Massive Cross-Platform Simulations of Online Social Networks

Goran Murić, Alexey Tregubov, Jim Blythe, Andrés Abeliuk, Divya Choudhary, Kristina Lerman,
Emilio Ferrara
{gmuric,tregubov,blythe,aabeliuk,dchoudha,lerman,ferrarae}@isi.edu
USC Information Sciences Institute
Marina del Rey, CA, USA

ABSTRACT

As part of the DARPA SocialSim challenge, we address the problem of predicting behavioral phenomena including information spread involving hundreds of thousands of users across three major linked social networks: Twitter, Reddit and GitHub. Our approach develops a framework for data-driven agent simulation that begins with a discrete-event simulation of the environment populated with generic, flexible agents, then optimizes the decision model of the agents by combining a number of machine learning classification problems. The ML problems predict when an agent will take a certain action in its world and are designed to combine aspects of the agents, gathered from historical data, with dynamic aspects of the environment including the resources, such as tweets, that agents interact with at a given point in time. In this way, each of the agents makes individualized decisions based on their environment, neighbors and history during the simulation, although global simulation data is used to learn accurate generalizations. This approach showed the best performance of all participants in the DARPA challenge across a broad range of metrics. We describe the performance of models both with and without machine learning on measures of cross-platform information spread defined both at the level of the whole population and at the community level. The best-performing model overall combines learned agent behaviors with explicit modeling of bursts in global activity. Because of the general nature of our approach, it is applicable to a range of prediction problems that require modeling individualized, situational agent behavior from trace data that combines many agents.

KEYWORDS

massive scale simulations; collaborative platforms; agent based simulation; AI agents; online social networks

1 INTRODUCTION

Agent-based simulations derived from observational data can be used as a powerful tool to model and predict how users react to different information to which they are exposed on social media platforms. However, a number of challenges present themselves in effectively using available data to build agent models that can be

used in high-fidelity simulations to make such predictions. First, the phenomena of interest may be the product of many agents following decision workflows that are not easily recoverable from the available data. Second, machine learning approaches have typically been used in this area to learn particular focused aspects of user behavior via classification methods (e.g. link-prediction, content suggestions), and have not been applied to larger questions of information flow and evolution. Third, ML models that do address global information flow tend to create a single global model, that is limited in its predictive power because it does not capture the differing reactions and activities of individual agents that might behave differently in the future because of variations to their individual environment and goals. Fourth, the data usually embodies observations of events that took place within a given set of social media platforms, meaning that positive examples of user activity abound while it is harder to find negative examples, for example of when and why a specific action was not taken. Finally, any learned or rule-based behavior must be designed to run very efficiently in order to scale simulations to hundreds of thousands or millions of users while modeling individual decisions taken in their dynamic contexts.

In this paper we present a framework for agent modeling of user behavior from observational data that addresses these challenges, and show its use in building an effective model of cross-platform behavior across GitHub, Twitter and Reddit. Some key contributions of this paper include:

- We develop a set of ML models that learn aspects of user behavior in Twitter and Reddit as classification problems, combining features of users and resources.
- We use a set of simple agent behavior models implemented in DASH framework [6, 9]. DASH allows developing cognitive agents and integrating our ML models along with simpler statistical and rule-based models of behavior.
- We develop features for ML learning that can be interpreted differently for different agents, such as 'the volume of the Reddit channel I have most frequently contributed to', to combine generalization from the data with individualized agent behavior.
- We present performance results of four model configurations that were rigorously evaluated during the DARPA SocialSim Challenge event along with two baseline comparisons. Our models performed the best of all participants across a broad range of metrics defined for the event, indicating the promise of the approach we describe here. The best performing model overall combines learned agent behavior with statistical modeling of bursty behavior, indicating a beneficial interaction between this agent-centered learning and the architecture of the simulation.

The next section introduces the domain in which we modeled cross-platform information spread and the available data. We then introduce our agent model and the machine learning approaches used to define agent behavior. Next we present our experimental results and related work.

2 CHALLENGE PROBLEM DESCRIPTION

The DARPA SocialSim Challenge aims at simulating specific types of dynamics between users and content on three major online social platforms: GitHub, Twitter and Reddit. The goal of a challenge is to predict how a given unit of information will spread across a multi–platform online environment. In particular, it is designed to focus on the simulation of social structure and temporal dynamics with the focus on: 1) cross-platform information spread, 2) spread of specific units of information: Common vulnerabilities and exposures (CVE) IDs, URLs and malwares and 3) quantifying how the newly created information spreads. A unit of information is any trackable string that can be identified across platforms, including specific URLs, CVE IDs and malware-related keywords. For example, suppose that an important malware has been discussed in Twitter with hashtag *#malware_name*, and consequently the associated anti-malware software starts appearing in GitHub and is discussed in Reddit. The simulation should capture as precisely as possible the behaviors of all actors involved in the information spread.

All the teams participating in the challenge were provided with the *training data*: a set of relevant actions extracted from GitHub, Reddit and Twitter over the course of 32, 32 and 19 months respectively. The data covers the period before the end of August 2017, which corresponds to the end of the training data period in Figure 1. Some additional data has been provided such as additional information on users, the list of potential bot accounts or information on GitHub repositories. The teams had the freedom to use the provided data as needed to make the appropriate models. At the beginning of the challenge, the teams receive some additional data covering an unidentified period before the simulated time, which can be used to setup the *initial conditions* of the system (Figure 1). *Initial conditions* data cover period of 2 weeks right before test period (simulation time). A time *gap* between the training data and the initial conditions data could be introduced. The teams are requested to simulate a specific time interval after the initial conditions. Lenght of the *Test period or simulation time* is 6 weeks. The "accuracy" of a simulation has been assessed by comparing the *simulation output* to the *ground truth* using the set of measures as explained in Section 4.2.

At the beginning of the simulation, the initial condition data contains a set of actions and content which could contain the relevant keywords or no keywords at all. The users can interact with one or many pieces of content in any time during the simulation.

The simulation output is the list of all the actions performed by the relevant actors in all three platforms. For **GitHub**, we simulate ten actions that users can make: create or delete either a repository, a tag, or a branch (respectively *Create* and *Delete*), create or comment a pull request (respectively *PullRequest* and *PullRequestReviewComment*), create an issue (*Issues*, *IssueComment*), and push (*Push*, *CommitComment*). Moreover, a user can *watch* and *fork* existing repositories. For **Reddit** we simulate two user actions: *post* – creating the original content; and *comment* – commenting on a
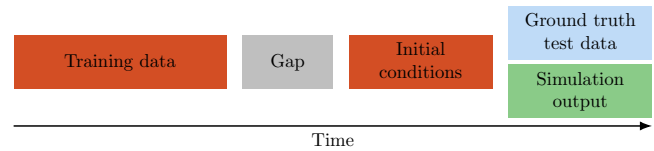

Figure 1: Simulation timeline. *Training data* has been used to train the models. *Initial conditions* data has been provided at the day of the challenge. It is used to set the initial state of the system before the simulation. After the simulation, the *simulation output* is then compared to *ground truth* to assess the accuracy of the simulation.

post or on another comment. For **Twitter** we focus on two user actions: *tweet* – creating an original tweet; and *retweet* – broadcasting the other user's tweet.

3 AGENT FRAMEWORK AND MODELS

To simulate multiple social platforms with millions of users interacting with hundreds of thousands of pieces of content, we use the DASH agent framework implemented in the FARM simulation platform [9], implemented in Python. At the core of the simulation framework is discrete event simulation where the event queue contains two different kinds of events. First, individual agents appear on the queue with specific time-points. When these reach the front of the queue they begin a decision process to determine what action will be taken, *e.g.* a comment on a Reddit post. On completing an action the agent is placed back on the queue at a time point that implements a basic rate of activity for the agent. The second kind of object on the event queue is a trigger, which on reaching the front of the queue may conditionally initiate new events based on the state of the simulation. As we describe below, in this case we use triggers to model bursty behavior, or rapid increases in the rate of activity perhaps triggered by some event that is external to the social media environment. We call this technique a dual event queue approach. While it is simple, it combines flexibility in modeling global behavior with the efficiency of the discrete event model.

3.1 ML ready agents and simulation components

We describe our general approach to agent modeling of multiple social media platforms. Some researchers have modeled cross-platform information spread at the global level or with very simple agents [13, 18]. In contrast, we use a high-fidelity multi-agent model that allows agents to make independent choices using potentially complex decision algorithms. In reality, users of a social network have their personal preferences and habits. For example, one Twitter user may be interested in UK politics, while the other follows sports, and both of them are also interested in crypto coins. The first one might be very active and often engage in discussions, while the other one rarely replies. Users can vary widely across a number of different dimensions. In order to create such a model, however, we must specify this variety of behaviors for possibly millions of agents in a way that matches our observable data, and for this we turn to machine learning approaches as described below.
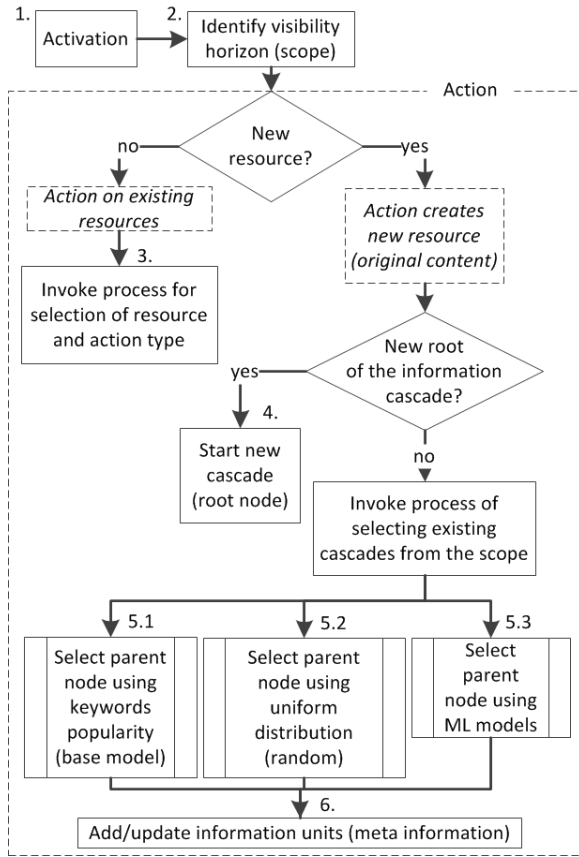
Figure 2: Agents' workflow

Agents in our simulation interact with each other by interacting with the social media resources they produce or are associated with. The nature of interactions can be various and multiple. For example, in Twitter we define agents as users and resources as tweets. Users can interact with tweets by retweeting them, liking them or replying to them.

The environment ϵ, then, consists of the agents $\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \in A$ and resources $\{r_1, r_2, \ldots, r_m\} \in R$. Agents and resources are defined by their state. States include two sets of features X_{α_i} and X_{r_j} for agents and resources respectively.

All models used agents' decision process shown in Figure 2. This process is implemented as a DASH agent and consists of modular components that can have multiple implementations (marked 1-5). Experiments can also be configured to have heterogeneous agents using different modules implementations (algorithms).

The major components of the agents' decision process are the following:

(1) *Activation.* Agents become active and ready to take an action in one of two ways: internal self-activation by coming to the front of the event queue as an agent or external activation via a trigger that comes to the front of the queue. These methods can be used simultaneously or separately. This dual queue approach for incorporating external signal into simulations described in Section 3.4

(2) *Identification of the visible horizon.* It is both computationally and cognitively infeasible for the active agent to consider all possible resources in making an action decision. The agent's visible horizon h is defined by a set of rules that limit the resources the agent can possibly interact in a given time step. The users of a social network are presented with a very small subset of the resources they could in theory interact with, as illustrated in Figure 3a. For example, a Twitter user can interact only with tweets that are displayed on their device. Our attempt is to emulate this limited visibility of the available resources by allowing agents to interact only with a specific fraction of resources. The models for calculating the visibility horizon h are described more in details in Section 3.5

(3) *Invoking the process of selecting a resource and action type.* If an agent makes a decision to take an action on an already existing resource (e.g. edit or delete its tweet) then it needs to select resources to interact with. This decision can be translated into the following set of problems: given the agent-resource pair with the corresponding sets of features, whether the action will be selected. These are classification problems, where a combined set of features $x = x_{\alpha_i} \parallel x_{r_j}$ needs to be classified into positive or negative class. Models developed for this problem are described in Section 3.2

(4) *Starting a new information cascade.* Starting a information cascade means creating an original resource rather than acting on an already existing resource. Our simulation models use training data to sample the roots of information cascades. Information units (keywords, hashtags, URLs) are also sampled from the training data.

(5) *Reacting to an existing information cascade.* This process is similar to (3). This decision can be translated into the following question: given the agent-resource pair with the corresponding sets of features, whether the action will be selected. Twitter and Reddit models developed for this problem are described in Section 3.2

(6) *Adding/Updating information units.* Agents can add and update meta information of the resource (e.g. keywords, hashtags, URLs). Information spread is measured per each information unit and discussed in section 4.

Each of the agents in this framework are cross-platform agents in the sense that they can take actions on different platforms. The social platform, where the action happens, is specified when the agents' activation is scheduled on the queue. Parameterization and training required for each component (1-5) in Figure 2 is done for each social platform.

During the simulation, this process is repeated for each agent whenever it is activated as illustrated in Figure 3b. To optimize the use of the resources, we sometimes pre-compute, parallelize or bundle some of the steps when possible.

3.2 Twitter and Reddit ML models

ML models predict the probability of interaction between an agent and a resource. In the "real world" the users are presented with a set of objects they can interact with. For instance, a user on Twitter can see a few dozen tweets in a given moment. For each of the
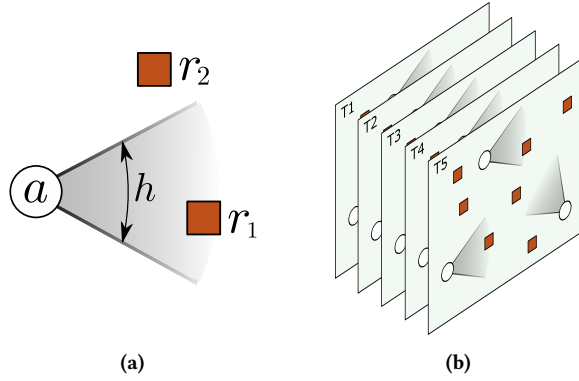
Figure 3: Agents, visibility horizon and simulation in time: a) An agent can interact with any resource which falls within their visibility horizon h. The decision if a particular agent will interact with a particular resource is left to other decision models; b) Multiple agents are active within the simulated environment, and they repeat their decision process over time

tweets, a user makes a decision either to like a tweet, retweet, reply or ignore it. In our simulation, for each of the actions the decision process is translated into a binary classification problem for a given feature vector $X_k = X_i^\alpha \parallel X_j^r \parallel X_k^c$. The feature vector is made up of feature vector X_i^α for an agent i, a feature vector X_j^r for a resource j and a combined feature vector X_k^c which are concatenated into a single vector X_k. This way, for each combination of agent–resource pairs we can construct the unique feature vector that will be used to make a decision on an action. During the simulation, as each agent reaches the front of the event queue, it solves a binary classification problem on the set of resources that are within agent's horizon h. For a given agent α_i, a vector X_k is constructed for all resources that are within the horizon h.

Reddit training. In the case of Reddit, we solve two classification problems: (1) *post→comment*: whether the user will make a comment on a given post and (2) *comment→comment*: whether the user will make a sub–comment on a given comment. For each of the two problems we build separate training datasets.

For the *post→comment* problem, we first match all the users with posts they commented on. Then, for each user–post pair we create a unified feature vector by concatenating user features, post features and combined features in a single vector as illustrated in Table 1. All the vectors created this way are assigned a positive label as they represent the instances of users commenting on the posts. The initial dataset consists only of positive examples. This problem is different from the standard supervised classification problem by the lack of negative examples in the training set. Usually, to properly train a binary classifier, one needs the negative examples. The data on "real" negatives could be harvested by carefully observing user's behavior within the Reddit interface and select the posts user saw but decided to ignore. Such information is not available to us and we approach this task as the one-class classification problem.

Often referred to as *PU learning*, this problem is solved by building a binary classifier from a training set consisted of positive P

and unlabeled U data. To create the U part of the dataset, we match users with posts they did not comment to and create the feature vector from such user–post pairs. A fairly naive interpretation of the *PU learning* process is that unlabeled vectors that are similar to true positives will be labeled as positive and unlabeled vectors which are different from true positive will be labeled as negative. It is out of the scope of this paper to discuss the multiple nuances of *PU learning*. To build our training sets, we use an approach similar to one proposed in [20].

Finally, with the training set ready, we build a classifier able to predict if a given *user–post* pair will result in a user commenting on a post. After comparing multiple classification algorithms and parameter optimization, we decided to use Multi-layer Perceptron classifier with 15 hidden layers, which yielded the highest AUC score.

Table 1: To build the agent-based ML models, the training set has been build by assembling *user* features, *post* features and *combined* features.

	User features			Post features			Combined features			
	U_0	...	U_n	P_0	...	P_m	C_0	...	C_z	y
X_0	$x^\alpha_{0,0}$	...	$x^\alpha_{0,n}$	$x^r_{0,0}$	...	$x^r_{0,m}$	$x^c_{0,0}$	...	$x^c_{0,m}$	1
X_1	$x^\alpha_{1,0}$	...	$x^\alpha_{1,n}$	$x^r_{1,0}$	...	$x^r_{1,m}$	$x^c_{1,0}$	...	$x^c_{1,m}$	1
.										
.										
X_i	$x^\alpha_{i,0}$	...	$x^\alpha_{i,n}$	$x^r_{i,0}$	...	$x^r_{i,m}$	$x^c_{j,0}$	...	$x^c_{j,m}$	0
X_k	$x^\alpha_{-,0}$	...	$x^\alpha_{-,n}$	$x^r_{-,0}$	...	$x^r_{-,m}$	$x^c_{k,0}$	...	$x^c_{k,m}$	0

We approach the *comment→comment* problem in a similar way. The main difference is in the set of features used to predict the child comment. For the *post→comment* problem we match user features with the *post* features, while for the *comment→comment* problem we match user features with the *comment* features. This way, we allow the agents to apply the different models when deciding to comment, depending on the resource they are commenting on.

Reddit features. For the *post→comment* problem the final feature vector X_k consists of ≈ 200 features made from user features X_i^α, post features X_j^r and combined features X_k^c:

- *User features X_i^α* consist of *preference features*, *sentiment features* and *frequency features*.
  Preference features are multidimensional vectors which represent the user's preference towards *keywords*, *subreddits* and *domains*. The main idea behind building the preference vectors is to quantify the user's interests and leaning towards certain topics. The preference vectors are built using FastText word embedding technique [10]. First we train a set of FastText models using the corpus from the relevant content on Reddit. Then, we assign the embedded vectors to each keyword, subreddit name and domain which appear in the training data. During the simulation, there is a possibility of encountering the keywords or subreddit names which do not exist in the training data. FastText can embed words which were never seen before but appear in a similar context or have a similar lexical features as the words from the training corpus. For each user, the preference vector is computed as the separate weighted averaged vector of all the

keywords, subreddits and domains they used. For example, the keyword preference vector for the user i is calculated as $Pk_i = \sum_{j=1}^{n} f_j K_j$, where K_j is an embedded vector of a keyword j used by the user, and f_j is the number of instances the keyword has been used. This way, we are able to identify the direction of the user interests. The preference vectors for the subreddits and domains are calculated in a similar fashion.

Sentiment features quantify the two dimensions of user's writing on Reddit, namely *polarity* and *subjectivity*. We calculate the means and the standard deviations of polarity and subjectivity of all user's posts and comments separately.

Frequency features quantify the user's activity by measuring the number of their posts, comments and subreddits together with the additional temporal measures of user's activity.

- *Post features* X_j^r, similarly to user features, consist of embedded vectors of keywords, subreddits and domains related to the post. Additionally, we use a single measure of sentiment polarity and subjectivity extracted from the post content. Furthermore, we include the info about the author of the post.
- *Combined features* X_k^c are the cosine and euclidean distances of user preference vectors and post-related embedded vectors. This way we quantify the similarity between the given post and user's interests in the past.

For the *comment→comment* problem we use a similar approach. Instead of the *post* features we calculate the similar set of *comment* features and concatenate it together with the user features and combined features. We ad an additional feature which represents the position of a comment in a discussion tree.

Twitter Model For Twitter, we focus on modeling the 'retweet' action of a user. At any instance, a user has access to multitude of tweets on the platform. In our simulation, we develop a classification model for 'retweet' action *tweet, user→retweet*: whether a tweet will be retweeted for a given (tweet,user) combination. This model is applied to all tweets that an agent has access to whenever it reaches the front of the event queue. The simulation selects one of the tweets with highest probability according to the model and creates a retweet event for this agent and tweet. The final model used for the task was a 'Word2Vec' model combined with a 'random forest' model.

Twitter Feature Engineering In order to build a predictive retweet model, we need true indicative features from textual data to help the model learn. There are two main types of retweet activity by users on the platform- 'normal retweet' and 'quoted retweet'. A normal retweet is when a user retweets the original tweet without any additions, while the quoted retweet has additions to the original tweet. So a quoted retweet can give increased information of overlap of the tweet with the user profile as compared to the normal tweet. This understanding is important to decipher the learning of the model. The feature space used for the model consists of 3 types of features: *user features, tweet features* and *combined features.*

- *User features:* We create user profile in terms of their historical tweet activity on the platform. The intent is to model the user's persona on the platform based on their past activities. There are 84 features describing a user. One set of features is focused on *user's profile* including user description, number of followers, number of friends etc. The other set is focused on the *tweet activity of a user* such as their average number of tweets per week, maximum number of user mentions per tweet, average retweet period etc. We also exploit hashtags used by users to model their interest in multiple topics floating on the platform in the form of tweets at any given time. The user description field has an interesting potential to define users' interests at a broader level.

  A retweet score is separately assigned to each of the connections of a user. Every tweet has an author and the authors' retweet score for the user will be added to the final set of features. This is to capture the hypothesis that a user is more likely to retweet a tweet from one of their connections with the highest retweet score.

- *Tweet features:* Tweets are described with features including time of the day, day of the week, hashtags used, user mentions, number of characters and several others.
- *Combined features:* User historical features are adjoined with features of the entire set of tweets the user can take an action on at a given time. 'Word2Vec' model (W) is used to find out the similarity of the user's profile with the tweet features. It is trained on all hashtags of the corpus along with user descriptions. A representing feature vector V_k is obtained for all the hashtags. We create an average hashtag feature vector for a user H_{u_i} based on the most frequently used hashtags of the user. The user description vector D_{u_i} is created for each user. H_{u_i} and D_{u_i} are then used to find similarity of the user features with tweets. For each hashtag (V_k) of each tweet(t), cosine similarity is calculated with the H_{u_i} and D_{u_i}.

Feature engineering equipped the data with multiple defining features as mentioned above. For the current model, we use hashtags as major input for Word2Vec features for both normal retweet and quoted retweet. The model is then trained with the given data of feature set, retweet labels. A separate development set is used to tune parameters to eventually obtain a model accuracy of 92% on the test data.

3.3 GitHub Inverse Tree (ITree) model

Consider the following task: given a user u_i and the repository r_j, predict the number of actions $y_{i,j}$ the user u_i will perform on a repository r_j. This translates to a typical regression problem and could be approached in multiple ways. However, in GitHub the target repository is often uncertain while the rate of user's actions is typically already known, since the users activity rate varies slowly, and it is easily predicted just by observing the rates in the past. The active users remain active and the new users usually start with a slow rate of actions. Thus we solve the inverse problem: given a user u_i and the number of actions y_i, predict the repository r_i that user will most likely perform an action on. To predict the repository, we build a model that narrows down our choice to the set of repositories that share the same set of features.

For the given vectors of input and target variables, the regression tree finds a mapping from the input variables to a finite number of groups of output variables such that the average within group squared error of the target variables is minimized. The IT traverses

the trained regression tree backwards and for a given target variable and set of known features, extracts the boundaries of the unknown features. A regression model relates y to a function of X, so that $y \approx f(X)$. In our case, $X = X^u \parallel X^r \parallel X^c$, and we need to solve $X^r = f(y, X^u, X^c)$.

The problem of predicting the set of target variables resembles the problem known as a multi-output regression [5, 11], known also as the multi-target [1, 2] or multi-response regression [21]. Multi-output regression models are built to simultaneously predict multiple real-valued target variables. The result is usually a vector of values. However, instead of predicting a single vector, the ITree algorithm outputs two vectors: the first one L_i with lower bounds and the second one U_i with upper bounds for each feature we want to predict. Finally the output is processed as a set of tuples where $Y_i = \{(l_1, u_1), \ldots, (l_m, u_m)\}$.

The ITree model works in the opposite direction compared to the "traditional" regression tree model. First, we build a training data set D of N instances such that each instance is characterized by an input vector of m real variables $X_k = (x_k^1, \ldots, x_k^m)$ and a single target variable y_k. The input vectors are built by concatenating the user features X_i^u, repository features X_j^r and combined features X_k^c so that $X_k = X_i^u \parallel X_j^r \parallel X_k^c$. Then, we build a regression tree able to predict the number of user–repository actions, which is the target variable y. During the simulation, we do not predict y, as it is already estimated based on the previous user's activity. However, we use the target variable y_i together with known features X_i^u and X_k^c and back-traverse the previously trained regression tree. By traversing the regression tree backwards, we extract the boundaries of unknown features X_j^r which describe the repository. Based on the lower and upper feature bounds, we select the set of repositories to choose from. The initial version of the model picks a single repository r_j from the set of chosen repositories uniformly at random and assigns the y_i actions from a user u_i to a repository r_j. For instance, assume it is known that user u_i will make z push actions. The ITree model will first calculate the feature boundaries of the repositories the user is most likely to *push*. This way we narrow down our selection to multiple repositories as the potential targets. Then, ITree model randomly chose z repositories from the narrow selection and simulates the *push* action of user u_i to chosen repositories.

3.4 Dual queue event scheduling and burstiness

The framework has two queues: one for scheduling agents own activation time (self-activation) and one for external signal or triggers. Events that are scheduled in the main event queue are executed as a discrete event schedule. Events from external signal queue represent triggers that activate events on the main queue. In this paper external queue was used for burst events. Use of the external signal queue is optional because agents can schedule their next events on the main queue using self-activation rate.

In this paper we discuss only one type of external triggers - burst events. In our experiments we observed that using only agent self-activation rate does not simulate burstiness of events well. For this reason we used the external signal queue to trigger bursts of events. Bursts were measured in training data. Sequence of bursts observed in training data was sampled replayed in simulation. Each burst

event was scheduled in the external signal queue. When a burst event is handled by the simulation controller, it creates multiple events associated with this burst and schedules them in the main queue. Size and length of each burst as well as participating users were sampled from bursts observed in training data. Kleinberg's algorithm [17] was used to detect bursts.

3.5 Horizon and limited visibility models

Limited visibility models determine what active discussions and events are visible to agents to react to. Formally we define the agent's horizon at a time step as all resources that are visible to an agent at the step. Depending on the platform it may be a list of repositories for GitHub, tweets and retweets for Twitter, posts and comments for Reddit that are visible to an agent. In our experiments we used resource age (time between when it was posted and when is viewed) to determine if it should be visible to an agent.

Resources from different platforms have different life spans. For GitHub we used all repositories observed during the training period (2-6 weeks of training data) and kept them during the test period (6 weeks). Twitter, Reddit and Telegram are more fast-paced social platforms. For example, the majority of Twitter discussions have a life span of 2 days.

We used the following algorithm to determine visible resources for each agent. Each information cascade (resource tree) stays visible x number of days where x is minimum between a fixed threshold (2 days) and estimated lifespan of the information cascade. Lifespan of the information cascade is estimated using training data. It is done as follows. When new information cascade is created (agent creates a root resource, e.g. original tweet), its metadata (e.g. features) is sampled from training data. These features include lifespan, which becomes an estimation for a lifespan of a new cascade.

4 RESULTS

4.1 Experiments and models

Our simulation produces a sequence of events that represent users' activity on Reddit, Twitter, and GitHub. The resulting sequence of events is then compared to the ground truth according to a set of metrics described below. The ground truth is the sequence of events recorded from the actual social networks for the simulated period, as illustrated in Fig 1. In this paper we discuss the following model configurations:

- Random null model - this does not use the simulation framework but instead replays the training data in the test time period, randomly shuffling the order of events.
- Base model - a model where each step in agent's decision workflow is executed using simple probabilistic model. Probability of each decision is based on frequency of that decision in the training data. For example, steps such as selecting action type or selecting a resource are separately determined using this approach.
- ML model (Twitter and Reddit) - an extension of the base model where for Reddit and Twitter platforms we use the ML model to determine how cascades grow (what tweets are retweeted and what posts are commented on). This is step 5 in Figure 2.

- Bursts model - an extension of the base model where in additions to self-activation agents are activated by burst events using the external signal queue as described in section 3.4. The approach samples from a set of bursts detected in the training data using Kleinberg's algorithm [17]. For each burst, we sample from a subset of agents that are activated during a very short period of time (within 30 minutes) after the burst.
- Combined model - a combination of the Bursts and ML models from above.
- ITree for GitHub model - an extension of the base model where GitHub agents use ITree models to determine action type and resource (step 3 in Figure 2)

The ML model, Bursts model, Combined model and ITree for GitHub model are all built on top of the base model. The ML and ITree models augment the agent's decision flow, and the Bursts model adds bursts in the event queue.

4.2 Evaluation metrics

In order to capture information spread across platforms we track *information units*, which are keywords, hashtags or associated URLs, depending on the platform. These are typically gathered from metadata, and one action or resource may be associated with multiple information units. In order to evaluate the predicted information spread across platforms, we measure the number of shares (e.g. number of retweets in Twitter, or comments in Reddit), the size of the audience (the number of unique users who shared), the speed of information spread, and its lifetime.

For model performance evaluation we use the following metrics:

- Temporal correlation of shares (audience sizes) - the Pearson correlation between the number of shares (audience sizes) time series is computed between all pairs of platforms. This distribution is computed for both simulation and the ground truth. Then the Kolmogorov-Smirnov test and Jensen-Shannon divergence are computed to measure similarity.
- Time delta - determines the number of hours it takes for a piece of information to appear on another platform. The Kolmogorov-Smirnov test and Jensen-Shannon divergence are computed to measure similarity.
- Correlation of shares (audiences) - the Pearson correlation between the activity (audience size) time series between all pairs of platforms. Measured as *RMSE*.
- Correlation of lifetimes - the Pearson correlation of the average lifetime for each platform, then correlation between simulation and the ground truth is computed. Measured as *RMSE*.
- Correlation of speeds - the Pearson correlation between speeds of information across platforms for each information unit, then correlation between simulation and the ground truth is measured. Measured as *RMSE*.

For burst measurements we use Kleinberg's algorithm [17] to detect bursts on all platforms (population level measurement). Each metric related to bursts is computed as the *RMSE* of the correlation between simulation and the ground truth. Figure 4 shows *RMSE* of correlation of the following distributions:
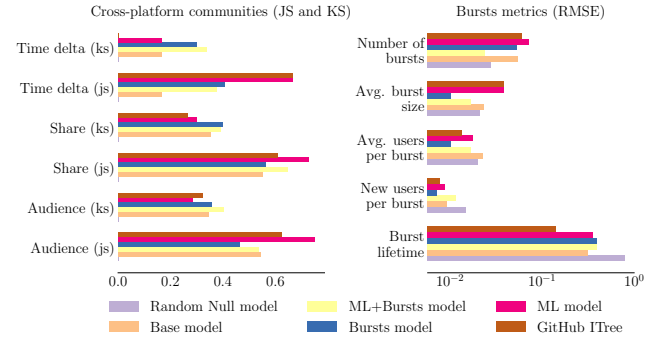
- Distribution of the number of bursts

Figure 4: Cross-platform community and burst metrics. Our models outperform the base model in the majority of metrics.
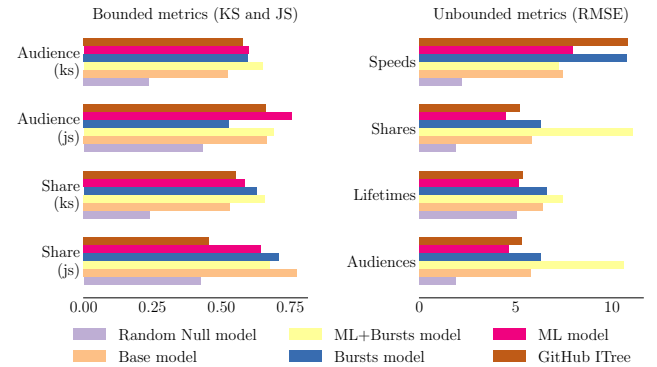

Figure 5: Cross-platform population metrics. Our models outperform both the base model and the random model in the majority of metrics.

- Distribution of the average burst size
- Distribution of the average number of users per burst
- Distribution of the number of new users per burst
- Distribution of bursts lifetime

In this paper we present measurements for two scales: population level and community level. Community is a subset of users grouped by a certain criteria (e.g. active on the same sub-reddit, use the same repositories or use similar information units, etc.). We identified several dozens of communities. Community level metrics were computed separately for each community and average is shown in Figure 4. Results for population level metrics (burst and cross-platform) are shown in Figure 5. Higher value are better in Figure 4 and 5 as we show $1/RMSE$.

We discuss these results in greater detail in below in Section 6.

5 RELATED WORK

There were several approaches proposed to study structure and user behavior on social media networks. In [4] authors explore various models to simulate a static structure of the social networks enabled by social media. The authors introduced several metrics for

network characterization generated by several agent-based simulations. Similar metrics were used in the DARPA SocialSim Challenge. Models discussed there focus on a static structure of the network and often limited to simulation of one social platform.

In recent years there has been evolving interest in studying cross-platform information spread (e.g. [13, 15, 18]). For example, in [18] authors studied a cross-platform spillover effects of viral videos. They found that when a video content appears on a lag platform it doubles its subsequent view growth on the original platform.

Simulation of information spread across-multiple platforms on the scale of tens of thousands and hundreds of thousands of agents can be computationally intensive. Previous work on scalability of agent-based simulation frameworks was done in [7, 8].

Applications of machine learning methods were also widely discussed for link prediction in social networks. There has been an extensive research done in the area of network embedding and it's application in link prediction [14, 16, 19, 22]. Wang et al [23] shows a link prediction problem as a two class discrimination problem and used machine learning approaches to learn prediction models. Link prediction is often used for recommending links on social networks [3].

Content-based link prediction methods have been developed and gained popularity among researchers. The large set of those models focus on predicting the activity in social networks based on combination of numerous content and user features [12, 25] Link-prediction and information spread prediction on social networks is often challenging because available training data can only provide positive feedback. Authors in [24] use retweet networks to address this problem (retweet network was used to reconstruct negative feedback).

6 CONCLUSION AND DISCUSSION

In this paper we present several agent-base simulation models that utilize machine learning in the agent's decision workflow. We also introduce a dual queue technique for discrete event scheduling that allows incorporating external event signals such as bursts. Our experiments on cross-platform information spread show that machine learning methods can be successfully used in multi-platform agent-based simulations, and they improve performance metrics compared to simple probabilistic models (base model and random null model).

In our experiments, all models perform significantly better in terms of the temporal correlation of cross-platform information spread than the random null model (see Figure 4 and Figure 5.). In fact, the random null model scores zeros across all community level metrics.

Both ML models and ITree model score on average 23% higher in Jensen-Shannon divergence of the temporal correlation of audiences. The results are similar for population and community level measurements, showing that the Reddit and Twitter ML models help agents to make a better (relative to base model) decisions about what tweets and posts should be retweeted. Also the GitHub ITree model helps make better decisions about which repositories agents interact with.

All models presented in the paper are data-driven, which allows us to build large-scale high-fidelity simulations. The results also suggest that predicting bursts of events is hard without external signals. Reproducing bursts using burst samples taken from training data can improve temporal metrics.

The base model and random null model perform relatively well in burst metrics. In most burst metrics the bursts model does not perform well compared to all other models. This suggest that re-playing sequence of bursts from the initial conditions data-set does not improve the metrics that measure burst properties and distributions. However, the combined model (ML + Bursts) scores the best across almost all population level cross-platform information spread metrics while the bursts model and ML model separately come second and third respectively in these metrics. This can be explained by the fact that explicit modeling of bursts (bursts model) affects all time-sensitive metrics. Introducing bursts alone did not improve the burstiness metrics but improved temporal correlations of content shares and audience sizes.

The results presented in this paper lay ground for the future research and extensions of the proposed models. This includes incorporating external signals that drive activation of the agents. It can potentially improve prediction of event bursts (reaction of a subset of agents on external signal). Machine learning methods can also be applied to other aspects of agent's decision workflow. For example, when agents generate new information cascades they have to choose information units. This choice can also be learned from observational data.

Finally, there are multiple ways how event visibility horizon can be implemented. In this paper we used a naive approach with an estimation of the resource life span (which determines how long it will stay within the horizon). What rules define visible events can affect agents decisions. The visibility horizon model can be also used as a tool for modeling social network users' cognitive load and limitations.

7 ACKNOWLEDGEMENTS

REFERENCES

[1] Timo Aho, Bernard Ženko, Sašo Džeroski, and Tapio Elomaa. 2012. Multi-Target Regression with Rule Ensembles. *Journal of Machine Learning Research* 13, Aug (2012), 2367–2407.

[2] Annalisa Appice and Saso Džeroski. 2007. Stepwise Induction of Multi-target Model Trees. In *Machine Learning: ECML 2007*. Springer Berlin Heidelberg, Berlin, Heidelberg, 502–509. https://doi.org/10.1007/978-3-540-74958-5_46

[3] Lars Backstrom and Jure Leskovec. 2011. Supervised Random Walks: Predicting and Recommending Links in Social Networks. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM '11)*. ACM, New York, NY, USA, 635–644. https://doi.org/10.1145/1935826.1935914

[4] Federico Bergenti, Enrico Franchi, and Agostino Poggi. 2011. Selected models for agent-based simulation of social networks. In *3rd Symposium on Social Networks and Multiagent Systems (SNAMAS 2011)*. 27–32.

[5] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. 2000. Top-down induction of clustering trees. (nov 2000). arXiv:cs/0011032

[6] Jim Blythe. 2012. A dual-process cognitive model for testing resilient control systems. In *2012 5th International Symposium on Resilient Control Systems*. IEEE, 8–12. https://doi.org/10.1109/ISRCS.2012.6309285

[7] James Blythe, Emilio Ferrara, Di Huang, Kristina Lerman, Goran Muric, Anna Sapienza, Alexey Tregubov, Diogo Pacheco, John Bollenbacher, Alessandro Flammini, et al. 2019. The DARPA SocialSim Challenge: Massive Multi-Agent Simulations of the Github Ecosystem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1835–1837.

[8] J. Blythe and A. Tregubov. 2018. FARM: Architecture for Distributed Agent-Based Social Simulations. In *Massively Multi-Agent Systems II*, Donghui Lin, Toru Ishida, Franco Zambonelli, and Itsuki Noda (Eds.). Springer International Publishing, 96–107.

[9] Jim Blythe and Alexey Tregubov. 2019. FARM: Architecture for Distributed Agent-Based Social Simulations. Springer, Cham, 96–107. https://doi.org/10.1007/978-3-030-20937-7_7

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. (jul 2016). arXiv:1607.04606

[11] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larrañaga. 2015. A survey on multi-output regression. *WIREs Data Mining Knowl Discov* 5 (2015), 216–233. https://doi.org/10.1002/widm.1157

[12] Ethem F. Can, Hüseyin Oktay, and R. Manmatha. 2013. Predicting retweet count using visual cues. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management - CIKM '13*. ACM Press, New York, New York, USA, 1481–1484. https://doi.org/10.1145/2505515.2507824

[13] Joseph D O'Brien, Ioannis K Dassios, and James P Gleeson. 2019. Spreading of memes on multiplex networks. *New Journal of Physics* 21, 2 (2019), 025001.

[14] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* (2018). https://doi.org/10.1016/j.knosys.2018.03.022 arXiv:1705.02801

[15] Anna-Katharina Jung, Milad Mirbabaie, BjÃrn Ross, Stefan Stieglitz, Christoph Neuberger, and Sanja Kapidzic. 2018. Information Diffusion between Twitter and Online Media. (2018).

[16] Seyed Mehran Kazemi and David Poole. 2018. Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems*, Vol. 2018-December. Neural information processing systems foundation, 4284–4295. arXiv:1802.04868

[17] Jon Kleinberg. 2003. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery* 7, 4 (2003), 373–397.

[18] Haris Krijestorac, Rajiv Garg, and Vijay Mahajan. 2019. Cross-Platform Spillover Effects in Consumption of Viral Content: A Quasi-Experimental Analysis Using Synthetic Controls. *Available at SSRN 3011533* (2019).

[19] L. L. Linyuan and Tao Zhou. 2011. Link prediction in complex networks: A survey. (mar 2011), 1150–1170 pages. https://doi.org/10.1016/j.physa.2010.11.027 arXiv:1010.0725

[20] F. Mordelet and J.-P. Vert. 2014. A bagging SVM to learn from positive and unlabeled examples. *Pattern Recognition Letters* 37 (feb 2014), 201–209. https://doi.org/10.1016/J.PATREC.2013.06.010

[21] Timo Similä and Jarkko Tikka. 2007. Input selection and shrinkage in multiresponse linear regression. *Computational Statistics & Data Analysis* 52 (2007), 406–422.

[22] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale information network embedding. In *WWW 2015 - Proceedings of the 24th International Conference on World Wide Web*. Association for Computing Machinery, Inc, 1067–1077. https://doi.org/10.1145/2736277.2741093 arXiv:1503.03578

[23] Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. 2015. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences* 58, 1 (01 Jan 2015), 1–38. https://doi.org/10.1007/s11432-014-5237-y

[24] Tauhid R Zaman, Ralf Herbrich, Jurgen Van Gael, and David Stern. 2010. Predicting information spreading in twitter. In *Workshop on computational social science and the wisdom of crowds, nips*, Vol. 104. Citeseer, 17599–601.

[25] Qi Zhang, Yeyun Gong, Jindou Wu, Haoran Huang, and Xuanjing Huang. 2016. Retweet Prediction with Attention-based Deep Neural Network. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16*. ACM Press, New York, New York, USA, 75–84. https://doi.org/10.1145/2983323.2983809