A task-driven approach to time scale detection in dynamic networks

Benjamin Fish
University of Illinois at Chicago
1200 W. Harrison St.
Chicago, Illinois 60607
bfish3@uic.edu

Rajmonda S. Caceres
MIT Lincoln Laboratory
244 Wood St.
Lexington, Massachussetts 02421
rajmonda.caceres@ll.mit.edu

ABSTRACT

For any stream of time-stamped edges that form a dynamic network, an important choice is the aggregation granularity that an analyst uses to bin the data. Picking such a windowing of the data is often done by hand, or left up to the technology that is collecting the data. However, the choice can make a big difference in the properties of the dynamic network. Finding a good windowing is the time scale detection problem. In previous work, this problem is often solved with an unsupervised heuristic. As an unsupervised problem, it is difficult to measure how well a given windowing algorithm performs. In addition, we show that there is little correlation between the quality of a windowing across different tasks. Therefore the time scale detection problem should not be handled independently from the rest of the analysis of the network.

Given this, in accordance with standard supervised machine learning practices, we introduce new windowing algorithms that automatically adapt to the task the analyst wants to perform by treating windowing as a hyperparameter for the task, rather than using heuristics. This approach measures the quality of the windowing by how well a given task is accomplished on the resulting network. This also allows us, for the first time, to directly compare different windowing algorithms to each other, by comparing how well the task is accomplished using that windowing algorithm. We compare this approach to previous approaches and several baselines on real data.

1 INTRODUCTION

Data mining on social and other types of networks often either requires information about dynamics or is improved by such information. Incorporating temporal information can improve the efficacy and lead to a more detailed analysis.

As data collection becomes cheaper and easier, the rate at which the data is being collected is often orders of magnitude more frequent than the underlying system. The rate of the data collection process is typically a function of the technology used and not necessarily related to the evolution or dynamics of the network itself. Thus the data collection process makes a choice about the bin size - also called the *resolution*, *aggregation granularity*, or *time scale* of the dynamic network - that may not be the correct choice. Binning the data at coarser resolutions may make it possible to distinguish between the (temporal) orders of interactions that are the result of noise and the orders of interactions that are critical in the network. Indeed, the time scale of the network strongly impacts what structures and dynamics may be observed in the network [4, 11, 14, 19, 21, 22]. Moreover, the choice of time scale impacts the efficacy of data mining on networks [9]. For any data mining task over dynamic networks, choosing the bin size is not only important, but a necessary choice the data scientist must make, and leaving it up to the data collection process is not ideal. This is the problem we take up in this paper, which we will refer to as *windowing* to emphasize the fact that the bin sizes - or windows - may not necessarily all be the same size.

If sufficient knowledge of the data is at hand already, a time scale may be chosen manually to represent natural scales, such as the diurnal or weekly scales common to dynamics among people. However, frequently, getting such domain-knowledge through a data exploration phase or otherwise may be prohibitively difficult or expensive. For these reasons, we seek an automated algorithm for the windowing problem.

In this paper, we give concrete evidence that any windowing algorithm needs to take into account what task the data analyst wants accomplished on the resulting data because there is little correlation between the quality of time scales on each task. The time scale for predicting new links appearing in the network is often different than the time scale for predicting the values of vertex attributes, for example. This may appear obvious, but it implicitly runs contrary to unsupervised and heuristic approaches to choosing an appropriate time scale that have been proposed in previous work. Due to this observation, we need to explicitly take the task into account when windowing.

Thus we set up the windowing problem as a hyperparameter search needed for a supervised machine learning task. For example, for predicting attribute values of vertices, we set aside earlier training data from the network with known attribute values. We then find the right time scale for this labeled data, and apply the time scale to new data to find the corresponding new attribute values. In other words, the windowing algorithm takes as a parameter an algorithm that performs the desired task, such as attribute prediction, and finds the right time scale for that task on the input data. We call this the *task algorithm*. Of course, the downside to this approach is that it requires training data, such as historic attribute values. However, we regard this as both a natural assumption and a necessary assumption: Many popular prediction and classification problems have some notion of ground truth. In addition, since the best time scale depends

on the task anyway, we might as well tailor our time scale detection towards a particular goal.

In this paper, we consider the tasks of link prediction and attribute prediction. We use these two tasks as popular examples of tasks analysts perform on dynamic networks that have some notion of ground truth. It is important to note that the goal of this paper is not to produce state-of-the-art results for each of these tasks. Instead, our aim is to find the best windowing we can for a fixed task algorithm. That is, we treat the task as a black box and we do not take advantage of which task algorithm is used. We do this so that our methods may be used on any algorithm for any task, not just the tasks we offer as examples in this paper.

Our contributions are as follows: We introduce a windowing algorithm that automatically adopts to a given task at hand. This approach treats the windowing problem as an optimization of a hyperparameter for a supervised machine learning task. We motivate this approach by providing evidence that there is little correlation between the quality of time scales on each task. This algorithm can be used for any task that comes equipped with a notion of ground truth, not just the example tasks we consider in this paper. In addition, for the first time, we describe a simple method for directly comparing the performance of windowing algorithms that leverages task-dependency. We compare our approach against several baselines and previous work. We demonstrate that our approach is often superior to other approaches.

1.1 Previous Work

In numeric time series, this problem is often termed 'segmentation,' and segmentation of numeric times series has a long history which is outside the scope of this work; see [12] for an overview.

For dynamic networks, there has been some work related to our problem in the area of change point detection, which seeks to find points in time where the dynamic network has changed abruptly. Typical methods include using generative models of dynamic networks [18] or clustering similar time slices [3]. This literature is marginally different from the problem addressed in this paper because - while finding change points do implicitly segment the dynamic network - the goal of change point detection is not necessarily to find a good representation of the network for the purpose of binning each segment but rather just to find the points when the dynamic network undergoes significant change.

Also related is graph compression, which tries to find a representation of the dynamic network that minimizes the bits needed to store it while simultaneously retaining sufficient information about the network, under the general heading of graph summarization algorithms. See [16] for a survey of graph summarization techniques. This problem is slightly different from ours because we do not necessarily seek a small representation, merely an accurate representation for the task at hand.

More broadly, optimizing hyperparameters has a vast literature encompassing many different techniques and kinds of parameters, such as for classical supervised machine learning [2, 28] or in Bayesian settings [25]. In the context where the hyperparameter is the windowing, for many tasks on dynamic networks, proposed algorithms have included some way to search for a good time scale, e.g. for attribute prediction [20].

However, to the best of our knowledge, such methods have previously been confined to the context of specific tasks or algorithms, and do not necessarily generalize. Meanwhile, previous work on the time scale detection problem in general has focused either on task-independent heuristics or methods that attempt to optimize for a specific metric on graphs. Caceres et al. maps the dynamic network to a time series using a metric on graphs (such as the number of triangles in each graph) and then analyzes that time series [27]. Soundarajan et al. determines a windowing of the data with respect to some metric (such as the exponent of the degree distribution) by measuring when that metric has converged [26]. In our view, these approaches have the downside that they require a specific metric. For a given data set or task to accomplish on a data set, it is not necessarily clear what metric to choose. Darst et al. use a parameter-free approach that seeks to find an appropriate time scale by measuring the similarity between graphs using the Jaccard index [6], similar to an approach used by Krings et al. [14]. Most closely related to the approach that we take in this paper, Fish and Caceres use the quality of the performance of link prediction algorithms to determine the best time scale [9]. These are parameter-free methods or, relatedly, methods that assume that there is some 'ground truth' time scale via a generative model or the like, as in [4, 6, 9]. In this paper, we do not take this tactic because the choice of time scale may be dependent on the task at hand, making the task a de facto parameter for time scale detection.

1.2 Background

Consider a dynamic network over a fixed set of vertices, represented as a stream of time-stamped edges, represented as a sequence of graphs $G_1,\ldots,G_T$. The goal of windowing is to segment this input stream of edges into intervals to form a sequence of graphs $H_1,\ldots,H_m$, each graph representing all of the edges that occurred within an interval. A *window* of size k is a sequence $G_i, G_{i+1},\ldots,G_{i+k-1}$. A *windowing* is a sequence of windows $\{G_1,\ldots,G_{k_1}\}$, $\{G_{k_1+1}, G_{k_1+2},\ldots,G_{k_2}\}$, $\ldots,\{G_{k_{m-1}},\ldots,G_T\}$. In this paper we focus on non-overlapping windows. The resulting sequence is $H_1,\ldots,H_m$, where $H_i = \cup_{j=k_{i-1}+1}^{k_i} G_j$. As a slight abuse of notation, we will refer to both the segmentation of the input graph sequence and the resulting sequence $H_1,\ldots,H_m$ as a windowing.

If all windows have the same size w (except possibly the last window if the length of the sequence does not divide w), we refer to this as a *uniform windowing* and w the *bin size*, *window size*, or *time scale*. A window size of $w = 1$ represents the time scale of the collection process and a window size of $w = T$, where T is the duration of the observed network, means that all temporal information is ignored. The goal of

this paper is to describe and evaluate algorithms for finding a windowing given an input sequence of graphs. Once we have found a windowing $H_1, \ldots, H_m$, it can now be the input for any task that operates over a dynamic network.

In general, we are given the edges of a dynamic network up to some time t as a training set, and performance is evaluated on the dynamic network from time $t+1$ onwards. The windowing algorithm gets ground truth on the training set, e.g. the links that have formed unto time t, and the task algorithm uses only the windowed graph sequence to conduct its analysis, say finding new links in the test set.

2 TASK-DRIVEN WINDOWING ALGORITHMS

We start by describing our proposed windowing algorithm in both offline and online settings. This algorithm automatically adapts to whichever task needs to be accomplished on the dynamic network by treating the task algorithm as a parameter, and so is able to be used for any task and corresponding task algorithm. So that the algorithms we introduce remain useful for not just the tasks we consider, but any task, our algorithms do not attempt to take advantage of the particular nature of the task and corresponding algorithm at hand: we treat the task algorithms as black boxes. However, as we show in Section 8, this approach is still able to perform well despite this self-imposed constraint.

2.1 Offline task-driven windowing

The idea is very simple: since we know what task we want to accomplish on the test set, we use the same task algorithm to learn the window size on the training set. In the offline setting, in order to try to prevent overfitting and to make the search space smaller, we only consider uniform windowings. We then do a simple hyperparameter search: For each window size, up to the length of the training set, window the training set at that size and use that as input for the task algorithm. Measure the performance of the task algorithm (remember we assume we have ground truth for the training set) and use that as the score for the window size. Window the test set with the window size that received the highest score. Of course, this means running the task algorithm $O(T)$ times (where T is the length of the training set), which is not particularly efficient. However, we make the assumption that since this is an offline setting, this blowup in running time in the training phase is not prohibitively large. We will refer to this as the offline *supervised* method, because it uses the training data to predict how well a window size will perform in the future on unseen testing data.

2.2 Online task-driven windowing

In the online setting, we require a windowing of the data seen so far at every time step. We could at each time step perform a similar procedure as in the offline case, leading to $O(i)$ runs of the task algorithm at the ith step, for a total of $O(T^2)$ times, where T is the total length of the sequence. This will often be prohibitively expensive. We introduce an approximate online windowing algorithm to deal with this issue. We illustrate with link prediction, as a natural example of an online task. For details on link prediction, see Section 4.

Algorithm 1 Online windowing for link prediction

Parameters: Integers M and B, Link predictor L
  Initialize $\texttt{scores}_w$ as the empty list
  for each new graph G_i **do**
    Let `new window sizes` include w for $1 \leq w < i$ if length($\texttt{scores}_w$) $< M$
    Let `best window sizes` include the top B window sizes by average($\texttt{scores}_w$)
    for w in `new window sizes`, `best window sizes` **do**
      Let $\mathcal{H}_w = H_1, \ldots, H_{\lceil \frac{i-1}{w} \rceil}$ be the windowing of $G_1, \ldots, G_{i-1}$ at window size w
      `predicted links` $= L(\mathcal{H}_w)$
      Append AUC(new links in G_i, `predicted links`) to $\texttt{scores}_w$
    end for
    $w^* = \arg\max_w$ average($\texttt{scores}_w$)
    Let $\mathcal{H}_{w^*} = H_1, \ldots, H_{\lceil \frac{i}{w^*} \rceil}$ be the windowing of $G_1, \ldots, G_i$ at window size w^*
    return $L(\mathcal{H}_{w^*})$
  end for

Every time we receive a new graph G_i, representing the edges that occurred in the next time step, we can test each window size w by binning the sequence so far at size w and then use the last graph in the windowed sequence to predict the edges that will appear in G_i. We then compare the predicted edges to the actual edges in G_i, producing an AUC score for that window size[1]. The window size w^* chosen next to bin the sequence seen so far including G_i is the window size that maximizes the average of all scores for that window size so far. However, this still means testing $O(i)$ window sizes at each time step, for a total of $O(T^2)$ tests.

To decrease the number of tests, we instead use an approximate version of this where only some of the window sizes are tested, as described in Algorithm 1. This online algorithm is described explicitly for link prediction for ease of reading, but the same algorithm may in principle be used for any online task. Given hyperparameters to the algorithm M and B, we test a window size if it has either been tested fewer than M times, or if the average score so far ranks it amongst the top B window sizes. The intuition behind this approach is that a window size that has been performing badly will not suddenly become the best performing window size, and thus doesn't need to be tested. This requires only $O((M+B) \cdot T)$ total runs of the link prediction algorithm instead of $O(T^2)$, where we think of as M and B as constants. In our experiments, we set $M = B = 10$ (We also demonstrate the effect of changing these hyperparameters in Section 8). We also test a weighted variant of Algorithm 1 (*Weighted Algorithm 1*) by instead using a weighted average

[1]For the sake of computational efficiency, we only score pairs of vertices with non-zero degree, since the score will produce a score of 0 for all other pairs.

of the scores for each window size, in order to privilege scores closer to the present than the past. We use an exponential weighting scheme, where the weight for the score tested on the jth graph where t graphs have been seen already is α^{t-j}. For the purpose of our experiments, we use $\alpha = 1/2$.

We also consider a version of this that stops generating new scores after the training period is over, and sticks with the best window size for the training data for all future time steps (*Training only*).

3 OTHER WINDOWING ALGORITHMS AND BASELINES

We compare against several baselines: the first is always using a 'hand-chosen' value (which we will refer to as the *hand-chosen* algorithm), which represents what we could have done if an expert already had insight into the particular data set. It is important to note that this represents how well we would have done if we didn't need a windowing algorithm at all, and as such should be seen as closer to an upper-bound on performance rather than a lower bound. The second baseline is the random algorithm (*Random*), which chooses a random windowing of the test set. In addition, for attribute prediction, we also consider the windowing that removes all temporal information, i.e. the window size that is always the length of the test sequence (*No time*). The final baseline we consider is slightly more sophisticated: Given a time series that assigns a real value to every graph in the sequence, we may compute its discrete Fourier transform (DFT). For frequency f, denote by x_f the amplitude of that frequency. The score we assign w is the maximum magnitude $|x_f|$ of any frequency in the transform such that f rounds to w. We then choose the window size with the maximum score (*Fourier*). In this paper, we consider the DFT under the commonly-used Hanning window where the metric is the number of edges in each graph. This serves as a proxy for the amount of activity at any given time. The DFT of this particular time series has been used before, e.g. to study Reality Mining [8].

We compare against *ADAGE*, the method of Soundarajan et al. [26]. Like them, we use the exponent of the degree distribution as the metric. We also compare against the Jaccard-index-based method (*Jaccard*) of Darst et al. [6] and the entropy-based method (*Entropy*) of De Domenico et al. [7]. Since this last method is really a graph compression algorithm and allows for graphs with any types of layers, we treat each time step as a layer and modify their method to only allow adjacent time steps to be merged.

All of these algorithms can be used in the offline setting, but only ADAGE, the hand-chosen baseline, and the random baseline can be used in the online setting.

4 TASKS

Given a candidate windowing, we evaluate its quality by performing a task algorithm on that windowing. We then evaluate the windowing by the performance of the task algorithm when using that windowing. We consider two tasks: link prediction and attribute prediction. We treat link prediction as an online task and attribute prediction as an offline task. We do this to demonstrate windowing algorithms on both kinds of tasks. In what follows, we describe the algorithms we use for each task and how we judge their performances.

4.1 Link prediction

In link prediction, the goal is to predict the edges that are most likely to appear in the future. In the online setting, at every time step, our goal is to predict the edges that will appear in the next time step (the next step in the initial input sequence before windowing).

While there are many methods for link prediction (see [1] for a survey), one of the most common is a simple scoring function that scores every pair of vertices by how likely an edge is to appear between them. In this paper, we use the Katz_β score, an efficient and well-performing score [15]. β is a damping parameter that weights shorter paths exponentially higher than longer paths in the most recent graph in the input sequence. For our experiment results, we use $\beta = 0.005$, which has been used before [9, 15]. We also use the simple common neighbor score [15] to compare our results across multiple algorithms for the same task.

The performance of the link prediction algorithm is evaluated using the AUC of the precision-recall curve, as recommended by [29], averaged over all predictions made, one set of predictions for each graph.

4.2 Attribute prediction

We use the Time Varying Relational Classifier (TVRC) algorithm of Sharan and Neville [24] to determine the unknown value of a binary vertex attribute (that doesn't change over time). The goal is then to infer the missing attribute values using a Bayesian model that takes advantage of temporal information. In our implementation, we use add-one smoothing for categorical features and a Gaussian distribution for continuous features. The goal is then to find a windowing where TVRC builds the best performing model.

We use a form of leave-one-out testing to measure the performance of TVRC. In this setting, the target attribute of one of the vertices is removed from both training and testing, a model is trained with the training set, and gives a prediction for the value of the missing attribute using the test set. To make the problem harder and more realistic, instead of just removing one target attribute, we remove a whole batch of them at once, and use the trained model to predict the values of all of their target attributes. The vertex set is partitioned into batches using a batch size parameter b, and this is repeated for each batch. Once a prediction has been made for all batches, we measure the performance or TVRC as the standard AUC of the ROC.

5 TASK DEPENDENCE

Ideally, it would be nice to have just one windowing that performs well regardless of whether your goal is link prediction, attribute prediction, or any other task, which would mean we would not have to take the task into account. However, we

Table 1: Each row gives the performances of the corresponding task algorithm (as described in Section 4). Each column corresponds to the data being aggregated at the window size that maximizes the performance for that column's task algorithm. For example, the first row and second column is how well the Katz algorithm performed, if the data was aggregated at the window size that maximized the performance of TVRC, given all knowledge of the attribute values. Scores are averaged over the different intervals of the data.

	Enron		Reality Mining		Badge	
	Link prediction	Attr. prediction	Link prediction	Attr. prediction	Link prediction	Attr. prediction
Link pred.	**0.188**	0.599	**0.277**	0.961	**0.438**	0.646
Attr. pred.	0.163	**0.649**	0.220	**0.983**	0.331	**0.740**

Table 2: Spearman correlation coefficients and p-values between link prediction (Katz) and attribute prediction (TVRC).

	Correlation coefficient	p-value
Enron	0.093	0.005
Reality Mining	-0.050	0.493
Badge	-0.733	2.282e-85

demonstrate that this does not appear to be feasible while still maximizing the performance of the task algorithm.

To do this, we score each window size by the performance of each of the task algorithms when the data set is windowed at that size, so we have a score representing the quality of the window size for each of the two tasks. Table 1 shows the performance for those tasks when the data is aggregated to maximize the performance of each of the tasks. For example, choose the best window size for link prediction using Katz by testing all possible window sizes. At that window size, on the same data, attribute prediction does not achieve as high a score as if you had chosen the best window size for attribute prediction. This means that the windowing algorithm should choose a different window size for each of the tasks in order to maximize performance.

This effect is not limited to just the top-scoring window size. More generally, the scores between two tasks do not positively correlate with each other, so that a higher score for a given window size on the first task does not necessarily mean a higher score for that window size on the second task. In order to demonstrate this, we use Spearman's correlation coefficient, which tests the monotonicity between two variables. Table 2 show the correlation coefficients and their associated p-values for each pair of tasks: they are all close to zero or actually negative. This is further evidence that the quality of window sizes depend on the task, and hence that we should use the task as supervision for windowing.

6 DATA SETS

We use five data sets: Enron, MIT Reality Mining, Badge, Hypertext09, and Haggle. We treat all of these as undirected dynamic networks. For both convenience and uniformity, we bin each of these data sets at a initial window size at a 'natural' size, i.e. a choice that a data analyst might make, such as an hour or a day. This is so that a window size of $w = 1$ is a baseline representing how well a hand-chosen windowing would perform. We only consider window sizes at least as large as this baseline.

Each of these are suitable for link prediction. Of these, Enron, Reality Mining, and Badge are equipped with vertex attributes in order to test attribute prediction.

Enron is an email network between employees of Enron Inc. from January 1999 to July 2002, during the period of Enron's market manipulation scandal and subsequent collapse [13]. We use the number of occurrences of the top fifty words used in all emails in each employee's outgoing emails (a list of stop words were excluded from the top fifty words) as attributes. For attribute learning, we determine whether the employee was a manager or not, which we took from [5]. The initial bin size is one day.

Reality Mining is a proximity network of 90 MIT students and faculty using data taken from cell phones from September 2004 to May 2005 [8]. We use as edges both phone calls between participants and whenever two participants are close to each other. Each of the participants filled out a survey about their cell phone usage, which we use as the categorical attributes for each vertex. The attribute we use for testing attribute prediction is whether they are part of the business school or the MIT Media Lab. The initial bin size is one day.

Badge is a proximity network of 23 employees at a data server configuration firm for a month [17]. Each edge represents when two employees are in close proximity to each other, representing an interaction. Each employee was assigned a certain number of tasks, and data about these tasks was recorded, e.g. average completion time, whether they took on a difficult task or not, etc. For attribute prediction, we predict whether or not they made an error in one of their tasks. The initial bin size is one hour.

Haggle INFOCOM is a proximity network consisting of interactions, recorded using Bluetooth, among attendees at an IEEE INFOCOM conference over four days [23]. 41 attendees participated in this network. The initial bin size is 10 minutes.

Hypertext09 is another proximity network of attendees at the ACM Hypertext 2009 conference, held over three days [10]. Each vertex is one of the 113 attendees, and an edge represents a interaction between two attendees that

Table 3: Performance (PR-AUC) of each of the algorithms on five data sets with respect to link prediction using Katz.

	Enron	Reality Mining	Badge	Hypertext	Haggle
Random	0.099	0.205	0.208	0.049	0.215
Hand-picked	0.148	**0.266**	**0.619**	**0.146**	**0.485**
Weighted Algorithm 1	**0.186**	0.256	**0.521**	**0.122**	0.471
Algorithm 1	0.183	**0.266**	0.472	0.103	**0.474**
Training only	0.159	0.240	0.418	0.065	0.437
ADAGE	0.149	0.198	0.394	0.044	0.298

Table 4: Performance (PR-AUC) of each of the algorithms on five data sets with respect to link prediction using common neighbors.

	Enron	Reality Mining	Badge	Hypertext	Haggle
Random	0.075	0.181	0.167	0.030	0.138
Hand-picked	0.039	0.183	**0.346**	0.037	0.230
Weighted Algorithm 1	**0.106**	**0.210**	**0.325**	**0.039**	**0.233**
Algorithm 1	0.104	0.209	0.302	0.037	0.196
Training only	0.097	0.191	0.294	0.027	0.172
ADAGE	0.094	0.179	0.266	0.031	0.152

was active for at least 20 seconds. The initial bin size is 10 minutes.

7 EXPERIMENTAL SETUP

Before we can actually compare our windowing algorithm to others, we must describe how we test the performance of windowing algorithms. In the offline setting, we set aside a previous interval of the dynamic network for training, and the next interval for testing. The training interval includes ground-truth information. For example for attribute prediction, it includes all known attribute values on vertices included in the training interval. A windowing algorithm may use this information to decide on a window size to use in the test set. A windowing algorithm, uniform or otherwise, is also allowed to see the edges in the test set to determine the windowing for the test set, but of course no ground truth information about the task.

Once the test set is windowed, we perform our task on the windowed data and measure its performance, as described in Section 4. The windowing algorithm's score is then just the score that the task algorithm received. For each data set, we split it up into six consecutive intervals, and then do training and testing on consecutive intervals, where the previous test set becomes the next training set, so there are five total tests. We do this in order to promote generalizability of our results. For attribute prediction, we use pooling: we take the AUC as described in Section 4 over all vertices in all test sets, instead of averaging the individual AUC's of each test set, because the population, i.e. the vertices, is the same in each test set.

To perform attribute prediction, TVRC requires training data to build a model. We therefore need to make sure to decouple the training data for the model and the training data used to find the best window size. To do this, we split the training data into two, use the first half as the training data for the TVRC model, and the second half of the training data to test out how well the model does when windowed at each window size. We perform this test by taking the vertices that still have the value of the target attribute and using the same process as when we use the model on test data: remove the remaining attribute values in batches, build the model at each window size, and then test which values TVRC predicts on the second half of the training data. As described above, we then use the AUC as the quality of that window size.

In the online setting, a new graph from the initial input sequence is given to the windowing algorithm, and the windowing algorithm must make a decision as to how to incorporate the new graph into the windowing so far. After windowing, a prediction is made, and then the process is repeated. Since the link prediction algorithms we use only ranks the likelihood of an edge appearing rather than determining the number of new links, we only do this process for those time steps where at least one new edge appears. The score for the windowing algorithm is the average over all scores received for each prediction. As in the offline setting, we split each data set into six consecutive intervals and then do training and testing on each pair of intervals, where the testing phase is online.

8 RESULTS AND DISCUSSION

Tables 3, 4, and 5 show our results on link prediction and attribute prediction.

Table 5: Performance (AUC) of each of the algorithms on three data sets with respect to attribute prediction.

	Enron	Reality Mining	Badge
Random	0.566	0.966	0.583
Hand-picked	0.560	0.960	0.646
Supervised	**0.587**	**0.974**	**0.656**
No time	0.584	0.971	0.568
Fourier	0.567	0.967	0.568
Jaccard	0.576	0.973	0.571
Entropy	0.555	0.970	0.562
ADAGE	0.564	0.973	0.572

The supervised approach does do well, but there are certainly caveats. In attribute prediction, the supervised approach is the best performing algorithm, although the absolute difference over the others is sometimes rather small. On the other hand for link prediction, the 'hand-picked' window size is often the best performer when using the Katz score, probably due to the fact that these are well-studied data sets with natural periodicities dictated by human activity, and those window sizes reflect that. Nonetheless, this is not universal: using the common neighbor score, our supervised method almost always outperformed the hand-picked window size. In addition, in general, we will want to have windowing algorithms that do not rely on an analyst's knowledge of the data set: acquiring domain-specific knowledge like this can be time-consuming, expensive, or difficult. Outside of this approach, at least one of the supervised methods does the best for all of the data sets. The weighted version is the overall winner, with a caveat: in a few cases the unweighted version outperforms the weighted version, perhaps because the weighted version may be overfitting to more recent data that is not representative of future data. To what degree this is the case, however, we leave for future work.

We also test the effect of the hyperparameters B and M, shown in Figure 1. As would be expected, by increasing B or M (which increases the number of window sizes tested) performance is generally improved. This effect, however, is extremely weak on the Reality Mining data set. That and our ability to perform relatively well even for small values of B and M validates our choice to use small constant values of B and M to use for Algorithm 1 and forego any attempt to optimize their values in the course of the windowing algorithm.

Our approach also reveals other differences between the tasks. Figure 2 shows the quality of every window size on each of the first four intervals of Reality Mining (we don't use the last interval because each interval needs a subsequent interval for testing attribute prediction). TVRC is more stable under changes to window size compared with link prediction. Such sensitivity makes it much more difficult to speed up the process by testing fewer window sizes. Indeed, our results do not appear as good on link prediction, where we don't test all window sizes. We leave for future work determining if there is a way to test a fewer number of window sizes.

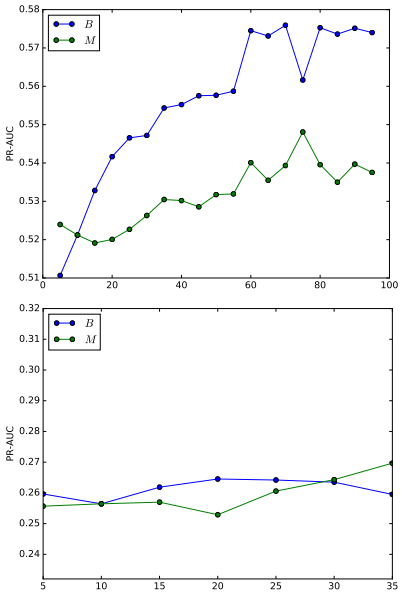

Figure 1: The effect of the hyperparameters B and M on quality for link prediction (Katz), where we fix B=10 and vary M, and fix $M = 10$ and vary B. The figure on the top is the results on the Badge data set, while the bottom is the results on the Reality Mining dataset.

9 CONCLUSION AND FUTURE WORK

In this paper, we have provided a simple and easy-to-use framework for directly comparing the quality of windowing algorithms and moreover, introduced windowing algorithms that leverage our ability to test a windowing in order to do a hyperparameter search. Nonetheless, we leave for future work several challenges: Like any supervised machine learning, the quality of the learner depends on the quantity and quality of training data. Improving windowing algorithms in the face of environments with little training data remains an issue. Even with training data, this can be a computationally-expensive procedure if the windowing algorithm has to repeatedly invoke an expensive task algorithm. We leave for future work finding heuristics that approximate well how a windowing will perform for a given task.

10 ACKNOWLEDGEMENTS

We would like to thank our anonymous reviewers for helpful feedback. This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the

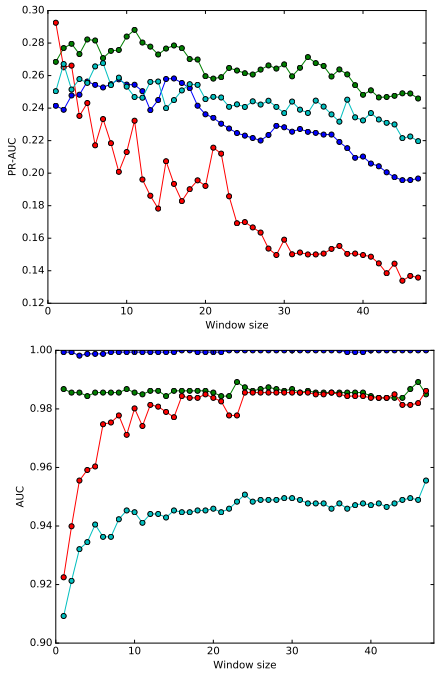

Figure 2: Performance for link prediction (top figure) and attribute prediction (bottom figure) as a function of window size. Each colored curve represents a different time interval along the Reality Mining timeline.

author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

REFERENCES

[1] Mohammad Al Hasan and Mohammed J. Zaki. 2011. A survey of link prediction in social networks. In *Social Network Data Analytics*. Springer, 243–275.

[2] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (2012), 281–305.

[3] Michele Berlingerio, Michele Coscia, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. 2010. As time goes by: Discovering eras in evolving social networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 81–90.

[4] Aaron Clauset and Nathan Eagle. 2007. Persistence and periodicity in a dynamic proximity network. *DIMACS Workshop on Computational Methods for Dynamic Interaction Networks* (2007).

[5] Germán Creamer, Ryan Rowe, Shlomo Hershkop, and Salvatore J Stolfo. 2009. Segmentation and automated social hierarchy detection through email network analysis. In *Advances in web mining and web usage analysis*. Springer, 40–58.

[6] Richard K. Darst, Clara Granell, Alex Arenas, Sergio Gómez, Jari Saramäki, and Santo Fortunato. 2016. Detection of timescales in evolving complex systems. *Scientific Reports* 6 (2016), 39713.

[7] Manlio De Domenico, Vincenzo Nicosia, Alexandre Arenas, and Vito Latora. 2015. Structural reducibility of multilayer networks. *Nature communications* 6 (2015).

[8] Nathan Eagle and Alex Sandy Pentland. 2006. Reality mining: sensing complex social systems. *Personal and ubiquitous computing* 10, 4 (2006), 255–268.

[9] Benjamin Fish and Rajmonda S Caceres. 2015. Handling oversampling in dynamic networks using link prediction. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 671–686.

[10] Lorenzo Isella, Juliette Stehlé, Alain Barrat, Ciro Cattuto, Jean-François Pinton, and Wouter Van den Broeck. 2011. What's in a crowd? Analysis of face-to-face behavioral networks. *Journal of theoretical biology* 271, 1 (2011), 166–180.

[11] Márton Karsai, Nicola Perra, and Alessandro Vespignani. 2014. Time varying networks and the weakness of strong ties. *Scientific Reports* 4 (2014), 4001.

[12] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. 2004. Segmenting time series: A survey and novel approach. *Data mining in time series databases* 57 (2004), 1–22.

[13] Bryan Klimt and Yiming Yang. 2004. The enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*. Springer, 217–226.

[14] Gautier Krings, Márton Karsai, Sebastian Bernhardsson, Vincent D Blondel, and Jari Saramäki. 2012. Effects of time window size and placement on the structure of an aggregated communication network. *EPJ Data Science* 1, 1 (2012), 4.

[15] David Liben-Nowell and Jon Kleinberg. 2003. The Link Prediction Problem for Social Networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM '03)*. ACM, New York, NY, USA, 556–559.

[16] Yike Liu, Abhilash Dighe, Tara Safavi, and Danai Koutra. 2016. A Graph Summarization: A Survey. *arXiv:1612.04883* (2016).

[17] Daniel Olguín Olguín, Benjamin N Waber, Taemie Kim, Akshay Mohan, Koji Ara, and Alex Pentland. 2009. Sensible organizations: Technology and methodology for automatically measuring organizational behavior. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39, 1 (2009), 43–55.

[18] Leto Peel and Aaron Clauset. 2015. Detecting Change Points in the Large-Scale Structure of Evolving Networks. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[19] Bruno Ribeiro, Nicola Perra, and Andrea Baronchelli. 2013. Quantifying the effect of temporal resolution on time-varying networks. *Scientific Reports* 3 (2013), 3006.

[20] Ryan A. Rossi and Jennifer Neville. 2012. Time-Evolving Relational Classification and Ensemble Methods. In *PAKDD 2012, Kuala Lumpur, Malaysia, May 29-June 1, 2012, Proceedings, Part I*. 1–13.

[21] Jari Saramäki and Esteban Moro. 2015. From seconds to months: an overview of multi-scale dynamics of mobile telephone calls. *European Physical Journal B* 88 (2015), 164.

[22] Ingo Scholtes, Nicolas Wider, and Antonios Garas. 2016. Higher-order aggregate networks in the analysis of temporal networks: path structures and centralities. *European Physical Journal B* 89 (2016), 1–15.

[23] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau. 2009. CRAWDAD dataset cambridge/haggle (v. 2009-05-29). Downloaded from http://crawdad.org/cambridge/haggle/20090529. (May 2009).

[24] Umang Sharan and Jennifer Neville. 2008. Temporal-relational classifiers for prediction in evolving domains. In *Eighth IEEE International Conference on Data Mining, 2008*. IEEE, 540–549.

[25] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *26th Annual Conference on Neural Information Processing Systems 2012*. 2960–2968.

[26] Sucheta Soundarajan, Acar Tamersoy, Elias B Khalil, Tina Eliassi-Rad, Duen Horng Chau, Brian Gallagher, and Kevin Roundy. 2016. Generating Graph Snapshots from Streaming Edge Data. In *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 109–110.

[27] Rajmonda Sulo, Tanya Berger-Wolf, and Robert Grossman. 2010. Meaningful selection of temporal resolution for dynamic networks. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. ACM, 127–136.

[28] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*. 847–855.

[29] Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. 2015. Evaluating link prediction methods. *Knowledge and Information Systems* 45, 3 (2015), 751–782.