# Social Media Time Series Forecasting and User-Level Activity Prediction with Gradient Boosting, Deep Learning, and Data Augmentation

by

Fred Mubang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Lawrence O. Hall, Ph.D.
Adriana Iamnitchi, Ph.D.
John Skvoretz, Ph.D.
Yu Sun, Ph.D.
Mingyang Li, Ph.D.

Date of Approval:
October 19, 2022

Keywords: Graph Embedding, Machine Learning, Neural Networks, XGBoost, SMOTE

## Dedication

I dedicate this dissertation to my late father, John Mubang, who was always supportive and emphasized the importance of a good education. *Bonum certamen certavit et vicit genus. Fidem servavit.*

I also dedicate this dissertation to my mother, Angeline Mubang; my brothers, JJ and Eric; and my love, Lindsay Patenaude. I am grateful to have all of you in my life.

**Acknowledgments**

I would like to thank my advisor, Dr. Lawrence Hall, for his great mentorship and guidance over the years. He helped me transition from the world of musical notes to the world of 0's and 1's.

Thank you to my committee members, Dr. Adriana Iamnitchi and Dr. Skvoretz for your guidance throughout all these years on Socialsim and how to become a better researcher and thinker. I also thank my committee members Dr. Yu Sun and Dr. Mingyang Li for all of their guidance and support.

Thank you to all of the collaborators I have had throughout the years. They are Renhao Liu, Kin NG Lugo, Sameera Horawalavithana, Abhishek Bhattacharjee, Anthony Hernandez, and Nazim Choudhury.

Thank you to DARPA, Sloan Foundation, and Bernard Batson. This work was supported by the DARPA SocialSim Program and the Air Force Research Laboratory under contract FA8650-18-C-7825, and Alfred P. Sloan Foundation University Center of Exemplary Mentoring (UCEM) under Grant G-2020-12674. I also thank Pacific National Northwest Laboratory (PNNL), and Leidos for their support through out the SocialSim project for providing data and evaluation code.

I would like to thank my late father, John Mubang, who was a larger-than-life figure who always stressed the importance of a good education. His spirit will always live on. I also thank my mother Angeline Mubang, my brothers, JJ and Eric; and my love and partner Lindsay Patenaude. I am blessed to have all of you in my life, and you inspire me.

## Table of Contents

## List of Tables

## List of Figures

## Abstract

In the overall history of technological innovations, social media has only existed for a brief time, however its influence is undeniable. Researchers have found that it can be used to influence elections, spread health misinformation, and aid with financial pump-and-dump schemes. Keeping all this in mind, it is clear that more research is needed to predict the spread of information on social media in order to combat its malicious use.

To that end, in this dissertation, we explore the use of Machine Learning algorithms to perform time series forecasting and user-level activity prediction in social media. We address the different types of challenges that come with predicting social media activity such as (1) accounting for the differences in user engagement among different social media platforms, (2) identifying the data required for accurate predictions, (3) selecting the appropriate prediction framework, and (4) metric selection.

We address the aforementioned challenges in multiple ways. Firstly, we introduce an end-to-end simulator called the Volume Audience Match simulator, or VAM. VAM is comprised of two modules called the (1) Volume Prediction Module and (2) the User-Assignment Module. VAM performs both time series prediction and user-assignment. It predicts the overall volume time series of (1) new users, (2) old users, and (3) activities. It then assigns the predicted actions to both old and new users over time.

We evaluate VAM's predictive prowess on 2 geo-political datasets, the Venezuela 2019 Twitter dataset (Vz19), and the China-Pakistan Economic Corridor Twitter dataset (CPEC). We show that VAM outperforms various traditional time series baselines for the Volume-Prediction task, specifically the Persistence Baseline, ARIMA, ARMA, AR, and MA. We show that it outperforms the Persistence Baseline and several state-of-the-art embedding

methods for the user-assignment task, specifically, tNE-node2vec-S, tNE-node2vec-H, and tNE-DeepWalk.

We also find that exogenous features from Reddit and YouTube improve VAM's time series prediction accuracy. Furthermore, we perform an in-depth analysis of VAM's performance using a wide-range of metrics that analyze many dimensions of the resulting predictions, such as magnitude, burstiness, temporal pattern matching, user-level prediction accuracy, and overall network structure. Lastly, we compare the XGBoost-based VAM models to the Recurrent Neural Network-based (RNN) VAM models and find that the XGBoost models are much faster to train and more accurate. This is notable because RNNs are one of the most frequently used machine learning algorithms for social media prediction. Perhaps this insight will prompt other researchers to consider using XGBoost for their own modeling purposes instead of RNNs.

We also introduce a variant of VAM that performs data-augmentation called SMOTER-VAM. This version of VAM utilitzes data-augmentation as a prepreprocessing step via 2 different algorithms: SMOTER-Binning (SMOTER-B) and SMOTER-NB (No-Binning). These two algorithms are variants of the SMOTER algorithm (Synthetic Minority Oversampling Technique for Regression).

Two different VAM models are trained on the 2 augmented datasets. We found that using the SMOTER-B and SMOTER-NB algorithms improve VAM's performance on time series prediction, especially on low-volume topics. These SMOTER variations are also generalizable to any machine learning algorithm and any dataset that has multiple-continuous outputs. Therefore, these variations can have many potential applications beyond VAM or social media time series prediction usage.

Lastly, we analyze the differences between 2 commonly used baselines within the realm of social media prediction, the Persistence Baseline and ARIMA. We evaluate their performances on different datasets and in different contexts, and through our analysis, we better understand which situations the baselines are useful and why.

# Chapter 1: Introduction

## 1.1 Motivation

Social media's vast influence is apparent. Recent research has shown its effect in many aspects of society. The authors of [10] analyzed UK voting data from the 2015 and 2017 elections and found that Twitter-based campaigning helped UK politicians win votes in these elections. The authors of [59] note multiple instances in which the spread of COVID-19 misinformation may have led to needless deaths. For example, in Nigeria, health officials found several cases of overdose of chloroquine (a drug used to treat malaria). These deaths were found to have occurred after various news media outlets and social media sites erroneously cited chloroquine as a treatment against COVID-19 [59].

Lastly, the authors of [37] discuss the rise of cryptocurrency pump-and-dump groups on social media sites such as Discord and Telegram. The authors note that the number of members in some of these groups was as high as 200,000, with smaller groups still running about 2000. These groups have artificially raised the price of various cryptocurrencies by up to 950% to the detriment of unwitting investors.

Clearly, in order to combat the spread of such misinformation and disinformation, it would be ideal to predict the future phenomena related to any discussion on any social media platform. Specifically, from a macroscopic perspective, it would be of interest to know the future overall volume time series of (1) posts (or activities), (2) new users, and (3) old users, on a given social media platform. We refer to the prediction of these 3 time series as the Volume Prediction Task. Furthermore, on a more microscopic level, it would be of interest to know what the user-to-user interactions (edges) over time would be. This would

apply to a specific social media platform, for a particular topic, at a specific time. We refer to the prediction of the user-to-user interactions as the User Assignment Task.

## 1.2 Challenges

There are, of course, various challenges involved with the Volume Prediction and User-Assignment tasks in social media. They are as follows.

### 1.2.1 User Engagement Differences

There are challenges in how user engagement is configured across various social media platforms. The ramifications of these differences is that the task of building a general social media prediction model that can account for the different idiosyncrasies across each platform becomes more difficult.

Firstly, the types of content each platform has differs. For example, on Twitter, users interact with micro-blogs (called tweets), whereas on Instagram, users interact with pictures. On YouTube and Tiktok, users interact with videos, and so on.

Beyond content, different social media platforms are driven by different types of user-to-user interactions. For example, user engagement on Twitter is heavily driven by its "retweet" feature, which is how users share posts to others. User engagement on Reddit, on the other hand, is heavily driven by users replying to posts.

Lastly, each platform differs in terms of the user demographics. For example, 31% of all Facebook users are between the ages of 25 to 34, whereas on Linkedin that percentage is 59% [31]. As a result, the topics posted about or discussed on each platform may differ greatly.

### 1.2.2 Identifying and Acquiring the Required Data

There are multiple challenges in social media prediction related to identifying and acquiring the data required for accurate predictions. One such issue is that social media companies heavily restrict access to historical data. In 2015, the UK Data Service commissioned the

Digital Preservation Coalition (DCP) to perform a 12-month study into the preservation of social media [60]. The DCP found that while most social media platforms provide access to their API, they restrict the amount of data that can be requested and how often through rate limits [60]. As a result, researchers or developers may only have access to small amounts of historical data. This can make it difficult to properly train models.

Also when modeling social media activity, one must also consider whether to use external sources of data, and which sources should be used. Previous works have shown that exogenous triggers from other social media platforms can improve predictive performance on another. For example, the authors of [39] and [40] found that Reddit and Twitter exogenous features helped predict activity in Github. The authors of [46] found that different news and social media exogenous sources can improve time series prediction accuracy for certain Twitter and YouTube topics.

### 1.2.3 Selecting the Appropriate Prediction Framework

As of January 2022, the number of social media users worldwide was 4.62 billion [31]. That is more than half of the entire population of earth. Furthermore, as of 2022, Twitter's average daily active users is 217 million. Facebook's average daily active users is over 2.9 billion [31].

With these statistics in mind, one can see that a prediction framework needs to not only be accurate, but also practical and scalable to account for the large amounts of users that could participating in a given topic discussion. One needs to keep in mind the time required to train and test a model. For example, if one requires a model to predict the number of Twitter activities in the next 24 hours, but the model of interest takes 48 hours to train and test, the model would be useless, regardless of how accurate it is.

Furthermore, although there are a large number of users on social media networks, most of them are infrequently active, which must be accounted for as well when creating a model. This can be problematic because a model trained on sparse data will erroneously predict

that users performed no actions when in fact they did. This was the issue that the authors of [30] had with their user-prediction models.

There is also the challenge of predicting the creation of new users. Even if the number of new users can be predicted, how can one determine what their patterns of behavior will be? With old users, one may simply be able to observe their past behavior to predict future behavior, but new users have no past behavior to model to begin with. Most user-level prediction approaches do not model new user activity, and those that do suffer from scalability issues for networks with large numbers of users, such as [39] and [66].

Due to the aforementioned challenges that user data can present, selecting the appropriate modeling approach for social media prediction is important. The benefits and tradeoffs of various approaches must be considered. For example, many social media prediction approaches model user interactions as adjacency matrices [51, 68, 21, 55, 30, 56]. The benefit of adjacency matrices is that a large amount of information is passed to a model - both temporal and edge-level information. However, adjacency matrices are computationally complex in terms of space, and the sparsity of adjacency matrices may cause many erroneous "no-activity" user predictions. Furthermore, by their very nature, adjacency matrices do not allow for the prediction of new users.

Agent-based approaches [8, 24, 45] and hand-crafted-feature-based approaches [67, 3, 21, 44] avoid the space and sparsity issues of adjacency-matrix-based approaches. Also, they can be used to predict new user activity. However, they can still suffer from time complexity issues. Furthermore, they rely upon the modeler to create and select the best of rules or features for prediction, which can be an arduous task. Embedding-based approaches, on the other hand, are powerful in that they automate the feature-creation process, however they can take a very long time to train [15, 50, 27, 58].

In terms of specific machine-learning models, Recurrent Neural Networks are among the most frequently used models for social media activity prediction in both the Volume Prediction and User Assignment tasks [40, 33, 39, 30, 56, 58]. However, RNNs, like vanilla

neural networks, suffer from long training times. This can be problematic if one wishes to create models that can perform next-day user activity predictions.

#### 1.2.4 Selecting the Appropriate Metrics for Model Evaluation

Lastly, it is important to use appropriate metrics that capture multiple important aspects of both the overall volume predictions, as well as the user-level predictions. Most literature regarding social media time series regression strongly focus on the RMSE, MSE, MAE, MAPE, and/or SMAPE metrics, which measure the exact volume of users or activities in each time step [46, 41, 29, 38]. While these metrics are useful, they can fail to capture other important aspects of a predicted time series such as the "burstiness" or "spikiness" of a predicted time series. Capturing spikes or bursts are important because they can be indicators of potentially malicious activity, such as a large increase in the sharing of disinformation or misinformation.

At the user-prediction level, there are two types of users that can be predicted - old users and new users. Old users are users that have been previously seen in the network, so it is trivial to find and apply metrics that measure how well they have been predicted such as the F1 score, AUC, Precision, and Recall metrics, among many others [50, 58, 27, 35, 64, 38].

Measuring new user predictive success is a more difficult task. One obviously does not know the names of new users in the ground truth, so one must resort to measuring macroscopic aspects of the simulated network in order to ascertain predictive success.

### 1.3 Contributions

We address the aforementioned challenges throughout this dissertation in the following ways.

Firstly, we introduce the Volume Audience Match simulator, of VAM. It is an end-to-end simulator of social media activity over time. It is comprised of two modules, a (1) Volume Prediction (VP) Module and (2) User Assignment (UA) Module. The Volume Prediction

Module predicts the overall volume time series of new users, old users, and activities in a given social media network. The User Assignment Module uses the VP Module predictions and uses those to predict the user-to-user interactions of old and new users over time in a social media network. We show that VAM outperforms multiple baseline and state-of-the-art methods for both the VP and UA tasks. Furthermore, we show that VAM is quick to train and test on multiple networks with millions of users, showing that it is quite scalable.

We show that VAM outperforms state-of-the-art embedding approaches for both the VP and UA tasks. Furthermore, we even found that the embedding models are strongly outperformed by a simple Persistence Baseline on the UA task. This is notable because embedding-based models are widely used for social media user-level activity prediction [15, 50, 27, 58]. This might inspire researchers to re-evaluate their potential use of embedding models for future research both in the social media domain, and perhaps network prediction in general.

An application of VAM to the real-world would be to predict the potential spread of misinformation and disinformation on social media networks. The VP Module could predict the volume time series of new users, old users, and activities related to malicious topics. The UA Module could then be used to predict the activity of old and new users that participate in these malicious topics. The influence of specific users over the network could be predicted, allowing for potential intervention of malicious content spread.

In this dissertation, we also show the efficacy of XGBoost models in the domain of time series forecasting for social media data. We show that they are faster to train and more accurate than Recurrent Neural Network (RNN) models. The XGBoost models took anywhere from 3 to 7 minutes to train, whereas the RNN models took from 3 to 4.5 hours to train. This is notable because RNNs are among the most frequently used prediction algorithms for time series prediction in social media prediction literature [40, 33, 39, 30, 56]. In a real-world setting, it is important to have models that can quickly be trained so that they can perform predictions in a timely manner. This is especially important if one wishes to

have models that can perform next-day predictions. Furthermore, the fact that such strong performance was acheived with XGBoost models may cause future researchers to reconsider the usage of RNNs for prediction tasks in both social media prediction and domains outside of it as well.

We also perform a thorough analysis of the time series VAM's Volume Prediction used to perform time series forecasting. We do this by clustering the time series based on various time series attributes. Through this analysis, we gain a better understanding of the types of time series VAM performs better or worse on. The type of analysis we perform can be applied to any sort of time series forecasting model and dataset. The analysis is useful because it presents more explainability to model performance. Most previous literature for social media time series prediction do not show any sort of analysis indicating what types of time series their models perform worse or better on [46, 40, 39, 56, 30, 66, 36].

We also introduce two variations of the SMOTER algorithm for multiple-output time series. We then show the usefulness of these SMOTER data augmentation within the context of social media time series prediction, which no other social media time series prediction literature show [46, 40, 39, 56, 30, 66, 36]. We show that SMOTER data augmentation improves VAM's time series prediction accuracy. Note that even though we show that SMOTER helped improve the prediction accuracy of an XGBoost model, we note that it can be applied to any machine learning model as a data pre-processing step. Furthermore, our SMOTER variations can be applied to any dataset with multiple continuous-value target variables, making the algorithms quite versatile.

## 1.4 Dissertation Outline

The outline of this dissertation is as follows.

### 1.4.1 Chapter 2 - Background

Chapter 2 is the Background Section. In it, we discuss background information and previous literature regarding social media prediction.

### 1.4.2 Chapter 3 - VAM Simulator Introduction and Vz19 Dataset

Chapter 3 introduces the Volume Audience Match Simulator, or VAM, which is the aforementioned approach to both the Volume Prediction and User Assignment tasks. We evaluate VAM's performance on the Vz19 dataset and show that VAM strongly outperforms multiple statistical and state-of-the-art approaches across a myriad of metrics for the time series prediction task. Specifically, we compare VAM to the Auto Regressive Integrated Moving Average (ARIMA) models and its variations: ARMA, AR, and MA. We also compare VAM to the Persistence Baseline. In terms of state of the art approaches, we compare VAM to 3 tNodeEmbed models: tNE-node2vec-H, tNE-node2vec-S, and tNE-DeepWalk. VAM outperforms all of these approaches.

For the Volume Prediction task, we compare the use of XGBoost models vs. Recurrent Neural Network models (RNNs) as VAM's "backends". We show that the XGBoost versions of VAM are more accurate and faster to train the RNN versions. This is noteable because RNNs have been used extensively in previous social media prediction literature, while XGBoost has not.

We provide an analysis of the use of social media platform features to determine what helps VAM achieve the best time series prediction performance in Twitter. We show that using features from activity on Reddit improves predictions of Twitter activity. Lastly, we show that VAM greatly outperforms the baseline and state-of-the-art models in the User Assignment task. We then evaluate VAM's performance over 3 measurement categories:

(1) old user prediction success, (2) the capturing of the ground truth indegree, and (3) the capturing of the ground truth Page Rank distribution.

### 1.4.3 Chapter 4 - VAM and the CPEC Dataset

In Chapter 4 we evaluate VAM's performance on the CPEC dataset. By showing VAM's strong performance on another dataset, we further show that VAM has the potential to generalize across multiple datasets, and not simply be domain-specific to the Vz19 dataset.

We compare VAM against the ARIMA, ARMA, AR, MA, and Persistence Baseline models. We further study the use of exogenous features, and find that YouTube features, in addition to Reddit features aid with the Volume Prediction task (not just Reddit features, unlike in the previous chapter). Lastly, we analyze the ground truth time series of both the CPEC and Vz19 datasets to better understand what types of time series VAM performs well or poorly on. We find, across both datasets, that time series with higher skewness and coefficients of variation are harder to predict.

### 1.4.4 Chapter 5 - SMOTER-VAM

In Chapter 5 we apply the Synthetic Minority Oversampling Technique for Regression, or SMOTER, to the CPEC Twitter dataset from 4. We introduce 2 variations of SMOTER: *SMOTER-Binning* (*SMOTER-BIN*) and *SMOTER-Non-Binning* (*SMOTER-NB*).

In the SMOTER-BIN algorithm, the Frobenius Norm and binning is used to convert the outputs of the Volume Prediction samples into 4 classes. Then, SMOTER was applied to the 3 minority classes. For the SMOTER-NB algorithm, SMOTER is applied uniformly to all samples without regard to minority or majority classes. VAM is trained on these different datasets, and the results are analyzed.

We found that both SMOTER-VAM models outperformed the regular VAM model in different settings. Furthermore, we created multiple ensemble models between the SMOTER-VAM and the regular VAM models to understand what types of time series it is most

appropriate to use augmentation and when not to. Through this experiment, we found that the SMOTER-VAM models performed particularly well on time series with low-volume. This suggests that SMOTER-based data augmentation algorithms help improve prediction accuracy on low-volume time series.

### 1.4.5 Chapter 6 - Time Series Baseline Analysis

In Chapter 6, we analyze the performance differences between two common baseline models used in the social media literature - the Persistence Baseline and ARIMA. We compare the performance across 6 datasets in two settings - *long-term* vs. *short-term* configurations. We find that the Persistence Baseline is the more powerful baseline for short-term predictions, especially in the measurement of scale of activity (as measured by S-APE), and "burstiness" of activities (as measured by Skewness Error and Volatitlity Error). We find that in the long-term, ARIMA is preferred if one wishes to capture overall exact-timing, as measured by the RMSE and MAE metrics.

### 1.4.6 Chapter 7 - Future Work and Conclusion

Chapter 7 contains a discussion regarding future work and closing thoughts.

# Chapter 2: Background

In this chapter, we discuss previous literature related to user-level social media prediction. First, preliminaries are discussed. Next, background on social media prediction approaches are given. This information is needed to understand chapters 3 and 4. Then, we discuss previous literature on SMOTER for social media time series, which is needed for Chapter 5. Lastly, we provide background on the Persistence Baseline and ARIMA models, which are needed for Chapter 6

## 2.1 Preliminaries

In this section, the various types of machine learning models used in this work are discussed.

### 2.1.1 XGBoost

XGBoost is an ensemble model of CART trees (Classification and Regression Trees). It utilizes boosting to train an ensemble on a given dataset. Boosting is an ensemble method that reduces bias and variance. It is based on combining multiple weak learners to act as one strong learner [16].

Similar to other boosting algorithms, XGBoost trains on a given dataset by iteratively building weak learners and creating a "strong learner" ensemble. When new weak learners are added to the current ensemble, they are weighted in a way that is related to the weak learners' accuracy. The "learners" in this context, are CART trees.

Several hyperparameters are predefined by the user to control the size of each CART tree. The *max_depth* hyperparameter determines the maximum height that any given CART tree

can have. It is a *prepruning* hyperparameter. The higher this value is, the more complex the XGBoost model will become (and more likely to overfit). The $\gamma$ (gamma) hyperparameter is a value that determines the minimum loss reduction required in order for a CART tree to continue to split on the leaf node of a tree. Note that the higher the gamma value is, the less complex the XGBoost model will be. Obviously, if this value is too high, it can lead to underfitting [16].

After a weak learner is added, the data weights are readjusted, known as "re-weighting". Incorrectly predicted input data gain a higher weight and examples that are predicted correctly lose weight. Thus, future weak learners focus more on the examples that previous weak learners predicted incorrectly.

XGBoost specifically is a Gradient Boosting algorithm, which is a type of boosting algorithm that optimizes a differentiable loss function in order to create the best ensemble of weak learners [16].

### 2.1.2 Neural Networks

A neural network is a machine learning algorithm inspired by the architecture of the biological brain. It is comprised of stacks of "layers", with each layer containing a collection of units called "artificial neurons". The neurons in a given layer are connected to neurons to the succeeding layer, and these connections have "weights". The neurons in each layer receive and send signals to neurons in the succeeding layers. The final layer of a neural network is an output layer, which contains 1 or more outputs, depending on how many values one wishes to predict [65].

### 2.1.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are types of neural networks made for predicting temporal or sequential data. They have a similar architecture to vanilla neural networks, except their outputs at a timestep $t$ are used as additional inputs at timestep $t+1$. By using

this methodology, RNNs are trained to model sequences of values over time. This makes them ideal for forecasting time series, which is what we use them for in Chapter 3. RNNs are among the most frequently used machine learning approaches for social media activity prediction as shown in [40, 33, 39, 30, 56, 58].

### 2.1.4 Long Short Term Memory Neural Networks

In principle, vanilla RNNs are capable of modeling temporal sequences of data. In practice, however, they are quite limited. The authors of [25] note that vanilla RNNs can only learn to model sequences of about 5-10 discrete steps, due to a phenomenon known as the "exploding or vanishing gradient problem". This gradient problem occurs because during the back propagation phase of the RNN-training process, the long-term gradients used to update the model weights can either tend to infinity ("explode") or tend toward zero ("vanish") [25].

The authors of [32] created the Long Short Term Memory RNN (LSTM) to alleviate this issue. LSTMs mitigate the vanishing/exploding gradient problem by incorporating 3 mechanisms in their design called "gates", specifically they are called the *input*, *output*, and *forget* gates. The authors of [26] note that these 3 gates provide continuous analogues of write, read, and reset operations found in the memory chips in a digital computer. LSTMs have been found to perform much better than vanilla RNNs. The authors of [25] found that they were able to learn sequences of about 1000 discrete steps, which are much higher than the 5-10 steps of vanilla RNNs.

### 2.1.5 Gated Recurrent Unit Neural Networks

The Gated Recurrent Unit (GRU) neural network is another type of RNN that also mitigates the vanishing/exploding gradient problem, similar to LSTMs. However, unlike LSTMs, GRUs have just 2 gates - called the "update" and "reset" gates. As a result, GRUs have less parameters and are faster to train than LSTMs, assuming all other hyperparameters

are the same [20]. They have even been found to perform better than LSTMs on smaller datasets [20].

### 2.1.6 Transformer Neural Network

A Transformer Neural Network is another type of neural network for processing sequential data, similar to RNNs. However, where Transformers differ is that they process the input data all at once, whereas RNNs process the values of a sequence one at a time. Transformers are able to do this using what are called "attention mechanisms". Transformers are quite popular for Natural Language Processing tasks, such as language translation [63].

### 2.1.7 BERT

The Bidirectional Encoder Representations from Transformers (BERT) is a machine learning model for Natural Language Processing (NLP) [22]. As its name suggests, its architecture is very similar to the original Transformer Neural Network. BERT is useful when trying to automate the labelling of a large number of documents. In this work, BERT models were used to label the social media posts used to train our models. The inputs were the text of a post and the output was the most likely topic the post belonged to.

## 2.2 Background on Prediction Methods for Social Media Data

A large portion of this dissertation focuses upon the use of the Volume Audience Match Simulator (VAM) for the task of social media prediction. Because of this, in this section, we discuss previous work related to social media prediction methods. Specifically, there are two main types that we focus upon: (1) *General Popularity Prediction* and (2) *User-Level Prediction*. The General Popularity Prediction Section gives context to VAM's Volume Prediction module, and the User-Level Prediction section provides context to VAM's User-Assignment module.

### 2.2.1 General Popularity Prediction in Social Media

The term "general popularity prediction" refers to the prediction of the overall future volume of activities in social media networks. In these works, the user-level activity prediction is not considered.

The TAP model of [46] uses LSTMs to predict future Twitter and YouTube time series. They use exogenous news article, ACELD, and Reddit features, and find that certain platforms improve predictive accuracy for certain topics. The VAM and TAP models differ in that VAM predicts new users and old users, in addition to activities. Furthermore, VAM also predicts user-to-user edge interactions.

VAM also differs in that it was trained and tested with multiple "backend" models, specifically, XGBoost, and 4 different RNN models (LSTMs, GRUs, Bidirectional-LSTMs, and Bidirectional-GRUs). It was found that the VAM-XGBoost models were much faster to train and more accurate than the RNN-based models. Lastly, the VAM models in this work are also shown to perform strongly in predicting volatile and assymetric time series, as evident through the Volatility Error and Skewness Error metrics shown in Chapters 3 and 4.

The work [41] utilizes LSTM neural networks to predict the popularity of an event before its start date. The authors define popularity as the number of tweets that discuss an event. A limitation of this work is that it only predicts tweets related to a specific event. Tweets that discuss the consequences or aftermath of an event are not predicted. Also, since this model predicts only tweets related to pre-defined future events, it is limited in what it can predict. In this dissertation, we use our VAM models to predict tweets related to various topics, which are more broad than events. This more broad level of popularity prediction allows us to predict tweets related to spontaneous events, such as the "protests" or "violence" topics of the Vz19 dataset. Events such as protests, or violent outbreaks tend to be spontaneous and often do not occur at a predefined time. Lastly, note that unlike the VAM models introduced

in the dissertation, the event-prediction models of [41] do not predict the future volume of old and new users.

The authors of [36] used various regression models to predict the future volume of users in Facebook, Twitter, and Linkedin . They used various curve fitting models such as Polynomial, Logarithmic, and Exponential Regressions. The models in this work are limited in that they only perform univariate time series prediction, meaning, they only use the past volume of social media users to predict the future volume of users. They do not utilize other internal or external platform features for prediction. In contrast, the VAM models in this dissertation utilize multiple internal and external time series to predict multiple output time series: the (1) number of new users, (2) the number of old users, and (3) number of activities on a given social media platform. Furthermore, the authors of [36] found that only their simple Linear (or Auto-Regressive) model outperformed the Persistence Baseline (which they called the "naive" model). The Polynomial, Logarithmic, and Exponential Regression models failed to do better. The VAM models in this dissertation outperformed their respective baselines across multiple metrics.

In [6], the authors used LSTMs to predict bursts of Github activity by utilizing exgogenous features from Reddit and Twitter. A limitation of this work however, is that their LSTM models formulate the problem as that of binary classification ("burst" or "no burst"). It would be of interest to know the volume of activities in a burst, which is what our VAM models predict in this work.

Lastly, there are cascade-based popularity prediction methods such as DeepCas [38] and DeepHawkes [12]. They both use GRU neural networks to predict the future number of posts in a social network cascade at time $T + 1$ given the initial size of the cascade at time $T$. While both methods performed well at the task, they differ from the VAM models in this work in terms of prediction task. These cascade methods only predict the size of old user cascades, whereas the VAM models in this work predict the overall volume of activities, new users, and old users at the broader granularity of topic and not just individual cascades.

### 2.2.2 User Level Prediction Category #1: Decompositional

Next, there are *Decompositional* prediction approaches. These include the approaches that break the main prediction task into smaller subtasks. There are 2 ways that the prediction tasks were decomposed in the literature. They are the *Clustering-Based* and *Volume-to-User* based approaches.

#### *2.2.2.1 Clustering-Based Decompositional Approaches*

In the *Decompositional Clustering-Based* approach, user-repository (repo) pairs are predicted by first clustering the users into different categories based on 1 or more attributes, and then using these clusters to make more informed predictions of which user-repo pairs were active at time $T + 1$.

For example, the Archetype-ABM [53] is a framework that aims to predict user-repository activity through clustering and user archetypes. In this approach, K Means clustering was used to divide users into 16 different groups based on their average monthly activity for 14 different Github events [53]. These clusters were known as "archetypes". These archetypes were then used to predict the actual users, repositories, and events at some future timestep $T + S$.

Archetype-ABM was used to predict a month of future user-repo activity on Github. However, it was only compared to a historical average baseline. Furthermore, its predictive success was only evaluated at a month-granularity for 1 month. This is quite a large granularity, with only 1 timestep measured (1 month). In this work, we evaluate our VAM models at the hourly granularity over multiple days, so we have many timesteps of results that are analyzed. Furthermore, we compare VAM to multiple baselines and models.

Another clusering-based approach is the Proposed Community Features Model (PCFM) [28], which utilizes community clustering to predict user-to-repo activity. Each community is defined using a topic based approach using the profiles of the GitHub repositories in order

to generate a fixed set of communities. Some examples of topics used include programming languages, operating systems, and profile keywords [28].

The PCFM model was shown to outperform multiple baseline models across various metrics. However, the limitation of this model is that it is Github-specific. It relies upon clustering users based on Github-specific attributes such as programming languages, operating systems, and profile keywords. It is unclear how it would generalize to other social networks. The VAM models used in this dissertation are platform-agnostic. They simply rely on historical time series and historical user edge information to predict user activity, which can be found on any platform.

#### *2.2.2.2 Volume-to-User Decompositional Approaches*

The *Volume-to-User Decompositional* approaches in this work aim to predict user-repo interactions in two steps. First, the overall activity time series is predicted for a particular Github event. Then, these macroscopic activity counts, along with user-repo pair features are sent through a 2nd module to perform the more microscopic task of predicting user-to-repo activity over time. The 3 frameworks that use this approach are the *CVE-Action-to-Pair (CVE-ATP) Model* [33], the *Cyber-Action-to-Pair (Cyber-ATP) Model* [40] and the *SocialCube* model as discussed in [1] and [66].

The *CVE-ATP* and *Cyber-ATP* frameworks are 2 variations of the Action-to-Pair framework (ATP) [40, 33]. *ATP* is an LSTM neural network approach to Github user prediction. It decomposes the prediction problem into two tasks. Firstly, *ATP* predicts the daily-level volume of Github activities in the prediction time period of interest. This is the *Daily Level Prediction Task* [40]. It uses external features from Reddit and Twitter in order to make more accurate predictions. Then, *ATP*'s predicted daily counts, along with user-repo time series features are used to predict the number of events a user $u$ performs on a repository $r$ for each hour within each day of the prediction time frame of interest. This is the *Hourly User-Level Prediction Task* [40]. LSTM neural networks are used for both tasks.

The *ATP* framework was used to predict Github activity in two different datasets in two different works - [33] and [40]. With this in mind, we will refer to this framework as two different frameworks. The *ATP* implementation in [33] will be referred to as *CVE-ATP* because it was used to predict activity related to repositories in the Common Vulnerabilities and Exposures database (CVE). The data company Leidos used this database to create the CVE repo dataset.

The *ATP* implementation in [40] will be referred to as *Cyber-ATP* because it was used to predict activity related to cybersecurity repositories. This data was also gathered by Leidos. They mined the text of issue comments in Github and added a repo to the cybersecurity list if the repo's associated issue comments contained keywords related to cybersecurity such as "security" or "bot", etc.

Both the Cyber-ATP and CVE-ATP models were shown to perform strongly, however a shortcoming of these models is that they do not predict new user activity. In the Twitter networks used in this dissertation, new users comprise a considerable portion of the user population, so to that end, we use our VAM models predict their activity.

The Socialcube [1, 66] model predicts user-to-repo Github activity in two steps. First, an ARIMA model is used to predict the overall activity time series for 10 different Github events. Then, multiple ARIMA and deep-learning models are used to predict the activity time series of each user-repo pair.

The authors found that Socialcube was successfully able to outperform the Persistence Baseline in terms of user activity rate prediction. However, a limitation of Socialcube is its scalability. It relies upon training many models for multiple groups of users order to predict their future activity. This can be problematic in terms of time complexity for networks with millions of users

### 2.2.3 User Level Prediction Category #2: Direct

The direct user-level prediction methods predict future user activity in social media networks, but do so directly, unlike the decompositional approaches.

#### *2.2.3.1 Adjacency-Matrix-Based Approaches*

The works of [51, 68, 21, 55, 30, 56] perform temporal link prediction on sequences of adjacency matrices in various social media networks such as Twitter, Facebook, and StackExchange, among others. The benefit of using adjacency matrices is that a large amount of information is passed into the model - both the temporal information and node pair information. However, since these approaches rely on training and predicting with sequences of adjacency matrices, they are computationally complex both in terms of time and space. Representing the user network as an adjacency matrix can also be problematic if the user-to-user network is sparse, because that can lead to many erroneous predictions of no user activity, as noted by [30]. Lastly, none of these adjacency matrix based methods can predict the creation and activity of new users.

#### *2.2.3.2 Hand-Crafted-Feature Approaches*

The *Hand-Crafted-Feature* approaches are direct user prediction approaches that heavily utilize hand-crafted features in the task of user level-prediction. They rely upon the modeler to create and select features to be used in a model. Works that utilize this approach include [67, 3, 21, 44]. For individual nodes, they'll often utilize centrality metrics as features such as Betweenness, Degree, or Katz Centrality. For pairs of nodes, they'll often use similarity metrics as features such as Common Neighbors or Jaccard Similarity.

Hand-crafted features can also be used to generate features for cascades in a social network. The Genetic-LSTM algorithm of [34] is an LSTM-driven approach that uses spatio-temporal, user-level, and content-level features to predict user-level responses in cascades.

The benefit of these approaches is that they structure graph data in a way that can be utilized by a machine learning model. In the case of adjacency matrices, they can potentially reduce the space complexity of inputting entire adjacency matrices into a model. Furthermore, they can reduce the sparsity that often occurs in adjacency matrices. The downside is that calculating the various features can be time intensive. Figuring out the right combination of features can be a nontrivial task as well.

#### *2.2.3.3 Agent-Based Approaches*

Agent-based modeling approaches involve the modeling of entities called "agents" that behave within a closed system, or "environment". The modeler defines characteristics of these agents, and also defines rules for how agents interact with one another. Furthermore, the modeler defines "updating rules", for these agents, which describe how the agents' characteristics might change due to their interactions with one another [8].

In [24], the authors introduce the DeepAgent Framework. It is comprised of agent-based models that are generative rule-driven models, whose purpose is to simulate user actions in Github, Twitter, and Reddit. The authors created rules that determine when users can perform 1 of 4 actions: Post, Vote, Create, or Follow. The agent-based models are designed based on traditional Diffusion of Information models. The benefit of these models is that they provide high explainability with regards to what drives user activity on various social networks. However, the authors did not compare these models to any baseline or state-of-the-art methods, so their true efficacy is unknown.

In [7], the authors used 4 different agent-based sampling-driven models to predict future user-to-repo interactions in Github. The dataset they used included about 2 million users and 3 million repositories. Their agent-based framework was highly scalable due to their use of FARM, a distributed computing framework for agent-based modeling. While the different models were all highly scalable, the sampling-based approaches employed were somewhat

simplistic, and might fail to capture some intricate user-repo dynamics. Furthermore, the approach does not model new user activity.

Lastly, there's the work of [45], in which the authors used a combination of agent-based rules and machine learning that predicted user-to-repository links in Github, as well as user comment threads in Twitter and Reddit. In this work, the authors were also able to support the simulation of millions of users using FARM. Furthermore, the authors were also able to predict the various bursts of activity on each platform. However, this approach is highly computationally complex and does not model new user activity.

#### *2.2.3.4 Embedding-Based Approaches*

There have also been many approaches that utilize node embedding to predict user-level activity. The NPP model of [15] uses a neural network comprised of a Bidirectional GRU and an attention layer to predict whether or not a user's tweet will be popular in some future time period. While this model performed strongly, it is a binary classification task, and does not indicate how many retweets the user will get. Furthermore, it does not perform user-to-user link prediction. In contrast, the VAM models in this dissertation perform user-to-user link regression over multiple time steps.

There are also cascade-embedding approaches, such as DeepDiffuse [35] and TopoLSTM [64], that use cascade embedding to predict the most likely user to engage in a cascade at some point in the future. While these methods performed well, they only predict the activity of old users, and do not predict user-to-user interactions, which VAM does.

The CVE-seq2seq model of [39] uses LSTMs with embedding layers to perform the sequence-to-sequence prediction of events in the Github, Twitter, and Reddit social media networks. It was able to outperform the Persistence Baseline over the span of a year in predicting the user activities on each platform. It can also predict the creation of new users. The shortcoming is that it uses sequences of "event blocks" in each platform to predict future event blocks in each platform. So, this approach would potentially struggle to simulate the

activity of large amounts in events. In contrast, VAM predicts the overall volume of events (a.k.a. activities) with the Volume Prediction module, and then probilistically assigns new and old users to each event with the User Assignment Module, which is much more efficient in terms of both time and space.

Lastly, there are the popular embedding methods DeepWalk [50], Node2vec[27], and tNodeEmbed [58]. DeepWalk embeds nodes in a graph by performing numerous random walks, and then feeding these walks into the popular Word2Vec algorithm [4]. Node2vec is a modification of DeepWalk that allows one to embed users both in terms of "neighborhood similarity" and "structural similarity". In the neighborhood similarity paradigm, nodes with similar neighbors have similar embeddings. In the "structural similarity" paradigm, nodes with similar roles in the network have similar embeddings. An example of a structural role a node could have would be a hub node.

tNodeEmbed is an embedding technique for embedding nodes in a temporal graph. It uses the current and historical graph information to achieve this task. tNodeEmbed is first initialized using the static embeddings of each node over time using some other embedding technique. In this work (Chapter 3) we use both node2vec [27] and DeepWalk [50] static embeddings to initialize our tNodeEmbed embeddings.

After initializing the static embeddings of each node across all desired timesteps, tNodeEmbed then performs a "rotation" operation that aligns the embeddings over time. The goal of this operation is to make the embedding for a particular node at time T+1 similar to its embedding at time T. The issue with making static embeddings for a node at each timestep is that these different embeddings have different random weight initializations. As a result, the same node at time $T$ and $T + 1$ can have drastically different embeddings, making the overall graph embeddings unsuitable for any temporal downstream tasks (such as temporal node or link prediction). tNodeEmbed's rotation operation mitigates this issue [58].

Within the context of temporal social media prediction, DeepWalk and Node2vec have been used for user prediction in [56]. In these works, they were used to embed users in a Twitter graph, and then these embeddings were fed into a neural network to predict the number of activities these users would perform in the future. In [58], tNodeEmbed was used for temporal link prediction of users on the social media site Facebook.

Although embedding approaches are popular, a major shortcoming is that they take a long time to train. Also, they do not perform well with highly-sparse networks. In Chapter 3 we compare VAM with the tNodeEmbed embedding approach and show that VAM is much faster to train and is much more accurate.

## 2.3 Background on SMOTER for Social Media Time Series

As previously mentioned, Chapter 5 utilizes the SMOTER algorithm for data augmentation of social media time series data. In this section, we discuss related work pertaining to this topic.

### 2.3.1 Time Series Prediction of Social Media Data

Several of the previously mentioned *General Popularity Prediction* approaches and *User-Level Prediction* approaches are modelled as time series forecasting problems. As previously mentioned, there are the VAM models from [42, 43] and Chapters 3 and 4, which use XGBoost models to predict the future new users, old users, and activities over a 24-hour time period on Twitter.

The authors of [36] performed time series regression in the social media networks Facebook, Twitter, and Linkedin to predict the future volume of user activities in these platforms. The authors of [46] use LSTM models to predict Twitter and YouTube activity time series. They also explored the effect of exogenous time series such as Reddit and ACLED, and found that the use of exogenous features improved prediction accuracy. Lastly, the works of [30]

and [56] use neural networks on sequences of adjacency matrices to predict user activity time series in Twitter.

It is notable that while there are various previous works that explore time series prediction in social media, none of them explore the potential benefits of data augmentation, which is what we do in Chapter 5.

### 2.3.2 SMOTE

The SMOTER algorithm used to augment the time series data in Chapter 5 is based on the SMOTE algorithm. SMOTE stands for Synthetic Minority Oversampling Technique [14]. It is a technique typically used to perform "dataset balancing". In some classification datasets, the number of majority class samples far outnumbers the amount of minority class samples. This is problematic if there is a classifier trying to model the samples across the different classes. It is likely to erroneously classify the minority class samples as belonging to the majority class.

The methodology of SMOTE is as follows [14]:

1. The inputs to SMOTE are the number of minority class samples, $T$, the number of new synthetic SMOTE samples per minority sample $r$, and number of k nearest neighbors, $k$.

2. For each sample, obtain the $k$ nearest neighbors.

3. Randomly select $r$ neighbors from your set of $k$ neighbors for the particular sample. For example, if $k = 5$, and $r = 2$, randomly choose 2 of the 5 nearest neighbors for the particular minority sample of interest.

4. For each minority feature vector (sample) of interest $x$, take the difference between $x$ and the nearest neighbor vector of interest, $x^{rand\_nbr}$.

5. Multiply the difference by a random number between 0 and 1, called *epsilon* (epsilon), and then add it to $x$ to create a new synthetic example of the same class.

The formula for a newly generated input vector, $x'$ is as follows:

$$x' = x + \epsilon * (x^{rand\_nbr} - x)$$

By using this methodology, a new synthetic sample is created within the "bounds" of the feature space between any 2 given features between a minority sample and one of its neighbors. This causes the creation of new samples that are more diverse, but still similar enough to the original data that they do not add much detrimental noise to the overall dataset [14].

### 2.3.3 SMOTER

SMOTER (SMOTE for Regression) is a variant of SMOTE used for regression datasets [61]. Instead of creating synthetic samples for minority-class samples, synthetic samples are created for samples with "rare" or "relevant" target values relative to the dataset.

The authors of [61] define relevance as a continuous function $\psi(Y) : y \rightarrow [0, 1]$ that maps the target variable domain $y$ into a [0,1] scale of relevance, where 0 represents the minimum and 1 represents maximum relevance. The authors note that the user can define what samples are considered "relevant", however, a straightforward calculation they offer is relevance as the inverse of the target variable probability density function. For example, if a target value's occurrence in the dataset is 5%, then its relevance can be interpreted as 95%.

SMOTER works in the following way. The user assigns a relevance score to each sample in their dataset. The user then defines a threshold between 0 and 1 for relevance. This threshold is used to determine which samples will be augmented with synthetic samples. For example, if the user sets a threshold of 0.8, this means that all samples above 0.8 will be considered "relevant" and all samples below 0.8 will be considered "irrelevant".

SMOTE is then applied to the "relevant" samples. The number of new synthetic samples generated is determined by the user [61]. The authors of [61] also have a formula for computing the new target value. The formula involves using a weighted average of the target values of the two seed examples in order to generate the target value of the newly generated synthetic sample ($x'$). The two seed examples are the current input $x$ vector of interest, and a randomly chosen (from K candidates) nearest neighbor of that $x$ vector, or $x^{rand\_nbr}$. The user can define the distance formula he or she desires.

So, in other words, let $d_1$ be the distance from $x$ to $x'$. Let $d_2$ be the distance from $x$ to $x^{rand\_nbr}$. Lastly, let $y^{rand\_nbr}$ be the corresponding target value for $x^{rand\_nbr}$. The formula for the new target variable, $y'$ would be the following according to [61]:

$$y' = \frac{d_2 * y + d_1 * y^{rand\_nbr}}{d_1 + d_2}$$

In our SMOTER_NB and SMOTER_BIN algorithms that we introduce in Chapter 5, we simply used a random distance between the target values of the two examples as the new target value, $y'$. However, to maintain consistency with the newly generated input vector, $x'$, we use the same random value between 0 and 1 (*epsilon*) when generating both $x'$ and $y'$. So, the formulas for both $x'$ and $y'$ would be:

$$x' = x + \epsilon * (x^{rand\_nbr} - x)$$

$$y' = y + \epsilon * (y^{rand\_nbr} - y)$$

We used this approach for calculating $y'$ instead of the original SMOTER approach because we believe it is more intuitive and explainable. Furthermore, we found it to yield strong results, as shown in Chapter 5. However, future work would involve trying the SMOTER method of calculating $y'$ and finding out if it works better.

### 2.3.4 SMOTER for Time Series Regression

SMOTER was previously used for time series regression in [62]. However, we note several key differences between their work and ours.

Firstly, the authors do not use SMOTER on social media time series data. Social media has such a large role in today's world, so for that reason we aim to explore SMOTER's utility in that domain in this work. Secondly, the authors of [62] only augment the "rare" or "relevant" examples. In this dissertation, we do both. We train a model that only augments rare examples (SMOTER_BIN) and we also train a model that augments all examples (SMOTER_NB). Thirdly, the authors of [62] only use SMOTER and regression models on datasets in which each sample is 1 input time series and one output value. In this dissertation, we use SMOTER and our machine learning models on datasets in which each sample is comprised of multiple input time series and the outputs are comprised of 3 time series of length 24 (72 values).

Fourthly, the authors of [62] only evaluate predictive success on the rare samples in the test set, whereas we evaluate all test samples. We wanted to make sure that a model trained with SMOTER preprocessing would be competitive against a model trained without it. Fifthly and finally, the authors of [62] only use the Utility-Based F1 score to measure success. We, on the other hand, evaluate all of test set samples with the RMSE, MAE, S-APE, VE, SkE, and NC-RMSE metrics. These metrics measure a wide-range of time-series properties such as exact-timing, magnitude, volatility, and assymetry. Our motivation for doing this is that it is important to make sure that a model is able to capture the wide range of properties of the ground truth time series.

## 2.4 Background on Time Series Baselines

### 2.4.1 Persistence Baseline

The "Persistence Baseline" model is a simple, common baseline approach we use for comparison throughout this work. It is defined as follows. Let $T$ be the current timestep of interest, and let $S$ is the length of the desired predicted time series. The Persistence Baseline predicts the time series for period $T + 1$ to $T + S$ by moving forward, or "shifting forward" the time series that spans period $T - S$ to $T$. The underlying assumption of this model is that the immediate future of the time series will simply approximately replicate its immediate past.

The Persistence Baseline is a common baseline for social media time series prediction as shown in [56, 30, 1].

### 2.4.2 Auto-Regressive Integrated Moving Average Models

The Auto-Regressive Integrated Moving Average Model (ARIMA) is a type of classical time series forecasting model. They can be used to model both stationary or non-stationary time series data. Stationary time series are time series whose mean and variance remain the same over time. Non-stationary time series are time series whose mean and/or variance change over time [9].

The ARMA, AR, and MA models are variants of ARIMA depending on what the *p, d,* and $q$ parameters are set to. The ARIMA model has $p > 0$, $d > 0$, and $q > 0$. The AR model has $p > 0$, $d = 0$, and $q = 0$. The ARMA model has $p > 0$, $d = 0$, and $q > 0$. Lastly, the Moving Average (MA) model has $p = 0$, $d = 0$, and $q > 0$ [9].

# Chapter 3: The VAM Simulator Introduction and Vz19 Dataset[1]

## 3.1 Introduction

Social media's vast societal influence is apparent. Recent research has shown its effect in many aspects of society, such as election campaigns [10], the spread of COVID-19 misinformation [59], and the promotion of pump and dump cryptocurrency schemes [37].

Clearly, it would be ideal to predict the future phenomena related to any topic on any social media platform. To that end, we created an end-to-end simulator, called the *Volume-Audience-Match Algorithm*, or VAM. VAM's goal is to predict what will happen for a given topic on a social media platform. Experimental results are shown on 18 topics appearing on Twitter.

VAM is a model that consists of two components, or *modules* that work in the following way. Firstly, for each topic in a given social media platform, at some time step of interest, $T$, the *Volume Prediction Module* of VAM takes as input a set of past time series features both related to that topic, as well as external exogenous features that may influence that topic's behavior in the future. It then uses these features in order to perform time series forecasting. For any given *topic-timestep* pair, VAM predicts three time series of length $S$ which are: (1) the topic's future event volume time series, (2) the topic's newly active user time series, and (3) the topic's active old user time series.

Secondly, the *User-Assignment Module* of VAM uses these 3 time series predictions, as well as previous user interaction history, to tackle the more fine-grained task of predicting, for a given topic, within the timespan of $T+1$ up to $T+S$: (1) which user performs which

[1]The material in this chapter was published in the IEEE Transactions on Social Computing Journal 2022 [43]. 

action and (2) with whom each user interacts. We frame this problem as a link prediction problem. An edge is comprised of a child user $u$ and a parent user, $v$. An edge exists between $u$ and $v$ if $u$ reacts to a post written by $v$. In Twitter, this reaction takes the form of a retweet, or tweet in the case of an initial tweet (i.e. self-loop).

Note that we use the term "module" to differentiate from the term "model" for clarity throughout this work. VAM is the name of the overall model, while the *Volume-Prediction* Module is the component of VAM that predicts the volume predictions, and the *User-Assignment Module* is the component of VAM that performs the user-to-user predictions.

In this chapter, we evaluate VAM's predictive power on the Twitter dataset related to the Veneuzuelan political crisis [19][5]. A time period spanning from December 28, 2018 up until March 7, 2019 was used.

In this chapter, we show the following. Firstly, we show a simulation pipeline that performs both time series regression and temporal link prediction tasks in an end-to-end manner. We show that VAM can predict the creation of new users and their activites. We show that VAM strongly outperforms multiple statistical and state-of-the-art comparisons across a myriad of metrics for the time series prediction task. We show how VAM performs when using XGBoost as its "backend", versus when it uses recurrent neural networks (RNNs) as its backend. We show that the XGBoost versions of VAM are more accurate and faster to train the RNN versions. This is notable because RNNs have been used extensively in previous social media prediction literature, while XGBoost has not. We provide an analysis of the use of social media platform features to determine what helps VAM achieve the best time series prediction performance in Twitter. We show that using features from activity on Reddit improves predictions of Twitter activity. Lastly, we show that VAM greatly outperforms the baseline and state-of-the-art models in multiple user-assignment tasks, which were the old user prediction task, the indegree prediction task, and Page Rank prediction task.

## 3.2 Importance of Predicting New Users

In this work we define a "new" user at timestep $T$ as someone who has not previously been involved with a topic in the period spanning $t = 1$ up to $t = T - 1$. Our data analysis showed that for certain topic-platform pairs, there are a considerable number of new users that appear each day. For 5 out of 18 topics, on average at least 40% of the active users within a given day are new. For 9 out of 18 topics, on average at least 25% of the users in a given day are new. For this reason, it is important to predict their appearance and activities in addition to that of the old users.

Table 3.1 shows the percentages and Figure 3.1 shows the bar plot for the average daily new and old user ratios per topic. These ratios were obtained by calculating the average number of new users per day, and then calculating the average old users per day, per topic. These values were then normalized between 0 and 100%. In Figure 3.1, the orange bars represent the old user average frequencies, and the blue bars represent the new user average frequencies. In the plot it can be seen that on average per day for each topic, most of the users are old, however for some topics there are a considerable number of new users. For example, the *other/censorship_outage* topic has about 60% new users on average per day, and the *other/anti_socialism* topic has roughly 50% new users on average per day.

## 3.3 Problem Statements

VAM addresses 2 problems, namely the (1) volume prediction of users and activities, as well as (2) the assignment of the predicted activities to the appropriate users within the context of a user-to-user link prediction. In this section we will discuss these two problems in more detail.

Table 3.1: Vz19 Twitter avg. new and old user frequencies per day.

| **Twitter Avg. New and Old User Frequencies Per Day** | | |
|---|---|---|
| **Topic** | **New User Avg. Freq. (%)** | **Old User Avg. Freq. (%)** |
| other/censorship_outage | 62.22 | 37.78 |
| other/anti_socialism | 52.15 | 47.85 |
| military/desertions | 43.94 | 56.06 |
| maduro/cuba_support | 41.61 | 58.39 |
| international/aid_rejected | 41.53 | 58.47 |
| maduro/narco | 39.21 | 60.79 |
| other/chavez/anti | 38.24 | 61.76 |
| other/chavez | 27.55 | 72.45 |
| maduro/dictator | 26.34 | 73.66 |
| arrests/opposition | 23.7 | 76.3 |
| maduro/legitimate | 22.84 | 77.16 |
| arrests | 21.88 | 78.12 |
| international/respect_sovereignty | 21.74 | 78.26 |
| guaido/legitimate | 20.64 | 79.36 |
| international/aid | 20.49 | 79.51 |
| protests | 20.43 | 79.57 |
| violence | 18.2 | 81.8 |
| military | 15.27 | 84.73 |

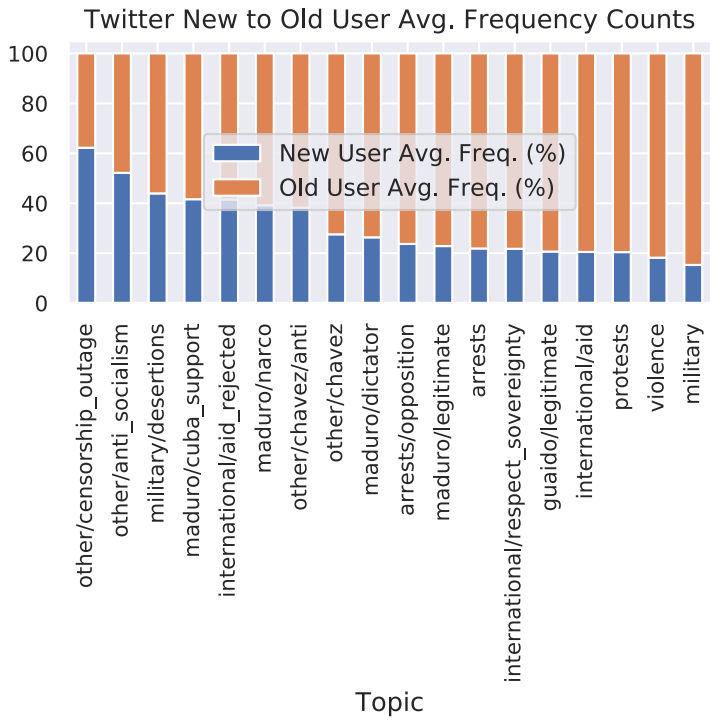



Figure 3.1: The new/old user proportion plots for Vz19 Twitter.

### 3.3.1 Volume Prediction Problem

In this subsection we describe the *Volume Prediction Problem*. Let us say, that for some given platform, one is given static and temporal features for some topic, $q \in Q$, at some timestep $T$. Intuitively, $T$ can be thought of as the current time step of interest. One must then predict the following 3 future time series relating to this *topic-timestep* pair, $(q, T)$: (1) the activity volume time series, (2) the active old user volume time series, and (3) the new active user volume time series. Each time series must span from time $T+1$ up to $T+S$, with $S$ being an integer that represents the length of the predicted time series. Furthermore, let $\hat{\boldsymbol{Y}} \in \mathbb{R}^{3\times S}$ be a time series matrix that represents the aforementioned predicted time series. In other words, $\hat{\boldsymbol{Y}}$ represents a prediction matrix such that each row represents one of the 3 output time series, and each column represents a time step in the repsective time series.

$\hat{\boldsymbol{Y}}$ approximates the ground truth matrix, $\boldsymbol{Y}$. The time frame that $\boldsymbol{Y}$ encompasses ($T+1$ to $T+S$) is called *forecast period of interest*, or $F_T$. $F_T$ can be thought of as a tuple of the

form $(T+1, T+S)$. $T+1$ is the first time step in the *forecast period of interest* and $T+S$ is the last time step.

In order to address the volume prediction problem, we created both XGBoost and recurrent neural network (RNN) regression models. The inputs are time series features and static features related to a given topic. The granularity of time step information given the models is hourly, and they predicted 24-hour time series (1 day). We chose XGBoost models because they are known for being relatively quick to train, while retaining high predictive accuracy [16]. Training time is important to consider when performing daily predictions. From a practical stand point, if a model that must perform daily predictions takes too long to train, it is not usable, even if the predictions are accurate. We also used RNNs because they have been widely used in the previous literature for social media prediction [40, 33, 39, 30, 56, 58].

### 3.3.2 User-Assignment Link Prediction Problem

In this subsection the problem statement for the User-Assignment problem is introduced. Let $\{G\}_{t=1}^{t=T}$ be a sequence of static graphs such that $G = \{G_1, G_2, ...G_T\}$. $G$ represents the user-interaction history of some topic, $q$, on some social media platform.

Each graph at time step $t$, $G_t$ can be viewed as a tuple of sets of the form $(V_t, E_t, w)$. $V_t$ is the set of all users (nodes) $u \in V_t$ present in graph $G_t$. $E_t$ is the set of all edges that exist in graph $G_t$. The edges in $E_t$ are of form $(u, v, w(u, v, t))$. An edge exists in $E_t$ if user $u$ responded to a post made by user $v$ at time step $t$. The term, $w$ represents a weight function such that $w(u, v, t)$ represents how many times user $u$ responded to $v$ at time step $t$. Using this information, we can now define VAM's *User Assignment Problem* as follows.

Let us say, for some *topic-timestep* pair, $(q, T)$, one is given a matrix, $\hat{\boldsymbol{Y}} \in \mathbb{R}^{3\times S}$. This matrix contains, for $(q, T)$, the future volume prediction time series for the (1) number of activities, (2) number of old users, and (3) number of new users. Furthermore, let us say one is given a set, $\{G\}_{t=1}^{t=T}$, that represents the user-interaction history of topic $q$, for each of the $T$ time steps. Given these 2 input items, predict a sequence, $\{\hat{G}^{future}\}_{t=1}^{t=S}$. Intuitively,

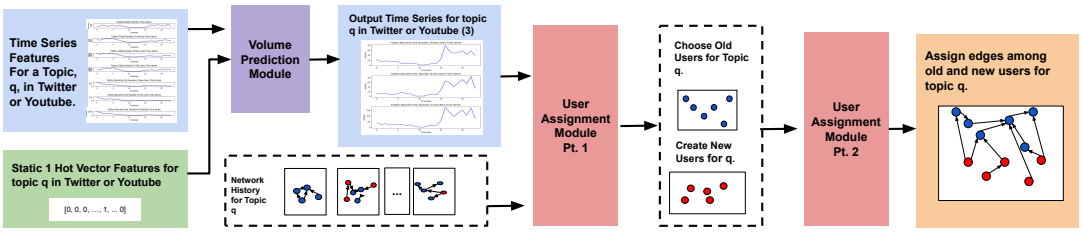



Figure 3.2: Framework for the Volume-Audience Match algorithm (VAM).

one can think of $\hat{G}^{future}$ as a set containing the future user interactions over the next $S$ timesteps with regards to *topic-timestep* pair, $(q, T)$. So, a sequence $\hat{G}^{future}$ would have the following form: $\hat{G}^{future} = \{\hat{G}_1^{future}, \hat{G}_2^{future}, ..., \hat{G}_{24}^{future}\}$. Graph $\hat{G}_1^{future}$ represents the future user interactions for topic $q$ at time step $T + 1$. Graph $\hat{G}_2^{future}$ represents the future user interactions for topic $q$ at time step $T + 2$, and so on. Note that $\hat{G}^{future}$ is an approximation of the ground truth graph set, $G^{future}$.

Intuitively, one can view the User-Assignment problem as a temporal link prediction problem, but with the added "assistance" of the future volume counts from some predictive model. A pictorial overview of VAM is shown in Figure 3.2.

## 3.4 Data Collection

### 3.4.1 Twitter Data Collection

The raw data on the 2019 Venezuelan political crisis used in these experiments were originally collected by data collectors at the Leidos company. For the Twitter data, Subject Matter Experts (SMEs) compiled a list of keywords and Twitter handles that would allow for the collection of the most relevant Venezuela tweets. These subject matter experts were individuals hired by Leidos who were fluent in Spanish and very familiar with the political situation in Venezuela. The keywords were evaluated by the SMEs for both their precision and recall with regard to tweets about the Venezuelan political crisis. Frequently co-occurring groups of keywords were then used to create 18 "topics". For example, the topic *international/aid_rejected* is a topic comprised of two separate keywords "international" and

Table 3.2: Vz19 Twitter network statistics.

| **Twitter Graph Information** | | | |
|---|---|---|---|
| **Topic** | **# Nodes** | **#Edges** | **Total Edge Weight (a.k.a. Total # of Activities)** |
| military | 457,200 | 2,458,703 | 4,580,984 |
| international/aid | 484,405 | 2,018,902 | 3,530,265 |
| protests | 451,542 | 2,058,608 | 3,083,175 |
| violence | 400,141 | 1,957,442 | 3,031,137 |
| guaido/legitimate | 355,381 | 1,437,221 | 2,122,211 |
| international/respect_sovereignty | 205,180 | 815,250 | 1,635,717 |
| maduro/dictator | 355,552 | 1,137,656 | 1,528,799 |
| other/chavez | 222,025 | 697,542 | 1,154,887 |
| arrests | 175,685 | 687,628 | 935,191 |
| arrests/opposition | 147,454 | 551,617 | 718,539 |
| international/aid_rejected | 211,168 | 518,668 | 662,886 |
| maduro/legitimate | 94,424 | 351,705 | 655,588 |
| other/chavez/anti | 142,556 | 300,346 | 398,892 |
| military/desertions | 125,257 | 285,934 | 365,718 |
| maduro/narco | 92,208 | 190,973 | 244,958 |
| other/anti_socialism | 119,519 | 184,152 | 238,342 |
| maduro/cuba_support | 62,904 | 112,281 | 153,640 |
| other/censorship_outage | 62,603 | 110,097 | 122,581 |

"aid_rejected". This topic refers to the disputed President of Venezuela, Maduro, rejecting humanitarian aid from other countries to the people of Venezuela [48].

The SMEs then labelled a small subset of tweets with the best-fitting topic. This subset was then used to train a BERT model [22] that would then be used to label the 25,163,510 tweets used in this work.

Table 3.2 contains the node, edge, and activity (tweet) counts of the networks used in this work. Since there were 18 topics in our dataset, we had 18 Twitter networks. Each node in a Twitter graph represents a user and each edge represents an interaction between users, such as a retweet, or tweet (which can occur in the case of a self-loop). Note, that we did not include quotes or replies in our Twitter data, because they comprised such a small portion of overall Twitter activity (3.6%).

The network with the largest number of edges was *military* with 4,580,984 edges. The network with the smallest number of edges was *other/censorship_outage* with 110,097 edges.

### 3.4.2 Reddit Collection

Reddit is a social media platform in which users read and comment on various message boards, known as *subreddits*. Reddit posts and comments spanning from Decemeber 28, 2018 to March 7, 2019 related to the Venezuelan political crisis were collected by our team at USF.

## 3.5 Volume Prediction Methodology

### 3.5.1 Use of Lookback Factor and Exogenous Data

We were interested in knowing if the next 24 hours of social media activity could be predicted from some initial timestep, $T$, so to that end, we set $S = 24$ in our experiments. We believe 24 hours is long enough to be useful in a practical application, but still short enough that it is a reasonable period for a model to predict within.

Secondly, we needed to define an appropriate lookback period, or *volume lookback factor* for prediction, which we defined as $L^{vol}$. This means that we used historic data from $L^{vol}$ timesteps to make a prediction. In our experiments we tried 96, 72, and 48 hours as values for $L^{vol}$.

Furthermore, we wanted to know whether it was sufficient to use Twitter data alone in order to perform successful future activity predictions on Twitter, or if exogenous features from Reddit were helpful. To that end, we trained and tested 2 different types of Twitter models, which were Twitter-only models (T) and Twitter and Reddit models (TR).

### 3.5.2 Feature Configuration

To better understand how the features are configured we shall describe an example in Table 3.3. Take, for example, the fourth row for the model, *VAM-TR-96*. This is a *VAM*

Table 3.3: Vz19 feature configurations for each VAM model.

| **VAM Model Feature Configurations** | | | | |
|---|---|---|---|---|
| **Model** | **Time Series Used** | **Volume Lookback Factor** | **Num Static Fts.** | **Total Features** |
| VAM-T-96 | (1, 2, 3, 4, 5, 6) | 96 | 18 | 6 * 96 + 18 = 594 |
| VAM-T-72 | (1, 2, 3, 4, 5, 6) | 72 | 18 | 6 * 72 + 18 = 450 |
| VAM-T-48 | (1, 2, 3, 4, 5, 6) | 48 | 18 | 6 * 48 + 18 = 306 |
| VAM-TR-96 | (1, 2, 3, 4, 5, 6, 7) | 96 | 18 | 7 * 96 + 18 = 690 |
| VAM-TR-72 | (1, 2, 3, 4, 5, 6, 7) | 72 | 18 | 7 * 72 + 18 = 522 |
| VAM-TR-48 | (1, 2, 3, 4, 5, 6, 7) | 48 | 18 | 7 * 48 + 18 = 354 |

model trained on Twitter and Reddit time series features. The *Time Series Used* column illustrates which time series were fed in from Table 4.3. It says that features 1-7 were used. As one can see in Table 4.3 these are all time series related to Twitter and Reddit, which explains the "TR" in the model tag. Note that each platform in 3.3 is represented with a letter. "T" stands for "Twitter", "R" stands for "Reddit".

The *volume lookback factor* column for *VAM-TR* indicates it's "96 hours". So, for each time series category listed in the *time series used* category, a time series of 96 elements is placed into the feature set for the *VAM-TR*. Since there are 7 time series, a *volume lookback factor* of 96, and 18 topic features; the calculation for number of features is 7 * 96 + 18, which equals 690. Therefore, the dataset for the *VAM-TR* model was comprised of 690 features (as shown by the *Total Features* column).

### 3.5.3 Sample Tuples

As previously mentioned, each sample in each dataset represents a *topic-timestep* pair, $(q, T)$. The variable, $q$ represents the topic of interest, and $T$ represents the current time step of interest. Each *topic-timestep* sample is comprised of input features and output values. The inputs and outputs are described as follows.

Firstly, there is the *static input feature set*. This is a 1-hot vector that represents the topic of interest, $q$. Secondly, there are the *temporal input features*. These are the time

series input features for our given sample. They differ depending on the model. The types of temporal features used are listed in Table 4.3. Lastly, there are *the output targets*. The output for a given *topic-timestep* pair, ($q$, $T$), is the matrix $\boldsymbol{Y} \in \mathbb{R}^{3 \times S}$. This matrix is comprised of the 3 output time series for the volume prediction task: the event volume time series, the new user volume time series, and the old user volume time series.

Table 3.4: All possible time series feature categories.

| **Time Series Index** | **Time Series Description** |
|---|---|
| **1** | New user volume time series for a given topic in Twitter. |
| **2** | Old user volume time series for a given topic in Twitter. |
| **3** | Activity volume time series for a given topic in Twitter. |
| **4** | Activity volume time series across all topics in Twitter. |
| **5** | New user volume time series across all topics in Twitter. |
| **6** | Old user volume time series across all topics in Twitter. |
| **7** | Activity volume time series in Reddit. |

We shall illustrate this point with an example. Let us define the Twitter prediction matrix to be $\boldsymbol{Y}$, for topic $q$ = *arrests*, and current timestep of interest $T$= 200. Furthermore, we define a *volume lookback factor* of $L^{vol} = 96$ and we define the output time series size, $S$, to be equal to 24.

Given these definitions, our model would use temporal information from time steps 105 up to 200 (96 time steps including 200) to predict the output values related to the Twitter phenomena for the *arrests* topic from time steps 201 to 224.

Our test period contained 21 *forecast periods of interest* ($F_T$). These periods were the 21 days spanning February 15th, 2019 to March 7th, 2019. Recall that there were $n^{topics} = 18$ topics. So, in total, there were $18 * 21 = 378$ test samples.

For the training period, the prediction days of interest spanned from December 28th 2018 to February 7th, 2019 (42 days). For the validation period, the prediction days of interest spanned from February 8th to February 14th (7 days).

For the training and validation sets, we wanted to generate as much data as possible, so, we calculated each daily sample both in terms of day and hour. That is, a sliding window was used and advanced 1 hour to create a new overlapping example. By using this method, we generated 17,730 samples for each training set, and 2,610 samples for each validation set.

### 3.5.4 XGBoost Setup

In this subsection we discuss our setup of the VAM XGBoost models. Let $\mathcal{D}$ be a dataset such that: $\mathcal{D} = (\mathbf{x}_i, \boldsymbol{Y_i})$. Furthermore let the following be true:

$$|\mathcal{D}| = n^{samples} = n^{topics} * \tau; \mathbf{x}_i \in \mathbb{R}^m, \boldsymbol{Y_i} \in \mathbb{R}^{3\times S}.$$

The terms $n^{samples}$, $n^{topics}$, and $\tau$ represent the number of samples, topics, and prediction timesteps of interest, respectively. $\mathbf{x}_i \in \mathbb{R}^m$ represents an input feature vector of $m$ features, and $\boldsymbol{Y_i} \in \mathbb{R}^{3\times S}$ represents the output matrix.

We then define a matrix of functions, $\Phi(\mathbf{x}_i) \in \mathbb{R}^{3\times S}$ such that:

$$\begin{aligned}\hat{\boldsymbol{Y}}_{\boldsymbol{i}} = \Phi(\mathbf{x}_i) = \begin{bmatrix} \phi_{1,1}(\mathbf{x}_i) & \phi_{1,2}(\mathbf{x}_i) & \dots & \phi_{1,S}(\mathbf{x}_i) \\ \phi_{2,1}(\mathbf{x}_i) & \phi_{2,2}(\mathbf{x}_i) & \dots & \phi_{2,S}(\mathbf{x}_i) \\ \phi_{3,1}(\mathbf{x}_i) & \phi_{3,2}(\mathbf{x}_i) & \dots & \phi_{3,S}(\mathbf{x}_i) \end{bmatrix} = \\ = \begin{bmatrix} \hat{y}_i^{1,1} & \hat{y}_i^{1,2} & \dots & \hat{y}_i^{1,S} \\ \hat{y}_i^{2,1} & \hat{y}_i^{2,2} & \dots & \hat{y}_i^{2,S} \\ \hat{y}_i^{3,1} & \hat{y}_i^{3,2} & \dots & \hat{y}_i^{3,S} \end{bmatrix}\end{aligned} \tag{3.1}$$

Each function $\phi_{a,b}(\mathbf{x}_i)$ in the matrix represents a separate XGBoost model, and each of these models maps to an output-type-and-timestep pair value, $\hat{y}_i^{a,b}$. An integer variable, $a$ can be used to indicate any particular row in the matrix, such that $1 \leq a \leq 3$, and an integer variable, $b$ can be used to indicate any particular column of the matrix such that $1 \leq b \leq S$.

Recall that the rows represent one of the 3 output types (actions, new users, or old users), while the columns represent one of the $S$ future time steps.

The function, $\Phi(.)$, represents the *Volume-Prediction Module*, which contains $3 * S$ XGBoost models, $\phi_{a,b}(.)$, and each XGBoost model is an ensemble of CART trees. Intuitively, one can think of each of the XGBoost models as "specializing" on a particular *(output-type, timestep)* pair.

There are $3 * S$ models used because XGBoost is comprised of regression trees. A regression tree can only predict 1 output. So, to predict a time series, one would need a regression tree for each timestep in the time series. The alternative to the multiple-model approach would be to predict an output, feed that output back into the XGBoost model as an input, predicting the 2nd output, and so on. The problem with this approach is that one will run into the issue of compounding errors over time. As a result, these errors could cause these model to predict time series that do not come close to approximating the ground truth at all.

### 3.5.5 XGBoost Parameter Selection

We used the *XGBoost* [16] and *sk-learn* [49] libraries to create and train our models. The parameters used for our XGBoost models are as follows. The subsample frequency, gamma, and L1 regularization were set to 1, 0, and 0 respectively. For the other parameters, we performed a grid search over a pool of candidate values. We used our validation set to evaluate for the best parameters to use. For the *column sample frequency*, the candidate values were 0.6, 0.8, and 1. For the *number of trees* parameter, the candidate values were 100 and 200. For the *learning rate*, the values were 0.1 and 0.2. For *L2 Regularization*, the values were 0.2 and 1. Lastly, for *maximum tree depth*, the values were 5 and 7.

For the loss function, Mean Squared Error was used. For normalization, log normalization was used.

### 3.5.6 Log Normalization

For all Twitter-related features, we rescaled the data by first taking the natural log of all samples twice. Before taking the logs, we added 1 to all values in order to avoid taking the natural log of 0. For the Reddit features, we only took the natural log once, because the magnitude of those features was not as large as the Twitter ones.

### 3.5.7 Recurrent Neural Network Overview

We experimented with 4 different recurrent neural networks: GRU [17], LSTM [32], Bidirectional LSTM [54], and Bidirectional GRU [54] networks. Unlike XGBoost, RNNs have the ability predict multiple outputs within one model, so we did not have to make multiple RNNs per *(output-type, timestep)* pair in the same manner as the XGBoost VAM models.

### 3.5.8 RNN Hyperparameters

The hyperparameters of the RNN are as follows. All 4 RNNs had a batch size of 32, MSE loss function, 0.0001 learning rate, and RMSProp optimizer. Dropout layers with a rate of 20% were used as well. All RNN models had their epochs set to 100, with a *patience* parameter set to 10 epochs. Every epoch, the model's MSE loss was evaluated on the validation set. If this validation loss did not decrease for 10 epochs in a row, the model stopped training.

### 3.5.9 RNN Architectures

Figures 3.5, 3.6 3.3, and 3.4 show the architectures for the VAM-GRU-TR-96, VAM-LSTM-TR-96, VAM-Bi-GRU-TR-96, and VAM-Bi-LSTM-TR-96 models, respectively. These models were made with Keras [18].

As seen in the diagrams, the input to each model was a tensor with the dimensions (?, 96, 7). In Keras, the question mark (?) simply refers to the batch size of the model, which

was 32. The 96 represents the length of the input time series (96 hours). The 7 refers to the 7 features used in each TR model:

1. number of Twitter topic-level new users
2. number of Twitter topic-level old users
3. number of Twitter topic-level activities
4. number of Twitter global (all-topics) new users
5. number of Twitter global old users
6. number of Twitter global activities
7. number of Reddit activities

The final output of the model is a tensor with dimensions (?, 72). Once again, the question mark refers to the batch size of 32. The 72 refers to the total number of outputs: 24 hours * 3 output-types (new users, old users, activities).

Also, each model has a *static_input* layer with 18 inputs. These 18 inputs represent the 1-hot vector for the 18 topics of the Venezuela dataset. These static feature vectors are concatenated with the recurrent layer output in each model.

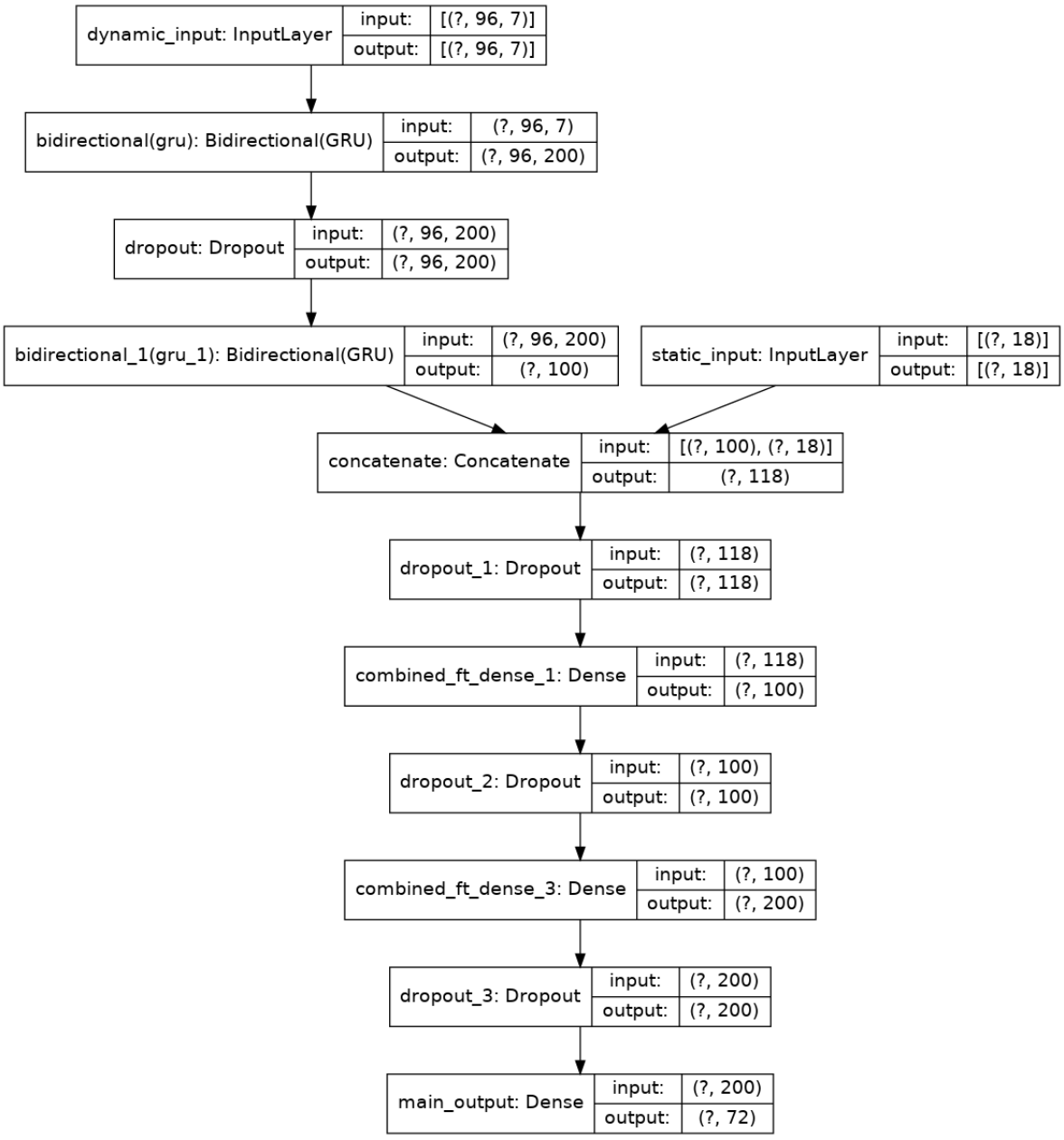



Figure 3.3: The VAM-Bi-GRU-TR-96 architecture.

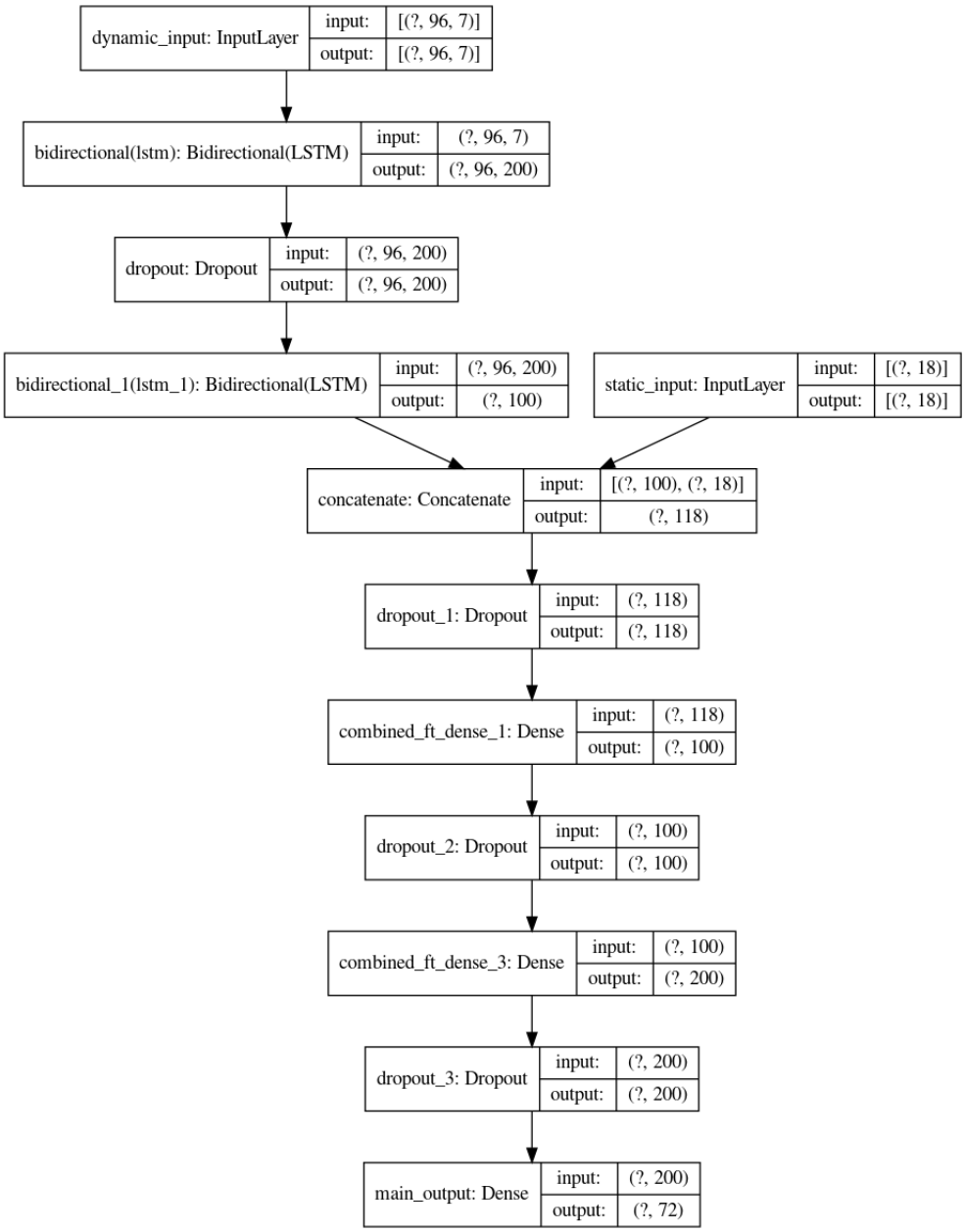



Figure 3.4: The VAM-Bi-LSTM-TR-96 architecture.

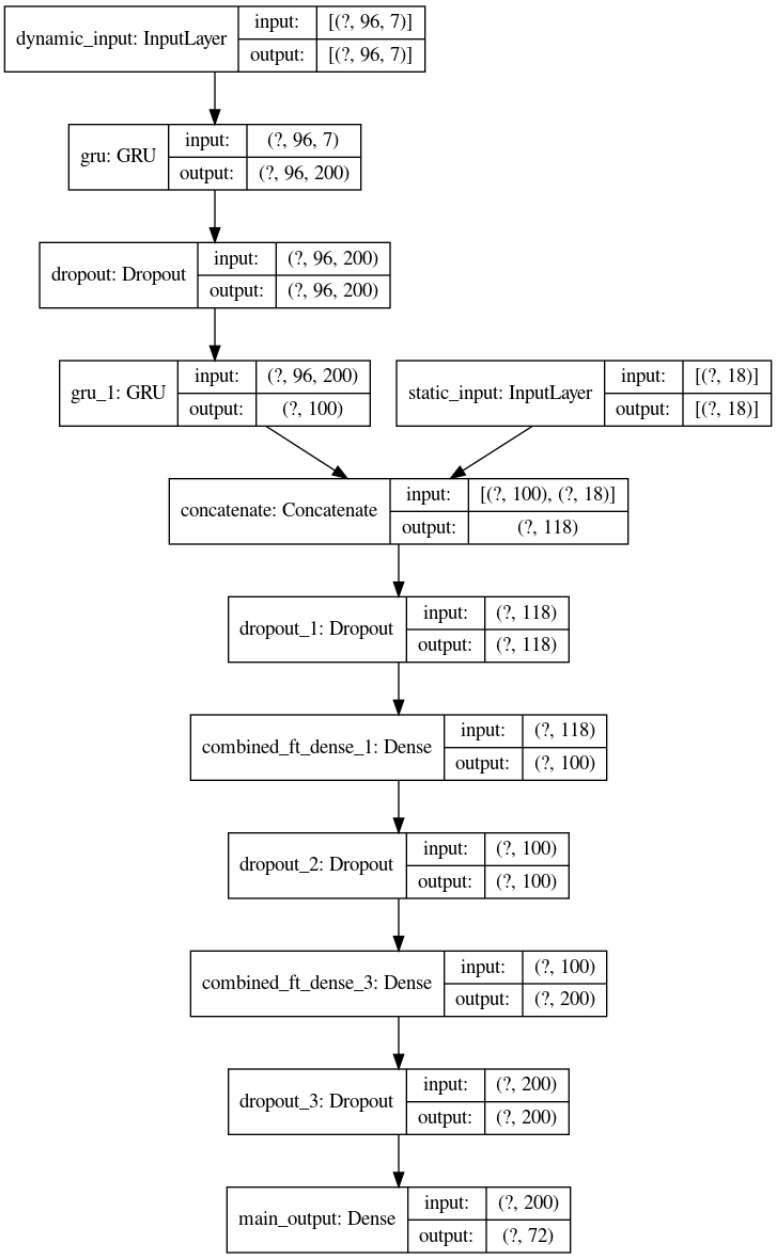



Figure 3.5: The VAM-GRU-TR-96 architecture.

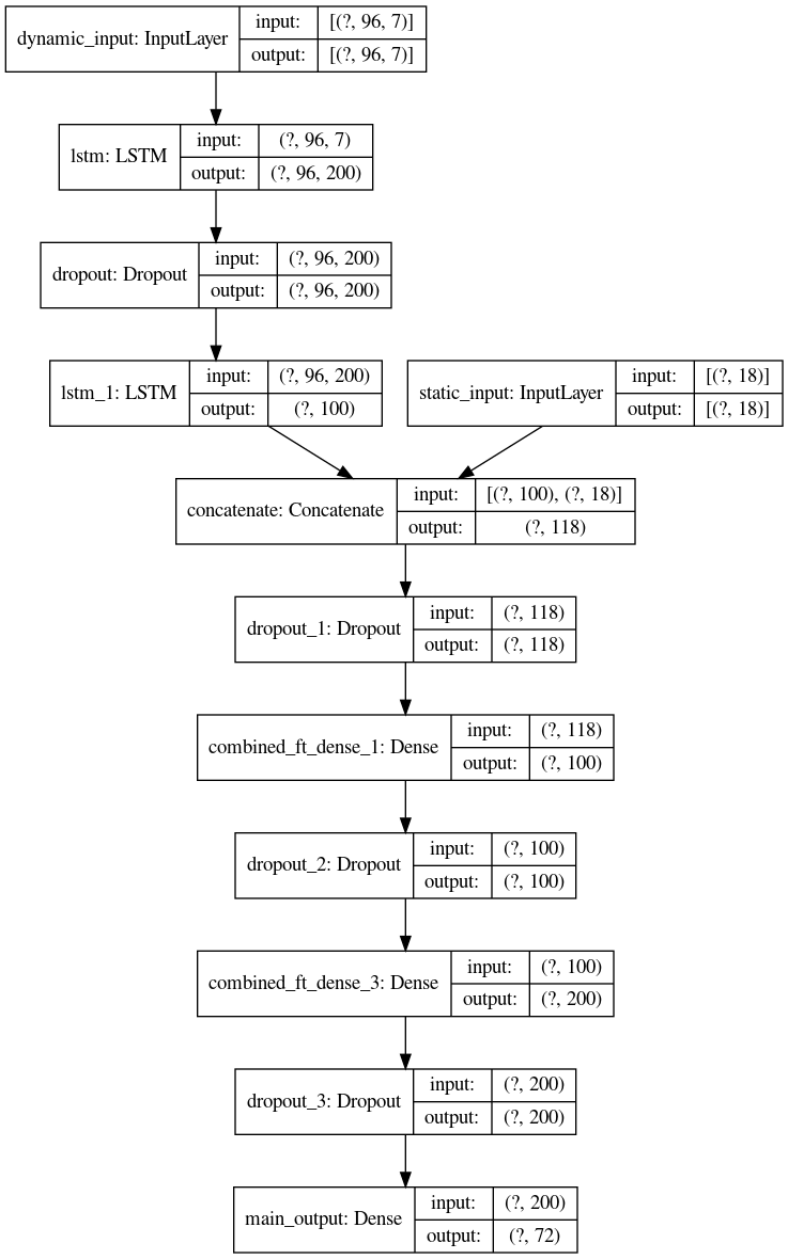



Figure 3.6: The VAM-LSTM-TR-96 architecture.

### 3.5.10 Statistical Baselines

We compared VAM to 5 statistical baseline models in this work, which were the Persistence Baseline, ARIMA, ARMA, AR, and MA models [9]. In this subsection, we describe them in detail.

The Persistence Baseline model predicts the events during time frame $T+1$ to $T+S$ by simply outputting the events that occurred at time $T-S$ to $T$. The assumption of this model is that the future will approximately resemble the recent past. This assumption may sound naive, however we found this baseline to perform very well against the other baselines and state-of-the-art models.

The Auto-Regressive Integrated Moving Average model (ARIMA) and its variants (ARMA, AR, and MA) are widely used statistical models and, hence, used for comparison as well. The ARMA, AR, and MA models are variants of ARIMA depending on what the $p$, $d$, and $q$ parameters are set to. The ARIMA model has $p > 0$, $d > 0$, and $q > 0$. The AR model has $p > 0$, $d = 0$, and $q = 0$. The ARMA model has $p > 0$, $d = 0$, and $q > 0$. Lastly, the Moving Average (MA) model has $p = 0$, $d = 0$, and $q > 0$.

To train each of these ARIMA-based models, a grid search was performed with p and q's possible values being 0, 24, 48, 72, and 96, and d's possible values being 0, 1, and 2. A different model was trained per topic/output-type pair. So, for example, the *(Maduro, # of new users)* pair had its own ARIMA, ARMA, AR, and MA models. The validation set was used to select the best model parameters for the test period and the *RMSE* metric was used to select the best model parameters.

### 3.5.11 State of the Art Comparisons

For the state-of-the-art comparisons, we used 3 variations of the tNodeEmbed embedding algorithm [58] because it has been shown to work well on link prediction tasks. Furthermore, embedding based approaches in general have been widely used for temporal network prediction tasks.

tNodeEmbed is a variation of node2vec [27, 50] that incorporates temporal information from the graph into its embeddings. It utilizes a rotation operation that aligns embeddings of nodes across time for more accurate predictions [58]. For clarity throughout this work, we named each tNodeEmbed variation based on the underlying embedding algorithm it uses for initialization. We refer to tNE-DeepWalk as the tNodeEmbed algorithm that is initialized with the DeepWalk graph algorithm. Likewise, tNE-node2vec-H and tNE-node2vec-S refer to the variations that are initialized with the *Homophilic* and *Structural* variations of node2vec, respectively.

In this work, each embedding represents a *(child, parent, topic, day)* tuple. Furthermore, these embeddings were each fed into one of 3 different fully-connected neural networks (1 per embedding approach). The output of one of the neural networks was a vector of 24-values representing the number of activities a particular child-user edge would perform under a particular topic over the next 24 hours. Since these models predicted activity at the user-to-user level of granularity, we aggregated these counts to topic and timestep granularity for consistent comparison to the VAM and statistical baseline models. An in-depth description of how the tNE neural networks were trained can be found in Appendix A.

The *DeepWalk* [50] algorithm uses random walks to create latent representations of nodes within a graph. These random walks are then used to create "sentences" that are fed into a *Word2Vec* [4] embedding algorithm. *Node2vec* is a variation of *DeepWalk* that introduces two new parameters, $p$ and $q$ that can be used to bias the random walks. When $p = q = 1$, the node2vec embedding is the same as a DeepWalk embedding, meaning the random walks have not been biased in any way. When $p=1$ and $q=0.5$, the node2vec embeddings are considered *Homophilic*, meaning that nodes that are close to each other have similar embeddings. When $p=1$ and $q=2$, the embeddings are *Structural*, meaning nodes with a similar role in the network (such as being a "hub node"), will have similar embeddings to one another [27].

In this work, we use the terms *node2vec-H* and *node2vec-S* to refer to our node2vec models that use the Homophilic (p=1, q=0.5) and Structural (p=1, q=2) parameters, respectively. For more details regarding how the tNE models were setup, refer to Appendix A.

## 3.6 Volume Prediction Results

### 3.6.1 Volume Prediction Metrics

In order to ensure that the time series predictions were correctly measured for accuracy, 6 different metrics were used over each of the 21 *forecast period of interest* instances in the test period spanning February 15th, 2019 to March 7th, 2019. Results were averaged across the 21 instances for each metric.

We used *RMSE* and *MAE* to measure how accurate each time series was in terms of "volume over exact time step". *Normalized Cumulative RMSE*, which converts the simulated and ground truth time series into cumulative sum time series, and then divides each by their respective maximum values was also used. This metric allows us to know how well predicted a time step was without considering the overall scale or "exact timing" of each value in the time series. This type of measurement is important because sometimes a time series would predict a burst within some range of timesteps, but not in the exact spot. However, knowing a burst of activities will occur within some range of timesteps is better than not knowing at all.

*Symmetric Absolute-Percentage-Error* (*S-APE*) measures how accurate the total number of events was for each model, without regard to the temporal pattern. Let $F$ be the forecast time series, and let $A$ be the actual time series. The formula is as follows.

$$SAPE = \frac{|sum(F) - sum(A)|}{sum(F) + sum(A)} * 100\%$$

The *Volatility Error (VE)* and *Skewness-Error (SkE)* metrics were used to measure how well the simulated times series captured the "burstiness" of the ground truth time series.

The *Volatility Error* is measured by calculating the standard deviation of both the ground truth and simulated time series, and then calculating their absolute difference. The *SkE* metric is measured by calculating the skewness of both the ground truth and simulated time series, and then calculating their absolute difference. The skewness statistic used utilizes the adjusted Fisher-Pearson standardized moment coefficient. It can be found at the top of page 7 in [23]. The formula is as follows:

$$G_1 = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} (\frac{x_i - \bar{x}}{s})^3$$

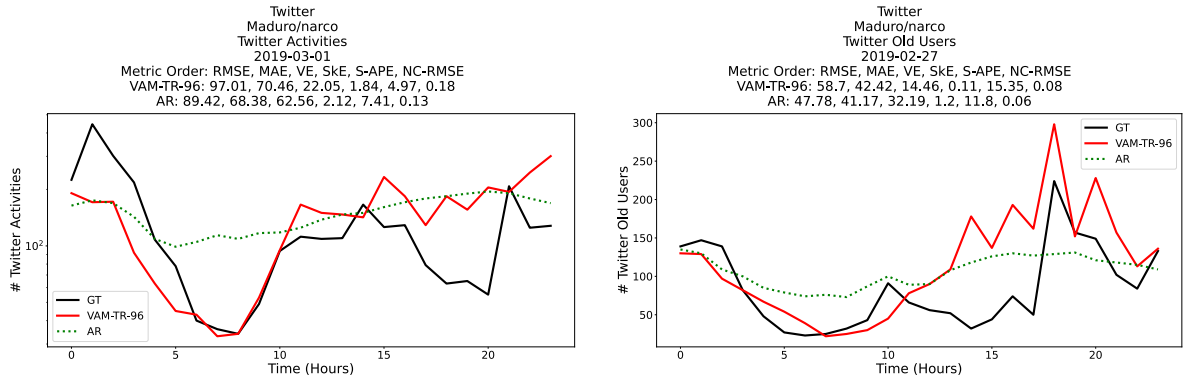
$G_1$ is the skewness. The variable $n$ is the sample size. The variables $x_i$, $\bar{x}$, and $s$ represent the value of an individual sample, the mean, and standard deviation, respectively.

### 3.6.2 Issues with Using Only RMSE and MAE as Metrics

RMSE and MAE are commonly used metrics for time series regression problems, however we note their limitations. When plotting the VAM models against the baseline models, we found that there are some instances in which the baseline time series has better RMSE and MAE results than the VAM prediction, however when visually inspecting the time series plots, the VAM models seem to better match the ground truth time series. For this reason, we also utilized 4 more metrics that measure other elements of time series prediction performance besides just the "exact timing" measurement of RMSE and MAE. Figure 3.7 contains 2 examples of this phenomenon.

In 3.7a, the RMSE and MAE of the AR model is 89.42 and 68.38, respectively, whereas for VAM it's 97.01 and 70.46, respectively. However, as one can see, visually the VAM model prediction (red) looks more similar to the ground truth (black) curve within the first 15 hours or so of the simulation. The VAM prediction manages to capture the major dip in the ground truth, unlike the AR prediction. This is captured in the prediction metrics in which VAM had a Volatility Error and Skewness Error of 22.05 and 1.84, versus the AR model's results of 62.56 and 2.12, respectively.

In 3.7b, a similar phenomenon can be observed. The AR model has better RMSE and MAE metrics than VAM (47.78 and 41.17 vs. VAM's 58.7 and 42.42, respectively), however, VAM has better VE and SkE metric results (VAM has 14.46 and 0.11 vs. the AR's 32.19 and 1.2, respectively).




(a) Maduro/Narco Activities on March 1st (b) Maduro/Narco Old Users on Feb 27th

Figure 3.7: RMSE and MAE plot examples for VAM and AR. Here are some examples in which a baseline model had better RMSE and MAE performance than the *VAM-XGB-TR*, but worse performance in other metrics. Visually, one can see that RMSE and MAE alone do not equate to time series that approximately match the pattern of the ground truth.

### 3.6.3 The Overall Normalized Volume Metric

Table 3.5 shows the the overall results for the models on the 6 aforementioned metrics. Table 3.6 shows the metric results per model on the RMSE, MAE, and VE metrics. Table 3.7 shows the metric results per model on the SkE, S-APE, and NC-RMSE metrics.

Since there were many useful metrics, we calculated 1 "overall" metric that represents how well each model performed across all 6 metrics, as shown in Table 3.5. We call this new metric the "Overall Normalized Metric Error (ONME)". It was calculated by creating 6 "metric groups", each comprised of the 14 model metric results for that particular metric. A similar "normalized error metric" was used in [24]. The model results within each of the 6 groups were normalized between 0 and 1 by dividing each model metric result by the sum

of all model metric results within that particular group. The models in each table are then sorted and ranked from lowest to highest *ONME*.

In order to illustrate how well each model performed against the best performing baseline we used a metric that we call the "Percent Improvement From Best Baseline" ($PIFBB$). These values represent, as a percent, how much the *ONME* improved from the best baseline, which in this case was the Persistence Baseline. The formula for this value is as follows:

$$PIFBB = 100\% * \frac{BestBaselineError - ModelError}{BestBaselineError}$$

The upper bound of $PIFBB$ is 100%, which occurs if a model's *ONME* is 0. This is clearly the best possible result. The lower bound for *ONME* is negative infinity because any given model could potentially perform infinitely worse than the best baseline.

### 3.6.4 Overall Metric Result Analysis

The best Volume-Prediction (VP) Module for Twitter belonged to the *VAM-XGB-TR-96* model. This was the XGBoost model trained on Twitter and Reddit data with a lookback factor of 96 hours. The *ONME* for this model was 0.05394, which was about a 17.53% improvement from the best baseline, the Persistence Baseline. Furthermore, all 3 of the Twitter and Reddit ($TR$) VAM models outperformed all 3 of the Twitter-only ($T$) VAM models. This suggests that the Reddit exogenous platforms contain information that can aid with prediction.

The ARIMA-based models (ARIMA, ARMA, MA, and AR), could not outperform the Persistence Baseline, despite its simplicity. The closest baseline to it was the Moving Average (MA) baseline, with a PIFBB of about -10.90%.

Lastly, despite being state-of-the-art approaches, the tNodeEmbed models did not outperform any of the 5 basic statistical baselines. The best tNodeEmbed model was the tNE-DeepWalk model, with a PIFBB score of about -42.31%, which was 12 percent lower

than the worst statistical baseline, ARIMA, which had a PIFBB score of about -34.92%. A plausible reason for the weak performance of this set of models is because they directly predict the user-to-user interactions, in contrast to the VAM Volume Prediction Module and ARIMA models that predict total hourly activity. Performing such a granular task makes it more difficult for these models to accurately predict the more macroscopic phenomenon of hourly user activity. Most of the user-to-user edges perform no activities at the hourly level, so when training models with such samples, the models are inclined to predict mostly 0 activity.

### 3.6.5 VAM XGBoost vs. VAM RNN

Since we observed that the best VAM XGBoost model had Twitter and Reddit features (VAM-XGB-TR-96), we then trained several RNN models with the same features in order to compare their performance. Table 3.8 shows these results. Similar to Table 3.5, there is an *Overall Normalized Metric Error (ONME)* metric, used to show the relative performance among all models, as well as a PIFBB score to show how well each model performed against the best baseline (Persistence Baseline). Table 3.9 contains the results for RMSE, MAE, and VE; and Table 3.10 contains the results for SkE, S-APE, and NC-RMSE.

The 4 different RNN models used were a GRU RNN, LSTM RNN, Bi-directional GRU RNN, and Bi-directional LSTM RNN. We were particularly interested in comparing the XGBoost VAM models to RNN VAM models because RNNs are among the most frequently used machine learning approaches for social media activity prediction as shown in [40, 33, 39, 30, 56, 58].

Despite the wide popularity of RNNs, we found that the XGBoost VAM model (VAM-XGB-TR-96) outperformed all RNN approaches. It had a PIFBB score of 17.47%. The best RNN model was trained with a GRU RNN (VAM-GRU-TR-96). It had a PIFBB score of 15.21%. Overall, the RNNs were able to strongly outperform the Persistence Baseline. The least accurate RNN (VAM-Bi-LSTM-TR-96) had a PIFBB score of 11.03%.

These results were found to be significant using the Wilcoxon Signed Rank Test with an alpha of 0.05. Table 3.11 contains the p-values.

### 3.6.6 Training Time Analysis

Tables 3.5 and 3.8 also show the training times for each model. Each XGBoost and ARIMA model was trained on a computer with an Intel Xeon E5-260 v4 CPU. Each CPU was comprised of 2 sockets, 8 cores, and 16 threads. Each computer had 128 GB of memory. The tNodeEmbed and VAM RNN models were trained on GeForce GTX 1080 Ti GPUs.

In addition to being the best-performing models, the XGBoost VAM models were also the quickest to train, with training times spanning from 3 to 7 minutes.

The RNN models took much longer to train, with the fastest model (VAM-LSTM-TR-96) taking 2 hours and 54 minutes, and the slowest model (VAM-GRU-TR-96) taking 4 hours and 36 minutes.

The Persistence Baseline has "n/a" marked as its training time because this model is trivially created by moving historical predictions forward. There is no training phase involved.

The ARIMA based models performed worse than the RNN models, and were even slower, taking anywhere from roughly 10 hours (AR) up to 26 hours (ARIMA).

The embedding models took the longest time to train, in addition to being the worst performing. This is because of the cost of creating the original embeddings themselves, and the cost of training neural networks with these embeddings as input features. Furthermore, the embedding methods predict at the hour and user-level, in contrast to the ARIMA models and VAM Volume-Prediction modules that predict the total number of users and activities hourly level. The fastest embedding model was the tNE-node2vec-S model, with about 47 hours of training time. The slowest model was the tNE-node2vec-H model, with almost 61 hours of training time.

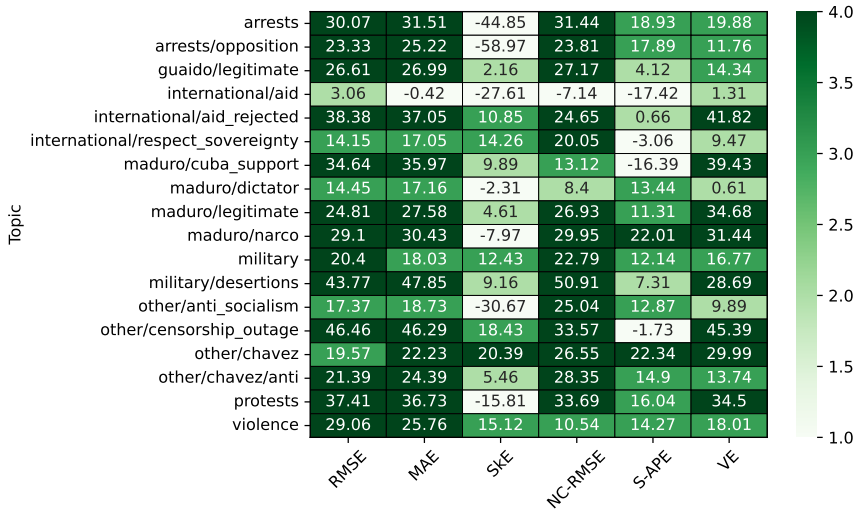



Figure 3.8: Here are the Vz19 VP Module metric results. The numbers represent VAM-TR-96's Percent Improvement From Best Baseline (PIFBB), which was the Persistence Baseline.

As one can see, the VAM XGBoost models are the best models for the time series prediction task, because they are both quick to train and highly accurate relative to the other models.

### 3.6.7 VAM-XGB-TR-96 Metric Results by Topic

We wanted to better understand model performance per topic. To that end, we compared the best VAM model's metric results per topic (VAM-XGB-TR), with the best baseline's metric results per topic (Persistence Baseline).

Figure 3.8 is a heatmap showing VAM-TR-96's metric results per topic. The value in each cell is the PIFBB score of VAM-TR-96 against the Persistence Baseline. White cells are instances in which VAM performed worse than the Persistence Baseline. There are then 3 shades of green to represent instances in which VAM-TR-96 outperformed the Persistence Baseline. The values in white cells are negative numbers. The lightest green cells indicate instances in which VAM's PIFBB was between 0 and 10%, the slightly darker green cells indicate instances in which VAM's PIFBB was between 10 and 20%, and lastly, the dark green cells indicate instances in which VAM's PIFBB was greater than 20%.

For the RMSE metric, VAM won against the best baselines on 18 out of 18 topics. For MAE, VAM won 17 out of 18 times; for Normalized Cumulative RMSE (NC-RMSE), VAM won 17 out of 18 times; for Symmetric Absolute Percentage Error (S-APE), VAM won 14 out of 18 times; for Skewness Error (SkE), VAM won 11 out of 18 times; and for Volatility Error (VE), VAM won 18 out of 18 times.

Overall, VAM outperformed the Persistence Baseline 95 out of 108 metric comparisons, or about 88% of the time. VAM performed particularly well at the "volume over time" metrics (RMSE MAE, and NC-RMSE), as well as the volatility metric (VE). It performed decently for the "magnitude" or "scale" metric (S-APE). Lastly, it struggled the most when using the Skewness Error metric, which measures the asymmetry of the time series.

Figure 3.9 shows the performance of the *VAM-XGB-TR-96* model against the 5 baselines on various topics and days. As one can see, VAM was able to more closely approximate the ground truth than the baseline models (with some error of course).

There were also some instances in which *VAM-XGB-TR-96* performed relatively poorly. Figure 3.10 contains examples. Notice in both examples, VAM missed huge spikes in activity that occurred in the ground truth. In Section 3.6.8, we performed analysis of all of the ground truth time series in the Vz19 dataset and found that VAM tended to perform worse on time series with relatively high Skewness and a relatively high Coefficient of Variation (COV). More details on this phenomenon are in that section.

### 3.6.8 Time Series Attribute Analysis

We sought to better understand what types of time series VAM performed better on. So, to that end, we performed some time series clustering analysis. We clustered the ground truth time series in the test set, and analyzed the S-APE and NC-RMSE metrics for each cluster.

S-APE and NC-RMSE in particular were chosen because unlike RMSE, MAE, VE, and SkE metrics, the S-APE and NC-RMSE metrics have a defined range of possible values.

Table 3.5: ONME Twitter VP results.

| **Twitter Volume Prediction Results for Overall Normalized Metric Error (ONME)** | | | | |
|---|---|---|---|---|
| **Rank** | **Model** | **ONME** | **PIFBB (%)** | **Training Time** |
| **1** | **VAM-XGB-TR-96** | **0.05394** | **17.52938** | 0 hrs, 7 min |
| 2 | VAM-XGB-TR-48 | 0.05397 | 17.48034 | 0 hrs, 3 min |
| 3 | VAM-XGB-TR-72 | 0.05461 | 16.49569 | 0 hrs, 6 min |
| 4 | VAM-XGB-T-96 | 0.05468 | 16.3866 | 0 hrs, 7 min |
| 5 | VAM-XGB-T-48 | 0.0552 | 15.5942 | 0 hrs, 3 min |
| 6 | VAM-XGB-T-72 | 0.05546 | 15.20475 | 0 hrs, 5 min |
| 7 | Persistence Baseline | 0.0654 | 0.0 | n/a |
| 8 | MA | 0.07253 | -10.90084 | 11 hrs, 53 min |
| 9 | ARMA | 0.07632 | -16.70079 | 24 hrs, 26 min |
| 10 | AR | 0.08168 | -24.89077 | 9 hrs, 43 min |
| 11 | ARIMA | 0.08824 | -34.91949 | 26 hrs, 22 min |
| 12 | tNE-DeepWalk | 0.09307 | -42.31413 | 53 hrs, 37 min |
| 13 | tNE-node2vec-S | 0.0972 | -48.62708 | 47 hrs, 2 min |
| 14 | tNE-node2vec-H | 0.09769 | -49.37005 | 61 hrs, 37 min |

Table 3.6: RMSE, MAE, and VE Twitter VP results.

| **Twitter Volume Prediction Results for RMSE, MAE, and VE** | | | |
|---|---|---|---|
| **Model** | **RMSE** | **MAE** | **VE** |
| **VAM-XGB-TR-96** | 675.08053 | 482.97939 | 358.63956 |
| VAM-XGB-TR-48 | 666.11639 | 472.04979 | 356.92729 |
| VAM-XGB-TR-72 | 681.71863 | 483.31309 | 369.61879 |
| VAM-XGB-T-96 | 682.29561 | 488.55522 | 370.65342 |
| VAM-XGB-T-48 | 691.26144 | 490.17236 | 376.60796 |
| VAM-XGB-T-72 | 696.14708 | 494.04453 | 376.05549 |
| Persistence Baseline | 888.9082 | 619.26606 | 454.85759 |
| MA | 922.13789 | 701.64627 | 444.88704 |
| ARMA | 1068.57479 | 823.89253 | 531.86923 |
| AR | 1174.3006 | 904.72248 | 605.11153 |
| ARIMA | 1321.44676 | 1034.54112 | 658.98357 |
| tNE-DeepWalk | 854.14389 | 655.98387 | 468.61746 |
| tNE-node2vec-S | 947.90737 | 751.56235 | 494.58402 |
| tNE-node2vec-H | 852.48697 | 658.4185 | 489.23354 |

Table 3.7: SkE, S-APE, and NC-RMSE Twitter VP results.

| **Twitter Volume Prediction Results for SkE, S-APE, and NC-RMSE** | | | |
|---|---|---|---|
| **Model** | **SkE** | **S-APE** | **NC-RMSE** |
| **VAM-XGB-TR-96** | 0.99388 | 26.91419 | 0.11566 |
| VAM-XGB-TR-48 | 0.98988 | 27.50714 | 0.11939 |
| VAM-XGB-TR-72 | 0.97503 | 27.51952 | 0.11992 |
| VAM-XGB-T-96 | 0.99218 | 27.63607 | 0.11627 |
| VAM-XGB-T-48 | 0.95685 | 28.29505 | 0.12152 |
| VAM-XGB-T-72 | 0.97999 | 28.0642 | 0.12094 |
| Persistence Baseline | 0.96809 | 29.42484 | 0.15699 |
| MA | 1.38811 | 37.35475 | 0.14152 |
| ARMA | 1.24775 | 34.36105 | 0.1353 |
| AR | 1.37021 | 34.54514 | 0.12422 |
| ARIMA | 1.21444 | 37.36525 | 0.14517 |
| tNE-DeepWalk | 1.36376 | 81.31111 | 0.27117 |
| tNE-node2vec-S | 1.33858 | 85.27032 | 0.26504 |
| tNE-node2vec-H | 1.38445 | 86.87079 | 0.30195 |

Table 3.8: VAM XGBoost and RNN comparisons for ONME. The Persistence Baseline is also included for performance comparison.

| **VAM XGBoost and RNN Comparisons - ONME** | | | | |
|---|---|---|---|---|
| **Rank** | **Model** | **ONME** | **PIFBB (%)** | **Training Time** |
| **1** | **VAM-XGB-TR-96** | **0.15615** | **17.47352** | 0 hrs, 7 min |
| 2 | VAM-GRU-TR-96 | 0.16043 | 15.21455 | 4 hrs, 36 min |
| 3 | VAM-LSTM-TR-96 | 0.16087 | 14.98211 | 2 hrs, 54 min |
| 4 | VAM-Bi-GRU-TR-96 | 0.16501 | 12.79083 | 4 hrs, 9 min |
| 5 | VAM-Bi-LSTM-TR-96 | 0.16833 | 11.03596 | 4 hrs, 28 min |
| 6 | Persistence Baseline | 0.18921 | 0.0 | n/a |

Table 3.9: VAM XGBoost and RNN comparisons for RMSE, MAE, and VE. The Persistence Baseline is included for performance comparison.

| **VAM XGBoost and RNN Comparisons - RMSE, MAE, and VE** | | | |
|---|---|---|---|
| **Model** | **RMSE** | **MAE** | **VE** |
| **VAM-XGB-TR-96** | 675.08053 | 482.97939 | 358.63956 |
| VAM-GRU-TR-96 | 650.9308 | 463.02403 | 369.95245 |
| VAM-LSTM-TR-96 | 682.18312 | 492.39017 | 397.65227 |
| VAM-Bi-GRU-TR-96 | 694.0311 | 506.28413 | 403.98542 |
| VAM-Bi-LSTM-TR-96 | 692.84236 | 501.98398 | 394.13319 |
| Persistence Baseline | 888.9082 | 619.26606 | 454.85759 |

Table 3.10: VAM XGBoost and RNN comparisons for SkE, S-APE, and NC-RMSE. The Persistence Baseline is included for performance comparison.

| **VAM XGBoost and RNN Comparisons - SkE, S-APE, and NC-RMSE** | | | |
|---|---|---|---|
| **Model** | **SkE** | **S-APE** | **NC-RMSE** |
| **VAM-XGB-TR-96** | 0.99388 | 26.91419 | 0.11566 |
| VAM-GRU-TR-96 | 1.07092 | 29.73336 | 0.11889 |
| VAM-LSTM-TR-96 | 0.84077 | 30.67232 | 0.12543 |
| VAM-Bi-GRU-TR-96 | 0.96457 | 31.445 | 0.1173 |
| VAM-Bi-LSTM-TR-96 | 0.9701 | 33.22728 | 0.12872 |
| Persistence Baseline | 0.96809 | 29.42484 | 0.15699 |

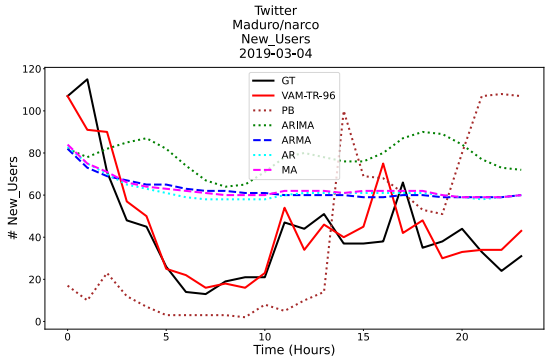
Table 3.11: VAM-RNN vs. VAM-XGB p-values. Wilcoxon Signed Rank Test was used.

| **VAM-RNN vs. VAM-XGB P-Values** | |
|---|---|
| **RNN Model** | **p_value** |
| VAM-GRU-TR-96 | 0.00097 |
| VAM-Bi-LSTM-TR-96 | 2.13582e-20 |
| VAM-LSTM-TR-96 | 0.00099 |
| VAM-Bi-GRU-TR-96 | 3.79186e-11 |

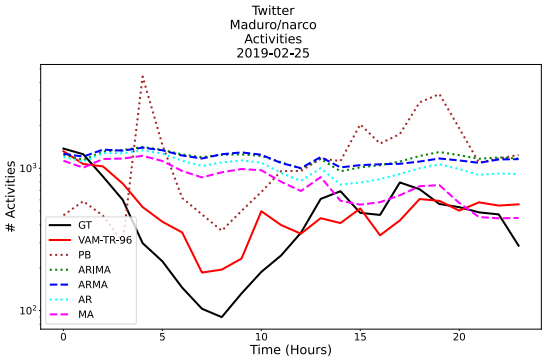
The S-APE metric can only be between 0 and 100%, and the NC-RMSE metric can only be between 0 and 1. Since these metrics have a pre-defined range, it is easier to compare results across clusters.

After selecting our metrics, 3 time series attributes were chosen for clustering: Skewness, Volume, and Coefficient of Variation (COV). As previously mentioned, Skewness is a measure of the asymmetry of a time series. Volume, is simply the total scale of a time series. It is calculated by adding up all the values in a particular time series. Lastly, Coefficient of Variation measures the "volatility " of a time series. It is calculated by dividing the standard deviation by the mean. The formula is as follows: $COV = \frac{\sigma}{\mu}$.

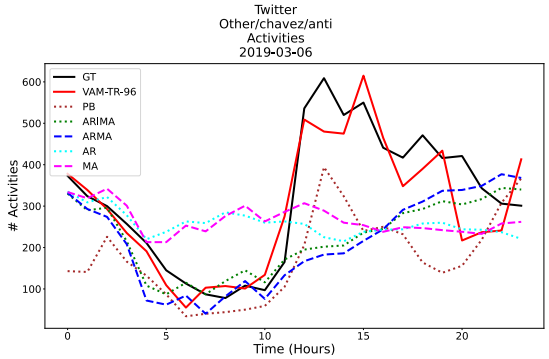
We calculated, for all time series, the Skewness, Volume, and COV. Recall that in the test set there are 18 topics, 21 days, and 3 output types. As a result, we calculated all 3 time series attributes for all 1,134 time series, yielding 3 *time series attribute groups*, each with 1,134 values.



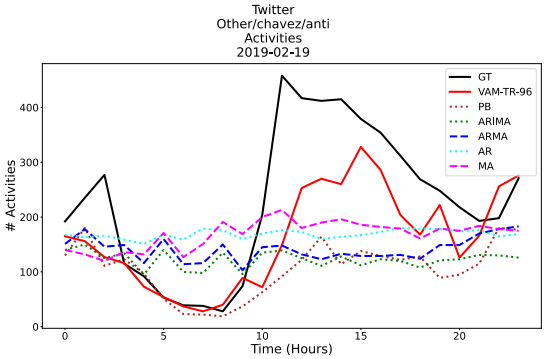

(a) Maduro/Narco New Users on March 4th



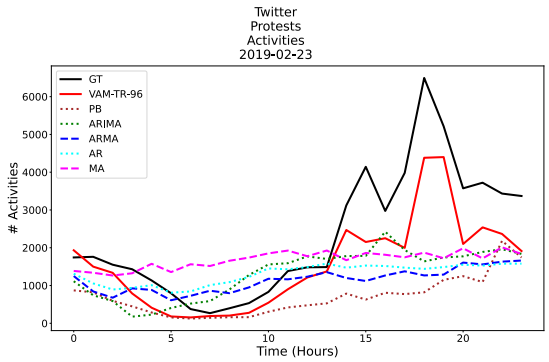

(b) Maduro/Narco Activities on Feb. 25th



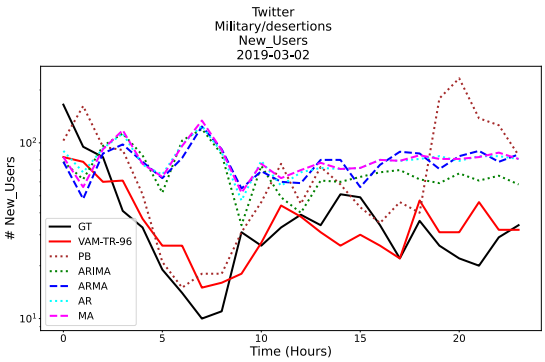

(c) Other/Chavez/Anti Activities on March 6th




(d) Other/Chavez/Anti Activities on Feb. 19th



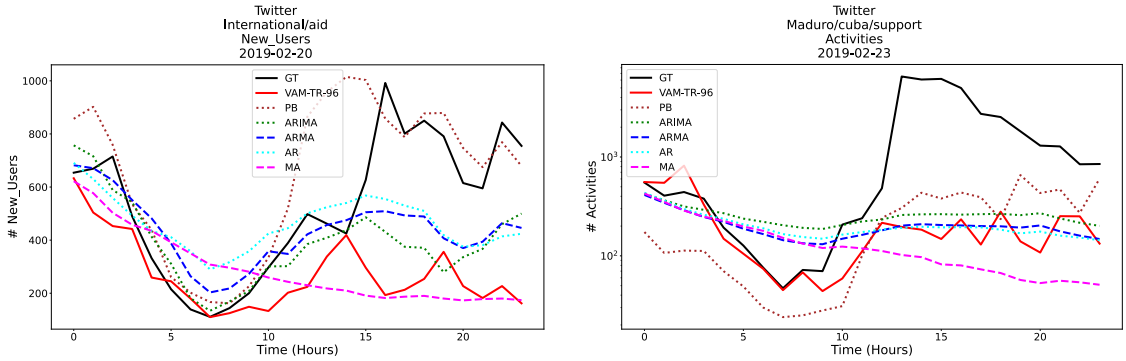

(e) Protests Activities on Feb. 23rd




(f) Military/Desertions New Users on March 2nd

Figure 3.9: Some 24-hour time series predictions plots. The solid black curves are ground truth time series and the solid red curves are the *VAM-XGB-TR-96* predicted time series. The dotted and dashed curves are different baseline time series.

We then created 6 clusters using these time attribute values. For each *time series attribute group*, we calculated the 80th percentile of that group and then created two clusters for a particular time series attribute, called a *High* or *Low* cluster. Using this methodology yielded




(a) Maduro/Narco New Users on March 4th (b) Maduro/Narco Activities on Feb. 25th

Figure 3.10: Two particular instances VAM-TR-96 performed poorly.

6 time series clusters. They are the (1) High-Skewness Cluster, (2) Low-Skewness Cluster, (3) High-COV Cluster, (4) Low-COV Cluster, (5) High-Volume Cluster, and (6) Low-Volume Cluster.

We further explain how each cluster works with a few examples. The High-Skewness Cluster is the cluster containing all time series whose Skewness was above 80th percentile. The Low-Skewness Cluster is the cluster containing all the time series that were below the 80th percentile.

Each of the High-Clusters contained 227 time series, and each of the Low-Clusters contained 907 time series.

We then retrieved the metric scores for each of the time series in each cluster and calculated the median values for each cluster. We created 2 barplots using these values. Figure 3.11 is the barplot for the S-APE results, and Figure 3.12 is the bar plot for the NC-RMSE results. In both Figures, there are 6 bars, each representing the median S-APE or NC-RMSE for each of the 6 clusters. Lower bars obviously indicate better results. The blue bar (first bar in each pairing) represents the High-Cluster error for a particular time series attribute. The orange bar (second bar in each pairing) represents the Low-Cluster error.

In both figures, one can see that VAM has lower S-APE and NC-RMSE for both the Skewness and COV Low-Clusters. The High-Cluster errors for these 2 attributes are quite higher in comparison. This indicates that VAM performs much better on time series with

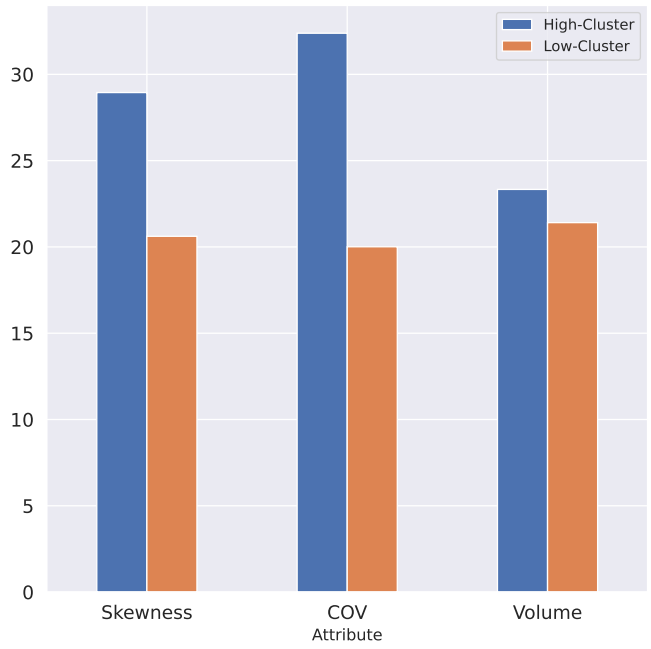



Figure 3.11: High and low cluster results for S-APE.

relatively low Skewness and COV, or asymmetry and volatility. Intuitively, this makes sense because a time series with more "erratic" activity would obviously be more difficult to predict compared to one with more "stationary" activity.

According to both figures, the amount of volume, or "scale" of a time series plays less of a role in its predictability. VAM performs slightly better in terms of S-APE for low-volume time series, and very slightly better in terms of NC-RMSE for low-volume time series. However, the improvement is much less apparent in comparison to the Skewness and COV cluster results.

## 3.7 User Assignment Methodology

### 3.7.1 Overview

Recall the user-assignment task for VAM. Once the *VP-Module* predicts matrix $\hat{\boldsymbol{Y}}$ for *topic-timestep* pair, $(q, T)$, the task for the *UA-Module* is to use $\hat{\boldsymbol{Y}}$ and the graph history set,

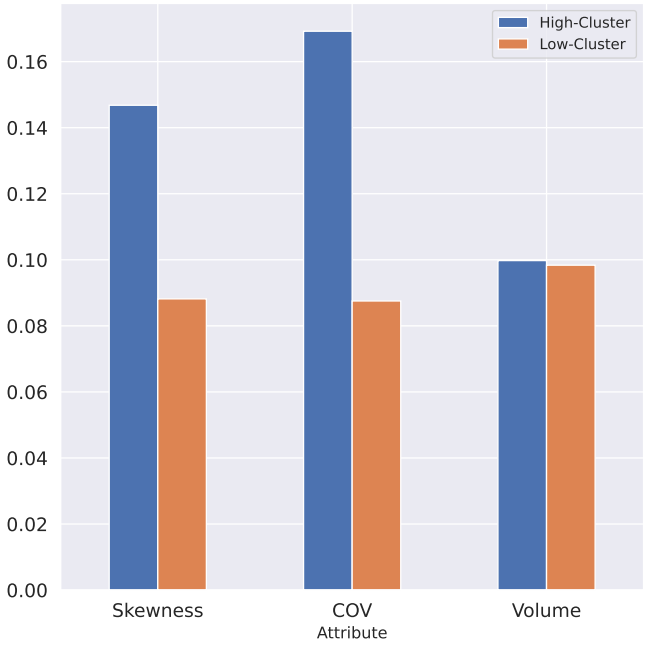



Figure 3.12: High and low cluster results for NC-RMSE.

$\{G\}_{t=1}^{t=T}$ to predict the future graph sequence, $\{\hat{G}^{future}\}_{t=1}^{t=S}$. As mentioned earlier, $\hat{G}^{future}$ is an approximation of the ground truth graph set, $G^{future}$.

The user-assignment is done in the following way. For $S$ iterations, a graph $G_s^{future}$ ($s \leq S$) is generated and added to the overall final $G^{future}$ sequence. Eight main data structures are used to aid in the user-assignment task as described in the following subsections.

### 3.7.2 The Recent History Table

Firstly, there's a recent history table, called $H^{recent}$. This is a table containing event tuples generated using information from $G$. Each tuple contains the following information: (1) the child (acting) user, (2) the parent (receiving) user, (3) the number of interactions between child and parent at some timestep $t$, (4) a flag indicating whether the child is new at timestep $t$, and (5) a flag indicating whether the parent is new at timestep $t$.

$H^{recent}$ is known as a "recent" history table because it is made from only the most recent graph snapshots from $G$. The lookback factor parameter $L^{user}$ is used to determine the number of snapshots to use. For example, if $L^{user} = 5$, then only the 5 most recent graphs

in sequence $G$ will be used to make $H^{recent}$. The assumption here is that recent history is all that is needed to make temporal network predictions.

### 3.7.3 Old and New Users

The next two data structures are the set of selected old users, $\hat{O}_s$ and set of generated new users, $\hat{N}_s$. Note that VAM "knows" the number of old and new users because that is what was predicted by the *Volume-Prediction* Module.

### 3.7.4 Old and New User Probability Tables

The 4th data structure is the *Old User Activity Probability Table* ($W^{old}$). It is a table containing each old user's probability of being active (e.g. tweeting/retweeting) at some timestep $t$.

The 5th data structure is the *New User Archetype Table* ($W^{new\_arch}$). This table models how different "archetypes" of new users have behaved in the past. These archetypes are generated using recently active user information from $H^{recent}$. These attributes are the (1) probability of acting and (2) probability of being influential (e.g. being retweeted). This archetype table is then used to create the 6th data structure $W^{new}$ which contains the activity and influence probabilities for the users in $\hat{N}_s$.

### 3.7.5 Old and New Parent Tables

The last two tables are the old and new user parent tables, $D^{old\_parent}$ and $D^{new\_parent}$, respectively. These are hash tables in which each key is a user, and the value is a table containing (1) a list of that user's historical "parents" (a.k.a. users that the user of interest is most likely to retweet) and (2) the probability that the user of interest will retweet or comment that particular parent.

These 8 data structures are used to predict each $G_s^{future}$ in the temporal sequence $G^{future}$. Algorithm 3.1, labelled *Assign_Users* contains the pseudocode for the User-Assignmnent algorithm. For an in-depth explanation of the algorithm, please refer to Appendix B.

---

**Algorithm 3.1** Assign_Users

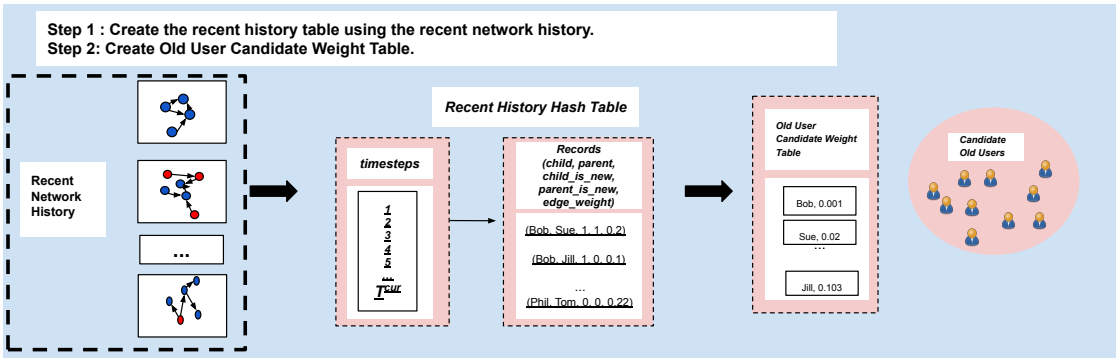
**Input:** The full temporal graph $G$; the number of output timesteps to be predicted $S$; the user assignment lookback factor $L^{user}$; the volume prediction matrix $\hat{\boldsymbol{Y}} \in \mathbb{R}^{3\times S}$; Old user index *old_idx*; New user index *new_idx*; Activity index *act_idx*

**Output:** The predicted temporal graph sequence $\hat{G}^{future}$

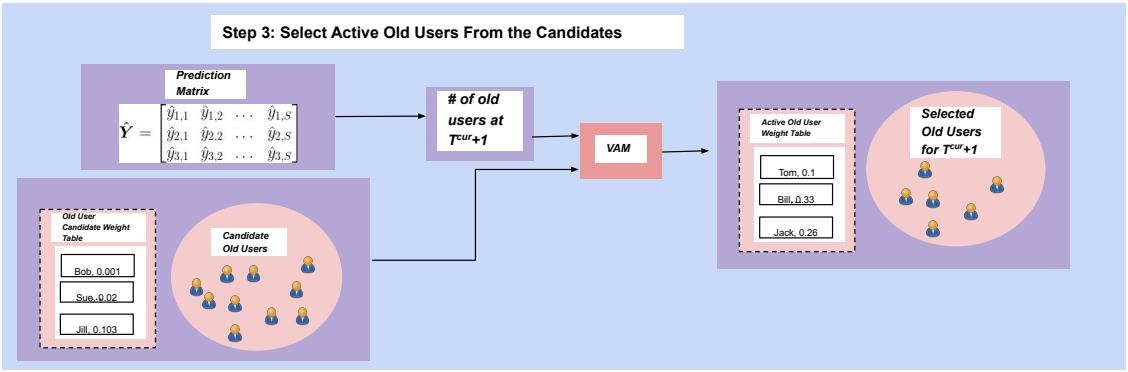
$G^{recent} = Get\_Recent\_Temporal\_Graph(G, L^{user})$
Initialize $G^{future}$ as an empty array
**for** $s = 1$ up to $S$ **do**
    $num\_old\_users \leftarrow \hat{\boldsymbol{Y}}[old\_idx][s]$ # predicted old users at $s$
    $num\_new\_users \leftarrow \hat{\boldsymbol{Y}}[new\_idx][s]$ # predicted new users at $s$
    $num\_acts \leftarrow \hat{\boldsymbol{Y}}[act\_idx][s]$ #predicted actions at $s$
    $H^{recent} \leftarrow Get\_Recent\_History\_Table(G^{recent})$ #get recent history
    $W^{old\_cand} \leftarrow Get\_Active\_Old\_User\_Candidates(H^{recent})$ #get "pool" of potentially active old users
    $W^{old}, \hat{O}_s \leftarrow Get\_Most\_Likely\_Active\_Old\_Users(W^{old\_cand})$ # select most likely active users from pool
    $\hat{N}_s \leftarrow Generate\_New\_Users(num\_new\_users)$ #create IDs to represent new users and store in a set
    $W^{new\_arch} \leftarrow Get\_New\_User\_Archetype\_Table(H^{recent})$ #create new user arch. table
    $W^{new} \leftarrow Assign\_Attributes\_to\_New\_Users(W^{new\_arch}, num\_new\_users)$ #assign attributes to new users
    $D^{old} \leftarrow Create\_Old\_User\_Parent\_Table(H^{recent}, \hat{O}_s)$ # get old users' most likely parents
    $D^{new} \leftarrow Create\_New\_User\_Parent\_Table(H^{recent}, \hat{N}_s, W^{new\_arch})$ # get new users' most likely parents
    $\hat{G}_s^{future} \leftarrow Create\_Links(\hat{O}_s, \hat{N}_s, num\_acts, W^{old}, W^{new}, D^{old}, D^{new})$ #perform link prediction with user sets
    Append $\hat{G}_s^{future}$ to $\hat{G}^{future}$ #append newly predicted graph to array
    $G^{recent} = Get\_Recent\_Temporal\_Graph(G^{recent}, \hat{G}_s^{future}, L^{user})$ #update $G^{recent}$ with predicted $\hat{G}_s^{future}$ graph
**end for**
**return** $\hat{G}^{future}$

---

Figures 3.13 on page 68, 3.14 on page 69, and 3.15 on page 70 contain illustrations of all 7 steps of the User-Assignment Algorithm.

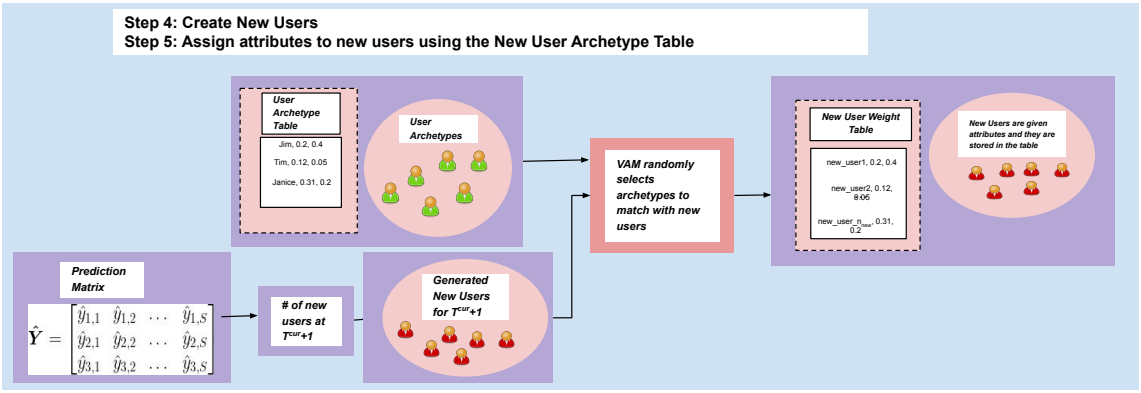



(a) Step 1: Use the recent network history, $G^{recent}$ to create the recent history hash table, $H^{recent}$. Step 2: Use the recent history table to create the *Old User Candidate Weight Table*, $W^{old\_cand}$.

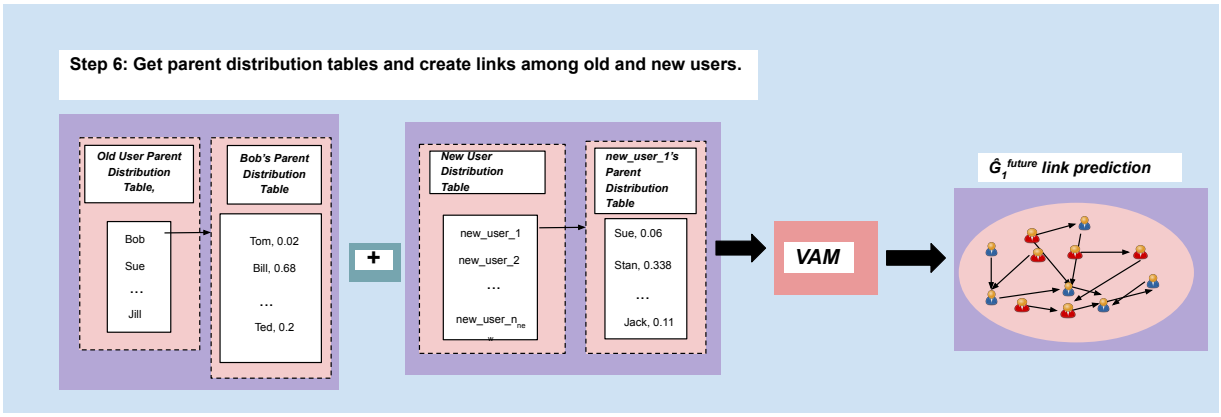



(b) Step 3: Use the old user candidate weight table and the $\hat{Y}$ volume matrix to select the active old users from the candidates.

Figure 3.13: Steps 1-3 of the VAM User-Assignment algorithm.

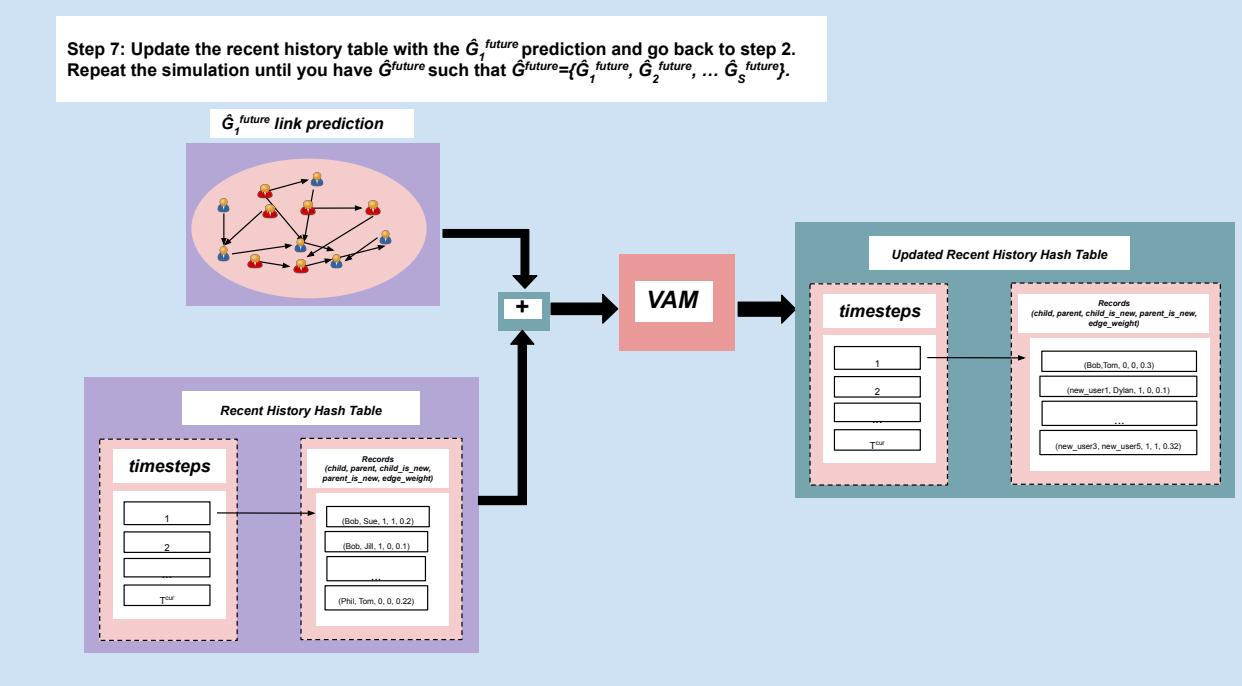



(a) Step 4: Use the $\hat{Y}$ volume matrix to generate new users. Step 5: Create the new user archetype table to assign attributes to the generated new users.




(b) Step 6: Create the new and old user parent distribution tables. Use these tables to assign edges among the old and new users using probabilities. This new set of links makes up the graph, $\hat{G}_1^{future}$.

Figure 3.14: Steps 4-6 of the VAM User-Assignment algorithm.




(a) Step 7: Use $\hat{G}_1^{future}$ to update the recent history table, $H^{recent}$. Go back to step 2 and continue the simulation algorithm until $VAM$ has created $\hat{G}_1^{future}$ such that $\hat{G}^{future} = \hat{G}_1^{future}, \hat{G}_2^{future}, \ldots \hat{G}_S^{future}$.

Figure 3.15: Step 7 of the VAM User-Assignment algorithm.

## 3.8 User-Assignment Metrics

### 3.8.1 Jaccard Similarity for Old Users

To measure how well VAM predicted old users, we used both the unweighted and weighted Jaccard Similarity metric, which is also known as the Ruzicka Similarity [13].

These metrics were used to measure how well VAM predicted influential old users. We define influential users as users who are retweeted at least once in a given timestep. Unweighted Jaccard Similarity was used to measure if a user was retweeted at least once or not. Weighted Jaccard Similarity was used to measure the similarity of the predicted number of times a user was retweeted to the ground truth number of times a user was retweeted.

Let $A$ represent the set of the actual old users within a particular hour, and let $P$ represent the predicted set of old users within a particular hour. The unweighted Jaccard similarity is trivially calculated as the cardinality of the intersection of $A$ and $P$ divided by the cardinality of the union of $A$ and $P$.

Furthermore, let $\mathbf{a}$ and $\mathbf{p}$ represent vectors that contain the weights of each user in the $A$ and $P$ sets, respectively. For example, $\mathbf{a}_k$ represents the weight of user $A_k$ from the $A$ set. With this in mind, the Weighted Jaccard Similarity is defined as follows:

$$J(\mathbf{a}, \mathbf{p}) = \frac{\sum_k min(\mathbf{a}_k, \mathbf{p}_k)}{\sum_k max(\mathbf{a}_k, \mathbf{p}_k)}$$

.

### 3.8.2 Defining Success for New User Prediction

Since our task also involves predicting the creation and activity of new users, in addition to old users, defining and measuring predictive success becomes a bit more difficult. Since we do not "know" the names of a new user before they appear in the ground truth, it is impossible to exactly match a new user that VAM generates, with a new user that exists in the ground truth. So, in order to work around this issue, we measure success using

more macroscopic views of the network, specifically the Page Rank Distribution and the Complementary Cumulative Degree Histogram (CCDH).

### 3.8.3 Page Rank and Earth Mover's Distance

The Page Rank score [11] measures how influential a particular node is upon the entire network. In our experiments, we calculated Page Rank on the weighted indegree of our networks. If VAM properly simulated the activities of old and new users, that means that VAM's simulated network Page Rank Distribution should closely approximate the ground truth network's Page Rank Distribution. In order to measure the distance between the predicted and actual Page Rank distributions we used the Earth Mover's Distance Metric [52].

### 3.8.4 The CCDH and Relative Hausdorff Distance

The Complementary Cumulative Degree Histogram (CCDH) of a graph $G$ is defined as $(N(k))_{k=1}^{\inf}$, in which $N(k)$ denotes the number of vertices of degree *at least* $k$ [2]. It is closely related to the more well-known concept of *degree distribution*. In our experiments, we calculated the CCDH on the unweighted indegree distribution of the ground truth and simulated networks. Success is defined by how closely the predicted CCDH matches the ground truth CCDH.

In order to measure the distance between the predicted network CCDH and the ground truth network CCDH we used the Relative Hausdorff (RH) Distance. Previous work has shown the RH-Distance to be a suitable metric for measuring the distance between two CCDHs [57].

## 3.9 User-Assignment Results

In this section we discuss the results of VAM's User-Assignment module. Since the *VAM-XGB-TR-96* model was the best performing VAM model for the Volume Prediction task,

we used that VAM model's volume predictions for the user-assignment task. We also used a *User Assignment Lookback Factor* $L^{user}$ of 24 hours. So in other words, the past 24 hours of user activity history was used when VAM assigned actions to users. We found 24 hours to work the best.

### 3.9.1 Multiple Trials

Since VAM's User-Assignment algorithm is probabilistic, it was run 5 times with 5 different seeds used for initializations. The 3 user-assignment metrics (Jaccard Similarity, EMD, and RHD) were then calculated across each of the 5 trials and averaged together. These averaged results are shown.

### 3.9.2 Overall Jaccard Similarity Results

Table 3.12 shows model performance for the user prediction measurements (weighted and unweighted Jaccard similarity). The results were calculated across all 18 topics and averaged together for both the Weighted and Unweighted scores. Then, a final *Average JS* score was calculated by averaging the Weighted and Unweighted scores for each model. The PIFBB score was calculated against the best baseline, which was the Persistence Baseline.

As seen in the table, the *VAM-XGB-TR-96* model had the best Average Jaccard Similarity (JS) score of about 0.12, and a PIFBB of 36.89%. The Persistence Baseline came in 2nd with an Average JS of about 0.09. The tNodeEmbed models were much worse, with Average JS scores of around 0.009, and PIFBB scores of around -88%.

### 3.9.3 Overall EMD and RHD Results

Table 3.13 shows model performance using the network measurements (EMD and RHD). Similar to the Jaccard Similarity results, each metric was calculated individually for all 18 topics and then averaged together.

Table 3.12: Jaccard Similarity results for each model. The PIFBB shows the Percent Improvement From Best Baseline score against the Persistence Baseline.

| **Weighted, Unweighted, and Average Jaccard Similarity Results** | | | | | |
|---|---|---|---|---|---|
| **Rank** | **Model** | **Weighted** | **Unweighted** | **Average** | **PIFBB (%)** |
| **1** | **VAM-XGB-TR-96** | 0.080489 | 0.155461 | **0.117975** | **36.892828** |
| 2 | Persistence Baseline | 0.057867 | 0.114494 | 0.086181 | 0.0 |
| 3 | tNE-DeepWalk | 0.009935 | 0.009983 | 0.009959 | -88.443896 |
| 4 | tNE-node2vec-S | 0.009927 | 0.009956 | 0.009941 | -88.464474 |
| 5 | tNE-node2vec-H | 0.009926 | 0.009954 | 0.00994 | -88.466521 |

Table 3.13: Model network measurement comparisons. The best baseline used for the PIFBB column was the Persistence Baseline.

| **Model Network Structure Results** | | | | | |
|---|---|---|---|---|---|
| **Rank** | **Model** | **EMD** | **RHD** | **ONME** | **PIFBB (%)** |
| **1** | **VAM-XGB-TR-96** | 0.036104 | 1.00641 | **0.10142** | **15.321424** |
| 2 | Persistence Baseline | 0.043553 | 1.092286 | 0.11977 | 0.0 |
| 3 | tNE-DeepWalk | 0.048136 | 7.177918 | 0.258682 | -115.982142 |
| 4 | tNE-node2vec-H | 0.048211 | 7.178201 | 0.258855 | -116.126741 |
| 5 | tNE-node2vec-S | 0.049282 | 7.180201 | 0.261273 | -118.145748 |

Similar to the volume result Table 3.5, we calculated an ONME metric to obtain relative model performance, and a PIFBB score to obtain relative improvement over the best baseline (Persistence Baseline).

Once again, the *VAM-XGB-TR-96* was the best model, with an ONME of about 0.10 and PIFBB of 15.3%. The Persistence Baseline came in 2nd place with an ONME of 0.13. The tNodeEmbed models were much worse with ONMEs of around 0.26 and PIFBB scores spanning from about -116 to -118%.

As previously discussed, the tNE models predicted user-to-user activity directly at the hourly granularity. This is problematic because most users in most hours perform little to no actions. This issue is reflected in the tNE models' poor performance. VAM has an advantage because first macroscopic activities are predicted, and then the user-assignment algorithm uses these volume predictions to predict which most likely users will perform the most likely actions. By using this methodology, VAM avoids the sparse activity problem.

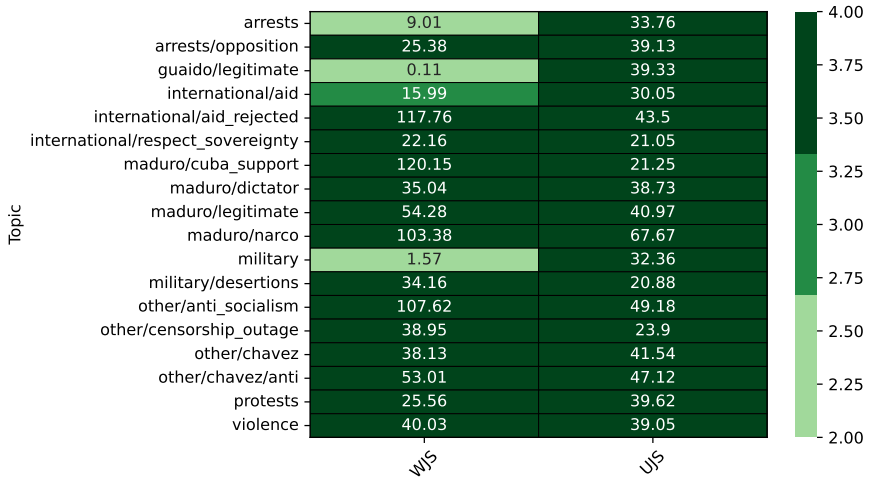



Figure 3.16: Weighted and Unweighted Jaccard Similarity Metric results. The numbers represent VAM-TR-96's Percent Improvement From Best Baseline (PIFBB), which was the Persistence Baseline.

### 3.9.4 Per Topic Result Analysis

Since the *VAM* model and the *Persistence Baseline* were the two best models, we wanted to do a more granular comparison between the two. To that end, we compared the metric results on a per-topic basis in a similar fashion to the per-topic comparison done in Section 3.6.7. To that end, we created 2 heatmaps in a similar fashion to Figure 3.8. Figure 3.16 is the heatmap for the Jaccard Similarity results, and Figure 3.17 is the heatmap for the network measurement results.

We counted the number of times *VAM* outperformed the *Persistence Baseline* for each of the 18 topics for each of the 4 user-assignment metrics.

For Weighted Jaccard Similarity, *VAM* outperformed the *Persistence Baseline* on 18 out of 18 topics. Similarly for the Unweighted Jaccard Similarity, *VAM* outperformed the *Persistence Baseline* on 18 out of 18 topics. In Figure 3.16, cells in which the WJS or UJS PIFBB were between 0-10% were colored light green, while cells that are medium green indicate instances in which the PIFBB was between 10-20%. Dark cells indicate instances in which the PIFBB was above 20%. As one can see in this table, there were many instances in

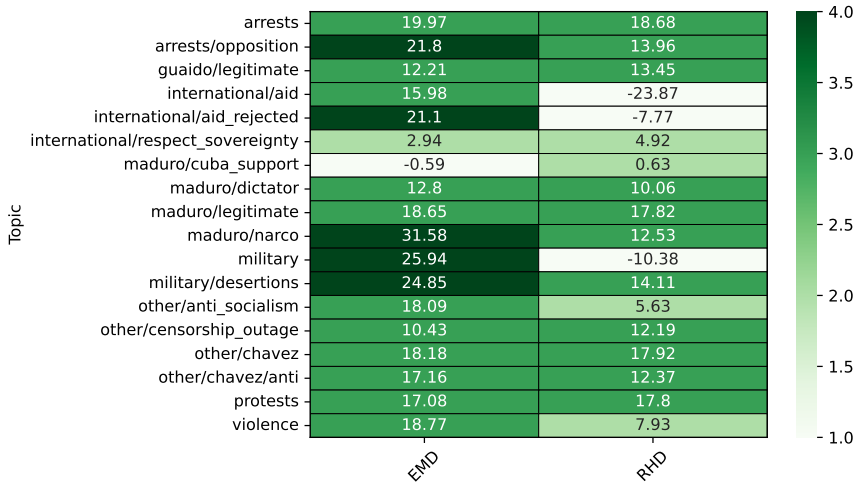



Figure 3.17: EMD and RHD metric results. The numbers represent VAM-TR-96's Percent Improvement From Best Baseline (PIFBB), which was the Persistence Baseline.

which VAM's WJS and UJS PIFBB scores were well over 20%, indicating that VAM strongly outperformed the Persistance Baseline in the old user prediction task.

For the Earth Mover's Distance metric, *VAM* had 17 out of 18 topic wins. Lastly, for Relative Hausdorff Distance, *VAM* had 15 out of 18 wins. In Figure 3.17, cells in which the WJS or UJS PIFBB were negative were colored white. The lightest green cells indicate instances in which the PIFBB was between 0-10%. Cells that are slightly darker green indicate instances in which the PIFBB was between 10-20%. And lastly, dark cells indicate instances in which the PIFBB was above 20%. As one can see in the table, there were many instances in which the EMD and RHD PIFBB scores were between 10-20%, but not as many instances in which the PIFBB scores were over 20%. This shows that VAM's EMD and RHD scores still outperformed the Persistence Baseline, but to a lesser extent than the Weighted and Unweighted Jaccard Similarity metrics.

In summary, *VAM* was good at predicting which old user edges would exist for a given timestep. It was also quite good at predicting what "type" of user would be active at each timestep in terms of Page Rank influence (as measured by Earth Mover's Distance). VAM was slightly worse (but still good overall) at predicting the unweighted indegree distribution of the users (As measured by the Relative Hausdorff Distance).

### 3.9.5 User-Assignment Runtime Information

Similar to the Volume-Prediction module, the User-Assignment module of VAM was run on computers with an Intel Xeon E5-260 v4 CPU. Each CPU was comprised of 2 sockets, 8 cores, and 16 threads. Each computer had 128 GB of memory. The User-Assignment Module was run in parallel over 5 computers (1 per trial). The average runtime of the User-Assignment Algorithm across the 5 trials was about 2 hours and 13 minutes, which is quite reasonable considering that there were 18 topics and millions of edges. Since there were 21 days in the test period, on average the User-Assignment algorithm took about 6.33 minutes to simulate the activities for 1 day (or 24 hours) across all 18 topics.

## 3.10 Conclusions and Future Work

In this chapter we presented the *Volume Audience Match* simulator, VAM. It is an end-to-end simulator of user activity in social media platforms that utilizes time series prediction and probabilistic link prediction to estimate future activity of both old and new users. In this work, VAM was used to predict both overall and user-level activity from the recent Venezuela political crisis on a per-topic basis.

On the Volume-Prediction task, VAM was shown to have strong performance against multiple widely used statistical models (ARIMA, ARMA, AR, MA, Persistence), as well as several tNodeEmbed models. As previously mentioned, it outperformed the best baseline (Persistence Baseline) on 95 out of 108 topic-metric pairs, or about 88% of the time. On the User-Assignment task, VAM strongly outperformed the Persistence Baseline on 68 out of 72 topic-metric pairs, or about 94% of the time. With refinement, VAM could be used as an alert system for potential future real world activity.

Future work includes a variety of tasks. Firstly, we would aim to use 2 machine learning models in the *User Assignment Module* to predict the most likely active users and the final link predictions. Perhaps these models could outperform the weighted random sampling approach that VAM's *User-Assignment* module currently employs. For the *Volume Prediction*

*Module* we would try other machine learning models, such as Transformer neural networks, and compare their performance to the XGBoost and RNN VAM models.

# Chapter 4: VAM and the CPEC Dataset[1]

## 4.1 Introduction

In Chapter 3, VAM was introduced, and its predictive power was shown on a Twitter dataset related to the 2019 Venezuelan Political Crisis. In this chapter, we show VAM's performance on a different dataset, namely, the Chinese-Pakistan Economic Corridor (CPEC) Twitter dataset.

The contributions of this chapter are as follows. Firstly, we show that the Volume Audience Match algorithm can be used to predict user-level activity on a dataset related to international economics (Chinese-Pakistan Economic Corridor), a dataset different than the Venezuelan Political Crisis dataset used in Chapter 3. By using a different dataset, this lends more credence to the idea that VAM is a generalizable framework for predicting user-level activity on social media networks. Secondly, we show that VAM outperforms a multitude of baselines in the time series prediction and user-level link prediction tasks. We used VAM to predict Twitter tweets, retweets, quotes, and replies, unlike the work of Chapter 3, which only predicted tweets and regular retweets (no quotes or replies).

Thirdly, similar to Chapter 3, we show that VAM can predict the creation of new users, unlike many previous works that only focus on the prediction of old users. Fourthly, we show that using external social media features from Reddit and YouTube can aid with predicting future Twitter activity, unlike Chapter 3, which only focused on external Reddit features. Fifthly, we examine the time series features used in the Volume Prediction module of VAM to better understand which features help the most with the prediction task.

[1]The material in this chapter has been previously published on arxiv in [42] by the same author of this dissertation. The permission is shown in Appendix C.

Lastly, we compare VAM's CPEC and Vz19 metric results, and find that in both platforms, VAM performs better on time series with low Skewness and low Cofficient of Variation values.

## 4.2 Motivation for Predicting CPEC Twitter Activity

In this chapter, we apply the VAM simulation system to another domain, which is the Chinese-Pakistan Economic Corridor (CPEC), an infrastructure initiative between China and Pakistan. There are 10 topics in this domain. If VAM could in fact, accurately predict future user-activity related to the CPEC initiative, that would allow some government or organization to have a better understanding of public opinion related to CPEC. For example, if VAM predicts that there will be an increase in tweets related to the *benefits/development/jobs* or *benefits/development/roads* topics, this lets some government or corporate entity know that people may be focusing on potential benefits of the CPEC initiative such as an more jobs or better roads. Beyond the domain-specific applications, by applying VAM to another dataset besides the Venezuelan Political dataset of the previous chapter, we show that VAM can serve as a general social media activity simulator.

## 4.3 New and Old User Information

Similar to the Venezeulan Twitter dataset of 3, there are a large number of new users in the CPEC dataset. Table 4.1 contains the average hourly proportion of new to old users in the Twitter dataset. As shown in the table, for some topics, there is a particularly high frequency of average new users per hour. For example, in controversies/china/uighur, on average, every hour 78.72% of the active users were new and 21.28% were old. Topics such as this are the reason we aim to use *VAM* to predict both new and old user activity, unlike most previous works that only focus on old/previous user activity prediction.

Table 4.1: Average hourly proportion of new to old users per topic.

| **Twitter Hourly Active New/Old Frequencies** | | |
|---|---|---|
| **Topic** | **Avg. New User Freq (%)** | **Avg. Old User Freq (%)** |
| controversies/china/uighur | 78.72 | 21.28 |
| controversies/pakistan/students | 75.0 | 25.0 |
| benefits/jobs | 66.67 | 33.33 |
| opposition/propaganda | 59.74 | 40.26 |
| controversies/pakistan/baloch | 50.0 | 50.0 |
| leadership/bajwa | 47.62 | 52.38 |
| benefits/development/energy | 47.5 | 52.5 |
| benefits/development/roads | 42.55 | 57.45 |
| controversies/china/border | 34.94 | 65.06 |
| leadership/sharif | 28.26 | 71.74 |

## 4.4 Problem Statements

As previously discussed in Chapter 3, there are 2 problems *VAM* attempts to solve, the *Volume Prediction Problem* and the *User-Assignment Problem*.

The *Volume Prediction Problem* is to predict the overall volume of Twitter activities. Note that we do not distinguish whether a particular action is a tweet, retweet, quote, or reply because the focus of this work is to predict the overall volume of Twitter activities. For a given topic $q$, at some time step $T$, the volume prediction task is to predict 3 time series of length $S$ which are (1) the number of activities, (2) the number of new users, and (3) the number of old users.

The *User-Assignment Problem* is as follows. Given the 3 time series predicted from the Volume Prediction problem, and the user-to-user interaction history, predict the future user-to-user edges among the old and new users. This problem is framed in a similar fashion to the previous chapter.

Table 4.2: CPEC Twitter and YouTube post counts per topic. Twitter counts refer to tweets, retweets, quotes, and replies. YouTube posts refer to videos and comments.

| **Twitter and YouTube Topic Counts** | | |
|---|---|---|
| **Topic** | **Twitter Counts** | **Youtube Counts** |
| controversies/china/border | 1,509,000 | 1,081 |
| controversies/pakistan/baloch | 344,289 | 856 |
| opposition/propaganda | 309,378 | 455 |
| benefits/development/roads | 189,082 | 937 |
| leadership/sharif | 185,851 | 648 |
| controversies/china/uighur | 173,431 | 440 |
| benefits/development/energy | 160,874 | 436 |
| leadership/bajwa | 144,277 | 494 |
| benefits/jobs | 112,769 | 267 |
| controversies/pakistan/students | 37,891 | 6 |

## 4.5 Data Collection

Data was collected and anonymized by Leidos. Annotators and subject matter experts (SMEs) worked together to annotate an initial set of 4,997 tweet and YouTube comments. These posts were related to 10 different topics. All topics are related to the Chinese-Pakistan Economic Corridor. The time period was from April 2, 2020 to August 31, 2020.

Similar to the methodology described in Chapter 3, a BERT model was then used to label topics for 3,166,842 Twitter posts (tweets, retweets, quotes, and replies) and 5,620 YouTube posts (videos and comments). Table 4.2 shows the counts of the Twitter and YouTube posts per topic. BERT was not applied to the Reddit data, so the Reddit data used as additional features in this work is not split by topics.

## 4.6 Volume Prediction Methodology

### 4.6.1 Data Processing

Our training period was from April 2, 2020 to August 10, 2020 (4 months). The validation period was August 11 to August 17th, 2020 (1 week). Lastly, the test period was August 18, 2020 to August 31, 2020 (2 weeks).

Similar to Chapter 3, each sample represents a *topic-timestep* pair. The input features represent multiple time series leading up to a given timestep of interest $T$. Also, a 1 hot vector of size 10 was used to indicate which topic each sample represented.

Table 4.3 shows the time series features and Table 4.4 shows the feature vector sizes for each model trained. The *model* column shows the name of the model. The abbreviation represents the platform time series features used to train the particular model. "T", "Y", and "R" represent Twitter, YouTube, and Reddit respectively. The numbers represent the hourly length of the time series input to each model. However, note that the 3 output time series of each model are each of length 24 in order to maintain consistency in evaluation. For example, the *VAM-TR-72* model is a model trained on Twitter and Reddit time series that are all of length 72.

Using Table 4.3, one can see that these time series indices would be 1-3, 7-9, and 13. These are 7 different time series features. Also recall that there are 10 static features (for the 1 hot vector). By adding these values together, one can see that this model had 7*72 + 10 = 514 features, as shown in the table.

There were 31,210 training samples used for each model - 1,450 validation samples, and 140 test samples. There were 140 test samples because of 10 topics and 14 days for testing. However, for training and validation, we wanted to generate as many samples as possible so our models had adequate data. So, for those datasets, we created samples by creating "days" both in terms of hour and day, which was the same approach used in Chapter 3.

In total, we trained 12 different VAM models. Each model was trained on a different combination of platform features which were some combination of Twitter, Reddit, and YouTube. Furthermore, we also used different *volume lookback factors* ($L^{vol}$). The $L^{vol}$ parameter determines the length of each time series input to the model. For example, the *VAM-TRY-24* model was the model trained on Twitter, Reddit, and YouTube time series, all of length 24.

### 4.6.2 XGBoost

Similar to the setup used in the previous chapter, *VAM*'s *Volume Prediction module*, which we call $\Phi$, is comprised of multiple XGBoost models. As previously discuseed, XGBoost is a gradient-boosting method that utilizes ensembles of CART trees in order to perform classification or regression predictions with lower bias and variance than standalone CART trees [16].

### 4.6.3 Baselines Used

Similar to Chapter 3, VAM was compared to 5 baseline models, which are the Persistence Baseline, ARIMA, ARMA, AR, and MA models [9]. However, in this chapter, we omit comparisons to the VAM-RNN and tNodeEmbed [58] models because in Chapter 3 we established that the XGBoost VAM models outperform both.

Table 4.3: All possible CPEC time series feature categories.

| **Time Series Index** | **Time Series Description** |
|---|---|
| **1** | New user volume time series for a given topic in Twitter. |
| **2** | Old user volume time series for a given topic in Twitter. |
| **3** | Activity volume time series for a given topic in Twitter. |
| **4** | New user volume time series for a given topic in YouTube. |
| **5** | Old user time series for a given topic in YouTube. |
| **6** | Activity volume time series for a given topic in YouTube. |
| **7** | Activity volume time series across all topics in Twitter. |
| **8** | New user volume time series across all topics in Twitter. |
| **9** | Old user volume time series across all topics in Twitter. |
| **10** | Activity volume across all topics in YouTube. |
| **11** | New user volume time series across all topics in YouTube. |
| **12** | Old user volume time series across all topics in YouTube. |
| **13** | Activity volume time series in Reddit. |

Table 4.4: CPEC Twitter volume model input sizes.

| **Model Input Feature Sizes** | |
|---|---|
| **Model** | **Features** |
| VAM-TR-72 | 514 |
| VAM-TY-72 | 874 |
| VAM-TRY-48 | 634 |
| VAM-TR-48 | 346 |
| VAM-TRY-72 | 946 |
| VAM-T-72 | 442 |
| VAM-TY-48 | 586 |
| VAM-T-48 | 298 |
| VAM-TR-24 | 178 |
| VAM-TRY-24 | 322 |
| VAM-TY-24 | 298 |
| VAM-T-24 | 154 |

## 4.7 Volume Prediction Results

### 4.7.1 Metrics Used

Similar to Chapter 3, RMSE, MAE, SkE, VE, S-APE, and NC-RMSE were used as metrics.

### 4.7.2 Metric Results

Table 4.5 contains the 6 metric results for the 12 VAM models and 5 baselines. Similar to Chapter 4, we also used the *Overall Normalized Metric Error (ONME)* in order to get an overall picture of how well each model performed across all 6 metrics. Table 4.6 contains the ONME results.

Note that ARMA was the best baseline model because it had the lowest Overall Normalized Metric Error out of all 5 baselines. We wanted to know how well each model performed in comparison to this baseline. To that end, we also created the *ONME Percent Improvement From Best Baseline Metric (PIFBB)*, in a similar fashion to Chapter 3.

Recall that it is calculated as follows.

$$PIFBB = 100\% * \frac{BestBaselineError - ModelError}{BestBaselineError}$$

According to Table 4.6, the best model was the *VAM-TR-72* model. This was the VAM model trained on both Twitter and Reddit features with a lookback factor of 72. Its ONME Percent Improvement From the Best Baseline (ARMA) was 16.92%. It is noteworthy that the 5 best models all used Reddit and/or YouTube features in addition to Twitter features in order to predict the Twitter time series. This suggests that external platform features from Reddit and YouTube can be helpful in predicting future events on Twitter.

It would also appear that Reddit is slightly more helpful than YouTube for predicting Twitter activity. The top model uses only exogenous features from Reddit. The YouTube model is in 2nd place. Also, the YouTube models tend to have worse rankings in Table 4.6 than the Reddit ones. One might assume the YouTube models would have higher rankings because the YouTube features are tied to topics, while the Reddit ones are not. One possible reason for this phenomenon could be that most Reddit posts are comprised of written text, which is quick to create. YouTube content, on the other hand, is driven by user-uploaded videos. Even though there are many comments on YouTube, these comments cannot exist without videos to post to. Since it is much easier and faster to write a text post on Reddit, than it is to film, edit, and upload a video on YouTube, Reddit users are more able to quickly react to real-world events and discuss them, than people on YouTube. As a result, Reddit might have more informative exogenous features for a predictive model to use.

Lastly, we note that the 4 worst VAM models all had lookback factors of 24. In contrast, the models with lookback factors of 48 or 72 had higher rankings. This suggests that the longer lookback periods of 48 or 72 are more helpful for accurate Twitter time series prediction.

Table 4.5: CPEC VP results - RMSE, MAE, VE, SkE, S-APE, and NC-RMSE.

| VAM and Baseline Volume Prediction Results | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Rank** | **Model** | **RMSE** | **MAE** | **VE** | **SkE** | **S-APE** | **NC-RMSE** |
| **1** | **VAM-TR-72** | 63.7693 | 45.77 | 35.8454 | 1.0726 | 37.9726 | 0.1253 |
| 2 | VAM-TY-72 | 65.7877 | 47.0955 | 35.0636 | 0.9546 | 37.6423 | 0.1353 |
| 3 | VAM-TRY-48 | 66.2068 | 47.181 | 34.2751 | 1.0059 | 37.5466 | 0.1322 |
| 4 | VAM-TR-48 | 66.5991 | 47.284 | 36.5073 | 1.0468 | 37.6235 | 0.1283 |
| 5 | VAM-TRY-72 | 64.0651 | 45.8827 | 35.7631 | 1.1476 | 38.2717 | 0.1287 |
| 6 | VAM-T-72 | 63.5627 | 45.6483 | 36.9797 | 1.1979 | 37.9557 | 0.127 |
| 7 | VAM-TY-48 | 66.5644 | 47.7133 | 35.566 | 1.0358 | 38.5408 | 0.137 |
| 8 | VAM-T-48 | 64.0599 | 46.0157 | 36.7059 | 1.1699 | 38.338 | 0.1292 |
| 9 | VAM-TR-24 | 64.8761 | 47.0618 | 38.0413 | 1.1434 | 40.2569 | 0.1235 |
| 10 | VAM-TRY-24 | 65.1584 | 47.2283 | 37.6308 | 1.1141 | 41.0268 | 0.1289 |
| 11 | VAM-TY-24 | 65.7277 | 47.6674 | 37.7781 | 1.1192 | 40.9077 | 0.1311 |
| 12 | VAM-T-24 | 65.3621 | 47.4219 | 38.0849 | 1.1958 | 40.079 | 0.1289 |
| 13 | ARMA | 72.5972 | 55.1047 | 39.4702 | 1.6593 | 42.9807 | 0.143 |
| 14 | PB | 85.6484 | 61.8182 | 42.5324 | 0.9628 | 38.525 | 0.1764 |
| 15 | ARIMA | 71.414 | 54.6664 | 38.9028 | 1.8779 | 45.0738 | 0.1602 |
| 16 | AR | 71.0034 | 54.5305 | 39.9082 | 2.2177 | 42.3746 | 0.1393 |
| 17 | MA | 79.2893 | 61.0594 | 42.7798 | 2.1334 | 44.4995 | 0.1432 |

Figure 4.1 shows an example of VAM-TR-72's strong performance against the various baselines.

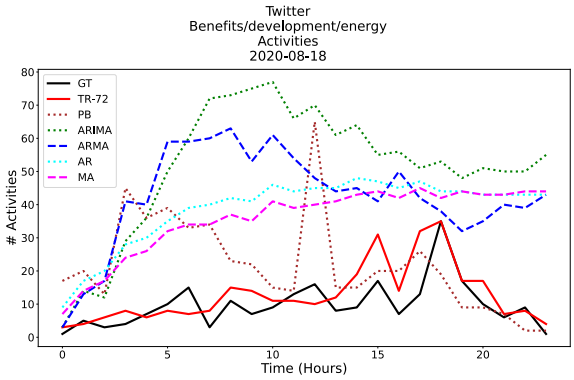
### 4.7.3 Per-Topic Analysis

We also analyzed the results of each metric across each topic. Figure 4.2 is a heatmap that contains this analysis. Each cell represents VAM's PIFBB score against ARMA. Similar to the previous chapter, white cells represent instances in which VAM performed worse than ARMA. Light green cells represent instances in which the PIFBB score was between 0 and 10%. Darker green cells represent instances in which VAM's PIFBB was between 10-20%. Lastly, the darkest green cells represent instances in which VAM's PIFBB score was over 20%.

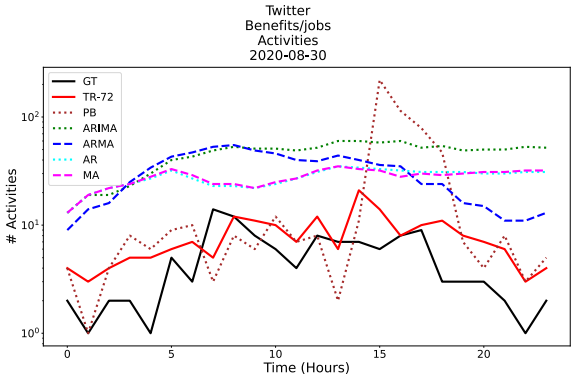
VAM outperformed ARMA for RMSE on 7 out of 10 topics; MAE on 9 out of 10 topics, SkE on 4 out of 10 topics, NC-RMSE on 7 out of 10 topics; S-APE on 3 out of 10 topics;

Table 4.6: CPEC Volume Prediction results for ONME.

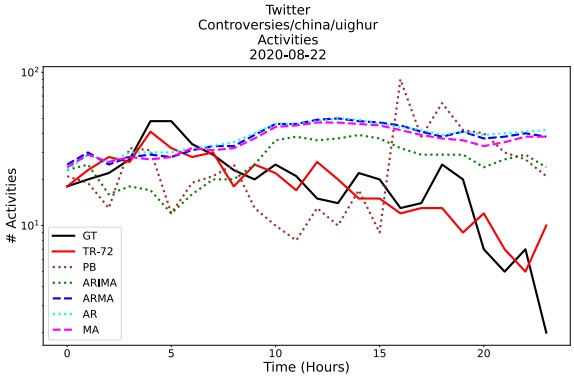
| **VAM and Baseline Volume Prediction Results - ONME** | | | |
|---|---|---|---|
| **Rank** | **Model** | **Overall Normalized Metric Error** | **ONME PIFBB (%)** |
| **1** | **VAM-TR-72** | **0.0539** | **16.9263** |
| 2 | VAM-TY-72 | 0.054 | 16.7831 |
| 3 | VAM-TRY-48 | 0.054 | 16.7577 |
| 4 | VAM-TR-48 | 0.0547 | 15.676 |
| 5 | VAM-TRY-72 | 0.0548 | 15.4899 |
| 6 | VAM-T-72 | 0.0552 | 14.9155 |
| 7 | VAM-TY-48 | 0.0553 | 14.7477 |
| 8 | VAM-T-48 | 0.0553 | 14.7338 |
| 9 | VAM-TR-24 | 0.0558 | 13.9192 |
| 10 | VAM-TRY-24 | 0.0561 | 13.4203 |
| 11 | VAM-TY-24 | 0.0565 | 12.8468 |
| 12 | VAM-T-24 | 0.0567 | 12.5483 |
| 13 | ARMA | 0.0648 | 0.0 |
| 14 | PB | 0.0649 | -0.0431 |
| 15 | ARIMA | 0.0678 | -4.6265 |
| 16 | AR | 0.0684 | -5.5195 |
| 17 | MA | 0.0718 | -10.7239 |




(a) Benefits/Dev/Energy




(b) Benefits/Jobs




(c) Controverseries/China/Uighur

Figure 4.1: Examples of *VAM-TR-72* model against the baselines. The red curves represent VAM's predictions, the black curves represent the ground truth, and the other curves represent the 5 baseline models.

and lastly VE on 5 out of 10 topics. Overall, it outperformed ARMA on 35 out of 60, or about 58% of all topic-metric pairs. We note that this Volume Prediction result is much lower than the roughly 88% score of Chapter 3.

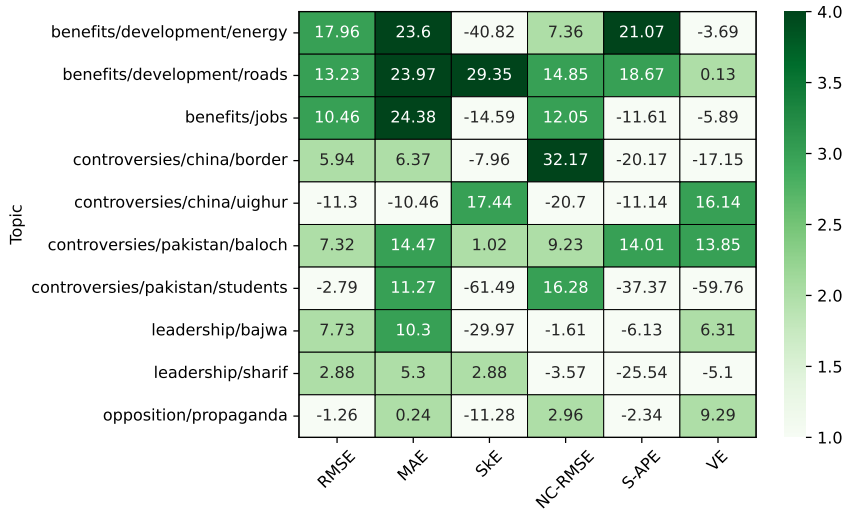



Figure 4.2: CPEC Volume Prediction metric results. The numbers represent VAM-TR-72's Percent Improvement From Best Baseline (PIFBB), which was the Persistence Baseline.

In summary, VAM did well on the metrics that focus on scale over the exact timestep (as measured by RMSE and MAE). It also performed decently on the metric that focuses on temporal pattern without regard to scale (NC-RMSE). It struggled on the metrics related to asymmetry and volatility (SkE and VE). Lastly, it performed poorly in terms of overall magnitude (S-APE).

### 4.7.4 Temporal Feature Importances

In Figure 4.3 we show a bar plot of the temporal feature importances of the XGBoost models for the *number of actions* output category for the *VAM-TR-72* model. The feature importances are calculated by adding up the number of times a feature is used to split the data across all trees and was calculated using the XGBoost library [16]. In this figure we refer to that output category as *Num. Twitter Actions For Topic*.

Along the Y-axis one can see the name of each feature category. There are 6 time series feature categories, 3 for the "global count" time series (the ones labelled with "All Topics"), and 3 categories for the "Twitter-topic" pair time series (the ones labelled with "For Topic").

We normalized all the feature category importance values between 0 and 1. These normalized values are what is shown in each bar plot.

As one can see, for the *VAM-TR-72* model, the *Num. Twitter Old Users For Topic* input time series is the most helpful time series for predicting the output time series *Num. Twitter Actions For Topic*. In other words, according to this plot, if one wished to predict the number of actions for the topic *benefits/jobs* (for example) at some future timestep, the most useful input time series would be the number of old user time series for *benefits/jobs*. In second place in terms of importance, is the feature category *Num. Twitter Activity Users For Topic*, and in third place is the feature category *Num. Twitter Old Users For All Topics*. The *Num. Reddit Actions* was the 5th most important feature category, ahead of *Num. Twitter New users For Topic*.

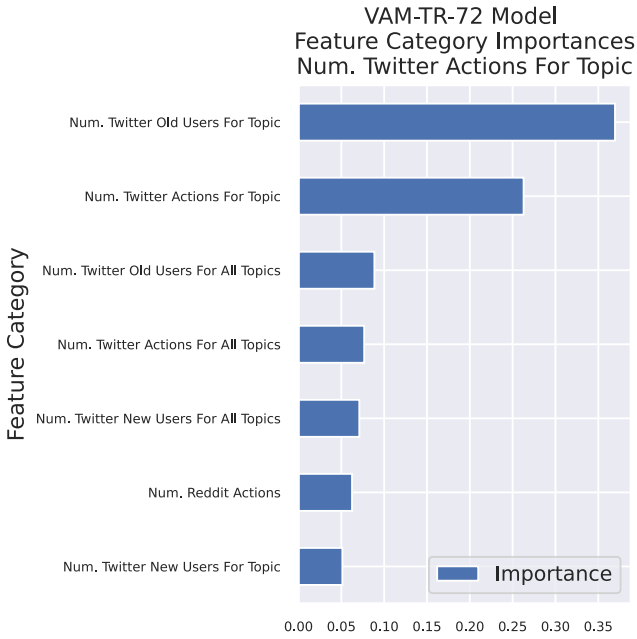



Figure 4.3: VAM-TR-72 feature importances.

## 4.8 Time Series Attribute Analysis

### 4.8.1 Vz19 and CPEC Time Series Attribute Comparison

As previously mentioned, within the CPEC dataset, VAM outperformed the best baseline (ARMA) on 35 out of 60 topic-metric pairs (58%) for the Volume-Prediction task. Although this is decent performance, it is much worse than the roughly 88% topic-metric wins that VAM had over the best baseline (Persistence Baseline) of the Vz19 dataset for the Volume-Prediction task.

We wanted to understand the disparity in success, so to that end, we compared the time series attributes of the Vz19 and CPEC datasets. Table 4.7 contains this comparison. Three attributes were analyzed: the median Skewness, Coefficient of Variation, and Volume of the time series in each domain.

The test set time series attributes were used to retrieve these values. Vz19's test set had 18 topics, 21 days, and 3 output-types, so 1,134 time series in total. CPEC had 10 topics, 14 days, and 3 output-types, so 420 time series in total. The time series attribute values were calculated for each domain's set of time series. In order to better compare the Skewness and COV attributes between the two domains, the time series were Min-max normalized between 0 and 1 before calculating Skewness and COV. Volume, of course, was not normalized. The median values were then calculated for each time series attribute, which are the values one can see in Table 4.7.

Also note in Table 4.7, there is a column called "Percent Difference from Vz19 to CPEC". The values in this column indicate the percent increase or decrease of each median time series attribute from Vz19 to CPEC.

Let $P$ represent the percent difference from Vz19 to CPEC. Furthermore, let $x^{CPEC}$ represent the median time series attribute value for CPEC and let $x^{Vz19}$ represent the median

time series attribute value for Vz19. The formula for the percent difference is as follows:

$$P = \frac{x^{CPEC} - x^{Vz19}}{x^{Vz19}}$$

.

As one can see in the table, Vz19's median Skewness and median COV (0.9418 and 0.8034, respectively) are lower than CPEC's values (1.274 and 0.9494, respectively). The median Skewness of the CPEC time series was 35.27% higher than the median Skewness of the Vz19 dataset. The median COV of the CPEC dataset was 18.17% higher than the median Skewness of the Vz19 dataset. In other words, CPEC's time series were much more asymmetrical and volatile compared to Vz19's time series, overall. Intuitively, this stands out as a reason for VAM's worse performance on CPEC. If a time series is more "erratic", it would be harder to predict.

It is also notable that the median Volume of CPEC's time series was much smaller than the median Volume of Vz19's time series. The percent difference from Vz19 to CPEC in this category was -88.46%. This is also another potential reason for VAM's worse performance on CPEC.

Figure 4.4 is a bar plot visualization of this comparison. There are 3 pairs of bars for Skewness, COV, and Volume. The blue bar (1st bar) in each pair represents the median time series attribute for Vz19. The orange bar (2nd bar) in each pair represents the median time series attribute for CPEC. As one can see in the bar plot, similar to the table, the median Skewness and median COV in Vz19 are lower than the median Skewness and median COV in CPEC. Also, one can see that Vz19's median time series volume was much higher than CPEC's. Note that the bar pairs are each normalized between 0 and 1 for easier visualization and comparison. Table 4.7, of course, contains the original raw values.

Table 4.7: Vz19 and CPEC attribute analysis.

| **Vz19 and CPEC Median Time Series Attribute Analysis** | | | |
|---|---|---|---|
| **Attribute** | **Vz19** | **CPEC** | **Percent Difference From Vz19 to CPEC (%)** |
| Skewness | 0.9418 | 1.274 | 35.27 |
| COV | 0.8034 | 0.9494 | 18.17 |
| Volume | 3582.0 | 413.5 | -88.46 |

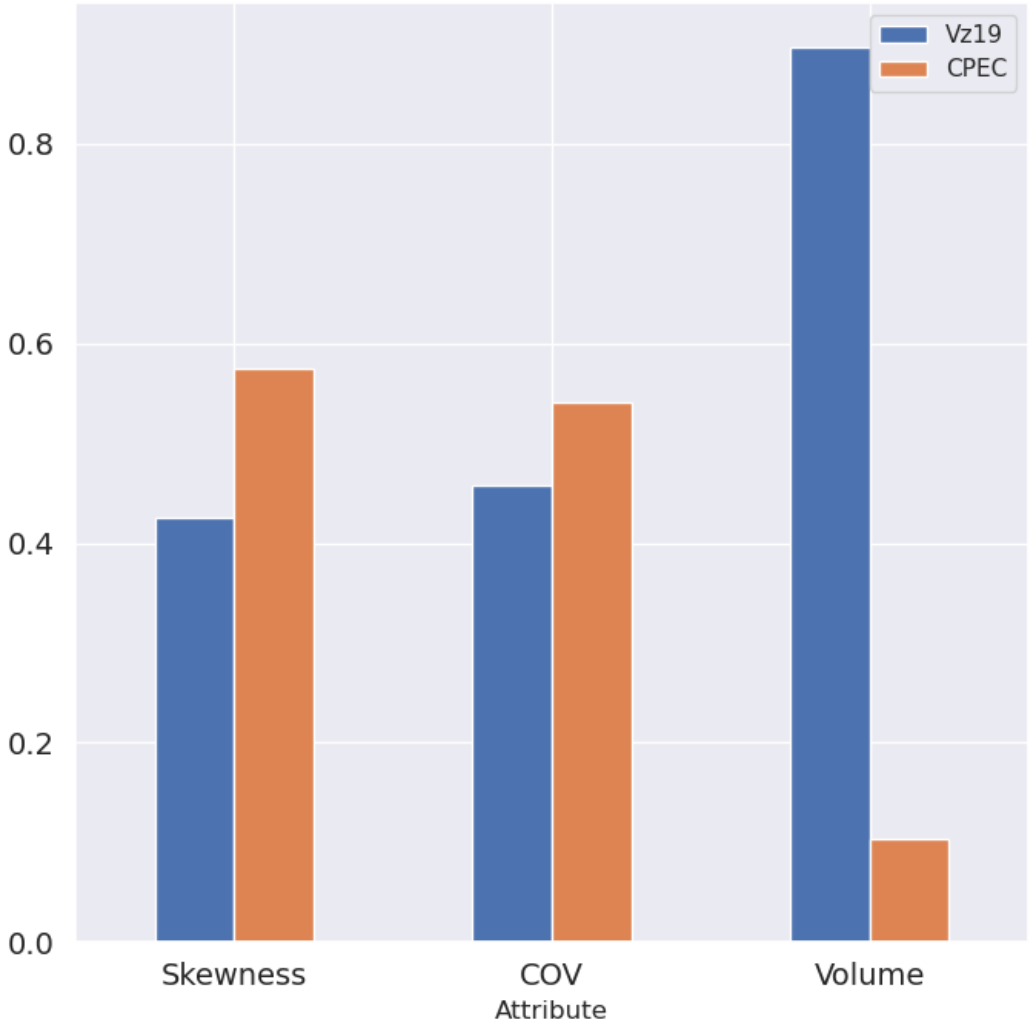



Figure 4.4: Vz19 and CPEC attribute comparisons.

### 4.8.2 Time Series Cluster Analysis

In order to further understand the reason for VAM's worse CPEC Volume-Prediction performance, we also performed clustering analysis on the results in a similar vein to Chapter 3. Once again, we analyzed the S-APE and NC-RMSE metrics. Also, once again we created the same 6 clusters from Chapter 3: the (1) High-Skewness Cluster, (2) Low-Skewness Cluster, (3) High-COV Cluster, (4) Low-COV Cluster, (5) High-Volume Cluster, and (6) Low-Volume Cluster.

In the test set for this CPEC data, there were 10 topics, 14 days, and 3 output types, so we calculated 420 values for each of the 3 time series attributes (Skewness, COV, and Volume). Once again, we placed time series into their respective clusters using the 80th percentile for each time series attribute. Each High Cluster contained 84 time series, and each Low Cluster contained 336 time series.

Figures 4.5 and 4.6 contain the S-APE and NC-RMSE cluster error results, respectively. We also included the cluster error results for the Vz19 dataset in Chapter 3 for comparison.

The blue and orange bars (the first two bars in each bar quartet, respectively), are for the Vz19 High and Low clusters, respectively. The green and red bars (the last two bars in each bar quartet, respectively) are for the CPEC High and Low clusters, respectively.

As apparent in Figure 4.5, VAM performed worse on S-APE in each CPEC cluster, compared to each Vz19 cluster. It performed especially worse on each of the High Clusters. This suggests that VAM struggled to predict time series that were relatively highly-skewed, highly-volatile, or high in volume.

In Figure 4.6, one can observe the cluster results on the NC-RMSE metric. The NC-RMSE for the Skewness and COV High Clusters in both the Vz-19 and CPEC domains were much higher than the NC-RMSE for the lower clusters in both domains.

Lastly, Volume has less of a dramatic effect on NC-RMSE performance compared to Skewness and COV. However, it should be noted that in CPEC, it would appear that the

CPEC-High-Volume Cluster has a somewhat higher NC-RMSE compared to the CPEC Low-Cluster.

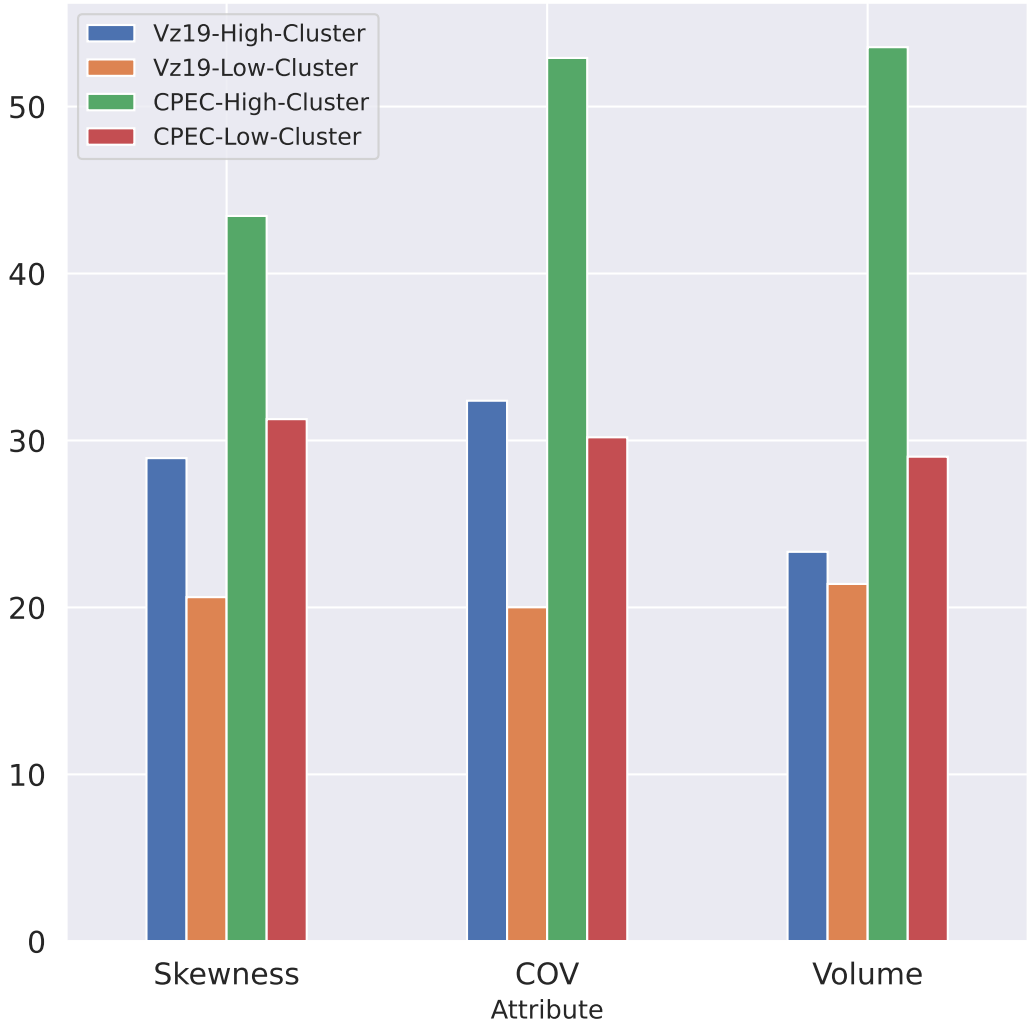



Figure 4.5: S-APE Vz19 and CPEC cluster comparison results.

The overall takeaway from these two figures is that across both Vz19 and CPEC, highly-skewed and highly-volatile time series are harder to predict. The effect of volume on predictability is different across domains. In Vz19, volume had little effect on predictability, but in CPEC, volume had a huge impact on predictability, particularly for the S-APE metric. Future work would involve a further analysis of the high volume time series in CPEC to see if there's a reason VAM struggled to predict them.

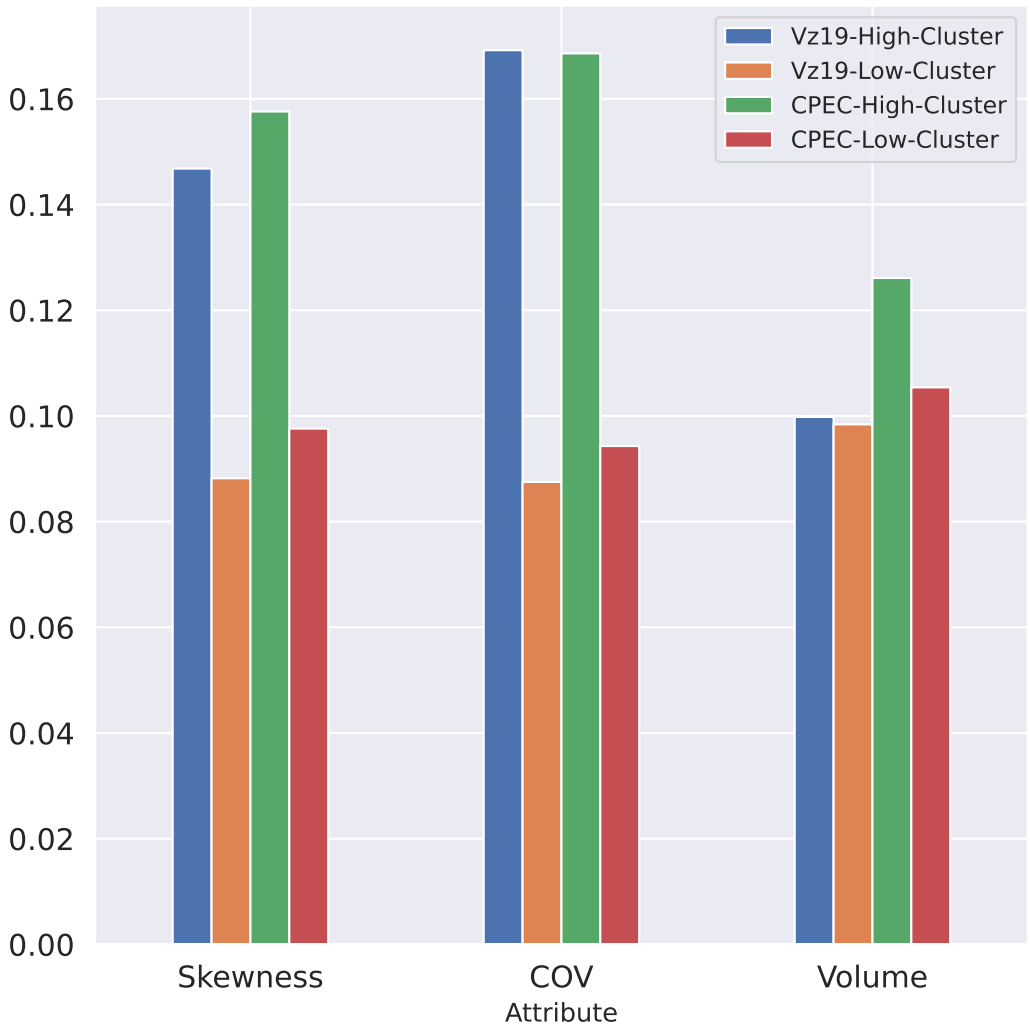



Figure 4.6: NC-RMSE Vz19 and CPEC cluster comparison results.

## 4.9 User Assignment Results

In this section we discuss the User Assignment results. We evaluate them in a similar fashion to Chapter 3. To measure old user prediction accuracy we use Weighted and Unweighted Jaccard Similarity. To measure network structure accuracy, we use Earth Mover's Distance and Relative Hausdorff Distance.

### 4.9.1 Jaccard Similarity Per-Topic Results

Similar to Chapter 3, the Jaccard Similarity metrics compare the ground truth and predicted sets of influential users. We define an influential user as a user who has been retweeted in a particular timestep $T$. The weighted version of the metric uses the number of times a user has been retweeted as the weight, whereas the unweighted version solely focuses on whether or not a user has been retweeted or not within a given timestep $T$.

For Weighted Jaccard Similarity (WJS), VAM outperformed the Persistence Baseline on 8 out of 10 topics. Some topics in which VAM did particularly well on were the *leadership/sharif*, *benefits/development/roads*, and *controversies/china/uighur* topics, with PIFBB scores of 214.18%, 219.75%, and 119.68%, respectively.

For Unweighted Jaccard Similarity (UJS), VAM also outperformed the Persistence Baseline on 8 out of 10 topics. VAM's PIFBB scores for UJS were not as high as its scores for WJS, but nonetheless VAM still performed relatively well on the *leadership/sharif*, *benefits/development/roads*, and *controversies/china/uighur* topics, with PIFBB scores of 80.9%, 45.71%, and 43.51%, respectively.

All in all, VAM's Jaccard Similarity results indicate that overall VAM does well against the Persistence Baseline in terms of predicting influential users. Since VAM did better at predicting Weighted Jaccard Similarity over Unweighted Jaccard Similarity, that indicicates that VAM does better at predicting how influential each user will be compared to just predicting whether or not a user will be influential.

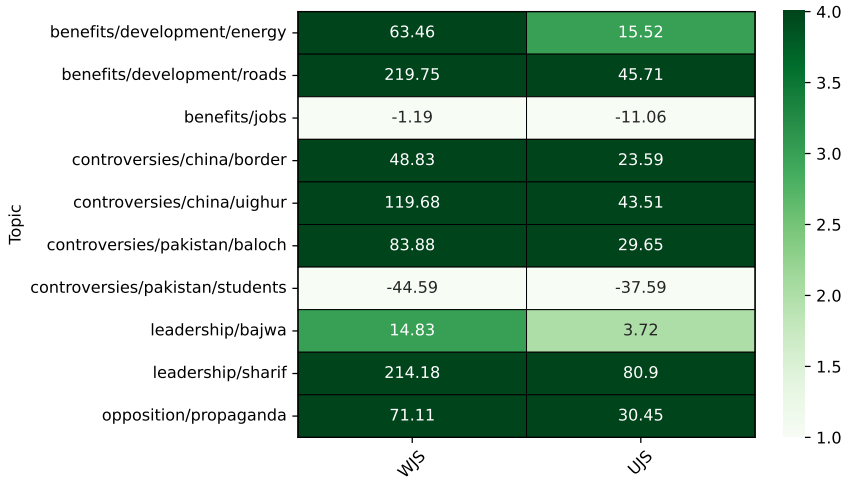



Figure 4.7: CPEC Jaccard Similarity results. The numbers represent VAM-TR-72's Percent Improvement From Best Baseline (PIFBB), which was the Persistence Baseline.

### 4.9.2 Network Structure Per-Topic Results

Recall from Chapter 3, we used Earth Mover's Distance (EMD) to compare the Page Rank Distributions of the ground truth and simulated networks. Relative Hausdorff Distance (RHD) is used to compare the unweighted indegree distributions of the ground truth and simulated networks.

For Earth Mover's Distance, VAM outperformed the Persistence Baseline on 8 out of 10 metrics. It performed particularly well on the *controversises/pakistan/baloch*, *benefits/development/roads*, and *opposition/propoganda* topics, with PIFBB scores of 27.29%, 20.89%, and 19.2%, respectively.

For Relative Hasdorff Distance, VAM also outperformed the Persistence Baseline on 8 out of 10 topics. It performed particularly well on *controversies/pakistan/baloch*, *leadership/sharif*, and *controversies/pakistan/students* topics, with PIFBB scores of 25.24%, 22.8%, and 18.94%, respectively.

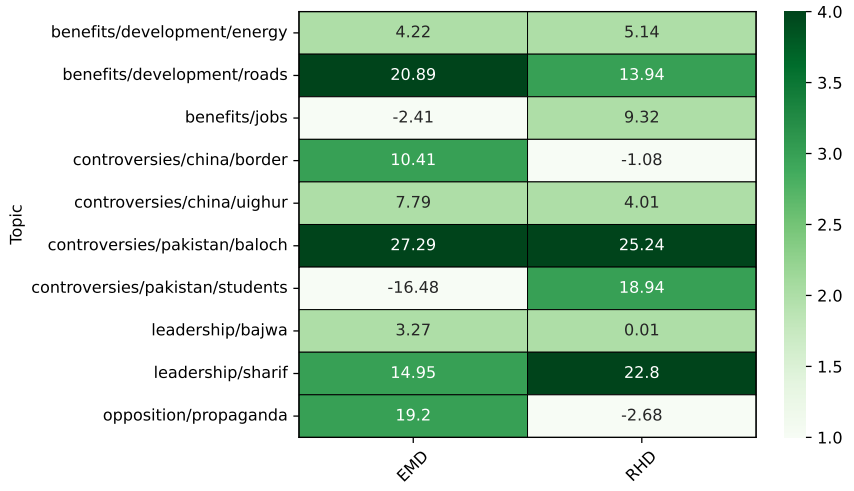



Figure 4.8: CPEC EMD and RHD results. The numbers represent VAM-TR-72's Percent Improvement From Best Baseline (PIFBB), which was the Persistence Baseline.

## 4.10 Conclusion

In this chapter, we discussed the *VAM* simulator, an end-to-end approach for time series prediction and temporal link prediction and applied it to the CPEC Twitter dataset. We showed that VAM outperformed the baseline models on the Volume Prediction and User-Assignment tasks.

On the Volume Prediction task, for the Overall Normalized Metric Error, VAM outperformed the best baseline model (ARMA) on 35 out of 60, or about 58% of all topic-metric pairs. We acknowledge that this is worse than VAM's roughly 88% win rate from the Venezuelan Twitter dataset of Chapter 3, however VAM's win-rate over ARMA still indicates clear superiority. We also showed that external Reddit and YouTube features aid VAM with the Volume Prediction task.

In order to gain a better understanding of the types of time series VAM performs well on, we clustered both the Vz19 and CPEC time series according to Skewness, Coefficient of Variation, and Volume. We found that, across both datasets, VAM performs better on time series with low Skewness and low Coefficient of Variation. Intuitively, this makes sense because less "erratic" time series should be easier to predict.

For the User-Assignment task, VAM outperformed the Persistence Baseline on 32 out of 40 topic-metric pairs.

In addition, we showed that VAM can predict the creation of new users, unlike many previous link prediction approaches that only focus on the prediction of old user-to-user interactions. Lastly, VAM's user-assignment is quite fast, taking only 27 minutes to simulate the activity of millions of user-to-user edges.

By showing VAM's strong performance on the CPEC dataset, we show that VAM can serve as a general social media simulator, and not one that is just specific to the Venezuelan Political dataset of Chapter 3. Future work involves utilizing a machine-learning model for the *User-Assignment* module, as well as trying Transformer neural networks for both the *Volume Prediction* and *User-Assignment* modules.

# Chapter 5: Data Augmentation with VAM

## 5.1 Introduction

In this chapter, the Synthetic Minority Oversampling Technique for Regression (a.k.a. SMOTER) was applied to the CPEC dataset used in Chapter 4. The two research questions (RQs) we seek to answer in this chapter are the following:

- RQ1: Does a SMOTER-based data augmentation algorithm aid with predicting topic time series in a social media network?
- RQ2: If so, are there any types of time series that a SMOTER-based data augmentation algorithm helps with predicting more than others?

These questions are of interest to us because while there have been many previous works on time series regression in social networks, none of these works have utilized data augmentation of any kind [46, 40, 39, 56, 30, 66, 36].

To answer the above questions, we created two variations of SMOTER and applied them to the CPEC VAM dataset from Chapter 4. One is called *SMOTER-BIN*. SMOTER-BIN uses binning to split up the training samples into different classes before applying SMOTER. We refer to the VAM model trained on SMOTER-BIN data as SMOTER-B-VAM.

The 2nd variation of SMOTER we created is valled *SMOTER-No-Binning* (SMOTER-NB). This variation does not use binning before applying SMOTER. It simply augments all samples without regard to the concept of a majority or minority class. The VAM model trained on the SMOTER-NB data is called SMOTER-NB-VAM.

We found that both SMOTER-B-VAM and SMOTER-NB-VAM were better than the regular VAM model in different ways. So, the answer to RQ1 is yes, data augmented with SMOTER does improve topic time series prediction performance.

SMOTER-NB-VAM (non-binning method) was better in terms of overall average performance. SMOTER-B-VAM (binning method) had more statistically significant topic-wins. It had 5 out of 10 statistically significant topic-wins, versus the 3 out of 10 statistically significant topic-wins of SMOTER-NB-VAM.

Lastly, it was found that one can use time series features to "pre-select" when to use SMOTER-VAM or Regular-VAM to perform a prediction for a particular input. Ensemble models of SMOTER-VAM and regular VAM models can be made using this methodology. The best ensemble made using this approach was the *Non-Binning SMOTER-VAM Ensemble for Low Volume* (NB-SVE-low-volume) model. This ensemble used SMOTER-NB-VAM for low-volume time series, and regular VAM for high-volume time series. It had 6 out of 10 statistically significant topic wins against the regular VAM model. Furthermore, it had a 18.97% percent improvement score from an ARMA baseline, which is higher than the 17.45% score that regular VAM model had. This suggests that SMOTER can be particularly useful for improving prediction accuracy of low-volume time series, which answers RQ2.

## 5.2 Overall Data Methodology

Before describing how SMOTER was used it is first useful to understand how each sample was set up. This work uses the same dataset of the VAM-TR-72 model in Chapter 4. The dataset for this model was chosen in particular because it was the best performing model from that chapter's set of CPEC VAM models.

In the training, validation, and test sets, each sample represents the state of a topic $q$ at time $T$, or a topic-timestep pair, $(q, T)$. The input for a given sample is comprised of a time series matrix, made up of 7 time series (each 72 hours each), and a vector of 10 1-hot static features to represent what the topic of interest is for the given sample. Altogether,

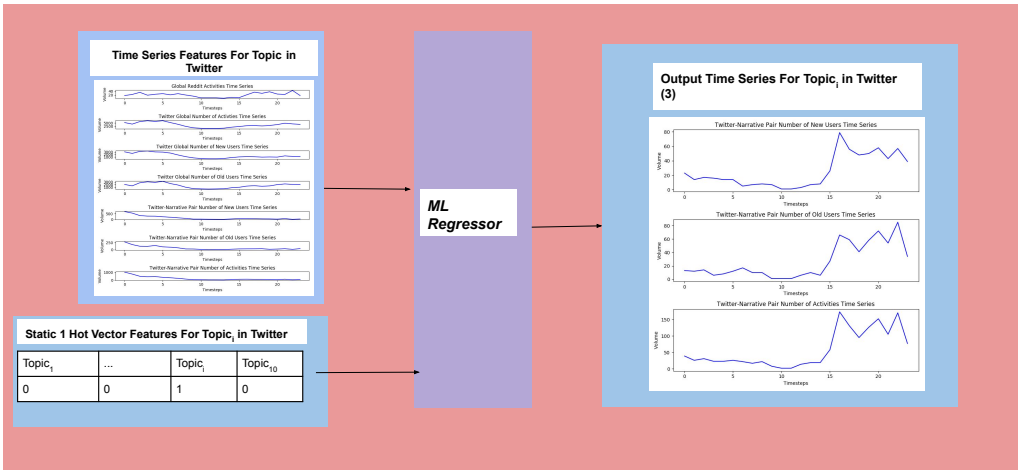



Figure 5.1: How each sample is setup in CPEC.

this creates an input vector with 514 features (72 * 7 + 10). Figure 5.1 illustrates this. It is the same sample methodology used in the previous VAM chapters 3 and 4.

Overall there were 31,210 samples in the original training set, 1450 validation samples, and 140 test samples. Similar to Chapter 4, the training period was from April 2, 2020 to August 10, 2020 (4 months). The validation period was August 11 to August 17th, 2020 (1 week). Lastly, the test period was August 18, 2020 to August 31, 2020 (2 weeks).

## 5.3 SMOTER-Non-Binning-VAM Methodology

The SMOTER-NB-VAM model does not use binning. SMOTER is simply directly applied all training samples. A parameter, $\alpha$ is used to determine by how much to augment the data. Let $N$ be the size of the original training set. The new size of the training set after augmentation is $\alpha * N$. In our experiments, there were $N = 31,210$ training samples and $\alpha$ was set to 3, so in total we had 93,630 samples in the augmented dataset for SMOTER-NB-VAM. We used $\alpha = 3$ because it yielded the best results over the validation set, using RMSE as the metric. Other values tried were 2 and 4. The K parameter for the nearest neighbors component of SMOTER was set to 3. Other values were tried such as 5, 7, 10, and 15, however 3 worked the best. The Algorithm 5.1 contains the SMOTER-NB pseudocode.

**Algorithm 5.1** SMOTER-NB

**Input:** Matrix $X$ which is the input matrix of training set samples of dimensions $N \times m$. Each row represents a sample and each column represents a feature; Matrix $Y$ which is the output matrix of dimensions $N \times p$, which is the output matrix. Each row is a sample and corresponds to the sample row in $X$. Each column is one of the output values; Integer $\alpha$ that indicates by how much to augment each sample; Integer $N$, which is the size of the training set; Integer K, the number of nearest neighbors.

**Output:** Matrix $X'$ which is the augmented input feature matrix; Matrix $Y'$ which is the augmented output matrix.

Initialize empty matrices $X'$ and $Y'$
**for** i=1 up to $N$ **do**
    $x = X[i]$
    $y = Y[i]$
    **for** j =1 up to $\alpha$ **do**
        Get $K$ nearest neighbor vectors of vector x. Save results in matrix $X^{nbr}$
        Randomly select a neighbor feature vector from $X^{nbr}$, called $x^{rand\_nbr}$.
        Also retrieve $y^{rand\_nbr}$, which is the output vector that corresponds to the input vector $x^{rand\_nbr}$.
        Create a random float between 0 and 1 called $\epsilon$.
        Create new augmented input feature vector: $x' = x + \epsilon * (x^{rand\_nbr} - x)$
        Create new augmented output vector: $y' = y + \epsilon * (y^{rand\_nbr} - y)$
        Append $x'$ to $X'$
        Append $y'$ to $Y'$
    **end for**
**end for**
**return** $X'$, $Y'$

## 5.4 SMOTER-Binning-VAM Methodology

### 5.4.1 Converting Time Series Matrices to Singular Values

In order to use SMOTER on binned data, the data must first be binned to begin with. However, the samples we are working with are comprised of multiple time series as outputs: three 24-hour time series for the number of new users, old users, and activities. In other words, the output is a matrix with 3 rows and 24 columns. So, before applying SMOTER for the SMOTER-B-VAM model, we needed a way to divide the samples into classes.

In order to transform the data to be suitable for SMOTER, we calculated the Frobenius Norm of each output matrix, converting each matrix into a singular value. It is calculated by taking the square root of the sum of the squares of its elements. Let $A$ represent a matrix, and let $a_{ij}$ represent any given value in that matrix. The Frobenius Norm formula is as follows:

$$\|A\|_F = \left( \sum_{i,j=1}^{n} |a_{ij}|^2 \right)^{1/2}$$

We further explain the Frobenius Norm with an example. For example let there be a matrix $A$ such that:

$$A = \begin{bmatrix} 5 & 3 \\ 6 & 4 \end{bmatrix}$$

The Frobenius Norm would then be:

$$\|A\|_F = \sqrt{5^2 + 3^2 + 6^2 + 4^2} \approx 9.27$$

### 5.4.2 Binning

These new Frobenius Norm outputs were then split into classes using binning. We used binning because it was similar to the approach used in the original SMOTER paper [61].

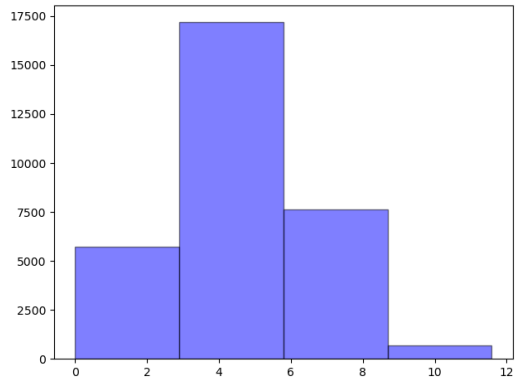



Figure 5.2: The 4 bin categories for the CPEC training dataset. There are 31,210 samples. The X axis shows the range of log-normalized, Frobenius Norm values that the samples in each bin map to. The Y-axis shows the number of samples contained within a given bin. The number of samples in bins 1, 2, 3, and 4 were as follows: 5,729; 17,183; 7,626; and 672, respectively.

As previously mentioned, the Frobenius Norm of each output matrix across all samples was calculated. By doing this, each sample in my dataset mapped to 1 value. The log norm of each norm was then calculated. We tried doing this experiment without log-normalization, however, by skipping this step, too many of the norms fell into 1 bin. If too many values fall into 1 bin, then it would be difficult to perform interpolation with SMOTER for the minority classes. This log-normalization step resulted in a range of values spanning from 0 to 11.58, which one can also see in Figure 5.2.

We then performed 4 "cuts" at equal intervals along this range, creating 4 bins. We tried other bin sizes besides 4, but when we evaluated the different SMOTER-VAM models on the validation data, the models trained on the 4-bin-dataset performed the best.

Bin 1 contained all values (log-normalized Frobenius norms) spanning 0 to 2.897. Bin 2 contained all values spanning 2.897 to 5.794. Bin 3 contained all values spanning 5.794 to 8.691. Lastly, bin 4 contained all values spanning 5.794 to 11.588. The number of samples in bins 1, 2, 3, and 4 were as follows: 5,729; 17,183; 7,626; and 672, respectively. There were 31,210 training samples in total. Figure 5.2 contains the histogram.

Different bin divisions were used, but during validation we found that 4 worked the best. Other values tried were 2, 3, 5, and 10.

### 5.4.3 Applying SMOTER to the Binned Data

SMOTER was then used to augment the training set.The K parameter for the nearest neighbors component of SMOTER was set to 15. Other values were tried such as 3, 5, 7, and 10, however 15 worked the best. Each of the 4 bins acted as the classes used for SMOTER. Since the 2nd bin had the most samples (17,183), that was the majority class. The other bins were considered as the minority class. So, the new dataset contained 17,183 samples per class, or 68,732 samples in total.

We note that the SMOTER-NB-VAM model uses 93,630 samples, while the SMOTER-B-VAM model used has 68,732 samples. This is a difference of about 25,000 samples. One may intuitively assume that such a disparity in sample amount may pose as an unfair advantage to the SMOTER-NB-VAM model. Keep in mind however, that the way the SMOTER-BIN algorithm works does not allow for the number of samples to surpass 68,732. There are 4 bins, or 4 classes. SMOTER-BIN creates samples for each minority class until they match the number of samples in the majority class. There were 3 minority classes, and 1 majority class. The majority class had 17,183 samples. So, after applying SMOTER-BIN to each minority class, one will get 17,183*4 samples in the new dataset, or 68,732.

Future work would involve making a variant of SMOTER-BIN that would allow for augmenting the number of samples to a higher amount. However, as mentioned earlier in this work, the objective of these experiments is to determine (1) does a SMOTER-based algorithm improve prediction performance for topic time series prediction and (2) if so, what types of time series does a SMOTER-based augmentation algorithm aid with the most. Whether or not SMOTER-BIN or SMOTER-NB is better is of less concern.

Algorithm 5.2 contains the pseudocode for the SMOTER-BIN algorithm.

**Algorithm 5.2** SMOTER-BIN

**Input:** Matrix $X$ which is the input matrix of training set samples of dimensions $N \times m$. Each row represents a sample and each column represents a feature; Matrix $Y$ which is the output matrix of dimensions $N \times p$, which is the output matrix. Each row is a sample and corresponds to the sample row in $X$. Each column is one of the output values; K, the number of nearest neighbors; B, the number of bins.

**Output:** Matrix $X'$ which is the augmented input feature matrix; Matrix $Y'$ which is the augmented output matrix.

1: Initialize empty matrix $Y^{Frob}$.
2: **for** $i = 1$ up to $N$ **do**
3: $y = Y[i]$
4: $y^{Frob}$= Calculate the natural log of the Frobenius norm of matrix $y$
5: Append $y^{Frob}$ to $Y^{Frob}$.
6: **end for**
7: Initialize empty matrix $X^{temp}$
8: **for** $i = 1$ up to $N$ **do**
9: $x = X[i]$
10: $y = Y[i]$
11: $x^{temp} = concatenate(x, y)$
12: Append $x^{temp}$ to $X^{temp}$
13: **end for**
14: Create a vector called $Y^{bin\_class}$ of size $N$, which contains the bin class of each corresponding value in $Y^{Frob}$. There will be $B$ unique classes.
15: Use SMOTE to augment the minority classes of $Y^{bin\_class}$ with their corresponding vectors in $X^{temp}$. Call this new matrix $X^{temp\_aug}$.
16: Split $X^{temp\_aug}$ into $X'$, which is the new augmented $X$ matrix, and $Y'$, which is the new augmented $Y$ matrix.
17: **return** $X'$, $Y'$

## 5.5 Results

### 5.5.1 Overall Results

Table 5.1 contains the overall results of the VAM, SMOTER-B-VAM, and SMOTER-NB-VAM models. The metrics used were RMSE, MAE, VE, SkE, S-APE, and NC-RMSE. For each model, each metric was calculated on each of the 420 time series in the test set, and then averaged. Recall that there are 420 time series in the test set because there are 10 topics, 14 days, and 3 output-types (new users, old users, and activities).

The 6 metrics were then used to create the combined Overall Normalized Metric Error (ONME), which is also shown in Table 5.1. Recall ONME was also used as a metric in Chapters 3 and 4. It was calculated by creating six "metric groups," each comprising 14 model metric results for that particular metric. A similar "normalized error metric" was used in [19]. The model results within each of the six groups were normalized between 0 and 1 by dividing each model metric result by the sum of all model metric results within that particular group. The models in each table are then sorted and ranked from lowest to highest ONME.

As one can see in the table, the best overall model was the SMOTER-NB-VAM model, with an ONME of 0.3279 and Percent Improvement From Baseline (PIFB) of 1.984%. The baseline compared to in this case was the regular VAM model, which had an ONME of 0.3345.

In terms of overall average performance, SMOTER-B-VAM model was the worst of the 3 models, with an ONME of 0.3376 and PIFB of -0.937%.

So, in terms of overall average performance, applying SMOTER without binning seemed to improve VAM's performance, while using SMOTER with binning seemed to make it slightly worse.

Table 5.1: VAM and SMOTER-VAM comparisons.

| **VAM and SMOTER-VAM Comparisons** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Model** | **RMSE** | **MAE** | **VE** | **SkE** | **S-APE** | **NC-RMSE** | **ONME** | **PIFB (%)** |
| **SMOTER-NB-VAM** | 64.15 | 45.86 | 34.69 | 0.9978 | 36.92 | 0.1261 | ***0.3279*** | ***1.984*** |
| VAM | 63.77 | 45.77 | 35.84 | 1.0726 | 37.97 | 0.1253 | 0.3345 | 0.0 |
| SMOTER-B-VAM | 65.34 | 47.35 | 35.91 | 0.9955 | 38.72 | 0.1319 | 0.3376 | -0.937 |

### 5.5.2 Per-Topic Results

Tables 5.2 and 5.3 contain the per-topic results of the SMOTER-NB-VAM and SMOTER-B-VAM models, respectively. The numbers shown in the model columns are the ONME results of each model per topic, which is why each pair of models for each topic add up to 1. The Wilcoxon Signed Rank Test with an alpha of 0.05 was used to test significance of each topic comparison. The Percent Improvement score indicates by how much the SMOTER-NB-VAM and SMOTER-B-VAM models improved from the regular VAM model. Asterisks (*) indicate statistically significant results.

As one can see in Table 5.2, SMOTER-NB-VAM outperformed the regular VAM model on 5 out of 10 topics. Out of these 5 wins, 3 were found to be significant.

The SMOTER-B-VAM model, in Table 5.2, outperformed VAM on 6 out of 10 topics. Furthermore, 5 out of the 6 wins were found to be significant.

It is interesting to note that although the SMOTER-NB-VAM model had the best overall performance in Table 5.1, the SMOTER-B-VAM model was best in terms of number of statistically significant topic wins in Table 5.3.

### 5.5.3 Time Series Plot Analysis

Figures 5.3 and 5.4 contain instances in which the SMOTER-B-VAM and SMOTER-NB-VAM models outperformed the regular VAM model, respectively. As one can see in the plots, the two SMOTER-VAM models were able to predict spikes that the regular VAM

Table 5.2: SMOTER-NB-VAM vs. VAM per-topic results. Asterisks indicate statistical significance using the Signed Wilcoxon Rank Test with an alpha of 0.05.

| **SMOTER-NB-VAM vs. VAM Per-Topic Results** | | | | |
|---|---|---|---|---|
| **Topic** | **VAM** | **SMOTER-NB-VAM** | **Winner** | **Percent Imp.** |
| controversies/china/uighur | 0.5175 | **0.4825** | S-NB-VAM | 6.7579* |
| leadership/bajwa | 0.514 | **0.486** | S-NB-VAM | 5.4566 |
| benefits/jobs | 0.5091 | **0.4909** | S-NB-VAM | 3.5923* |
| benefits/development/energy | 0.5088 | **0.4912** | S-NB-VAM | 3.4471 |
| controversies/pakistan/students | 0.5006 | **0.4994** | S-NB-VAM | 0.2431* |
| benefits/development/roads | **0.4999** | 0.5001 | VAM | -0.0573* |
| controversies/pakistan/baloch | **0.4988** | 0.5012 | VAM | -0.4766* |
| leadership/sharif | **0.4972** | 0.5028 | VAM | -1.1352 |
| controversies/china/border | **0.4953** | 0.5047 | VAM | -1.9172* |
| opposition/propaganda | **0.4928** | 0.5072 | VAM | -2.9346 |

Table 5.3: SMOTER-B-VAM vs. VAM per-topic results. Asterisks indicate statistical significance using the Signed Wilcoxon Rank Test with an alpha of 0.05.

| **SMOTER-B-VAM vs. VAM Per-Topic Results** | | | | |
|---|---|---|---|---|
| **Topic** | **VAM** | **SMOTER-B-VAM** | **Winner** | **Percent Imp.** |
| controversies/pakistan/baloch | 0.5095 | **0.4905** | S-B-VAM | 3.7172* |
| benefits/development/energy | 0.5086 | **0.4914** | S-B-VAM | 3.3854* |
| controversies/pakistan/students | 0.5043 | **0.4957** | S-B-VAM | 1.7183* |
| controversies/china/border | 0.5033 | **0.4967** | S-B-VAM | 1.3129* |
| controversies/china/uighur | 0.5032 | **0.4968** | S-B-VAM | 1.2616* |
| benefits/jobs | 0.5013 | **0.4987** | S-B-VAM | 0.5148 |
| benefits/development/roads | **0.498** | 0.502 | VAM | -0.8171* |
| leadership/bajwa | **0.4949** | 0.5051 | VAM | -2.0539* |
| leadership/sharif | **0.4927** | 0.5073 | VAM | -2.947* |
| opposition/propaganda | **0.4797** | 0.5203 | VAM | -8.4716* |

model missed in certain instances, albeit not always in the exact location. This is still acceptable however, because if one can at least know that one or more bursts of activity will take place within a 24-hour period, that is still useful information, even if the exact hour is not known.

Figures 5.4a and 5.4c contain instances in which the SMOTER-NB-VAM model predicted that little to no bursty activity would occur in contrast to the regular VAM model, which incorrectly predicted bursty activity. This is useful information as well because one would not want to falsely believe bursty activity is due to happen when it is not.

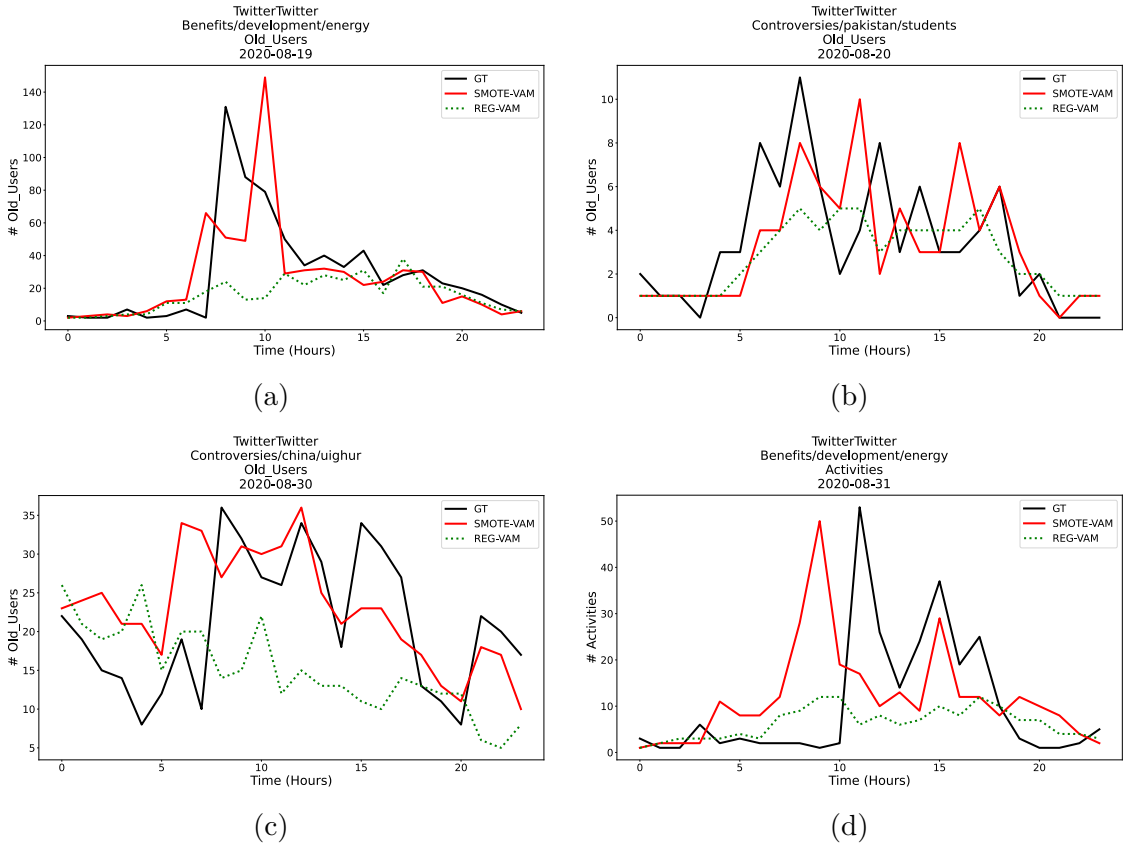



Figure 5.3: SMOTER-B-VAM outperforms the regular VAM model.

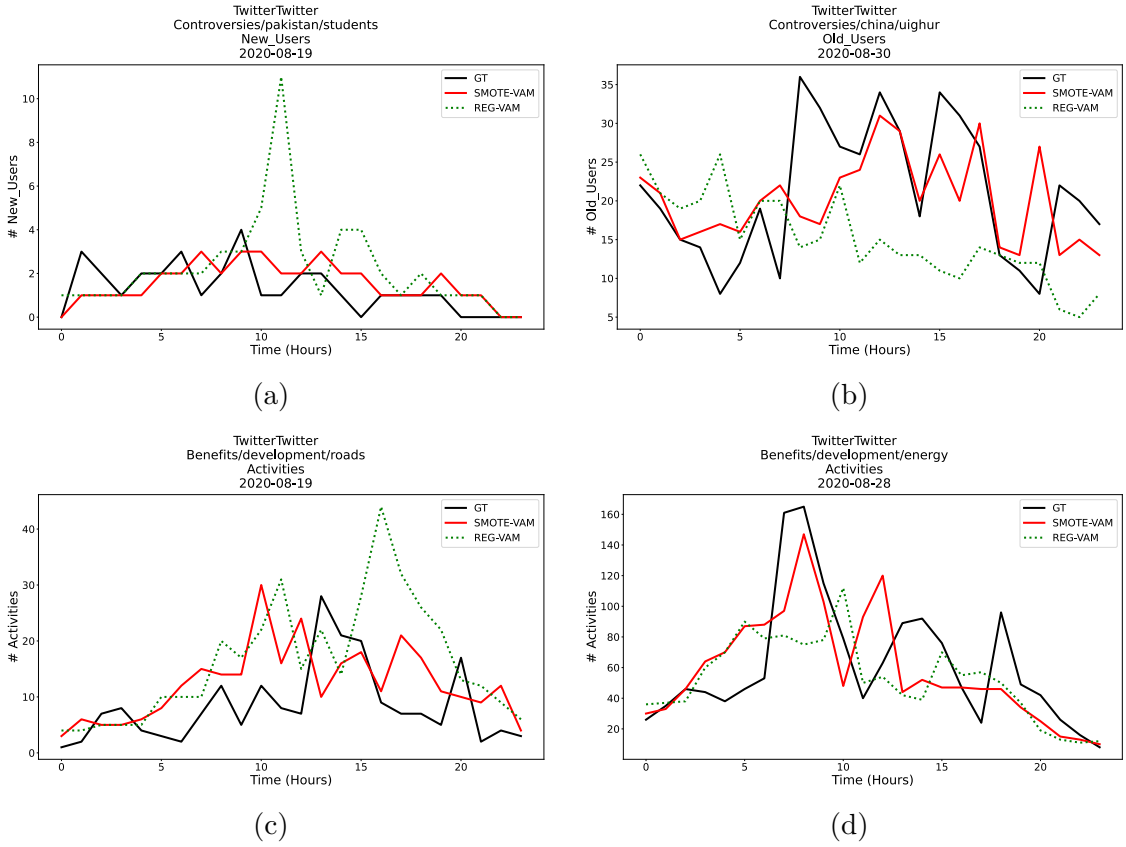



Figure 5.4: SMOTER-NB-VAM outperforms the regular VAM model.

## 5.6 Additional Ensemble Experiment Methodology

### 5.6.1 SMOTER-VAM Ensemble - Time Series Attributes

As one can see from the previous sections, we showed that the answer to RQ1 is that yes, SMOTER can improve performance of topic time series prediction models. We now turn our focus to RQ2 which was "Does a SMOTER-based data augmentation algorithm work better on time series with certain attributes?" If so, that means a heuristic could potentially be created so that one can determine when to use SMOTER-VAM vs. regular VAM for a particular time series.

The time series of interest we would be looking at would of course be the input time series in each of the 140 test samples. Specifically, they would be the input time series related to the 3 output-types of interest: (1) new users, (2) old users, and (3) activities. What we want

to know is if there are any attributes of these input time series that can indicate whether to use a SMOTER-VAM model or regular VAM model for predicting the output time series. Note that since there are 140 test samples, and since each test sample is related to 3 output-types, there are 420 input time series of interest, and 420 output time series of interest. Each input time series maps to an output time series, of course.

### 5.6.2 Cluster Ensemble Models

In order to answer the aforementioned question, 8 ensemble SMOTER-VAM models were created. They were made using 4 time series attributes: (1) volume, (2) coefficient of variation, (3) skewness, and (4) sparsity.

Volume refers to the total counts of a time series, such as total number of users or activities. Coefficient of variation is a ratio that is calculated by dividing the standard deviation by the mean. Skewness is a measure of the asymmetry of a time series. Sparsity is a measure of the number of 0's in a time series.

Furthermore, these 4 attributes were used to created 2 types of clusters per attribute: (1) high-value clusters and (2) low-value clusters. In these experiments, we defined "high" is defined as any value that is above the 80th percentile value and "low" is defined as any value that is equal to or less than the 80th percentile value. There were 420 input time series of interest in the test set, so 84 (420 * 0.2) of these input time series went into the high cluster and 336 (420 * 0.8) went into the low cluster.

Each of the 8 ensembles utilized both the SMOTER-VAM model and the regular VAM model, albeit with different heuristics. Table 5.4 shows how each ensemble works. Each ensemble is made up of an augmented model and regular VAM. For example, the *SVE-low-volume* ensemble is the model that uses SMOTER-VAM for input time series that have low volume, and regular VAM for input time series that have high volume. The assumption behind this model is that the SMOTER-VAM model will be better for input time series that

Table 5.4: SMOTE-VAM ensemble cluster information.

| **SMOTE-VAM Ensemble Cluster Information** | | |
|---|---|---|
| **Model** | **Input Time Series That SMOTE-VAM Predicts** | **Input Time Series That regular VAM Predicts** |
| SVE-low-volume | Low Volume | High Volume |
| SVE-high-coefficient of_variation | High Coefficient of Variation | Low Coefficient of Variation |
| SVE-high-skewness | High Skewness | Low Skewness |
| SVE-high-sparsity | High Sparsity | Low Sparsity |
| SVE-low-sparsity | Low Sparsity | High Sparsity |
| SVE-low-skewness | Low Skewness | High Skewness |
| SVE-low-coefficient of_variation | Low Coefficient of Variation | High Coefficient of Variation |
| SVE-high-volume | High Volume | Low Volume |

have a low volume of activities or users, while the regular VAM model will be better for input time series with a high volume of users or activities.

Another ensemble example is the "SVE-high-skewness" ensemble. This would the ensemble that uses SMOTER-VAM on input time series with a high skewness, and regular VAM on input time series with a low skewness.

Table 5.4 shows how each ensemble was set up.

## 5.7 Ensemble Model Experiment Results

### 5.7.1 Overall Ensemble Comparisons

Table 5.5 contains the comparisons of the SMOTER-VAM, VAM, ARMA, and SMOTER-VAM ensemble models across the RMSE, MAE, VE, SkE, S-APE, and NC-RMSE metrics. Table 5.6 contains the ONME and PIFB scores using the metric results from Table 5.5.

As one can see in these tables, some models contain the prefixes "B-SVE" and "NB-SVE", which stand for "Binning-SMOTER-VAM-Ensemble" and "Non-Binning-SMOTER-VAM-Ensemble", respectively.

Similar to Table 5.1, Overall Normalized Metric Error (ONME) is used as an overall error score. Also, since the regular VAM model is the same one used as in Chapter 4, the ARMA baseline from Chapter 4 is also used in this table as well for comparison. This baseline is included to show how the models also perform against a typical time series baseline. Note that augmenting time series data is pointless if the new model trained on this data cannot outperform a basic time series baseline. ARMA in particular was chosen because it was the best performing time series baseline in Chapter 4 out of ARMA, ARIMA, AR, MA, and the Persistence Baseline. Note that Percent Improvement From Baseline (PIFB) is calculated against ARMA.

The best overall model in terms of PIFB was the SMOTER-NB-VAM model with a score of 19.07% (higher is better). Recall this is the SMOTER-VAM model that did not use binning before applying SMOTER. In 2nd and 3rd place are the NB-SVE-low-volume and NB-SVE-low-skewness models, respectively. They had PIFB scores of 18.97% and 18.63%, respectively. The NB-SVE-low-volume was the ensemble that used SMOTER-NB-VAM for low-volume input time series with low-volume, and regular VAM for high-volume input time series. The NB-SVE-low-skewness ensemble used SMOTER-NB-VAM for low-skewness input time series, and regular VAM for high-skewness input time series.

It is notable that 8 out of the top 10 models are SMOTER-NB-VAM models, indicating that non-binning SMOTER-NB-VAM models perform consistently better than the SMOTER-B-VAM models. The best binning model does not occur until 4th place, which is the B-SVE-low-skewness ensemble. What is also notable is that in terms of overall PIFB, the ensemble NB-SVE models did not outperform the "non-ensembled" SMOTER-NB-VAM models. However, the ensemble B-SVE models did outperform the "non-ensembled" SMOTER-B-VAM model. The B-SVE-low-skewness model had a PIFB of 18.54%, while SMOTER-B-VAM had a PIFB of 16.64%.

Lastly, the most notable takeaway is that the best non-binning SMOTER-VAM model (SMOTER-NB-VAM) and best binning SMOTER-VAM model (B-SVE-low-volume) both outperformed the regular VAM model, which had a PIFB of 17.45%.

Tables 5.7 and 5.8 contain the results of significance testing on the results of Table 5.5. The metric results from the Regular-VAM model were compared against the metric results of each of the SMOTER-VAM and SVE ensemble models using the Wilcoxon Signed Rank Test. Recall that there are 14 test days, 10 topics, and 3 output-types (new users, old users, and activities). So, 420 ONME scores were calculated for each SMOTER-VAM/SVE model and 420 ONME scores were calculated for the regular VAM model. The errors were used for the significance test. The p-values are shown in the table, as well as a column indicating whether or not the result is significant with an alpha of 0.05. As one can see, all the B-SVE ensemble results were significant (Table 5.7). Six out of the 8 NB models were significant in Table 5.8. The two that were not significant were the NB-SVE-high-skewness and NB-SVE-high-volume models.

### 5.7.2 Ensemble Per-Topic Analysis

Tables 5.9 and 5.10 are per-topic analysis tables of the B-SVE-low-volume and NB-SVE-low-volume ensemble models, respectively. These models were chosen because they were each the best ensemble models out of the B-SVE and NB-SVE ensemble models, respectively. The B-SVE-low-volume model had 5 out of 10 statistically significant topic wins, and the NB-SVE-low-volume model had 6 out of 10 statistically significant topic wins.

### 5.7.3 Topic-Metric Comparison vs. ARMA

A topic-metric comparison was also done for the B-SVE-low-volume and NB-SVE-low-volume models in a similar manner to what was done in Chapter 4. The two models' metrics per each topic were calculated and compared against ARMA's results, in a similar manner to the heatmap Figure 4.2 from Chapter 4.

Table 5.5: SMOTER-VAM and VAM comparisons across 6 main metrics. The ARMA baseline model is also included for comparison.

| **All VAM and SMOTER-VAM Models vs. ARMA** | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **RMSE** | **MAE** | **VE** | **SkE** | **S-APE** | **NC-RMSE** |
| **SMOTER-NB-VAM** | 64.1524 | 45.8621 | 34.6903 | 0.9978 | 36.9157 | 0.1261 |
| NB-SVE-low-volume | 63.9089 | 45.886 | 35.5927 | 0.9871 | 37.1502 | 0.1247 |
| NB-SVE-low-skewness | 64.0643 | 45.8184 | 34.6964 | 1.0197 | 37.0173 | 0.1274 |
| B-SVE-low-volume | 63.6242 | 45.6153 | 35.2648 | 0.9949 | 37.8055 | 0.128 |
| NB-SVE-high-sparsity | 63.8489 | 45.8568 | 35.8172 | 1.0053 | 37.5534 | 0.1246 |
| NB-SVE-high-coefficient_of_variation | 63.8614 | 45.7475 | 35.0629 | 1.0419 | 37.7268 | 0.1242 |
| NB-SVE-low-coefficient_of_variation | 64.0603 | 45.8846 | 35.4728 | 1.0285 | 37.1615 | 0.1272 |
| NB-SVE-low-sparsity | 64.0727 | 45.7753 | 34.7185 | 1.0651 | 37.3349 | 0.1268 |
| NB-SVE-high_skewness | 63.8574 | 45.8138 | 35.8393 | 1.0507 | 37.871 | 0.124 |
| B-SVE-high-coefficient_of_variation | 63.8156 | 45.7187 | 35.5568 | 1.0387 | 38.1664 | 0.1259 |
| B-SVE-high-skewness | 63.8454 | 45.821 | 35.6585 | 1.0393 | 38.151 | 0.1256 |
| B-SVE-high-sparsity | 63.7906 | 45.7851 | 35.7685 | 1.0396 | 38.3977 | 0.1257 |
| NB-SVE-high-volume | 64.0128 | 45.7461 | 34.9429 | 1.0833 | 37.7381 | 0.1267 |
| VAM | 63.7693 | 45.77 | 35.8454 | 1.0726 | 37.9726 | 0.1253 |
| SMOTER-B-VAM | 65.3416 | 47.3483 | 35.9175 | 0.9955 | 38.7184 | 0.1319 |
| B-SVE-low-sparsity | 65.3202 | 47.3332 | 35.9943 | 1.0285 | 38.2933 | 0.1315 |
| B-SVE-low_skewness | 65.2654 | 47.2973 | 36.1043 | 1.0288 | 38.5401 | 0.1316 |
| B-SVE-low-coefficient_of_variation | 65.2952 | 47.3996 | 36.2061 | 1.0294 | 38.5246 | 0.1313 |
| B-SVE-high-volume | 65.4866 | 47.5031 | 36.498 | 1.0732 | 38.8855 | 0.1292 |
| ARMA | 72.5972 | 55.1047 | 39.4702 | 1.6593 | 42.9807 | 0.143 |

Table 5.6: SMOTER-VAM and VAM comparisons for ONME and PIFB. Percent Improvement From Baseline (PIFB) is calculated against ARMA.

| **All VAM and SMOTER-VAM Models vs. ARMA ONME and PIFB Scores** | | |
|---|---|---|
| **Model** | **ONME** | **PIFB (%)** |
| **SMOTER-NB-VAM** | **0.0486** | **19.0706** |
| NB-SVE-low-volume | 0.0487 | 18.9715 |
| NB-SVE-low-skewness | 0.0489 | 18.6368 |
| B-SVE-low-volume | 0.0489 | 18.5428 |
| NB-SVE-high-sparsity | 0.0489 | 18.533 |
| NB-SVE-high-coefficient_of_variation | 0.0491 | 18.3585 |
| NB-SVE-low-coefficient_of_variation | 0.0492 | 18.1714 |
| NB-SVE-low-sparsity | 0.0493 | 17.997 |
| NB-SVE-high_skewness | 0.0493 | 17.8931 |
| B-SVE-high-coefficient_of_variation | 0.0493 | 17.8831 |
| B-SVE-high-skewness | 0.0494 | 17.8371 |
| B-SVE-high-sparsity | 0.0494 | 17.7124 |
| NB-SVE-high-volume | 0.0495 | 17.5585 |
| VAM | 0.0496 | 17.4594 |
| SMOTER-B-VAM | 0.0501 | 16.6441 |
| B-SVE-low-sparsity | 0.0502 | 16.3911 |
| B-SVE-low_skewness | 0.0503 | 16.2664 |
| B-SVE-low-coefficient_of_variation | 0.0503 | 16.2204 |
| B-SVE-high-volume | 0.0507 | 15.5607 |
| ARMA | 0.0601 | 0.0 |

Table 5.7: SMOTER-B-VAM Signed Wilcoxon Rank Test p-values. Each model was compared to the VAM-TR-72 model (Regular VAM) from Chapter 4.

| **SMOTER-B-VAM Signed Wilcoxon Rank Test P-Values** | | |
|---|---|---|
| **Model** | **p_value** | **Is Significant** |
| B-SVE-low-volume | 3.95e-09 | 1 |
| B-SVE-high-coefficient_of_variation | 1.43e-09 | 1 |
| B-SVE-high-skewness | 1.12e-09 | 1 |
| B-SVE-high-sparsity | 2.21e-10 | 1 |
| B-SVE-low-sparsity | 3.89e-23 | 1 |
| B-SVE-low-skewness | 5.79e-22 | 1 |
| B-SVE-low-coefficient_of_variation | 1.61e-22 | 1 |
| B-SVE-high-volume | 9.51e-29 | 1 |

Table 5.8: SMOTER-NB-VAM Signed Wilcoxon Rank Test p-values.

| **SMOTER-NB-VAM Signed Wilcoxon Rank Test P-Values** | | |
|---|---|---|
| **Model** | **p_value** | **Is Significant** |
| NB-SVE-low-volume | 3.447571e-06 | 1 |
| NB-SVE-low-skewness | 2.500920e-05 | 1 |
| NB-SVE-high-sparsity | 0.023270 | 1 |
| NB-SVE-high-coefficient_of_variation | 0.015863 | 1 |
| NB-SVE-low-coefficient_of_variation | 0.000190 | 1 |
| NB-SVE-low-sparsity | 0.000269 | 1 |
| NB-SVE-high-skewness | 0.150680 | 0 |
| NB-SVE-high-volume | 0.647207 | 0 |

Table 5.9: B-SVE-low-volume vs. VAM per-topic results. Astericks indicate statistically significant using the Signed Wilcoxon Test, with an alpha of 0.05.

| **B-SVE-low-volume vs. VAM Per-Topic Results** | | | |
|---|---|---|---|
| **Topic** | **VAM** | **B-SVE low-volume** | **Percent Improvement (%)** |
| benefits/development/energy | 0.5117 | **0.4883** | 4.5885* |
| controversies/pakistan/baloch | 0.5111 | **0.4889** | 4.3493 |
| opposition/propaganda | 0.5057 | **0.4943** | 2.2493 |
| controversies/china/border | 0.5034 | **0.4966** | 1.3381* |
| controversies/pakistan/students | 0.5023 | **0.4977** | 0.9089* |
| benefits/jobs | 0.5019 | **0.4981** | 0.7415* |
| controversies/china/uighur | 0.5016 | **0.4984** | 0.6473* |
| leadership/sharif | 0.5015 | **0.4985** | 0.6135 |
| benefits/development/roads | **0.497** | 0.503 | -1.1941* |
| leadership/bajwa | **0.497** | 0.503 | -1.1947* |

Table 5.10: NB-SVE-low-volume vs. VAM per-topic results. Astericks indicate statistical significance using the Signed Wilcoxon Test, with an alpha of 0.05.

| **NB-SVE-low-volume vs. VAM Per-Topic Results** | | | |
|---|---|---|---|
| **Topic** | **VAM** | **NB-SVE low-volume** | **Percent Improvement (%)** |
| controversies/china/uighur | 0.5121 | **0.4879** | 4.7396* |
| benefits/jobs | 0.5097 | **0.4903** | 3.8173* |
| benefits/development/energy | 0.5092 | **0.4908** | 3.5965 |
| controversies/pakistan/students | 0.5067 | **0.4933** | 2.6275* |
| leadership/sharif | 0.5063 | **0.4937** | 2.5046 |
| opposition/propaganda | 0.5028 | **0.4972** | 1.1234* |
| controversies/pakistan/baloch | 0.5028 | **0.4972** | 1.0969* |
| leadership/bajwa | 0.5024 | **0.4976** | 0.9513* |
| benefits/development/roads | **0.4955** | 0.5045 | -1.8294 |
| controversies/china/border | **0.4934** | 0.5066 | -2.6645* |

Figures 5.5 and 5.6 are the heatmaps for B-SVE-low-volume and NB-SVE-low-volume, respectively. Figure 5.7 is a heatmap of the regular VAM results from Chapter 4. It was included here again so one could more easily compare the 3 models' results.

For each heatmap, each cell represents the VAM model's (B-SVE-low-volume's, NB-SVE-low-volume's, or regular VAM's) PIFB score against ARMA. Similar to the previous chapter, white cells represent instances in which the model of interest performed worse than ARMA. Light green cells represent instances in which the PIFB score was between 0 and 10%. Darker green cells represent instances in which the model's PIFB was between 10-20%. Lastly, the darkest green cells represent instances in which the model's PIFB score was over 20%.

As one can see, the B-SVE-low-volume and NB-SVE-low-volume models outperformed the regular VAM model in terms of topic-metric pair wins. In Figure 5.5, one can see that the B-SVE-low-volume model outperformed ARMA on 51 out of 60 topic-metric pairs, or about 85% of the time. In Figure 5.6, one can see that the NB-SVE-low-volume model also outperformed ARMA on 51 out of 60 topic-metric pairs, or about 85% of the time. Lastly, one can see that in 5.7, the regular VAM only outperformed ARMA on 35 out of 60 topic-metric pairs, or about 58% of the time. This suggests that augmenting the training data

with the two SMOTER-based algorithms (SMOTER-NB and SMOTER-BIN) helps improve VAM's predictive accuracy even at the topic-metric pair granularity.

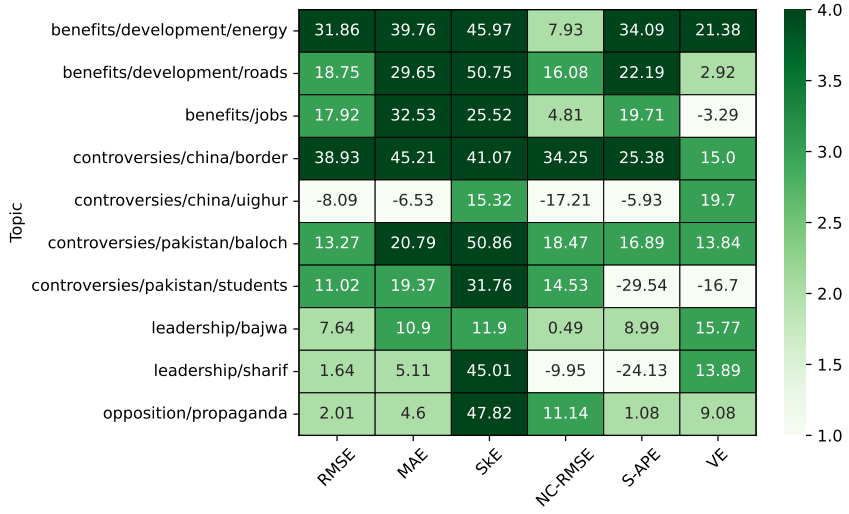



Figure 5.5: B-SVE-low-volume topic-metric heatmap. This model outperformed ARMA on 51 out of 60 topic-metric pairs, or about 85% of the time.

Also note that both the B-SVE-low-volume and NB-SVE-low-volume models have the same win-rate in terms of topic-metric pairs (85% each). However, despite this fact, the NB-SVE-low-volume model may be the more preferable one because at the topic granularity, it has 6 out of 10 statistically significant wins, versus the B-SVE-low-volume model which was 5 out of 10.

## 5.8 Conclusion

In this chapter, we showed how SMOTER-augmentation was applied to the VAM CPEC dataset from Chapter 4. Using this method, two models were created, SMOTER-B-VAM and SMOTER-NB-VAM, which used binning and no-binning, respectively.

We sought to answer two research questions with these models:

- RQ1: Does a SMOTER-based data augmentation algorithm aid with predicting topic time series in a social media network?

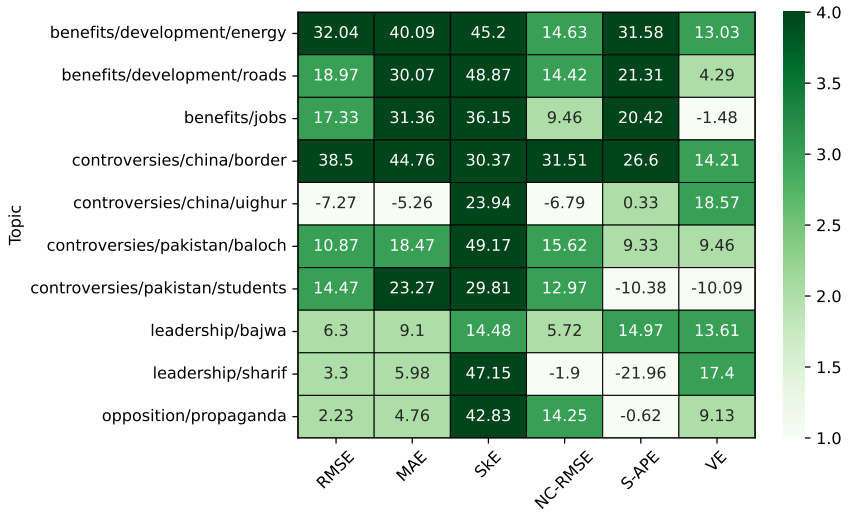



Figure 5.6: NB-SVE-low-volume topic-metric heatmap. This model outperformed ARMA on 51 out of 60 topic-metric pairs, or about 85% of the time.

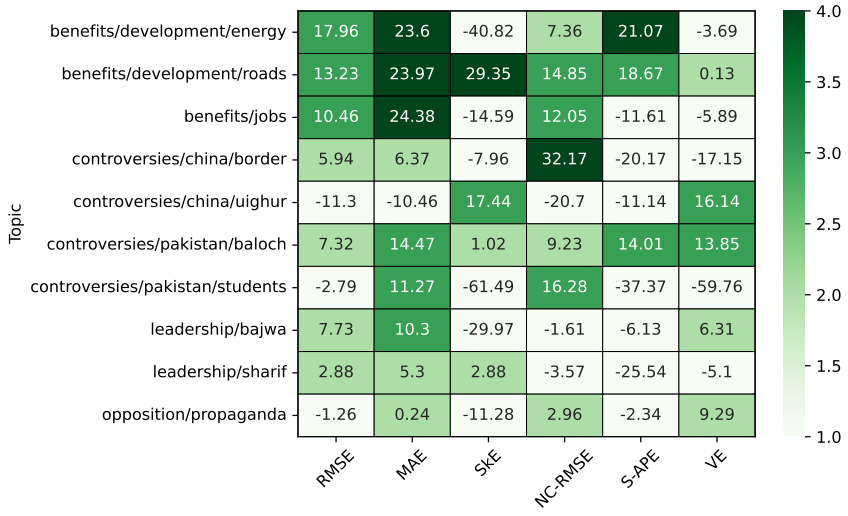



Figure 5.7: Regular VAM topic-metric heatmap. This is the same heatmap from Chapter 4. This model only outperformed ARMA on 35 out of 60 topic-metric pairs, or about 58% of the time. The SMOTER-VAM models performed much better against ARMA than this one did.

- RQ2: If so, are there any types of time series that a SMOTER-based data augmentation algorithm helps with more than others?

It was found that both of these models outperformed the regular VAM model in different ways, showing that the answer to RQ1 was "yes". Furthermore, it was found that time series attributes could be used to cluster the dataset, and create ensemble models consisting of a pair of SMOTER-VAM and regular VAM models. Using this methodology, it was shown that the SMOTER-VAM models perform strongly on input time series that have low-volume. This insight answers RQ2 with a "yes" as well.

Future work would involve using a modified version of SMOTER-BIN that generates more samples so that the training data size is equivalent to that used by SMOTER-NB. By doing this, we can determine if that allows it to achieve better accuracy. Another avenue we could explore is whether there are any other time series attributes that could be used for created SMOTER-VAM and regular VAM ensembles.

# Chapter 6: Time Series Forecasting Baseline Analysis[1]

Throughout this work, we have used ARIMA and Persistence Baseline models to compare against the VAM models. In this chapter, we seek to better understand the performance differences between ARIMA and the Persistence Baseline under different prediction window durations.

## 6.1 Data Processing

We used 3 social media domains in our analysis - The Vz19 domain from Chapter 3, the CPEC domain in Chapter 4, and a new domain, called the Belt and Road Initiative in East Africa domain (BRIA).

For each of the 3 social media domains, we focus on the same 2 platforms for each - Twitter and YouTube. As a result, there are 6 different datasets used in these experiments. The time periods for each domain were as follows:

- Vz19: December 24, 2018 to February 28, 2019
- CPEC: March 30, 2020 to June 4, 2020
- BRIA: February 1, 2020 to April 7, 2020

Each dataset contained 5 different topics. These topics were determined by a combination of annotators and BERT models. Annotators labelled the topic of each tweet or YouTube post using their best judgment. BERT models were then used to label unlabeled posts for a

[1]The work in this section came from [47], a journal paper that is still under review. The Long-Term models were created by one of the authors, Kin NG, while the Short-Term models were created by Fred Mubang (the author of this dissertation). The heatmaps shown in this section were created by Fred Mubang. The analysis and comparisons done between the two models was done by Fred Mubang as well in that previous work [47] with some editing done by the co-authors on that work.

faster and more efficient labeling process. Table 6.2 contains the topic names, post counts, Inter-Annotator Agreements, and topic descriptions. As one can see in the table, there are 5 topics per domain. In the original datasets labelled by the annotators, there were more topics, however, for each domain, only the top 5 topics in terms of Inter-Annotator agreement were kept. This was done in order to ensure less "noisy" datasets. As a result, there are 15 topics evaluated in the following experiments.

Table 6.1: Topic Inter-Annotator Agreement scores (IAA). There are also corresponding descriptions in each context. We include the aggregated volume of activities per topic in both Twitter and YouTube over the periods we consider.

| Domain | Topic | IAA | Volume | | Description |
|---|---|---|---|---|---|
| | | | Twitter | YouTube | |
| Vz19 | maduro/dictator | 0.78 | 1,281,021 | 25,776 | Posts that describe Maduro as a dictator, or when its administration is represented as a dictatorship. |
| | arrests | 0.70 | 842,546 | 1,944 | Describes arrests or people who have been taken prisoners during on-the-ground events. |
| | other/chávez | 0.70 | 958,450 | 18,372 | Posts with terms related to the late president of Venezuela, Hugo Chávez. |
| | military | 0.69 | 3,502,074 | 12,895 | Posts about the Venezuelan military, security forces, or other armed militarized organization. |
| | international/aid | 0.68 | 2,336,919 | 12,168 | Descriptions of foreign humanitarian aid sent or requested to be sent to Venezuela. |
| CPEC | controversies/china/border | 0.91 | 180,078 | 260 | Discussions related to the disputed Chinese-Indian border. |
| | leadership/sharif | 0.89 | 82,327 | 132 | Positively discusses leadership by Nawaz Sharif, former prime minister of Pakistan. |
| | controversies/china/uighur | 0.85 | 25,854 | 52 | Posts that bring attention to the Uighur controversies in Xinjiang China. |
| | controversies/pakistan/baloch | 0.85 | 84,621 | 410 | Statements that advocate for Baloch independence, or emphasize exploitation of Baloch by Pakistan or China. |
| | benefits/development/roads | 0.83 | 43,831 | 222 | Discussions on road or railway projects in Pakistan related to CPEC. |
| BRIA | covid | 0.82 | 774,284 | 2,018 | Coronavirus and related topics such as vaccines in Africa. |
| | debt | 0.82 | 23,803 | 205 | Debts or loans, where the lender is China and the lendee is an African nation |
| | infrastructure | 0.78 | 28,397 | 205 | Roads, railways, ports or other infrastructure project as part of the BRI in Africa. |
| | travel | 0.78 | 224,582 | 815 | Travel between countries, usually by flights. Often in the context of COVID and related to Africa and China. |
| | mistreatment | 0.68 | 58,306 | 670 | Description of mistreatment of particular groups of people such as Kenyans, Africans, or Chinese people. |

Table 6.2: Time periods chosen for data. Split into training, validation and testing sets.

| Domain | Training (52 days) | Validation (1 week) | Testing (1 week) |
|---|---|---|---|
| Vz19 | 2018-12-24 to 2019-02-14 | 2019-02-15 to 2019-02-21 | 2019-02-22 and 2019-02-28 |
| CPEC | 2020-03-30 to 2020-05-21 | 2020-05-22 to 2020-05-28 | 2020-05-29 and 2020-06-04 |
| BRIA | 2020-02-01 to 2020-03-24 | 2020-03-25 to 2020-03-31 | 2020-04-01 and 2020-04-07 |

## 6.2 Methodology

### 6.2.1 Overall Model Setups

The prediction windows of both the ARIMA and Persistence Baseline models were 168 hours (i.e. 1 week). The Persistence Baseline models simply predict by shifting events from the past 168 hours.

The ARIMA models had different hyperparameters based on the results of a grid search over the validation set. Recall that ARIMA's hyperparameters are *p, d,* and *q*. The p and q hyperparameters are lookback parameters, and the values tried for them were 24, 48, 72, and 96. The *d* values tried were 0, 1, and 2. The RMSE metric was used to decide which combination of hyperparameter values to use.

### 6.2.2 Model Setup

As previously mentioned, the objective is to understand how the baselines perform under different prediction window durations.

To this end, we developed 2 sets of models, *Short-Term* and *Long-Term* models. Both sets of models predict 168 hours out into the future, but in different ways.

The Long-Term models predict 168 hours of Twitter or YouTube activity in one full block. For example, the Long-Term Persistence Baseline uses the activity volume time series from the past 168 hours to predict the activity volume time series in the future 168 hours. The Long-Term ARIMA model predicts 168 hours out into the future without any additional input ground truth data than what it was initially provided with.

The Short-Term models predict 168 hours in 24 hour chunks, with ground truth from the previous time period fed in at each interval. Since the Short-Term models receive more "input ground truth feedback" over time, one can intuitively expect the Short-Term models to have less predictive error compared to the Long-Term models.

### 6.2.3 Metrics Used

In order to be consistent, we used the same 6 metrics used throughout this dissertation: RMSE, MAE, SkE, VE, S-APE, and NC-RMSE.

Recall that we use RMSE and MAE to measure "volume over exact timesteps". We use Skewness Error (SkE) to measure the skewness of the predicted time series. Volatility Error (VE) measures the volatility of the predicted time series. Symmetric Absolute Percentage Error, or S-APE, measures the overall volume of the predicted time series. Lastly, NC-RMSE is used to measure the "overall temporal pattern" of the predicted time series without regard to volume or exact timestep. As previously mentioned throughout this dissertation, these metrics were chosen because they capture multiple characteristics of a predicted time series.

## 6.3 Results

The Figure 6.1 heatmap shows the Short-Term prediction results and the Figure 6.2 heatmap shows the Long-Term prediction results of the Persistence Baseline and ARIMA models. In these heatmaps, if the Persistence Baseline Model outperforms the ARIMA model, a cell is green, otherwise, the cell is white. The values in each cell are the percent improvement scores of the Persistence Baseline models over the ARIMA models (thus, the larger the value, the better Persistence Baseline performs compared to ARIMA in the metric shown on the x axis and on the topic shown on the y axis).

For short-term (one day) predictions (Figure 6.1) the Persistence Baseline model outperforms ARIMA on most topics and most metrics on both platforms. For the Short-Term Twitter predictions (Figure 6.1a), the Persistence Baseline model wins in 55 out of 90 trials, or about 61% of the time. ARIMA won only won about 39% of the time. Similarly, for YouTube, the Persistence Baseline model wins 56 out of 90, or 62% of the time (Figure 6.1b). ARIMA only won about 38% of the time.

As one can see in Figures 6.1 and 6.1a, the Persistence Baseline is especially strong in the SkE, S-APE, and VE metrics in this configuration. It struggles, to a degree, with NC-RMSE. Lastly, it performs much worse than ARIMA on the RMSE and MAE metrics.

However, the Persistence Baseline model loses its relative advantages over ARIMA in the Long-Term configuration, as shown in Figure 6.2. In the Long-Term setup on Twitter data (6.2a), the Persistence Baseline model wins only 49% of the time, or 44 out of 90 times.

For YouTube (Figure 6.2b), the Persistence Baseline model outperforms ARIMA about 46% of the time, or 41 out of 90 times.

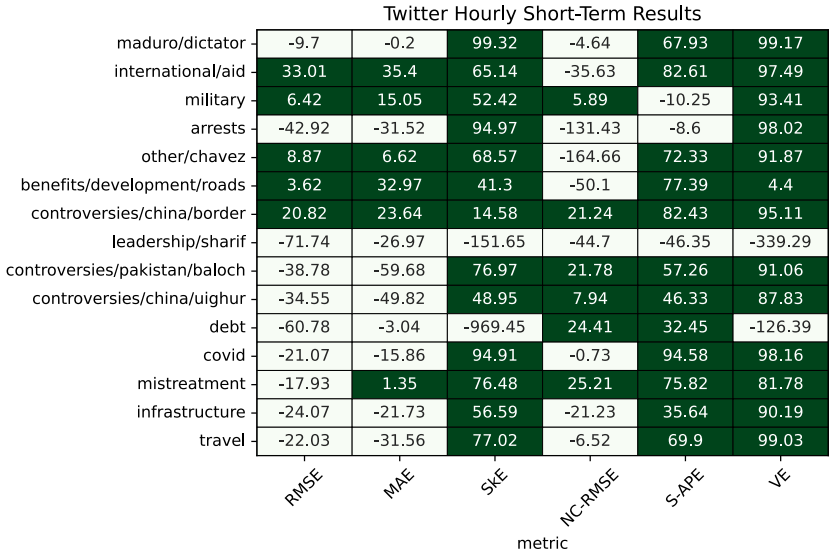
Moreover, the advantage that ARIMA gained in the long term predictions are mainly in the volume-related metric (S-APE), the exacting-timing-volume metrics (RMSE and MAE), and the "temporal pattern" metric, NC-RMSE. The Persistence Baseline only maintained its advantages in the skewness and volatility metrics (SkE and VE). They are both metrics related to the "burstiness" or "erraticness" of a time series.

So, in conclusion, if one wishes to predict if a time series will have any erratic or bursty behavior, whether short or long-term, one should use the Persistence Baseline model because it performs well on SkE and VE in both the short-term and long-term scenarios. If one also wishes to predict overall volume on a more short-term basis, and the exact-timing and temporal pattern are of less concern, one should use the Persistence Baseline, because it performs well on S-APE in the short term.

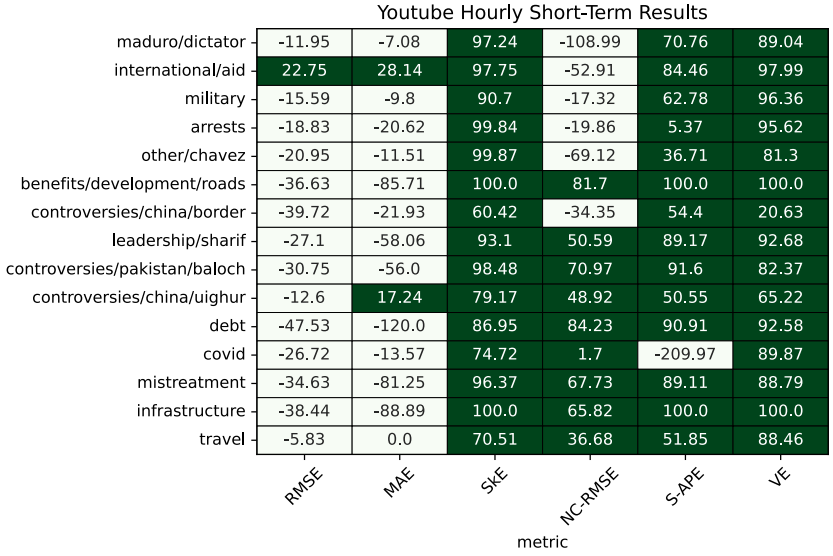
On the other hand, if one wishes to capture activity volume over exact timing, one should use ARIMA, because it performed the best on RMSE and MAE in both the long and short-term settings.

## 6.4 Conclusion

In this chapter, we sought to better understand the differences between the ARIMA and Persistence Baseline models. We examined their predictive power over 6 datasets, which were the VZ19, CPEC, and BRIA Twitter and YouTube datasets.

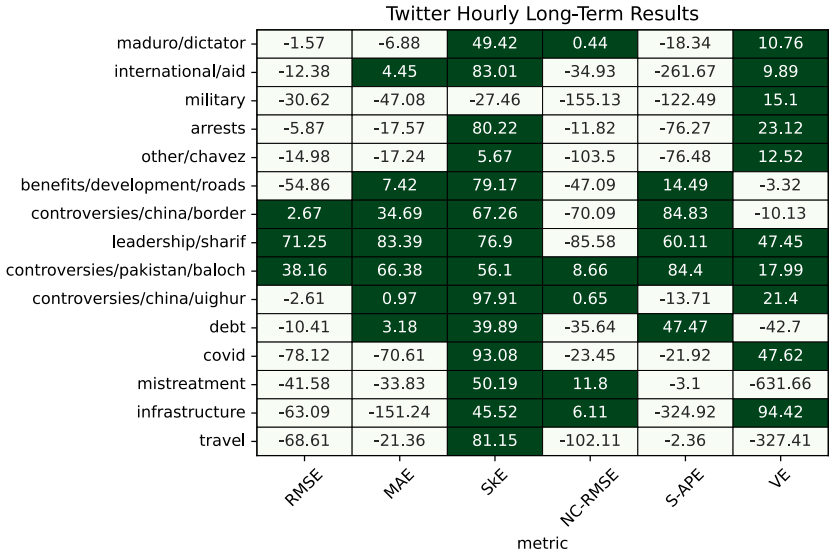



(a) Twitter (Short-Term)

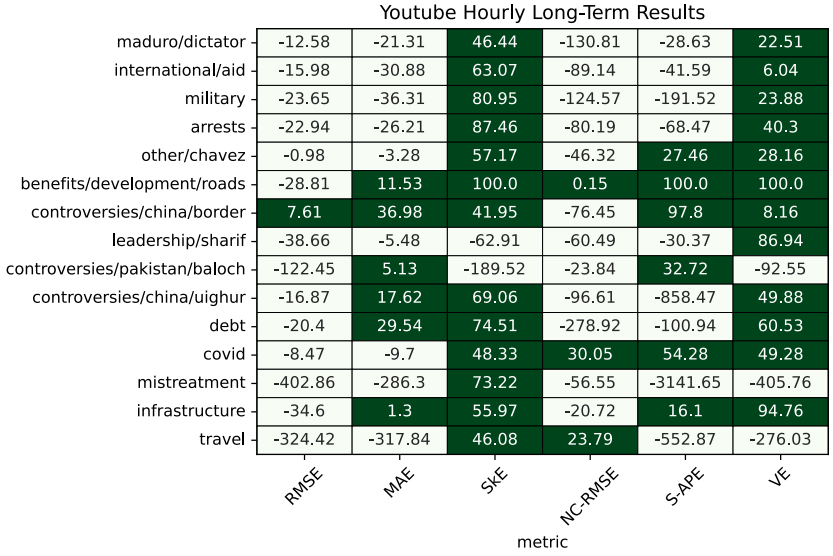



(b) YouTube (Short-Term)

Figure 6.1: Persistence Baseline vs. ARIMA (short term) heatmaps. A cell is green if the Persistence Baseline model outperformed the ARIMA model on the particular topic-metric pair.




(a) Twitter (Long-Term)




(b) YouTube (Long-Term)

Figure 6.2: Persistence Baseline vs. ARIMA (long term) heatmaps. A cell is green if the Persistence Baseline model outperformed the ARIMA model on the particular topic-metric pair.

We found that the Persistence Baseline is the more powerful baseline for short-term predictions, especially when trying to predict the scale of activities (as measured by S-APE) and "burstiness" of activities (as measured by SkE and VE).

We found that in the Long-Term, the Persistence Baseline tends to degrade, and that ARIMA is preferred if one wishes to capture overall scale (S-APE) or exact-timing and scale (MAE and RMSE).

The insights obtained from this study could potentially aid a future researcher in deciding the best type of time series model to use, and to understand how to interpret any new models he or she creates.

# Chapter 7: Conclusion and Future Work

## 7.1 Conclusion

In this dissertation, we explored the use of various machine learning methods for time series prediction and user-level activity prediction in social media networks. We addressed the various types of challenges that these tasks entail including: (1) accounting for the differences in user engagement among different social media platforms, (2) identifying the data required for accurate predictions, (3) selecting the appropriate prediction framework, and (4) metric selection.

In Chapter 3 we introduced VAM, an end-to-end approach for time series prediction and user-to-user temporal link prediction. We showed its predictive prowess on the Venezuela 2019 (Vz19) Twitter dataset. It was found that VAM outperformed both the baseline and state-of-the-art methods for the time series prediction task of (1) number of new users, (2) number of old users, and (3) number of activities. The baselines compared against were the Persistence Baseline, ARIMA, ARMA, AR, and MA models. The state-of-the-art methods compared against were the tNE-node2vec-H, tNE-node2vec-S, and tNE-DeepWalk embedding models.

For the Volume-Prediction task, we found that exogenous features from Reddit help improve prediction accuracy of Twitter activity. We also found that XGBoost models are the more ideal choice for volume prediction instead of RNNs, because they are much quicker to train and more accurate. This is notable because RNNs are one of the most frequently used approaches for social media prediction.

For the user-assignment task, we showed that VAM was able to predict the activity of both old and new users. Old user activity was modelled using historical data. Synthetic new

users were created using what we called a New User Archetype Table. We found that VAM was able to outperform the Persistence Baseline and the tNE models for the user-assignment task.

We noted that despite being state-of-the-art embedding approaches, the tNE models performed the worst out of all models. We mentioned that this is possibly due to the way in which they were modelled. For the time series prediction task, VAM directly predicted the overall volume time series. However, the tNE models predicted the time series of the user-to-user edges, which is a much more granular task. To obtain the macroscopic time series, the user-to-user time series were aggregated to the appropriate overall hourly granularity. We noted that it is problematic because on the granularity of user-to-user activity, most users perform very little activity. Because of this, models trained on this data are likely to predict that nearly nothing happens. In this work, we showed it is better to predict user-level activity by first predicting the overall volume of users and activities, and then using historical data to predict who the most likely users are to act in the future.

In Chapter 4, we applied VAM to the China-Pakistan economic corridor (CPEC) Twitter dataset. By using a different dataset than the Vz19 dataset of 3, we lend more credence to the idea that VAM is a generalizable framework for predicting user-level activity on social media networks. Also, unlike the Vz19 dataset, we used VAM to predict tweets, retweets, quotes, and replies, as opposed to just tweets and regular retweets (no quotes or replies which are less frequent). Similar to Chapter 3, we analyzed VAM's performance against 5 traditional time series baselines for the Volume-Prediction task: Persistence Baseline, ARIMA, ARMA, AR, and MA.

Furthermore, unlike Chapter 3, we examined the time series features used in the Volume Prediction module of VAM to better understand which features helped with predicting each time series. Lastly, we showed that across both the Vz19 and CPEC datasets, VAM performs better on time series with low Skewness and low Coefficients of Variation.

In Chapter 5 we created two variations of SMOTER - SMOTER-BIN and SMOTER-NB. SMOTER-BIN bins the time series samples into different classes based on the Frobenius norm of their outputs. Then SMOTER is applied to the minority samples. SMOTER-NB applies SMOTER indiscriminately to all samples without regard to minority or majority classes. We created two variations of VAM using these datasets, SMOTER-B-VAM and SMOTER-NB-VAM.

We then created ensemble models of the SMOTER-VAM models and regular VAM models using various time series attributes. The best performing non-binning ensemble model was the NB-SVE-low-volume ensemble (Non-Binning SMOTER-VAM Ensemble for low volume). The best binning ensemble model was the B-SVE-low-volume. This indicates that SMOTER-based data augmentation is especially effective on time series with low-volume.

In Chapter 6 we compared two common time series baselines in social media data, the Persistence Baseline and ARIMA baseline. We compared their performances across 6 datasets in two settings - *long-term* vs. *short-term* configurations. We found that the Persistence Baseline is the more powerful baseline for short-term predictions, especially for the scale of activity (as measured by S-APE), and "burstiness" of activities (as measured by Skewness Error and Volatitlity Error). We also found that in the long-term, ARIMA is preferred if one wishes to capture overall scale of activities (S-APE) or exact-timing and scale (MAE and RMSE).

## 7.2 Future Work

There are a myriad of directions for future work. For the VAM models, we could explore the use of other types of Volume Prediction "backend" models. One that we could try, for instance, is the Transformer neural network, which is a more recent type of temporal neural network. Unlike RNNs, transformers process the entire input sequence at once using "attention mechanisms". RNNs on the other hand, process one input sequence at a time.

For the User-Assignment modules, we could try to integrate more machine learning components rather probabilistic ones. For example, we could use a machine learning model to predict the most likely active users, and another machine learning model to predict the links between users.

For the SMOTER experiments, future work would include trying other time series attributes for ensembling each model. We could also try comparing SMOTER's performance to other data augmentation methods such as GANs or Variational Autoencoders.

Lastly, for the baseline analysis, future work would involve the comparison of other baseline approaches in addition to the ARIMA and Persistence Baseline, such as ARMA, AR, and MA.

As one can see, this dissertation has provided a myriad of approaches and analysis for machine learning methodologies in social media prediction. It is our hope that this work could be used as a guide to other researchers who are also trying to create social media prediction models.

The code used in this work can be found at:
https://github.com/fmubang/Dissertation-Code/tree/main.

## 7.3 Publications

Here is a list of publications of the author of this work:

- Fred Mubang and Lawrence O. Hall. VAM: An End-to-End Simulator for Time Series Regression and Temporal Link Prediction in Social Media Networks. *IEEE Transactions on Social Computing* (2022)

- Fred Mubang and Lawrence O. Hall. Simulating New and Old Twitter User Activity with XGBoost and Probabilistic Hybrid Models. *21st International Conference on Machine Learning and Applications* (2022 - Accepted)

- Kin Ng, Fred Mubang, Lawrence O. Hall, John Skvoretz, and Adriana Iamnitchi. Experimental evaluation of baselines for forecasting social media timeseries. *EPJ Data Science (Under Review)*

- Renhao Liu, Fred Mubang, and Lawrence O. Hall. Simulating Temporal User Activity on Social Networks with Sequence to Sequence Neural Models. *IEEE SMC International Conference* (2020)

- Renhao Liu, Fred Mubang, Lawrence O. Hall, Sameera Horawalavithana, Adriana Iamnitchi, and John Skvoretz. Predicting Longitudinal User Activity at Fine Time Granularity in Online Collaborative Platforms. *IEEE SMC International Conference* (2019)

## References


[1] Tarek Abdelzaher, Jiawei Han, Yifan Hao, Andong Jing, Dongxin Liu, Shengzhong Liu, Hoang Nguyen, David Nicol, Huajie Shao, Tianshi Wang, Shuochao Yao, Yu Zhang, Omar Malik, Stephen Dipple, James Flamino, Fred Buchanan, Sam Cohen, Gyorgy Korniss, and Boleslaw Szymanski. Multiscale online media simulation with socialcube. *Computational and Mathematical Organization Theory*, 26, 06 2020.

[2] Sinan G. Aksoy, Kathleen E. Nowak, Emilie Purvine, and Stephen J. Young. Relative hausdorff distance for network analysis. *Appl Netw Sci 4*, page 80, 2019.

[3] Ramya Akula, Ivan Garibay, and Niloofar Yousefi. Deepfork: Supervised prediction of information diffusion in github. In *Conference: Proceedings of the International Conference on Industrial Engineering and Operations ManagementAt: Bangkok, Thailand, March 5-7, 2019*, 03 2019.

[4] Meysam Asgari-Chenaghlu. Word vector representation, word2vec, glove, and many more explained, 02 2017.

[5] BBC. Venezuela crisis: How the political situation escalated. August 2021.

[6] Neda Hajiakhoond Bidoki, Alexander V. Mantzaris, and Gita Sukthankar. An lstm model for predicting cross-platform bursts of social media activity. *Information*, 10(12):1–13, 2019.

[7] J. Blythe, Emilio Ferrara, D. Huang, Kristina Lerman, Goran Murić, Anna Sapienza, A. Tregubov, Diogo Pacheco, John Bollenbacher, A. Flammini, Pik-Mai Hui, and F. Menczer. The darpa socialsim challenge: Massive multi-agent simulations of the github ecosystem. In *AAMAS*, 2019.

[8] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl_3):7280–7287, 2002.

[9] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[10] Jonathan Bright, Scott Hale, Bharath Ganesh, Andrew Bulovsky, Helen Margetts, and Phil Howard. Does campaigning on social media make a difference? evidence from candidate use of twitter during the 2015 and 2017 u.k. elections. *Communication Research*, 47, 10 2017.

[11] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, pages 30(1–7):107–117, 1998.

[12] Qi Cao, Huawei Shen, Keting Cen, Wentao Ouyang, and Xueqi Cheng. Deephawkes: Bridging the gap between prediction and understanding of information cascades. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 1149–1158, New York, NY, USA, 2017. Association for Computing Machinery.

[13] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *Int. J. Math. Model. Meth. Appl. Sci.*, 1, 01 2007.

[14] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 16:321–357, 06 2002.

[15] Guandan Chen, Qingchao Kong, Nan Xu, and Wenji Mao. Npp: A neural popularity prediction model for social media content. *Neurocomputing*, 333:221–230, 2019.

[16] Tianqi Chen and Carlos Gestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, August 2016.

[17] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[18] Francois Chollet et al. Keras. *GitHub*, 2015.

[19] B. Christofaro and S. Baker. A timeline of the political crisis in venezuela, which began with claims of election rigging and has now led to an attempted military coup.

[20] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[21] Gerrit Jan de Bruin, Cor J. Veenman, H. Jaap Herik, and Frank W. Takes. Supervised temporal link prediction in large-scale real-world networks. *Social Network Analysis and Mining*, 11:1–16, 2021.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

[23] David Doane and Lori Seward. Measuring skewness: A forgotten statistic? *J. Stat. Educ.*, 19, 07 2011.

[24] Ivan Garibay, Toktam A. Oghaz, Niloofar Yousefi, Ece C. Mutlu, Madeline Schiappa, Steven Scheinert, Georgios C. Anagnostopoulos, Christina Bouwens, Stephen M. Fiore, Alexander Mantzaris, John T. Murphy, William Rand, Anastasia Salter, Mel Stanfill, Gita Sukthankar, Nisha Baral, Gabriel Fair, Chathika Gunaratne, Neda B. Hajiakhoond, Jasser Jasser, Chathura Jayalath, Oliva Newton, Samaneh Saadat, Chathurani Senevirathna, Rachel Winter, and Xi Zhang. Deep agent: Studying the dynamics of information spread and evolution in social networks, 2020.

[25] Felix Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12:2451–71, 10 2000.

[26] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4, 2005.

[27] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[28] Neda Hajiakhoond Bidoki, Madeline Schiappa, Gita Sukthankar, and Ivan Garibay. Modeling social coding dynamics with sampled historical data. *Online Social Networks and Media*, 16:100070, 03 2020.

[29] Anthony Hernandez, Kin Ng, and Adriana Iamnitchi. Using deep learning for temporal forecasting of user activity on social media: Challenges and limitations. In *Companion Proceedings of the Web Conference 2020*, pages 331–336, April 2020.

[30] Anthony Hernandez, Kin Wai NG, and Adriana Iamnitchi. Using Deep Learning for Temporal Forecasting of User Activity on Social Media: Challenges and Limitations. In *Proceedings of Temporal Web Analytics Workshop, Companion Proceedings of The 2020 World Wide Web Conference (TempWeb'20)*, Taipei, Taipei, April 2020.

[31] Alyssa Hirose. 114 social media demographics that matter to marketers in 2022.

[32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[33] Sameera Horawalavithana, Abhishek Bhattacharjee, Renhao Liu, Nazim Choudhury, Lawrence O. Hall, and Adriana Iamnitchi. Mentions of Security Vulnerabilities on Reddit, Twitter and GitHub. In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI'19)*, Thessaloniki, Greece, Oct 2019.

[34] Sameera Horawalavithana, Nazim Choudhury, John Skvoretz, and Adriana Iamnitchi. Online discussion threads as conversation pools: predicting the growth of discussion threads on reddit. *Computational and Mathematical Organization Theory*, 28, 06 2022.

[35] Mohammad Raihanul Islam, Sathappan Muthiah, Bijaya Adhikari, B. Aditya Prakash, and Naren Ramakrishnan. Deepdiffuse: Predicting the 'who' and 'when' in cascades. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1055–1060, 2018.

[36] M.A. Jayaram, Gayitri Jayatheertha, and Ritu Rajpurohit. Time series predictive models for social networking media usage data: The pragmatics and projections. *Asian Journal of Research in Computer Science*, pages 37–50, August 2020.

[37] Josh Kamps and Bennett Kleinberg. To the moon: defining and detecting cryptocurrency pump-and-dumps. *Crime Science*, 7, 11 2018.

[38] Cheng Li, Jiaqi Ma, Xiaoxiao Guo, and Qiaozhu Mei. Deepcas: An end-to-end predictor of information cascades. In *Proceedings of the 26th International Conference on World Wide Web*, pages 577–586. International World Wide Web Conferences Steering Committee, 2017.

[39] Renhao Liu, Frederick Mubang, and Lawrence O. Hall. Simulating temporal user activity on social networks with sequence to sequence neural models. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1677–1684, 2020.

[40] Renhao Liu, Frederick Mubang, Lawrence O. Hall, Sameera Horawalavithana, Adriana Iamnitchi, and John Skvoretz. Predicting longitudinal user activity at fine time granularity in online collaborative platforms. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2535–2542, 2019.

[41] Sreekanth Madisetty and Maunendra Sankar Desarkar. Social media popularity prediction of planned events using deep learning. In Djoerd Hiemstra, Marie-Francine Moens, Josiane Mothe, Raffaele Perego, Martin Potthast, and Fabrizio Sebastiani, editors, *Advances in Information Retrieval*, pages 320–326, Cham, 2021. Springer International Publishing.

[42] Fred Mubang and Lawrence O. Hall. Simulating user-level twitter activity with xgboost and probabilistic hybrid models. *arXiv:2202.08964*, 2022.

[43] Fred Mubang and Lawrence O. Hall. Vam: An end-to-end simulator for time series regression and temporal link prediction in social media networks. *IEEE Transactions on Computational Social Systems*, pages 1–12, 2022.

[44] Carlos Muniz, Ronaldo Goldschmidt, and Ricardo Choren. Combining contextual, temporal and topological information for unsupervised link prediction in social networks. *Knowledge-Based Systems*, 156, 05 2018.

[45] Goran Murić, Alexey Tregubov, Jim Blythe, Andrés Abeliuk, Divya Choudhary, Kristina Lerman, and Emilio Ferrara. Massive cross-platform simulations of online social networks. AAMAS '20, page 895–903, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[46] Kin Ng, Sameera Horawalavithana, and Adriana Iamnitchi. Social media activity forecasting with exogenous and endogenous signals. *Social Network Analysis and Mining*, 12, 12 2022.

[47] Kin Ng, Frederick Mubang, Lawrence Hall, John Skvoretz, and Adriana Iamnitchi. Experimental evaluation of baselines for forecasting social media timeseries. *EPJ Data Science (Under Review)*, 2022.

[48] Siobhán O'Grady. The u.s. says maduro is blocking aid to starving people. the venezuelan says his people aren't beggars.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[50] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[51] Zhenyu Qiu, Jia Wu, Wenbin Hu, Bo Du, Guocai Yuan, and Philip Yu. Temporal link prediction with motifs for social networks. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[52] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *IEEE Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 59–66, 1998.

[53] Samaneh Saadat, Chathika Gunaratne, Nisha Baral, Gita Sukthankar, and Ivan Garibay. Initializing agent-based models with clustering archetypes. In Robert Thomson, Christopher Dancy, Ayaz Hyder, and Halil Bisgin, editors, *Social, Cultural, and Behavioral Modeling*, pages 233–239, Cham, 2018. Springer International Publishing.

[54] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[55] Kalyani Selvarajah, Ziad Kobti, and Mehdi Kargar. *Link Prediction by Analyzing Temporal Behavior of Vertices*, pages 257–271. 06 2020.

[56] Prasha Shrestha, Suraj Maharjan, Dustin Arendt, and Svitlana Volkova. Learning from dynamic user interaction graphs to forecast diverse social behavior. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 2033–2042, New York, NY, USA, 2019. Association for Computing Machinery.

[57] Olivia Simpson, C. Seshadhri, and Andrew McGregor. Catching the head, tail, and everything in between: A streaming algorithm for the degree distribution. *2015 IEEE International Conference on Data Mining*, pages 979–984, 2015.

[58] Uriel Singer, Ido Guy, and Kira Radinsky. Node embedding over temporal graphs. In *Proceedings of the 28th International Joint Conference on AI (IJCAI-19)*, August 2019.

[59] Samia Tasnim, Md Hossain, and Hoimonty Mazumder. Impact of rumors or misinformation on coronavirus disease (covid-19) in social media. *Journal of Preventive Medicine and Public Health*, 53, 04 2020.

[60] Sara Thomson and William Kilbride. Preserving social media: The problem of access. *New Review of Information Networking*, 20:261–275, 07 2015.

[61] Luís Torgo, Rita P. Ribeiro, Bernhard Pfahringer, and Paula Branco. Smote for regression. In Luís Correia, Luís Paulo Reis, and José Cascalho, editors, *Progress in Artificial Intelligence*, pages 378–389, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[62] Luís Torgo, Rita Ribeiro, Bernhard Pfahringer, and Paula Branco. Smote for regression. volume 8154, pages 378–389, 09 2013.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 06 2017.

[64] Jia Wang, Vincent Zheng, Zemin Liu, and Kevin Chang. Topological recurrent neural network for diffusion prediction. 11 2017.

[65] Z.R. Yang and Z. Yang. 6.01 - artificial neural networks. In Anders Brahme, editor, *Comprehensive Biomedical Physics*, pages 1–17. Elsevier, Oxford, 2014.

[66] Shuochao Yao, Yifan Hao, Dongxin Liu, Shengzhong Liu, Huajie Shao, Jiahao Wu, Mouna Bamba, Tarek Abdelzaher, James Flamino, and Boleslaw Szymanski. A predictive self-configuring simulator for online media. In *2018 Winter Simulation Conference (WSC)*, pages 1262–1273, 2018.

[67] Yuhang Zhu, Shuxin Liu, Yingle Li, and Haitao Li. Tlp-ccc: Temporal link prediction based on collective community and centrality feature fusion. *Entropy*, 24:296, 02 2022.

[68] Alper Özcan and Şule Gündüz Öğüdücü. Multivariate temporal link prediction in evolving social networks. In *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, pages 185–190, 2015.

## Appendix A: VAM - State of the Art Comparison Methodology

In this section, we describe how the tNodeEmbed models were setup in detail.

### A.1 Creating the Initial DeepWalk and Node2Vec Embeddings

Before creating tNodeEmbed embeddings, we had to firstly create DeepWalk and node-2vec embeddings. We call these initial embeddings DeepWalk, node2vec-H, and node2vec-S models.

Each node embedding represents a user and topic on a particular day, or a *(user, topic, day)* tuple. Embeddings were made for the time period spanning February 12th up to March 7th, 2019 (24 days). The parameters of the node2vec/DeepWalk embeddings are as follows. The dimensions of each embedding were set to 32. The dimension parameter controls how "large" one wants the embeddings to be. The number of random walks per node was set to 10. The walk length of each random walk was set to 200. These values were found via grid search. The dimensions tried were 8, 16, 32, and 64. The walks tried were 10 and 50. The lengths tried were 50 and 200.

### A.2 Creating the tNodeEmbed Embeddings

After creating the DeepWalk, node2vec-H, and node2vec-S embeddings, the next step was to create the tNodeEmbed embeddings. These embeddings were made by performing an "alignment" operation on the aforementioned node2vec and DeepWalk embeddings as discussed in [58]. In other words, the node embeddings on day 2 are aligned with the embeddings on day 1, and the embeddings on day 3 are aligned with day 2, etc. By performing this operation, performance is improved on downstream tasks as shown in [58]. The new

embeddings created from this operation are known as "tNodeEmbed" embeddings. The newly aligned *DeepWalk* embeddings are referred to as *tNE-DeepWalk*. The newly aligned *node2vec-H* embeddings are referred to as *tNE-node2vec-H*, and the newly aligned *node2vec-S* embeddings are referred to as *tNE-node2vec-S*. For more details on how the alignment operation works, refer to [58].

### A.3 Training and Testing Neural Networks

Once all 3 sets of user embeddings are created, they can then be used to perform prediction tasks with a neural network. The prediction task was as follows: given a *(child, parent, topic, day)* tuple (represented as a vector), predict the number of topic-related tweets/retweets that the child-parent edge will post over the next 24 hours.

To represent a child-parent edge, the node embeddings for the child and parent were concatenated with one another. Recall that the number of dimensions used for each node embedding was 32. Since, this was the case, the concatenated child-parent vector had 64 values. The output vector represented the time series, and had 24 values.

Three fully-connected neural networks were trained (1 for each embedding approach). Each network was comprised of 3 hidden layers with 100, 50, and 25 hidden units respectively. Dropout layers were used after each layer with the rate set to 20%. The activation function used after each hidden layer was tanh. The linear activation function was used in the output layer. MinMax scaling was used to normalize the features. The learning rate was 0.0001. The batch size was 32. The number of epochs was set to 100. Early stopping was used with a "patience" parameter set to 10. An Adam optimizer was used.

The loss function used was *Weighted Cumulative MSE*. It is a variation of the *MSE* loss function that performs the MSE operation on the cumulative sum of the ground truth and prediction vectors. We took the cumulative sum because it does not punish the model as harshly for predicting the "correct value" but at the "wrong timestep". In our initial experiments we used MSE as a loss function, but our models predicted all 0's. We believe

this is because most users perform no actions most of the time, so a model trying to minimize loss is incentivized to predict only 0's. Since MSE requires "exact timing", it exacerbates the tendency of a model to do this.

Furthermore, we weighted our loss function so that incorrectly predicting a 0 when an action did occur would incur a greater penalty. We used a cost parameter we call $C$ and it was set to 2. So in other words, the loss would be twice as large for predicting a 0 instead of a non-zero value where appropriate.

### A.4 Data Splits

Table A.1 shows the dates used for the training, validation, and test sets. Table A.2 shows the sizes of each dataset across each time period.

Recall that the test period is comprised of 21 days (Feb 15th to March 7th). For each of these 21 days, for each of the 3 different embedding approaches, a fully-connected neural network was trained on a training period of 3 days preceding the test day of interest, and then tested upon the test day of interest. We used only 3 days for training data because we tried multiple days between 1-4, and found 3 to yield the best results during validation. Furthermore, since there are so many edges in a given day, a few days is all that is needed in order to generate the tens of thousands of samples sufficient to train a neural network properly.

For validation, the samples in each training period were split in an 85%/15% ratio for train/validation sets. For example, in test period 1, there were 616,881 total samples (edges) between the train period of Feb 12-14, 2019. After performing the split, 524,348 of these samples went into the training set, and 92,533 were used for validation. A neural network was then trained and validated on these edges, and tested with the test set of 83,095 samples.

## A.5 Creating the Test Day Initial Conditions

An important decision to make is figuring out how many samples, or user-to-user edges, one will use in the test set for prediction. A simple solution would be to use the entire historical dataset of edges as the initial condition set (from Dec 28th 2018, the start of our dataset to the present). However, since there are so many edges in this timespan, that would be very expensive in terms of computational time and space.

We decided upon using a lookback period of 1 day (24 hours) to generate the initial condition period. For example, the 83,095 initial condition edges in the 1st test period (February 15th) is the set of all active edges from the previous day (February 14th). The assumption here is that the active future edges will be similar to the recently past active edges. We used this lookback period of 1 day because it is the same lookback period we used for the VAM user-assignment module. This allows for a fair comparison between the VAM and node-embedding approaches.

Something that one must keep in mind is that the initial condition edges are from the previous day of ground truth edges. The true number of active edges is obviously not "known" until that day has passed. This is why the number of some of the test edges in Table A.2 exceeds that of the training period edges.

For example, on test period 10 in the table, the number of edges in the training set is about 1.9 million, while the test set has about 2.2 million initial condition edges. This is because the final day of training for this test period in February 23rd. What this means is that during this training period, the model is taking as input in the set of old edges that exist up to February 22nd, and predicting whether or not they will be active on Febrary 23rd. Obviously, on February 22nd, the model will not "know" of new edges that will exist on February 23rd. It can only predict using the set of previously existing edges. It is not until the initial condition day of February 23rd that the model will "know" the full set of new and old edges it can use to predict the activity for the next day of February 24th.

## A.6 Predicting New Users

The embedding methods do not have the ability to predict new users, so in order to incorporate new users into our predictions, within each daily temporal graph we created a node called "New User" which encapsulates all the activities that all new users would perform within a given day. For example, let us say for some daily temporal graph, $G^{day}$, all of the new users perform a total of 30,000 actions. Then we create a node for that day called "New User" that performs 30,000 actions. We found that adding this node to each daily graph improved prediction results for the tNodeEmbed models.

Even though this "New User Node" predicts how many activities new users will perform within a given day, we still need to assign these actions to actual new users. We do so in the following way.

Firstly, we predict the number of new users within a given day using the Persistence Baseline. So, if the Persistence Baseline predicted 10,000 new users, we use 10,000 new users as our prediction of the next day. We then generate new users and assign them different identifier strings.

Then, we randomly assign our prediction activities from the "New User" node to these newly generated users. So, let us say that the Persistence Baseline predicted 10,000 new users, and our "New User" node from the tNodeEmbed model predicted 30,000 new user activities. We then randomly assign these 30,000 activities to the 10,000 newly generated users. We note that this new user prediction method is rather imprecise and has room for improvement. However, it is one approach to a hard problem.

Table A.1: Embedding neural network date information.

| **Embedding Neural Network Date Info** | | | |
|---|---|---|---|
| **Test Period** | **Train Start** | **Train End** | **Test Day** |
| 1 | 2019-02-12 | 2019-02-14 | 2019-02-15 |
| 2 | 2019-02-13 | 2019-02-15 | 2019-02-16 |
| 3 | 2019-02-14 | 2019-02-16 | 2019-02-17 |
| 4 | 2019-02-15 | 2019-02-17 | 2019-02-18 |
| 5 | 2019-02-16 | 2019-02-18 | 2019-02-19 |
| 6 | 2019-02-17 | 2019-02-19 | 2019-02-20 |
| 7 | 2019-02-18 | 2019-02-20 | 2019-02-21 |
| 8 | 2019-02-19 | 2019-02-21 | 2019-02-22 |
| 9 | 2019-02-20 | 2019-02-22 | 2019-02-23 |
| 10 | 2019-02-21 | 2019-02-23 | 2019-02-24 |
| 11 | 2019-02-22 | 2019-02-24 | 2019-02-25 |
| 12 | 2019-02-23 | 2019-02-25 | 2019-02-26 |
| 13 | 2019-02-24 | 2019-02-26 | 2019-02-27 |
| 14 | 2019-02-25 | 2019-02-27 | 2019-02-28 |
| 15 | 2019-02-26 | 2019-02-28 | 2019-03-01 |
| 16 | 2019-02-27 | 2019-03-01 | 2019-03-02 |
| 17 | 2019-02-28 | 2019-03-02 | 2019-03-03 |
| 18 | 2019-03-01 | 2019-03-03 | 2019-03-04 |
| 19 | 2019-03-02 | 2019-03-04 | 2019-03-05 |
| 20 | 2019-03-03 | 2019-03-05 | 2019-03-06 |
| 21 | 2019-03-04 | 2019-03-06 | 2019-03-07 |

Table A.2: Embedding neural network dataset size information.

| **Embedding Neural Network Sample Size Info** | | | | |
|---|---|---|---|---|
| **Test Period** | **Total Train and Val Samples** | **Training Samples** | **Validation Samples** | **Test Samples** |
| 1 | 616,881 | 524,348 | 92,533 | 83,095 |
| 2 | 592,584 | 503,696 | 88,888 | 121,373 |
| 3 | 386,265 | 328,325 | 57,940 | 189,394 |
| 4 | 393,862 | 334,782 | 59,080 | 175,432 |
| 5 | 486,199 | 413,269 | 72,930 | 205,704 |
| 6 | 570,530 | 484,950 | 85,580 | 293,092 |
| 7 | 674,228 | 573,093 | 101,135 | 329,941 |
| 8 | 828,737 | 704,426 | 124,311 | 452,037 |
| 9 | 1,075,070 | 913,809 | 161,261 | 1,080,346 |
| 10 | 1,862,324 | 1,582,975 | 279,349 | 2,161,546 |
| 11 | 3,693,929 | 3,139,839 | 554,090 | 908,470 |
| 12 | 4,150,362 | 3,527,807 | 622,555 | 379,520 |
| 13 | 3,449,536 | 2,932,105 | 517,431 | 210,723 |
| 14 | 1,498,713 | 1,273,906 | 224,807 | 184,576 |
| 15 | 774,819 | 658,596 | 116,223 | 229,731 |
| 16 | 625,030 | 531,275 | 93,755 | 91,076 |
| 17 | 505,383 | 429,575 | 75,808 | 92,142 |
| 18 | 412,949 | 351,006 | 61,943 | 136,468 |
| 19 | 319,686 | 271,733 | 47,953 | 146,449 |
| 20 | 375,059 | 318,800 | 56,259 | 77,073 |
| 21 | 359,990 | 305,991 | 53,999 | 96,980 |

# Appendix B: VAM User Assignment Methodology - Additional Details

This section contains additional details regarding the VAM User-Assignment algorithm first introduced in Chapter 3.

## B.1 User-Assignment Symbols

Table B.1 on page 156 contains various symbols used in the User-Assignment algorithm for quick reference.

## B.2 Additional User-Assignment Implementation Details

If the *VP-Module* predicts more users than actions at some future timestep, $T+1$, then VAM changes the number of actions to equal the number of users. For example, if 2 users and 1 action were predicted to occur at $T+1$, then VAM simply changes the number of predicted actions to be 2.

If, for some future timestep $T+1$, the *VP-Module* predicted more old users than exist in the recent history table, $H^{recent}$, then VAM simply changes the number of predicted active old users to equal the number of old users that do exist in $H^{recent}$. For example, if the *VP-Module* predicts 100 old users will be active at $T+1$, but $H^{recent}$ only contains 90 old users, then VAM will change the number of predicted old users to be 90, and then sample all 90 active old users from $H^{recent}$ for user-activity assignment. Future work would involve experimenting with other ways to address this history table issue. For example, we could try expanding the history table to include more old users.

Table B.1: The various symbols used in the User-Assignment section of this work.

| **User-Assignment Module Symbols** | |
|---|---|
| ***Symbol*** | ***Meaning*** |
| $G$ | A temporal graph. It is a series of static graphs. |
| $u$ | a child user in edge $(u, v, w(u, v, t))$ |
| $v$ | a parent user in edge $(u, v, w(u, v, t))$ |
| $w(u, v, t)$ | The number of times child user u interacted with parent $v$ at time $t$. |
| $T$ | The prediction time step of interest |
| $S$ | The length of the output time series we wish to predict |
| $\hat{\boldsymbol{Y}}$ | The time series matrix consisting of the (1) number of actions, (2) number of new users, and (3) number of old users from time $T+1$ to $T+S$ |
| $\hat{G}^{future}$ | The temporal graph sequence that the user-assignment module must predict |
| $L^{user}$ | The user-assignment lookback factor. |
| $L^{vol}$ | The volume prediction lookback factor. |
| $s$ | Any given time step between 1 and $S$, inclusive. |
| $T+s$ | The current timestep of interest within the scope of the user assignment algorithm. $T+s = T+s-1$. |
| $p^{old\_act}$ | The probability that an old user will be active in a given time step. |
| $W^{old\_cand}$ | The probability weight table for old user candidates. |
| $W^{old}$ | The probability weight table for selected users from the $W^{old\_cand}$ table. |
| $\hat{O}_s$ | The set of predicted active old users in time step $T+s$ |
| $\hat{N}_s$ | The set of newly generated active users in time step $T+s$ |
| $W^{new\_arch}$ | The new user archetype weight table. Used to select the mostly likely archetypes a newly generated user will behave like. |
| $u^{arch}$ | A user archetype. |
| $p^{act\_arch}$ | How likely a user of a particular archetype will be active in $T+s$ |
| $p^{infl\_arch}$ | How likely a user of a particular archetype will be influential in $T+s$ |
| $u^{new}$ | A newly generated user. |

## B.3 User-Assignment Algorithm Step-by-Step in Detail

In this section we describe in more detail the steps to assign a user to an action via the user-assignment module.

### B.3.1 User Assignment - Inputs and Outputs

The inputs to the algorithm are the full temporal graph $G$, the number of output timesteps to be predicted, $S$, the user-assignment lookback factor, $L^{user}$, the volume prediction matrix $\hat{\boldsymbol{Y}} \in \mathbb{R}^{3\times S}$, the old user index *old_idx*, the new user index *new_idx*, and the activity index *act_idx*. The three indices are used to access the predicted number of old users, new users, and activities on the predicted timestep of interest from $\hat{\boldsymbol{Y}}$. The output of the *Assign_Users* algorithm is the temporal graph sequence, $\hat{G}^{future}$.

### B.3.2 Initializations

At the beginning of the algorithm, $G^{recent}$ is constructed with the *Get_Recent* function using the full temporal graph, $G$, and the lookback parameter, $L^{user}$. As mentioned in the main paper, the lookback factor parameter $L^{user}$ is used to determine the number of snapshots to use. For example, if $L^{user} = 24$, then only the 24 most recent graphs in sequence $G$ will be used to make $H^{recent}$. The assumption here is that recent history is all that is needed to make temporal network predictions.

Furthermore, note that the length of the full temporal graph sequence $G$ is $T$. So, for example, if $G$ contains $T = 100$ graphs, and $L^{user} = 5$, then only graphs 96 up to 100 will be in the sequence $G^{recent}$. Or, in other words, only graphs $T - L^{user} + 1$ up to $T$ are included in $G^{recent}$. Then, an empty array, $\hat{G}^{future}$ is created.

Recall that the number of prediction timesteps of interest is $S$. There is a for loop that iterates $S$ times for each $s \leq S$. This $s$ represents the current prediction timestep of interest. At each timestep $s$, the number of old users, new users, and activities are retrieved from their respective location in the $\hat{\boldsymbol{Y}}$ matrix.

### B.3.3 The History Table

The sequence $G^{recent}$ is used to construct a history table, $H^{recent}$, that contains the event tuples from $G$. For each iteration $s$, the *UA-Module* first uses $G^{recent}$ to construct a recent history table called $H^{recent}$. This table can be thought of as a hash table. Each key into the table is an integer representing a time step, $t$, spanning from $T - L^{user} + s$ up to $T + s - 1$. This range can be also be defined as the *current lookback period of interest*. Each time step key maps to an event table, $H_t^{recent}$. This table can be thought of as an array of "event tuples". Each tuple has the following form.

$$(u, v, w(u, v, t), \mathbb{I}_{V_t^{new}}(u), \mathbb{I}_{V_t^{new}}(v))$$

$u$ is the child user and $v$ is the parent user. $w(u, v, t)$ is a numerical value that represents how many times $u$ interacted with $v$ at time step $t$ in $G_t$.

$V_t$ is the set of all users in $G$ at some particular timestep $t$. $V_t^{new}$ is the set of all new users in $V_t$. $\mathbb{I}_{V_t^{new}}$ is an indicator function that returns 1 if a particular user, $u$ was in the new user set, $V_t^{new}$ at time step $t$, and 0 otherwise.

### B.3.4 Retrieving Old User Candidates

The *Get_Active_Old_User_Candidates* function is used to construct a table of old user candidates ($W^{old\_cand}$) from $H^{recent}$ and their likelihood of being active at time $s$.

This table can be thought of as an array of tuples, each with the following form: ($u$, $p^{old\_act}(u, T + s)$). The term $p^{old\_act}(u, T + s)$ represents the probability that user $u$ will be active at time step $T + s$. This probability is obtained by calculating, for each user $u$, the normalized average activity frequency of $u$ during the *current lookback period of interest*.

The assumption here is that a user's future probability of acting is equivalent to his past probability of acting.

### B.3.5 Retrieving Most Likely Old Users from Candidates

The *Get_Most_Likely_Active_Old_Users* function is then used to retrieve the set of most likely old users ($\hat{O}_s$) from this table of candidates, as well as a table containing their re-weighted probabilities ($W^{old}$).

In order to retrieve ($\hat{O}_s$) and $W^{old}$, *VAM* performs a weighted random sample on the $W^{old\_cand}$ table in order to create the set of most likely active old users at time $T+s$. The weights used for this random sample are the activity probabilities ($p^{old\_act}(u, T+s)$) for each old user that were calculated in the previous step. We call the set of predicted active users, $\hat{O}_s$. Furthermore, *VAM* creates a new weight table, called $W^{old}$, which is the same as $W^{old\_cand}$ minus any users that were not chosen by the weighted random sample. This new table, $W^{old}$ is needed when the time comes for *VAM* to predict how many actions each old user will perform. Old users with a higher probability of acting, a.k.a. $p^{old\_act}(u, T+s)$, are more likely to perform more actions.

### B.3.6 Creating the New User Set

Next, the set of new users ($\hat{N}_s$) is generated using the *Generate_New_Users* function. Recall that the number of new users is known because that was predicted from the *Volume-Prediction Module* and it is contained in the matrix $\hat{\boldsymbol{Y}}$.

### B.3.7 Assigning Attributes to New Users

The question that remains at this point is "How does one decide what actions the new users will perform?" *VAM* does this by constructing a *New User Archetype Table*. This table is comprised of *recently active users*. These are users that have appeared as new within the lookback factor period of $T+s-L^{user}$ up to $T+s-1$, which can also be referred to as the "recent history". The opposite of a *recently active* user would be a *long-acting user*, which would be a user who has appeared in $G$ before timestep $T+s-L^{user}$. The assumption

behind the archetype table is that new users in the future will behave in a similar manner to previous new users from the recent past.

The *New User Archetype Table* table contains the following information: (1) the name of the *recently active* a.k.a. *archetype user*, (2) the probability that this archetype would be active in any given timestep (e.g. via tweeting or retweeting), and (3) the probability that this archetype will be "influential" in any given timestep. In Twitter, probability of influence is measured by how often a user is retweeted.

With this in mind we now define a *new user archetype* record in the following way.

$$(u^{arch}, p^{act\_arch}(u^{arch}), p^{infl\_arch}(u^{arch}))$$

The term $p^{act\_arch}(u^{arch})$ is a weight that describes how likely it is that a user of archetype, $u^{arch}$ will be active in a future time step. The term, $p^{infl\_arch}(u^{arch})$ describes how likely it is that a user of archetype $u^{arch}$ will be influential in some future time step. We define influence as the quantity by which other users will respond to a post created by $u^{arch}$ in some social media platform with regards to topic $q$.

As an example, say there is some recently active user named Carol with an action probability of 0.2 and an influence probability of 0.1. Since Carol is a recently active user, she will be considered an archetype of new user and added to the new user archetype table, $W^{new\_arch}$. The record for "Carol" will be as follows.

$$(\text{“}Carol'', 0.2, 0.1)$$

Now, "Carol" archetype could be applied to a new generated user arbitrarily given the identifier "Phil". Even though Phil has been generated as a new user, he still needs to be assigned a probability of activity and probability of influence. *VAM* will randomly sample a user archetype from $W^{new\_arch}$ in order to assign attributes to Phil. If *VAM* randomly selects user archetype Carol, then *VAM* will assign Phil an activity probability of 0.2 and

an influence probability of 0.1. These values are, of course, then normalized relative to the other generated new users so that all users' probabilities lie between 0 and 1.

*VAM* then iterates over every new user $u^{new} \in \hat{N}_s$ and performs a weighted random sample to select a new user archetype tuple from $W^{new\_arch}$. This process then yields a new table, called the *new user attribute table*, or $W^{new}$. This table can be thought of as any array of tuples of the following form.

$$(u^{new}, u^{arch}, p^{act\_arch}(u^{new}), p^{infl\_arch}(u^{new}))$$

$u^{new}$ is the new user of interest. $u^{arch}$ is the archetype that this new user was created from. $p^{act\_arch}(u^{new})$ and $p^{infl\_arch}(u^{new}))$ are $u^{new}$'s probabilities of activity and influence, respectively. They are equivalent to the probability and influence probabilities of user archetype, $u^{arch}$. In other words, each user $u^{new} \in \hat{N}_s$ was assigned probability and activity attributes from some user archetype, $u^{arch} \in U^{arch}$. Note that $U^{arch}$ is the set of all user archetypes.

The *New User Archetype Table* is then used to assign the activity and influence probabilities to each new user in $\hat{N}_s$. These probabilities are stored in $W^{new}$.

### B.3.8 Creating the Old and New User Parent Tables

Now, *VAM* needs the most likely sets of parents that the old and new users will interact with. To that end, it creates what we call *parent distribution tables*.

Firstly, using the *Create_Old_User_Parent_Table* function, the old parent distribution table, $D^{old\_parent}$, is created. This table can be thought of as a hash table, in which each key is a user, $u$, and each user maps to a parent distribution table for that particular user. Each table has the form: $(v, p^{edge}(u, v, T + s))$. The term $p^{edge}(u, v, T + s))$ represents the probability that an edge will form between $u$ and $v$ at time $T + s$.

Next, VAM must create a parent distribution table for the new users using the *Create_New_User_Parent_Table*. In order to do so, it iterates over every new user record in

$W^{new}$. It then checks which user archetype, $u^{arch}$ the new user $u^{new}$ was created from. Recall that each new user archetype was created from users who have actually existed in the data, so it is possible for *VAM* to collect information regarding who their previous parents were. *VAM* will then create a parent distribution table for each $u^{arch}$ and will assign this parent distribution table to the appropriate new user $u^{new}$. The final table created, $D^{new\_parent}$, will be a new user parent distribution table, similar to $D^{old\_parent}$. Each key of the hash table is a new user, $u^{new}$, and each key hashes into a new user parent distribution table of the form $(v, p^{edge}(u^{new}, v, T+s))$. The term $p^{edge}(u^{new}, v, T+s))$ represents the probability that an edge will form between $u^{new}$ and $v$ at time $T+s$.

### B.3.9 Creating the Links

At this point, *VAM* now has the information it needs to perform link prediction. To that end, it uses the function, $Create\_Links$ to perform link prediction and create the final graph, $G_s^{future}$. The arguments to $Create\_Links$ are $\hat{O}_s$,$\hat{N}_s$,$num\_acts$ , $W^{old}$, $W^{new}$, $D^{old}$, and $D^{new}$. Note that *VAM* "knows" the total edge weight of all links in $G_s^{future}$ because the *Volume-Prediciton Module* predicted the total number of activities for each timestep $s \leq S$, hence the use of the argument, $num\_acts$.

### B.3.10 Updating the Recent Temporal Graph Sequence

The predicted graph, $\hat{G}^{future}$ is then used to update $G^{recent}$. The user-assignment for-loop then continues $S-1$ more times until the full $\hat{G}^{future}$ graph is predicted such that $\hat{G}^{future} = \{\hat{G}_1^{future}, \hat{G}_2^{future}, ..., \hat{G}_S^{future}\}$.

## Appendix C: Copyright Permissions

The permission as shown below is for the reuse of content as appeared in Chapter 3. This screenshot was taken via the following link: IEEE license info webpage.

### Can I Reuse My Published Article in My Thesis?

You may reuse your published article in your thesis or dissertation without requesting permission, provided that you fulfill the following requirements depending on which aspects of the article you wish to reuse.

- **Text excerpts:** Provide the full citation of the original published article followed by the IEEE copyright line: © 20XX IEEE. If you are reusing a substantial portion of your article and you are not the senior author, obtain the senior author's approval before reusing the text.
- **Graphics and tables:** The IEEE copyright line (© 20XX IEEE) should appear with each reprinted graphic and table.
- **Full text article:** Include the following copyright notice in the references: "© 20XX IEEE. Reprinted, with permission, from [full citation of original published article]."

When posting your thesis on your university website, include the following message:

> "In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [name of university or educational entity]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org /publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation."

Only the accepted version of your article, ***not the final published version***, may be posted online in your thesis.

The permission as shown below is for the reuse of content as appeared in Chapter 4. This screenshot was taken via the following link: https://arxiv.org/help/license/reuse.

## Do I need arXiv's permission to repost the full text?

**Note:** All e-prints submitted to arXiv are subject to copyright protections. arXiv is not the copyright holder on any of the e-prints in our corpus.

In some cases, submitters have provided permission in advance by submitting their e-print under a permissive Creative Commons license. The overwhelming majority of e-prints are submitted using the arXiv perpetual non-exclusive license, which does not grant further reuse permissions directly. In these cases you will need to contact the author directly with your request.

## How can I determine what license the version was assigned?

All arXiv abstract pages indicate an assigned license underneath the "Download:" options.

The link may appear as just the text `(license)`, such as at arXiv:2201.14176. Articles between 1991 and 2003 have an assumed license. These are functionally equivalent to the arXiv non-exclusive license.

If the license applied by the submitter is one of the Creative Commons licenses, then a "CC" logo will appear, such as at arXiv:2201.04182.

## I want to include a paper of mine from arXiv in my thesis, do I need specific permission?

If you are the copyright holder of the work, you do not need arXiv's permission to reuse the full text.

## I want to include a paper of mine from arXiv in an institutional repository, do I need permission?

You do not need arXiv's permission to deposit arXiv's version of *your* work into an institutional repository. For all other institutional repository cases, see our help page on institutional repositories.

## Can I harvest the full text of works?

Plase see our bulk data help page, and the API Terms of Use for specific options. Note that the license for the full text is not a part of the current search API schema. The license is, however, provided within arXiv's output from the OAI-PMH in either `arXiv` or `arXivRaw` formats.