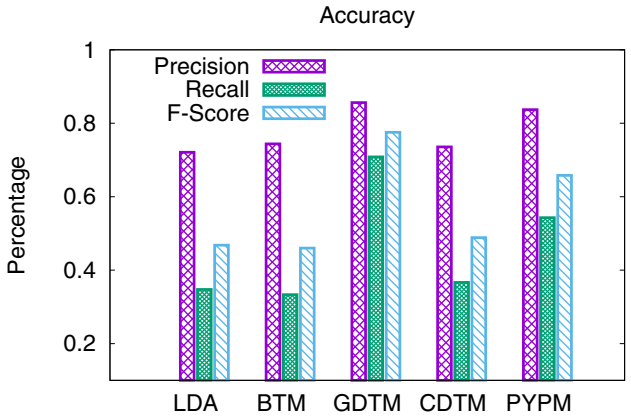

Fig. 4 Comparing the average performance accuracy of GDTM with four state-of-the-art approaches. The experiments were run on a tagged Twitter dataset containing $26K$ tweets from 22 different topics, and the results are reported as the average over 100 runs. GDTM shows the best performance on both precision and recall compared to the other solutions

5 Discussion

We now turn into the details of the results over the two sets of experiments.

Accuracy. Figure 4 shows the results of the accuracy of the topic assignment task performed by different algorithms over the tagged Twitter dataset. The results show that GDTM significantly outperforms the other approaches with a P value less than 0.01 over 95% confidence interval, in all cases. GDTM shows the largest value on precision, which is an illustration of its strong language modeling approach. In particular, the application of the skip-gram method enables GDTM to cope with the sparsity by extracting the modest amount of information from the sparse contexts of the tweets and enriching the feature vector representations.

A more interesting outcome to be considered is the remarkable improvement over the value of precision on GDTM compared to all other approaches. This is due to the strong partitioning mechanism in GDTM that allows the algorithm to automatically choose the best number of topics and prevents the incorrect mixing of the documents. Note that

[3] http://gibbslda.sourceforge.net/.

[4] https://github.com/xiaohuiyan/BTM.

[5] https://github.com/blei-lab/dtm.

[6] https://github.com/qiang2100/PYPM.

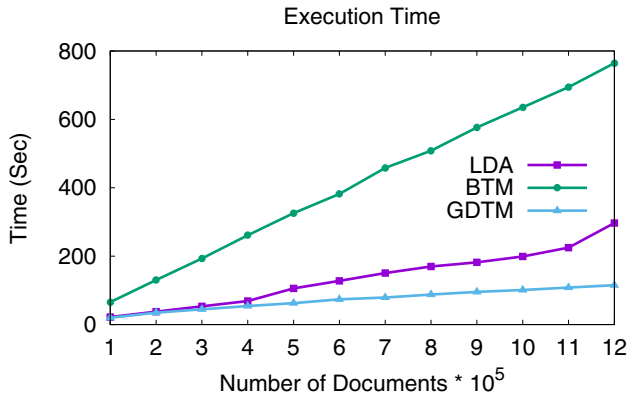

Fig. 5 Average execution time for running LDA, BTM and GDTM over the large-scale Twitter dataset. As we can see, GDTM shows a linear time complexity as opposed to the baseline models. In particular, GDTM performs at least three times faster than both models

the higher recall value on PYPM compared to the other three approaches, namely LDA, BTM and CDTM, confirms this statement, as PYPM supports an infinite number of topics, similar to GDTM. In summary, GDTM is the only approach with high values on both precision and recall, which ensue the largest overall F-score of 77.5%. This result is around 11% larger than the second best approach, PYPM, with B-Cubed score of 65.8%.

Scalability Figure 5 shows the comparison between the execution time of running different algorithms over the large-scale Twitter dataset. GDTM has a constant execution time compared to the other approaches. In particular, it performs by around 3 times faster than the LDA and an order of magnitude faster than the BTM over $1.2M$ documents. At the same time, GDTM does not sacrifice the quality to gain such significant performance gain. In fact, the quality of extracted partitions is significantly higher than both LDA and BTM as shown in Fig. 6. The coherence scores of the partitions created by GDTM are between four and five times larger than those extracted by BTM and LDA, respectively. Also, the analogous results over multiple experiments with 5, 10 and 20 number of top words per partition confirm the significance of the outcomes. The main justification

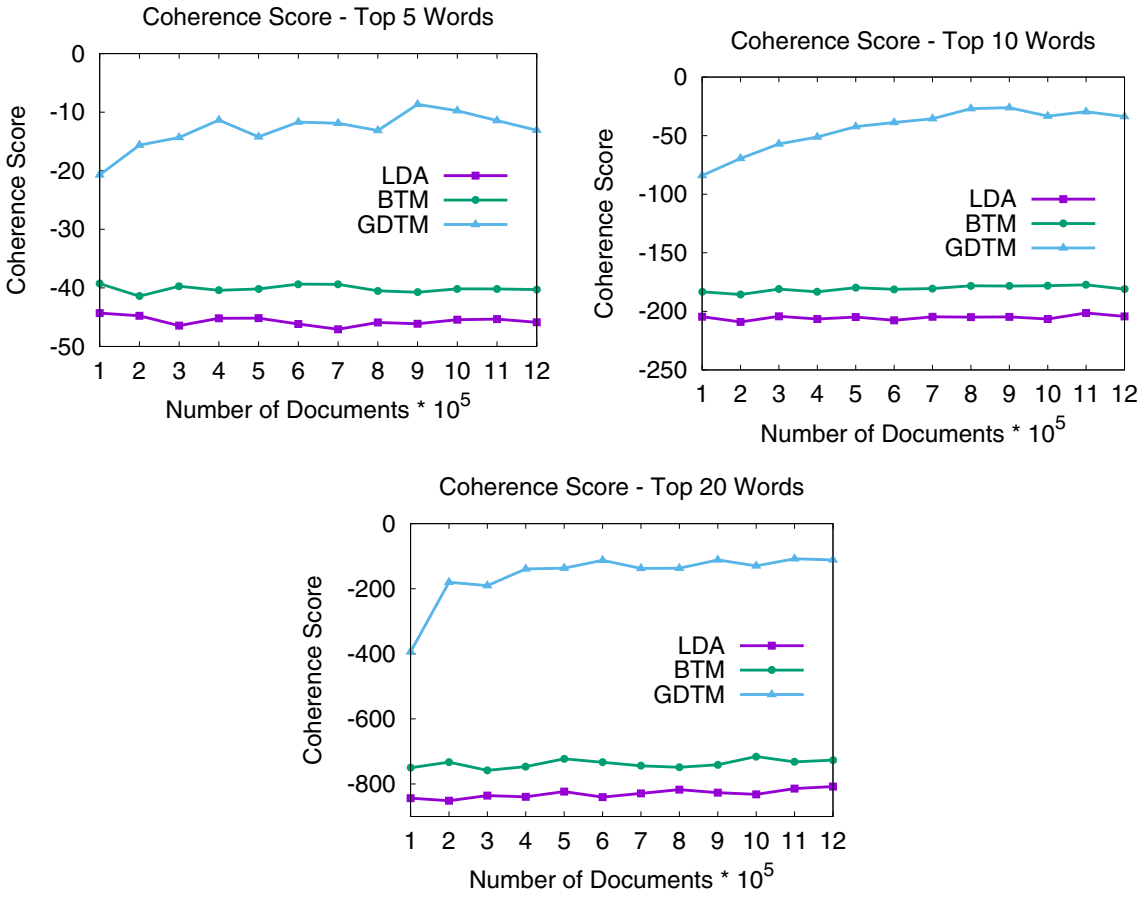

Fig. 6 Average coherence score over 100 runs on different snapshots. The documents are sorted by date, and for every 10k documents, a snapshot is created and the coherence score is calculated. (The figures are summarized to $100k$ for the sake of presentation). The score is measured over three different numbers of top sample words from different partitions including 5, 10 and 20. GDTM shows a higher coherence score in all three cases

behind this remarkable result is the rich feature representation model in GDTM that enables the algorithm to create and extract high-quality partitions without requiring the usual iterative optimization algorithms used in other approaches. The other justification lies in the automatic feature representation model in GDTM that enables the emergence and disappearance of the partitions following the natural dynamics of their representative topics in the stream, which enables the algorithm to adapt to the changes of the topics in the stream.

6 Conclusion

We developed GDTM, a solution for dynamic topic modeling on short texts in online social networks. Natural language is the best model for its own representation; however, the sparsity, velocity and dynamicity of short texts make it a difficult task to develop appropriate models for extracting topics from these texts. GDTM overcomes this problem with an online topic modeling approach. It first combines an incremental dimensionality reduction method called Random Indexing with a language representation technique called Skip-gram to construct a strong feature representation model. Then, it uses a novel graph representation technique and a graph partitioning algorithm to extract the topics in an online approach. We examine the accuracy and scalability of GDTM and compare the results with four state-of-the-art approaches. The results show that GDTM significantly outperforms all other solutions on both accuracy and scalability.

Even though we only applied GDTM on short texts in this paper, we strongly claim that application is not limited to linguistic data. In fact, GDTM provides a generic algorithm for automatic feature extraction over any stream of data that can be presented in some form of discrete representation level. This opens a new track of research to be considered in our future plans.

Acknowledgements Open access funding provided by Royal Institute of Technology.

References

1. Allan, J., Carbonell, J., Doddington, G., Yamron, J., Yang, Y., Umass, J.A., Cmu, B.A., Cmu, D.B., Cmu, A.B., Cmu, R.B., Dragon, I.C., Darpa, G.D., Cmu, A.H., Cmu, J.L., Umass, V.L., Cmu, X.L., Dragon, S.L., Dragon, P.V.M. Umass, R.P., Cmu, T.P., Umass, J.P., Umass, M.S.: Topic detection and tracking pilot study final report. In: Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop, pp. 194–218, (1998)
2. Crain, S.P., Zhou, K., Yang, S.-H., Zha, H.: Dimensionality Reduction and Topic Modeling: From Latent Semantic Indexing to Latent Dirichlet Allocation and Beyond Mining Text Data, pp. 129–161. Springer, Berlin (2012)
3. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
4. Yan, X., Guo, J., Lan, Y., Cheng, X.: A Biterm topic model for short texts. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 1445–1456, (2013)
5. Blei, D. M., Lafferty, J. D.: Dynamic topic models. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 113–120, (2006)
6. Wang, C., Blei, D., Heckerman, D.: Continuous time dynamic topic models. In: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, pp. 579–586, (2008)
7. Liang, S., Yilmaz, E., Kanoulas, E.: Dynamic clustering of streaming short documents. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 995–1004, (2016)
8. Yin, J., Wang, J.: A dirichlet multinomial mixture model-based approach for short text clustering. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 233–242, (2014)
9. Yin, J., Wang, J.: A Text clustering algorithm using an online clustering scheme for initialization. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1995–2004, (2016)
10. Jipeng, Q., Yun, L., Yunhao, Y., Xindong, W.: Short Text clustering based on Pitman–Yor process mixture model. Appl. Intell. **48**, 1802–1812 (2018)
11. Nigam, K., Mccallum, A.K., Thrun, S., Mitchell, T.: Text classification from labeled and unlabeled documents using EM. Mach. Learn. **39**, 103–134 (2000)
12. Geman, S., Geman, D.: Stochastic relaxation gibbs distributions, and the Bayesian restoration of images. IEEE Trans. **6**, 721–741 (1984)
13. Hoffman, M.D., Blei, D.M., Wang, C., Paisley, J.: Stochastic variational inference. J. Mach. Learn. Res. **14**, 1303–1347 (2013)
14. Sahlgren, M.: An introduction to random indexing. In: Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, Litera, (2005)
15. Tomas, M., Kai, C., Greg, C., Dean, J.: Efficient estimation of word representations in vector space, CoRR, arXiv:1301.3781 (2013)
16. Bagga, A., Baldwin, B.: Entity-based Cross-document Coreferencing using the vector space model. In: Proceedings of the 17th International Conference on Computational Linguistics, vol. 1, pp. 79–85, (1998)
17. Mimno, D., Hanna Wallach, M., Talley, E., Leenders, M., McCallum, A.: Optimizing semantic coherence in topic models. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 262–272, (2011)
18. Hofmann, T.: Probabilistic latent semantic indexing. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 50–57, (1999)

19. Kálmán, R.E.: A New Approach to Linear Filtering and Prediction Problems. J Basic Eng **82**(1), 35–45 (1960)
20. Sato, I., Nakagawa, H.: Topic models with power-law using Pitman–Yor process. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 673–682, (2010)
21. Johnson, W.B., Lindenstrauss, J., Schechtman, G.: Extensions of lipschitz maps into banach spaces. Israel J. Math. **54**, 129–138 (1986)
22. Firth, J.: Studies in Linguistic Analysis. Philological Society, Oxford (1957)
23. Harris Zellig, S.: Distributional Structure, pp. 3–22. Springer, Berlin (1981)
24. Mitchell, J., Lapata, M.: Language models based on semantic composition. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, Volume 1, pp. 430–439, (2009)
25. Guthrie, D., Allison, B., Liu, W., Guthrie, L., Wilks, Y.: A closer look at skip-gram modelling. In: Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'06), European Language Resources Association (ELRA), (2006)
26. Ghoorchian, K., Girdzijauskas, S., Rahimian, F.: DeGPar: Large scale topic detection using node-cut partitioning on dense weighted graphs. In: IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 1.0, pp. 775–785, (2017)