Abstract—Density-based spatial clustering of applications with noise (DBSCAN) is the most commonly used density-based clustering algorithm, where it can discover multiple clusters with arbitrary shapes. DBSCAN works properly when the input data type is homogeneous, but the DBSCAN’s approach may not be sufficient when the input dataset has textual heterogeneity (e.g., when we intend to find clusters from geo-tagged posts on social media relevant to a certain point-of-interest (POI)), thus leading to poor performance. In this paper, we present DBSTexC, a new density-based clustering algorithm using spatio–textual information on Twitter. We first define POI-relevant and POI-irrelevant tweets as the records that contain and do not contain a POI name or its coherent variations, respectively. By taking into account the fractions of POI-relevant and POI-irrelevant tweets, our DBSTexC algorithm shows a much higher clustering quality than the DBSCAN case in terms of the $F_1$ score and its variants. DBSTexC can be thought of as a generalized version of DBSCAN due to the findings that it performs identically as DBSCAN when the inputs are homogeneous and far outperforms DBSCAN when the heterogeneous input data type is given.

I. INTRODUCTION

Several clustering algorithms such as K-means [1], Gaussian mixture models [2], hierarchical clustering [3], graph-based analysis [4], and density-based clustering have been introduced. Among them, density-based clustering algorithms have steadily been investigated to discover insights in geospatial data.

Recently, together with the growth of online social networks, the volume of spatio–textual data is increasing drastically. As a result, researches on clustering algorithms based on spatio–textual data have gained a growing interest among researchers [5]. From a density-based approach, density-based spatial clustering of applications with noise (DBSCAN) [6] stands out as the most commonly used algorithm due to the robustness to noise, discovering clusters with arbitrary shapes, and proper operation without prior assumptions about the number of clusters. Its variations have also been well studied in [7], [8]. However, when we aim at finding clusters and their geographic regions from geo-tagged posts on social media relevant to a certain point-of-interest (POI), DBSCAN and its variations may not work properly. This is because while the region around a POI generally includes geo-tags that contain and do not contain annotated POI keywords (denoted as POI-relevant and POI-irrelevant geo-tags, respectively), DBSCAN takes only into account the former in the clustering process. Although clusters via DBSCAN seem to correctly identify groups of relevant geo-tags on the surface, they often blindly include regions containing a large number of irrelevant geo-tags, resulting in a poor clustering quality. Thus, using homogeneous inputs consisting only of relevant geo-tags is an incomplete approach to finding clusters. It is needed to perform clustering based on heterogeneous inputs including both POI-relevant and POI-irrelevant geo-tags since they provide the comprehensive picture of POIs on social media.

In this paper, to solve this inherent problem of DBSCAN, we propose DBSTexC, a novel density-based clustering algorithm using spatio–textual information on Twitter [9], [10], which takes into account heterogeneous inputs composed of both relevant and irrelevant geo-tagged tweets in the clustering process. Our contributions are threefold as follows:

• We introduce a new algorithm, named DBSTexC, for density-based clustering on Twitter, which incorporates textual information into the DBSCAN framework to avoid geographical regions with numerous irrelevant geo-tagged posts.
• We formulate our performance metric in terms of the $F_1$ score and its variants, and then extensively evaluate the clustering performance of our DBSTexC algorithm while showing its superiority over DBSCAN.
• We also analyze the computational complexity.

II. DATASET

In this section, we describe how we collect POIs and the Twitter data associated with the POI locations. For each POI, we also present our approach to searching for relevant and irrelevant tweets.

A. Collecting POIs

We choose POIs as specific point locations that people may find useful or interesting. In addition, to increase the
geographic diversity, we consider POIs from both populous metropolitan areas and small cities. The list of chosen POIs is summarized in Table I.

<table>
<thead>
<tr>
<th>POI name</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyde Park</td>
<td>Metropolitan area</td>
</tr>
<tr>
<td>Regent’s Park</td>
<td>Metropolitan area</td>
</tr>
<tr>
<td>University of Oxford</td>
<td>Small city</td>
</tr>
<tr>
<td>Edinburgh Castle</td>
<td>Small city</td>
</tr>
</tbody>
</table>

### B. Collecting Twitter Data

For data collection, we use Twitter Streaming Application Programming Interface (API). Our dataset is composed of a large set of geo-tagged tweets collected from Twitter users for about one month in June, 2016 in the UK. We removed the content that was created by users who tweeted more than three times consecutively without moving, since it was likely to be generated by other services such as Tweetbot, TweetDeck, and so forth. We observe that each tweet contains a number of entities that can be differentiated by their attributed field names. For data analysis, we adopt the following three fields from the collected tweets:

- text: actual UTF-8 text of the status update
- lat: latitude of the tweet’s location
- lon: longitude of the tweet’s location

### C. Searching for POI-Relevant Tweets

Since Twitter users have the tendency to tag or to mention a POI name in their tweets to express their interest in the POI, we can easily query all relevant tweets by searching for keywords associated with that POI in the users’ text field. Since users tend to type the real-world terms of each POI into the tweet box, a POI name may be misspelled or have other words tacked on to it. We perform a keyword-based search by querying semantically coherent variations of a POI, which would include its abbreviated names, its nicknames (if any), etc. For a POI having a large geographic area, names of famous attractions inside the POI itself are also included to improve the search accuracy. The list of search queries for each POI is summarized in Table II. Consequently, the dataset can be partitioned into two subsets of geo-tagged tweets that contain and do not contain the annotated POI keywords.

### III. Proposed Methodology

In this section, we first introduce important definitions that are necessary to design our algorithm and then describe the proposed DBSTexC algorithm.

#### A. Definitions

We begin by presenting the definition of a query region. A query region is a geographic area from which we collect the geo-tagged tweets about a certain POI. Apparently, we expect to find both POI-relevant and POI-irrelevant tweets inside the region. However, since the relevance of data to the POI varies inversely with the geographic distance between the POI and the locations where the data are generated, tweets posted in locations far away from the POI generally have little or no textual description for the POI. To reduce the computational complexity, we thus define a region that includes almost all relevant tweets but excludes the majority of irrelevant tweets that were posted geographically far from the POI. Motivated by this observation, we define a query region as follows:

**Definition 1:** (Query region) Given a POI, a query region is a circle whose center corresponds to the center point of the POI’s administrative bounding box provided by Google Maps. We then increase the radius of the circle stepwise until the number of new POI-relevant tweets found in one increment step is lower than a threshold \( \eta \), where \( \eta \) can be set differently according to POI types.

Similarly as in DBSCAN [6], we utilize the neighborhood of a point (See Definition 2) and a series of density-connected points to find clusters. On the other hand, to improve the clustering quality, we introduce a new parameter \( N_{\text{max}} \) to control the number of POI-irrelevant tweets. Thus, we can acquire a core point having not only at least \( N_{\text{min}} \) relevant tweets but also at most \( N_{\text{max}} \) irrelevant tweets around the point (See Definition 3). The result of DBSTexC, whose clusters consist of connected neighborhoods of core points, is now of much higher quality than that of DBSCAN that uses only relevant tweets.

**Definition 2:** (\( \epsilon \)-neighborhood of a point) Let \( X \) and \( Y \) denote the sets of POI-relevant and POI-irrelevant tweets, respectively. For a point \( p \in X \), the sets of \( \epsilon \)-neighborhoods containing relevant and irrelevant tweets, denoted by \( X_{\epsilon}(p) \) and \( Y_{\epsilon}(p) \), are defined as the geo-tagged tweets within a scan circle centered at \( p \) with radius \( \epsilon \) that satisfy

\[
X_{\epsilon}(p) = \{ q \in X | \text{dist}(p, q) \leq \epsilon \}
\]

\[
Y_{\epsilon}(p) = \{ q \in Y | \text{dist}(p, q) \leq \epsilon \},
\]

respectively, where \( \text{dist}(p, q) \) is the geographic distance between coordinates \( p \) and \( q \). Note that we define the \( \epsilon \)-neighborhood only for POI-relevant tweets while ignoring the neighborhood of POI-irrelevant tweets, because our DBSTexC algorithm connects a series of \( \epsilon \)-neighborhoods of relevant tweets in the clustering process.

**Definition 3:** (Core point) A point \( p \in X \) is called a core point if the following condition is fulfilled:

\[
|X_{\epsilon}(p)| \geq N_{\text{min}} \text{ and } |Y_{\epsilon}(p)| \leq N_{\text{max}}.
\]
Given the above definition of a core point, the subsequent notions of (directly) density-reachable, density-connected, cluster, and noise originated from DBSCAN can be applied into our DBSTexC framework accordingly.

### B. DBSTexC Algorithm

In this section, we elaborate on our DBSTexC algorithm that takes into account both POI-relevant and POI-irrelevant tweets. To find a cluster, DBSTexC begins with a random point \( p_i \) in the set of POI-relevant tweets for \( i \in \{1, ..., |X|\} \) and retrieves all points that are density-reachable from \( p_i \) with respect to \( \epsilon, N_{\min}, \) and \( N_{\max} \) (See Algorithm 1). If \( p_i \) is a core point, then a cluster is created and expanded until all points belonging to the cluster are added (See Algorithm 2). Otherwise, there is no point that is density-reachable from \( p_i \).

In this case, DBSTexC moves on to the next point in the set of POI-relevant tweets.

**Algorithm 1** DBSTexC \((X', Y', \epsilon, N_{\max}, N_{\max})\)

**Input:** \( X', Y', \epsilon, N_{\min}, N_{\max} \)

**Output:** Clusters with different labels \( C \)

**Initialization:** \( C \leftarrow 0; n \leftarrow |X'|; m \leftarrow |Y'|; p_i \) is a point in the set \( X' \)

1: for each \( p_i \) do
2: if \( p_i \) is not visited then
3: Mark \( p_i \) as visited
4: \( \{X_i(p_i), Y_i(p_i)\} = \text{RegionQuery}(p_i) \)
5: if \( |X_i(p_i)| \geq N_{\min} \) and \( |Y_i(p_i)| \leq N_{\max} \) then
6: \( C \leftarrow C + 1 \)
7: ExpandCluster\((p_i, X_i(p_i), Y_i(p_i))\)

In Algorithm 1, RegionQuery() is a function to retrieve points in an \( \epsilon \)-neighborhood, where it can be executed using spatial access methods such as R-trees. By querying both relevant and irrelevant points and using two parameters \( N_{\min} \) and \( N_{\max} \) to make a decision on whether to create a new cluster and/or expand the current cluster, our DBSTexC algorithm effectively excludes noisy areas from its clusters.

**Algorithm 2** ExpandCluster\((p_i, X_i(p_i), Y_i(p_i))\)

**Input:** \( p_i, X_i(p_i), Y_i(p_i) \)

**Output:** Cluster \( C \) with all of its members

1: Add \( p_i \) to the current cluster
2: for each point \( p_j \) in the set \( X_i(p_i) \) do
3: if \( p_j \) is not visited then
4: Mark \( p_j \) as visited
5: \( \{X_j(p_j), Y_j(p_j)\} = \text{RegionQuery}(p_j) \)
6: if \( |X_j(p_j)| \geq N_{\min} \) and \( |Y_j(p_j)| \leq N_{\max} \) then
7: \( X_i(p_i) = X_i(p_i) \cup X_j(p_j) \)
8: \( Y_i(p_i) = Y_i(p_i) \cup Y_j(p_j) \)
9: if \( p_j \) does not have a label then
10: Add \( p_j \) to the current cluster
11: if \( |Y_j(p_j)| \neq 0 \) then
12: for each point \( q_k \) in the set \( Y_j(p_j) \) do
13: if \( q_k \) is not visited then
14: Mark \( q_k \) as visited
15: if \( q_k \) does not have a label then
16: Add \( q_k \) to the current cluster

In Algorithm 2, for every point \( p_j \) in the neighbor set \( X_i(p_i) \), we examine the \( \epsilon \)-neighborhood of \( p_j \). If \( p_j \) is a core point, then \( p_j \) is added to the current cluster and we proceed on by appending its neighbors to the neighbor sets \( X_i(p_j) \) and \( Y_i(p_j) \). This process is repeated until all the points in the set \( X_i(p_j) \) are examined. Finally, when the process is terminated, we add points in the set \( Y_i(p_j) \) to our current cluster.

### IV. Experimental Results and Discussion

Using the proposed DBSTexC algorithm in Section III-B, we show experimental results based on our performance metric and then analyze the overall average computational complexity.

#### A. Performance Metric

We use the \( F_1 \) score as a component of our performance metric, since it is widely used in machine learning and statistical analysis as a measure of a test’s accuracy and thus can be considered a good tool to assess the clustering quality. The \( F_1 \) score is expressed as

\[
F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}},
\]

which indicates the harmonic mean of Precision and Recall. Here, Precision is the ratio of true positives (the number of POI-relevant points in clusters) to all predicted positives (the number of points in clusters), that is \( \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \), and Recall is the ratio of true positives (the number of POI-relevant points in clusters) to actual positives that should have been returned (the total number of POI-relevant points), that is \( \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \).

In the problem of finding clusters from geo-tagged tweets relevant to a POI, the area covered by the clusters can raise a big concern, as several applications such as geo-marketing may desire a widespread geographic area. Thus, although it is desirable to find clusters with the highest \( F_1 \) score, it would also be good to considerably extend the area of the resulting clusters at the expense of a slightly reduced value of \( F_1 \) in some applications. Therefore, we formulate the following new performance metric expressed as the product of a power law in the area \( A \) (in km\(^2\)) and the \( F_1 \) score:

\[
A^\alpha F_1,
\]

where \( \alpha > 0 \) is the area exponent, balancing between different levels of geographic coverage. For small \( \alpha \), clusters with the almost highest \( F_1 \) score are returned. However, as \( \alpha \) increases, clusters that cover a wide area are obtained at the cost of a reduced \( F_1 \). Therefore, given parameters for the two algorithms (i.e., \( \epsilon, N_{\min} \) for DBSCAN and \( \epsilon, N_{\min}, N_{\max} \) for DBSTexC), we can calculate the metric (1) along with the corresponding \( F_1 \) score and the cluster area in each case.

#### B. Experimental Results

We show the experimental results according to different values of \( \alpha > 0 \). As for the query region, we assume that \( \eta = 5 \) for Hyde Park and Regent’s Park; and \( \eta = 3 \) for University of Oxford and Edinburgh Castle, which can also be set to other values to control the quality constraint. The performance of
both DBSTexC and DBSCAN for four POIs is summarized and compared in Table III, where \( \alpha \in \{0.5, 0.75, 1\} \). From the table, one can see that DBSTexC outperforms DBSCAN for all four chosen POIs, especially for Hyde Park, which is one of the largest and most popular parks in London. In Fig. 1, we illustrate the clustering results of DBSCAN and DBSTexC for Hyde Park when \( \alpha = 0.5 \). To highlight the performance difference, we depict the actual cluster region together with the distribution of irrelevant tweets. From the figure, we observe that DBSTexC successfully excludes a large number of irrelevant tweets out of the cluster region, while covering a much bigger geographic area compared to that of DBSCAN. This underscores the robust ability of DBSTexC to find high-quality clusters in terms of our performance metric \( A^\alpha F_1 \).

C. Computational Complexity

We hereby analyze the runtime complexity of the DBSCAN and DBSTexC algorithm. The complexity of both algorithms is given by the input size times the basic operation \( \epsilon \)-neighborhood query, which indeed dominates the complexity. Then, without a spatial index, the overall runtime complexities of DBSCAN and DBSTexC are \( O(n^2) \), and \( O(n^2 + nm) \), respectively, where \( n \) and \( m \) denote the number of POI-relevant and irrelevant tweets, respectively. With a spatial index such as an \( R \)-tree, the overall runtime complexities of DBSCAN and DBSTexC are given by \( O(n \log n) \) and \( O(n \log nm) \), respectively. Hence, it is shown that the complexity of the two algorithms is comparable.

V. CONCLUDING REMARK

We introduced DBSTexC by utilizing spatio-textual information on Twitter. We showed that the proposed DBSTexC outperforms DBSCAN in terms of maximizing \( A^\alpha F_1 \), where \( \alpha \) is the area exponent. In addition, we analyzed the average runtime complexity of DBSTexC, which is given by \( O(n \log nm) \) when using a spatial index, and \( O(n^2 + nm) \) otherwise. DBSTexC can be viewed as a generalized version of DBSCAN since it not only performs identically as DBSCAN when inputs are homogeneous but also is extended for the case where the heterogeneous input data type is given.

ACKNOWLEDGMENT

This research was supported by the Basic Science Research Program through the National Research Founda-

<table>
<thead>
<tr>
<th>POI name</th>
<th>DBSCAN</th>
<th>DBSTexC</th>
<th>Improvement Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyde Park</td>
<td>0.7096</td>
<td>1.0483</td>
<td>47.73%</td>
</tr>
<tr>
<td>Regent’s Park</td>
<td>1.0947</td>
<td>1.0959</td>
<td>0.11%</td>
</tr>
<tr>
<td>University of Oxford</td>
<td>0.6038</td>
<td>0.6884</td>
<td>14.01%</td>
</tr>
<tr>
<td>Edinburgh Castle</td>
<td>0.3377</td>
<td>0.4380</td>
<td>29.70%</td>
</tr>
</tbody>
</table>

\[
A^\alpha F_1 (\alpha = 0.5) \]

<table>
<thead>
<tr>
<th>POI name</th>
<th>DBSCAN</th>
<th>DBSTexC</th>
<th>Improvement Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyde Park</td>
<td>0.9444</td>
<td>1.4325</td>
<td>51.68%</td>
</tr>
<tr>
<td>Regent’s Park</td>
<td>1.4011</td>
<td>1.4179</td>
<td>1.20%</td>
</tr>
<tr>
<td>University of Oxford</td>
<td>1.4036</td>
<td>1.4040</td>
<td>0.03%</td>
</tr>
<tr>
<td>Edinburgh Castle</td>
<td>0.5506</td>
<td>0.5954</td>
<td>8.14%</td>
</tr>
</tbody>
</table>

\[
A^\alpha F_1 (\alpha = 0.75) \]

<table>
<thead>
<tr>
<th>POI name</th>
<th>DBSCAN</th>
<th>DBSTexC</th>
<th>Improvement Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyde Park</td>
<td>2.0932</td>
<td>2.6576</td>
<td>26.96%</td>
</tr>
<tr>
<td>Regent’s Park</td>
<td>1.9406</td>
<td>2.5698</td>
<td>32.42%</td>
</tr>
<tr>
<td>University of Oxford</td>
<td>3.2643</td>
<td>3.2651</td>
<td>0.02%</td>
</tr>
<tr>
<td>Edinburgh Castle</td>
<td>0.8977</td>
<td>0.8980</td>
<td>0.03%</td>
</tr>
</tbody>
</table>

\[
A^\alpha F_1 (\alpha = 1) \]

REFERENCES