Article

Outlier Detection and Prediction in Evolving Communities

Nikolaos Sachpenderis * and Georgia Koloniari

Department of Applied Informatics, School of Information Sciences, University of Macedonia, 540 06 Thessaloniki, Greece; gkoloniari@uom.edu.gr
* Correspondence: sachpenderis@uom.edu.gr

Abstract: Community detection in social networks is of great importance and is used in a variety of applications such as recommendation systems and targeted advertising. While detecting dense groups with high levels of connectivity and similar interests between their members is the main target of traditional network analysis, finding network members with quite different behavior than the majority of nodes is important as well. These nodes are known as outliers, and their accurate detection can be very useful; when outliers are marked as noisy nodes, their early exclusion from analysis can lead to high computational profits. On the other hand, they can represent interesting components that call for further investigation to find the reasons for their outlying behavior and possible ways to include them in a neighboring community. Both community and outlier detection are challenging in temporal environments where changes occur in real time; thus, dynamic methods need to be deployed rather than static methods. In our work, we take into account the content of the network, in contrast to most of related studies, where only the network’s structure contributes to community formation. We define an adaptive outlier score to be assigned to each node in order to quantify its outlierness, and introduce a complete online community detection algorithm that analyzes both the network’s structure and content while at the same time detecting community outliers. To evaluate our method, we retrieved and processed two real datasets regarding social networks with temporal and content information. Experimental results show that our method is capable of detecting outliers in real-time evolving communities and provides an outlier score which is a better metric of each node’s outlierness compared to widely used metrics. Finally, experimental results indicate that our method is suitable for predicting the status of future nodes based on their current outlier score.

Keywords: community detection; outlier detection; social networks; attributed networks

1. Introduction

One of the most important parts of social network analysis is community detection, that is, finding highly connected groups of nodes; this important task consists of studying the network and optimizing a function such as modularity to divide nodes into dense sub-networks, called communities. A wide definition of a community remains under discussion in the related literature; the main reason for this is its highly context-dependent nature. The basic idea for describing a community inside a social network is the existence of more intra-community edges than inter-community edges, reflecting more relations between community members than with other nodes outside the group [1, 2].

However, a social network often provides more information about relations of its members in the form of context or semantics. The inclusion of this information through node- or edge-attributed graphs can add important details to the analysis and conclude with more cohesive communities that do not solely rely on structural connections. For example, content information of a co-authorship network could include the conferences, journals, or general subject matter of the published papers, while in an online forum it could include the tags and labels of each post or even the entire post’s text. A possible advantage of content inclusion in such cases could be the formation of communities including nodes that...
not only have members exchanging messages between them but share the same context, underlining the group’s main topics. A widely discussed drawback of content inclusion in community detection methods is increased complexity [3]. From our point of view, “semantic connection” is equally important to structural, as social actors with few links between them and similar content could share the same interests, ideas, or actions. As such, it would be beneficial to include them in the same community.

In parallel, one of the most important and continuously rising topics in social networking and data mining, in general, is the search for social actors that can be differentiated from the typical actors’ behavior in a network. This procedure is referred to as outlier detection or anomaly detection. In addition to being widely studied in social networks, this has considerable enterprise value. For example, anomaly detection can identify spam posts on review systems when their content follows a different motif than real reviewers’ posts.

Focusing on social graphs, an outlier is a node that is loosely connected to the rest of the graph and in most cases not a member of any community. Outlier nodes in a social network are often treated as noise, representing, for example, users of social media who rarely interact with others, and who as such have only a few connections with the network and can be excluded from the computations. On the other hand, many interesting applications of outlier detection can be found in the domain of cybersecurity, e.g., fraud detection, as well as in other domains such as advertising. The early detection of outlying nodes can help a company to keep its customers close to its community while trying to attracting other companies’ outlying customers through marketing techniques.

During our research, we noted a lack of related works on outlier detection with a hybrid approach combining network structure and content. Instead, most outlier detection techniques [4,5], similarly to community detection ones, rely solely on structural information. In addition, while most real world applications include graphs that change over time, the temporal dimension of the problem is often neglected. Therefore, in this paper we propose a novel algorithm that leverages the dimensions of both content and structure and can perform community detection in dynamic social networks in parallel with outlier detection.

In particular, we propose a method for addressing both community and outlier detection simultaneously based on COTILES [6], which is an evolutionary community detection algorithm that leverages both structural and content information to form more cohesive communities of densely connected nodes with similar contents. More specifically, we extend COTILES so that it also identifies outlier nodes while incrementally forming overlapping communities in a streaming context, where each new interaction may update the formed communities. To identify outliers, we define a novel measure called the outlier score, which again is based on both structural and content criteria. The outlier score is dynamically evaluated for each node and captures its “outlierness”, i.e., the degree to which a node can be considered as having outlier behavior. The extended algorithm can be leveraged into assigning higher significance to either structure or content depending on the application requirements concerning both community and outlier detection. Furthermore, we argue that the outlier score can be used as an indication of a node’s future behavior, that is, the current outlier score of a node can be exploited as a feature for predicting whether the node will remain an outlier or become one in the next time instances. We evaluated our algorithms on two real datasets regarding social networks. Our experiments show that the proposed methods can be used to study both current communities and outlier nodes while predicting outlying behaviors with higher accuracy compared to other predictive features.

The rest of this paper is structured as follows. In Section 2, related work regarding both community and outlier detection is presented. Section 3 includes basic definitions and describes the original COTILES algorithm. In Section 4, we present our method for outlier detection. In Section 5, we present our experimental results on outlier detection in two real social networks, while Section 6 tests the proposed prediction capability. Finally, Section 7 consists of our conclusions and directions for future work.
2. Related Work

In this section, we discuss related works while discerning between approaches addressing community detection and outlier detection. In both categories we distinguish between static and dynamic approaches, where the social graph topology is respectively known a priori or retrieved in a stream. For community detection, we discuss only works that are similar to COTILES in that take into account both structure and content in their methodology. For outlier detection, where related research is more limited, we include structural, content-based, and hybrid methods while differentiating between them.

2.1. Community Detection

The majority of methods for community detection are exclusively structure- or content-oriented. We consider that leveraging both yields the best results in terms of cohesive community structure, and as COTILES is just such a hybrid approach, we discuss works that exploit both structural and content information. The survey in [7] proceeded via an exhaustive search and categorization of such works, dividing them based on the timing when the structure and the content of the network were examined: early fusion methods, in which connections and attributes are fused before the community detection process; simultaneous fusion methods, where this happens at the same time; and late fusion methods, where community detection takes place initially. The authors concluded with the idea that analyzing attributed networks can boost community detection in sparsely connected networks while handling missing or noisy attributes, and is in most cases more effective than analyzing networks by taking into account only one dimension. While the survey included both static and dynamic methods, it did not focus on the differences between them.

Traditional community detection takes place in static graphs, where both the nodes and the edges between them are known in advance. It is obvious that having additional information about the time at which each interaction occurs can add useful granularity to the analysis. For example, the author of a scientific paper is much more likely to be a member of a community in which they share recent co-authorship than other communities whose members they collaborated years ago with. This additional information is received from graphs, referred to as temporal graphs, which in addition to structural information (and content, in our case) include a timestamp for each interaction specifying the exact time when it occurred.

2.1.1. Static Methods

We first consider static community detection methods that ignore the time dimension. Following the categorization in [7], we start with early fusion methods, which are more popular. In [8], the authors proposed an efficient iterative algorithm for finding communities in attributed multi-graphs where nodes are connected by multiple types of edges. They first ranked each of the different types of edges based on their importance and assigned a corresponding weight, then the proposed CAMIR algorithm was assigned the number of communities as input and used to perform clustering based on k-means. Similarly, in [9] a graph augmentation method was performed prior to graph partitioning (and community detection) in order to conclude with a unified framework based on both the structure and attribute connections of the nodes. The new augmented graph preserves the structural connectivity while adding new edges between nodes with the same attributes. Afterwards, a random walk algorithm is used to optimize a distance measure based on the new graph. In the same direction, CESNA [10] is a linear-runtime community detection algorithm that outperforms its competitors in terms of accuracy and scalability and characterizes each community using relevant attributes in their nodes. The main drawback of this algorithm is that the number of communities needs to be assigned or calculated a priori. In [11], the authors merged structure and content into a multiplex network, then performed community detection on this network based on modularity. They underlined the importance of overlapping communities because of multiple relations between two nodes in a network.
Our method considers this fact as well; real-world communities require overlap between them, as social actors are often members of more than one group.

In [12], the fusion between structure and content was performed simultaneously. The authors proposed the metric of attribute-aware modularity, which they attempted to maximize at each step of their Louvain-type algorithms. In this approach, after an initial clustering, when a node is a candidate for exclusion from its community, it is only allowed to move if the modularity is increased.

Finally, a late fusion algorithm for community detection was proposed in [13]. First, on a lower level, communities are detected based on either structure or content; later, they are fused using a weighted co-association matrix-based algorithm.

2.1.2. Dynamic Methods

When incorporating time into community detection, temporal graphs can be analyzed in a static way as well as online.

In the first scenario, the usual technique is to segment the network into time-sorted sub-networks, called snapshots, in which community detection takes place independently. Then, interconnections between communities are requested from different snapshots. An example of dynamic clustering and early fusion based on snapshots is [14], which introduced the LabelRankT algorithm. This algorithm performs an initial clustering and then updates nodes only when they are significantly different from their neighbors in terms of the labels between two consecutive snapshots. The main drawback of such methods is the need for advance knowledge of the network’s evolution over time, making them unsuitable for real-time analysis. Moreover, determining the optimal snapshot size selection is challenging, as different inputs can result in quite different structures.

Our work targets online networks with temporal information, where little related research exists. With regard to early fusion methods, in [15] links between social actors were formed based on the use of common keywords, then community and clique detection was applied on the resulting network, where detected overlapping communities could represent important events in real time. Similarly, in [16] an interest social network model was created with links referring to structural connections and similar interests of entities, based on which modularity-driven clustering was performed in a dynamic network to detect overlapping communities.

Using simultaneous fusion, in [17] a real-world social network incorporating content was used to identify interesting patterns in data which could be used to detect overlapping communities as time evolves. Finally, COTILES, introduced in [6], is an online community detection algorithm based on the TILES algorithm [18], which is extended to form overlapping communities based on both structural and attribute criteria.

Table 1 includes a summary of the discussed works on community detection, comparing them based on (i) the fusion method for content and structure they select, i.e., early, simultaneous, or late; (ii) whether they detect overlapping or non-overlapping communities; and (iii) for the dynamic methods, whether they rely on snapshots or are online.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Fusion Method</th>
<th>Communities</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8–10]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[11]</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>[12]</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[13]</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[14]</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>[15]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[16]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COTILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
2.2. Outlier Detection

The most widespread definition of an outlier comes from Hawkins [19], according to whom an outlier “is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism”. Outliers are often treated as noise, and the techniques for detecting them are typically characterized as data cleaning and pruning, as in [19,20]. These works deal with outliers as noisy nodes that incommode the clustering procedure, meaning that their early detection and exclusion from the network can reduce the procedure’s processing time and spare valuable memory resources, especially in online environments. In our work, we view outliers as nodes with potentially interesting behavior that should be better investigated; thus, we aim to combine outlier detection with a community detection algorithm in a temporal environment. As such, we highlight other related research that focuses on investigating outlying behaviors.

2.2.1. Static Methods

With regard to structural static methods, OutRank [21] is an efficient stochastic graph-based outlier detection algorithm. In contrast to our work and other proposed methods, it considers weighted graphs, with each node’s weight corresponding to its similarity with other nodes. Using random walks, the algorithm first finds central nodes, then detects outliers based on their similarity with these nodes. OutRank can detect micro-clusters of outliers as well. The authors of [22] achieved significant speedup in their outlier detection technique by introducing permanence as a node-based metric and focusing only on specific groups of nodes that are more likely to be outliers. In a recent paper [23], the fluctuation of a node was introduced as the basis of an outlier factor which, similarly to our outlier score, shows how much a node differentiates from its neighbors. In another static structural outlier detection method, in [24] the authors developed a model to perform anomaly detection in which anomalous edges are identified as those deviating from regular patterns determined by community structure. The model outputs labels for each node, defining regular nodes as legitimate and anomalous nodes as outliers. Similarly to our method, outlier detection is performed simultaneously with community detection, and anomalous edges are not necessarily assessed as undesirable interactions or malicious activity.

Regarding content-based outlier detection techniques, Radar [25] is a framework for anomaly detection in networks containing content information. Radar starts from the networks’ attributes and detects anomalies based on the residual errors of the instances while trying to optimize the objective function. As the authors mention in their conclusion, however, this method is not yet suitable for online attributed networks. The context of outliers was discussed in [26], where local outliers were based only on a single attribute or a selection of attributes. Similarly to our work, outliers were ranked based on the degree of deviation. However, the considered attributes were only numeric, while we focus on labels generated by the nodes’ attributes. Moreover, instead of selecting a set of interesting attributes, we consider a node’s outlierness to be influenced by all its labels and its structural properties, allowing for leveraging between the two.

Finally, considering hybrid static methods, the authors of [27] grouped outliers into two basic categories, namely, structural and contextual. Structural outliers were defined based on their connections, while contextual outliers were those with attributes differing significantly from their neighboring nodes. The authors benchmarked different algorithms in order to assess how the different detection methods responded to the two types of outliers in static graphs. Non-overlapping groups were required during the community formation procedure. In [28], the authors proposed a method for outlier detection in multi-view systems; initially, they categorized outliers as class outliers, equivalent to to structure-based outliers, and attribute outliers, equivalent to content-based ones. They additionally proposed a mixed type of outliers, class–attribute outliers. In our approach, instead of separating outliers into structural- and content-based groups, we consider both criteria at the same time, thereby creating a more complete view of a node’s behavior. The use of the weighting factor between structural and content criteria enables us to focus
solely on content or structure as required, or to assign more importance to one over the other as necessary. Finally, in [29], anomaly detection was performed in social graphs based on anomaly scoring, which consisted of a structure-based score and an attributed-based score, quite similar to the outlier score that we propose. First, modularity-based community detection was performed; then, in a second phase, each node was ranked with an anomaly score. The main difference with our work is the static nature of [29], which makes it unsuitable for outlier detection in streaming environments. Moreover, we perform community detection and outlier score evaluation simultaneously, yielding a more efficient approach.

### 2.2.2. Dynamic Methods

The survey in [30] provides an overall view of the field, categorizing outlier detection methods for time series, stream data, spatio-temporal data, and temporal networks. The authors observed a lack of methods for online outlier detection, as most window-based methods are offline methods.

Starting with structural methods, in [31] the authors categorized outliers into global and local outliers, which are respectively differentiated from the network or from their neighbors. Focusing on local outliers, they proposed a corresponding streaming method. In [32], the authors considered a snapshot-based approach for modeling network evolution and proposed the notion of community trend outliers, which evolve in a highly different way compared to other community members. Their model extracts soft patterns from the community evolution procedure and then marks as outliers those nodes that deviate from their best-matching soft pattern in more than one snapshot. Similarly, [33] proposed evolutionary community outliers, or ECOutliers, focusing more on their temporal dimension and how these nodes evolve differently with time. The challenging problem of community matching in a two-snapshot setting was tackled by an objective function that weights the lower contributions made by outliers.

Continuing with content-based methods, in [34] the authors used an attributed graph with a set of attributes associated with the nodes to determine communities in which members achieve highly topical activity. In this way, they introduced the notion of temporal activeness, inspired by the fact that many nodes may be inactive during several steps of the temporal evolution, as a way to decrease the number of low-activity users in a community by considering a slightly different definition for outliers. In [35], the authors proposed a framework applied to Twitter data to detect sudden rises of emergency topics such as the COVID-19 pandemic. Their method uses k-means to generate a cluster of unusual tweets; however, preprocessing and topic modeling are needed in advance, and the method is not yet suitable for online networks.

Finally, regarding outlier detection performed on graphs that consider both structure and content, we found a serious lack of related works. This is confirmed by the authors of [36], an exhaustive review of papers corresponding to outlier detection in attributed networks. The authors of this review, much as we do here, distinguished anomaly detection techniques into static and dynamic; each one of these categories was further separated into attributed and unattributed categories. However, although they examined many of state-of-the-art papers, they quoted only static methods. Regarding bipartite graphs, in [37] the authors found groups of possible outliers after tracking the history of employment and disclosures in a real dataset taking the form of a bipartite graph of individual entities and sequentially ordered attributes. The authors searched for groups of outliers, or ‘tribes’ as they refer to them, which are groups of individuals who share unusual sequences of affiliations. They defined a significance score for each edge, to measure the significance or the anomalousness of shared jobs. Similarly, rather than simply classifying the nodes into outliers, we provide a score that quantifies the outlierness of each node. However, we evaluate each node based on its outlierness, not on the basis of each interaction.

Table 2 summarizes the works discussed above and compares them concerning: (i) the dimensions they consider, i.e., content and structure; (ii) whether they assign an
Table 2. Outlier detection techniques.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Dimensions</th>
<th>Outlierness</th>
<th>Information</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>[21]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[22,23]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[24]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[25]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[26]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[27,28]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[29]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[30]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[31]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[32,33]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[34]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[35]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[36]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ext. COTILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

3. Background

A social graph can be modeled as an undirected graph $G = (V, E)$. Each node $u \in V$ of the graph corresponds to a user of the social network. Between these nodes, there are edges $(u, v) \in E$ that correspond to interactions between users of the network. To model time evolution, edges in the graph require a timestamp $t_{(u,v)}$ that represents the time point when the interaction occurs. Furthermore, to incorporate content, graphs are attributed such that for each edge $(u, v)$ we can extract its textual attributes in the form of labels. This results in a labeled set $L_{(u,v)}$ that corresponds to the content of the interaction, which usually is a set of tags or text exchanged between nodes $u$ and $v$. Thus, based on the definition in [38], we have the following definition.

**Definition 1 (Temporal-Labeled Graph).** A temporal-labeled graph on $G$ is an attributed graph $G(T) = (V, E, L, T)$ from which the attributes of time and content have been extracted; $T = \{T_e \subseteq \mathbb{N} : e \in E \}$ refers to the set of timestamps of the edges of $G$, while $L$ refers to the global set of labels that appear in $G$.

**Community Detection with COTILES**

The widespread definition referring to communities in social graphs describes them as sub-graphs with higher connectivity between their members than with nodes outside of the community. COTILES [6] is an online community detection algorithm implemented to be applied in temporal-labeled social networks that, in addition to the structural information on which most community detection algorithms focus, takes content into account in order to form more thematically cohesive communities. It additionally considers interaction expiration and supports overlapping communities. The advantages of COTILES are that, in contrast to many other community detection algorithms, it does not require the number of communities to be provided as input; it allows overlapping communities; it is, as mentioned before, online; and it applies an edge aging factor, meaning that edges and labels can be removed from the network if they stay inactive for a predefined period.

The algorithm identifies two roles of nodes participating in communities: core and peripheral nodes. A community is formed when at least three nodes have connections
between them which are valid in terms of both structure and content. A node is characterized as core if it is involved in at least one triangle with other nodes in the same community. A node is characterized as peripheral if it is a one-hop neighbor of a core node. A community may consist of both of these node categories. Core nodes are the main community representatives that spread community membership during role propagation; this is the main procedure of the algorithm and distributes community membership to nodes based on their neighborhoods. Community membership is propagated by core nodes of each community as they add their neighbors to the community’s periphery. Community members turn into core nodes when they form at least one triangle with in-community nodes, and can spread community membership to their new connections by adding them to the community’s periphery.

The basic characteristic of COTILES is that, in addition to the structural procedure we have described thus far, it exploits the content of the network in the form of labels. To achieve this, the algorithm intervenes in community formation procedures to ensure that a node can become a community member only if its content is similar to the content of that specific community. To describe the content of a community, the notion of the Community Label Set $L_c$ is introduced, which represents the set of labels that accompany the community’s nodes at that time.

**Definition 2 (Community Label Set).** Given a community $c$ at time point $t$, $L_c$ represents the set of labels which at that time describe the contents of $c$.

In particular, the community label set is formed by accumulating the labels of each edge leading to a node entering a community. When an edge connects a node outside the community with a community member, the Community Label Set is compared with the Edge Label Set; if they match, the node can enter the community with which it already has structural connections. In the same way, each node inherits the content of the interactions in which it takes part in the form of Node Labelset $L_u$. The LabelConstraint needs to be validated before a node enters a community’s periphery. This ensures that the intersection of the label sets of the community and the candidate node is not empty, while the user can set a context-dependent threshold to restrict the similarity terms.

The importance assigned to each of the characteristics (structure and content) can be leveraged through the proposed alpha $\alpha$ weight, as different $\alpha$ values lead to community detection being more focused on structural or content relations.

As referred to previously, COTILES, which we further extend in this paper, is an incremental algorithm for analyzing the topology of social networks in an online way. The algorithm incorporates a time-to-live (ttl) parameter to reflect decaying social relationships in evolving graphs. Each interaction has a limited lifetime specified by the ttl input and disappears from the network unless it appears again; thus, the ttl needs to be updated. The contents of each interaction inherit its lifetime to ensure that the communities’ content evolves and does not remain restricted to its early topics. When the time-to-live of an edge expires and it has not been updated, the edge is deleted from the graph along with its corresponding labels. The labels are deleted from the corresponding Node, Edge, and Community Label Sets only if they were not updated either. It is possible that the same edge could appear however with different content, meaning that only its structural part would update its time-to-live. Furthermore, a Node or Community Label Set could accumulate the same labels from another edge, meaning that they are not necessarily deleted.

The evolution of the network is tracked through a predefined observation window time ($obs$), with the updates on the network occurring on its end, as the incoming edges are in the form of a stream. Whether the deletion of an edge leads to a community disconnection is checked at the end of each observation window time when the update of the graph is performed.

The main steps of the algorithm are as follows:
1. Preprocessing, where labels are extracted from the attributed graph.
2. For each incoming edge, corresponding timestamps and label sets are appropriately updated, then the edge is examined.
3. If the edge leads a node into a community’s periphery, the node’s content is checked; if it matches the content of the community, then it is inserted into the community and its labels are inserted into the Community Label Set.
4. At the end of every observation window time, the graph, communities, and label sets are updated.

In summary, COTILES is a community detection algorithm requiring an attributed social graph with temporal and content information as input, resulting in communities formed by nodes based on their structural and content relations. Intermediate formations can be tracked through the configurable observation window and a node lifetime is set to ensure that older nodes and their attributes lose significance over time.

4. COTILES for Outlier Detection

We propose an algorithm based on COTILES with additional functionality to assign an outlier score to each node, thereby aiding in the study of outlying behaviors of social actors and the prediction of such behaviors in the future. In this section, we define the outlier score and describe the extended COTILES algorithm in detail.

4.1. Outlier Score

To allow COTILES to detect community outliers, it is extended to compute an outlier score of each node. Nodes inherit their edges’ labels, and can be considered as outliers or “inliers” based on their outlier score. The main idea for a node to be assigned with a high outlier score is based on the widely accepted definition of outliers in [19], which refers to them as nodes that deviate from neighboring observations. However, here we check their deviation in terms of both structure and content. In the algorithm we propose, each node is assigned an outlier score that is then updated, with every interaction (new edge) affecting the corresponding node.

In order to take into account both the structure and content of the network, we propose a score consisting of two context-dependent weighted parts. Thus, the outlier score consists of two parts that can be equally or unequally weighted, assigning the same or different amount of importance to the nodes’ structure and content. The structure part of the outlier score refers to whether the node is highly connected to one or more communities, while the content part refers to the popularity of the node’s labels.

Regarding structure, nodes that share few edges with community members while having many connections with nodes outside the community are more likely to leave the community during the evolution process, as their connectivity can easily decay. To this end, we define Community Focus as a metric indicating the amount of connectivity that a node has with its most adjacent community.

\[ \text{Community Focus} = \forall C \in G, \max \sum \frac{1}{|C|} \text{intra-edge} \sum \frac{1}{|C|} \text{total-edge} \]

For each valid community in the network, the algorithm computes the node’s community focus index irrespective of whether or not it is a member of a group. For example, when a node’s edges are all adjacent to one community, the structure part equals 1. In contrast, when a node has its edges scattered between members of different communities or nodes not participating in any community, the structural part of its score tends to 0. Thus, community focus can be viewed as a node’s degree of membership with respect to its adjacent communities.

**Definition 3 (Community Focus).** We define Community Focus as the number of intra-community edges between a node and the community it is most connected to in comparison to the total number of edges in the network.

\[ \text{Community Focus}_u = \forall C \in G, \max \sum \frac{1}{|C|} \text{intra-edge} \sum \frac{1}{|C|} \text{total-edge} \]
On the other hand, to quantify the content part of the outlier score we define the Label Set Match.

**Definition 4 (Label Set Match).** We define LabelSetMatch as the fraction of the intersection of the label sets of a node and its most connected community in comparison to the number of that community’s distinct labels.

\[
\text{LabelSetMatch}_u = \forall C \in G, \max \left( \frac{|\text{Labelset}_u \cap |\text{Labelset}_C|}{|\text{Labelset}_C|} \right)
\]

Label Set Match stands for the maximum match index between the label sets of the node and each community, and indicates the degree to which a node’s content is significant for a community. For example, when most members of the community share the same labels with the node, the match index is higher, tending to 1; otherwise, it tends to 0, indicating low content similarity. Note that this index is similar to the Jaccard index of the two label sets, however, only the Community Label Set is kept in the denominator. In this way, central nodes with rich label sets are not penalized with higher outlier scores under the condition of having significant overlap with the corresponding communities’ label sets.

Our goal is to compose a metric of outlierness for each node using both its structure and content. As we have defined Community Focus and Label Set Match as metrics for structural and content outlierness, respectively, we now combine them to obtain a node’s outlier score.

**Definition 5 (Outlier Score).** The outlier score of a node is inversely proportional to the node’s Community Focus and Label Set Match.

\[
OS_u = 1 - \left[ (w \cdot \text{CommunityFocus}_u) + (1 - w) \cdot \text{LabelSetMatch}_u \right]
\]

We use the inverse because our score is a metric of outlierness for a node; thus, a high outlier score would refer to possible outliers with low Community Focus and Label Set Match. In addition, as \(w \in [0, 1]\) is a tunable weight, its value can leverage the attention paid to outlying structural or content behavior. Setting high values for \(w\) would result in a more structure-based outlier view, meaning that a node with increased Community Focus would result in lower outlier scores even if its content were quite different than most of its neighbors. However, our approach makes it possible for a node to achieve a low outlier score even if it is not a member of a community as long as its content has high similarity with a community’s content. Table 3 summarizes the most important notation we use in our algorithms.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L_{(u,v)})</td>
<td>Edge labelset</td>
</tr>
<tr>
<td>(L_u)</td>
<td>Node labelset</td>
</tr>
<tr>
<td>(L_C)</td>
<td>Community labelset</td>
</tr>
<tr>
<td>(a)</td>
<td>(\alpha) weight</td>
</tr>
<tr>
<td>(w)</td>
<td>(w) weight</td>
</tr>
<tr>
<td>(OS_u)</td>
<td>Node outlier score</td>
</tr>
<tr>
<td>(\text{thres}_{OS})</td>
<td>Outlier score threshold</td>
</tr>
</tbody>
</table>

### 4.2. Extending COTILES

We extend COTILES to evaluate the nodes’ outlier scores while at the same time performing community detection. The algorithm receives a streaming source of temporally labeled edges as input, then performs clustering, leading to community detection, while the Label Set Match of each node computed in tandem with the Community Focus derived from the extracted communities leads to an outlier score for each node. At the end of each observation window, the current state of the community and outliers structure is extracted. Based on this, outliers are detected and assumptions can be made about the nodes that are most likely to play a secondary role in the network in the future.
Algorithm 1 details the extended COTILES procedure. Initially, it starts the observation window time and extracts the labels from the network’s attributes (lines 1–2). Then, the algorithm retrieves the next interaction of the graph, \((u, v)\), consisting of two end nodes \(u\) and \(v\), its set of labels \(L_{u,v}\), and its timestamp \(t_k\) (line 3). The timestamps of both edges and labels are updated or initialized based on whether or not these edges or labels already exist in the graph (line 4). In parallel, expired labels and edges are removed from the graph (line 5). Immediately afterwards, every new edge retrieved from the stream is added to the graph if it was not already part of it (line 7).

Proceeding to the main functionality of COTILES for outlier detection in evolving communities, we distinguish four cases based on the neighborhoods of the two nodes \(u\) and \(v\), where we denote the set of neighbors of \(u\) as \(\Gamma(u)\).

1. If the nodes of the new edge have only one neighboring node, this means that they have no other connections, and cannot be members of any community yet; thus, the algorithm does not take any actions in terms of community detection. Only the outlier scores of the nodes are computed (lines 8–11). These outlier scores will be high as concerns their structure, as Community Focus is 0, although their Label Set Match could balance their outlierness.

2. Next, the algorithm checks whether each of nodes \(u\) and \(v\) belong to any community core (lines 12–13). Because peripheral nodes are not allowed to propagate community membership, no action is performed if neither node is core (line 14).

3. If one of the nodes is a core node of a community with a neighborhood greater than 0 and the other node is appearing for the first time, then the core node spreads its community membership to its neighbors through peripheral propagation, which includes checking the constraint for content similarity before adding a node to a community. At the same time, the outlier score of this node and its neighborhood is re-evaluated, as the updating of the community structure affects all of them (lines 16–25).

4. The final case is when both nodes \(u\) and \(v\) are existing core nodes in \(G\) (lines 26–46). Then, the common neighbors of the two core nodes are computed (line 27); based on this, two more scenarios are possible:

(a) If nodes \(u\) and \(v\) do not have common neighbors, peripheral propagation takes place, as in the previous case (lines 28–30).

(b) If \(u\) and \(v\) have common neighbors, core propagation takes place. For each common neighbor of the nodes, if it is not a member of any same community, a new community is formed (lines 33–37); otherwise, for each pair of these three nodes, if they are members of the same communities, then they propagate the community membership to the third node (lines 38–46).

The outlier score of each node and its neighbors are immediately computed (lines 47–52).

At the end, the algorithm checks whether the observation window threshold is reached; if it is, then the detected communities and outliers are returned (lines 53–56). The output consists of the community structure based on returning present communities, their members, and their computed label set and the outlier scores, on which basis the user can set appropriate thresholds to characterize each node as an outlier or not. Further information is extracted, such as merged or split communities, though it is not necessary for outlier detection and can be excluded for efficiency purposes. The flowchart of Figure 1 shows an overall view of the algorithm’s steps.
Algorithm 1 COTILES for Outlier Detection in Evolving Communities.

Require: \( G \) : undirected attributed graph, \( \tau \) : temporal observation window, \( t_{tl} \) : time to live, \( L \) : set of available labels, \( w \) : outlier score weight.
Ensure: \( OS_n \) : outlier score of node \( n \)

1: \( actual_t = 0 \)
2: \( G.extract(L) \)
3: \( (u, v, l_u, t_u) = newInteraction \)
4: \( G.updateTimestamps() \)
5: \( G.removeExpired() \)
6: if \( (u, v) \notin G \) then
7: \( G.addEdge(u, v) \)
8: if \( |\Gamma(u)| = 1 \) and \( |\Gamma(v)| = 1 \) then
9: \( \text{ComputeOutlierScore}(u, w) \)
10: \( \text{ComputeOutlierScore}(v, w) \)
11: \( \text{Continue} \)
12: \( \text{core}_u = G.getCommunityCore(u) \)
13: \( \text{core}_v = G.getCommunityCore(v) \)
14: if \( \text{core}_u == \emptyset \) and \( \text{core}_v == \emptyset \) then
15: \( \text{Continue} \)
16: if \( |\Gamma(u)| > \emptyset \) and \( |\Gamma(v)| == \emptyset \) then
17: \( \text{PeripheralPropagation}(u, \{v\}, t_u) \)
18: for \( i \in \Gamma(u) > 1 \) do
19: \( \text{ComputeOutlierScore}(i, w) \)
20: \( \text{ComputeOutlierScore}(v, w) \)
21: if \( |\Gamma(u)| == \emptyset \) and \( |\Gamma(v)| > \emptyset \) then
22: \( \text{PeripheralPropagation}(v, \{u\}, t_v) \)
23: for \( i \in \Gamma(v) > 1 \) do
24: \( \text{ComputeOutlierScore}(j, w) \)
25: \( \text{ComputeOutlierScore}(v, w) \)
26: \( \text{else} \)
27: \( \text{CN} = \Gamma(u) \cap \Gamma(v) \) \( \triangleright \) \( \text{CN} : \text{Common Neighbors} \)
28: if \( |\text{CN}| == 0 \) then
29: \( \text{PeripheralPropagation}(u, \{v\}, t_u) \)
30: \( \text{PeripheralPropagation}(v, \{u\}, t_v) \)
31: \( \text{else} \)
32: for \( z \in \text{CN} \) \( \triangleright \) \( \text{Core Propagation} \)
33: if \( (u, v, z \notin \text{CommonCommunities}) \) then
34: \( \text{G.createCommunity}(u, v, z, Labels_{(u,v,z)}) \)
35: \( \text{PeripheralPropagation}(u, \Gamma(u), t_u) \)
36: \( \text{PeripheralPropagation}(v, \Gamma(v), t_v) \)
37: \( \text{PeripheralPropagation}(z, \Gamma(z), t_z) \)
38: if \( (u, u, \text{GetCommunities}() \cap v, \text{GetCommunities}()) > 0 \) then
39: \( \text{G.AddToCommunityCore}(u, u \cap v, \text{GetCommunities}()) \)
40: \( \text{PeripheralPropagation}(z, \Gamma(z), t_z) \)
41: if \( (u, u, \text{GetCommunities}() \cap z, \text{GetCommunities}()) > 0 \) then
42: \( \text{G.AddToCommunityCore}(v, u \cap z, \text{GetCommunities}()) \)
43: \( \text{PeripheralPropagation}(z, \Gamma(z), t_z) \)
44: if \( (z, z, \text{GetCommunities}() \cap v, \text{GetCommunities}()) > 0 \) then
45: \( \text{G.AddToCommunityCore}(u, z \cap v, \text{GetCommunities}()) \)
46: \( \text{PeripheralPropagation}(v, \Gamma(v), t_v) \)
47: for \( i \in \Gamma(u) > 1 \) do
48: \( \text{ComputeOutlierScore}(i, w) \)
49: \( \text{ComputeOutlierScore}(u, w) \)
50: for \( i \in \Gamma(v) > 1 \) do
51: \( \text{ComputeOutlierScore}(j, w) \)
52: \( \text{ComputeOutlierScore}(v, w) \)
53: if \( t - actual_t == \tau \) then
54: \( \text{OutputCommunities}(G) \)
55: \( \text{OutputOutlierScores}(G) \)
56: \( t = actual_t \)

To provide more insight into the main procedures of community detection in COTILES, we present the algorithm for peripheral propagation (Algorithm 2), which adds nodes to communities’ peripheries after validating that their content matches the corresponding community’s content. This algorithm takes as input the corresponding node along with the
set of nodes to which it propagates the community periphery. A timestamp $t_k$ indicating the exact execution time is passed along as well.

**Algorithm 2 Peripheral Propagation** $(u, \text{nodes}, t_k)$

**Input:** $u$: node of $G$, $\text{nodes}$: a set of nodes, $t_k$: timestamp.

1. for $v \in \text{nodes}$ do
2.   for $z \in G$.GetCommunityCore($u$) do
3.     for $C_o \in z$.GetCommunities() do
4.       if LabelConstraint($u, v, C_o, G$) == true then
5.         $G$.AddToCommunityPeriphery($v, C_o$)

![Figure 1. High-level flowchart of COTILES for outlier detection.](image)

Finally, Algorithm 3 refers to the $\text{ComputeOutlierScore}$ function is applied within the extended COTILES. It takes a node $u$ along with its $w$ weight as inputs. Recall that $w \in [0, 1]$ is a tunable weight allowing significance to be leveraged between content and structure for the calculation of outlier scores. For every active community the Community Focus of the node is computed as the maximum connectivity achieved by the node at the specific time when the function is called (lines 4–6). Specifically, for each community $C$, $\text{Focus}$ is calculated as the fraction of the number of edges of $u$ with members of $C$ with respect to the number of all edges of $u$; its maximum overall community value is selected. Immediately afterwards (lines 7–9), the Label Set Match is computed. The main component of this sub-score referring to content is the maximum $\text{Match}$ of the node with respect to the communities, that is, the maximum matching between the content of the node and the content of each community. This calculation is performed in line 7, where the Match of node $u$ with community $C$ is the fraction of the number of common labels of $u$ and $C$ with respect to the total number of labels included in $C$. Finally, the outlier score is computed based on the Community Focus and Label Set Match, with the weight parameterized by $w$ (line 10).
In addition to community detection, the algorithm we propose assigns an outlier score to every active node. The threshold with which nodes can be characterized as inliers or outliers can be set in different context-dependent ways. For instance, we may consider the nodes with scores above the average outlier score to be outliers. The main advantage of our method is the ability to simultaneously detect communities and outliers in an efficient way, as the outlier score is computed for nodes that are affected by each interaction instead of overall. Another advantage is that we measure the outlierness of each node, allowing each node to be directly compared to others. Thus, while two nodes with scores above the average would be outliers, the node with the higher score would be more of an outlier.

\begin{algorithm}
\caption{Compute Outlier Score \((u, w)\)}
\begin{algorithmic}[1]
\Input \(u\): node of graph \(G\), \(w\): outlier score weight
\Output \(OS_u\): Outlier Score of node \(u\)
\State \(CommunityFocus_u = 0\)
\State \(LabelSetMatch_u = 0\)
\For {\(C\) in \(G.\)Communities}
\State \(Focus_u = \frac{\text{intraEdges}(C)}{\text{allEdges}(C)}\)
\If {\(Focus_u > CommunityFocus_u\)}
\State \(CommunityFocus_u \leftarrow Focus_u\) 
\Comment Structure Score
\EndIf
\State \(Match_u(C) = |\text{Labelset}_u \cap \text{Labelset}_C|\)
\If {\(Match_u(C) > LabelSetMatch_u\)}
\State \(LabelSetMatch_u \leftarrow Match_u(C)\)
\Comment Content Score
\EndIf
\EndFor
\State \(OS_u = 1 - [w \cdot CommunityFocus_u + (1 - w) \cdot LabelSetMatch_u]\)
\Comment Outlier Score
\State \Return \(OS_u\)
\end{algorithmic}
\end{algorithm}

4.3. Prediction

In the final stage of our proposed framework, prediction for each node outlier behavior takes place based on each node’s outlier score. We argue that our proposed outlier score indicates the outlierness of each node in terms of structure and content, and as such can provide a valid indication of the nodes’ fluctuations in future connectivity. In particular, we argue that a high outlier score at a current time point describes a node that is more likely to become a community outlier or even be eliminated from the network at subsequent time points.

Thus, we model the problem of predicting outlier behavior as a classification problem and propose using the nodes’ outlier score as a predictive feature. A reasonable outcome of our predictions would be a high proportion of nodes that become outliers in later time instances if their outlier score at a specific time point is high.

A node, even one that is a member of a community, might reach a high outlier score because of a possible low connection with the community members (community focus) or low similarity with the community’s content. In such cases, our method leads to predicting outlying behavior of the node in the subsequent time slices, not necessarily the immediate next one, thereby detecting the outlying tendencies of nodes even earlier.

In addition, the community and outlier detection process in our framework takes place in an online way without taking into account previous stages or snapshots of the network. For prediction purposes, however, the recent history of the node’s outlierness can be used in the form of chronological chains which demonstrate a node’s evolution over time. A chronological chain can be a series of a specific node’s outlier scores in a few consecutive observation window points. If a node has a history of increasing outlier scores, it is more likely that it will become an outlier, leaving all communities or even the network in the future; on the other hand, a history of decreasing outlier scores, even those that are not members of any community, could indicate nodes that are gradually becoming more embedded in the network’s communities and are less likely to leave.

5. Evaluation Results

To study the effectiveness of the proposed approach in detecting outliers, we used two real datasets. First, we describe the preprocessing procedures used to collect, clean, and
model the data to match the problem constraints. Then, we describe several experiments performed to examine the impact of each different setup or parameter on the outlier detection procedure, concluding with a set of parameter values for lifetime \((\text{ttl})\), observation window time-span \((\text{obs})\), and weight \((w)\). Finally, we study the performance of our approach in detecting outliers and explore the resulting outlier scores for the nodes of the two datasets. The implemented code used for the experiments is publicly available [39].

5.1. Datasets

We used two different datasets in order to better evaluate our results. The first dataset StackExchange, referring to Unix.StackExchange.com, was retrieved from a forum available via the Stack Exchange Network [40], while the second, MovieLens, consisted of rating and tagging activity from MovieLens, a movie recommendation service [41]. We constructed a temporally labeled graph from each dataset.

In StackExchange, nodes correspond to forum users and edges represent interactions between users that occur when members answer or comment on each other’s posts. The moment at which each interaction occurs is indicated by the timestamp \(T\), and the labels of the post are included in \(L\). Label selection was performed by the forum users by choosing one to five labels from a list. The constructed graph from StackExchange has 542,120 edges formed between 87,438 nodes; 2615 different labels appear throughout the graph’s lifetime, which covers the ten years from July 2010 to June 2019.

MovieLens was further processed to have its labels extracted in a proper form. In the second dataset, an edge is created when users provide tags for the same film. In contrast to the first dataset, users are free to write their labels in free text form. Similar to StackExchange, \(L\) stands for the set of labels MovieLens users refer to, while \(T\) stands for the set of timestamps of the interactions. We deleted edges without labels, as they did not match our content-related approach, and applied a stemming technique to \(L\) in order to match words with the same root and remove stopwords. Then, we proceeded to keep only the most used labels, avoiding those with less than 50 appearances in the network. After the processing, the second dataset consisted of 111,621 edges, 6113 nodes, and 1043 different labels. The period covered by the dataset started from January 2006 and lasted until November 2019, lasting 14 years.

Table 4 shows an overall depiction of the two datasets’ characteristics. The two datasets in their final form are available online [42].

Table 4. Dataset information.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Edges</th>
<th>Nodes</th>
<th>Labels</th>
<th>Timespan (Years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Exchange</td>
<td>542,120</td>
<td>87,438</td>
<td>2615</td>
<td>10</td>
</tr>
<tr>
<td>MovieLens</td>
<td>111,621</td>
<td>6113</td>
<td>1043</td>
<td>14</td>
</tr>
</tbody>
</table>

5.2. Parameter Tuning

Two of the main inputs in the algorithm are the \(\text{obs}\) and \(\text{ttl}\) values, respectively standing for the observation window period and time-to-live of each edge and its labels. Both parameters are context-dependent.

The flow of the algorithm is not affected by differentiating the observation window period; however, the user will be aware of more or less information regarding the graph’s formations over time, that is, \(\text{obs}\) determines the granularity of the evolution monitoring. Higher \(\text{obs}\) values result in fewer algorithm outputs, and consequently in a speedup in execution time, while intermediate changes describing the graph’s evolution are not recorded.

Changing the \(\text{ttl}\) value, on the other hand, results in quite different behavior and intermediate outcomes, as certain connections may appear and disappear during the same observation window period if their lifetime is set too low. The \(\text{ttl}\) can be set to \(+\infty\) in an accumulation growth scenario; however, this results in challenges around memory.
and computational costs. In general, high \(\text{ttl}\) values are appropriate for networks describing more permanent relationships between the participating entities, such as friendships in social networks, while low \(\text{ttl}\) values better fit networks with transient interactions, e.g., communication networks.

Both \(\text{obs}\) and \(\text{ttl}\) are related, and setting high values of both is appropriate when the network is more stable and the analyst is not interested in monitoring all fluctuations, while low values enable more changes to be captured. In order to study these two parameters appropriately and configure them for our two datasets, we experimented with various pairs of values to observe the difference they would depict in the results. The time spans of the two datasets are quite similar, as shown in Table 4; thus, we experimented with similar value pairs.

Table 5 shows the total number of detected communities along with the average number of community members and labels when different value pairs were selected. In order to record all edges at least one time, we only studied pairs where \(\text{ttl}\) was higher than \(\text{obs}\). Observing the table, the differences between the two datasets are obvious; StackExchange tends to form more communities but is smaller in terms of cardinality with less varying content, while MovieLens users tend to form a smaller number of larger groups. We selected the 60/30 pair of \(\text{ttl}/\text{obs}\) parameters for our experiments, as it derives a higher number of communities with more members for both datasets compared to pairs with lower \(\text{ttl}\), while setting \(\text{ttl}\) higher leads to insignificant differences.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>(\text{ttl} / \text{obs})</th>
<th>#Coms</th>
<th>Mean Members</th>
<th>Median Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Exchange</td>
<td>30/15</td>
<td>4118</td>
<td>5.80</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>30/30</td>
<td>2848</td>
<td>6.10</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>60/30</td>
<td>5225</td>
<td>9.25</td>
<td>8</td>
</tr>
<tr>
<td>Movie Lens</td>
<td>30/30</td>
<td>294</td>
<td>13.45</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>60/30</td>
<td>631</td>
<td>25.21</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>120/60</td>
<td>656</td>
<td>32.55</td>
<td>26</td>
</tr>
</tbody>
</table>

Weight Value \((w)\)

The weight value \((w)\) is responsible for balancing the structure and content part of the outlier score. If it is set to 0, outliers are characterized only by their content score, while when it is set to 1 only the structural score of each node matters. Thus, the weight value can help in obtaining a context-dependent threshold reflecting the relative importance that the user wishes to assign to structure or content.

Figure 2 depicts the distribution of outlier scores among the two datasets’ nodes for different values of \(w\). The green line refers to the percentage of nodes with an outlier score over 0.50 (High OS), while the blue line refers to the percentage of nodes with an outlier score below or equal to 0.50 (Low OS). When the weight is balanced between structure and content \((w = \frac{1}{2})\), the detection of nodes with outlier scores higher than 0.50 is relatively low for StackExchange, at 45.95%, and particularly low for MovieLens at 24.29%. For a lower weight of \(w = \frac{1}{4}\), the percentage of nodes with high outlier scores increases in both datasets. A low \(w\) yields higher importance of the nodes’ content; the increase in possible outliers can be explained by the fact that even nodes with strong structural connections may achieve high outlier scores if their content differs from their neighbors. In contrast, a higher weight value of \(w = \frac{2}{3}\) means that the outlierness criterion is closer to plain structural outlier detection. In this latter case, the outlier scores tend to be lower in both datasets, falling closer to the case where content and structure are equally weighted.
Comparing the two datasets, it can be seen that the effect of $w$ is much stronger for the MovieLens dataset, showing that while the nodes in this dataset are densely connected, they have more diverse content compared to StackExchange. For the rest of our experiments, we set $w = \frac{1}{2}$ to assign equal importance to both structure and content.

5.3. Outlier Score Distribution

In this set of experiments, we tried to verify that a node to which we assigned a high outlier score would be a member of hardly any communities. We used the traditional definition of an outlier as not belonging to any community and investigated whether our outlier score reflects this. The outlier score we propose is a metric of outlierness, or how much of an outlier a node is; it ranges from 0 for totally central nodes to to 1 for theoretically total outliers, respectively. The intuitive decisive threshold for outlier assignment lies at the halfway mark of this range; for example, a node with an outlier score less than 0.5 is closer to totally central than to a total outlier. This can be adjusted based on the context.

Thus, to confirm the validity of the outlier score, we studied the distribution of outlier scores among nodes in total (Figure 3) and then separately for nodes that were members of at least one community (Figure 4) and for nodes that did not belong to any communities (Figure 5). Our results confirm that, based on the traditional definition, most outliers have high outlier scores and vice versa. Thus, we can conclude that our proposed outlier score was able to determine the community membership of each node based on its connectivity and its content.
For StackExchange, Figure 3 (left) shows that most outlier scores are gathered around values near 0.50 or higher (81.30% of nodes have outlier scores over 0.40). This reflects a graph in which most nodes tend to be outliers, a usual phenomenon in forum networks, where a few users share the most interactions and the vast majority uses the interface only to ask a specific question when needed. In contrast, for MovieLens, Figure 3 (right) depicts more evenly distributed scores with a preferred direction towards smaller outlier scores, reflecting nodes that are less likely to be outliers. Indeed, 78.60% of nodes have an outlier score of 0.50 or less. This can be explained in part based on dataset construction, as we selected only those movie ratings which included content information (tags). It is obvious that users who are involved in commenting on a movie and not just rating it are likely to be more active in the network than other users.

When studying community members for both datasets (Figure 4), we observed that most of the nodes have low outlier score values; thus, they are less likely to be assigned as outliers. Specifically, 71.90% of community nodes in StackExchange and 88.50% of nodes in MovieLens had outlier scores less than 0.50. On the contrary, in Figure 5 the outlier scores are quite high for most nodes that are not members of any community.

However, our proposal for defining an outlier score characterizes a percentage of nodes as outliers or not in all cases, despite their degree of community membership. This is because the content of the network is taken into account. We argue that the definition of an outlier should not rely on structural criteria. A node which is a member of a community but with quite different labels assigned to it and probably fewer connections to other members can be considered to have outlying behavior. In the same way, a node with fewer connections but quite similar content with a specific community, e.g., two forum users that did not exchange answers or comments between them but used similar sets of labels in their interactions, should not be considered an outlier despite not being a member of it.

6. Predicting Outlying Behavior of Nodes

In this section, we present the results of predictive techniques for nodes’ future behavior based on their outlier score. We model the problem as a classification problem,
with each node labeled as either an outlier or an inlier. Our goal is to study the performance of the outlier score as a predictive feature for this classification problem. To this end, we compared its behavior against other commonly used node features based on nodes’ structural characteristics. In particular, we utilized the node degree, betweenness centrality, and PageRank to experimentally verify that using our proposed outlier score provides a better understanding of a node’s outlying behavior and its possible outcome in later time slices.

6.1. Exploration of Outlier Scores and Future Behavior

First, we wanted to explore whether there exists a correlation between the current outlier score $OS$ of a node and its future behavior; that is, we wanted to determine whether nodes with high outlier scores are more likely to leave all communities or even disappear from the network in the next time slices, while nodes with low outlier scores are more likely to join communities and remain in the network in the future, indicating that the proposed outlier score has predictive power.

Tables 6 and 7 depict the results of our exploration for the StackExchange and MovieLens datasets, respectively. For each dataset, the nodes were split between those with high ($OS > 0.50$) and low ($OS \leq 0.50$) outlier scores at time point $t_i$, and we examined their community membership at two consecutive time points $t_i$ and $t_{i+1}$. For each node category, we measured the percentage of nodes that were a member of at least one community and the nodes outside of all communities at each of the two time points.

The desirable conclusion after this experiment would be a robust connection between one node’s outlier score at the current time point and its community membership at the next time point. In the two tables, we indicate the percentages in bold, with high percentages showing a positive correlation between the outlier score and the nodes’ outlying behavior. In Table 6, for StackExchange, it can be observed that, at $t_i$, 41.48% of the nodes with low outlier scores are members of at least one community at the same time point, while at the next time point $t_{i+1}$ this percentage increases to 62.04%. This indicates that 62.04% of the nodes with low outlier scores at $t_i$ were community members at $t_i$, a relative increase of 49.55% concerning the percentage of community members at $t_i$. Regarding high outlier score values, we verified a correlation between high outlier score and the probability of a node not being a community member at the next time slice, with a percentage of 72.06% of such nodes. The reduction in the percentage with respect to $t_i$ is due to the nodes that have left the network in $t_{i+1}$, which are not included in the measurements. Similar conclusions can be drawn for MovieLens, as depicted in Table 7, confirming the predictive characteristics of the proposed outlier score.

<table>
<thead>
<tr>
<th>$t_i$</th>
<th>$t_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community Member</td>
<td>Not Community Member</td>
</tr>
<tr>
<td>Low OS in $t_i$</td>
<td>41.48%</td>
</tr>
<tr>
<td>High OS in $t_i$</td>
<td>17.97%</td>
</tr>
</tbody>
</table>

Table 7. Percentage of nodes in the MovieLens dataset with high/low outlier scores which are or are not members of at least one community at the current and next time slices.

<table>
<thead>
<tr>
<th>$t_i$</th>
<th>$t_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community Member</td>
<td>Not Community Member</td>
</tr>
<tr>
<td>Low OS in $t_i$</td>
<td>66.14%</td>
</tr>
<tr>
<td>High OS in $t_i$</td>
<td>14.86%</td>
</tr>
</tbody>
</table>

6.2. Classification Evaluation

Finally, we evaluated the performance of the outlier score when used as the input feature for a binary classifier that determines whether or not a node is an outlier. Again,
we follow the traditional outlier definition, labeling a node as an outlier when it does not belong to any communities and as not an outlier when it belongs to at least one community. Three different classifiers were examined: Support Vector Classification (SVC), $k$-Nearest Neighbors with the number of closest neighbors set to 5 ($k$NN5), and Decision Tree Classifier (DT). We compared the proposed outlier score against widely used structural node features: degree; betweenness centrality, which measures the number of shortest paths going through a node; and PageRank for solving graph-based ranking, as well as combinations of the three. We split the datasets into training and test sets with an 80–20 split, and provided the classifiers with the corresponding input features.

In Table 8, we report the F-Score for each of the classifiers and the two datasets when using a combination of different features. In this first experiment, we differentiated between structural features and content to better assess the impact of including content in our problem. Thus, we considered the outlier score as two features, consisting of a structural score and content score. In particular, for each dataset, i.e., the first three result rows, the first part of Table 8 reports results relying solely on the structural feature indicated in the corresponding column, while the second part, i.e., the next three results rows, reports results for each of the corresponding features combined with the proposed content score. Thus, we compare our outlier score against popular structural features as well as against these features enhanced with content-based information. The scores in bold are the highest observed in each case. For both datasets, it can be observed that when only structural features are exploited, the node degree and the combination of all three popular structural measures achieve the highest scores at up to 73.79% for StackExchange and up to 78.63% for MovieLens. However, when the content part is included, our outlier score is superior to all others, outperforming all of them in five of six cases. In addition, it achieves higher performance overall, with an F-score of up to 79.48% for StackExchange and up to 85.86% for MovieLens, which is a relative improvement of more than 7% for StackExchange and 10% for MovieLens.

Further focusing on time evolution, we investigated whether exploiting the history of nodes in the past can contribute to improved performance on our classification problem. We modeled a node’s history by collecting its feature values at a series of time points during which the node survives, and used these as the input to our classifiers. To this end, we only focused on nodes that survived for at least three time slices and did not turn inactive (i.e., leave the network) within this period. We measured the features of each node in each time slice, constructing chains of such features. Recall that the $ttl$ and $obs$ pair selected is 60/30; thus, most nodes are investigated during two time slices, and unless their lifetime is refreshed they stay inactive and are not considered.

Tables 9 to 10 show the results achieved in accuracy, F-score, and more specifically precision and recall measures per class, respectively, for the nodes whose past characteristics were taken into account for our two datasets. We compared the outlier score with betweenness centrality combined with PageRank. We selected the Support Vector Classifier as our classifier, as in the previous experiments it showed better results and as, in contrast to $k$–NN, it is not parametric. We experimented with chain lengths of 3 and 4, expecting the higher chain length to provide more accurate results, as it incorporates additional information. The distribution between the two classes in the StackExchange dataset contained 32.09% of the node members in communities (TRUE), while on the MovieLens dataset the rate was 59.52%. Focusing on the results, where the highest scores are depicted in bold, it can be seen that using our outlier score as input can result in better overall accuracy scores as well as better per-class precision and recall. As expected, using a chain length of 4 resulted in better accuracy. Comparing the results with the previous set of experiments, where only one time slice of each node was taken into account, it can be concluded that using a chain of consecutive node observations can result in improved observations. as we achieved an F-score of up to 91.5% for chains of length 4. Notably, all metrics in this case were 88% or above, without any class underperforming in either precision nor recall, as can occasionally be observed with chains of smaller length.
Table 8. Classification results.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Structure Score</th>
<th>Betweenness Centrality</th>
<th>PageRank</th>
<th>Degree</th>
<th>Centrality + Rank + Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC</td>
<td>0.5862</td>
<td>0.6872</td>
<td>0.7097</td>
<td>0.7308</td>
<td>0.7308</td>
</tr>
<tr>
<td>kNN5</td>
<td>0.7094</td>
<td>0.6583</td>
<td>0.6917</td>
<td>0.7166</td>
<td>0.7352</td>
</tr>
<tr>
<td>DT</td>
<td>0.6838</td>
<td>0.6722</td>
<td>0.7097</td>
<td>0.7379</td>
<td>0.7205</td>
</tr>
</tbody>
</table>

+ Content Score

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Structure Score</th>
<th>Betweenness Centrality</th>
<th>PageRank</th>
<th>Degree</th>
<th>Centrality + Rank + Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC</td>
<td>0.7779</td>
<td>0.7454</td>
<td>0.7454</td>
<td>0.7419</td>
<td>0.7419</td>
</tr>
<tr>
<td>kNN5</td>
<td>0.7999</td>
<td>0.7744</td>
<td>0.7998</td>
<td>0.7589</td>
<td>0.7744</td>
</tr>
<tr>
<td>DT</td>
<td>0.7565</td>
<td>0.7844</td>
<td>0.7948</td>
<td>0.7586</td>
<td>0.744</td>
</tr>
</tbody>
</table>

MovieLens

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Structure Score</th>
<th>Betweenness Centrality</th>
<th>PageRank</th>
<th>Degree</th>
<th>Centrality + Rank + Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC</td>
<td>0.6231</td>
<td>0.6266</td>
<td>0.7177</td>
<td>0.7863</td>
<td>0.7863</td>
</tr>
<tr>
<td>kNN5</td>
<td>0.6916</td>
<td>0.6266</td>
<td>0.6694</td>
<td>0.7562</td>
<td>0.7821</td>
</tr>
<tr>
<td>DT</td>
<td>0.6803</td>
<td>0.6215</td>
<td>0.7056</td>
<td>0.7863</td>
<td>0.7738</td>
</tr>
</tbody>
</table>

+ Content Score

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Structure Score</th>
<th>Betweenness Centrality</th>
<th>PageRank</th>
<th>Degree</th>
<th>Centrality + Rank + Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC</td>
<td>0.8701</td>
<td>0.8149</td>
<td>0.8149</td>
<td>0.8508</td>
<td>0.8508</td>
</tr>
<tr>
<td>kNN5</td>
<td>0.8427</td>
<td>0.8335</td>
<td>0.8297</td>
<td>0.8347</td>
<td>0.8105</td>
</tr>
<tr>
<td>DT</td>
<td>0.8586</td>
<td>0.8335</td>
<td>0.8177</td>
<td>0.8056</td>
<td>0.8245</td>
</tr>
</tbody>
</table>

Table 9. Chain classification results: StackExchange.

<table>
<thead>
<tr>
<th>Chain length: 3</th>
<th>Betweenness Centrality + PageRank</th>
<th>Outlier Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.7054</td>
<td>0.7455</td>
</tr>
<tr>
<td>F-score</td>
<td>0.6735</td>
<td>0.6627</td>
</tr>
<tr>
<td>Avg Precision</td>
<td>“TRUE”</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>“FALSE”</td>
<td>0.72</td>
</tr>
<tr>
<td>Avg Recall</td>
<td>“TRUE”</td>
<td>0.49</td>
</tr>
<tr>
<td></td>
<td>“FALSE”</td>
<td>0.68</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chain length: 4</th>
<th>Betweenness Centrality + PageRank</th>
<th>Outlier Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.7246</td>
<td>0.7591</td>
</tr>
<tr>
<td>F-score</td>
<td>0.6935</td>
<td>0.7065</td>
</tr>
<tr>
<td>Avg Precision</td>
<td>“TRUE”</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>“FALSE”</td>
<td>0.71</td>
</tr>
<tr>
<td>Avg Recall</td>
<td>“TRUE”</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>“FALSE”</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Table 10. Chain classification results: MovieLens.

<table>
<thead>
<tr>
<th>Chain length: 3</th>
<th>Betweenness Centrality + PageRank</th>
<th>Outlier Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.6129</td>
<td>0.7234</td>
</tr>
<tr>
<td>F-score</td>
<td>0.4946</td>
<td>0.6965</td>
</tr>
<tr>
<td>Avg Precision</td>
<td>“TRUE”</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>“FALSE”</td>
<td>0.92</td>
</tr>
<tr>
<td>Avg Recall</td>
<td>“TRUE”</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>“FALSE”</td>
<td>0.62</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chain length: 4</th>
<th>Betweenness Centrality + PageRank</th>
<th>Outlier Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.8056</td>
<td>0.9167</td>
</tr>
<tr>
<td>F-score</td>
<td>0.8017</td>
<td>0.9150</td>
</tr>
<tr>
<td>Avg Precision</td>
<td>“TRUE”</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>“FALSE”</td>
<td>0.81</td>
</tr>
<tr>
<td>Avg Recall</td>
<td>“TRUE”</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>“FALSE”</td>
<td>0.85</td>
</tr>
</tbody>
</table>

7. Conclusions

In this paper, we have proposed an online method for detecting communities and outlier nodes in time-evolving social networks where the structure and content of the network are both taken into account. The main idea behind our approach is to perform community detection in parallel to the assignment of an outlier score to each node that depicts its abnormality in terms of both structure and content. The outlier score is a degree of a node’s outlier behavior, with higher values representing nodes that are less connected to the graph and have content that is differentiated from their neighbors. Our approach is adaptive, as users can set an appropriate context-dependent threshold for a node’s
minimum outlier score to be considered as an outlier. The user can leverage the importance assigned to each dimension in both community and outlier detection procedures through the use of appropriate weights; thus, the weight of content characteristics can be lowered when structural connections are assessed as more important, and vice versa.

To validate the usability of the proposed outlier score, we conducted an extensive experimental study based on two real world datasets with different characteristics in terms of structure and content. Our results reveal that the proposed outlier score can successfully serve a twofold purpose. First, its effectiveness as a measure of a node’s outlying behavior was assessed by exploring the correlation of high outlier scores with nodes that are not members of any community and the correlation of low outlier scores with nodes that belong to at least one community. Second, our experiments showed that the proposed outlier score is even more effective as a predictive feature for predicting the behavior of a node at future time points. The proposed score outperformed popular structural node measures even when content was incorporated. Furthermore, we have demonstrated that exploiting the history of nodes by forming chains of features that model a node’s past behavior can prove even more effective in predicting its outlying behavior in the future. Thus, we envision that our score can be utilized for early detection of nodes that exhibit outlying behavior. Consequently, these can be exploited in cases such as early detection of fraudulent behavior to isolate and remove such nodes from the network, or in other cases to detect customers or members likely to leave a service in order to provide them with incentives that reinforce their loyalty.

In the future, we plan to steer our focus towards the importance of the outlier score in prediction-making by experimenting with different classification models and new real world datasets. We additionally plan to enhance outlier detection by parallelizing the algorithms’ procedures and focusing on a more dynamic way to simultaneously assign outlier scores to nodes and predict their future actions based on their history.

Author Contributions: Conceptualization, N.S. and G.K.; methodology, N.S. and G.K.; software, N.S.; validation, N.S. and G.K.; formal analysis, G.K.; investigation, N.S. and G.K.; resources, N.S.; data curation, N.S.; writing—original draft preparation, N.S.; writing—review and editing, N.S. and G.K.; visualization, N.S.; supervision, G.K.; project administration, N.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: https://github.com/sachpenderis/datasets.

Conflicts of Interest: The authors declare no conflicts of interest.

References

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.