2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom) | 978-1-6654-6497-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISPA-BDCLOUD-SOCIALCOM-SUSTAINCOM57177.2022.00030

Context-aware Community Evolution Prediction in Online Social Networks

Alexander Lercher*, Nishant Saurabh†, Radu Prodan*
*Institute of Information Technology, University of Klagenfurt, Austria
†Department of Information and Computing Sciences, Utrecht University, Netherlands
*{alexander.lercher, radu.prodan}@aau.at, †{n.saurabh}@uu.nl

***Abstract*—Community evolution prediction enables business-driven social networks to detect customer groups modeled as communities based on similar interests by splitting them into temporal segments and utilizing ML classification to predict their structural changes. Unfortunately, existing methods overlook business contexts and focus on analyzing customer activities, raising privacy concerns. This paper proposes a novel method for community evolution prediction that applies a context-aware approach to identify future changes in community structures through three complementary features. Firstly, it models business events as transactions, splits them into explicit contexts, and detects contextualized communities for multiple time windows. Secondly, it uses novel structural metrics representing temporal features of contextualized communities. Thirdly, it uses extracted features to train ML classifiers and predict the community evolution in the same context and other dependent contexts. Experimental results on two real-world data sets reveal that traditional ML classifiers using the context-aware approach can predict community evolution with up to three times higher accuracy, precision, recall, and F1-score than other baseline classification methods (i.e., majority class, persistence).**

***Index Terms*—Social networks, context-awareness, community evolution prediction, machine learning.**

I. Introduction

Recently, community detection [1], [2] and evolution prediction [3] gained traction to accurately detect changes in demand and predict customer behavioral patterns in business-centric social media platforms. Existing works in this area typically split communities represented as a set of densely-connected users with similar interests into independent time windows, linked together to form community evolution chains [4]. The communities in consecutive time windows with many similar users are part of the same community chain [5]. On top, current methods employ machine learning (ML) classification techniques [4], [6], [7], trained based on the evolution chain from previous time windows to predict the community evolution in a future time window. Unfortunately, these methods [5], [7] overlook business contexts and focus on analyzing customer activities raising critical privacy issues. Online social platforms adopting privacy-by-design [8], [9] mitigate such privacy challenges but present traceability problems in detecting users with similar interests.

In this paper, we propose a novel *context-aware community evolution prediction* for business-driven online social platforms that focuses instead on critical business events stored in a persistent database and does not require information about individual users and their interactions. For example, specific business events in a taxi social network represent the time of joining a trip, the chosen car type, the start and end locations, and the price. We consider these event properties as the contexts for building contextual communities, providing different viewpoints about the user preferences. We apply temporal segmentation to these communities and create community chains with changing context sizes over time. We employ ML-based methods based on historical data to predict and classify the community evolution:

a) Single-context community prediction: determines the future state of a community based on its previous states.

b) Cross-context community prediction: determines the future state of a community based on states from other communities in another context.

We define in this paper new structural metrics for individual contextual communities to represent the network history used to train ML models for classification and prediction. We implemented the single-context and cross-context community predictions with six ML classifiers using novel context and community metrics to represent detected communities' structures. We performed a series of experiments to study the benefits of single and cross-context methods using two real-world YouTube and Portuguese Taxi data sets. Experimental results demonstrate up to three times improvement in classification accuracy, precision, recall, and F1-score compared to two baseline methods [10], [11].

The paper has six sections. Section II summarizes the related works in community evolution prediction. Section III describes the proposed context-aware community prediction six-phase methodology in detail. Section IV presents the underlying algorithms for temporal community segmentation and single-context and cross-context community prediction. Section V evaluates the single- and cross-context prediction methods on two real-world data sets. Finally, Section VI concludes the paper and outlines the future work.

II. Related Work

Bródka et al. [5] proposed the group evolution discovery (GED) framework for predicting the community evolution based on the number of members and their social position. Saganowski et al. [7] extended the GED method and produced community evolution chains containing features and patterns belonging to the last three disjoint time windows.

Takaffoli et al. [4] and İlhan et al. [12] identified multiple network and node metrics to predict community evolution. The former applied a multi-stage ML model that predicts the community's survival or death based on influential members, communities, and temporal changes. The latter focused on reducing the model training computational cost by selecting the three representative metrics depending on the dataset and relevant features based on nodes and their interactions.

On the contrary, Dakiche et al. [6] applied the GED method that uses the change rates of community structures and influential members to predict community chains using three time windows. However, the conclusion was that using absolute values for the first community features and only changing rates for the next two time windows provides the best results.

Wang et al. [13] approached the community evolution prediction problem across two granular stages. The first stage computes the features for individual communities by considering user roles, then predicts intra- and inter-community relationships in the second stage.

These works analyzed the network stages based on individual users and their interactions to predict community evolution. Hence, they require unique user identifiers over time windows and use user interactions for modeling edges and extracting derived features, including temporal information. In contrast, we propose a novel community evolution prediction that does not analyze individual users or their behavior but focuses on business-relevant interactions stored as transactional events. Our method is suitable for online social networks where user privacy or pseudonymity is a design requirement [14].

III. Context-aware Prediction Methodology

This section explains the context-aware community evolution prediction methodology consisting of six phases.

A. Phase 1: Dataset Preparation

1) Transactional data: labeled with distinct timestamps are the basis for applying context-aware community prediction. We formally represent a *transaction* as an $(l+2)$-tuple:

$$T = (id, ts, L_1, \dots L_l),$$

where id represents its identifier, ts the timestamp and L_i a specific *context* $(1 \le i \le l)$.

2) Data pre-processing: cleans the dataset and prepares the values for ML. In the first step, we convert enumerated types and strings to numbers. Next, we apply value imputation by estimating the missing fields in the original transaction dataset. Deleting transactions with missing values is infeasible, as it changes the number of transactions for a particular time window that can affect the community evolution type.

B. Phase 2: Contextualization

This phase splits all transactions in l contexts. Each identified context presents a different view on the original dataset's transactions, and contains an equal number of *contextual transactions* $T_L = (id, ts, L)$ with three fields: global identifier id, timestamp ts, and a specific context L.

C. Phase 3: Contextual Community Detection

This phase clusters each context L by assigning similar transactions to the same *contextual community* c_i^L. This results in a set of contextual communities C^L for each context L:

$$C^L = \left\{c_1^L, c_2^L, \dots, c_n^L\right\}.$$

D. Phase 4: Temporal Contextual Community Detection

In this phase, we identify the temporal evolution chains for each community in each context by assigning the contextualized transactions to specific time windows based on their timestamps `ts`. Formally, we split each contextual community c_i^L in k *temporal contextual communities*:

$$c_i^L = \left\{c_{i,1}^L, c_{i,2}^L, \dots, c_{i,k}^L\right\},$$

where $1 \le i \le n$ and k is the last known time window. The identified community evolution chains allow comparing the changes of temporal contextual communities across consecutive time windows. The resulting community evolution based on size can be of the type `continuing`, `shrinking`, `growing`, `dissolving`, or `forming` [5].

E. Phase 5: Feature Engineering

Related works employ metrics based on individual nodes, communities, and entire networks to predict the community evolution [7]. However, we cannot use these metrics as their definition depends on the user's interaction. Hence, we define and use context and community metrics without considering individual transactions during ML training.

1) Context metrics: reflect the structure of a context L for a single time window t, computed as aggregates of all communities in C^L and their metrics.

a) Context size: $s(L,t)$ represents the number of contextual transactions aggregated over all non-empty temporal contextual communities AC_t^L at a time window t, indicating the interest in the network:

$$AC_t^L = \left\{ c_{i,t}^L \in C^L \,\middle|\, \left|c_{i,t}^L\right| > 0 \right\};$$
$$s(L,t) = \sum_{c_{i,t}^L \in AC_t^L} \left|c_{i,t}^L\right|.$$

b) Context variety: represents the number of non-empty temporal contextual communities AC_t^L in each context:

$$v(L,t) = \left|AC_t^L\right|.$$

A high context variety indicates many scattered transactions, while a low variety many similar transactions during t.

c) Context entropy: $H(L,t)$ is an important indicator to understand the context stability [15]. A high entropy represents a uniform distribution of transactions within communities and increases the uncertainty over their relevance. A low entropy indicates a distribution of transactions into fewer communities of larger size. We employ Shannon's entropy [15] to calculate the community relevance and uncertainty in a time window t by using their relative sizes representing the probability that a randomly selected transaction belongs to each community:

$$H(L,t) = -\sum_{c_{i,t}^L \in AC_t^L} \left(\frac{\left|c_{i,t}^L\right|}{s(L,t)} \cdot \log_2 \frac{\left|c_{i,t}^L\right|}{s(L,t)} \right).$$

d) Community size aggregates: consist of the minimum, maximum, average, and sum over all active communities in a context L. The four aggregate metrics are useful to represent a variable number of communities or contexts in a network for training an ML model [7].

e) Community popularity aggregates: consist of the minimum, maximum, average, and sum of all active community popularity metrics defined in Section III-E2. In contrast to community size aggregates, the popularity metrics represent relative sizes allowing better comparison of communities over multiple time windows. For instance, popularity metrics remain stable, while the absolute community sizes decrease upon periods with fewer overall network transactions.

f) Community temporal center distance aggregates: are the minimum, maximum, average, and sum of all active community temporal center distance metrics defined in Section III-E2, indicating deviations of community averages over time and potentially rendering the entire context an outlier.

2) Community metrics: reflect the structure of temporal contextualized communities $c_{i,t}^L$ in each context L for a single time window t. Community metrics aggregate transactions' context values and compute ratios compared to other communities within the same context based on the context metrics.

a) Community size: $\left|c_{i,t}^L\right|$ represents the number of transactions in the community. Intuitively, this metric provides the community raw size to indicate its evolution type directly.

b) Community standard deviation: $\sigma\left(c_{i,t}^L\right)$ indicates the density around a community center, calculated as the average of all individual contexts. A higher density represents more substantial communities with fewer outliers, more consistent and stable than scattered communities.

c) Community magnitude: $\left\|c_{i,t}^L\right\|$ represents the space between the extreme context values of a community, representing a range in a one-dimension, an area in two-dimensions, or a volume in three-dimensions. A smaller magnitude indicates a more substantial community with similar context values, while a higher magnitude represents an outlier.

d) Community scarcity: $scarce\left(c_{i,t}^L\right)$ is the average distance between the context values of a community, where a lower scarcity means a more substantial and similar community. Contrary to standard deviation, the scarcity considers the shape of the context distribution focused on the outliers:

$$scarce\left(c_{i,t}^L\right) = \begin{cases} \left(\frac{\left\|c_{i,t}^L\right\|}{\left|c_{i,t}^L\right|}\right)^{\frac{1}{dim\left(c_{i,t}^L\right)}}, & \left|c_{i,t}^L\right| > 0 \\ 0, & \left|c_{i,t}^L\right| = 0, \end{cases},$$

where $dim\left(c_{i,t}^L\right)$ represents the number of context dimensions. A higher-dimensional community increases the magnitude as its context values occupy more space (e.g., three points triangle in a two-dimensions versus three similarly close points in a one-dimension). The exponent $\frac{1}{dim\left(c_{i,t}^L\right)}$ standardizes the division allowing direct comparison of resulting scarcity values independently of the context dimension.

e) Community temporal center distance: is the Euclidean distance between the center of the temporal contextual community $c_{i,t}^L$ and the contextual community c_i^L.

f) Community popularity: represents its size relative to the aggregated sizes of all communities in a context:

$$p\left(c_{i,t}^L\right) = \begin{cases} \frac{\left|c_{i,t}^L\right|}{s(L,t)}, & \left|c_{i,t}^L\right| > 0; \\ 0, & \left|c_{i,t}^L\right| = 0. \end{cases}$$

This metric avoids the bias towards stability if the absolute community size is larger than the average because of a disproportionate number of transactions in a time window.

g) Temporal contextual diversity: is the inverse number of non-empty contextual communities in a time window t:

$$diverse\left(c_{i,t}^L\right) = \begin{cases} \frac{1}{v(L,t)}, & \left|c_{i,t}^L\right| > 0; \\ 0, & \left|c_{i,t}^L\right| = 0. \end{cases}$$

A higher temporal contextual diversity indicates the existence of a more prominent contextual community.

F. Phase 6.1: Single-Context Community Prediction

The single-context prediction model estimates the evolution of a community $c_{i,t+1}^L$ based on its metrics from N previous states $c_{i,t}^L, c_{i,t-1}^L, \ldots, c_{i,t-N+1}^L$. This method captures user transactions that do not change randomly but with a latent pattern identifiable by ML methods.

1) Training data: We derive single-context community prediction training data in five steps.

a) Community metrics: We calculate all community metrics for each temporal contextual community $c_{i,t}^L \in C^L$ in each time window $1 \le t \le k$, where k is the last known window.

b) Time conversion: We convert the time window values into a two-dimensional complete residue system (i.e., modulo 52 for weeks, 24 for hours, 12 for months) using sine and cosine transformations.

c) Community metrics nonuple: We store the metrics for a single temporal contextual community $c_{i,t}^L$ as the following nonuple input to the ML classification:

$$X\left(c_{i,t}^L\right) = \left(\left|c_{i,t}^L\right|, \sigma\left(c_{i,t}^L\right), \left\|c_{i,t}^L\right\|, scarce\left(c_{i,t}^L\right),\right.$$
$$\left. dist\left(c_{i,t}^L\right), p\left(c_{i,t}^L\right), diverse\left(c_{i,t}^L\right), \sin(t), \cos(t)\right).$$

d) Community evolution: We identify the community evolution type by comparing the community size metrics between two adjacent time windows t and $t+1$:

$$y(c_{i,t+1}^L) = \begin{cases} \texttt{continuing}, & \left|c_{i,t}^L\right| = \left|c_{i,t+1}^L\right|; \\ \texttt{shrinking}, & \left|c_{i,t}^L\right| > \left|c_{i,t+1}^L\right| > 0; \\ \texttt{growing}, & 0 < \left|c_{i,t}^L\right| < \left|c_{i,t+1}^L\right|; \\ \texttt{dissolving}, & \left|c_{i,t}^L\right| > \left|c_{i,t+1}^L\right| = 0; \\ \texttt{forming}, & 0 = \left|c_{i,t}^L\right| < \left|c_{i,t+1}^L\right|. \end{cases}$$

e) Temporal community metrics concatenation: We concatenate the community metric nonuples for N consecutive time windows t that serve as input variables $X\left(c_{i,t}^{L}\right)$ for the ML training. The classifier learns the patterns in the structural evolution of the community c_i^L in previous time windows to predict the evolution in the next time window. Hence, the training data consists of evolution chains of size N as ML input variable X and the correct evolution label y as output:

$$X = \left[X\left(c_{i,t-N+1}^{L}\right), \ldots, X\left(c_{i,t-1}^{L}\right), X\left(c_{i,t}^{L}\right)\right];$$
$$y = y\left(c_{i,t+1}^{L}\right).$$

G. Phase 6.2: Cross-Context Community Prediction

The cross-context community prediction model estimates the evolution of a temporal contextual community $c_{i,t}^{L} \in C^L$ based on contextual metrics of the N latest states of a different *reference context* $L_R \neq L$. This approach captures situations when changes in some communities cause correlated changes in other contexts. We only consider the correlation between communities and the entire reference contexts, as the correlation between all communities for all context combinations has a high complexity of $\mathcal{O}\left(n \cdot m \cdot (l-1)\right)$, where n is the number of communities in L, m is the number of communities in $L_R \neq L$, $l-1$ is the number of reference contexts L_R and $n \approx m \gg l$. Considering only reference context metrics reduces this complexity to $\mathcal{O}\left(n \cdot (l-1)\right)$ per context L.

1) Training data: We derive in five steps the cross-context community prediction training data for a context pair $L \neq L_R$.

a) Contextual metrics: We calculate the contextual metrics for L_R based on its communities for each time window $1 \leq t \leq k$, where k is the last known time window.

b) Time conversion: We convert the time window values into two-dimensional features by applying sine and cosine transformations, as in the single-context case.

c) Contextual metric septendecuple: We merge all context metrics for a reference context L_R and time window t. Similarly to the single-context metrics, the resulting tuple $X\left(L_R, t\right)$ contains 17 values with all context metrics (i.e. context size, context variety, context entropy, four community size aggregates, four community popularity aggregates, and four community temporal center distance aggregates) and the time window information (i.e. $\sin(t)$ and $\cos(t)$).

d) Community evolution: We identify the evolution type of a community $c_{i,t+1}^{L} \in C^L$ by using the $y\left(c_{i,t+1}^{L}\right)$ definition from the single-context method (see Section III-F1d).

e) Temporal contextual metrics concatenation: We concatenate the reference contextual metrics of L_R for N consecutive time windows representing the latest structural evolution. Additionally, we added the unique community identifier i to differentiate between the evolution types of predicted communities c_i^L in a context L. We describe the input X and output y variables for learning the evolution of $c_{i,t}^{L} \in C^L$ based on the metrics from the reference context $L_R \neq L$:

$$X = \left[X\left(L_R, t-N+2\right), \ldots, X\left(L_R, t\right), X\left(L_R, t+1\right), i\right];$$
$$y = y\left(c_{i,t+1}^{L}\right).$$

Algorithm 1: Temporal community segmentation.

```
Input  : $TS = \{ T | T = (id, ts, LS) \}$: Transaction dataset
         $L$: Context
Output: $\left(L, C^L\right)$: Context and temporal contextual community pairs
1  Function segmentation($TS$, $L$):
2      $TS \leftarrow$ preprocess($TS$)                          // Phase 1
3      $TS_L \leftarrow \emptyset$
4      forall $T \in TS$ do
5          $TS_L \leftarrow TS_L \cup$ contextualize($T, L$)        // Phase 2
6      end
7      $CS \leftarrow$ OPTICS($TS_L$)                           // Phase 3
8      $C^L \leftarrow \emptyset$;
9      forall $c \in CS$ do
10         $c_i^L \leftarrow$ detectTempCommunities($c$)         // Phase 4
11         $C^L \leftarrow C^L \cup \{ c_i^L \}$
12     end
13 return $\left(L, C^L\right)$
```

IV. Community Prediction Algorithms

This section presents the algorithms for context-aware community evolution prediction.

A. Temporal Community Segmentation

Algorithm 1 receives as input a transactional dataset TS fulfilling the necessary prerequisites specified in Section III-A and a context L. The `segmentation` function in lines 1 – 13 splits contextual communities based on timestamp and returns a pair containing a context L and temporal contextual communities C^L. To detect temporal communities, line 2 first pre-processes the input data by replacing enumerable data with integers (e.g. A replaced by 1) and performing missing value imputation on the original dataset TS (see Section III-A). Thereafter, lines 3 – 6 contextualize each transaction $T \in TS$ (see Section III-B) and create a set TS_L. Next, line 7 detects contextual communities based on the OPTICS density-based clustering algorithm [16] applied on the transaction set TS_L. The lines 9 – 12 iterate over each detected contextual community and divide it into a set of temporal contextual communities c_i^L based on multiple time windows (line 10), and add it to the C^L set (line 11). Finally, line 13 returns the pair of context L and obtained temporal contextual communities C^L representing the algorithm output.

B. Single-Context Community Prediction

Algorithm 2 for single-context community prediction receives as input a set of transactions TS, a context L, an ML classifier and a number of time windows N. Line 1 initializes an empty training dataset. The `segmentation` function defined in Algorithm 1 computes the temporal communities from a set of transaction TS and the context L in line 2. Lines 3 – 10 perform feature engineering (see Section III-E) by iterating each contextual community $c_i^L \in C^L$. Lines 5 – 8 calculates all community metrics (see Section III-E2) for each temporal community $c_{i,t}^{L} \in c_i^L$ and stores it as nonuple $X\left(c_{i,t}^{L}\right)$ (line 6) appended to a community metrics list (line 7). Afterwards, line 9 prepares the ML training data for c_i^L using the computed community metrics concatenated for N consecutive time windows. Finally, line 11 uses the final training dataset to train the input ML classifier as a result.

Algorithm 2: Single-context community prediction.

```
Input  : $TS = \{ T | T = (id, ts, L_1, \dots L_l) \}$: Dataset with transactions
         $L$: Context
         $N \geq 2$: Number of time windows
         $ML$: Prediction classifier
Output: Single-context community prediction classifier
1  $mlDat \leftarrow \emptyset$
2  $\left(L, C^L\right) \leftarrow$ segmentation($TS, L$)
3  forall $c_i^L \in C^L$ do
4      $metrics_C \leftarrow \emptyset$
5      forall $c_{i,t}^L \in c_i^L$ do
6          $X(c_{i,t}^L) \leftarrow$ communityMetrics($c_{i,t}^L, c_i^L$)    // Phase 5
7          $metrics_C \leftarrow metrics_C \cup \left\{ X(c_{i,t}^L) \right\}$
8      end
9      $mlDat \leftarrow mlDat \cup$ prepare($metrics_C, N$)  // Phase 6.1
10 end
11 return train($ML, mlDat$)
```

Algorithm 3: Cross-context community prediction.

```
Input  : $TS = \{ T | T = (id, ts, L_1, \dots L_l) \}$: Transaction dataset
         $L$: Context
         $L_R$: Reference context
         $N \geq 2$: Number of time windows
         $ML$: Prediction classifier
Output: Cross-context community prediction classifier
1  $mlDat \leftarrow \emptyset$
2  $\left(L, C^L\right) \leftarrow$ segmentation($TS, L$)
3  $\left(L_R, C^{LR}\right) \leftarrow$segmentation($TS, L_R$)
4  $metrics_C \leftarrow \emptyset$
5  forall $c_i^L \in C^L$ do
6      forall $c_{i,t}^L \in c_i^L$ do
7          $X(c_{i,t}^L) \leftarrow$ communityMetrics($c_{i,t}^L, c_i^L$)    // Phase 5
8          $metrics_C \leftarrow metrics_C \cup \left\{ X(c_{i,t}^L) \right\}$
9      end
10 end
11 $metrics_L \leftarrow$ contextMetrics($C^{LR}$)              // Phase 5
12 $mlDat \leftarrow$ prepare($metrics_C, metrics_L, N$)      // Phase 6.2
13 return train($ML, mlDat$)
```

C. Cross-Context Community Prediction

Algorithm 3 for cross-context community prediction receives the same input as the single-context method, plus a reference context L_R. Lines 2 – 3 initially obtain the temporal contextual community pairs for the context L and their reference context L_R by calling the `segmentation` function from Algorithm 1. Lines 5 – 10 iterate each contextual community $c_i^L \in C^L$ and perform the feature engineering (similar to Algorithm 2), including the community metrics for all temporal contextual communities in C^L. Line 11 calculates the context metrics (see Section III-E1) using the temporal contextual communities in C^{LR} for the reference context L_R. Finally, line 12 prepares the septendecuples for training the ML classifier as the algorithm's output in line 13.

V. Experimental Evaluation

We implemented our method and ML models in Python 3.6 using `scikit-learn 0.24.0`, `imbalanced-learn 0.8.0`, `numpy 1.19.3`, and `pandas 1.1.5` modules. We executed all experiments on an Intel Xeon Gold 5218 server at 2.3 GHz with 384 GB memory and running Ubuntu Linux 18.04 LTS operating system.

A. Experimental datasets

We chose two datasets for the experimental evaluation.

1) YouTube dataset: from Kaggle[1] contains ten individual top trending video lists per country and additional files containing the video categories. Naturally, the dataset does not contain information about individual users but only aggregates attributes such as the number of likes. Hence, we merged the individual files in a single dataset with seven contexts that build communities and require prediction of their evolution:

a) `Views`, `Likes`, `Dislikes`, `Comments`: indicate corresponding attention metric received by trending videos.

b) `Country`: represents the origin of trending videos.

c) `Category`: shows the most trending video types (e.g. latest movie trailers, pet videos, documentaries).

d) `Trend Duration`: indicates the duration for the videos from upload until receiving the top trending label.

2) Taxi dataset: from Kaggle[2] contains one year of taxi rides from the city of Porto (Portugal) with no explicit passenger information. We do not analyze the whole path taken by a taxi and its customer but only focus on four contexts:

a) `Order Type`: indicates the method of ordering a taxi, such as calling taxi central, visiting a taxi stand, or stopping the free taxi on an arbitrary street.

b) `Day Type`: indicates a normal workday, weekend, holiday, or a day preceding a work-free day.

c) `Start` and `End Location`: indicate origin and destination of a taxi trip.

B. ML data preparation

Applying the context-aware prediction method on both YouTube and Taxi datasets generated imbalanced training data for single and cross-context prediction due to the high number of empty communities, creating more prominent evolution types and classification biases. Hence, we combined undersampling and oversampling estimates to avoid such biases towards frequent community evolution types during training. We chose the median number of community evolutions over all types as the sampling size, containing sufficient ML training entries. We undersampled more frequent evolution types to match the median class size and oversampled rare types by applying the synthetic minority technique [17]. Contrary to the training data, we did not undersample or oversample the testing data to correctly represent the original dataset distribution when evaluating the trained ML models.

C. ML classification models

We implemented six ML models to evaluate our method: naïve Bayes (NB), support vector machine (SVM), k-nearest neighbors (KNN), decision tree (DT), random forest (RF), and boosting. All models use training data generated by the single-context and cross-context algorithms, differently sized per context depending on the number of community and context metrics. We standardized each metric value range in

[1]https://www.kaggle.com/datasnaek/youtube-new

[2]https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data

TABLE I: ML hyperparameter tuning for single-context and cross-context community prediction.

Model	*Hyperparameter*	*Value range*	*Selection*	
			Single-context	*Cross-context*
NB	Prior distribution	class probabilities, none	none	class probabilities
	Smoothing	[0, 1]	0	1e−9
SVM	Regularization	$\mathbb{R}$	1	1
	Kernel	{linear, rbf, poly, sigmoid}	linear	linear
	Kernel coefficient	{scale, auto, $\mathbb{R}$ }	scale	scale
KNN	Number of neighbors	$\mathbb{N}$	20	30
	Neighbor weights	{uniform, distance, callable}	uniform	uniform
	Algorithm	{auto, ball_tree, kd_tree, brute}	auto	auto
	Tree leaf size	$\mathbb{N}$	30	50
DT	Split criterion	{gini, entropy}	gini	gini
	Splitter	{best, random}	random	random
	Maximum tree depth	$\mathbb{N}$, none	10	none
	Minimum leaf size	$\mathbb{N}$	1	2
	Minimum impurity decrease for split	$\mathbb{R}$	1e−5	1e−5
	Cost-complexity pruning	$\mathbb{R}^+$	1e−3	0
RF	Number of estimators	$\mathbb{N}$	100	100
	Split criterion	{gini, entropy}	gini	gini
	Maximum tree depth	$\mathbb{N}$, none	none	none
	Minimum leaf size	$\mathbb{N}$	2	2
	Minimum impurity decrease for split	$\mathbb{R}$	1e−5	1e−5
Boosting	Base estimator	ML classifier	DT	DT
	Number of estimators	$\mathbb{N}$	50	50
	Algorithm	{SAMME, SAMME.R}	SAMME.R	SAMME.R
	Learning rate	$\mathbb{R}$	0.3	0.3

the training and testing data to resemble a normal distribution with a mean of zero and a standard deviation of one, reducing bias towards features with larger numbers [18]. In addition, we applied dimensionality reduction with principal component analysis (PCA) on the standardized dataset. We used the first ten principal components that explain at least 80% of the data variance for training additional ML models on uncorrelated features, NB*, SVM*, KNN*, DT*, RF*, and boosting*. Finally, we used a sample of the balanced standardized training data for hyperparameter tuning of each model from the single and cross-context algorithms, as illustrated in Table I.

D. Baseline methods

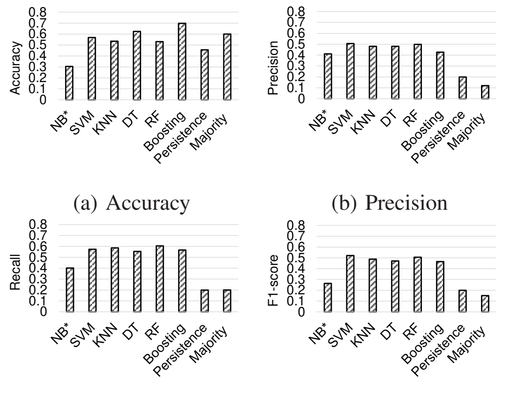
We compared all ML models to two related baseline models.

a) Majority class [11]: prediction assigns the most frequent community type used for training to all communities used for testing, leading to an accuracy equal to its proportion in the entire dataset.

b) Persistence method [10]: assumes stable change of communities preserving the evolution type across time windows. Contrary to the majority class, this method assigns one community evolution type considering the community's immediate history from the previous time window.

E. Evaluation and analysis method

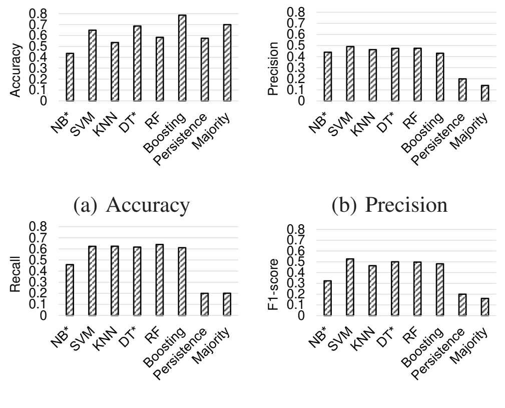
We applied 10-fold cross validation [19] on the ML datasets for both community prediction methods to evaluate all ML models. We evaluated the community evolution prediction using `Accuracy`, `Precision`, `Recall`, and `F1-score` metrics [20]. We present the averages of the prediction metrics for each ML classifier's five possible evolution types. We only visualize the better version of each classifier, trained directly on community metrics or after applying PCA for brevity reasons. We do not present results for all contexts in the single and cross-context community prediction methods but only show the contexts with the greatest variety per method.



(a) Accuracy (b) Precision

(c) Recall (d) F1-score

Fig. 1: `Likes` community prediction for the YouTube dataset.



(a) Accuracy (b) Precision

(c) Recall (d) F1-score

Fig. 2: `Start Location` prediction for Taxi dataset.

F. Single-context community prediction

We analyze the single-context community prediction results for the `Likes` and `Start Location` contexts in the YouTube (see Figure 1) and Taxi (see Figure 2) datasets.

a) Majority class: reached an accuracy of 60% for the YouTube `Likes` (see Figure 1a) and 70% for the Taxi `Start Location` context (see Figure 2a). This surprisingly good accuracy is due to imbalanced testing datasets with a disproportional amount of empty `continuing` communities as the majority class. The low precision, recall, and F1-score metrics below 20% for both contexts confirm this paradox [20].

b) Persistence method: improved over the majority class baseline and obtained 20% precision, recall, and F1-scores for both imbalanced testing datasets and contexts. Essentially, this method showed similar results as random theoretic guessing for the five evolution types (i.e., $\frac{1}{5}$) but assumes stable changes for predicting communities in both contexts. However, the persistence method obtained a lower accuracy of 46% for `Likes` and 57% for `Start Location` contexts, as it predicts based on the direct history across time windows instead of relying on frequent community evolution types.

c) NB: performed worse than the baseline classifiers due to the violation of the naïve independence assumption for input features, as community metrics from consecutive time windows do not change independently but depend on each other. Using transformed features based on PCA yielded better accuracy (30%), precision (41%), recall (40%), and F1-score (26%) for the YouTube Likes context, respectively 44%, 44%, 46%, and 32% for the Taxi Start Location context.

d) DT: performed better than the baseline methods and obtained a higher accuracy (62%), precision (48%), recall (55%), and F1-score (47%) for the YouTube Likes context. For this context, DT* with PCA performed worse than DT. On the contrary, DT* achieved better accuracy (69%), precision (47%), recall (62%), and F1-score (50%) than DT for the Taxi Start Location context. We analyze the single-context prediction results of boosting and RF ensemble techniques utilizing multiple DTs to mitigate this inconsistency.

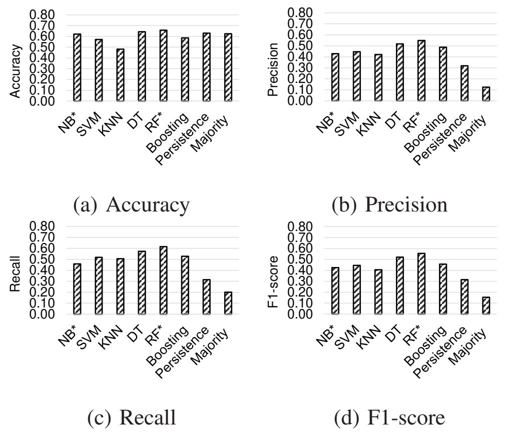
e) Boosting: reached the highest accuracy of 70% and 79% for both Likes and Start Location contexts compared to all prediction methods. However, it exhibited the accuracy paradox by ignoring the forming community evolution type with increased false positives of other evolution types. This weakness reflects in the low precision (43%) for the YouTube Likes and the Taxi Start Location context. Increasing the number of estimators or decreasing the learning rate (see Table I) can solve this paradox of ignoring under-represented evolution types but requires increased training. Changing the base estimator from DT to SVM improved the results for contexts with smaller training data (e.g., YouTube Country and Taxi Order Type) because SVM needs fewer data to learn the latent patterns. However, DT base estimators showed better results across contexts with all training data.

f) KNN: performed better than both baseline predictors and NB* achieving slightly better precision (48%), recall (59%), and F1-score (49%) than boosting for YouTube Likes context and similar for Start Location context (i.e. 46% precision, 62% recall, 46% F1-score). Contrary to boosting, KNN predicted all five evolution types in all contexts of both datasets. KNN* performed similarly to KNN with differences up to only 2%, indicating that PCA-transformed feature space did not change relative distances between nearest neighbors.

g) SVM and RF: performed similarly and achieved the best overall results across all contexts. Their average accuracy is 57% and 59%, average precision is 52% and 55%, average recall is 59% and 65%, and F1-score is 50% and 55%. Both SVM and RF improved the baseline prediction methods by up to 3 times in precision, recall, and F1-score. Finally, RF and SVM considered all five evolution types for all contexts during prediction, while RF* and SVM* only ignored the smallest shrinking evolution type for one context.

G. Cross-context community prediction

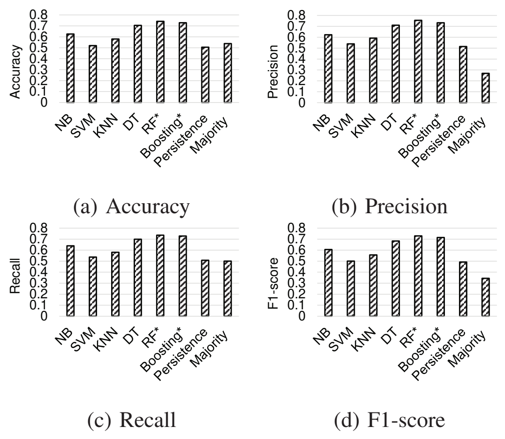
We analyze the Cross-context community prediction results for Trend Duration based on the number of Views reference context in the YouTube dataset, and Order Type based on the Day Type context in the Taxi dataset.



(a) Accuracy (b) Precision

(c) Recall (d) F1-score

Fig. 3: Trend Duration community prediction based on the Views reference context.



(a) Accuracy (b) Precision

(c) Recall (d) F1-score

Fig. 4: Order Type community prediction based on the Day Type reference context.

a) Majority class and Persistence method: achieved high accuracy of 62% and 63%, respectively, for Trend Duration (see Figure 3a) in YouTube dataset. Majority class exhibited accuracy paradox for Trend Duration by predicting a disproportionate amount of empty continuing communities with precision (12%), recall (20%), and F1-score (15%) metrics below theoretical balanced random guessing result (20%). Persistence method performed better with slightly improved precision (32%), recall (31%) and F1-score (31%) metrics. Similarly, for Order Type based on Day Type reference context in Taxi dataset, accuracy (54%), precision (27%), recall (50%), and F1-score (34%) for majority class is below the balanced random guessing result of 50% for the two evolution types shrinking and growing. This result is due to different majority classes for the training and testing data for 5 out of 10 cross-validation folds. On the contrary, the persistence method resembled random guessing (50%) results for Taxi Order Type based on Day Type context.

b) KNN: obtained a lower accuracy (48%), precision (42%), recall (51%), and F1-score (41%) than other ML methods for the YouTube `Trend Duration` context. However, it obtained slightly higher accuracy (58%), precision (59%), recall (58%) and F1-score (56%) for the Taxi `Order Type` context, performing only better than SVM.

c) SVM: improved over both baseline classifiers for the YouTube `Trend Duration` with 1.5 times higher precision (45%), recall (52%), and F1-score (45%). However, it performed similar to the persistence method and achieved random guessing results (50%) for the Taxi `Order Type` context.

d) NB: performed worse than the baseline classifiers for the YouTube `Trend Duration` based on the `Views` context, while it performed slightly better for the Taxi `Order Type` context. Conversely, its PCA version NB* improved prediction results for the `Trend Duration` context by 10%, but worsened results for the Taxi `Order Type` by 5%. Manually selecting the better version for `Trend Duration` and `Order Type` achieves high accuracy (62% and 63%), precision (43% and 62%), recall (46% and 64%), and F1-score (62% and 61%), respectively in each case. We also observed that NB and NB* ignored `shrinking`, `growing`, and `forming` evolution types for some contexts due to their under-representation in the testing dataset.

e) DT, boosting, and RF: delivered the best cross-context prediction compared to other methods. RF* achieved the highest accuracy (66% and 74%) followed by DT (64% and 70%) and boosting (59% and 73%) for `Trend Duration` and `Order Type`. DT predicted the `Trend Duration` context with higher precision, recall, and F1-score compared to the boosting technique that uses 50 DTs sequentially (see Table I). RF ensemble technique with 100 parallel DTs performed better for both `Trend Duration` and `Order Type` context due to its ability to combine best results across all DTs.

VI. Conclusions

We introduced a context-aware community evolution prediction method in business-driven social networks that initially split the business events modeled as transactions into explicit contexts, detect contextualized communities, and divide them temporally. Next, it enables feature extraction by calculating 15 context and 7 community metrics and applies six ML models to perform single and cross-context community prediction. Experimental results on two real-world datasets demonstrated that our methods improve the community prediction metrics by up to three times compared to baseline classifiers. In future, we plan to combine the single and cross-context methods and provide an automatic selection of the most relevant context and community metrics for improved prediction.

Acknowledgment

European Union's Horizon 2020 research and innovation project ARTICONF: "Smart Social Media Ecosytstem in a Blockchain Federated Environment" (grant 825134) funded this work.

References

[1] N. Karandikar, R. Abhishek, N. Saurabh, Z. Zhao, A. Lercher, N. Marina, R. Prodan, C. Rong, and A. Chakravorty, "Blockchain-based prosumer incentivization for peak mitigation through temporal aggregation and contextual clustering," *Blockchain: Research and Applications*, vol. 2, no. 2, p. 100016, 2021.

[2] B. Guidi, A. Michienzi, and G. Rossetti, "Towards the dynamic community discovery in decentralized online social networks," *Journal of Grid Computing*, vol. 17, no. 1, pp. 23–44, Mar 2019.

[3] C. Makris, G. Pispirigos, and I. O. Rizos, "A distributed bagging ensemble methodology for community prediction in social networks," *Information*, vol. 11, no. 4, 2020.

[4] M. Takaffoli, R. Rabbany, and O. R. Zaïane, "Community evolution prediction in dynamic social networks," in *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, 2014, pp. 9–16.

[5] P. Bródka, S. Saganowski, and P. Kazienko, "Ged: the method for group evolution discovery in social networks," *Social Network Analysis and Mining*, vol. 3, no. 1, pp. 1–14, Mar 2013.

[6] N. Dakiche, F. B.-S. Tayeb, Y. Slimani, and K. Benatchba, "Community evolution prediction in dynamic social networks using community features' change rates," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019, p. 2078–2085.

[7] S. Saganowski, P. Bródka, M. Koziarski, and P. Kazienko, "Analysis of group evolution prediction in complex networks," *PLOS ONE*, vol. 14, no. 10, pp. 1–18, 10 2019.

[8] R. Prodan, N. Saurabh, Z. Zhao, K. Orton-Johnson, A. Chakravorty, A. Karadimce, and A. Ulisses, "ARTICONF: Towards a smart social media ecosystem in a blockchain federated environment," in *Euro-Par 2019: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science, vol. 11997. Springer, May 2020, pp. 417–428.

[9] N. Saurabh, C. Rubia, A. Palanisamy, S. Koulouzis, M. Sefidanoski, A. Chakravorty, Z. Zhao, A. Karadimce, and R. Prodan, "The articonf approach to decentralized car-sharing," *Blockchain: Research and Applications*, vol. 2, no. 3, p. 100013, 2021.

[10] H. Bludszuweit, J. A. Dominguez-Navarro, and A. Llombart, "Statistical analysis of wind power forecast error," *IEEE Transactions on Power Systems*, vol. 23, no. 3, pp. 983–991, 2008.

[11] J. Thomason, D. Gordon, and Y. Bisk, "Shifting the baseline: Single modality performance on visual navigation & qa," *arXiv preprint arXiv:1811.00613*, 2018.

[12] N. İlhan and Şule Gündüz Öğüdücü, "Feature identification for predicting community evolution in dynamic social networks," *Engineering Applications of Artificial Intelligence*, vol. 55, pp. 202–218, 2016.

[13] Z. Wang, Q. Xu, and W. Li, "Multi-layer feature fusion-based community evolution prediction," *Future Internet*, vol. 14, no. 4, 2022.

[14] A. Palanisamy, M. Sefidanoski, S. Koulouzis, C. Rubia, N. Saurabh, and R. Prodan, "Decentralized social media applications as a service: a car-sharing perspective," in *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1–7.

[15] P. M. Cincotta, C. M. Giordano, R. Alves Silva, and C. Beaugé, "The shannon entropy: An efficient indicator of dynamical stability," *Physica D: Nonlinear Phenomena*, vol. 417, p. 132816, 2021.

[16] Z. Deng, Y. Hu, M. Zhu, X. Huang, and B. Du, "A scalable and fast optics for clustering trajectory big data," *Cluster Computing*, vol. 18, no. 2, p. 549–562, jun 2015.

[17] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, p. 321–357, Jun. 2002.

[18] P. Trebuňa, J. Halčinová, M. Fil'o, and J. Markovič, "The importance of normalization and standardization in the process of clustering," in *2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2014, pp. 381–385.

[19] T.-T. Wong and P.-Y. Yeh, "Reliable accuracy estimates from k-fold cross validation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586–1594, 2020.

[20] M. F. Uddin, "Addressing accuracy paradox using enhanched weighted performance metric in machine learning," in *2019 Sixth HCT Information Technology Trends (ITT)*, 2019, pp. 319–324.