PAPER MANUSCRIPT SUBMITTED TO THE COMPUTER JOURNAL

(Final Version)

Paper Title: **A Real-Time Self-Adaptive Classifier for Identifying Suspicious Bidders in Online Auctions** [1]

Authors:

Benjamin J. Ford, Graduate Student
Computer and Information Science Department
University of Massachusetts Dartmouth
Email: u_bford@umassd.edu

Dr. Haiping Xu, Associate Professor
Computer and Information Science Department
University of Massachusetts Dartmouth
Email: hxu@umassd.edu

Iren Valova, Professor
Computer and Information Science Department
University of Massachusetts Dartmouth
Email: ivalova@umassd.edu

Corresponding Author:

Dr. Haiping Xu, Associate Professor
Computer and Information Science Department
University of Massachusetts Dartmouth
285 Old Westport Rd.
North Dartmouth, MA 02747, USA

Phone: +1 508 910-6427
Fax: + 1 508 999-9144

Email: hxu@umassd.edu

[1] Paper Manuscript received 21 November 2011; revised 9 January 2012; accepted 21 February 2012

A Real-Time Self-Adaptive Classifier for Identifying Suspicious Bidders in Online Auctions

BENJAMIN J. FORD, HAIPING XU* AND IREN VALOVA

Computer and Information Science Department

University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA

**Corresponding author: hxu@umassd.edu*

Received 21 November 2011; revised 9 January 2012; accepted 21 February 2012

Abstract

With the significant increase of available item listings in popular online auction houses nowadays, it becomes nearly impossible to manually investigate the large amount of auctions and bidders for shill bidding activities, which are a major type of auction fraud in online auctions. Automated mechanisms such as data mining techniques were proven to be necessary to process this type of increasing workload. In this paper, we first present a framework of Real-Time Self-Adaptive Classifier (RT-SAC) for identifying suspicious bidders in online auctions using an incremental neural network approach. Then we introduce a clustering module that characterizes bidder behaviors in measurable attributes and uses a hierarchical clustering mechanism to create training datasets. The neural network in RT-SAC is initialized with the training datasets, which consist of labeled historical auction data. Once initialized, the network can be trained incrementally to gradually adapt to new bidding data in real-time, and thus, it supports efficient detection of suspicious bidders in online auctions. Finally, we utilize a case study to demonstrate how parameters in RT-SAC can be tuned for optimal operations and how our approach can be used to effectively identify suspicious online bidders in real-time.

Keywords: Self-adaptive classifier; incremental neural network; e-commerce; online auctions; auction fraud detection.

1. INTRODUCTION

Online auction houses, such as eBay, have recently experienced a dramatic increase in their popularity. From December 1, 2009 to July 1, 2010, the number of listings hosted by eBay has increased from approximately 37.2 million to a staggering 110 million [1]. However, this significant increase in popularity also means more chances for online auction fraud, such as shill bidding, bid

shading, and user collusion, to be conducted [2-4]. Some online auction houses (e.g., eBay) have recently instituted a policy where bidders' identities are obfuscated from other users. Such a policy makes things even worse since it becomes more difficult for users to carry out their own investigation on suspicious bidders. To protect online business from auction fraud, there is a pressing need to introduce effective mechanisms for detecting online auction fraud in real-time. Due to a large amount of auction data that needs to be processed for detection of auction fraud, automated mechanisms, such as data mining techniques, have become more important than ever [5-7].

Shill bidding is a type of auction fraud which refers to the practice of sellers using a "fake" bidder account or asking another bidder to place bids on their auctions for the purpose of raising the final auction prices. This fraudulent activity causes honest bidders to pay significantly more for winning an auction. In addition to this undesirable effect, Dong *et al.* discussed the potential market failure that could result from excessive shill bidding [2]. In a faceless environment, such as the Internet, a seller can easily create a secondary account or participate in an organized group dedicated to shill bidding. Unlike blatantly obvious forms of auction fraud, such as non-delivery fraud, shill bidding typically goes undetected by those victimized, especially by those who do not know how to recognize the subtle signs of shill bidding. Existing approaches, such as Dempster-Shafer (D-S) theory based shill verification, are capable of verifying shill bidding behavior using collected evidence [8]. However, since such approaches require collecting a large amount of evidence in addition to the auction bidding history, they are not efficient for analyzing a large number of bidders in real-time. Therefore, it is important to define automated mechanisms that can efficiently detect suspicious bidders, in order to narrow down the number of bidders to be verified for shill bidding. By reducing the workload for shill verifiers, our approach supports efficient detection of shill bidders and exposal of fraudulent sellers and colluding users in real-time.

In this paper, we introduce a Real-Time Self-Adaptive Classifier (RT-SAC) to identify suspicious bidders in online auctions. The main component of RT-SAC is a feedforward backpropagation neural network that can be trained incrementally in real-time. When the network receives a set of bidders, it automatically decides which ones are suspicious. Once suspicious bidders are identified, they can be sent to an external shill verifier for further investigation [8]. As time progresses and the popularity of particular items changes, bidding trends are sure to change. If a classifier is not constantly reevaluating itself, it is likely that the classifier will become outdated and unable to correctly classify new data. Therefore, we use the classification and verification results to adapt the network to new bidding data. Since reinitializing a network is a computationally expensive process, to meet the real-time requirements, we adjust the network incrementally by training it only on recent data points each time. By doing so, our classifier can accurately follow changing market trends without decreasing its classification accuracy or throughput.

Before the classifier can function properly, it needs to be initialized using a suitable training dataset. Although many areas of research, such as cancer study [9] and computational biology [10],

benefit from the availability of a vast selection of well-studied scientific data, there are no such existing training datasets for our research due to a lack of study on shilling behaviors. Therefore, we must collect, analyze, and label historical auction data in order to create a training dataset by ourselves. In our approach, we first define a set of attributes, in quantifiable terms, which can be used to describe behavior related to shill bidding. Then, we characterize auction data using these attributes and adopt a hierarchical clustering approach to arrange bidders into numerous groups. Finally, we manually examine each group and label each bidder as either *normal* or *suspicious*. The labeled bidders are saved in a central data pool, which is used to create a training dataset to initialize a supervised neural network and generate test sets for evaluation purpose.

In our recent work, we demonstrated some preliminary results on using data mining approaches for detecting suspicious bidders in online auctions [11]. We developed a training dataset using a hierarchical clustering algorithm and created decision trees to classify online bidders. Our previous work demonstrated that automated techniques such as the data mining approach are feasible for detection of suspicious bidders. However, the error rates of the resulting decision trees were generally over 5% and could be up to 13%, which are not suitable for practical usage. In addition to accuracy concerns, time performance is also an important measure of the usefulness of our approach. Decision trees cannot gradually adapt to new data points; thus, they need to be reconstructed in order to learn new data. This is a very time consuming process. To address the above concerns, in this paper, we adopt neural networks, which can be updated incrementally when learning new examples and also support more accurate classification than the existing decision tree based approach.

The rest of the paper is organized as follows. Section 2 summarizes the related work to our approach. Section 3 presents the overall framework and the dataset creation process, including attribute definitions and bidder labeling. The RT-SAC framework and its various components are discussed in Section 4. To demonstrate the effectiveness of our approach, we present a case study in Section 5. Section 6 concludes the paper and mentions future work.

2. RELATED WORK

Previous work on categorizing groups of bidders using data mining techniques is summarized as follows. Bapna *et al.* utilized *k*-means clustering to generate five distinct groups of bidding behavior in online auctions [12]. They demonstrated how the taxonomy of bidder behavior can be used to enhance the design of some types of information systems. Shah *et al.* analyzed collected auction data from eBay to generate four distinct groups of bidding behavior [13]. The analysis revealed that there were certain bidding behaviors that appeared frequently in online auctions. Hou and Rego used hierarchical clustering to generate four distinct groups of bidding behaviors for standard eBay auctions, namely goal-driven bidders, experiential bidders, playful bidders, and opportunistic bidders [14]. They concluded that online bidders were a heterogeneous group rather than a homogenous one.

Although the above approaches are closely related to our approach, they focus on creating clusters based on the assumption that bidders are honest and have no malicious intentions. On the other hand, Chau *et al.* proposed a 2-level fraud spotting method that could detect fraudulent personalities at user and network levels [5]. The features defined in the user level and network level can be used to establish the initial belief for spotting fraudsters and capture the interactions between different users for detection of auction fraudsters, respectively. Ku *et al.* attempted to detect Internet auction fraud using social network analysis (SNA) and decision trees [6]. They demonstrated that their approach could provide a feasible way of monitoring and protecting buyers from auction fraudsters. Ochaeta *et al.* used a data mining approach for fraud detection based on variables derived from auctioneers' transaction history [7]. The proposed approach employed two learning algorithms, namely C4.5 and Support Vector Machines (SVM), which could provide improved performance for fraud detection. Although the above approaches demonstrated their usefulness in detecting auction fraudsters using data mining techniques, they require analysis of large volumes of offline auction data, thus they do not support detection of auction fraud in real-time. Unlike these methods, our approach aims to design an on-the-fly classifier for detection of shill suspects using real-time auction data, so suspicious bidders with abnormal bidding behaviors can be discovered in a timely fashion.

There has been some recent work on estimating the outcomes of an online auction based on users' behaviors. Gelenbe and Gyorfi developed an automated auction model that can be used to predict the outcomes of users' decision mechanisms [15, 16]. In their approach, the final auction price and the resulting income per unit time are defined as a function of two inputs, namely, the rate at which bidders provide the bids and the time taken by the seller to decide whether to accept a bid. Dong *et al.* employed chi-square test of independence and logistic regression to examine whether the difference between the final auction price and the expected auction price implies shill bidding, where the expected auction price can be predicted based on users' bidding behaviors [17]. Gelenbe and Velan proposed a probabilistic auction model that can be used to study the interaction of bidders and sellers in sequential automated auctions [18]. Such model also allows us to compute the probabilistic outcomes of the auctions and to examine the bidder performance [19]. On the other hand, the topics of detecting and verifying shilling behavior in online auctions based on users' behaviors have attracted much research attention. Kauffman and Wood attempted to detect shilling behaviors in online auctions using a statistical approach and demonstrated how the statistical data of an online market would look when opportunistic behaviors exist [20]. They also defined an empirical model for detecting questionable behaviors. However, their approach requires review of multiple auctions over a long period of time, thus it is not suitable for real-time usage. Xu *et al.* applied a real-time model checking approach to calculating bidders' shilling scores in ongoing online auctions [21]. The approach monitors a bidder's behavior in the auction and updates the bidder's shilling score if any shilling behavior is detected. Although this approach can be used to efficiently detect shilling behavior in online auctions, it assumes predefined shill patterns. Thus, it is inflexible to changes in

bidding trends. In contrast, since our proposed neural network based approach can continuously adapt to new bidding data, it is able to account for changes in bidding trends without a significant loss in accuracy or efficiency. Dong *et al.* proposed a formal approach to verifying shill bidders using D-S theory [8]. The verification approach utilizes additional evidence, such as various bidding histories and statistics regarding bidder and seller interactions, to verify if an online bidder is a shill. The belief of whether a bidder is a shill is calculated using the D-S theory, which allows the verifier to reason under uncertainty. If the belief of a bidder for being a shill exceeds a certain threshold, the bidder is marked as a shill bidder. Goel *et al.* used a multi-state Bayesian network to verify detected shill suspects [22]. Similar to the D-S theory based approach, Bayesian networks are capable of reasoning under uncertainty and can be used to calculate the probability of a bidder being a shill. These techniques, however, suffer from being time consuming in their investigation of bidders. Since most bidders do not behave suspiciously, a verifier that processes every bidder will find that most of its execution time is spent on investigating normal bidders. By detecting suspicious bidders before being verified, our approach can significantly reduce the workload for shill verification. As such, this work is complementary to other research efforts that precisely verify shill bidders using additional evidence [8, 22].

Artificial neural networks (ANN) are used in a wide variety of fields due to their ability to automatically learn the underlying complexities of a dataset, given sufficient training data. Kamo and Dagli presented two models, namely a committee machine with simple generalized regression neural networks (GRNN) experts and a similar committee machine along with a hybrid type gating network that contains fuzzy logic, to forecast stock market [23]. Both models use the same simple candlestick patterns to provide a basis for comparison, and their experimental study show that the performance of the models was satisfactory based on the mean squared error. Järvelin *et al.* used ANN to computationally model word production and its disorders [24]. They developed an algorithm to generate distributed semantic coding from a given semantic tree-structure classification of words, which can account for a variety of performance patterns observed in four Finnish aphasia patients suffering from word-finding difficulties. There has also been previous work to use ANN to analyze online auctions. For example, Wang and Chiu applied a neural network and statistical models to identify instances of artificially inflated feedback scores [25]. Dong *et al.* used the large memory storage and retrieval neural network to predict the final auction prices [26]. However, to the best of our knowledge, there have been no existing efforts on using ANN for detection of shill suspects in real-time. In this paper, we propose an approach to using ANN for detection of shill suspects on-the-fly. Since there are readily available datasets that can be collected from existing online auction houses, such as eBay, ANN has been a natural choice for our research.

In addition to domain flexibility, ANN can adapt to changes in data quickly enough to be used in situations requiring a significant amount of time efficiency. Yin *et al.* developed a model that combined a fuzzy clustering approach and a standard neural network approach to predict urban traffic

flow [27]. Instead of training the neural network on a single large batch of data, they incrementally trained the network as the system received data so that the network could quickly adapt to changing traffic trends. Yang *et al.* used a neural network to control the speed of a complex permanent-magnet motor [28]. They trained the neural network with a large batch of data and used an incremental training approach to adapting the neural network to changing environmental conditions and demands without taking the motor offline. On the other hand, self-adaptive software has also been proposed for efficient operations in real-time environments such as avionics, air traffic control, and medical monitoring systems [29]. For example, Sztipanovits *et al.* showed how to implement self-adaptive system for large-scale applications such as turbine engine testing [30]. They proposed a two layered architecture, which supports efficient computations and reconfiguration management. Hou *et al.* introduced a conceptual framework for agent interaction in intelligent adaptive interface (IAI) design [31]. The framework and its associated IAI models can be used to develop high-performance knowledge-based systems, such as a UAV control station interface. Inspired by the above approaches, in this paper, we introduce a neural network based real-time self-adaptive classifier with an incremental training mechanism that can provide a viable way to detection of suspicious online bidders and can also quickly adapt to changing bidding trends.

3. IDENTIFYING SUSPICIOUS ONLINE BIDDERS

3.1. A Framework for Identifying Suspicious Bidders

A user's bidding behavior in an online auction can be described using a variety of measurable attributes or features. According to statistics we observed for typical datasets of online auctions, many bidders exhibit similar bidding behavior. For example, in a collected dataset of "*Used Playstation 3*" auctions, 57% of bidders only bid once during an auction, 19% of bidders bid twice, and 8% of bidders bid three times. Furthermore, 11% of bidders only bid in the early hours of an auction, whereas 46% of bidders only bid in the final hours of an auction. In addition to measuring these behaviors, we specifically define attributes that describe behaviors related to shill bidding. For example, *bid unmasking* refers to the practice of placing many small bids to uncover the true valuation of the current high bidder [13]. An attribute that measures the average time span between two bids from the same user is useful for identifying bid unmaskers. By using techniques such as hierarchical clustering to organize bidders into groups, bidders can be grouped with those who exhibit similar behavior [11, 32]. Since most bidders exhibit bidding behavior similar to that of others, bidders that do not appear in the large groups deviate from the norm and are possibly suspicious. Further investigation into the characteristics of a group of bidders may reveal whether or not those bidders are shill suspects.

The framework for identifying suspicious online bidders in real-time is illustrated in Figure 1. We first retrieve real auction data from an auction house (e.g., eBay), and store it as historical data for

creating training datasets. The clustering module then parses the bidding history of each auction, and uses the parsed bidding histories to calculate and normalize values for a collection of well-defined attributes related to shill bidding. Each bidder in the dataset has its own values for such attributes (e.g., a bidder's feedback rating). Once all of the bidders are described using these attributes, a hierarchical clustering algorithm is applied to the dataset to create sets of grouped bidders. By looking into the characteristics of each cluster, we manually label the clusters of bidders as either *normal* or *suspicious*. For example, if a cluster's average number of bids in the beginning of an auction is 9, we label this cluster as *suspicious*, since very few bidders place more than 3 bids in the beginning of an auction. On the other hand, if a cluster does not exhibit any suspicious behavior, it is labeled as *normal*. In a case when there is an *outlier*, who is a bidder that exists in its own group that does not exhibit similar behavior to bidders from any other groups, we consider it as a suspicious bidder. When all clusters have been labeled, the cluster labels are applied to their bidders. The resulting labeled bidders constitute a training dataset, which can be used to initialize the RT-SAC.

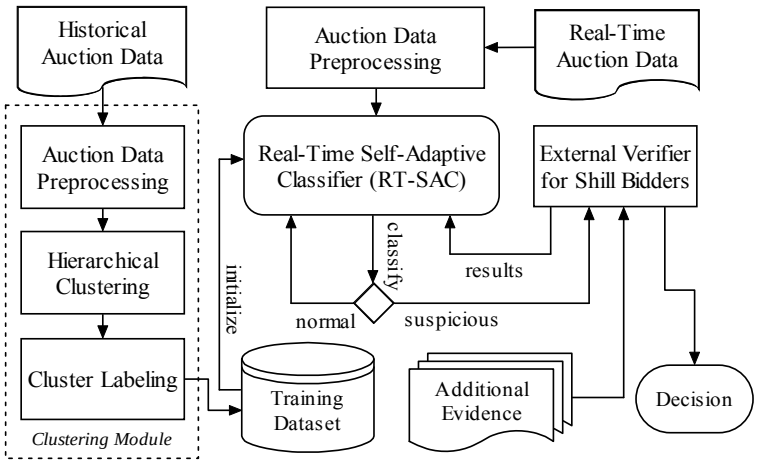

FIGURE 1. A framework for identifying suspicious online bidders in real-time.

After the RT-SAC is initialized, it can be used to classify and adapt to new auction data in real-time. If a bidder from the real-time auction data is classified as *normal*, the classification result is returned to the RT-SAC, and the result can be directly used for incremental training. On the other hand, if a bidder is classified as *suspicious*, it must be sent to an external shill verifier for further investigation. The verifier uses additional evidence to thoroughly inspect each of the suspicious bidders. When the verification procedure completes, the classifier makes a decision as to whether the bidder is a shill, based on the verification results, and then the suspicious bidder is labeled either as *normal* or *suspicious*. More specifically, if a suspicious bidder is verified as a normal one, it should be re-labeled as *normal*; otherwise, if a suspicious bidder is verified as a shill bidder, the label of *suspicious* is not changed. As such, any false positive classifications (i.e., normal bidders classified as suspicious) by the RT-SAC can be corrected by the external verifier. Note that the above scheme does

not handle the presence of false negatives (i.e., suspicious bidders classified as normal) because such handling requires verifying a large number of normal bidders. Although we do not use a verifier to correct any false negatives, we can adjust the classifier in such a way that it could classify more bidders as suspicious and significantly decrease the number of false negatives. For example, if the classifier displays some uncertainty in deciding whether a bidder is normal or suspicious, we can adjust the classifier so that it can always classify such bidders as suspicious. The details about the adjustment process are discussed in Section 4.2. Note that the design of an external shill verifier is beyond the scope of this paper. For more details on shill verification, refer to previous work [8, 22].

3.2. Definitions of Attributes Related to Shill Bidding

In order to identify suspicious bidders, we define a set of attributes that can be measured based on the bidding history of an auction [11]. These attributes effectively describe a bidder's behavior over the course of an entire auction or in a particular auction stage, as well as an auction's potential to be involved with shill bidders. We categorize the attributes into three groups, namely *user attributes*, *stage attributes*, and *auction attributes*. Figure 2 illustrates the organization of these three groups, with the major attributes defined in each group. In the following sections, we give detailed descriptions of the major attributes in each group.

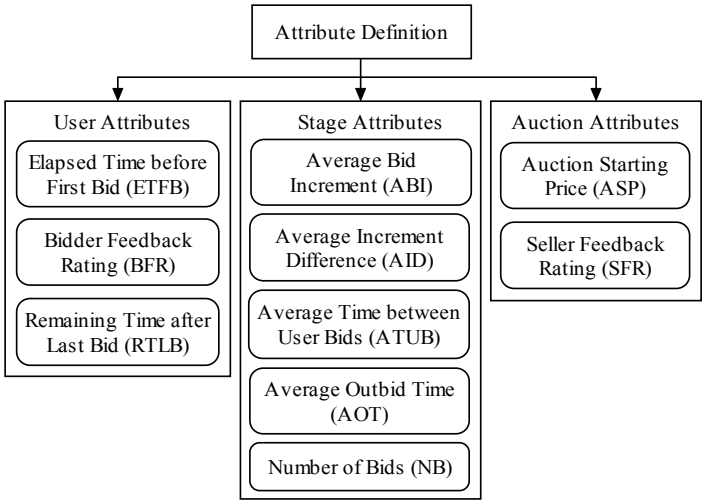

FIGURE 2. Three groups of attributes for identifying suspicious bidders.

3.2.1. User Attributes

User attributes are specific for each bidder, which examine immutable information about a user during an auction. Three examples of user attributes are described as follows.

Elapsed Time before First Bid (ETFB) is the time that elapses from the start of an auction to a bidder's first bid. A large value of *ETFB* indicates that the bidder started participating late in the

auction, whereas a small value of *ETFB* indicates that the bidder participated very early in the auction. Although it is possible for a normal bidder to place bids early in an auction, placing bids extremely close to the start of the auction implies the bidder's possible prior knowledge about the auction. Thus, a bidder with a very small value of *ETFB* is suspicious.

Bidder Feedback Rating (BFR) is useful in describing a bidder's experience level and established trustworthiness [8, 14]. However, we should also notice the potential for fabricating feedback rating through collusion between misbehaving users and fraudulent bidding rings [2, 4]. Thus, a feedback rating should not be considered as a primary factor for describing the trustworthiness of a user.

Remaining Time after Last Bid (RTLB) is the time left in an auction after the bidder places his last bid. A small value of *RTLB* indicates that the bidder has been actively participating in the final stage of the auction, likely with the intent of winning the auction. On the other hand, a large value of *RTLB* indicates that a bidder stopped bidding early before the auction ended. Note that shill bidders typically do not place bids late in an auction to avoid winning the auction [21].

3.2.2. Stage Attributes

The mutable attributes for each bidder are specified as stage attributes. Following the definitions in [21], we divide the auction duration into three stages, namely *early stage*, *middle stage* and *final stage*. The early stage refers to the first quarter of the auction duration; the middle stage refers to [0.25, 0.9] of the auction duration; and the final stage refers to the last 10% of the auction duration. Thus, each stage attribute of a bidder has three values that correspond to the three stages, respectively. Since a bidder can choose to participate in any number of stages, we set the bidder's stage attribute value of a certain stage to 0 if the bidder does not participate in that stage. Five major examples of stage attributes are described as follows.

Average Bid Increment (ABI) refers to the average amount that a bidder outbids the current high bidder during a certain auction stage. For example, if the current high bid is \$30.00 and a bidder places a new bid for \$40.00, the bidder's bidding increment is \$10.00. A very high value of *ABI* indicates a bidder's suspicious bidding behavior. Although a high value may be due to a bidder's significant interest in an item, this is unlikely for auctioned items that are in relatively high supply. Furthermore, a high *ABI* value at the early stage or middle stage is more suspicious than a high *ABI* value at the final stage because most shill bidders would not risk placing a significantly high bid in the final stage that results in a high possibility of winning the auction. The *ABI* value in a certain stage can be calculated as in (1).

$$ABI = \begin{cases} \dfrac{\sum_{i=1}^{n} X_i - Y_i}{n} & if \quad n >= 1 \\ 0 & otherwise \end{cases} \qquad (1)$$

where X_i is the user's new bid, Y_i is the previous bid of X_i, and n is the total number of bids placed by the user in this stage.

Average Increment Difference (AID) is the average difference of a bidder's bidding increments. For example, if a bidder's previous bidding increment is $20.00 and his current bidding increment is $30.00, the increment difference is $10.00. The average increment difference takes the average of the computed differences. The *AID* value in a certain stage can be calculated as in (2).

$$AID = \begin{cases} \dfrac{\sum_{i=1}^{n} (X_i - Y_i) - (X_{i-1} - Y_{i-1})}{n-1} & if \quad n > 1 \\ 0 & otherwise \end{cases} \qquad (2)$$

where X_i is the user's new high bid, Y_i is the previous bid of X_i, $X_0 - Y_0 = 0$, n is the total number of bids placed by the user in this stage. We divide the sum by n-1 because there are n-1 changes of bidding increment for n bids. Note that if n equals 1, *AID* is set to 0 since a change in bidding increment requires at least two bids placed by the user.

A significant increment difference indicates a change in a user's bidding behavior or valuation of the auctioned item. If a *bid fight* is occurring, where bidders are rapidly outbidding each other to be the high bidder, a large increment difference is likely to occur. If the difference is negative, it may indicate that the bidder does not want to raise auction price too much but is willing to increase his valuation slightly to win the item. On the other hand, if the difference is positive, it may indicate that the bidder intends to stop the bid fight by placing a bid for his true valuation of the item, possibly deterring others from rapidly placing bids to be the high bidder.

A substantial positive *AID* in the early or middle stage could indicate efforts to raise the price of the auction, after seeing initial bidder interest. A negative *AID* in the early or middle stage, combined with a number of bids placed close together, may indicate that a suspicious bidder does not want to scare off the currently active bidders and is possibly participating in bid unmasking. However, if the negative increment difference occurs in the final stage, it could indicate a desire of a normal bidder to stop raising the price of the auction too much.

Average Time between User Bids (ATUB) refers to the average time that elapses between two bids placed by the same bidder. Since we try to identify aggressive bidders with this attribute, we define *ATUB* as the inverse of the average elapsed time between bids as in (3).

$$ATUB = \begin{cases} \dfrac{n-1}{\sum_{i=2}^{n} T_i - T_{i-1}} & if \quad n > 1 \\ 0 & otherwise \end{cases} \qquad (3)$$

where T_i is the time of the user's bid, and n is the total number of bids placed by the user in this stage. Note that if n equals 0 or 1, *ATUB* is set to 0 because the calculation of *ATUB* requires at least two bids placed by the user.

A large value of *ATUB* indicates the bidder is actively participating in the auction by placing bids soon after he is outbid. On the other hand, a small value of *ATUB* implies that the bidder is not participating heavily in the auction and is cautious before placing a new bid. A large value of *ATUB* in the early or middle stage indicates possible bid unmasking behavior. For example, the proxy bidding

system used at eBay facilitates the practice of bid unmasking because a new bid is immediately outbid if another user's maximum bid is higher than the new bid. Therefore, a shill bidder can bait a proxy bidding system with small manual bids over the current high bid, and let the auction price quickly climb to other bidders' true valuations [33]. A large value of *ATUB* in the final stage indicates a bidder's strong desire to win the auction, because a bidder participating heavily in the final stage will likely win. In contrast, a shill bidder usually does not have a large value of *ATUB* in the final stage due to the risk of winning the auction.

Average Outbid Time (AOT) is the average time that elapses when a user places a new high bid since another user placed the previous high bid. For example, if a bidder placed a bid 20 seconds after another bidder placed a bid, the outbid time would be 20 seconds. The *AOT* value in a certain stage can be calculated as in (4).

$$AOT = \begin{cases} \dfrac{\sum_{i=1}^{n} T_i - U_i}{n} & if \quad n >= 1 \\ 0 & otherwise \end{cases} \qquad (4)$$

where T_i is the time of the user's bid, U_i is the time of the previous high bid, and n is the total number of bids placed by the user in this stage.

A small value of *AOT* indicates the user's strong interest in the auction and possible participation in a bid fight if n is large enough, whereas a large value of *AOT* typically indicates the user's passing interest in the auction or the bidder is evaluating the status of the auction before placing a new bid. A substantially small value of *AOT* in the early stage may indicate a suspicious bidder's desire to raise the price of the auction and uncover the true valuations of other bidders. On the other hand, a substantially small value of *AOT* along with many bids in the final stage typically indicates a bidder's strong desire to win the auction, as it is very likely that a bidder who participate frequently and respond timely in this auction stage will win.

Number of Bids (NB) refers to the number of bids placed by a bidder in a particular auction stage. A large value of *NB* at the early stage typically indicates a suspicious bidder's desire to raise the auction price quickly. A large value of *NB* at the middle stage might also be suspicious since the bidder might attempt to uncover the true valuations of other bidders; however, it is not considered as a strong indicator because a normal bidder may legally participate in a bid fight in the middle stage. A large value of *NB* at the final stage typically indicates a bidder's strong desire to win the auction. Note that due to the risk of winning the item, shill bidders typically has a very small value of *NB* in the final stage of an auction.

3.2.3. Auction Attributes

Various properties of an auction may influence a bidder's decision to participate in the auction, and particular values of auction attributes may cast more suspicion on the trustworthiness of an auction as well as the likelihood of shill bidding taking place. We now give two examples of auction

attributes as follows.

Auction Starting Price (ASP) may have an impact on a bidder's decision to participate in the auction. A high *ASP* can deter many bidders from participating in the auction, especially if they are bargain hunters. A low *ASP* combined with very early bids may indicate a seller's attempt to avoid the additional reserve price fee. Dong *et al.* discussed the effects of *ASP* on the possibility of an auction involving shills [8]. It was concluded that an auction with a low *ASP* is more likely to involve *reserve price shilling*, where a seller sets a low *ASP* and uses a "fake" bidder ID to place shill bids to raise the price up to an acceptable level in order to avoid paying additional fees associated with a high reserve price. Thus, bidders who participate in auctions with a low *ASP* are more likely to be suspicious.

Seller Feedback Rating (SFR) is another important factor that influences bidding behaviors in an auction since most bidders are more likely to bid in an auction if the seller has significant positive feedback [34-35]. However, as described by other researchers, *SFR* can also be fabricated through collusive users and fraudulent bidding rings, and thus cannot be entirely trusted to be accurate [2, 4].

3.3. Generation of Training Dataset

We adopt hierarchical clustering techniques to cluster collected data points. Hierarchical clustering is known to have significant advantages over flat clustering techniques such as *k*-means [32]. Flat clustering algorithms require the number of clusters to be defined prior to execution, which may significantly affect the clustering results. However, determining the proper number of clusters is not a trivial and arbitrary task. Flat clustering algorithms may also lead to nondeterministic results. For example, given an input, the algorithm generates many different sets of results, but it is impossible to know whether the set of results is complete or if the optimal result has been generated. Thus, justifying a set of results generated using a flat algorithm is very difficult. Although hierarchical clustering algorithms are quadratic in time complexity, this does not matter since the cluster analysis is performed offline and not constrained by time.

As shown in Algorithm 1, an important aspect of hierarchical clustering is the similarity measure, which determines when elements shall be added into a cluster. At a given point in a clustering process, two clusters deemed to be the most similar are the ones to be combined into a single cluster. As shown in (5), the similarity between two clusters is determined by the similarity between the centers, called *centroids*, of the two clusters. The centroid of a cluster is equivalent to the vector average of the cluster's members, as defined in (6).

$$SIM(C_a, C_b) = \vec{x} \bullet \vec{y} = \sum_{i=1}^{n} x_i * y_i \qquad (5)$$

$$\vec{x} = \frac{1}{N_a} \sum_{i=1}^{N_a} \vec{a}_i \qquad \vec{y} = \frac{1}{N_b} \sum_{i=1}^{N_b} \vec{b}_i \qquad (6)$$

where $\vec{x}$ and $\vec{y}$ refer to the vector average of the members in cluster C_a and C_b, respectively; N_a and N_b are sizes of the cluster C_a and C_b, respectively.

Algorithm 1: Cluster Generation

Input: A set of data points and a predefined minimal similarity
Output: a set of clusters that meet the minimal similarity requirement

```
1. GenerateClusters (DataSet dPoints, ClusterSet clusters, double minSimilarity)
2.    if size (clusters) == 0  // initially, there are zero clusters
3.       for each element e in dPoints
4.          create a new cluster c for e and add c into clusters
5.       return GenerateClusters (dPoints, clusters, minSimilarity)
6.    else if size (clusters) == 1  // there is only one cluster
7.       return clusters
8.    else  // there are at least two clusters in set clusters
9.    initialize maxSimilarity to 0
10.      initialize mergeClusters to false
11.      for each pair of clusters c1 and c2 in clusters
12.         calculate the similarity between c1 and c2
13.      if similarity > maxSimilarity && similarity ≥ minSimilarity
14.         maxSimilarity = similarity
15.         set mergeClusters to true
16.      if mergeClusters == true
17.         merge c1 and c2 into a new cluster c3
18.         replace c1 and c2 by c3 in clusters
19.      return GenerateClusters (dPoints, clusters, minSimilarity)
20.      else  // no more clusters can be merged
21.         return clusters
```

Clustering using centroids is known to be not as heavily affected by outliers as single-link or complete-link similarity measures [32]. Since bidding behavior can vary significantly, any auction dataset will contain outliers. Thus, it is important to adopt a clustering approach that can perform effectively with the presence of outliers. In addition to the similarity measure, it is also important to specify a minimum similarity cutoff. The cutoff value determines when the clustering process should terminate. If two clusters' similarity value does not exceed this cutoff value, they are not combined. Note that if the similarity cutoff is not specified, the clustering algorithm eventually outputs a single cluster containing all of the elements.

To fully utilize the available domain knowledge, we make use of weighted attribute values. We noticed that certain attributes should be more important in determining cluster membership than others. For instance, bidders with low feedback rating might be simply new users, who are not necessary suspicious; while bidders who place a lot of bids in an early auction stage are more likely to be suspicious bidders. By weighting the normalized values, our clustering method provides a more accurate solution to clustering bidders in terms of suspicious bidding behaviors. Once the auction data has been analyzed, normalized, and weighted, it is passed as a parameter *dPoints* to the cluster generation algorithm for hierarchical clustering (as defined recursively in Algorithm 1). Note that the cluster set *clusters* initially contains no clusters, but as a starting point for the clustering process, the algorithm creates a set of clusters in which each cluster consists of a single data point. The basic idea of the hierarchical clustering approach is to repeatedly merge the two closest clusters until there is only one cluster left or the similarity measures of all pairs of clusters become less than a predefined

minimum similarity *minSimilarity*.

The set of clusters generated using the cluster generation algorithm are merely numbered without any semantic meaning associated with membership in a particular cluster. In order to give each cluster and its members a semantic meaning, we need to properly label the clusters. This is done by manually inspecting each cluster and assigning a label (i.e., *normal* or *suspicious*) based on its most prevalent features. In Section 5 (case study), we give examples to show how such labeling can be done.

4. NEURAL NETWORK BASED RT-SAC

4.1. Architectural Design of RT-SAC

The architectural design of RT-SAC is illustrated in Figure 3. When real-time auction data comes in, the attributes related to shill bidding are preprocessed. The preprocessed real-time auction data is then sent to two modules, namely the *Adaptive Neural Network* (version *k*) and the *Outlier Detection* module for concurrent processing. The reason for outlier detection is that outliers typically have significant negative effects on a classifier's training performance [36-37]. By removing outliers, the performance of the classifier can be improved. During the outlier detection process, each bidder is compared with the groups of normal bidders and suspicious bidders from the data pool. If a bidder is detected as an outlier, it is marked as such. Note that the outlier detection process is supported by reasoning Prolog rules stored in the *Knowledge Base* using a Prolog engine.

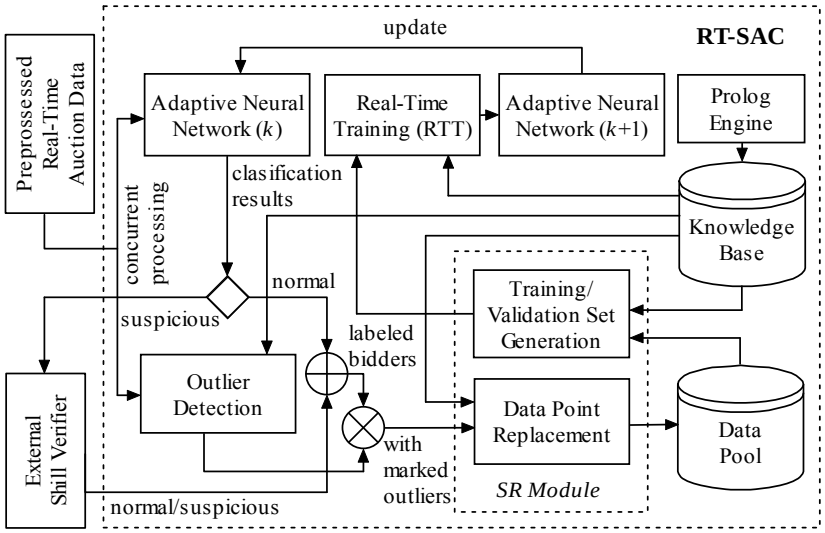

FIGURE 3. Architectural design of RT-SAC.

While the classifier checks for outliers, it concurrently classifies new bidders using the *Adaptive Neural Network* (version *k*). If a bidder is classified as *normal*, it is labeled as such. Otherwise, if the bidder is classified as *suspicious*, it is sent to an *External Shill Verifier* (e.g., a D-S theory based shill verifier [8]) for further investigation. Depending on the verification results, the suspicious bidder will

be labeled as either *normal* or *suspicious*. All labeled bidders are combined with the outlier detection results, so outliers among the labeled bidders are properly marked. The *Selection and Replacement (SR)* module then uses the labeled bidders with marked outliers to update the data pool according to predefined Prolog rules. After replacing data points in the data pool, the SR module uses the most recent data points that are not marked as outliers from the current data pool to create the training and validation set. The generated training and validation set is then sent to the *Real-Time Training (RTT)* module for incremental training. After the training is completed, a new *Adaptive Neural Network* (version k+1) is created, which is used to update the existing network (version k) for the next classification and incremental training phase.

The most recent data points are selected by the SR module using the *window* concept. As shown in Figure 4, when the window size is defined as 9, the SR module uses the latest 9 bidders (excluding those marked as outliers) to create a training set and a validation set. The 9 bidders include 3 new bidders recently processed by RT-SAC and 6 other recent data points from the data pool. The *window speed* refers to how many bidders RT-SAC processes before it adapts the network to the current window. In this example, since RT-SAC processes 3 bidders at a time, the window speed is 3. When new bidders are added into the data pool, the oldest bidders in the current window are pushed out of the window to make room for new bidders. Although the SR module should not use any bidders marked as outliers when creating the training set and validation set, outliers are still kept in the data pool because they are useful as test data to evaluate the performance of the network.

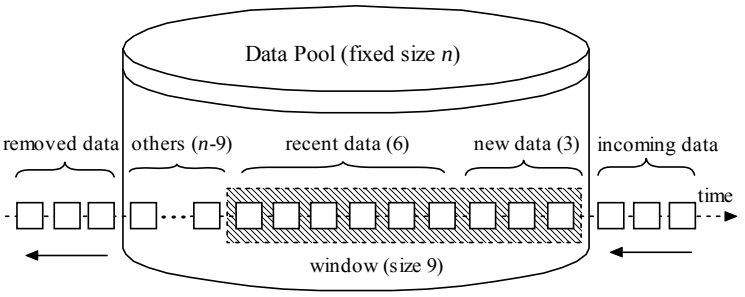

FIGURE 4. Illustration of data selection with a window size of 9

Note that when the window size is 9 and the window speed is 3, the 3 newest bidders will be used for training or validation three times before they are phased out of the window. This reintroduction of data to the network causes the network to adapt to new data in a smoother and more predictable manner than if the network was trained on a dataset only once. However, it is important not to set the window's moving speed too fast or too slow. A window speed that is too slow may cause *overfitting*, since bidders may be reused for training too many times. On the other hand, a speed that is too fast may cause *underfitting* because the network will not have sufficient time to learn the bidding behaviors of new bidders from the window.

4.2. A Neural Network for Classification of Bidders

One of the major components of RT-SAC is a neural network, which is used to classify participating bidders of a recently completed auction. Different from other classification approaches such as decision trees [11], neural networks are typically a black-box approach that uses a set of nodes, called *neurons*, to process an input signal and produce an appropriate output signal. A neuron in the network can have multiple input and output links to send and receive signals, respectively. A neural network processes its input by sending the input signals through those connecting input and output links. If a particular neuron's input exceeds a threshold, the neuron "fires" and sends a signal through its output links; otherwise, the neuron drops the signal. A feed-forward neural network transmits signals in one direction, from the input layer to the output layer. Each link in a neural network is associated with a weight that determines the relative importance of signals being sent. In addition, each neuron is associated with a bias term that determines the relative importance of the neuron for classifying data. For example, if a neuron has a higher bias, it "fires" more often than a node with a lower bias.

Basic neural networks, such as perceptrons, can be used to effectively classify data that is linearly separable. However, due to the changing nature of bidding behavior and the subtleties of shill bidding, it is difficult to justify bidding data as linearly separable. As a result, we looked into classifiers that are capable of classifying linearly inseparable data. Feed-forward backpropagating neural networks are known to possess enough complexity to effectively classify linearly inseparable data [38]. In addition, most types of neural networks do not require a complete reconstruction when new training data becomes available; instead, they can adapt to new data by training them on new datasets. The incremental nature of the neural network approach brings the distinct advantage of using a smaller new training set for a faster training time, while still retaining the network's ability to correctly classify old data points. Although this approach can be prone to overfitting on the current training data, our process overcomes this disadvantage by utilizing the previously described window concept and various stopping conditions discussed in Section 4.5.

The neural network we adopted in RT-SAC consists of three layers. The first layer, called the *input layer*, contains neurons for each of the input attributes related to shill bidding. The second layer, called the *hidden layer*, contains a reasonable number of neurons for computation. Note that an improper number of hidden neurons may lead to very poor performance of the classifier. For example, too few hidden neurons may result in fast but inaccurate training, whereas too many hidden neurons may lead to unnecessarily long processing times. Since there is no general approach to determining the optimal number of neurons in the hidden layer, this number is typically found by experimentation [38]. The last layer, called the *output layer*, consists of two neurons, one for each possible bidder class assignment, namely *normal* or *suspicious*. The activation function defined for a layer determines the firing rules for its residing neurons. In our approach, we adopted the commonly used *sigmoid* function as the activation function for the hidden layer and the output layer.

To handle the issue of false negatives we discussed earlier, we define an adjustable decision function that interprets the outputs of the neural network. When the neural network produces outputs at its output neurons, the outputs are in the form of real numbers from -1 to +1. A decision function can interpret the outputs of all output neurons and classify the input example. For our approach, we require that the decision function classifies a bidder as *normal* only if it is fairly certain of the bidder's innocence. For example, if the "normal" neuron outputs 0.9 and the "suspicious" neuron outputs -0.6, the decision function should interpret the classification as *normal* with a large degree of certainty. On the other hand, if both neurons output a value of 0.6, the decision function should interpret it as being unsure whether the bidder is *normal* or *suspicious*. In the case of uncertainty, the outputs of the classifier are interpreted as a *suspicious* classification. By doing so, we can reduce the number of potentially harmful false negatives in favor of insignificantly increasing the number of harmless false positives. The algorithm for the decision function is defined as in Algorithm 2.

Algorithm 2: Decision Making

Input: The outputs from the "normal" neuron and the "suspicious" neuron, and a threshold
Output: *Suspicious* / *Normal*

```
1. DecisionFunction (NormalOutput out1, SuspiciousOutput out2, float threshold)
2.    if (out1 and out2 are both negative)
3.       return Suspicious
4.    else if (out1 <= out2)
5.       return Suspicious
6.    else if (the difference between out1 and out2 is less than threshold)
7.       return Suspicious
8.    else
9.      return Normal
```

Note that the parameter threshold represents the similarity tolerance that allows for dynamic adjustments to the classification process. It is initially set to 0.8; however, if we notice that the network is not classifying enough bidders as suspicious, we should set the parameter to a larger value to increase the likelihood of *suspicious* classifications.

In our approach, we adopt a resilient backpropagation (Rprop) algorithm to train the network [39]. Rprop is a supervised learning technique that has shown promising classification accuracy and time performance results compared to other popular training techniques such as standard backpropagation and the SuperSAB algorithm. Rprop is similar to the standard backpropagation algorithm for training a network, but it updates the weights and biases of a network in a slightly different manner. For every training example, Rprop classifies the example and calculates the error, which is the difference between the network's output and the expected value of the output attribute. Instead of calculating a gradient descent term from the error using a derivative function and applying it directly to the weights and biases, Rprop calculates this term and only uses its sign in determining how it should modify the weights and biases. The algorithm interprets a positive sign as an increase of error, so it decreases a weight or bias by an update value; on the other hand, a negative sign is considered as a decrease of

error, so the weight or bias is increased by an update value. At the end of a training iteration, when all training examples have been processed once, the validation set is classified. If the data points in the validation set are not classified accurately enough, another training iteration takes place and more modifications to the weights and biases need to be done. This process stops when the network classifies the validation set well enough or any stopping condition is met. Refer to [39] for more detailed description of the Rprop algorithm.

4.3. Outlier Detection

Outliers are elements of a set that have significantly different values from the majority of the set. Osborne and Overbay discussed the potential causes of outliers and the adverse effects of including outliers for training on error rates [40]. Regardless of whether the outliers are legitimate members of the sampled population or they are due to sampling errors, in most scenarios, removal of outliers for training leads to improvements in error rate [36, 37]. In our approach, we consider a bidder to be an outlier if it possesses a value of an attribute that is significantly different from that of other bidders (e.g., more than 5 standard deviations away from the mean). Since we use Prolog rules to define parameters such as the number of required standard deviations, such parameters are adjustable at runtime. An example of such a rule is listed as follows.

```
isOutlier(B) :- standardDeviationThresholdExceeded(B).
standardDeviationThresholdExceeded(B) :-
  dataConnect(B, V1), dataConnect(B,V2),
  analyzeThresholdResult(V1, V2, 5.0).
dataConnect(B,V1) :-
  class('io.DatasetManager') <-
  getSDAwayFromNormalMean(B) returns V1.
dataConnect(B,V2) :-
  class('io.DatasetManager') <-
  getSDAwayFromSuspiciousMean(B) returns V2.
analyzeThresholdResult(V1,V2,X):- V1 > X, V2 > X.
```

The SR module evaluates the `isOutlier` predicate with the parameter of the bidder's ID `B`. A Prolog engine then analyzes if there is a solution to `standardDeviationThresholdExceeded` for the bidder. To determine this, the Prolog engine needs to access an external Java class called `DatasetManager` by invoking two methods, namely `getSDAwayFromNormalMean` and `getSDAwayFromSuspiciousMean`, to calculate the number of standard deviations that the bidder is away from the mean of the groups of normal bidders and suspicious bidders, respectively. Once the methods return the results `V1` and `V2`, the engine compares them with a predefined threshold, i.e., 5.0. If a bidder has an attribute value that is more than 5 standard deviations away from the means of that attribute for both the groups of normal and suspicious bidders, the predicate `isOutlier` evaluates to *true*, signaling the SR module that the bidder is an outlier.

4.4. Selection and Replacement of Data Points

The SR module makes decisions on replacement of data points in the data pool, and creates training, validation, and test sets. Since the size of the data pool may become oppressively large over time, it is necessary to continuously remove old data points from the data pool. In Figure 4, we demonstrated our basic strategy for replacing data points. However, since the majority of bidders in online auctions are normal, if we simply replace the oldest data points with the newest ones, the data pool may eventually contain too few suspicious bidders. In this case, when we generate a test set from the data pool to evaluate the network performance, it will likely retrieve a test set comprised solely of normal bidders, so it would not have an accurate representation of how well the network classifies suspicious bidders. Thus, it is necessary that our replacement policy ensures that the data pool contains enough normal and suspicious bidders no matter the distribution of arriving data. The current replacement policy considers the oldest bidders with matching classifications for replacement when adding new bidders into the data pool. For example, if 6 normal bidders and 1 suspicious bidder are added into the data pool, the oldest 6 normal bidders and the oldest suspicious bidder will be removed from the data pool. The following rules show an example of the replacement policies.

```
replaceBidder(B,C,R):- replaceOldDataWithMatchingClassification(C,R).
replaceOldDataWithMatchingClassification(C,R):-
  class('io.DatasetManager') <- retrieveOldestBidder(C) returns R.
```

In the above example, the SR module evaluates the predicate `replaceBidder` with the parameters of the bidder's ID `B` and the bidder's classification `C`. The variable `R` is a return variable that is used by the SR module to receive a bidder ID for replacement if the Prolog engine finds the oldest bidder with the matching classification `C` in the data pool.

To create training and validation sets is not a trivial process, and it greatly affects the training performance of a neural network. Based on the *window* concept, we further define that the training set and the validation set contain 75% and 25% of eligible bidders, respectively. For example, if the window size is defined as 9, the training set and the validation set contain 7 and 2 data points, respectively. Note that there is no overlapping between the two datasets. Thus, a validation set can be evaluated as an unbiased indicator of the network's true performance. The corresponding policies can be defined as Prolog facts as follows.

```
trainingProportion(T):- T is 75.
validationProportion(V):- V is 25.
windowSize(W):- W is 9. windowSpeed(S):- S is 3.
```

Ideally, the SR module should be able to automatically determine the optimal size of the created sets, and maintain proper percentages of old and new, as well as normal and suspicious bidders, within the sets. A more sophisticated SR module with the above advanced features is envisioned as a future, and more ambitious research direction.

4.5. Real-Time Incremental Training

The RTT module trains the classifier incrementally and ensures that each phase of the network training and validation does not degrade the classifier's performance. Since the sizes of the training set and validation set are quite small due to the small window size, the incremental training process typically takes very little time. Furthermore, by utilizing proper stopping conditions to determine whether the current training phase is complete, the module ensures that each training cycle leads to the network's gradual, yet timely, adaptation to new data. Note that if the stopping conditions are improperly defined, the training cycle may finish very quickly or never finish at all. There are many common stopping rules in use by other researchers for neural networks. For example, Zhou and Si defined a wide variety of stopping conditions, including a minimum and maximum number of training iterations (epochs), an acceptable training error threshold, and an acceptable validation error threshold [41]. Examples of the stopping conditions adopted in our approach are defined as Prolog rules as follows.

```
// 1st stopping condition
isFinishedTraining(E,V,I):- maxEpochExceeded(E).
// 2nd stopping condition
isFinishedTraining(E,V,I):- minEpochExceeded(E),
  validationPerformanceAcceptable(V).
// 3rd stopping condition
isFinishedTraining(E,V,I):-
  validationImprovementLimitReached(I).
maxEpochExceeded(E):- E >= 5000. minEpochExceeded(E):- E >= 100.
validationPerformanceAcceptable(V):- V >= 90.
validationImprovementLimitReached(I):- I >= 100.
```

Because network training may potentially never end, it is required to define a maximum number of epochs that are allowed to take place. As shown in the above Prolog rules, for the first stopping condition, we define the maximum number of epochs as 5,000 since they can complete in a reasonable amount of time without significantly sacrificing performance. In reality, due to other stopping conditions, this number is rarely reached. Note that too many epochs may lead to *overfitting* because the training process will continually present the same examples to the network. On the other hand, too few epochs can lead to *underfitting* since the network does not have sufficient time to learn the training data. Thus, we define the second stopping condition as a minimum number of epochs along with a satisfactory classification for the validation set. As shown in the Prolog rules, we impose a minimum of 100 training epochs, which is typically sufficient to change a network's weights and biases reasonably, and we also require 90% of the data points from the validation set be classified correctly before the training process can stop. Note that when the window size is small (e.g., a window size of 9), the second stopping condition essentially requires all data points in the validation set be correctly classified.

Since the validation set is not overlapped with the training set, it is possible that the validation performance stops improving too early, so the requirement of correctly classifying 90% of the data points from the validation set cannot be satisfied with continued iterations on the training set. In this

case, it is likely that the network has been *overfitted* to the training data, and the performance on the validation data will become worse if the training process continues. Thus, as a third stopping condition, if the network's performance on the validation set does not improve after 100 consecutive iterations, the RTT module discontinues training before the network become too heavily overfitted on the training data. Note that 100 consecutive iterations are sufficient for determining whether the network's performance on the validation set is improving; meanwhile, 100 consecutive iterations would not result in dramatic performance downgrading due to overfitting.

5. CASE STUDY

In the following case study, we present a series of experiments designed to demonstrate the feasibility and real-time performance of our RT-SAC approach. We first discuss the datasets used in the experiments and summarize the results of using hierarchical clustering for creation of the training and simulation datasets. Then we demonstrate how certain parameters can be tuned for optimal operations in RT-SAC based on experimental results. Finally, we present two real-time simulations to show how RT-SAC can effectively classify new data points and how it can adapt itself when there is a market change for the auctioned item.

5.1. Data Collection and Preprocessing

In order to demonstrate the classifier's ability to adapt to new auction data, we collected two separate datasets from eBay. Both datasets consist of completed 1-day eBay auctions for "*Used Playstation 3*" auctioned items. Dataset 1 contains 153 auctions with a total of 1,845 bidders, which was collected in October and November 2009 for over 30 days. Dataset 2 was collected after 6 months for 2 days and contains 23 auctions with a total of 180 bidders. Although the two datasets are auction data for the same auction type, during the 6 month period, the number of 1-day auction listings for "*Used Playstation* 3" dropped drastically, as most of the auctions available at that time were for 3 or 5-day auctions. In addition, there is a slight difference concerning the number of bids placed by the bidders in the datasets. For example, 57% of bidders in dataset 1 placed only one bid in an auction, while in dataset 2, the percentage of bidders who placed only one bid in an auction dropped to 48%.

Before applying the cluster generation algorithm to the collected auction data to generate training datasets, we first assign weights to the various attributes. We deem the attributes *NB (early, middle)*, *ETFB*, and *ATUB (early, middle)* to be the most important evidence for suspicious bidding behavior, so we assign these attributes a weight of 3. The attribute *ABI (early, middle)* is useful in describing how aggressive a bidder is, but it is not as strong as the above ones. Thus, we assign this attribute a weight of 2. Other attributes are assigned the default weight of 1. After running the cluster generation algorithm and examining the clustering results, we found that a minimum similarity cutoff point of 89.6% led to a reasonable number of clusters. Note that a more restrictive cutoff value results in more

clusters with similar behavior and a less restrictive cutoff value leads to fewer clusters with combinations of normal and suspicious behaviors.

After the clusters are generated, we label them with either *normal* or *suspicious* based on the common bidder behaviors in each cluster. Table 1 lists the manual labeling results for dataset 1. As an example for manual labeling, cluster 1 is the largest cluster of bidders, in which most bidders placed their bids very late in their auctions. Since shill bidders will not risk placing bids late in an auction for fear of winning, cluster 1 is considered to contain bidders with normal behavior. On the other hand, cluster 17 contains bidders that start bidding very early (during the first 45 minutes of the auction). Since such behavior denotes possible prior knowledge of the auction, we consider the bidders in this cluster to be suspicious. Note that most of the suspicious activity listed in the table can be directly linked to a certain type of shilling behavior described in [21, 42]. To ensure correct labeling, we further utilize the D-S based shill verifier described in [8] to verify the results of the labeling process. This verification process is necessary especially for bidders that are difficult to classify when they are near the boundaries of two or more differently labeled clusters. By verifying the clustering results using the shill verifier, we ensure that the boundary bidders can be correctly classified as suspicious ones if additional evidence supports their shilling behaviors.

TABLE 1. Cluster labeling for dataset 1.

Cluster	Size	Class	Description of major bidding behaviors
1	62%	Normal	Bids placed very late in auction (later middle or final stage).
2	<1%	Suspicious	Very high bidding amounts in middle stage.
3	3%	Suspicious	Bids placed close together in middle stage. Possible bid unmasking.
4	<1%	Suspicious	Large number of bids placed only in middle stage.
5	<1%	Suspicious	Bids placed in quick succession in middle stage. Possible bid unmasking.
6	8%	Normal	Few bids placed only in middle stage.
7	1%	Normal	Only one bid placed in middle stage.
8	<1%	Suspicious	Bids with moderate bidding amounts placed only in middle stage.
9	<1%	Suspicious	Moderate number of bids placed only in middle stage.
10	<1%	Suspicious	Moderate number of bids placed only at the end of middle stage.
11	8%	Normal	Few bids placed only in middle of auction.
12	1%	Normal	Very few bids placed moderately early (within 3 hours) in auction.
13	<1%	Suspicious	Bids placed moderately early (within 3 hours) with moderate amounts.
14	<1%	Suspicious	Bids placed early and in quick succession in early stage.
15	2%	Suspicious	Bids placed in quick succession in middle stage. Possible bid unmasking.
16	<1%	Suspicious	Very large bidding amount in middle stage.
17	9%	Suspicious	Bids placed very early (within 45 minutes) in auction.
18	1%	Suspicious	Bids placed early with moderate amounts in early stage.
19	<1%	Normal	Very few bids placed in early stage.
20	1%	Suspicious	Moderate number of bids placed only in early stage.
21	<1%	Suspicious	Bids placed in quick succession in early stage.
22	<1%	Suspicious	Large number of bids placed in early stage.

5.2. Experimental Results and Analysis

We developed a prototype RT-SAC based on the Encog neural network and machine learning framework [43], which provides efficient multi-threaded implementations of various neural network learning algorithms including Rprop. By utilizing the Encog implementation of the neural networks, RT-SAC can complete a whole phase of classification and incremental training typically in less than 1

second on a machine with a 2.1 GHz dual-core processor and 3.0 GB of RAM. Before we perform the experiments and simulations, we first sort dataset 1 by auction end time and divide it into two subsets: an initialization set that contains the first 135 auctions with a total of 1,656 bidders and a simulation set that contains the last 18 auctions with a total of 189 bidders. Since the initialization set is used to initialize the neural network, it should be large and representative enough so that the network has a sufficient baseline to correctly classify future data points. After the network has been initialized, the simulation set can be used by the classifier to simulate classification and incremental training in real-time. The classifier utilizes the workflow described in Section 4.1 and processes each auction in the sorted simulation set in sequential order.

In order to evaluate the real-time performance of the classifier for both recent and old data, we create two dynamic test sets with a combined size of 500. The first set is created from all previously processed data points from the simulation set except those belong to the current window. For example, if the window size is 9 and 50 bidders from the simulation dataset have been processed, the 41 data points outside of the window are used to create a test set for recent data. Note that the test set for recent data grows as the simulation goes on, which should lead to more accurate evaluation results for the performance of the classifier when the test set for recent data is getting sufficiently large. On the other hand, the test set for old data is created by randomly selecting data points from the initialization set. Since we define a fixed total size of 500 for the test sets of recent and old data, the test set for old data initially equals 500, and then shrinks as the simulation goes on. However, this does not matter because there are only 189 data points in the simulation set, the test set for old data will still be large enough towards the end of the simulation. Thus, the test set for old data can always be used to perform accurate evaluations for the old data. In addition to the error rates for recent and old data, we further define a combined error rate for both recent and old data as follows.

$$CombinedErrorRate = \frac{(so * e_o) + (sr * e_r)}{so + sr} \qquad (7)$$

where so is the size of the test set for old data, e_o is the error rate for old data points, sr is the size of the test set for recent data, and e_r is the error rate for recent data points. Note that in our case study $so + sr$ equals 500.

Since we introduced a number of parameters in RT-SAC, such as the window size, the outlier threshold, and the number of hidden neurons, which may affect the performance of the classifier, we are interested to know whether these parameters can be tuned for better performance of the classifier. Based on our experiments and observation, the parameters in question are relatively independent of each other. While tuning the parameters, we noticed that by optimizing one parameter, the previous optimal values for the other parameters are not necessary to change. For example, based on our experiments, an outlier threshold of 5.0 is an optimal threshold for RT-SAC regarding outlier removal. Similarly, a reasonable change from the optimal number of hidden neurons (based on our experimental results, the optimal number of hidden neurons is 5 for RT-SAC) also does not affect the

optimal values of the window size and the outlier threshold. On the other hand, the window size turns out to be a critical parameter to be adjusted. A window size that is too small may cause significant fluctuations in the error rate because the network adjusts its weights and biases too fast due to the small size of the training and validation set. However, a window size that is too large may slow down the process that the network adapts to new data points and increase the time required to complete the training and validation phase. We now use the window size as an example to demonstrate how to tune parameters for optimal operations in RT-SAC. For the experiments of tuning window size, we ran the prototype RT-SAC for window sizes of 3, 6, 9, and 12 with a constant window speed of 3. Figure 5 shows the combined error rates with different window sizes.

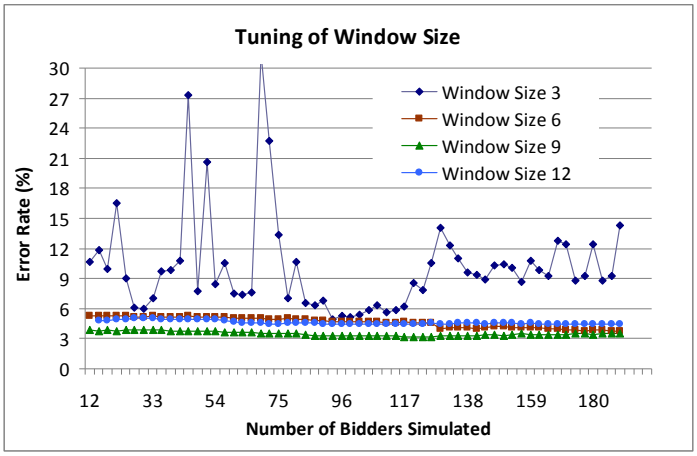

FIGURE 5. Experimental results for tuning of window size.

From the figure we can see that when the window size is 3, the experimental result indicates that the system performance in terms of error rate is very poor. This is because when the window size is 3, the training and validation set only consist of 2 and 1 data points, respectively. This leads to significant fluctuations in error due to insufficient number of elements in the training and validation datasets. The window size of 9 turns out to be the best case, where the training and validation sets are balanced out nicely by previously processed simulation data and newly processed auction data, causing the network to smoothly adapt to the new data. Note that a window size of 9 with a window speed of 3 allows each bidder to be used three times for incremental training, so the network has sufficient time to learn the new data effectively. The error rates for a window size of 6 and 12 are a little bit higher but do not appear to differ significantly from the error rates for a window size of 9. Bidders in the smaller window size of 6 with a window speed of 3 are used twice for incremental training, leading to slight *underfitting* since the network does not have sufficient time to learn the new data. On the other hand, bidders in the larger window size of 12 with a window speed of 3 are used four times for incremental training, potentially leading to *overfitting* since bidders are reused too often. As a result, we choose a window size of 9 as the optimal window size for the classifier.

Note that in the experiments, the outlier threshold and the number of hidden neurons are set to the

optimal values of 5.0 and 5, respectively. Based on our further experiments, when we used reasonable values for the above two parameters, the experimental results lead to the same conclusion that the window size of 9 is the optimal one. However, it is worth noting that if we increase the window speed, the optimal window size increases accordingly due to the issues of *underfitting* and *overfitting*, which also results in more computation time for each training and validation phase. Due to the real-time requirement for RT-SAT, we choose a window speed of 3 as the optimal value.

In order to evaluate the performance of RT-SAC, we designed two simulations. We are primarily concerned with whether or not the classifier can maintain a relatively low error rate and perform its classification and incremental training procedures quickly enough for use in a real-time environment. The goal of the first simulation is to demonstrate how the classifier performs when there is no time gap between the historical dataset and the simulation dataset. Figure 6 shows the simulation results for the combined error rate as well as error rate for old and recent data points during the simulation period.

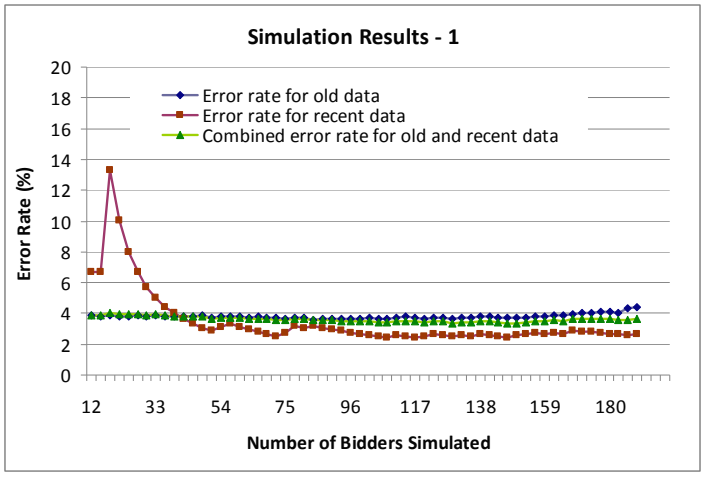

FIGURE 6. Simulation results for recent data collected without a time gap.

From the figure, we can see that at the beginning of the simulation, there is a very high error rate for recent data. This is because the test set for recent data starts very small due to insufficient data points, although it gradually increases in size as more simulation data is processed. Once there are enough new data points (around 40) included in the test set, the error rate for recent data drops significantly. On the other hand, the error rate for old data remains stable when the classifier adapts to the new data points. The results are as expected because the simulation dataset contains auction data collected in the same month as when the initialization dataset was collected, so the bidders contained in these two datasets should exhibit similar, if not nearly identical, bidding behaviors. Note that during the simulation, the combined error rate, as defined in (7), does not fluctuate significantly and remains stable. Thus, the results demonstrate that the network can successfully learn new data in a stable and controlled manner when there is no major market change.

Figure 7 shows the time performance of the entire simulation process, including all phases of

classification and incremental training on new data points from the simulation dataset. The relatively stable time performance demonstrates that the neural network can quickly classify and adapt to new data points. The few spikes in time performance could be due to validation sets that are difficult to classify, resulting in more iterations being performed in the incremental training processes. Although some validation sets might be difficult to classify, the accuracy results indicate that the network's performance was not significantly impacted.

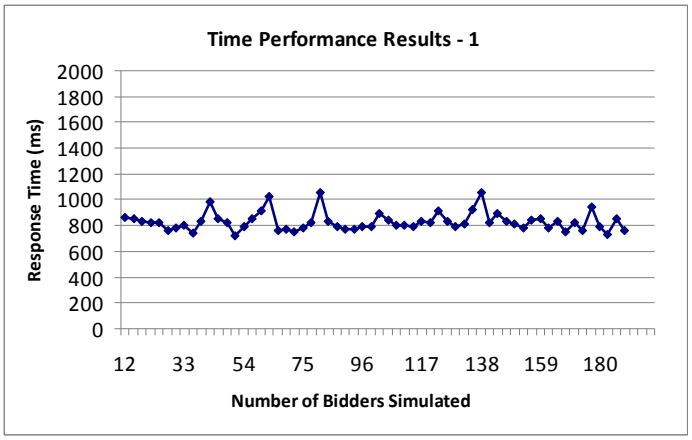

FIGURE 7. Time performance for recent data collected without a time gap.

In the second simulation, we demonstrate how the classifier performs when adapting to new data points that might involve a market change. In this experiment, we initialize the neural network with data points from dataset 1. Then we use the entire dataset 2 as the simulation dataset to perform the experiment. Note that there is a time gap of approximately six months between the collection times of the two datasets. Similar to the first simulation, the new simulation set, namely dataset 2, is also sorted by auction end time, and during the period of simulation, the classifier processes each auction in the sorted simulation set in sequential order. In order to evaluate the real-time performance of the classifier for both recent and old data, we create two dynamic test sets with a combined size of 500, where the test set for new data is created from previously processed data points from the simulation set (i.e., dataset 2) and the test set for old data is created from the initialization set (i.e., dataset 1). Figure 8 shows the simulation results for the combined error rate as well as error rates for old and recent data during the simulation period. As shown in the figure, there is a slight increase in the error rate for old data when around 110 bidders have been processed. The increase in error rates for old date indicates that the classifier has been changed during the simulation period due to a possible market change. On the other hand, the increase in the error rate for recent data turns out to be more significant. Although the error rate for recent data appears low in the beginning, the initial limited size of the test set for recent data prevents us from making any meaningful conclusions about the network's performance on new data. When approximately 90 bidders are processed, the classifier experiences an apparent increase in error for recent data, which represents the actual performance of

the network on recent data. As the simulation continues, the network adapts smoothly to the recent data, without increases further in error rate for the old data points. During the simulation, the combined error rate also starts to reflect the actual performance of the network on both old and recent data when there are approximately 90 bidders processed. After that, the combined error rate becomes stable (below 4%), which indicates the classifier's satisfactory performance.

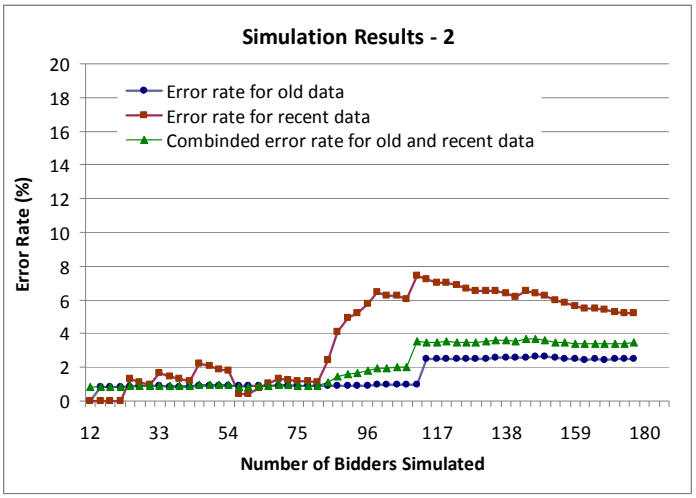

FIGURE 8. Simulation results for recent data collected with a time gap.

Figure 9 shows that the time performance of the second simulation is stable throughout the simulation process. The predictable efficiency of the classification and incremental training phases indicate that our process can be effectively applied in a real-time environment.

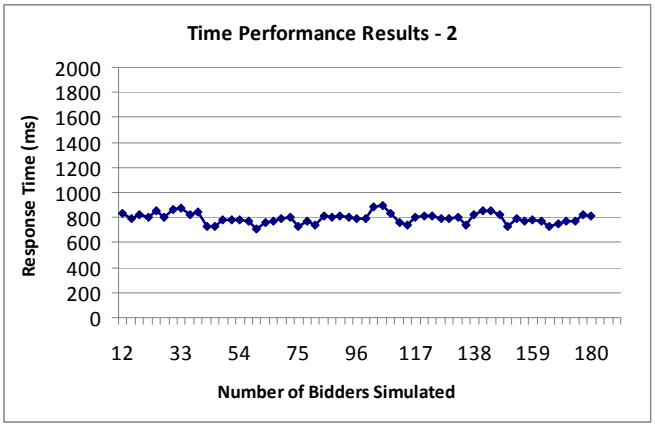

FIGURE 9. Time performance results for recent data collected with a time gap.

It is worth noting that we chose the auction type of "*Used Playstation 3*" for our case study because its popularity and price range make it a good target for shilling. When a different type of auctioned item is selected, users' bidding behaviors may change accordingly. In this case, the values of the system parameters will certainly be affected. To ensure good performance of RT-SAC, each classifier should undergo a tuning phase based on the selected type of auctions, in which the participating bidders are to be classified.

6. CONCLUSIONS AND FUTURE WORK

The increasing popularity of online auctions is almost sure to lead to an increase in auction fraud activities, either obvious or covert. Detecting covert shill bidding in this rapidly growing space requires the use of advanced automated techniques that can adapt to changes in bidding trends. In this paper, we introduced a series of attributes that effectively describe important aspects of bidders' bidding behavior as well as useful information about auctions in which the bidders participate. These attributes allow us to take advantage of automated data mining techniques to cluster and classify auction data. We presented a real-time self-adaptive classifier, called RT-SAC, which is capable of accurately classifying and smoothly adapting to new data. In addition, it performs fast enough to be utilized in a real-time environment. By utilizing a Prolog engine and a separate classification module, the classifier can be easily configured at runtime. When used in conjunction with existing shill verification techniques [8, 22], our RT-SAC approach can greatly increase the efficiency and effectiveness for real-time shill detection in online auctions.

For future work, we will apply our RT-SAC approach to more data sets from online auctions, especially for different types of auctioned items to further verify the effectiveness of our approach. We will consider introducing more input attributes such as the average final price of an auction, and using feature selection techniques to select the most significant input attributes for classification, thereby reducing the complexity of the neural network and improving the time performance of the classifier. We notice that it is important to study different ways of clustering such as the one proposed in Li *et al.*'s work to improve our clustering results [44]. Furthermore, we plan to utilize machine learning techniques to improve RT-SAC, so it could automatically determine the optimal values of various parameters such as the sizes of the training set and validation set, as well as the proper percentages of normal and suspicious bidders within these datasets. As an alternative to neural networks, we will also investigate the usage of SVM [45] in our RT-SAC approach. Due to the more accurate and faster performance of SVM comparing to traditional neural networks, an SVM-based RT-SAC could potentially outperform our current neural network based classification and incremental training method.

ACKNOWLEDGEMENTS

We thank the referees and the editors for their comments that helped us clarify some performance issues of the classifier, and also for their detailed comments concerning related work and future research directions.

FUNDING

This material is based upon work supported by the U.S. National Science Foundation under grant number CNS-0715648.

REFERENCES

[1] Power Sellers Unite (2010), Auction site count: keep track of the number of listings on auction sites, http://www.powersellersunite.com/auctionsitewatch.php (accessed 20 July, 2010).

[2] Dong, F., Shatz, S. M., and Xu, H. (2009) Combating online in-auction fraud: clues, techniques and challenges. *Computer Science Review*, **3**, 245-258.

[3] Trevathan, J. and Read, W. (2006) Secure online English auctions. *Proceedings of the Int. Conf. Security and Cryptography (SECRYPT)*, Setúbal, Portugal, 7-10 August, pp. 387-396.

[4] Swamynathan, G., Zhao, B. Y., Almeroth, K. C., and Jammalamadaka, S. R. (2008) Towards reliable reputations for dynamic networked systems. *Proceedings of the IEEE 27th Int. Symp. Reliable Distributed Systems (SRDS 2008)*, Naples, Italy, 6-8 October, pp.195-204.

[5] Chau, D. H., Pandit, S., and Faloutsos, C. (2006) Detecting fraudulent personalities in networks of online auctioneers. *Proceedings of the 10th European Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD 2006)*, Berlin, Germany, 18-22 September, pp. 103-114.

[6] Ku, Y., Chen, Y., and Chiu, C. (2007) A proposed data mining approach for internet auction fraud detection. *Proceedings of the 2007 Pacific Asia Conf. Intelligence and Security Informatics (PAISI'07)*, LNCS 4430, Chengdu, China, 11-12 April, pp. 238-243. Springer-Verlag Berlin, Heidelberg.

[7] Ochaeta, K. E. (2008) *Fraud detection for internet auctions: a data mining approach*. Master's Thesis, National Tsing Hua University, Taiwan.

[8] Dong, F., Shatz, S. M., and Xu, H. (2010) Reasoning under uncertainty for shill detection in online auctions using Dempster-Shafer theory. *Int. J. Software Engineering and Knowledge Engineering (IJSEKE)*, **20**, 943-973.

[9] Yoon, Y., Bien, S., and Park, S. (2010) Microarray data classifier consisting of k-top-scoring rank-comparison decision rules with a variable number of genes. *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, **40**, 216-226.

[10] Che, D., Hockenbury, C., Marmelstein, and R., Rasheed, K. (2010) Classification of genomic islands using decision trees and their ensemble algorithms, *BMC Genomics*, **11** (suppl 2):S1.

[11] Ford, B., Xu, H., and Valova, I. (2010) Identifying suspicious bidders utilizing hierarchical clustering and decision trees. *Proceedings of the 12th Int. Conf. Artificial Intelligence (ICAI'10)*, Las Vegas, Nevada, USA, 12-15 July, pp. 195-201.

[12] Bapna, R., Goes, P., and Gupta, A. (2004) User heterogeneity and its impact on electronic auction market design: an empirical exploration. *MIS Quarterly*, **28**, 21-43.

[13] Shah, H. S., Joshi, N. R., Sureka, A., and Wurman P. R. (2002) Mining for bidding strategies on eBay. *Proceedings of the 4th Int. Workshop Mining Web Data (WEBKDD 2002)*, LNAI 2703, Edmonton, Canada, 23 July, pp. 17-34. Springer-Verlag.

[14] Hou, J. and Rego, C. (2007) A classification of online bidders in a private value auction: evidence from eBay. *Int. J. Electronic Marketing and Retailing*, **1**, 322-338.

[15] Gelenbe, E. and Gyorfi, L. (2009) Performance of auctions and sealed bids. *Proceedings of the 6th European Performance Engineering Workshop (EPEW 2009)*, LNCS 5652, Imperial College London, 9-10 July, pp. 30-43. Springer Verlag, Berlin and Heidelberg.

[16] Gelenbe, E. (2009) Analysis of single and networked auctions. *ACM Trans. on Internet Technology*, **9**, Article No. 8.

[17] Dong, F., Shatz, S. M., Xu, H., and Majumdar, D. (2011) Price comparison: a reliable approach to identifying shill bidding in online auctions? *Electronic Commerce Research and Applications (ECRA)*, Article in Press, Available online December 19.

[18] Gelenbe, E. and Velan, K. (2009) An approximate model for bidders in sequential automated auctions. *Proceedings of the 3rd KES Int. Symp. Agent and Multi-Agent Systems: Technologies and Applications (KES-AMSTA 2009)*, LNAI 5559, Uppsala, Sweden, 3-5 June, pp. 70-79. Springer Verlag, Berlin.

[19] Velan, K. and Gelenbe, E. (2010) Analysing bidder performance in randomised and fixed-deadline automated auctions. *Proceedings of the 4th KES Int. Symp. Agent and Multi-Agent Systems: Technologies and Applications (KES-AMSTA 2010)*, LNCS 6071, Gdynia, Poland, 23-25 June, pp. 42-51. Springer Verlag, Berlin and Heidelberg.

[20] Kauffman, R. J. and Wood, C. A. (2000) Running up the bid: modeling seller opportunism in Internet auctions. *Proceedings of the 6th Americas Conf. Information Systems (AMCIS 2000)*, M. Chung (Ed.), Long Beach, CA, 10-13 August, pp. 929-935.

[21] Xu, H., Bates, C. K., and Shatz, S. M. (2009) Real-time model checking for shill detection in live online auctions. *Proceeding of the Int. Conf. Software Engineering Research and Practice (SERP'09)*, Las Vegas, Nevada, USA, 13-16 July, pp. 134-140.

[22] Goel, A., Xu, H., and Shatz, S. (2010) A multi-state Bayesian network for shill verification in online auctions. *Proceedings of the 22nd Int. Conf. Software Engineering and Knowledge Engineering (SEKE'2010)*, Redwood City, San Francisco Bay, USA, 1-3 July, pp. 279-285.

[23] Kamo, T. and Dagli, C. H. (2009) Hybrid approach to the Japanese candlestick method for financial forecasting. *Expert Syst. Appl.* **36**, 5023-5030.

[24] Järvelin, A., Juhola, M., and Laine, M. (2006) Neural network modelling of word production in Finnish: coding semantic and non-semantic features, *Neural Computing & Applications*, **15**, 91-104.

[25] Wang, J.-C. and Chiu, C.-Q. (2005) Detecting online auction inflated-reputation behaviors using social network analysis. *Proceedings of the Annual Conf. of the North American Association for Computational Social and Organizational Science (NAACSOS 2005)*, Notre Dame, Indiana, USA, 26-28 June, pp. 373-381.

[26] Dong, F., Shatz, S. M., and Xu, H. (2010) An empirical evaluation on the relationship between final auction price and shilling activity in online auctions. *Proceedings of the 22nd Int. Conf. Software Engineering and Knowledge Engineering (SEKE'2010)*, Redwood City, San Francisco Bay, USA, 1-3 July, pp. 286-291.

[27] Yin, H. B., Wong, S. C., Xu, J. M., and Wong, C. K. (2002) Urban traffic flow prediction using a fuzzy-neural approach. *Transportation Research Part C: Emerging Technologies*, **10**, 85-98.

[28] Yang, Y., Vilathgamuwa, D. M., and Rahman, M. A. (2003) Implementation of an artificial-neural-network-based real-time adaptive controller for an interior permanent-magnet motor drive. *IEEE Trans. Industry Applications,* **39**, 96-104.

[29] Musliner, D. J., Goldman, R. P., Pelican, M. J., and Krebsbach, K. D. (1999), Self-adaptive software for hard real-time environments, *IEEE Intelligent Systems*, **14**, 23-29.

[30] Sztipanovits, J., Karsai, G., and Bapty, T. (1998), Self-adaptive software for signal processing, *Communications of the ACM (CACM)*, **41**, 66-73.

[31] Hou, M., Zhu, H., Zhou, M., and Arrabito, G. R. (2011), Optimizing operator-agent interaction in intelligent adaptive interface design: a conceptual framework, *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, **41**, 161-178.

[32] Manning, C. D., Raghavan, P., and Schütze, H. (2008) *Introduction to Information Retrieval*. Cambridge University Press.

[33] Engelberg, J. and Williams, J. (2009) EBay's proxy bidding: a license to shill. *J. Economic Behavior & Organization*, **72**, 509-526.

[34] Houser, D. and Wooders, J. (2006) Reputation in auctions: theory and evidence from eBay. *J. Economics and Management Strategy*, **15**, 353-369.

[35] Livingston, J. A. (2005) How valuable is a good reputation? a sample selection model of Internet auctions. *The Review of Economics and Statistics*, **87**, 453-465.

[36] Fukunaga, K. (1990) *Introduction to Statistical Pattern Recognition*. Academic Press, Inc.

[37] Robson, G. (2003) *Multiple outlier detection and cluster analysis of multivariate normal data*. Master's Thesis, University of Stellenbosch, South Africa.

[38] Abdi, H. (1994) A neural network primer. *J. Biological Systems*, **2**, 247-283.

[39] Riedmiller, M. and Braun, H. (1993) A direct adaptive method for faster backpropagation learning: the RPROP algorithm. *Proceedings of the IEEE Int. Conf. Neural Networks (ICNN)*, San Francisco, CA, USA, pp. 586-591.

[40] Osborne, J. W. and Overbay, A. (2004) The power of outliers (and why researchers should always check for them). *Practical Assessment, Research & Evaluation*, **9**. Retrieved September 20, 2011 from http://PAREonline.net/getvn.asp?v=9&n=6

[41] Zhou, G. and Si, J. (1999) Subset-based training and pruning of sigmoid neural networks. *Neural Networks*, **12**, 79-89.

[42] Bates, C. K. (2009) *Agent-based shill detection in live online auctions using real-time model checking*. Master's Thesis, University of Massachusetts Dartmouth, MA, USA.

[43] Heaton, J. (2010) *Encog Java and dotNet neural network framework*. Heaton Research, Inc., Retrieved on July 20, 2010, from: http://www.heatonresearch.com/encog

[44] Li, T., Ogihara, M., and Ma, S. (2010) On combining multiple clusterings. *Applied Intelligence*, **33**, 207-219.

[45] Shawe-Taylor, J. and Cristianini, N. (2000) *Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, Cambridge, UK.

Benjamin J. Ford received the B.S. degree and the M.S. degree in Computer Science from University of Massachusetts Dartmouth, MA, in 2008 and 2010, respectively. His major research interests include neural networks, software engineering, and e-commerce.

Mr. Ford is a recipient of the Faculty Award for top students in the Computer and Information Science Department, University of Massachusetts Dartmouth, 2008.

Haiping Xu received the B.S. degree in Electrical Engineering from Zhejiang University, Hangzhou, China, in 1989, the M.S. degree in Computer Science from Wright State University, Dayton, OH, in 1998, and the Ph.D. degree in Computer Science from the University of Illinois at Chicago, IL, in 2003.

Prior to 1996, he successively worked with Shen-Yan Systems Technology, Inc. and Hewlett-Packard Co., as a Software Engineer, in Beijing, China. Since 2003, he has been with the University of Massachusetts Dartmouth, where he is currently an Associate Professor at the Computer and Information Science Department, and a Director of the Concurrent Software Engineering Laboratory (CSEL). His research has been supported by the National Science Foundation (NSF) and the U.S. Marine Corps. His major field of study is on distributed software engineering and formal methods. His current research interests include multi-agent systems, electronic commerce, data mining, service-oriented systems, Internet security, and cloud computing.

Dr. Xu is a senior member of the Association of Computing Machinery (ACM) and a senior member of the IEEE. He has served as program Co-Chairs for International Conference on Distributed Multimedia Systems (DMS) and the International Conference on Software Engineering Theory and Practice (SETP), and a program committee member for over 50 international conferences. He is a recipient of the Outstanding Ph.D. Thesis Award in 2004, and has been included in the 11th Edition of *Who's Who Among America's Teachers*, 2006.

Iren Valova received the B.S. degree in Computer Science from Technical University, Sofia, Bulgaria, in 1991, the M.S. degree in Applied Mathematics from Technical University, Sofia, Bulgaria, in 1993, and the Ph.D. degree in Computer Science from Tokyo Institute of Technology, Tokyo, Japan, in 1997.

She is currently a Professor with the Computer and Information Science Department, University of Massachusetts Dartmouth, North Dartmouth, and a Director of the Neural and Adaptive Research Laboratory (NASLAB). Her major field of study is on machine learning, expert systems, and artificial intelligence. Her current research interests include neural networks and learning algorithms, image processing, brain function modeling, medical image classification and processing.

Dr. Valova is a recipient of two Best Paper Awards at Artificial Neural Networks conferences, in the category of Theoretical Developments in Computational Intelligence, in 2002 and 2004, respectively.