5-2020

Dynamic Fraud Detection via Sequential Modeling

Panpan Zheng
University of Arkansas, Fayetteville

Dynamic Fraud Detection via Sequential Modeling

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Engineering with a concentration in Computer Science

by

Panpan Zheng
Nanchang Institute of Technology
Bachelor of Science in Computer Network, 2010
Northwest University
Master of Science in Computer Application Technology, 2013

May 2020
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council

---

Xintao Wu, Ph.D.
Dissertation Director

Qinghua Li, Ph.D.
Committee member

Song Yang, Ph.D.
Committee member

Lu Zhang, Ph.D.
Committee member

ABSTRACT

The impacts of information revolution are omnipresent from life to work. The web services have significantly changed our living styles in daily life, such as Facebook for communication and Wikipedia for knowledge acquirement. Besides, varieties of information systems, such as data management system and management information system, make us work more efficiently. However, it is usually a *double-edged sword*. With the popularity of web services, relevant security issues are arising, such as fake news on Facebook and vandalism on Wikipedia, which definitely impose severe security threats to OSNs and their legitimate participants. Likewise, office automation incurs another challenging security issue, insider threat, which may involve the theft of confidential information, the theft of intellectual property, or the sabotage of computer systems. A recent survey says that 27% of all cyber crime incidents are suspected to be committed by the insiders. As a result, how to flag out these malicious web users or insiders is urgent. The fast development of machine learning (ML) techniques offers an unprecedented opportunity to build some ML models that can assist humans to detect the individuals who conduct misbehaviors automatically. However, unlike some static outlier detection scenarios where ML models have achieved promising performance, the malicious behaviors conducted by humans are often dynamic. Such dynamic behaviors lead to various unique challenges of dynamic fraud detection:

- Unavailability of sufficient labeled data — traditional machine learning approaches usually require a balanced training dataset consisting of normal and abnormal samples. In practice, however, there are far fewer abnormal labeled samples than normal ones.

- Lack of high quality labels — the labeled training records often have the time gap between the time that fraudulent users commit fraudulent actions and the time that

they are suspended by the platforms.

- Time-evolving nature — users are always changing their behaviors over time.

To address the aforementioned challenges, in this dissertation, we conduct a systematic study for dynamic fraud detection, with a focus on: (1) *Unavailability of labeled data:* we present (a) a few-shot learning framework to handle the extremely imbalanced dataset that abnormal samples are far fewer than the normal ones and (b) a one-class fraud detection method using a complementary GAN (Generative Adversarial Network) to adaptively generate potential abnormal samples; (2) *Lack of high-quality labels:* we develop a neural survival analysis model for fraud early detection to deal with the time gap; (3) *Time-evolving nature:* we propose (a) a hierarchical neural temporal point process model and (b) a dynamic Dirichlet marked Hawkes process model for fraud detection.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor Dr. Xintao Wu, for his invaluable expertise and guidance provided throughout my doctorate, but also for his immeasurable support at both professional and personal levels, and without whom this work would have never seen the light. I would like to thank the other members in my dissertation committee, Qinghua Li, Song Yang and Lu Zhang, for their valuable interactions and feedback.

I would like to express my sincere and genuine thankfulness to Shuhan Yuan. In the past four years, I have been working closely with Shuhan and, without his encourage and help, my Ph.D study would be of more hardships. Also, I would like to thank all of the other team members, Depeng Xu, Yongkai Wu, Qiuping Pan, Wen Huang, Wei Du and Kevin Labille, who are always encouraging and helping me on the way to pursuing the truth. Meanwhile, I would like to thank Cheng Cao and Zheng Du who give me a lot of help in my internship at Amazon.

I would also like to thank my friends, Guogang Xu, Yansong Bai, Carlos Alberto Acosta, Jingming Liu, Andrew Power, Robert Yi Lee, Dennis Trinkle, Amy Trinkle, Stephen Cannes and Jacyln Cannes, who are always accompanying and helping me throughout my graduate studies.

Last but not least, I would like to express my deepest and wholehearted thanks to my parents, Jianxiang Zheng and Zengmei Liu, for their endless love and support. You raise me up and I love you forever.

DEDICATION

Dedicated to my mom and dad.

EPIGRAPH

God helps those who help themselves.

Benjamin Franklin

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Motivation

Information revolution, which was emerging in 1990s, brings a far-reaching influence to our society from life to work. The web services, such as online social networks (OSNs) and knowledge bases, have gained much attention in recent years. Social networks, such as Twitter and Facebook, have attracted a large number of people and are changing humans daily lives. One recent study says that, until 2019, active social media users have passed the 3.8 billion and these users spend over 2 hours per day [1]. Our working environment is also significantly influenced by the information revolution. The applicability of various information systems (IS), such as data management system, management information system and customer relationship management, has contributed to the improvement of productivity and efficiency. However, it is usually a *double-edged sword*. As we are obsessed in the prosperity of information age, some potential security issues arrive unconsciously. For instance, due to the openness of social media, some Twitter users who are induced by the commercial benefits, are spreading the rumors to misguide the public opinion, especially as some specific big events are coming. Bunches of fraudulent accounts also exist in Facebook to do harm to the community eco-system. A survey says that millions of people fell for Facebook scams in 2014: they lost money, reputation and even their jobs after simply clicking on the wrong social media link [2]. Similarly, with the popularity of office automation, organizations suffer from insider threats which originate from people that have been given access rights to an IS and misuse their privileges, thus violating the IS security policy of the organization [3]. The 2018 U.S. State of Cybercrime Survey indicates that 25% of the cyberattacks are committed

by insiders [4]. Information revolution brings us not only benefits but also security issues. Therefore, it becomes a challenging problem how to detect these fraudulent web users or malicious insiders. For the sake of simplicity, we call the malicious users in social media platforms as *web frauds* and denote as *insiders* the employees who take advantage of their access to inflict harm on an organization.

In practice, to flag out these fraudsters, there are some experienced investigators on each social network platform. Usually, they undergo several years of professional training and have a good discrimination for misbehaviors. However, facing the floods of Tweets and thousands of fresh news on Facebook walls every day, limited human labor investigation seems to be trivial. More importantly, the behavior patterns of web frauds are prone to be dynamic in nature. As a result, it could incur a high false-alarm rate if a fixed set of empirical rules are applicable to the detection. In addition, except for some objective criteria, human decisions are often subject to a variety of conscious and unconscious biases which easily bring a totally different judge for the same activity. Therefore, only depending on human investigation, web fraud detection is infeasible. Insider threat detection mostly replies on the exploitation of audit data. The dynamic, subtle, and multi-phase nature of insider attacks makes detection extremely difficult. So, insider threat detection also faces the similar issues: time-evolving behavior patterns and a tremendous volume of auditing records which can not be dealt with by limited human labors. Overall, it is very critical to build some data-driven tools which can provide insights into the existing investigation processes and aid human investigators to make the final decision.

Nowadays, machine learning models have been widely used in automating various tasks traditionally performed by humans such as character recognition, image classification, sentiment analysis and translation. Also, machine learning techniques may offer an

unprecedented opportunity to develop a dynamic fraud detection framework with a focus on time-evolving misbehaviors. However, the effectiveness of such frameworks is challenged by the following issues:

- Unavailability of sufficient labeled data — traditional machine learning approaches usually require a balanced training dataset consisting of positive (normal) and negative (abnormal) samples. In practice, however, there are far fewer negative labeled samples than positive ones.

- Lack of high quality labels — the labeled training records often have the time gap between the time that fraudulent users commit fraudulent actions and the time that they are suspended by the platforms.

- Time-evolving nature — users are changing their behaviors over time.

First, a tremendous volume of negative samples (frauds) are usually unavailable in practice although, in some cases, few of them could be provided. As a result, with an extremely imbalanced dataset or even a one-class dataset, it would be infeasible to train a detection model in a traditional supervised manner. Second, in dynamic fraud detection, the misbehavior labeling is often delayed due to the limited discrimination capability or the cautiousness to judge a fraud: considering low false-alarm, platform administrators would rather conduct a longer observation for a suspicious user than judging it as a fraud immediately. As a consequence, at the testing phase, the trained model can not detect the misbehaviors in real time due to the late-response labels. Third, normal users and frauds are both changing their behaviors over time. Meanwhile, following the changes of normal users, frauds adapt their behavior patterns to avoid the detection, which makes the abnormal fraudulent messages more subtle and difficult to capture.

1.2 Overview and Summary of Contributions

This dissertation is carried out around one general question of "*how do we leverage machine learning techniques to build a model for dynamic fraud or insider threat detection*?" More specifically, this dissertation addresses the following challenges:

- How do we build a machine learning model with only positive samples for dynamic fraud detection? (Chapter 3)
- How do we avail few negative samples to improve the discrimination capability of the trained model for misbehavior detection? (Chapter 4)
- How do we adapt the late-response labels for an early detection in a dynamic manner? (Chapter 5)
- How do we develop a multi-scale fraud detection method to capture the subtle and time-evolving misbehavior? (Chapter 6, Chapter 7)

The overall structure of the dissertation is as follows. Chapter 2 introduces some basic preliminaries and related work. The main technical contributions are discussed in Chapter 3-7. In Chapter 3, we present a novel one-class adversarial neural net for web fraud detection with only positive data. To involve few negative samples to enhance the detection capability of the trained model, in Chapter 4, we propose a few-shot learning framework for insider fraud detection. With late-response labels, Chapter 5 delivers a neural survival analysis model for web fraud early detection. Chapter 6 outlines a neural hierarchical temporal point process model for insider fraud detection, in which it captures the abnormal fraudulent messages in a multi-scale, intra-session and inter-session. Chapter 7 provides a Dirichlet Hawkes process model for insider fraud detection in which *user mode*s are adaptively generated to capture

the time-evolving behavior in a multi-scale manner. Chapter 8 delivers the final conclusions and future works.

1.3 A One-class Fraud Detection Model with Generative Adversarial Neural Net

Wikipedia is a multilingual online encyclopedia website: users can freely check or re-edit any wikipedia pages they are interested in; the edits contributed by users could be reverted by the administrators but the edit reversion does not mean that this user is fraudulent since the misleading edit may be due to lack of the domain knowledge. Therefore, in practice, for fraud detection on Wikipedia, the biggest challenge is lack of two-class labels: although it is not so much difficult to discriminate between right and wrong edits, however, it should be very challenging to claim one user as a *fraud* because you can not mark someone as *fraud* only depending on several edit reversions that may be led by shortage of domain knowledges. In Chapter 3, we propose a one-class fraud detection framework with only positive samples. The key contributions of this chapter are as follows:

- To capture the user's dynamic editing behavior, we develop an LSTM-autoencoder to embed the variational-length editing sequence into a fixed-length latent vector.
- We pretrain a multilayer perceptron (MLP) neural network to capture the empirical distribution of positive samples.
- We propose a complementary generative adversarial net (GAN) model for one-class fraud detection: rather than the distribution of positive samples, generator aims to approaching the complementary distribution of positive samples and, instead of fooling the discriminator, the iterative training procedure is to equip the discriminator with

a strong capability to distinguish positive samples and the complementary of positive samples, i.e. potential negative samples.

1.4 A Few-shot Learning Model for Insider Fraud Detection

Chapter 3 focuses on the scenario that there are only positive samples available. Noticeably, another case does exist as well that, besides positive samples, there are few negative samples while they can not support a supervised learning procedure due to its small volume. Therefore, it has become a challenge how to adjust these negative samples into the training phase. Some one-class models take these negative samples as a validation dataset to tune the decision boundary in the training phase. In contrast, Chapter 4 introduces a novel few-shot learning framework to involve few negative samples for insider fraud detection in a different way. The key contributions of this chapter are as follows:

- We propose a two phase training framework to improve the performance of insider fraud detection.
- Second, unlike most of the existing work for insider fraud detection that only consider the activity type information, we explicitly encode the activity time information into the model via the input representations.
- Evaluation results show that, compared with typical baselines, such as one-class SVM and isolation forest, the proposed few-shot learning framework achieves a better performance.

1.5 A Neural Survival Analysis Model for Fraud Early Detection

Lack of high-quality labels is another challenge for the web fraud detection, i.e. there is usually a time gap between the times when a user conduct a misbehavior and it is suspended by the platform. To reduce the time gap and flag the fraudulent users as early as possible, Chapter 5 proposes a neural survival analysis model for fraud early detection. The key distributions of this chapter are as follows:

- We propose a novel survival analysis model to capture users' time-varying behavior patterns across discrete timestamps in which Recurrent Neural Network is used to learn the hazard rates along the time.
- We revise the likelihood loss function to adapt the training data with late response labels for fraud early detection.
- We conduct the evaluation on two real-world datasets and the proposed model outperforms state-of-the-art fraud detection approaches.

1.6 A Hierarchical Temporal Point Process model for Insider Fraud Detection

The time-evolving nature of insider attacks makes detection extremely difficult. Rather than a single-and-coarse scale framework, which has been discussed in the aforementioned chapters, Chapter 6 involves continuous physical time and proposes a hierarchical neural temporal point process model by combining the temporal point processes and recurrent neural networks for insider threat detection. This model is capable of capturing a general nonlinear dependency over the action history in two inherent levels: intra-session and inter-session. The key contributions are as follows:

- Time point process is used to model the temporal dynamics in a activity streaming.

- The proposed model can capture the time information in a multi-scale manner.
- Evaluation results show that, compared with some state-of-the-art baselines, the proposed hierarchical time point process model achieves a better performance.

1.7 A Dirichlet Hawkes Process Model for Insider Fraud Detection

Chapter 7 involves the Dirichlet process and proposes a dynamic mixture model in which *user mode* can be adaptively generated to adjust the users' time-varying behavior patterns. Noticeably, the model proposed in Chapter 6 is session-oriented while the model proposed in Chapter 7 is activity-oriented. Therefore, compared with the former, the latter should be more flexible and sensitive in capturing the time-evolving behavior patterns of users for dynamic fraud detection. The key contributions of this work are as follows:

- We develop an unsupervised dynamic mixture model for fraud detection.
- The proposed model can effectively capture the sequence of user activities with unbounded number of user modes.
- Evaluation results demonstrate the effectiveness of our model for the dynamic fraud detection on an activity streaming.

2 Preliminaries

2.1 Related Work

In this section, we briefly summarize the related work in two aspects: application scenarios and techniques.

2.1.1 Fraud Detection

Many fraud detection techniques have been developed in recent years [5, 6, 7, 8, 9], including content-based approaches and graph-based approaches. The content-based approaches extract content features, (i.e., text, URL), to identify malicious users from user activities on social networks [10]. Research in [11] focused on predicting whether a Wikipedia user is a vandal by identifying a set of behavior features based on user edit-patterns. To improve detection accuracy and avoid manual feature construction, a multi-source long-short term memory network (M-LSTM) was proposed to detect vandals [12]. Meanwhile, graph-based approaches identify frauds based on network topologies. Often based on unsupervised learning, the graph-based approaches consider fraud as anomalies and extract various graph features associated with nodes, edges, ego-net, or communities from the graph [5, 13, 8, 14].

2.1.2 Fraud Early Detection

The misleading or fake information spread by malicious users could lead to catastrophic consequences because the openness of online social media enables the information to be spread in a timely manner. Therefore, detecting fake information or malicious users is a critical research topic [8, 15, 14, 13, 9]. In recent years, extensive studies focus on the

rumor early detection [16, 17]. Besides early detecting the fake information, early detecting the malicious users who create the fake information is also important. [11, 12] aim to early detect vandals in Wikipedia. All the existing approaches adopt classification models for fraud early detection. In this work, we combine the survival analysis with RNN to predict whether a user is a fraudster.

2.1.3 Insider Threat Detection

Insider attacks are increasingly sophisticated cyber-attacks by people that have the capability, intent, and domain knowledge about the system. Given different attack intentions, insiders can be categorized into three types: traitors, masqueraders and unintentional perpetrators. Traitors usually misuse their privileges to commit malicious activities; masqueraders refer to the people who often conduct illegal actions on behalf of legitimate employees of an institute; unintentional perpetrators are benign users who unintentionally make mistakes [18].

An insider attack is often hidden among a huge number of normal activities. The subtle and dynamic nature of insider attacks makes detection extremely difficult. It becomes a key challenge on how to describe and capture the insider' behaviors for insider threat detection. To handle this challenge, in recent years, varieties of insider detection algorithms are proposed. For example, anomaly detection algorithms, such as one-class SVM, are usually adopted for insider detection [19]. Meanwhile, rather than detecting the insiders, several approaches are proposed to detect fine-grained insider threats, e.g., whether there are malicious activities in a day. The idea is that we treat the employee's actions over a period of time as a sequence. The sequences that are frequently observed are normal behavior, while the sequences that are seldom observed are abnormal behavior that could be from insiders.

To model the user activity sequences, researchers adopt Hidden Markov Models (HMMs) to capture the behaviors of normal employees [20]. As a result, any employee activity sequences with low probability predicted by HMMs could indicate an abnormal sequence. Recently, the recurrent neural network is also adopted to detect malicious sequences based on manually designed input features [21]. In addition, some other supervised and unsupervised learning algorithms, such as Self Organizing Maps (SOM) and Decision Trees (DT), are also employed for insider threat detection [22]. However, there is still no discussion on how to enhance the performance of insider threat detection with a few observed insiders. To handle this problem, in this work, we propose a few-shot learning based method for insider threat detection.

2.1.4 One-class Classification

One-class classification (OCC) algorithms aim to build classification models when only one class of samples are observed and the other class of samples are absent [23], which is also related to the novelty detection [24]. One-class support vector machine (OCSVM), as one of widely adopted for one class classification, aims to separate one class of samples from all the others by constructing a hyper-sphere around the observed data samples [25, 26]. Other traditional classification models also extend to the one-class scenario. For example, one-class nearest neighbor (OCNN) [27] predicts the class of a sample based on its distance to its nearest neighbor in the training dataset. One-class Gaussian process (OCGP) chooses a proper GP prior and derives membership scores for one-class classification [28]. However, OCNN and OCGP need to set a threshold to detect another class of data. The threshold is either set by a domain expert or tuned based on a small set of two-class labeled data. In this work, we propose a framework that combines LSTM-Autoencoder and GAN to detect vandals with only knowing benign users. To our best knowledge, this is the first work that

examines the use of deep learning models for fraud detection when only one-class training data is available. Meanwhile, comparing to existing one-class algorithms, our model trains a classifier by generating a large number of "novel" data and does not require any labeled data to tune parameters.

2.1.5 Few-shot Learning

Few-shot learning (FSL) aims to learn new tasks from a limited amount of supervised information [29]. FSL plays as a testbed for AI since human can learn tasks by only observing a few samples. Meanwhile, FSL is also suitable for tasks where labeled data is hard to acquire. Due to the importance of generalization capability for a learning model, FSL has attracted increasing attention these years [30].

The few-shot learning algorithms can be categorized into metric-based and meta-learning based approaches. The main idea of metric-based approaches is to predict a new sample by comparing the sample with the limited observed samples. For example, Siamese Neural Network consists of twin networks which are identical to each other and learn the relationship between pairs of input data by metric-learning losses [31]. Matching Network predicts the new sample by using a weighted K-nearest neighbor classifier measured by the cosine distance [32]. Relation Network combines the embeddings of a new sample with the support samples and adopts a neural network to learn a distance between a new sample and each class of samples [33]. The meta-learning approaches are usually called learning to learn algorithms, which learn a new task by the limited samples and the meta knowledge extracted across tasks by a meta-learner. There are two classical meta learning approaches, Memory-Augmented Neural Network (MANN) and Model-Agnostic Meta-Learning (MAML). MANN adopts a controller (e.g., LSTM) to interact with an external memory that stores class label

information with a few labeled data [34]. MAML creates a model agnostic method that has a meta objective being optimized by a small number of gradient updates, leading to fast learning on a new task [35].

Unlike many few-shot learning tasks with $C(C > 2)$ classes, the insider threat detection task only has two classes and limited malicious samples. As a result, the comparison pairs that can be generated from the dataset are also limited. In this work, we propose a framework that is able to leverage a large amount of audit data and a small number of malicious samples.

2.1.6 Survival Analysis

Survival analysis is to analyze and model the data where the outcome is the time until the occurrence of an event of interest [36]. In survival analysis, the occurrence of an event is not always observed in an observation window, which is called *censored*.

Survival analysis is a widely-used tool in health data analysis [37, 38, 39] and has been applied to various application fields, such as students dropout time [40], web user return time [41, 42, 43], and user check-in time prediction [44]. To our knowledge, the survival analysis has not been investigated in the context of fraud detection.

Many approaches have been proposed to make use of censored data as well as the event data. The Cox proportional hazards model (CPH) [45] is the most widely-used model for survival analysis. CPH is semi-parametric and does not make any assumption about the distribution of event occurrence time. It is typically learned by optimizing a partial likelihood function. However, CPH makes strong assumptions that the log-risk of an event is a linear combination of covariates, and the base hazard rate is constant over time. Some researchers proposed parametric censored models, which assume the event occurrence time follows a

specific distribution such as exponential, log-logistic or Weibull [46, 38, 47]. However, it is common that the specific parametric assumptions are not satisfied in real data.

In recent years, researchers adopt neural networks to model the survival distribution [48, 49, 50, 51, 52]. For example, [48, 49] combine the feed-forward neural network with the classical Cox proportional hazard model. Although using the deep neural network can improve the capacity of models, these studies still assume that the base hazard rate is constant. [50] transfers the problem of learning the distribution of survival time to a discretized-time classification problem and adopts the deep feed forward neural network to predict the survival time. [51] adopts a conditional generative adversarial network to predict the event time conditioned on covariates, which implicitly specifies a time-to-event distribution via sampling. However, the existing models cannot handle the time-varying covariates. In this work, we adopt the RNN to take the time-varying covariates as inputs and fit the time-to-event distribution without making any of the above assumptions.

We also notice that some studies adopt RNN to model the time-to-event distributions. Those studies mainly focus on modeling the recurrent event instead of the terminated event. For example, [42, 41, 53] adopt RNN to model the web user return times, which focus on the recurrent event data other than the censored data. Hence, RNN is to capture the gap time between user active sessions. Moreover, unlike the existing work that focuses on "just-in-time" prediction, we adapt the survival analysis for fraud early detection in the scenario where training data contains late response labels.

2.1.7 Temporal Point Process

A temporal point process (TPP) is a stochastic process composed of a time series of events that occur in continuous time [54]. The temporal point process is widely used for

modeling the sequence data with time information, such as health-care analysis, earthquakes and aftershocks modeling and social network analysis [55, 56, 57]. The traditional methods of temporal point processes usually make parametric assumptions about how the observed events are generated, e.g., by Poisson processes or self-exciting point processes. If the data do not follow the prior knowledge, the parametric point processes may have poor performance. To address this problem, researchers propose to learn a general representation of the dynamic data based on neural networks without assuming parametric forms [42, 58]. Those models are trained by maximizing log likelihood. Recently, there are also emerging works incorporating the objective function from generative adversarial network [59, 60] or reinforcement learning [61] to further improve the model performance. However, the current TPP models only focus on one granularity of time. In our scenario, we propose a hierarchical RNN framework to model the multi-scale time information.

2.2 Datasets

We evaluate the proposed models, which will be detailed in the following chapters, mainly on two real-world datasets and one synthetic dataset. In this section, we provide a basic introduction for all of them.

2.2.1 Wikipedia

Wikipedia dataset origins from UMDWikipedia dataset [11] in which there are around 770K edits from Jan 2013 to July 2014 (19 months) with 17105 vandals and 17105 benign users and Each user edits a sequence of Wikipedia pages.

2.2.2 Twitter

We randomly collect 51608 Twitter users on August 13, 2017, monitor the user statuses every three days until October 13, 2017, and get the data with 21 timestamps. For each user, at each timestamp, the following 5 features are recorded: 1) the number of followers, 2) the number of followees, 3) the number of tweets, 4) the number of liked tweets, and 5) the number of public lists that the user is a member of. During this period, 7790 users (15.0%) are suspended; the remaining 43818 users (85.0%) are still active.

2.2.3 CERT

We adopt the CERT Insider Threat Dataset [62], which is the only comprehensive dataset publicly available for evaluating the insider threat detection. This dataset consists of five log files that record the computer-based activities for all employees, including **logon.csv** that records the logon and logoff operations of all employees, **email.csv** that records all the email operations (send or receive), **http.csv** that records all the web browsing (visit, download, or upload) operations, **file.csv** that records activities (open, write, copy or delete) involving a removable media device, and **decive.csv** that records the usage of a thumb drive (connect or disconnect). The CERT dataset also has the ground truth that indicates the malicious activities committed by insiders. We use the latest version (r6.2) of CERT dataset that contains 3995 benign employees and 5 insiders.

3 One-Class Adversarial Nets for Fraud Detection

Problem Statement: *How do we build a machine learning model with only positive samples for dynamic fraud detection?*

In this chapter, we will first review the background of the problem, then formulate the problem and present the proposed method. The real-world Wikipedia dataset will be availed to evaluate the effectiveness of the proposed method by comparing with the state-of-the-art baselines.

3.1 Introduction

Online platforms such as online social networks (OSNs) and knowledge bases play a major role in online communication and knowledge sharing. However, there are various malicious users who conduct various fraudulent actions, such as spams, rumors, and vandalism, imposing severe security threats to OSNs and their legitimate participants. To protect legitimate users, most Web platforms have tools or mechanisms to block malicious users. For example, Wikipedia adopts ClueBot NG to detect and revert obvious bad edits, thus helping administrators to identify and block vandals.

Detecting malicious users has also attracted increasing attention in the research community [63, 64, 15, 11, 12]. However, these detection models are trained over a training dataset that consists of both positive data (benign users) and negative data (malicious users). In practice, there are often no or very few records from malicious users in the collected training data. Manually labeling a large number of malicious users is tedious.

In this work, we tackle the problem of identifying malicious users when only benign

users are observed. The basic idea is to adopt a generative model to generate malicious users with only given benign users. Generative adversarial networks (GAN) as generative models have demonstrated impressive performance in modeling the real data distribution and generating high quality synthetic data that is similar to real data [65, 66]. However, given benign users, a regular GAN model is unable to generate malicious users.

We develop one-class adversarial nets (OCAN) for fraud detection. During training, OCAN contains two phases. First, OCAN adopts the LSTM-Autoencoder [67] to encode the benign users into a hidden space based on their online activities, and the encoded vectors are called benign user representations. Then, OCAN trains improved generative adversarial nets in which the discriminator is trained to be a classifier for distinguishing benign users and malicious users with the generator producing potential malicious users. To this end, we adopt the idea of bad GAN [68] that the generator is trained to generate complementary samples instead of matching the original data distribution. The generator of the complementary GAN aims to generate samples that are complementary to the representations of benign users, i.e., the potential malicious users. We revise the objective function of the discriminator in the regular GAN to achieve one-class classification. The discriminator is trained to separate benign users and complementary samples. Since the behaviors of malicious users and that of benign users are complementary, we expect the discriminator can distinguish benign users and malicious users. By combining the encoder of LSTM-Autoencoder and the discriminator of the complementary GAN, OCAN can accurately predict whether a new user is benign or malicious based on his online activities.

The advantages of OCAN for fraud detection are as follows. First, since OCAN does not require any information about malicious users, we do not need to manually compose a mixed training dataset, thus more adaptive to different types of malicious user identification

tasks. Second, different from existing one-class classification models, OCAN generates complementary samples of benign users and trains the discriminator to separate complementary samples from benign users, enabling the trained discriminator to better separate malicious users from benign users. Third, OCAN can capture the sequential information of user activities. After training, the detection model can adaptively update a user representation once the user commits a new action and predict whether the user is a fraud or not dynamically. Note that this chapter is originally from the published work [69].

3.2 Generative Adversarial Nets

Generative adversarial nets (GAN) are generative models that consist of two components: a generator G and a discriminator D. Typically, both G and D are multilayer neural networks. $G(\mathbf{z})$ generates fake samples from a prior $p_{\mathbf{z}}$ on a noise variable $\mathbf{z}$ and learns a generative distribution p_G to match the real data distribution p_{data}. On the contrary, the discriminative model D is a binary classifier that predicts whether an input is a real data $\mathbf{x}$ or a generated fake data from $G(\mathbf{z})$. Hence, the objective function of D is defined as:

$$\max_{D} \quad \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))], \tag{3.1}$$

where $D(\cdot)$ outputs the probability that $\cdot$ is from the real data rather than the generated fake data. In order to make the generative distribution p_G close to the real data distribution p_{data}, G is trained by fooling the discriminator not be able to distinguish the generated data from the real data. Thus, the objective function of G is defined as:

$$\min_{G} \quad \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))]. \tag{3.2}$$

Minimizing the Equation 3.2 is achieved if the discriminator is fooled by generated data $G(\mathbf{z})$ and predicts high probability that $G(\mathbf{z})$ is real data.

Overall, GAN is formalized as a minimax game $\min_G \max_D V(G, D)$ with the value function:

$$V(G, D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))]. \tag{3.3}$$

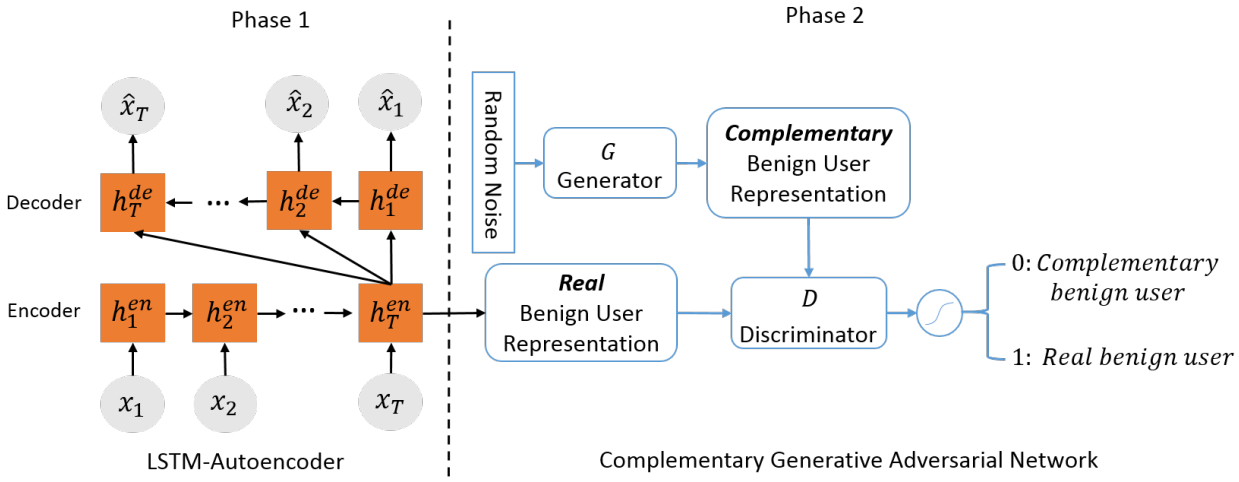


Figure 3.1: The training framework of OCAN

3.3 OCAN: One-Class Adversarial Nets

3.3.1 Framework Overview

OCAN contains two phases during training. The first phase is to learn user representations. As shown in the left side of Figure 3.1, LSTM-Autoencoder is adopted to learn benign user representations from benign user activity sequences. The LSTM-Autoencoder model is a sequence-to-sequence model that consists of two LSTM models as the encoder and decoder respectively. The encoder computes hidden representations of inputs, and the decoder computes the reconstructed inputs based on the hidden representations. The trained LSTM-Autoencoder can capture the salient information of users' activity sequences because

the objective function is to make the reconstructed input close to the original input. The encoder of the trained LSTM-Autoencoder, when deployed for fraud detection, is expected to map the benign users and malicious users to relatively separate regions in the continuous feature space because the activity sequences of benign and malicious users are different.

Given the user representations, the second phase is to train a complementary GAN with a discriminator that can clearly distinguish the benign and malicious users. The generator of the complementary GAN aims to generate complementary samples that are in the low-density area of benign users, and the discriminator aims to separate the real and complementary benign users. The discriminator then has the ability to detect malicious users which locate in separate regions from benign users. The framework of training complementary GAN is shown in the right side of Figure 3.1.

The pseudo-code of training OCAN is shown in Algorithm 1. Given a training dataset M_{benign} that contains activity sequence feature vectors of N benign users, we first train the LSTM-Autoencoder model (Lines 3–9). After training the LSTM-Autoencoder, we adopt the encoder in the LSTM-Autoencoder model to compute the benign user representation (Lines 11–14). Finally, we use the benign user representation to train the complementary GAN (Lines 16–20). For simplicity, we write the algorithm with a minibatch size of 1, i.e., iterating each user in the training dataset to train LSTM-Autoencoder and GAN. In practice, we sample m real benign users and use the generator to generate m complementary samples in a minibatch. In our experiments, the size of minibatch is 32.

Our OCAN moves beyond the naive approach of adopting a regular GAN model in the second phase. The generator of a regular GAN aims to generate the representations of fake benign users that are close to the representations of real benign users. The discriminator of a regular GAN is to identify whether an input is a representation of a real benign user or

a fake benign user from the generator. However, one potential drawback of the regular GAN is that once the discriminator is converged, the discriminator cannot have high confidence on separating real benign users from real malicious users. We denote the OCAN with the regular GAN as OCAN-r and compare its performance with OCAN in the experiment.

Algorithm 1: Training One-Class Adversarial Nets

Inputs : Training dataset $M_{\text{benign}} = \{\mathcal{X}_1, \cdots, \mathcal{X}_N\}$,
Training epochs for LSTM-Autoencoder
$Epoch_{AE}$ and GAN $Epoch_{GAN}$

Outputs: Well-trained LSTM-Autoencoder and complementary GAN

initialize parameters in LSTM-Autoencoder and complementary GAN;
$j \leftarrow 0$;
while $j < Epoch_{AE}$ **do**
  foreach *user* u *in* M_{benign} **do**
    compute the reconstructed sequence of user activities by LSTM-Autoencoder (Eq. 3.4, 3.6, and 3.7);
    optimize the parameters in LSTM-Autoencoder with the loss function Eq. 3.8;
  end
  $j \leftarrow j + 1$;
end
$\mathcal{V} = \emptyset$;
foreach *user* u *in* M_{benign} **do**
  compute the benign user representation $\mathbf{v}_u$ by the encoder of LSTM-Autoencoder (Eq. 3.4, 3.5);
  $\mathcal{V} += \mathbf{v}_u$;
end
$j \leftarrow 0$;
while $j < Epoch_{GAN}$ **do**
  foreach *benign user representation* $\mathbf{v}_u$ *in* $\mathcal{V}$ **do**
    optimize the discriminator D and generator G with loss functions Eq. 3.14, 3.12, respectively;
  end
end
return well-trained LSTM-Autoencoder and complementary GAN

3.3.2 LSTM-Autoencoder for User Representation

The first phase of OCAN is to encode users to a continuous hidden space. Since each online user has a sequence of activities (e.g., edit a sequence of pages), we adopt LSTM-Autoencoder to transform a variable-length user activity sequence into a fixed-dimension

user representation. Formally, given a user u with T activities, we represent the activity sequence as $\mathcal{X}_u = (\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$ where $\mathbf{x}_t \in \mathbb{R}^d$ is the t-th activity feature vector.

Encoder: The encoder encodes the user activity sequence $\mathcal{X}_u$ to a user representation with an LSTM model:

$$\mathbf{h}_t^{en} = LSTM^{en}(\mathbf{x}_t, \mathbf{h}_{t-1}^{en}), \tag{3.4}$$

where $\mathbf{x}_t$ is the feature vector of the t-th activity; $\mathbf{h}_t^{en}$ indicates the t-th hidden vector of the encoder.

The last hidden vector $\mathbf{h}_T^{en}$ captures the information of a whole user activity sequence and is considered as the user representation $\mathbf{v}$:

$$\mathbf{v} = \mathbf{h}_T^{en}. \tag{3.5}$$

Decoder: In our model, the decoder adopts the user representation $\mathbf{v}$ as the input to reconstruct the original user activity sequence $\mathcal{X}$:

$$\mathbf{h}_t^{de} = LSTM^{de}(\mathbf{v}, \mathbf{h}_{t-1}^{de}), \tag{3.6}$$

$$\hat{\mathbf{x}}_t = f(\mathbf{h}_t^{de}), \tag{3.7}$$

where $\mathbf{h}_t^{de}$ is the t-th hidden vector of the decoder; $\hat{\mathbf{x}}_t$ indicates the t-th reconstructed activity feature vector; $f(\cdot)$ denotes a neural network to compute the sequence outputs from hidden vectors of the decoder. Note that we adopt $\mathbf{v}$ as input of the whole sequence of the decoder, which has achieved great performance on sequence-to-sequence models [70].

The objective function of LSTM-Autoencoder is:

$$\mathcal{L}_{(AE)}(\hat{\mathbf{x}}_t, \mathbf{x}_t) = \sum_{t=1}^{T} (\hat{\mathbf{x}}_t - \mathbf{x}_t)^2, \tag{3.8}$$

where $\mathbf{x}_t$ ($\hat{\mathbf{x}}_t$) is the t-th (reconstructed) activity feature vector. After training, the last hidden vector of encoder $\mathbf{h}_T$ can reconstruct the sequence of user feature vectors. Thus, the representation of user $\mathbf{v} = \mathbf{h}_T^{en}$ captures the salient information of user behavior.

3.3.3 Complementary GAN

The generator G of complementary GAN is the same as that of the bad GAN in [68]. Basically, it is a feedforward neural network where its output layer has the same dimension as the user representation $\mathbf{v}$. Formally, we define the generated samples as $\tilde{\mathbf{v}} = G(\mathbf{z})$. Unlike the generator in a regular GAN which is trained to match the distribution of the generated fake benign user representation with that of benign user representation p_{data}, the generator G of complementary GAN learns a generative distribution p_G that is close to the complementary distribution p^* of the benign user representations, i.e., $p_G = p^*$.

Following [68], we define the complementary distribution p* as:

$$p^*(\tilde{\mathbf{v}}) = \begin{cases} \frac{1}{\tau} \frac{1}{p_{\text{data}}(\tilde{\mathbf{v}})} & \text{if } p_{\text{data}}(\tilde{\mathbf{v}}) > \epsilon \text{ and } \tilde{\mathbf{v}} \in \mathcal{B}_{\mathbf{v}} \\ C & \text{if } p_{\text{data}}(\tilde{\mathbf{v}}) \leq \epsilon \text{ and } \tilde{\mathbf{v}} \in \mathcal{B}_{\mathbf{v}}, \end{cases} \tag{3.9}$$

where ϵ is a threshold to indicate whether the generated samples are in high-density regions; τ is a normalization term; C is a small constant; $\mathcal{B}_{\mathbf{v}}$ is the space of user representation. To make the generative distribution p_G close to the complementary distribution p^*, the complementary generator G is trained to minimize the KL divergence between p_G and p^*.

Based on the definition of KL divergence, the objective function is:

$$\begin{aligned}\mathcal{L}_{KL(p_G\|p^*)} &= -\mathcal{H}(p_G) - \mathbb{E}_{\tilde{\mathbf{v}}\sim p_G} \log p^*(\tilde{\mathbf{v}}) \\ &= -\mathcal{H}(p_G) + \mathbb{E}_{\tilde{\mathbf{v}}\sim p_G} \log p_{\text{data}}(\tilde{\mathbf{v}}) \mathbb{1}[p_{\text{data}}(\tilde{\mathbf{v}}) > \epsilon] \\ &+ \mathbb{E}_{\tilde{\mathbf{v}}\sim p_G} (\mathbb{1}[p_{\text{data}}(\tilde{\mathbf{v}}) > \epsilon] \log \tau - \mathbb{1}[p_{\text{data}}(\tilde{\mathbf{v}}) \leq \epsilon] \log C),\end{aligned} \tag{3.10}$$

where $\mathcal{H}(\cdot)$ is the entropy, and $\mathbb{1}[\cdot]$ is the indicator function. The last term of Equation 3.10 can be omitted because both τ and C are constant terms and the gradients of the indicator function $\mathbb{1}[\cdot]$ with respect to parameters of the generator are mostly zero.

Meanwhile, following [68], the complementary generator G adopts the feature matching loss [71] to ensure that the generated samples are constrained in the space of user representation $\mathcal{B}_{\mathbf{v}}$.

$$\mathcal{L}_{\text{fm}} = \| \mathbb{E}_{\tilde{\mathbf{v}}\sim p_G} f(\tilde{\mathbf{v}}) - \mathbb{E}_{\mathbf{v}\sim p_{\text{data}}} f(\mathbf{v}) \|_2^2, \tag{3.11}$$

where $f(\cdot)$ denotes the output of an intermediate layer of the discriminator used as a feature representation of $\mathbf{v}$.

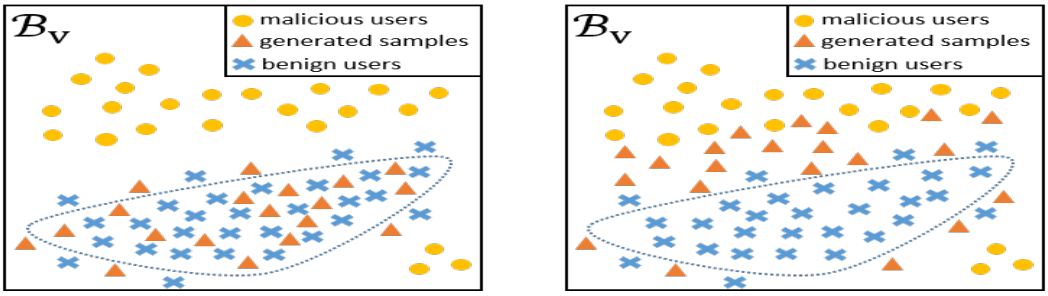
Thus, the complete objective function of the generator is defined as:

$$\begin{aligned}\min_G \quad & -\mathcal{H}(p_G) + \mathbb{E}_{\tilde{\mathbf{v}}\sim p_G} \log p_{\text{data}}(\tilde{\mathbf{v}}) \mathbb{1}[p_{\text{data}}(\tilde{\mathbf{v}}) > \epsilon] \\ & + \| \mathbb{E}_{\tilde{\mathbf{v}}\sim p_G} f(\tilde{\mathbf{v}}) - \mathbb{E}_{\mathbf{v}\sim p_{\text{data}}} f(\mathbf{v}) \|_2^2 .\end{aligned} \tag{3.12}$$

Overall, the objective function of the complementary generator aims to let the generative distribution p_G close to the complementary samples p^*, i.e., $p_G = p^*$, and make the generated samples from different regions (but in the same space of user representations) than those of the benign users.

Figure 3.2 illustrates the difference of the generators of regular GAN and complemen-

tary GAN. The objective function of the generator of regular GAN in Equation 3.2 is trained to fool the discriminator by generating fake benign users similar to the real benign users. Hence, as shown in Figure 3.3.3, the generator of regular GAN generates the distribution of fake benign users that have the similar distribution of real benign users in the feature space. On the contrary, the objective function of the generator of complementary GAN in Equation 3.12 is trained to generate complementary samples that are in the low-density regions of benign users (shown in Figure 3.3.3).



(a) Regular GAN (b) Complementary GAN

Figure 3.2: Demonstrations of the ideal generators of regular GAN and complementary GAN. The blue dot line indicates the high density regions of benign users.

To optimize the objective function of generator, we need to approximate the entropy of generated samples $\mathcal{H}(p_G)$ and the probability distribution of real samples p_{data}. To minimize $-\mathcal{H}(p_G)$, following [68], we adopt the pull-away term (PT) proposed by [72] that encourages the generated feature vectors to be orthogonal. The PT term increases the diversity of generated samples and can be considered as a proxy for minimizing $-\mathcal{H}(p_G)$. The PT term is defined as

$$\mathcal{L}_{PT} = \frac{1}{N(N-1)} \sum_{i}^{N} \sum_{j \neq i}^{N} \left(\frac{f(\tilde{\mathbf{v}}_i)^T f(\tilde{\mathbf{v}}_j)}{\| f(\tilde{\mathbf{v}}_i) \| \| f(\tilde{\mathbf{v}}_j) \|} \right)^2, \tag{3.13}$$

where N is the size of a mini-batch.

The probability distribution of real samples p_{data} is usually unavailable, and approximating p_{data} is computationally expensive. In this paper, we adopt the approach proposed

by [73] that a discriminator from a regular GAN can detect whether the data from the real data distribution p_{data} or from the generator's distribution. The basic idea is that the discriminator is able to detect whether a sample is from the real data distribution p_{data} or from the generator when the generator is trained to generate samples that are close to real benign users. Hence, the discriminator is sufficient to identify the data points that are above a threshold of p_{data} during training. We separately train a regular GAN model based on benign user representations and use the discriminator of the regular GAN as a proxy to evaluate $p_{\text{data}}(\tilde{\mathbf{v}}) > \epsilon$.

The discriminator D takes the benign user representation $\mathbf{v}$ and generated user representation $\tilde{\mathbf{v}}$ as inputs and tries to distinguish $\mathbf{v}$ from $\tilde{\mathbf{v}}$. As a classifier, D is a standard feedforward neural network with a softmax function as its output layer, and we define the objective function of D as:

$$\begin{aligned} \max_{D} \quad & \mathbb{E}_{\mathbf{v}\sim p_{\text{data}}}[\log D(\mathbf{v})] + \mathbb{E}_{\tilde{\mathbf{v}}\sim p_G}[\log(1 - D(\tilde{\mathbf{v}}))] + \\ & \mathbb{E}_{\mathbf{v}\sim p_{\text{data}}}[D(\mathbf{v}) \log D(\mathbf{v})]. \end{aligned} \tag{3.14}$$

Different from the objective function of the discriminator introduced in the bad GAN for the purpose of semi-supervised learning, we revise the objective function of D in our complementary GAN based on the regular GAN. The first two terms in Equation 3.14 are the objective function of discriminator in the regular GAN model. Therefore, the discriminator of complementary GAN is trained to separate the benign users and complementary samples. The last term in Equation 3.14 is a conditional entropy term which encourages the discriminator to detect real benign users with high confidence. Then, the discriminator is able to separate the benign and malicious users clearly.

Although the objective functions of the discriminators of regular GAN and comple-

mentary GAN are similar, the capabilities of discriminators of regular GAN and complementary GAN for malicious detection are different. The discriminator of regular GAN aims to separate the benign users and generated fake benign users. However, after training, the generated fake benign users locate in the same regions as the real benign users (shown in Figure 3.3.3). The probabilities of real and generated fake benign users predicted by the discriminator of regular GAN are all close to 0.5. Thus, giving a benign user, the discriminator cannot predict the benign user with high confidence. On the contrary, the discriminator of complementary GAN is trained to separate the benign users and generated complementary samples. Since the generated complementary samples have the same distribution as the malicious users (shown in Figure 3.3.3), the discriminator of complementary GAN can also detect the malicious users.

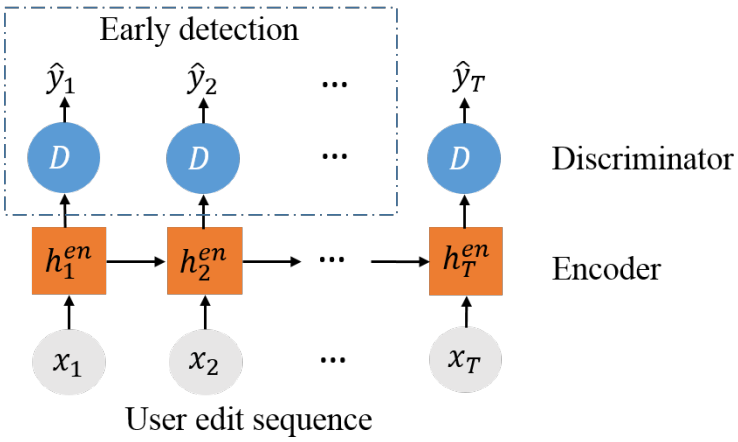


Figure 3.3: The fraud detection model

3.4 Fraud Detection Model

Although the training procedure of OCAN contains two phases that train LSTM-Autoencoder and complementary GAN successively, the fraud detection model is an end-to-end model. We illustrate its structure in Figure 3.3. To detect a malicious user, we first

compute the user representation $\mathbf{v}_u$ based on the encoder in the LSTM-Autoencoder model (Equations 3.4 and 3.5). Then, we predict the user label based on the discriminator of complementary GAN, i.e., $p(\hat{y}_u|\mathbf{v}_u) = D(\mathbf{v}_u)$.

Early fraud detection: The upper-left region of Figure 3.3 shows that our OCAN model can also achieve early detection of malicious users. Given a user u, at each step t, the hidden states $\mathbf{h}_{u_t}^{en}$ are updated until the t-th step by taking the current feature vector $\mathbf{x}_{u_t}$ as input and are able to capture the user behavior information until the t-th step. Thus, the user representation at the t-th step is denoted as $\mathbf{v}_{u_t} = \mathbf{h}_{u_t}^{en}$. Finally, we can use the discriminator D to calculate the probability $p(\hat{y}_{u_t}|\mathbf{v}_{u_t}) = D(\mathbf{v}_{u_t})$ of the user to be a malicious user based on the current step user representation $\mathbf{v}_t$.

3.5 Experiments

Table 3.1: Vandal detection results (mean±std.) on precision, recall and F1

Input	Algorithm	Precision	Recall	F1
Raw feature vector	OCNN	0.5680 ± 0.0129	0.8646 ± 0.0599	0.6845 ± 0.0184
	OCGP	0.5767 ± 0.0087	0.9000 ± 0.0560	0.7023 ± 0.0193
	OCSVM	0.6631 ± 0.0057	0.9829 ± 0.0011	0.7919 ± 0.0040
User representation	OCNN	0.8314 ± 0.0351	0.8028 ± 0.0476	0.8150 ± 0.0163
	OCGP	$0.8381\pm, 0.0225$	0.8289 ± 0.0374	0.8326 ± 0.0158
	OCSVM	0.6558 ± 0.0058	$\mathbf{0.9590 \pm 0.0096}$	0.7789 ± 0.0064
	OCAN	$\mathbf{0.9067 \pm 0.0615}$	0.9292 ± 0.0348	$\mathbf{0.9010 \pm 0.0228}$
User representation	OCAN-r	0.8673 ± 0.0355	0.8759 ± 0.0529	0.8701 ± 0.0267

3.5.1 Experiment Setup

Dataset: To evaluate OCAN, we conduct our evaluation on Wikipedia dataset, in which we focus on one type of malicious users, i.e., vandals on Wikipedia. We keep those users with the lengths of edit sequence ranging from 4 to 50 and, after that, the dataset contains 10528

benign users and 11495 vandals. To compose the feature vector $\mathbf{x}_t$ of the user's t-th edit, we adopt the following edit features: (1) whether or not the user edited on a meta-page; (2) whether or not the user consecutively edited the pages less than 1 minutes; (3) whether or not the user's current edit page had been edited before; (4) whether or not the user's current edit would be reverted.

Hyperparameters: For LSTM-Autoencoder, the dimension of the hidden layer is 200, and the training epoch is 20. For the complementary GAN model, both discriminator and generator are feedforward neural networks. Specifically, the discriminator contains 2 hidden layers which are 100 and 50 dimensions. The generator takes the 50 dimensions of noise as input, and there is one hidden layer with 100 dimensions. The output layer of the generator has the same dimension as the user representation which is 200 in our experiments. The training epoch of complementary GAN is 50. The threshold ϵ defined in Equation 3.12 is set as the 5-quantile probability of real benign users predicted by a pre-trained discriminator. We evaluated several values from 4-quantile to 10-quantile and found the results are not sensitive.

Repeatability: Our software together with the datasets are available online https://github.com/PanpanZh

3.5.2 Comparison with One-Class Classification

Baselines: We compare OCAN with the following widely used one-class classification approaches:

- One-class nearest neighbors (**OCNN**) [27] labels a testing sample based on the distance from the sample to its nearest neighbors in training dataset and the average distance of those nearest neighbors.

- One-class Gaussian process (**OCGP**) [28] is a one-class classification model based on

Gaussian process regression.

- One-class SVM (**OCSVM**) [25] adopts support vector machine to learn a decision hypersphere around the positive data, and considers samples located outside this hypersphere as anomalies.

For baslines, we use the implementation provided in NDtool. The hyperparameters of baselines set as default values in NDtool. Note that both OCNN and OCGP require a small portion (5% in our experiments) of vandals as a validation dataset to tune an appropriate threshold for vandal detection. However, OCAN does not require any vandals for training and validation. Since the baselines are not sequence models, we compare OCAN to baselines in two ways. First, we concatenate all the edit feature vectors of a user to a *raw feature vector* as an input to baselines. Second, the baselines have the same inputs as the discriminator, i.e., the *user representation* $\mathbf{v}$ computed from the encoder of LSTM-Autoencoder. Meanwhile, OCAN cannot adopt the raw feature vectors as inputs to detect vandals. This is because GAN is only suitable for real-valued data [65].

To evaluate the performance of vandal detection, we randomly select 7000 benign users as the training dataset and 3000 benign users and 3000 vandals as the testing dataset. We report the mean value and standard deviation based on 10 different runs. Table 3.1 shows the means and standard deviations of the precision, recall, F1 score for vandal detection. First, OCAN achieves better performances than baselines in terms of F1 score in both input settings. It means the discriminator of complementary GAN can be used as a one-class classifier for vandal detection. We can further observe that when the baselines adopt the raw feature vector instead of user representation, the performances of both OCNN and OCGP decrease significantly. It indicates that the user representations computed by the encoder of LSTM-Autoencoder capture the salient information about user behavior and can improve

the performance of one-class classifiers. However, we also notice that the standard deviations of OCAN are higher than the baselines with user representations as inputs. We argue that this is because GAN is widely known for difficult to train. Thus, the stability of OCAN is relatively lower than the baselines.

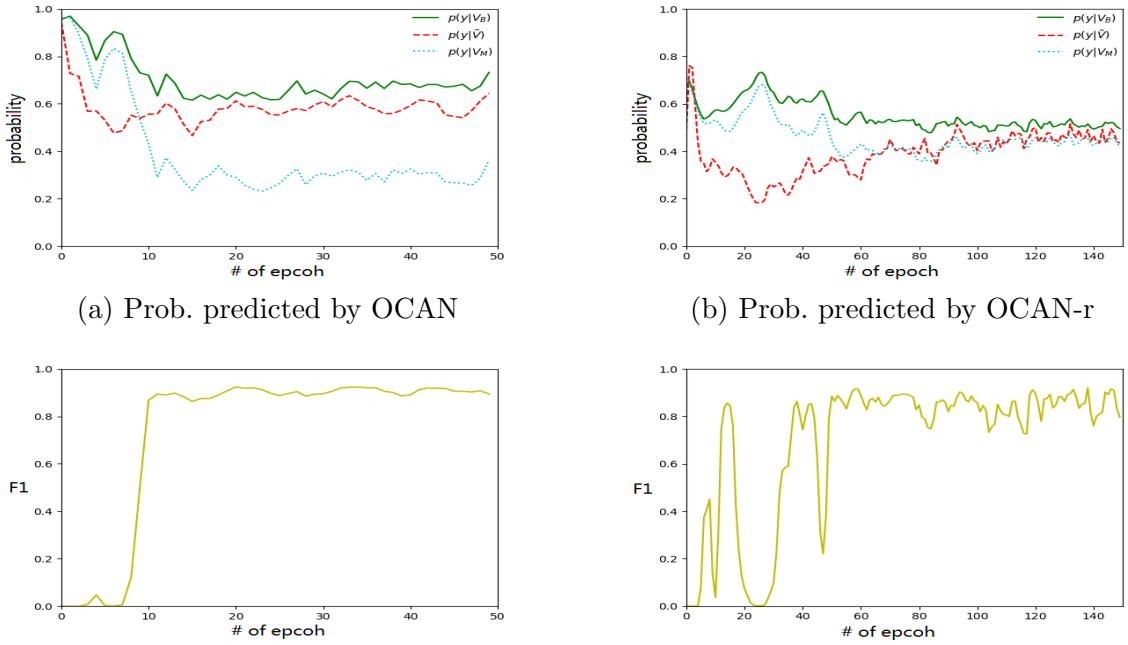
Furthermore, we show the experimental results of OCAN-r, which adopts the regular GAN model instead of the complementary GAN in the second training phase of OCAN, in the last row of Table 3.1. We can observe that the performance of OCAN is better than OCAN-r. It indicates that the discriminator of complementary GAN which is trained on real and complementary samples can more accurately separate the benign users and vandals.

Table 3.2: Early detection results on precision, recall, F1, and the average number of edits before the vandals are blocked

	Vandals	Precision	Recall	F1	Edits
M-LSTM	7000	0.8416	0.9637	0.8985	7.21
	1000	0.9189	0.8910	0.9047	5.98
	400	0.9639	0.6767	0.7951	3.64
	300	0.0000	0.0000	0.0000	0.00
	100	0.0000	0.0000	0.0000	0.00
OCAN	0	**0.8014**	**0.9081**	**0.8459**	**7.23**
OCAN-r	0	0.7228	0.8968	0.7874	7.18

3.5.3 Comparison with M-LSTM for Early Vandal Detection

We further compare the performance of OCAN in terms of early vandal detection with one latest deep learning based vandal detection model, M-LSTM, developed in [12]. Note that M-LSTM assumes a training dataset that contains both vandals and benign users. In our experiments, we train our OCAN with the training data consisting of 7000 benign users and no vandals and train M-LSTM with a training data consisting the same 7000 benign users and a varying number of vandals (from 7000 to 100). For OCAN and M-LSTM, we use the same testing dataset that contains 3000 benign users and 3000 vandals. Note that in OCAN and M-LSTM, the hidden state $\mathbf{h}_t^{en}$ of the LSTM model captures the up-to-date



(a) Prob. predicted by OCAN

(b) Prob. predicted by OCAN-r

(c) F1 score of OCAN

(d) F1 score of OCAN-r

Figure 3.4: Training progresses of OCAN (3.5.2,3.5.2) and OCAN-r(3.5.2,3.5.2). Three lines in Figures 3.5.2 and 3.5.2 indicate the probabilities of benign users predicted by the discriminator: real benign users $p(y|\mathbf{v}_B)$ (green line) vs. generated samples $p(y|\tilde{\mathbf{v}})$ (red broken line) vs. real malicious users $p(y|\mathbf{v}_M)$ (blue dotted line). Figures 3.5.2 and 3.5.2 show the F1 of OCAN and OCAN-r during training.

user behavior information and hence we can achieve early vandal detection. The difference is that the M-LSTM model uses $\mathbf{h}_t^{en}$ as the input of a classifier directly whereas OCAN further trains complementary GAN and uses its discriminator as a classifier to make the early vandal detection. In this experiment, instead of applying the classifier on the final user representation $\mathbf{v} = \mathbf{h}_T^{en}$, the classifiers of M-LSTM and OCAN are applied on each step of LSTM hidden state $\mathbf{h}_t^{en}$ and predict whether a user is a vandal after the user commits the t-th action.

Table 3.2 shows comparison results in terms of the precision, recall, F1 of early vandal detection, and the average number of edits before the vandals were truly blocked. We can observe that OCAN achieves a comparable performance as the M-LSTM when the number of vandals in the training dataset is large (1000, 4000, and 7000). However, M-LSTM has

very poor accuracy when the number of vandals in the training dataset is small. In fact, we observe that M-LSTM could not detect any vandal when the training dataset contains less than 400 vandals. On the contrary, OCAN does not need any vandal in the training data.

The experimental results of OCAN-r for early vandal detection are shown in the last row of Table 3.2. OCAN-r outperforms M-LSTM when M-LSTM is trained on a small number of the training dataset. However, the OCAN-r is not as good as OCAN. It indicates that generating complementary samples to train the discriminator can improve the performance of the discriminator for vandal detection.

3.5.4 OCAN Framework Analysis

Complementary GAN vs. Regular GAN: In our OCAN model, the generator of complementary GAN aims to generate complementary samples that lie in the low-density region of real samples, and the discriminator is trained to detect the real and complementary samples. We examine the training progress of OCAN in terms of predication accuracy. We calculate probabilities of real benign users $p(y|\mathbf{v}_B)$ (shown as green line in Figure 3.5.2), malicious users $p(y|\mathbf{v}_M)$ (blue dotted line) and generated samples $p(y|\tilde{\mathbf{v}})$ (read broken line) being benign users predicted by the discriminator of complementary GAN on the testing dataset after each training epoch. We can observe that after OCAN is converged, the probabilities of malicious users predicted by the discriminator of complementary GAN are much lower than that of benign users. For example, at the epoch 40, the average probability of real benign users $p(y|\mathbf{v}_B)$ predicted by OCAN is around 70%, while that of malicious users $p(y|\mathbf{v}_M)$ is only around 30%. Meanwhile, the average probability of generated complementary samples $p(y|\tilde{\mathbf{v}})$ lies between the probabilities of benign and malicious users.

On the contrary, the generator of a regular GAN in the OCAN-r model generates fake

samples that are close to real samples, and the discriminator of GAN focuses on distinguishing the real and generated fake samples. As shown in Figure 3.5.2, the probabilities of real benign users and probabilities of malicious users predicted by the discriminator of regular GAN become close to each other during training. After the OCAN-r is converged, both the probabilities of real benign users and malicious users are close to 0.5. Meanwhile, the probability of generated samples is similar to the probabilities of real benign users and malicious users.

We also show the F1 scores of OCAN and OCAN-r on the testing dataset after each training epoch in Figure 3.5.2 and 3.5.2. We can observe that the F1 score of OCAN-r is not as stable as (and also a bit lower than) OCAN. This is because the outputs of the discriminator for real and fake samples are close to 0.5 after the regular GAN is converged. If the probabilities of real benign users predicted by the discriminator of the regular GAN swing around 0.5, the accuracy of vandal detection will fluctuate accordingly.

We can observe from Figure 3.4 another nice property of OCAN compared with OCAN-r for fraud detection, i.e., OCAN is converged faster than OCAN-r. We can observe that OCAN is converged with only training 20 epochs while the OCAN-r requires nearly 100 epochs to keep stable. This is because the complementary GAN is trained to separate the benign and malicious users while the regular GAN mainly aims to generate fake samples that match the real samples. In general, matching two distributions requires more training epochs than separating two distributions. Meanwhile, the feature matching term adopted in the generator of complementary GAN is also able to improve the training process [71].

Visualization of three types of users: We project the user representations of the three types of users (i.e., benign, vandal and complementary benign generated by OCAN) to a two-dimensional space by Isomap [74] and show the projection in Figure 5. We observe that the

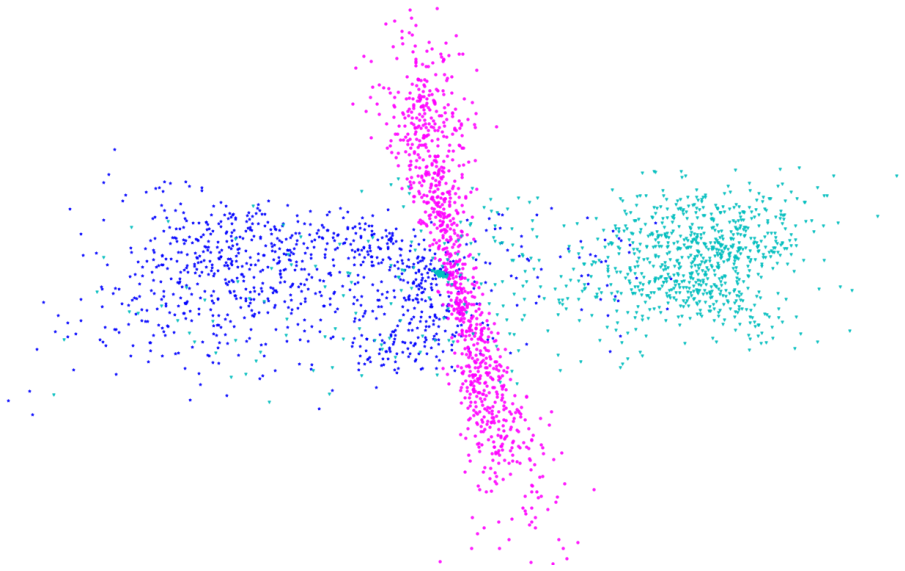


Figure 3.5: 2D visualization of three types of users: real benign (blue star), vandal (cyan triangle), and complementary benign (red dot)

generated complementary users lie in the low-density regions of real benign users. Meanwhile, the generated samples are also between the benign users and vandals. Since the discriminator is trained to separate the benign and complementary benign users, the discriminator is able to separate benign users and vandals.

3.6 Summary

In this chapter, we have developed OCAN for fraud detection when only benign users are available during the training phase. We conducted experiments on a real world dataset and showed that OCAN outperforms the state-of-the-art one-class classification models.

4 Few-shot Insider Threat Detection

Problem Statement: *How do we avail few negative samples to improve the discrimination capability of the trained model for fraud detection ?*

In this chapter, we first give a brief introduction to the problem background, then define the problem and deliver the proposed method. Two datasets, CERT and Wikipedia, will be utilized to evaluate the effectiveness of the proposed method by comparing with the state-of-the-art baselines.

4.1 Introduction

A malicious insider indicates an employee who intentionally used his authorized access in a manner that negatively affected the confidentiality, integrity, or availability of the organization's information [75]. The 2018 U.S. State of Cybercrime Survey indicates that 25% of the cyberattacks are committed by insiders [4].

Since the insiders' behaviors are different from the behaviors of legitimate employees, we can detect the insider threat by analyzing the employees' behaviors via the audit data. In general, user activities are often grouped into sessions that are separated by operations like "LogOn" and "LogOff". However, due to the small number of malicious sessions, the classical supervised learning algorithms cannot be employed. Currently, several unsupervised learning approaches are proposed to detect the malicious sessions [22, 20, 21].

However, in practice, we do know an extremely small number of insiders from the historical data. Instead of unsupervised learning, how to leverage the observed insiders to improve the performance of insider detection is an interesting problem. To tackle this challenge,

in this paper, we propose a framework of combining the idea of self-supervised pre-training [76] and metric-based few-shot learning to detect insiders [77, 33]. Specifically, we first design input representations of user activities in sessions, which capture both activity type and time information, and then adopt the transformer layer proposed in [78] to learn session representations. In order to achieve insider threat detection with only a small number of insiders, our framework is trained by two phases. The first phase is to pre-train the transformer layer by learning an "activity model" on a large number of user sessions. The pre-trained activity model provides strong prior knowledge on how the user sessions are composed. Then, the second phase is to fine-tune the model and learn a similarity function via few-shot learning where the objective is to separate the normal and malicious sessions in the embedding space. After training, new malicious sessions can be detected by having high similarity scores to the observed malicious sessions.

The contributions of our framework are as follows. First, we propose a two-phase training framework that leverages both a large amount of audit data and a small number of observed insiders to improve the performance of the model on insider threat detection. Second, unlike most of the existing works for insider threat detection that only consider the activity type information, we explicitly encode the activity time information into the model via the input representations. Third, experiments on a real insider threat dataset demonstrate that compared with the one-class SVM and isolation forest that only use the normal data and the recurrent neural network that is also trained by a few insiders, our framework achieves the best performance on insider threat detection.

4.2 Framework

We model a user's behavior as a sequence of activities that can be extracted from various types of raw data, such as user logins, emails, and Web browsing. Formally, we model the up-to-date activities of a user as a sequence of sessions $U = \{S_1, \cdots S_k, \cdots\}$ where $S_k = \{e_{k_1}, \cdots, e_{k_j}, \cdots, e_{k_T}\}$ indicates the k-th activity session. One session in our scenario is a sequence of activities starting with 'LogOn" and ending with "LogOff". $e_{k_j} = (t_{k_j}, a_{k_j})$ denotes the j-th activity in the user's k-th session and contains activity type a_{k_j} and occurred time t_{k_j}.

Given an extremely unbalanced training set $\mathcal{D} = \{(S_i, y_i)\}_{i=1}^{m}$, where S_i is the i-th session, and $y_i \in \{0, 1\}$ indicates malicious or not of the session, there is a very small number of sessions that are labeled as malicious in $\mathcal{D}$. Note that we consider the sessions that contain malicious activities, such as uploading documents to Wikileak, as malicious sessions. The goal of learning in our insider threat detection is to predict whether a new session $S_k = \{e_{k_1}, \cdots, e_{k_j}, \cdots, e_{k_T}\}$ is normal or malicious. To address the challenge that there are usually very few records of known insider attacks in the training data, we propose a framework that takes advantages of both self-supervised pre-training and metric-based few-shot learning. The pre-training phase uses the user activity sequence to learn session representations by self-supervised learning, while the fine-tuning phase is to train a feed-forward neural network with small parameters to derive the similarity scores among samples. Figure 4.1 shows the architecture of our proposed framework.

Specifically, we design the input representations for activities in a session by combining the activity type and time information. In order to model the user sessions, we adopt the architecture of transformer layer and use the similar idea as Bidirectional Encoder Representations from Transformers (BERT) to pre-train the model [76]. After pre-training, we

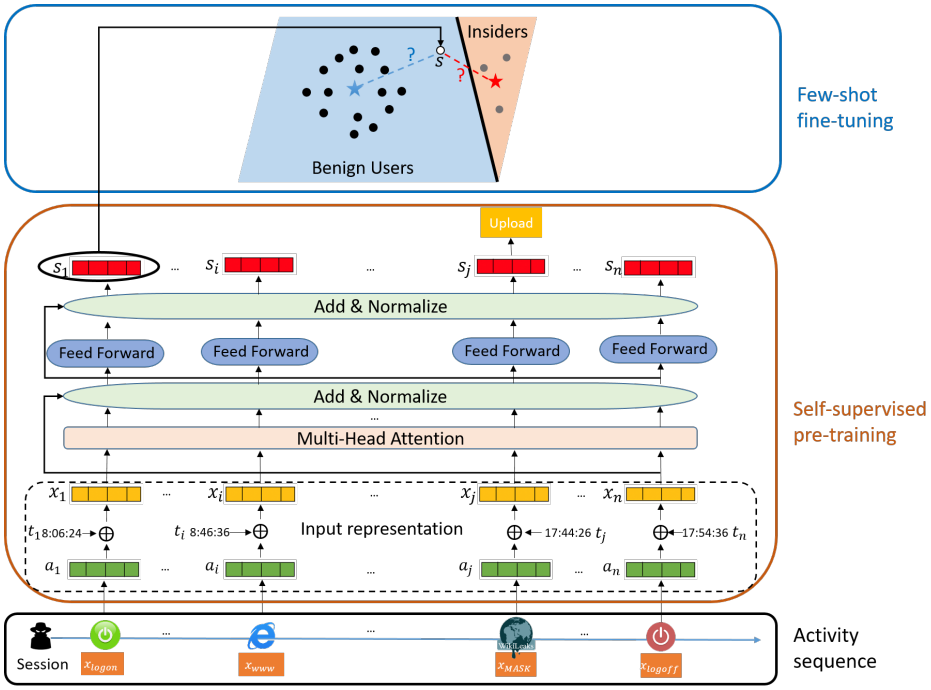


Figure 4.1: The architecture of our framework

fine-tune the model based on a few observed insiders to achieve malicious session detection.

The metric-based few-shot learning first computes the centers of normal and malicious session representations as prototype representations of normal and malicious sessions and then trains a neural network as a similarity function to compute similarity scores between user sessions and prototype representations. Then, new malicious sessions expect to be detected by having high similarity scores to the observed malicious sessions.

4.2.1 Self-supervised Pre-training

Input representation. Input representations aim to map user activities in a session into an embedding space so that the transformer layer can encode the user activities into a session representation. Given a user activity in a session, we have both activity type and time information. The existing work only considers the activity type information, like uploading documents to a removable disk or visiting a website. However, the timing of the user activity

is also an important feature for malicious session detection. For example, an insider may copy classified documents to a removable disk at midnight. Since both activity type and time are important for insider threat detection, input representations should encode both type and time information of user activities to our model. To this end, given an activity type and its corresponding time, we map them to an embedding space. Then, the input representation of the activity is constructed by summing the representations of type and time. The detailed descriptions of the activity type and time representations are given as follows.

Type representations: To obtain type representations, we first consider each activity type as a word in a sentence, each session as a sentence, and the activities of all users as a text corpus. Then, we adopt the word2vec [79] to train the type representations $\mathbf{A} \in \mathbb{R}^{a*d}$, where a is number of activity types.

Time representations: Similar to the position encodings [78], which aim to encode the relative or absolute position information to the model, we propose the time representations to inject the absolute time information of an activity to the input representation. Specifically, we represent the activity time as the offset minute from 12:00 am. For example, if an activity occurs at 1:00 am, we represent the time as 60 minutes past 12:00 am. As a result, the time representations can be represented as $\mathbf{T} \in \mathbb{R}^{1440*d}$, where 1440 is the total number of minutes in a day. There are two advantages to using absolute time representations. First, we can inject the physical time information to the model. Second, since the transformer layer do not have recurrent structure, the absolute time can also capture the order information of activities in a session. We adopt the same sinusoid function as the position encoding in [78] to generate the time representations, which is defined as $\mathbf{T}_{t,2i} = sin(t/10000^{2i/d}); \mathbf{T}_{t,2i+1} = cos(t/10000^{2i/d})$, where t is the t-th minute in a day; i is the i-th dimension of the d-dimensional representation. The advantage of using sinusoid function is that it allows the model to easily learn to attend

by relative positions, since for any fixed offset k, $\mathbf{T}_{t+k}$ can be represented as a linear function of $\mathbf{T}_t$.

Therefore, the input representation of the j-th activity in a session is defined as:

$$\mathbf{x}_j = \mathbf{a}_{a_j} + \mathbf{t}_{t_j}, \tag{4.1}$$

where a_j and t_j indicate the indices of representations of type and time given the j-th activity. The the input representation to the transformer can be represented as $\mathbf{X}^{n*d}$, where n indicates the length of session. With the well-designed input representation, the transformer layer can encode multiple aspects of user behaviors.

Pre-training. We adopt transformer layer to model the user activity sessions [78] and use a similar strategy as BERT to pre-train the transformer layer [76]. In our scenario, we consider the log files that record all the user activities as a pre-training corpus and adopt the masked language model to pre-train the transformer layer. Specifically, the model takes all activities in a session with random masks as inputs, where we randomly replace a small ratio of activities in a session with a specific MASK token. The training object is to accurately predict the randomly masked activity types. The purpose of predicting MASK token is to make the transformer layer capture the user behaviors in terms of activity types and time. Since the behaviors of normal and malicious sessions are different, we expect the transformer layer could encode the prior knowledge of user behavior by training to predict the MASK tokens.

In particular, the transformer layer consists of a multi-head self-attention and a position-wise feed forward sub-layer in which a residual connection is employed around each of two sub-layers, followed by layer normalization [80]. The multi-head attention employs h parallel self-attentions to jointly capture different aspect information at different positions

over the input activity sequence. Formally, for the j-th head of the attention layer, the scaled dot-product self-attention is defined as:

$$head_j = Attention(\mathbf{X}\mathbf{W}_j^Q, \mathbf{X}\mathbf{W}_j^K, \mathbf{X}\mathbf{W}_j^V), \tag{4.2}$$

where $Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_v}})\mathbf{V}$; $\mathbf{X} \in \mathbb{R}^{n*d}$ is the input representation of the session; $\mathbf{W}_j^Q$, $\mathbf{W}_j^K$ and $\mathbf{W}_j^V$ are linear projection weights with dimensions $\mathbb{R}^{d*d_v}$ for the j-th head. Each self-attention makes each activity attend to all the activities in an input session and computes the hidden representation for each activity with an attention distribution over the session.

The multi-head attention employs a parallel of self-attentions to jointly capture different aspect information at different activities. Formally, the multi-head attention concatenates parallel heads together as:

$$f(\mathbf{X}) = Concat(head_1, ..., head_h)\mathbf{W}^O, \tag{4.3}$$

where $\mathbf{W}^O \in \mathbb{R}^{hd_v*d_o}$ is a projection matrix.

Then, the position-wise feed forward sub-layer with a ReLU activation is applied to the hidden representation of each activity separately. Finally, by combining the position-wise feed forward sub-layer and multi-head attention, a transformer layer is defined as:

$$\begin{aligned} \text{transformer\_layer}(\mathbf{X}) &= FFN(f(\mathbf{X})) \\ &= ReLU(f(\mathbf{X})\mathbf{W}_1)\mathbf{W}_2, \end{aligned} \tag{4.4}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are trained projection matrices.

After pre-training, similar to BERT, we consider the final hidden state of the first

activity in the session, i.e., the hidden state of "LogOn", as the session representation $\mathbf{s}$:

$$\mathbf{s} = \text{transformer\_layer}(\mathbf{X})[0], \tag{4.5}$$

where $\mathbf{X} \in \mathbb{R}^{n*d}$ is the input representation of a session. Given a large amount of session sequences with random masks, the pre-training phase is trained to update the parameters $\theta_t = \left\{\{\mathbf{W}_j^Q, \mathbf{W}_j^K, \mathbf{W}_j^V\}_{j=1}^h, \mathbf{W}^O, \mathbf{W}_1, \mathbf{W}_2\right\}$ in transformer layer.

4.2.2 Few-shot Fine-tuning.

After self-supervised pre-training, the session representations derived from transformer layer capture the information of user behaviors in terms of activity types and time. We further fine-tune the session representations and design a similarity function to detect insider threats via a small number of malicious sessions. Concretely, the few-shot learning phase consists of two goals. The first goal is to fine-tune the transformer layer to make the representations of normal and malicious sessions locate separately in the embedding space. Then, we can derive prototype representations of normal and malicious sessions by adopting the mean operation on the normal and malicious samples separately so that each prototype representation is surrounded by sessions in that class. The second goal is to derive a similarity function to evaluate the similarity between a session and prototype representations. After fine-tuning, a new malicious session expect to have a higher similarity score to the malicious prototype representation.

In few-shot learning, each training iteration is formulated as a training episode. In each episode, we randomly sample k normal sessions and k malicious sessions as the support set $\mathcal{S} = \{(S_i, y_i)\}_{i=1}^{2*k}$, and further sample q samples from both normal and malicious sessions as the query set $\mathcal{Q} = \{(S_j, y_j)\}_{j=1}^{q}$ from the training set. We then compute the representation

of each session in the support and query set by Equation 4.5. We denote the representations of S_i from the support set and S_j from the query set as $\mathbf{s}_i$ and $\mathbf{s}_j$.

Based on the support set, we derive the prototype representations for normal and malicious sessions by a mean operation over representations of normal and malicious sessions separately:

$$\mathbf{c}_y = \frac{1}{k} \sum_{(S_i, y_i) \in \mathcal{S}_y} \mathbf{s}_i, \tag{4.6}$$

where $\mathcal{S}_y$ indicates the set of samples with label y and $y \in \{0, 1\}$.

After obtaining two prototype representations from the support set, we further derive the similarity scores of samples in the query set with the representation of each prototype. To this end, we first combine the representation of a query sample with a prototype representation by a concatenate operation $\mathbf{r}_{jy} = Concat(\mathbf{s}_j, \mathbf{c}_y)$. Then, we adopt a fully connected neural network as a similarity function to map the representation $\mathbf{r}_{jy}$ to a similarity score l_{jy}:

$$l_{jy} = g(\mathbf{r}_{jy}; \theta_g), \tag{4.7}$$

where l_{jy} measures the similarity between query sample j and prototype y ranging from 0 to 1; $g(\cdot)$ is a fully connected neural network parameterized by θ_g with a sigmoid function as the final activation function.

We adopt the mean square error as the loss function to fine-tune the transformer layer as well as the neural network $g(\cdot)$ [33]:

$$\mathcal{L} = \sum_{y \in \{0,1\}} \sum_{j=1}^{q} (l_{jy} - \mathbb{1}(y_j == y))^2. \tag{4.8}$$

The general idea is to consider the task as a regression problem. The training procedure is to make the similarity score close to 1 if the query sample and prototype belong to the same

Table 4.1: Statistics of two datasets

Dataset	# of Employees	# of Insiders	# of Sessions	# of Malicious Sessions
CERT	4000	5	1,581,358	48
Wikipedia	4073	822	10,113	4627

class, otherwise, the similarity score should be close to 0. In the fine-tuning phase, we update the parameters θ_g in $g(\cdot)$ and fine-tune the parameters θ_t in transformer layer.

Detection. When we deploy the model for malicious session detection, we adopt all the malicious sessions in the training set to compute the prototype representation of malicious sessions and randomly sample the same number of normal sessions to compute the prototype representation of normal sessions. Given an upcoming session as a new query set, we compute its similarity scores with the two prototype representations. The upcoming session will be detected as malicious if its similarity to the malicious prototype is higher than the one to the normal prototype; otherwise, it is a normal session.

4.3 Experiments

4.3.1 Experimental Setup

4.3.1.1 Dataset

We evaluate our proposed approach on two datasets: *CERT Insider Threat Dataset* [81] and *UMDWikipedia Dataset* [11]. For a summary, Table 4.1 shows the statistics of the dataset.

CERT. A session is a sequence of user activities between "Logon" and "Logoff". Based on the activities recorded in the log files, we extract fine-grained activity types as combinations of activity types and their context. For insider detection, the context of the activity type, such as the name of the visited website, is also crucial for insider threat detection. For example, a normal user usually uploads documents to websites which are

related to their work, but a user who foresees his potential layoff may upload documents to specific websites, like Dropbox or Wikileak. Hence, in our work, we design fined-grained activity types by combining the context information with the activity types, such as, "upload to website Wikileak.org". As a result, we extract 1435 fine-grained activity types. We split the dataset chronologically into a training set and a testing set. Without specific descriptions, we use the sessions occurred in the first 396 days as the training set and the rest 120 days as the testing set. There are 15 malicious sessions in the training set and 33 malicious sessions in the testing set.

Training Details. In our experiments, the model consists of 2 transformer layers, and the multi-head attention sub-layer consists of 4 heads. For transformer layers, the dimension of each attention in multi-head attention sub-layer is 64, so the dimension of feed-forward sub-layer is 256. For the CERT dataset, besides the time in a day, we further incorporate the day in a week and is-working-time as activity time information. Specifically, each day in a week is represented as day representations $\mathbf{D} \in \mathbb{R}^{7*d}$, and another embedding matrix $\mathbf{E} \in \mathbb{R}^{2*d}$ is to represent whether an activity occurs in working time. In our experiments, we define the working time as Monday to Friday and 8am to 6pm. As a result, the input representation for the CERT dataset is the sum of representations of activity type, time, day in a week, and is-working-time. In the few-shot training phase, in each episode, we randomly select 15 normal sessions and 15 malicious sessions to compose the support set. We augment the malicious sessions by randomly shuffling activities between "LogOn" and "LogOff".

Wikipedia. We can consider vandals as insiders in the Wikipedia community. Since Wikipedia dataset does not have explicit indicators, such as LogON or LogOff, to split the user activity sequence into sessions. We consider user activities in one day as a session. If the session contains the activities that are reverted by administrators, the session is malicious;

otherwise, this session is normal. We then filter out the session with activity numbers less than 15. Table 4.1 shows the statistics of the dataset.

Training Details. To represent the type information, we adopt 7 binary features: whether or not the user edited on a meta-page; if the edited page is a meta-page, whether or not this meta-page is empty; whether or not the user consecutively edited the pages in less than 1 minute, 3 minutes, or 15 minutes; whether or not the user's current edit page had been edited before; whether or not the current edit will be reverted by the platform later. For each binary feature, we use a matrix $\mathbf{A} \in \mathbb{R}^{2*d}$ as one type information. By summing of the 7 feature representations, we then get the type representation of an activity. The final input representation is the sum of type and time representation. The architecture of transformer layer is the same as the structure we used in CERT dataset. In the few-shot training stage, for each episode, we randomly pick 50 normal sessions and 50 malicious sessions to construct the support set.

4.3.1.2 Baselines

We compare our model with three baselines, Recurrent Neural Network (**RNN**) [82], One-class SVM (**OCSVM**) [25] and Isolation Forest (**iForest**), in which One-class SVM and Isolation Forest are for the case without any malicious samples and an adaptive few-shot recurrent Neural Network (**RNN**) is designed for the scenario that a small amount of malicious samples are observed in insider threat detection. In the implementation, we take advantage of *scikit-learn* for OCSVM and iForest and implement a few-shot RNN framework based on the original work [82]. In addition, we replace the transformer layers with RNN and adopt the same objective function defined in Equation 4.8 to train the RNN. For OCSVM and iForest, we use activity types to compose the input feature vector, and the value of each

Table 4.2: Comparison of our framework with baselines.

Dataset	Models	Precision	Recall	F1	FPR
CERT	OCSVM	0.0026	0.8182	0.0051	0.1818
	iForest	0.0056	0.3939	0.0110	0.6060
	RNN	0.3038	0.7273	0.4286	0.0055
	Our model	0.9200	0.6970	0.7931	0.0001
Wikipedia	OCSVM	0.5576	0.9870	0.7126	0.7830
	iForest	0.4920	0.1230	0.1968	0.1270
	RNN	0.9548	0.8257	0.8856	0.0393
	Our model	0.9920	0.8626	0.9228	0.0069

feature is the number of the corresponding activity in a session.

4.3.2 Experiment Results

Few-shot Malicious Session Detection. We apply the proposed method and baselines on two real-world datasets, CERT Insider Threat and Wikipedia. Table 4.2 shows the precision, recall, F1, and false positive rate (FPR). We can observe that, compared to baselines, our proposed model achieves the best performance with F1 score 0.7931 and FPR 0.0001 on CERT and F1 score 0.9228 and FPR 0.0069 on Wikipedia. Since OCSVM and iForest only adopt normal sessions for training, we can notice that these two approaches cannot achieve good performance for malicious session detection. Although OCSVM achieves high recall value, the precision is extremely low. Since the RNN model is also trained in the same setting of few-shot learning, it achieves better performance than OCSVM and iForest. However, the F1 score of RNN is still lower than that of our model. It indicates that using transformer layers with carefully designed input representations to model the user sessions can improve the performance of few-shot malicious session detection.

Various numbers of malicious sessions in the training set. We further evaluate the performance of our model trained by various numbers of malicious sessions in the training set. Concretely, the number of malicious sessions is reduced from 15 to 5 for CERT while it is changing from 50 to 5 for Wikipedia. Table 4.3 shows the experimental results. Overall,

Table 4.3: Performance of our framework trained by various numbers of malicious sessions.

Dataset	# of Malicious Sessions	Precision	Recall	F1	FPR
CERT	5	0.0047	0.8140	0.0096	0.7194
	8	0.8824	0.3750	0.5263	0.0001
	10	0.7333	0.5789	0.6471	0.0007
	15	0.9200	0.6970	0.7931	0.0001
Wikipedia	5	0.4994	0.9529	0.6554	0.9497
	15	0.6939	0.8440	0.7616	0.3709
	30	0.9807	0.8637	0.9185	0.0171
	50	0.9920	0.8626	0.9228	0.0069

on both CERT and Wikipedia, F1 scores are decreasing as the number of malicious sessions reduces. Meanwhile, we can observe that, even if with few malicious sessions, the proposed few-shot insider threat detection model can still achieve a reasonable F1 score and a low false positive rate. For CERT, given 10 malicious sessions, F1 score is 0.6471 and FPR is 0.0007. For Wikipedia, given 30 malicious sessions, F1 score is 0.9185 and FPR is 0.0171.

Ablation studies. In order to better understand the performance of our framework, we conduct several ablation experiments. First, the input representations consist of two components, i.e., the representations of activity type and time. We train the model by removing one of the components each time, rather than using all components. Meanwhile, since our framework is trained by two phases, we also study the performance of the framework without pre-training or without few-shot fine-tuning the model. In the scenario without few-shot fine-tuning, we adopt the L2-distance as the similarity score to label the sessions.

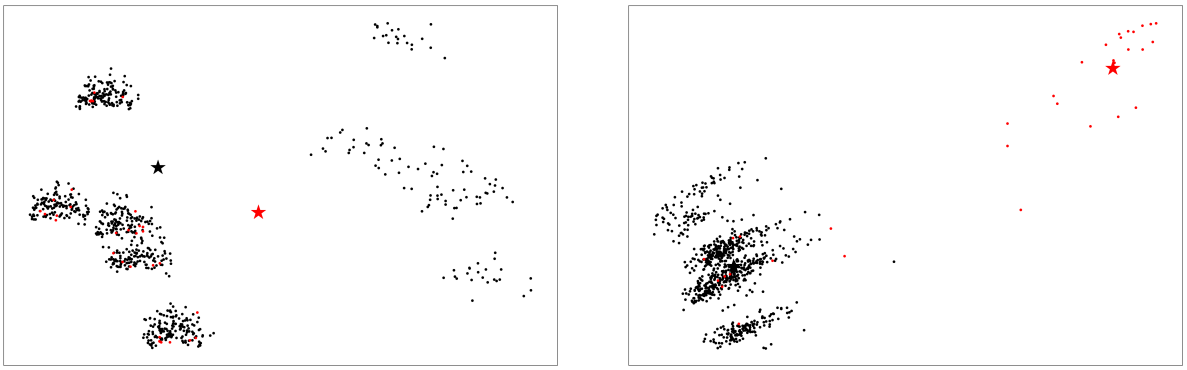
Table 4.4: Performance of our framework after removing various components.

	Precision	Recall	F1	FPR
w.o. type representation	0.1013	0.2424	0.1429	0.0070
w.o. time representation	0.8519	0.6970	0.7667	0.0003
w.o. pre-training	0.8519	0.6970	0.7667	0.0003
w.o. few-shot fine-tuning	0.0085	0.5758	0.0168	0.2197

Table 4.4 shows the experimental results. As we expect, without using the representations of activity types in a session, the model cannot achieve malicious session detection.

Meanwhile, by removing the time representations, the F1 score also slightly reduces. For the two training phases, we can also notice that the performance of the model reduces without pre-training, and the model cannot achieve reasonable performance without few-shot fine-tuning the model. It indicates that obtaining the prior knowledge about the user sessions via pre-training can improve the model performance, while using a few malicious sessions in the training process is the key to achieve malicious session detection.

Visualization. We adopt PCA to project the session embeddings to a two-dimensional space and visualize the normal and malicious sessions. We adopt all the malicious sessions and the same number of normal sessions in the training set to compute the prototype embeddings. Then, we select all the malicious sessions and randomly choose 800 normal sessions from the testing set. Figure 4.2 shows the visualization results of malicious and normal sessions as well as two prototype representations before and after the few-shot learning phase. We can observe that before the few-shot learning phase, the malicious and normal sessions are mixed together, and the prototypes of malicious and normal sessions close to each other. After the few-shot learning phase, the malicious and normal sessions are clearly separated into two parts of the figure, and the prototypes of two types of sessions also have long distance in the two-dimensional space. Meanwhile, all the normal sessions close to the prototype of normal sessions, which explains the low false positive rates in our experiments. Moreover, most of the malicious sessions close to the prototype of malicious sessions, and some of the malicious sessions are in the area of the normal sessions. Hence, we can achieve high true positive rates, and the recall of the malicious session detection is around 70%.



(a) Before few-shot learning (b) After few-shot learning

Figure 4.2: The visualizations of sessions embeddings. The red and black dots indicate the malicious and normal sessions, respectively, while the red and black star indicate the prototype of malicious and normal session, respectively.

4.4 Summary

In this chapter, we have developed a novel framework that consists of two training phases, self-supervised pre-training phase and few-shot learning phase, for insider threat detection. Experimental results on an insider threat detection dataset demonstrated the effectiveness of our framework.

5 SAFE: A Neural Survival Analysis Model for Fraud Early Detection

Problem Statement: *How do we utilize the late-response labels for fraud early detection*?

In this chapter, we first introduce the problem background, then formulate the problem and present the proposed method. The two real-world datasets, Twitter and Wikipedia, will be leveraged to evaluate the effectiveness of the proposed method by comparing with the state-of-the-art baselines.

5.1 Introduction

Due to the openness and anonymity of the Internet, online platforms (e.g., online social media or knowledge bases) attract a large number of malicious users, such as vandals, trolls, and sockpuppets. These malicious users impose severe security threats to online platforms and their legitimate participants. For example, the fraudsters on Twitter can easily spread fake information or post harmful links on the platform. To protect legitimate users, most web platforms deploy tools to detect fraudulent activities and further take actions (e.g., warning or suspending) against those malicious users. However, there is usually a gap between the time that fraudulent activities occur and the time that response actions are taken. Training datasets collected and used for building new detection algorithms often contain the labeled information about when users are suspended instead of when users take fraudulent actions. For example, using twitter streaming API and crawler can easily collect the suspended time information of fraudsters in addition to a variety of dynamically changing features (e.g., the number of posts or the number of followers). However, there is no ground truth about when fraudulent activities occur from the collected data. Hence, the algorithms

trained on such datasets cannot achieve in-time or even early detection if they do not take into consideration the gap between suspended time and fraudulent activity time. In this work, we aim to develop effective fraud early detection algorithms over such training data that contains time-varying features and late response labels.

Fraud early detection has attracted increasing attention in the research community [11, 12, 16, 17]. The existing approaches for fraud early detection are usually based on classification models (e.g., neural network, SVM). Given a sequence of user activities that contain intermittent fraudulent activities, the prediction at each timestamp from the built classifier is often independent to each other. Hence, these classification models tend to make inconsistent and ad-hoc predictions along the time. Figure 5.1 shows an illustrative example. A user takes a fraudulent action at time t_2, the classification model predicts the user as a fraudster at t_2 and t_4 but as normal user at t_3. This is because the prediction probabilities between consecutive timestamps do not have any relations.

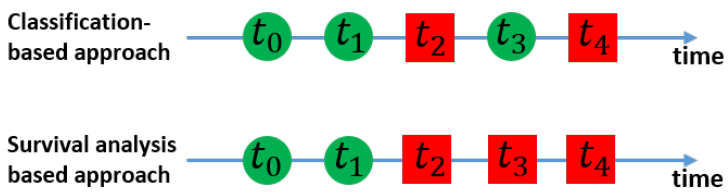


Figure 5.1: Comparison of the survival analysis-based approach and classification-based approach for fraud early detection. Red square indicates that the user is predicted as fraudsters at time t while the green circle indicates the user is predicted as normal.

In this work, we propose to use the survival analysis [83] to achieve consistent predictions along the time. Survival analysis models the time until an event of interest occurs and incorporates two types of information: 1) whether an event occurs or not, and 2) when the event occurs. In survival analysis, *hazard rate* and *survival probability* are adopted to model event data. The hazard rate at time t indicates the instantaneous rate at which events

occur, given no previous event whereas the survival probability indicates the probability that a subject will survive past time t.

In the fraud detection scenario, the event is that a fraudster is suspended by the platform. We use the survival function, which is monotonically decreasing, to model the likelihood of being fraudster for a given user based on his observed activities. Hence, unlike the classification model that makes ad-hoc predictions, the survival model can keep track of user survival probabilities over time and provide consistent prediction. When deployed, the survival analysis model can easily calculate the survival probability of a new user at each timestamp based on his activities and predict the user as a fraudster when the survival probability is below some threshold.

However, it is nontrivial to adopt survival analysis for fraud detection. Traditional survival analysis models often assume a specific parametric distribution of underlying data. However, it is generally unknown which distribution fits well in fraud detection scenarios. We need a model to handle the features of user activity sequences (time-varying covariates) and further capture general relationships between the survival time distribution and time-varying covariates. To tackle this challenge, we develop a neural Survival Analysis model for Fraud Early detection (SAFE) by combining the recurrent neural network (RNN) with the survival analysis model. SAFE adopts RNN to handle time-varying covariates as inputs and predicts the evolving hazard rate given the up-to-date covariates at each timestamp. RNN can capture the non-linear relations between the hazard rates and time-varying covariates and does not assume any specific survival time distributions. Moreover, to tackle the challenge due to the gap between suspended time (reported in training data) and fraudulent activity time (unavailable in training data), we revise the loss function of the regular survival model. In particular, SAFE is trained to intentionally increase the hazard rates of fraudsters before

they are suspended and decrease the hazard rates of normal users.

The contributions of this work are as follows. First, it is the first work to adopt survival analysis for fraud detection. Different from classification models, our approach achieves consistent predictions along the time. Second, our revised survival model is designed for the training data with late response labels and can achieve fraud early detection. Third, instead of assuming any particular survival time distributions, we propose the use of RNN to learn the hazard rates of users from user activities along time and do not assume any specific distribution. Fourth, we conduct evaluations over two real-world datasets and our model outperform state-of-the-art fraud detection approaches. Note that, this chapter is originally from the published work [84]

5.2 Survival Analysis

Survival analysis models the time until an event of interest occurs. Compared with the common regression models, in a survival analysis experiment, we may not always be able to observe event occurrence from start to end due to missing observation or a limited observation window size. For example, in health data analysis, the time of death can be missing in some patient records. Such phenomenon is called *censoring*. In this work, we focus on two types of censoring: 1) an *uncensored* sample indicates the event is observed; 2) a *right censored* sample indicates the event is not observed in the observation window but we know it will occur later.

Survival time T is a continuous random variable representing the waiting time until the occurrence of an event, with the probability density function $f(t) = \lim_{dt \to 0} \frac{P\{t \leq T < t+dt\}}{dt}$ and the cumulative distribution function $F(t) = P(T < t) = \int_0^t f(x)dx$.

The *survival function* $S(t)$ indicates the probability of the event having not occurred

by time t:

$$S(t) = P(T \geq t) = 1 - F(t) = \int_t^{\infty} f(x)dx. \tag{5.1}$$

The *hazard function* $\lambda(t)$ refers to the instantaneous rate of occurrence of the event at time t given that the event does not occur before time t:

$$\lambda(t) = \lim_{dt \to 0} \frac{P\{t \leq T < t + dt | T \geq t\}}{dt} = \frac{f(t)}{S(t)}. \tag{5.2}$$

Additionally, $S(t)$ is associated with $\lambda(t)$ by

$$S(t) = e^{-\int_0^t \lambda(x)dx}. \tag{5.3}$$

Discrete time. In many cases, the observation time is discrete (seconds, minutes or days). When T is a discrete variable, we denote t a timestamp index and have the discrete expression:

$$S_t = P\{T \geq t\} = \sum_{k=t}^{\infty} f_k, \tag{5.4}$$

$$\lambda_t = P\{T = t | T \geq t\} = \frac{f_t}{S_t}, \tag{5.5}$$

$$S_t = e^{-\sum_{k=1}^{t} \lambda_k}. \tag{5.6}$$

Likelihood function. Given a training dataset with N samples where each sample i has an aggregated covariate $\mathbf{x}^i$, a last-observed time t^i, and an event indicator c^i, the survival model adopts maximum likelihood to estimate the hazard rate and the corresponding survival probability. If a sample i has the event ($c^i = 1$), the likelihood function seeks to make the predicted time-to-event equal to the true event time t^i, i.e., maximizing $P\{T = t^i\}$; if a

sample i is censored ($c^i = 0$), the likelihood function aims to make the sample survive over the last-observed time t^i, i.e., maximizing $P\{T \geq t^i\}$. The joint likelihood function for a sample i is:

$$P\{T = t^i\}^{c^i} \cdot P\{T \geq t^i\}^{1-c^i} = f(t^i)^{c^i} S(t^i)^{1-c^i}. \tag{5.7}$$

The negative log-likelihood function for a sample i can be written as:

$$\begin{aligned} \ell_r^i &= -[c^i \ln(P\{T = t^i\}) + (1 - c^i) \ln(P\{T \geq t^i\})] \\ &= \big(\sum_{t=1}^{t^i} \lambda_t\big) - c^i \cdot \ln(e^{\lambda_{t^i}} - 1), \end{aligned} \tag{5.8}$$

where $\lambda_t = \lambda(t|\mathbf{x}_t^i; \theta)$ is the conditional hazard rate given covariate $\mathbf{x}$ with parameters θ.

The overall loss function over the whole training data is:

$$\mathcal{L}_r = \sum_{i=1}^{N} \ell_r^i = \sum_{i=1}^{N} \Big[\big(\sum_{t=1}^{t^i} \lambda_t\big) - c^i \cdot \ln\big(e^{\lambda_{t^i}} - 1\big)\Big]. \tag{5.9}$$

The survival analysis models learn the relationship between the covariate $\mathbf{x}^i$ and the survival probability $S(t)$ by optimizing parameters θ to estimate λ_t.

5.3 SAFE: A Neural Survival Analysis Model for Fraud Early Detection

In the fraud detection scenario, *event of interest* refers to users being suspended by platforms; then, *survival time* corresponds to the length of time that a user is active. Hence, users who are suspended in the observation window are event samples; users who are not suspended are right-censored samples.

5.3.1 Problem Statement

Let $\mathcal{D} = \{(\mathbf{x}^i, c^i, t^i)\}_{i=1}^N$ denote a set of training triplets, where $\mathbf{x}^i = (\mathbf{x}_1^i, \mathbf{x}_2^i, \cdots, \mathbf{x}_{t^i}^i)$ indicates the sequence data of user i; c^i indicates whether the user i is suspended ($c^i = 1$) or un-suspended ($c^i = 0$) in the observation window; t^i denotes the time when the user i is suspended by the platform or the last-observed time for an un-suspended user; N denotes the size of the dataset. We consider the problem of detecting fraudsters in a timely manner. Because t^i is the suspended time by the platform instead of the time of committing malicious activities, we require the detected time earlier than the suspended time t^i. The goal of learning is to train a mapping function between time-varying covariates and the survival probabilities, i.e., $S_t = f(\mathbf{x}_t^i)$. The learned mapping function can be deployed to predict whether a new user is a fraudster at time t based on his activities by comparing the survival probability S_t with a threshold τ.

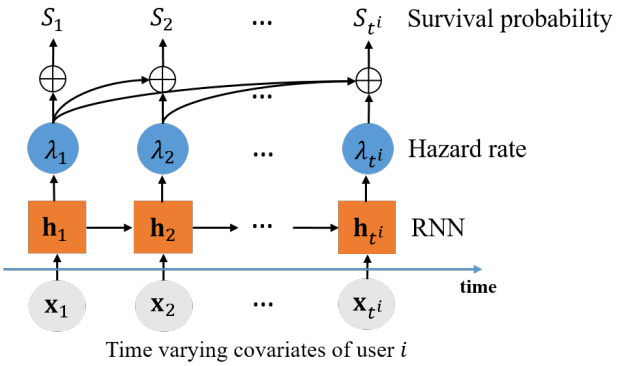


Figure 5.2: An RNN-based survival analysis model for fraud early detection

5.3.2 Model Description

Figure 5.2 describes the basic framework of SAFE. RNN is taken to handle the time-varying covariates and its outputs are hazard rates along time. At timestamp t, RNN maintains a hidden state vector $\mathbf{h}_t \in \mathbb{R}^h$ to keep track of users' sequence information from the

current input $\mathbf{x}_t$ and all of the previous inputs $\mathbf{x}_k$, s.t. $k < t$.

In this work, we adopt the gated recurrent unit (GRU) [70], a variant of the traditional RNN, to model the long-term dependency of time-varying covariates. With $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$, the hidden state $\mathbf{h}_t$ is computed by

$$\mathbf{h}_t = GRU(\mathbf{x}_t, \mathbf{h}_{t-1}). \tag{5.10}$$

As shown in Figure 5.2, at time t, hazard rate λ_t, which indicates the instantaneous rate of a user should be suspended given that the user is still alive at time t, is derived from $\mathbf{h}_t$ by

$$\lambda_t = softplus(\mathbf{w}_\lambda \mathbf{h}_t) = \ln(1 + \exp(\mathbf{w}_\lambda \mathbf{h}_t)), \tag{5.11}$$

where *softplus(·)* is deployed to guarantee that hazard rate λ is always positive, and $\mathbf{w}_\lambda$ is the weight vector of RNN output layer. Note that the softplus function can be replaced by other non-linear functions with positive outputs.

Based on Equation 5.6, the survival probability, which indicates the probability of a user having not been suspended until time t, can be calculates as $S(t) = e^{-\sum_{k=1}^{t} \lambda(k)}$. By comparing the survival probability with a threshold τ, we can predict whether a user should be suspended at time t. The survival probability $S(t)$ is monotonically decreasing along time, hence we can achieve consistent predictions.

For outputs, unlike previous works [47, 46], we do not assume hazard rate λ follows one certain parametric distribution, such as Weibull or Poisson, because, in context of fraud early detection, we do not know whether λ follows one particular distribution. Instead, SAFE directly outputs λ which actually follows a general distribution potentially captured by RNN. We conduct experiments to compare two designs and evaluation results demonstrate SAFE

outperforms the design with specific parametric distributions.

Loss function. The loss function shown in Equation 5.9 for traditional survival analysis cannot be used for learning fraud detection model over the training data with late response labels. In our fraud detection scenario, we aim to detect fraudsters as early as possible while let censored users survive over the last-observed time. However, Equation 5.9 can let censored users pass over the last-observed time but cannot detect fraudsters as early as possible.

Aiming to fraud early detection, a simple but non-trivial adaption is performed on Equation 5.9 to obtain our early-detection-oriented likelihood function, i.e. Equation 5.12. For simplicity, first, we take user i as an example to give the expression of likelihood and loss function, and then show the overall loss function for the whole dataset.

$$\begin{aligned} & P\{T < t^i\}^{c^i} \cdot P\{T \geq t^i\}^{1-c^i} \\ &= \big(F(t^i)\big)^{c^i} \cdot S(t^i)^{1-c^i} \\ &= \big(1 - e^{-\sum_{t=1}^{t^i} \lambda_t}\big)^{c^i} \cdot \big(e^{-\sum_{t=1}^{t^i} \lambda_t}\big)^{1-c^i} \\ &= \big(e^{\sum_{t=1}^{t^i} \lambda_t} - 1\big)^{c^i} \cdot e^{-\sum_{t=1}^{t^i} \lambda_t}. \end{aligned} \tag{5.12}$$

Compared with the likelihood function of a regular survival model shown in Equation 5.7, Equation 5.12 changes $P\{T = t^i\}^{c^i}$ to $P\{T < t^i\}^{c^i}$. After this adaption, intuitively, we can realize that it does match the fraud early detection: with user i being a fraudster ($c^i = 1$), all of hazard rates before t^i will naturally increase as maximizing the term $P\{T < t^i\}$.

Taking the negative logarithm, we could get loss function of user i:

$$\ell^i = \big(\sum_{t=1}^{t^i} \lambda_t\big) - c^i \cdot \ln(e^{\sum_{t=1}^{t^i} \lambda_t} - 1). \tag{5.13}$$

Then, given a set of training samples with N users, the overall loss function is defined as:

$$\mathcal{L} = \sum_{i=1}^{N} \ell^i = \sum_{i=1}^{N} \Big[\big(\sum_{t=1}^{t^i} \lambda_t \big) - c^i \cdot \ln \big(e^{\sum_{t=1}^{t^i} \lambda_t} - 1 \big) \Big]. \tag{5.14}$$

Next we illustrate why SAFE is appropriate for fraud early detection. We denote the model trained by the original loss function $\mathcal{L}_r$ (shown in Equation 5.9) as **SAFE-r**. For simplicity, instead of two overall loss functions, our following discussions focus on ℓ_r^i and ℓ^i.

The first partial derivatives of ℓ_r^i and ℓ^i w.r.t λ are listed as follows:

$$\frac{\partial \ell_r^i}{\partial \lambda_t} = \begin{cases} 1 & 0 < t < t^i \\ 1 - c^i \cdot \frac{e^{\lambda_t}}{e^{\lambda_t} - 1} & t = t^i \end{cases} \tag{5.15}$$

$$\frac{\partial \ell^i}{\partial \lambda_t} = 1 - c^i \cdot \frac{e^{\sum_{k=1}^{t^i} \lambda_k}}{e^{\sum_{k=1}^{t^i} \lambda_k} - 1} \quad 0 < t \leq t^i. \tag{5.16}$$

For a fraudster i ($c^i = 1$), we can see $\frac{\partial \ell_r^i}{\partial \lambda_t} = 1 > 0, (0 < t < t^i)$. It means ℓ_r^i is an increasing function w.r.t λ so that λ_t $(0 < t < t^i)$ is decreasing as minimizing ℓ_r^i. Moreover, in accordance with Equation 5.6, survival probability S_t is increasing with the decrement of λ_t, which means survival probability S_t is increasing with the minimization of ℓ_r^i. That is, instead of detecting the fraudster i before t^i, SAFE-r tends to make the fraudster i survive over t^i. On the contrary, for SAFE, we can observe that $\frac{\partial \ell^i}{\partial \lambda_t} = 1 - c^i \cdot \frac{e^{\sum_{k=1}^{t^i} \lambda_k}}{e^{\sum_{k=1}^{t^i} \lambda_k} - 1} = 1 - \frac{e^{\sum_{k=1}^{t^i} \lambda_k}}{e^{\sum_{k=1}^{t^i} \lambda_k} - 1} < 0$. It means ℓ^i is a decreasing function w.r.t λ so that λ_t $(0 < t < t^i)$ is increasing as minimizing ℓ^i. Similarly, we can achieve that survival probability S_t is decreasing with ℓ^i minimized, which implies that SAFE does have a tendency to detect fraudster i before the suspended time t^i.

For a censored user j ($c^j = 0$), we obtain $\frac{\partial \ell_r^j}{\partial \lambda_t} = \frac{\partial \ell^j}{\partial \lambda_t} = 1$. Both ℓ_r^j and ℓ^j are increasing functions w.r.t λ. As minimizing ℓ_r^j or ℓ^j, λ_t is becoming smaller. SAFE and SAFE-r both have a tendency to make censored user j survive over the last-observed time t^j.

The above theoretical analysis shows why SAFE can achieve the *fraud early detection* better than SAFE-r. Experimental results in the experiment section also validate this theoretical analysis.

5.4 Experiments

5.4.1 Experimental Settings

Datasets. We conduct our experiments on two real-world datasets:

- ***Twitter***. For data preprocessing, we first select suspended users who have the observed timestamps ranging from 12 to 21 and randomly choose the censored users to compose a balanced dataset. To this end, *twitter* consists of 2770 fraudsters and 2770 normal users. We take the change values of five features between two consecutive timestamps as inputs to RNN. Fig.5.4.1 details the components of *twitter* involving numbers of event-censored users at different last-observed timestamps.

- ***Wikipedia***. We also leverage some data preprocessing on Wikipedia and then derive a processed dataset called *wiki* via collecting eight features at each edit for each user: 1) whether the user edits a Wikipedia meta-page, 2) whether the category of the edit page is an empty set, 3) whether the consecutive re-edit is less than one minute, 4) whether the consecutive re-edit is less than three minutes, 5) whether the consecutive re-edit is less than fifteen minutes, 6) whether the current edit page has been edited before, 7) whether the user edits the same page consecutively, and 8) whether the consecutive

re-edit pages have the common category. Fig.5.4.1 illustrates the components of *wiki* involving event-censor numbers at different last-observed timestamps. Different from *twitter* where the censored users are in the last timestamp, there are censored users at each timestamp on *wiki*.

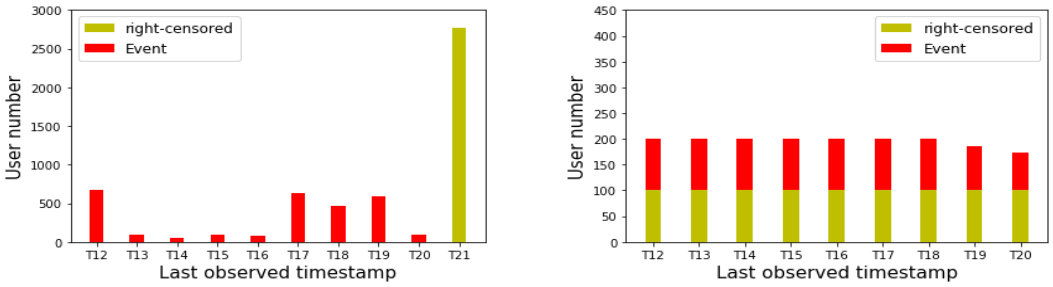
Baselines. We compare SAFE with the following baselines:

- **SVM** is a classical classifier. Given a user time-varying covariate, we average the sequence of each covariate as input to train the SVM and predict the user types (fraudsters or normal users) at each timestamp at the testing phase.

- **CPH** (Cox proportional hazard model) is a classical survival regression model [45]. Similar to SVM, we adopt the average covariates of users as input to train CPH and conduct fraud early detection with the first k timestamps. We adopt Lifelines to implement the CPH model.

- **M-LSTM** (Multi-source LSTM) is a classification-based fraud early detection model that adopts LSTM to capture the information of time-varying covariates and dynamically predict the user type at each timestamp based on the logistic regression classifier [12].

Hyperparameters. SAFE is trained by back-propagation via Adam [85] with a batch size of 16 and a learning rate 10^{-3}. The dimension of the GRU hidden unit is 32. We randomly divide the dataset into a training set, a validation set, and a testing set with the ratio (7:1:2). The threshold τ for fraud early detection is set based on the performance on the validation set. We run our approach and all baselines for 10 times and report the mean and standard deviation of each metric. For all the baselines, we use the default parameters provided by the public packages.



(a) *Twitter*

(b) *Wiki*

Figure 5.3: The distributions of event and right-censored users over the timestamps on *twitter* and *wiki* datasets

Evaluation Metrics. We use *Precision*, *Recall*, *F1* and *Accuracy* to evaluate the fraud early detection performance of various models given the first K-timestamps. For instance, *Accuracy@k* (*k=1,2,3,4,5*) indicates the accuracy given the first K-timestamp inputs. We further report the "percentage of early detected fraudsters" to show the portion of correctly early detected fraudsters and the "early detected timestamps" to show the number of early-detected timestamps of fraudsters. **Repeatability.** Our software together with the datasets are available at https://github.com/PanpanZheng/SAFE.

Table 5.1: The average performance of fraud early detection on the twitter and wiki datasets given the first 5-timestamps

Dataset	Algorithm	Precision	Recall	F1	Accuracy
twitter	SVM	0.7370	0.2733	0.3875	0.5916
	CPH	0.4594	0.7410	0.5440	0.5453
	M-LSTM	0.6336	0.3521	0.4400	0.5755
	SAFE	0.8198	0.5569	0.6537	0.7180
wiki	SVM	0.5484	0.6413	0.5911	0.6754
	CPH	0.5557	0.6206	0.5784	0.6679
	M-LSTM	0.5255	0.9044	0.6556	0.5528
	SAFE	0.7114	0.8798	0.7866	0.7640

Table 5.2: Experimental results (mean±std.) of fraud early detection on the twitter dataset at the first 5-timestamps

Timestamp	Algorithm	Precision	Recall	F1	Accuracy
@1	SVM	0.7500 ± 0.0000	0.2050 ± 0.0000	0.3220 ± 0.0000	0.5683 ± 0.0000
	CPH	0.1333 ± 0.0000	0.0035 ± 0.0000	0.0069 ± 0.0000	0.4901 ± 0.0000
	M-LSTM	0.6307 ± 0.1072	0.2350 ± 0.1174	0.3211 ± 0.1374	0.5483 ± 0.0331
	SAFE	0.8312 ± 0.0313	0.3731 ± 0.0987	0.5053 ± 0.0870	0.6495 ± 0.0309
@2	SVM	0.7260 ± 0.0000	0.1906 ± 0.0000	0.3019 ± 0.0000	0.5593 ± 0.0000
	CPH	0.6166 ± 0.0000	0.7971 ± 0.0000	0.6953 ± 0.0000	0.6508 ± 0.0000
	M-LSTM	0.6291 ± 0.0734	0.2952 ± 0.0500	0.3962 ± 0.0424	0.5584 ± 0.0300
	SAFE	0.8265 ± 0.0297	0.5206 ± 0.0564	0.6360 ± 0.0362	0.7070 ± 0.0154
@3	SVM	0.7473 ± 0.0000	0.2553 ± 0.0000	0.3806 ± 0.0000	0.5845 ± 0.0000
	CPH	0.5309 ± 0.0000	0.9389 ± 0.0000	0.6783 ± 0.0000	0.5547 ± 0.0000
	M-LSTM	0.6239 ± 0.0479	0.3579 ± 0.0458	0.4515 ± 0.0360	0.5720 ± 0.0223
	SAFE	0.8193 ± 0.0267	0.6016 ± 0.0260	0.6929 ± 0.0133	0.7361 ± 0.0089
@4	SVM	0.6463 ± 0.0000	0.1906 ± 0.0000	0.2944 ± 0.0000	0.5431 ± 0.0000
	CPH	0.5112 ± 0.0000	0.9820 ± 0.0000	0.6724 ± 0.0000	0.5215 ± 0.0000
	M-LSTM	0.6256 ± 0.0387	0.3988 ± 0.0600	0.4837 ± 0.0435	0.5822 ± 0.0200
	SAFE	0.8136 ± 0.0237	0.6330 ± 0.0322	0.7111 ± 0.0168	0.7456 ± 0.0108
@5	SVM	0.8156 ± 0.0000	0.5251 ± 0.0000	0.6389 ± 0.0000	0.7032 ± 0.0000
	CPH	0.5050 ± 0.0000	0.9838 ± 0.0000	0.6674 ± 0.0000	0.5098 ± 0.0000
	M-LSTM	0.6591 ± 0.0547	0.4739 ± 0.0793	0.5477 ± 0.0583	0.6167 ± 0.0374
	SAFE	0.8084 ± 0.0424	0.6564 ± 0.0337	0.7235 ± 0.0160	0.7519 ± 0.0107

5.4.2 Experimental Results

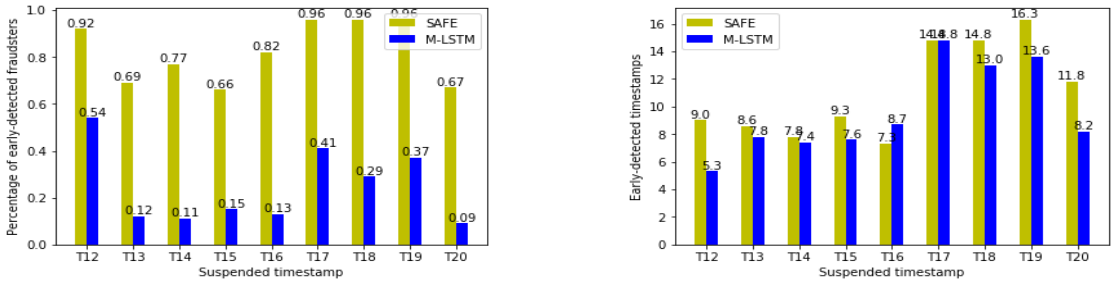
5.4.2.1 Fraud early detection.

Table 5.1 shows the average of metrics of SAFE and baselines for fraud early detection on *twitter* and *wiki* from @1 to @5. It is easily observed that SAFE significantly outperforms three baselines: on *twitter*, accuracies and F1 scores of three baselines are all under 0.60 and 0.55, respectively, especially for CPH with accuracy 0.5453 and SVM with F1 0.3875, while SAFE obtains the acceptable accuracy 0.7180 and F1 0.6537; although three baselines improve their performance on *wiki*, especially for SVM with accuracy 0.6754 and M-LSTM with F1 0.6556, however, SAFE is still far superior to them and achieves satisfiable accuracy 0.7640 and F1 0.7866. Noticeably, although CPH and M-LSTM achieve the best recall on *twitter* and *wiki* (0.7410 and 0.9044), however, they sacrifice their precisions with only 0.4594

and 0.5255 respectively, which indicate very high false positive rates; on the contrary, SAFE performs well on holding the balance between precision and recall such that it achieves precision 0.8198 and recall 0.5569 on *twitter* and precision 0.7114 and recall 0.8798 on *wiki*.

The reason why SAFE performs better than three baselines in early detection is owed to its early-detection-oriented loss function shown in Equation 5.14. Meanwhile, it also indicates that classification and typical survival models are not appropriate to early detection because their internal mechanisms do not support early detection.

Table 5.2 shows the comparison results performed on *twitter*. In accordance with Table 5.2, generally speaking, the F1 and accuracy of SAFE and three baselines increase from @1 to @5. That is, whether for SAFE or three baselines, there is actually some improvement, more or less, in the performance of early detection as timestamp extends. Furthermore, we can also see SAFE performs significantly better than three baselines: at @1, accuracies of three baselines are all under 0.57, especially CPH with 0.49, which to some extent equals to random guess, while SAFE obtains an acceptable accuracy 0.6464 underlying a tracking sequence with a minimum length 12; until @5, SAFE's accuracy reaches 0.7519 while, except for SVM, Cox and M-LSTM have only 0.5098 and 0.6167, respectively. Noticeably, it seems to be abnormal for CPH's recall trend that it starts with 0.0035, then reaches 0.7971, and ends up with 0.9838. Although its recall is big enough, however, it has a random-guess precision around 0.5 which is not acceptable. Moreover, the reason why CPH's recall trend is so weird, we suspect, it is related to that, at least in first five timestamps, the hazards provided by time-series CPH are extremely uneven so that an appropriate survival threshold is unavailable to balance well between recall and precision expected in early detection.



(a) Percentage of early detected fraudsters (b) Early detected timestamps of fraudsters

Figure 5.4: Comparison of SAFE and M-LSTM for fraud early detection on the twitter dataset

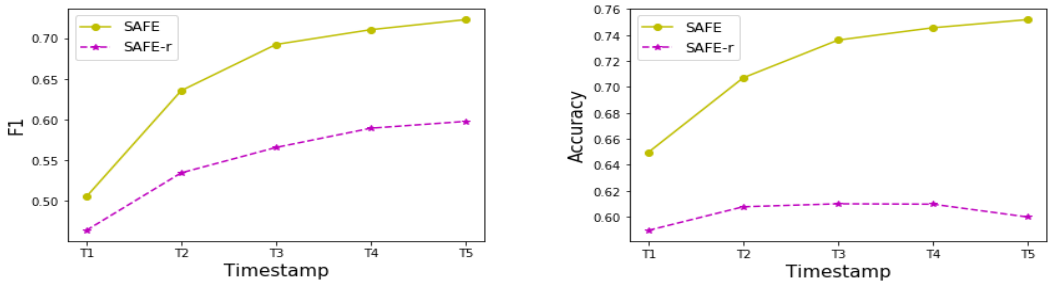
5.4.2.2 SAFE vs M-LSTM.

To show the advantage of survival analysis model, we further take a fine-grained comparison between SAFE and M-LSTM for fraud early detection. M-LSTM is a classification-based model, which adopts LSTM to handle time-varying covariates. SAFE and M-LSTM have the similar neural network structure but are trained by different objective functions. In this study, we separate all the fraudsters on *twitter* into different groups by their suspended timestamps, e.g., "T12" indicates the the group of fraudsters that are suspended at the 12-th timestamp. Figure 5.4.2.1 shows the percentages of early detected fraudsters for each group by SAFE and M-LSTM. We can clearly observe that, compared with M-LSTM, SAFE has a stronger early detection capability with more early-detected fraudsters in each group. For example, at the 12-th suspended timestamp, 92% of fraudsters are early-detected by SAFE while only 54% of fraudsters are early-detected by M-LSTM. Overall, for *twitter*, 82% of fraudsters can be correctly early-detected by SAFE, while only 24% of fraudsters can be early-detected by M-LSTM.

Figure 5.4.2.1 shows the number of early-detected timestamps of fraudsters for each group on *twitter*. We can observe that the early-detected timestamps of SAFE are still larger than those of M-LSTM in most cases. For example, for group "T12", SAFE can detect

fraudsters with 9 timestamps ahead of the true suspended time while the early-detected timestamp of M-LSTM is 5.3. For *twitter*, the average early-detected timestamp of SAFE is 11.1, while the average early-detected timestamp of M-LSTM is 9.6. Consequently, in terms of both the percentage of early-detected fraudsters and the number of early-detected timestamps, we can see SAFE obviously outperforms M-LSTM in the fraud early detection scenario.

5.4.3 Model Analysis



(a) F1 (b) Accuracy

Figure 5.5: Comparison of SAFE and SAFE-r for fraud early detection on the twitter dataset.

5.4.3.1 SAFE vs. SAFE-r.

To show the advantage of the early-detection-oriented loss function, we compare SAFE with SAFE-r that adopts regular loss function of survival analysis. Figures 5.4.3 and 5.4.3 show the variation of F1 and accuracy along the timestamps on *twitter*. Generally speaking, as the timestamp extends, the F1 and accuracy of SAFE and SAFE-r both increase, so their early detection performance roughly gets better. Nevertheless, we see SAFE is obviously superior to SAFE-r: from T1 to T5, the curves of SAFE for F1 and accuracy are significantly above the one of SAFE-r. Concretely, SAFE's accuracy reaches over 0.75 while SAFE-r just has 0.60 at T5. The reason behind this performance difference is associated

with their loss functions. For SAFE-r, there is no internal mechanism to support it for early detection and its small performance improvement, such as accuracy from 0.57 to 0.60, is mainly due to information accumulation between steps provided by RNN; however, based on the modification of survival analysis, SAFE has its internal mechanism for early detection.

Table 5.3: The average performance of neural survival model for fraud early detection on the twitter dataset with and without assuming prior distributions given the first 5-timestamps

Algorithm	Precision	Recall	F1	Accuracy
Rayleigh-RNN	0.5333	0.0012	0.0025	0.5051
Poisson-RNN	0.4857	0.0012	0.0024	0.5051
Exponential-RNN	0.7824	0.0589	0.1044	0.5267
Weibull-RNN	0.7381	0.2865	0.3850	0.5920
SAFE	0.8198	0.5569	0.6537	0.7180

5.4.3.2 SAFE vs. Specific Distributions.

One advantage of SAFE is that SAFE does not assume any specific distributions. We further evaluate the performance of the neural survival model with and without assuming any specific distributions. In this experiment, we train RNN to predict the parameters of a particular distribution instead of hazard rate given time-varying covariates. We adopt three common distributions for modeling the survival time, i.e., Rayleigh, Poisson, Exponential and Weibull distributions. Table 5.3 shows the average performance of fraud early detection on *twitter* given the timestamps from 1 to 5. We can observe that SAFE, which does not assume any survival time distribution, significantly outperforms the other approaches by at least 10% in terms of accuracy and 25% in terms of F1. The experimental results indicate that SAFE, a model without assuming any specific distribution, is more appropriate to fraud early detection.

5.5 Summary

In this chapter, we have developed SAFE that combines survival analysis and RNN for fraud early detection. Experimental results on two real world datasets demonstrate that SAFE outperforms classification-based models, the typical survival model, and RNN-based survival models with specific distributions.

6 Insider Threat Detection via Hierarchical Neural Temporal Point Processes

Problem Statement: *How do we develop a multi-scale fraud detection method via involving activity physical time and type?*

In this chapter, we first take a brief review of the problem background, then define the problem and deliver the proposed method. The two datasets, CRET and Wikipedia, will be used to evaluate the effectiveness of the proposed method.

6.1 Introduction

We study how to develop a detection model that captures both activity time and type information. In literature, the marked temporal point process (MTPP) is a general mathematical framework to model the event time and type information of a sequence. It has been widely used for predicting the earthquakes and aftershocks [56]. The traditional MTPP models make assumptions about how the events occur, which may be violated in reality. Recently, researchers [42, 58] proposed to combine the temporal point process with recurrent neural networks (RNNs). Since the neural network models do not need to make assumptions about the data, the RNN-based MTPP models usually achieve better performance than the traditional MTPP models.

However, one challenge of applying RNN-based temporal point processes in insider threat detection is it cannot model the time information in multiple time scales. For example, user activities are often grouped into sessions that are separated by operations like "LogOn" and "LogOff". The dynamics of activities within sessions are different from the dynamics of sessions. To this end, we propose a hierarchical RNN-based temporal point process model

that is able to capture both the intra-session and inter-session time information.

Our model contains two layers of long short term memory networks (LSTM) [86], which are variants of the traditional RNN. The lower-level LSTM captures the activity time and types in the intra-session level, while the upper-level LSTM captures the time length information in the inter-session level. In particular, we adopt a sequence to sequence model in the lower-level LSTM, which is trained to predict the next session given the previous session. The upper-level LSTM takes the first and last hidden states from the encoder of the lower-level LSTM as inputs to predict the interval of two sessions and the duration of next session. By training the proposed hierarchical model with the activity sequences generated by normal users, the model can predict the activity time and types in the next session by leveraging the lower-level sequence to sequence model, the time interval between two consecutive sessions and the session duration time from the upper-level LSTM. In general, we expect our model trained by normal users can predict the normal session with high accuracy. If there is a significant difference between the predicted session and the observed session, the observed session may contain malicious activities from insiders.

Our work makes the following contributions: (1) we develop an insider threat detection model that uses both activity type and time information; (2) we propose a hierarchical neural temporal point process model that can effectively capture two time-scale information; (3) the experiments on two datasets demonstrate that combining the activity type and multi-scale time information achieves the best performance for insider threat detection. Note that, this chapter is originally from the published work [87]

6.2 Marked Temporal Point Process

Marked temporal point process is to model the observed random event patterns along time. A typical temporal point process is represented as an event sequence $S = \{e_1, \cdots, e_j, \cdots, e_T\}$. Each event $e_j = (t_j, a_j)$ is associated with an activity type $a_j \in \mathcal{A} = \{1, \cdots, A\}$ and an occurred time $t_j \in [0, T]$. Let $f^*((t_j, a_j)) = f((t_j, a_j)|\mathcal{H}_{t_{j-1}})$ be the conditional density function of the event a_j happening at time t_j given the history events up to time t_{j-1}, where $\mathcal{H}_{t_{j-1}} = \{(t_{j'}, a_{j'})|t_{j'} <= t_{j-1}, a_{j'} \in \mathcal{A}\}$ as the collected historical events before time t_j. Throughout this paper, we use $*$ notation to denote that the function depends on the history. The joint likelihood of the observed sequence S is:

$$f\big(\{(t_j, a_j)\}_{j=1}^{|S|}\big) = \prod_{j=1}^{|S|} f((t_j, a_j)|\mathcal{H}_{t_{j-1}}) = \prod_{j=1}^{|S|} f^*((t_j, a_j)). \tag{6.1}$$

There are different forms of $f^*((t_j, a_j))$. However, for mathematical simplicity, it usually assumes the times t_j and mark a_j are conditionally independent given the history $\mathcal{H}_{t_{j-1}}$, i.e., $f^*((t_j, a_j)) = f^*(t_j)f^*(a_j)$, where $f^*(a_j)$ models the distribution of event types; $f^*(t_j)$ is the conditional density of the event occurring at time t_j given the timing sequences of past events [42].

A temporal point process can be characterized by the *conditional intensity function*, which indicates the expected instantaneous rate of future events at time t:

$$\lambda^*(t) = \lambda(t|\mathcal{H}_{t_{j-1}}) = \lim_{\mathrm{d}t \to 0} \frac{\mathbb{E}[N([t, t + \mathrm{d}t])|\mathcal{H}_{t_{j-1}}]}{\mathrm{d}t}, \tag{6.2}$$

where $N([t, t + \mathrm{d}t])$ indicates the number of events occurred in a time interval $\mathrm{d}t$. Given the conditional density function f and the corresponding cumulative distribution F at time t,

the intensity function can be also defined as:

$$\lambda^*(t) = \frac{f(t|\mathcal{H}_{t_{j-1}})}{S(t|\mathcal{H}_{t_{j-1}})} = \frac{f(t|\mathcal{H}_{t_{j-1}})}{1 - F(t|\mathcal{H}_{t_{j-1}})}, \tag{6.3}$$

where $S(t|\mathcal{H}_{t_{j-1}}) = exp(-\int_{t_{j-1}}^{t} \lambda^*(\tau)d\tau)$ is the *survival function* that indicates the probability that no new event has ever happened up to time t since $t_j - 1$. Then, the *conditional density function* can be described as:

$$f^*(t) = f(t|\mathcal{H}_{t_{j-1}}) = \lambda^*(t)exp\Big(-\int_{t_{j-1}}^{t} \lambda^*(\tau)d\tau\Big). \tag{6.4}$$

With an observation window $[0, T]$, the likelihood of the observed event time sequence $\mathcal{T} = \{t_1, \cdots, t_n\}$, s.t. $T > t_n$, is formulated as

$$\mathcal{L} = \prod_{t_i \in \mathcal{T}} f^*(t_i) = \prod_{t_i \in \mathcal{T}} \lambda^*(t_i) \cdot exp\Big(-\int_{0}^{T} \lambda^*(\tau)d\tau\Big). \tag{6.5}$$

Hawkes process is one category of temporal point processes and it has a distinctive feature *self-excitation*: the occurrence likelihood of an upcoming event increases due to the previous events which just occur. In Hawkes process, the conditional intensity function is defined as:

$$\lambda^*(t) = \lambda_0 + \sum_{t_i \in \mathcal{T}} \gamma(t, t_i), \tag{6.6}$$

where $\lambda_0 > 0$ is the base intensity which is independent of the historical events, $\gamma(t, t_i)$ is the triggering kernel that is usually a monotonically decreasing function to guarantee recent events have more influence on the occurrence of the upcoming event. The Hawkes process models the self-excitation phenomenon that a new event arrival increases the upcoming event's conditional intensity which then decreases back towards λ_0 gradually. Hawkes process

is widely used to model the cluster patterns, e.g., the information diffusion on online social networks or the earthquake occurrences.

6.3 Sequence-to-Sequence Model

In general, a sequence-to-sequence (seq2seq) model is used to convert sequences from one domain to sequences in another domain. The seq2seq consists of two components, one encoder and one decoder. Both encoder and decoder are long short-term memory (LSTM) models and can model the long-term dependency of sequences. The seq2seq model is able to encode a variable-length input to a fixed-length vector and further decode the vector back to a variable-length output. The length of the output sequence could be different from that of the input sequence.

The goal of the seq2seq model is to estimate the $P(y_1, \cdots, y_{T'}|x_1, \cdots, x_T)$, where $(x_1, \cdots, x_T)$ is an input sequence and $(y_1, \cdots, y_{T'})$ is the corresponding output sequence. The **encoder** encodes the input sequence to a hidden representation with an LSTM model $\mathbf{h}_j^{en} = LSTM^{en}(\mathbf{x}_j, \mathbf{h}_{j-1}^{en})$ where $\mathbf{x}_j$ is the up-to-date input, $\mathbf{h}_{j-1}^{en}$ is the previous hidden state, and $\mathbf{h}_j^{en}$ is the learned current hidden state. The last hidden state $\mathbf{h}_T^{en}$ captures the information of the whole input sequence. The **decoder** computes the conditional probability $P(y_1, \cdots, y_{T'}|x_1, \cdots, x_T)$ by another LSTM model whose initial hidden state is set as $\mathbf{h}_T^{en}$:

$$P(y_1, \cdots, y_{T'}|x_1, \cdots, x_T) = \prod_{j=1}^{T'} P(y_j|\mathbf{h}_T^{en}, y_1, \cdots, y_{j-1}). \tag{6.7}$$

In seq2seq model, $P(y_j|\mathbf{h}_T^{en}, y_1, \cdots, y_{j-1}) = g(\mathbf{h}_j^{de})$, where $\mathbf{h}_j^{de} = LSTM^{de}(\mathbf{y}_{j-1}, \mathbf{h}_{j-1}^{de})$ is the j-th hidden vector of the decoder; $g(\cdot)$ is usually a softmax function.

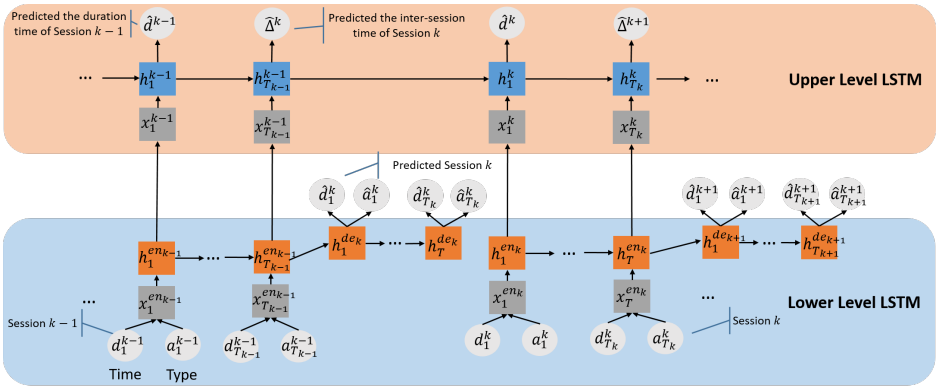


Figure 6.1: The framework for sequence generation with two time scales. The lower-level LSTM captures event patterns with the time and mark pairs in a session. The upper-level LSTM aims to predict the duration of sessions and inter-sessions.

6.4 Insider Threat Detection

6.4.1 Framework

We model a user's behavior as a sequence of activities that can be extracted from various types of raw data, such as user logins, emails, Web browsing, and FTP. Formally, we model the up-to-date activities of a user as sequence $U = \{S^1, \cdots S^k, \cdots\}$ where $S^k = \{e_1^k, \cdots, e_j^k, \cdots, e_{T_k}^k\}$ indicates his k-th activity session. For example, each session in our scenario is a sequence of activities starting with "LogOn" and ending with "LogOff". $e_j^k = (t_j^k, a_j^k)$ denotes the j-th activity in the user's k-th session and contains activity type a_j^k and occurred time t_j^k. We define $d_j^k = t_j^k - t_{j-1}^k$ as the inter-activity duration between activities a_j^k and a_{j-1}^k, $d^k = t_{T_k}^k - t_1^k$ as the length time of the k-th session, and $\Delta^k = t_1^k - t_{T_{k-1}}^{k-1}$ as the time interval between the $(k-1)$-th and k-th sessions. Note that $t_{T_{k-1}}^{k-1}$ is the occurred time of the last activity in the $(k-1)$-th session.

The goal of learning in our threat detection is to predict whether a new session $S^k = \{e_1^k, \cdots, e_j^k, \cdots, e_{T_k}^k\}$ is normal or fraudulent. To address the challenge that there are often no or very few records of known insider attacks for training our model, we propose a

generative model that models normal user behaviors from a training dataset consisting of only sequences of normal users. The learned model is then used to calculate the fraudulent score of the new session S^k. We quantify the fraudulence of S^k from two perspectives, activity information (including both type and time) within sessions, and session time information (i.e., when a session starts and ends). For example, a user who foresees his potential layoff may have activities of uploading documents to Dropbox and visiting job-searching websites although he may try to hide these abnormal activities in multiple sessions; he may have "LogOn" and "LogOff" times different from his normal sessions as he may become less punctual or may have more sessions during weekends or nights, resulting different session durations and intervals between sessions. Moreover, when a user's account is compromised, activity and session information from the attacker will also be different even if the attacker tries to mimic the normal user's behaviors.

We develop a unified hierarchical model capable of capturing a general nonlinear dependency over the history of all activities. Our detection model does not rely on any pre-defined signatures and instead use deep learning models to capture user behaviors reflected in raw data. Specifically, our hierarchical model learns the user behaviors in two time scales, intra-session level and inter-session level. For the intra-session level, we adopt the seq2seq model to predict $\hat{S}^k$ based on the previous S^{k-1} and use the marked temporal point process model to capture the dynamic difference of activities. Note that the number of activities of the predicted session $\hat{S}^k$ could be different from that of the previous S^{k-1} as well as the true S^k. For the inter-session level, we aim to model the session interval $\Delta^k = t_1^k - t_{T_{k-1}}^{k-1}$ and the session duration $d^k = t_{T_k}^k - t_1^k$ of the k-th session.

The whole framework of predicting future events with two time scales is shown in Figure 6.1. We do not assume any specific parametric form of the conditional intensity

function. Instead, we follow [42] to seek to learn a general representation to approximate the unknown dependency structure over the history. We also emphasize that the neural temporal point processes of two levels are connected in our framework. The upper-level LSTM takes the first and last hidden states from the encoder of the lower-level LSTM as inputs to predict the interval of two sessions and the session duration. This connection guarantees the upper-level LSTM incorporates activity type information in its modeling. For insider threat detection, since our model is trained by benign sessions, the predicted session $\hat{S}^k$ would be close to the observed S^k when S^k is normal, and different from S^k when S^k is abnormal. In Session 6.4.4, we will present details about how to derive fraudulent score by comparing $(\hat{S}^k, \hat{d}^k, \hat{\Delta}^k)$ with (S^k, d^k, Δ^k), where $\hat{\bullet}$ indicates the predicted value.

6.4.2 Intra-Session Insider Threat Detection

In this work, we propose to use the seq2seq model to estimate the joint likelihood of k-th session given the $(k-1)$-th session. In particular, the encoder of the seq2seq model is to encode the activity time and type information at $(k-1)$-th session to a hidden representation. The decoder is to model the activity time interval d^k_{j+1} and type a^k_{j+1} information at k-th session given the history.

Encoder: To map the $(k-1)$-th session to a hidden representation, the encoder first maps each activity occurring at time t^{k-1}_j with type a^{k-1}_j to an embedding vector $\mathbf{x}^{en_{k-1}}_j$:

$$\mathbf{x}^{en_{k-1}}_j = \mathbf{w}^t d^{k-1}_j + \mathbf{W}^{em}\mathbf{a}^{k-1}_j, \tag{6.8}$$

where d^{k-1}_j is the inter-activity duration between a^{k-1}_j and a^{k-1}_{j-1}; $\mathbf{w}^t$ is a time-mapping parameter; $\mathbf{W}^{em}$ is an activity embedding matrix; $\mathbf{a}^{k-1}_j$ is a one-hot vector of the activity type a^{k-1}_j. Then, by taking the entire sequence of $(k-1)$-th session as inputs to the encoder

LSTM, the encoder projects the $(k-1)$-th session to a hidden representation $\mathbf{h}_{T_{k-1}}^{en_{k-1}}$.

Decoder: The decoder is trained to predict the pairs of activity type and time at the k-th session given the information of $(k-1)$-th session. To predict the activity type information, given the hidden state of the decoder $\mathbf{h}_j^{de_k}$, the probability of the next activity having type value a can be derived by a softmax function:

$$P(a_{j+1}^k = a|\mathbf{h}_j^{de_k}) = \frac{exp(\mathbf{w}_a^s \mathbf{h}_j^{de_k})}{\sum_{a'=1}^{A} exp(\mathbf{w}_{a'}^s \mathbf{h}_j^{de_k})}, \tag{6.9}$$

where $\mathbf{w}_a^s$ is the a-th row of the weight matrix $\mathbf{W}^s$ in the softmax function.

To predict the activity time information, we adopt the conditional density function defined in Equation 6.4. First, inspired by [42], we derive the LSTM-based conditional intensity function $\lambda^*(t)$ as:

$$\lambda^*(t) = exp(\mathbf{v}\mathbf{h}_j^{de_k} + u^t(t - t_j^k) + b), \tag{6.10}$$

where the exponential function is deployed to ensure the intensity function is always positive; $\mathbf{v}$ is a weight vector; u^t and b are scalars. Then, we can derive the conditional density function given the history until time t_j^k:

$$\begin{aligned} f^*(t) &= \lambda^*(t)(\int_{t_j^k}^{t} \lambda^*(\tau)d\tau) \\ &= exp\big(\mathbf{v}^t\mathbf{h}_j^{de_k} + u^t(t - t_j^k) + b^t + \frac{1}{u}exp(\mathbf{v}^t\mathbf{h}_j^{de_k} \\ &+ b^t) - \frac{1}{u}exp(\mathbf{v}^t\mathbf{h}_j^{de_k} + u^t(t - t_j^k) + b^t)\big). \end{aligned} \tag{6.11}$$

Hence, given the observed activity time information, we can calculate the conditional density function of the time interval between two consecutive activities $d_{j+1}^k = t_{j+1}^k - t_j^k$ at

k-th session:

$$f^*(d^k_{j+1}) = f(d^k_{j+1}|\mathbf{h}^{de_k}_j). \tag{6.12}$$

Since the lower-level LSTM is to model the time interval d^k_{j+1} and type a^k_{j+1} information, given a collection of activity sessions from benign employees, we combine the likelihood functions of the event type (Equation 6.9) and time (Equation 6.12) to have the negative joint log-likelihood of the observation sessions:

$$\mathcal{L}_a = -\sum_{k=1}^{M}\sum_{j=1}^{T_k}\Big(\log P(a^k_{j+1}|\mathbf{h}^{de_k}_j) + \log f^*(d^k_{j+1})\Big), \tag{6.13}$$

where M is the total number of sessions in the training dataset; T_k is the number of activities in a session. The lower-level LSTM along with the decoder LSTM is trained by minimizing the negative log-likelihood shown in Equation 6.13.

When the model is deployed for detection, to obtain the predicted activity type $\hat{a}^k_{j+1}$, we simply choose the type with the largest probability $P(a|\mathbf{h}^{de_k}_j)$ (calculated by Equation 6.9):

$$\hat{a}^k_{j+1} = \operatorname*{argmax}_{a\in\mathcal{A}} P(a|\mathbf{h}^{de_k}_j). \tag{6.14}$$

We further calculate the expected inter-activity duration between $(j+1)$-th and j-th activities $\hat{d}^k_{j+1} = E(t^k_j)$:

$$\hat{d}^k_{j+1} = \int_{t^k_j}^{\infty} t f^*(t)dt. \tag{6.15}$$

The difference between $\hat{d}^k_{j+1}$ and the observed d^k_{j+1} will be used to calculate the fraudulent score in terms of the timing information of intra-session activities.

6.4.3 Inter-Session Insider Threat Detection

The inter-session duration is crucial for insider threat detection. To capture such information, we further incorporate an upper-level LSTM into the framework, which focuses on modeling the inter-session behaviors of employees. Specifically, the upper-level LSTM is trained to predict the inter-session duration between k-th and $(k-1)$-th sessions ($\Delta^k = t_1^k - t_{T_{k-1}}^{k-1}$) and the k-th session duration ($d^k = t_{T_k}^k - t_1^k$).

To predict the inter-session duration Δ^k, the input of the upper-level LSTM is from the last hidden state $\mathbf{h}_{T_{k-1}}^{en_{k-1}}$ of $(k-1)$-th session from the lower-level LSTM as shown in Equation 6.16, while to predict the k-th session duration d^k, the input of the upper-level LSTM is from the first hidden state $\mathbf{h}_1^{en_k}$ of k-th session as shown in Equation 6.17.

$$\mathbf{x}_{T_{k-1}}^{k-1} = \mathbf{U}\mathbf{h}_{T_{k-1}}^{en_{k-1}}, \tag{6.16}$$

$$\mathbf{x}_1^k = \mathbf{U}\mathbf{h}_1^{en_k}, \tag{6.17}$$

where $\mathbf{U}$ is an input weight matrix for the upper-level LSTM.

Then, we can get the hidden states ($\mathbf{h}_{T_{k-1}}^{k-1}$ and $\mathbf{h}_1^k$) of the upper-level sequence based on an LSTM model. Finally, the conditional density functions of the inter-session duration Δ^k and session duration d^k are:

$$f_s^*(\Delta^k) = f(\Delta^k|\mathbf{h}_{T_{k-1}}^{k-1}), \tag{6.18}$$

$$f_s^*(d^k) = f(d^k|\mathbf{h}_1^k), \tag{6.19}$$

where $f_s^*(\Delta^k)$ and $f_s^*(d^k)$ can be calculated based on Equation 6.11.

To train the upper-level LSTM, the negative log-likelihood of inter-session sequences

can be defined as:

$$\mathcal{L}_s = -\sum_{m=1}^{M'}\sum_{k=1}^{K_m}\big(\log f_s^*(\Delta_m^k) + \log f_s^*(d_m^k)\big), \tag{6.20}$$

where M' is the total number of inter-session level sequences in the training dataset; K indicates the number of sessions in an inter-session level sequence. In our experiments, we use the upper-level LSTM to model the employee sessions in a week. Then, M' indicates the total number of weeks in the training dataset, and K is the number of sessions in a week. The upper-level LSTM is trained by minimizing the negative log-likelihood shown in Equation 6.20. After training, the upper-level LSTM can capture the patterns of the inter-session duration and session duration.

When the model is deployed for detection, we calculate the predicted inter-session duration between k-th and $(k-1)$-th sessions $\hat{\Delta}^k$ and the k-th session duration $\hat{d}^k$, shown in Equations 6.21.

$$\hat{\Delta}^k = \int_{t_T^{k-1}}^{\infty} t f_s^*(t), \quad \hat{d}^k = \int_{t_1^k}^{\infty} t f_s^*(t)dt. \tag{6.21}$$

The difference between $\hat{\Delta}^k$ ($\hat{d}^k$) and the observed Δ^k (d^k) will be used to calculate the fraudulent score in terms of the session timing information.

6.4.4 Fraudulent Score

After obtaining the predicted session, we compare the generated times and types in a session with the observed session, respectively. For the activity types, we adopt the Bilingual Evaluation Understudy (BLEU) [88] score to evaluate the difference between the observed session and generated session. The BLEU metric was originally used for evaluating the similarity between a generated text and a reference text, with values closer to 1 representing more similar texts. BLEU is derived by counting matching n-grams in the generated text to

n-grams in the reference text and insensitive to the word order. Hence, BLEU is suitable for evaluating the generated sequences and the observed sequences. We define the fraudulent score in terms of intra-session activity type as:

$$score_a = 1 - BLEU(S_a^k, \hat{S}_a^k), \tag{6.22}$$

where S_a^k indicates the observed activity types in k-th session while $\hat{S}_a^k$ indicates the predicted session. If $score_a$ is high, it means the observed session is a potentially malicious session in terms of session activity types.

For the activity time, as shown in Equation 6.23, we define the fraudulent score in terms of intra-session activity time by computing the mean absolute error (MAE) of the predicted time of each activity with the observed occurring time:

$$score_t = \frac{1}{|S|} \sum_{j=1}^{|S|} |d_j^k - \hat{d}_j^k|. \tag{6.23}$$

Since the upper-level LSTM takes each session's first and last hidden states as inputs to predict the time lengths of sessions and inter-sessions, we can further derive the time scores by comparing the predicted time lengths with the observed ones. We define the fraudulent score in terms of inter-session duration as:

$$score_\Delta = |\hat{\Delta}^k - \Delta^k|. \tag{6.24}$$

Similarly, we define the fraudulent score in terms of session duration as:

$$score_d = |\hat{d}^k - d^k|. \tag{6.25}$$

Note that although $\hat{d}^k$ can be derived based on all the predicted activity time from lower-level LSTM, the error usually is high due to the accumulated error over the whole sequence. Hence, we use the upper-level LSTM to get the session time length. Finally, by combining Equations 6.22, 6.23, 6.24 and 6.25, we define the total fraudulent score (FS) of a session as:

$$FS = \alpha_1 score_a + \alpha_2 score_t + \alpha_3 score_d + \alpha_4 score_\Delta, \tag{6.26}$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are hyper-parameters, which can be set based on the performance of insider threat detection via using each score alone.

6.5 Experiments

6.5.1 CERT.

We adopt the CERT Insider Threat Dataset [62] for the evaluation. Before that, we join all the log files, separate them by each employee, and then sort the activities of each employee based on the recorded timestamps. We randomly select 2000 benign employees as the training dataset and another 500 employees as the testing dataset. The test dataset includes all sessions from five insiders. The statistics of the training and testing datasets is shown in Table 6.1. Based on the activities recorded in the log files, we extract 19 activity types shown in Table 6.2. The activity types are designed to indicate the malicious activities.

Table 6.1: Statistics of Training and Testing Datasets

	Training Dataset	Testing Dataset
# of Employees	2000	500
# of Sessions	1039805	142600
# of Insiders	0	5
# of Malicious Sessions	0	68

Baselines. We compare our model with two one-class classifiers: 1) One-class SVM **(OCSVM)**

Table 6.2: Operation Types Recorded in Log Files

Files	Operation Types
logon.csv	Weekday Logon (employee logs on a computer on a weekday at work hours)
	Afterhour Weekday Logon (employee logs on in a weekday after work hours)
	Weekend Logon (employees logs on at weekends)
	Logoff (employee logs off a computer)
email.csv	Send Internal Email (employee sends an internal email)
	Send External Email (employee sends an external email)
	View Internal Email (employee views an internal email)
	View external Email (employee views an external email)
http.csv	WWW Visit (employee visits a website)
	WWW Download (employee downloads files from a website)
	WWW Upload (employee uploads files to a website)
device.csv	Device Connect (employee connects a device at weekday working hours)
	After-hour Device Connect (employee connects a device in weekday after-hours)
	Weekend Device Connect (employee connects a device at weekends)
	Disconnect Device (employee disconnects a device)
file.csv	Open doc/jpg/txt/zip File (employee opens a doc/jpg/txt/zip file)
	Copy doc/jpg/txt/zip File (employee copies a doc/jpg/txt/zip file)
	Write doc/jpg/txt/zip File (employee writes a doc/jpg/txt/zip file)
	Delete doc/jpg/txt/zip File (employee deletes a doc/jpg/txt/zip file)

[25] adopts support vector machine to learn a decision hypersphere around the positive data, and considers samples located outside this hypersphere as anomalies; 2) Isolation Forest (**iForest**) [89] detects the anomalies with short average path lengths on a set of trees. For both baselines, we consider each activity type as an input feature and the feature value is the number of activities of the corresponding type in a session. In this paper, we do not compare with other RNN based insider threat detection methods (e.g., [21]) as these methods were designed to detect the insiders or predict the days that contain insider threat activities.

Hyperparameters. We map the extracted activity types to the type embeddings. The dimension of the type embeddings is 50. The dimension of the LSTM models is 100. We adopt Adam [85] as the stochastic optimization method to update the parameters of the framework. When training the upper-level LSTM by Equation 6.20, we fix the parameters in the lower-level LSTM and only update the parameters in the upper-level LSTM.

Experiment Results. We aim to detect all the 68 malicious sessions from the totally

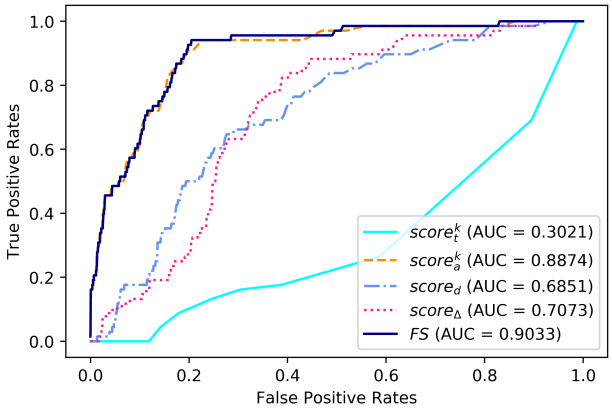


Figure 6.2: ROC curve of malicious session detection using various fraudulent scores

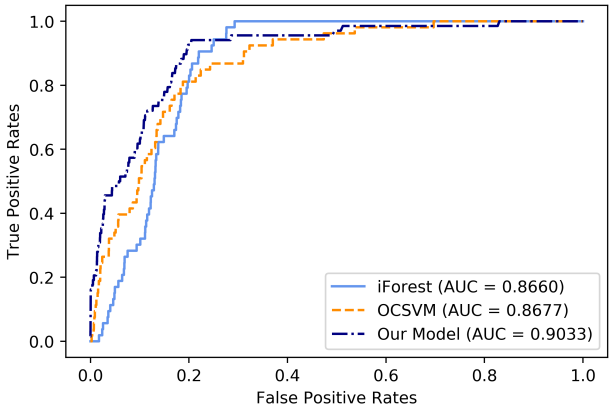


Figure 6.3: ROC curve of malicious session detection using various approaches

142,600 sessions in the testing set. Figure 6.2 shows the receiver operating characteristic (ROC) curves of our model for insider threat detection by leveraging various fraudulent scores. By using each fraudulent score separately, we can notice that the $score_a$ derived from intra-session activity types achieves the highest area under cure (AUC) score, which indicates the activity types of malicious sessions are different from the normal sessions. Meanwhile, the session duration time and inter-session duration time also make positive contributions to the malicious session detection. The $score_d$ derived from the session duration time and $score_\Delta$ derived from the inter-session duration time achieve good performance with AUC=0.6851 and 0.7073, respectively, which indicates the duration of malicious sessions and inter-sessions

are usually different from those of normal sessions. We also notice that the $score_t$ based on the inter-activity activity time information does not help much on insider threat detection. The AUC derived from $score_t$ is 0.3021. After examining the data, we find that there is no much difference in terms of inter-activity time information between malicious sessions and normal sessions. Since adopting $score_t$ does not achieve reasonable performance in the CERT Insider Threat dataset, we set $\alpha_2 = 0$ when deriving the total insider threat detection FS. As a result, our detection model using the total insider threat detection FS, which combines all the intra- and inter-session information, achieves the best performance with the AUC=0.9033 when the hyper-parameters in Equation 6.26 are $\alpha_1 = 1, \alpha_2 = 0, \alpha_3 = 1, \alpha_4 = 1$.

Figure 6.3 further shows the ROC curves of our model and two baselines. We can observe that our model achieves better performance than baselines in terms of AUC score. Especially, we can notice that when our model only adopts the activity type information ($score_a$) for malicious session detection, our model is slightly better than baselines in terms of AUC. With further combining the activity time and type information, our model significantly outperforms the baselines with the AUC=0.9033.

6.5.2 Wikipedia.

Due to the limitation of the CERT dataset where the inter-activity duration times are randomly generated, the inter-activity time in the intra-session level does not make contributions to the insider threat detection. To further show the advantage of incorporating activity time information, we apply our model for detecting vandals on Wikipedia. Vandals can be considered as insiders in the community of Wikipedia contributors. The study has shown that the behaviors of vandals and benign users are different in terms of edit time, e.g., vandals make faster edits than benign users [11]. Hence, we expect that using the inter-

activity time information can boost the performance of vandal detection. We adopt half of the benign users for training and the other half of the benign users and all the vandals for testing. Since user activities on Wikipedia do not have explicit indicators, such as LogON or LogOff, to split the user activity sequence into sessions, we consider user activities in a day as a user session. As a result, the session duration is always 24hrs, and the inter-session duration is 0. Therefore, in this experiment, we focus on vandalism session detection with only using information from the intra-session level and adopt the lower-level LSTM shown in Figure 6.1 accordingly. Note that we filter out all the sessions with number of activities less than 5. The seq2seq model takes a feature vector as an input and predicts the next edit time and type. In this experiment, we consider the activity type as whether the current edit will be reverted or not. The feature vector of the user's t-th edit is composed by: (1) whether or not the user edited on a meta-page; (2) whether or not the user consecutively edited the pages less than 1 minute, 3 minutes, or 5 minutes; (3) whether or not the user's current edit page had been edited before.

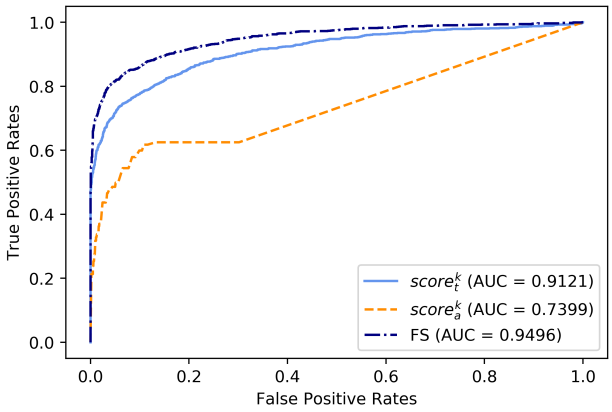


Figure 6.4: ROC curve of vandalism session detection using various fraudulent scores

Experiment Results. From Figure 6.4, we can observe that only using the inter-activity time information can achieve surprisingly good performance on vandalism session detection with AUC=0.9121, which indicates the inter-activity time information is crucial for vandal-

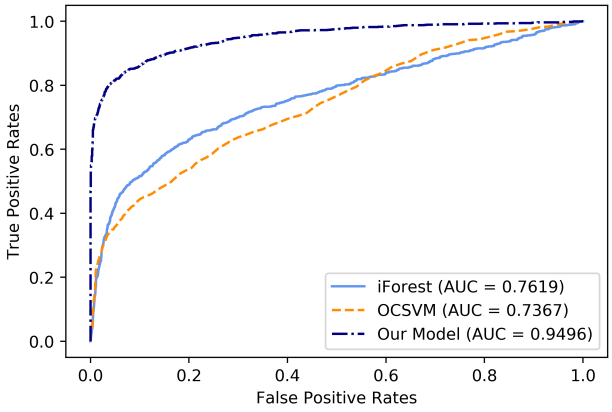


Figure 6.5: ROC curve of vandalism session detection using various approaches

ism session detection. Meanwhile, adopting the activity type information can also achieve the vandalism session detect with AUC=0.7399. Hence, using inter-activity time information achieves better performance than using the activity type information in terms of AUC. It also means vandals have significantly different patterns in activity time compared with benign users. Finally, with combining the activity type and time information, our model can achieve even better performance with AUC=0.9496.

We further compare our model with two baselines, i.e., One-class SVM and Isolation Forest. For the baselines, we consider the same features as the seq2seq model and further combine activity types. The value of each feature is the mean value of the corresponding feature in a day. Figure 6.5 indicates that our model significantly outperforms baselines in terms of AUC on the vandalism session detection task. Similar to the results on the CERT dataset, when our model only adopts the activity type information ($score_a$ shown in Figure 6.4), the model achieves similar performance as baselines. With considering the activity time information, the performance of our model is improved by a large margin.

6.6 Summary

In this chapter, we have proposed a hierarchical neural temporal point process model for insider threat detection. Experimental results, which are perfomred on an insider threat detection dataset and a Wikipedia vandal detection dataset, demonstrate the effectiveness of our model.

7 Using Dirichlet Marked Hawkes Processes for Insider Threat Detection

Problem Statement: *Could we develop a more flexible framework for dynamic fraud detection*?

In this chapter, we first introduce the problem background, then formulate the problem and present the proposed method. The two datasets, CRET and Wikipedia, will be used to evaluate the effectiveness of the proposed method.

7.1 Introduction

Various insider threat detection approaches have been proposed [90, 20, 22, 91, 19, 21, 92]. However, the existing works usually focus on detecting malicious sessions, e.g., a sequence of activities in a day, and cannot achieve the dynamic malicious activity detection. In this paper, we aim to develop a detection model that can dynamically detect the malicious activities. To this end, besides adopting the activity type (e.g., WWW visit, send email) information, we further consider the crucial activity time information since the time an activity occurred is also a strong indicator about whether the activity is malicious. For example, if a user usually does not work after-hours, an activity, say connecting a USB flash drive, from that user happens at midnight is potentially malicious.

The intuitive idea of detecting malicious insider activities is to model the occurrence likelihoods of activities in terms of activity type and time. Since the majority of activities are benign even for the insiders, the likelihoods of benign activities should be high. Hence, if an activity is occurred with a low likelihood in terms of activity type and time, it could be a malicious activity. In this work, we adopt the Marked Temporal Point Process (MTPP),

which is a general mathematical framework, to model the user activities over time. Specifically, we adopt the Hawkes process to model the temporal information of user activities, while the activity types are captured by a mark model. However, the activity sequence from a user consists of various temporal patterns that are driven by his working patterns. For example, if the user is in the work mode, his activities are dense in a short time window, and the activity types mainly consist of file editing. If the user is in the leisure mode, his activities would be sparse in a time window, and the activity types may be more about Web surfing. We call the different working patterns *user modes*. Hence, only using one marked Hawkes process model is unable to capture the dynamic user activities under various user modes. Meanwhile, due to the potential infinite number of user activities, the number of user modes is also large and unknown. Hence, it is also hard to use a fixed number of marked Hawkes processes to model the complicated user activities. The effectiveness of modeling the sequence of activities and the following detecting malicious insider activities depends on how to capture the unbounded user modes from the observed sequence.

In this paper, we develop the Dirichlet Marked Hawkes Process (DMHP) to detect malicious insider threats. DMHP adopts the Dirichlet process to determine the prior distribution of user modes. Each mode has its own temporal dynamics in terms of activity type and time. Hence we adopt one set of marked Hawkes processes to model the sequence of user activities in each specific mode. For each particular activity in the sequence, we then derive an occurrence likelihood based on DMHP and use it as an evaluation score for predicting whether or not the activity is malicious.

Our work makes the following contributions: (1) we develop a dynamic and unsupervised insider threat activity detection model that considers both activity type and time information; (2) to this end, we propose a Dirichlet Marked Hawkes Process that can ef-

fectively capture the sequence of user activities with unbounded number of modes; (3) we conduct experiments on two datasets, the CERT Insider Threat Dataset [62] and the UMD-Wikipedia dataset [11], and evaluation results demonstrate the effectiveness of our model for malicious activity detection. We emphasize that insider attacks are subtle and dynamic and the number of malicious insider activities is only a very small fraction of all activities from insiders (e.g., 0.01% to 0.3% in CERT data). Detection models based on supervised learning usually do not work here due to lack of labeled insider activity record. Detection models based on few-shot learning may not work either as the few known insider attack instances in training data are often very different from test data. Our DMHP detection model is based on unsupervised learning, does not assume any labeled attacks record, and can score each coming activity in a real-time manner.

7.2 Dirichlet Process

The Dirichlet process (DP) is a Bayesian nonparametric model, which is parameterized by a concentration parameter $\eta > 0$ and a base distribution G_0 over a space Θ. It indicates that a random distribution G drawn from DP is a distribution over Θ, denoted as $G \sim DP(\eta, G_0)$. The expectation of the distribution G is the base distribution G_0. The concentration parameter controls the variance of G, where the higher values of η lead to tight distributions around G_0. DP is widely used for clustering with the unknown number of clusters.

Chinese restaurant process (CRP) is one kind of representations for Dirichlet process. CRP assumes a restaurant with an infinite number of tables, each of which can seat an infinite number of customers. Within the context of clustering, each table indicates a cluster while each customer is a data point. The simulation process of CRP is as follows:

Table 7.1: Summary of the notation

Symbol	Definition
(t_n, a_n)	The n-th activity occurring at time t_n with activity type a_n.
z_n	The mode assignment for activity (t_n, a_n).
D	The category number of activity types.
K	The number of modes we have till t_n.
L	The number of triggering kernels for one specific mode.
λ_0	The base intensity of Hawkes process.
$(\alpha_1^{'}, \ldots, \alpha_L^{'})$	The concentration parameter of Dirichlet prior for Hawkes process.
$(\theta_1^{'}, \ldots, \theta_D^{'})$	The concentration parameter of Dirichlet prior for categorical distribution.
$s(t_n, a_n)$	The combined score provided by the Dirichlet mode-specific model for (t_n, a_n).
ϵ	The insider activity detection threshold.
ϕ_n^c	The weight of particle c at timestamp n.

1. The first customer always sits at the first table.

2. Customer n $(n > 1)$ is distributed to:

   (a) a new table with probability $\frac{\eta}{\eta+n-1}$.

   (b) an existing table z with probability $\frac{n_z}{\eta+n-1}$ where n_z is the number of customers at table z.

Let $\beta_1, ..., \beta_n$ be a sequence sampled from CRP. The conditional distribution of β_n can be written as:

$$\beta_n|\beta_{1:n-1} \sim \frac{1}{\eta + n - 1}\big(\eta G_0 + \sum_z n_z \delta_{\beta_z}\big), \tag{7.1}$$

where δ_{β_z} is a point mass centred at β_z. According to Eq. 7.1, we can see that the probability to be distributed to a new table is associated with concentration parameter η and the likelihood of an upcoming sample β_n to be allocated to an existing table z is proportional to the table size n_z. In other words, the table with a bigger size has more opportunity to serve the upcoming customer. Therefore, we can see DP has a clustering property ("The rich gets richer") for the online streaming data.

7.3 Framework

Considering an activity sequence $\mathcal{E} = \{e_1, \ldots, e_n, \ldots\}$, $e_n = (t_n, a_n)$ indicates the activity with type a_n occurring at time t_n, where $a_n \in \mathcal{A} = \{1, \ldots, D\}$ and D is the number of activity types. Our insider threat detection task is to detect anomalous activities that have different behavior patterns with normal ones. For this task, we develop the Dirichlet Marked Hawkes Process (DMHP) framework that is an infinite mixture model based on the Chinese restaurant process (Section 7.2). Our mixture model assumes that, underlying the activity stream $\mathcal{E}$, there potentially exists a countable-but-unknown number of hidden clusters $Z = \{z_k\}_{k=1}^{K}$ where $z_k \in \mathbb{Z}$ and K, like the number of occupied *tables* in CRP, represents the number of hidden clusters. Note that K is dynamically changing along the activity stream and its value is fixed-but-unknown at one specific moment. In our insider threat detection scenario, we call these hidden clusters as *user modes* and use the term *mode* throughout the paper. Table 7.1 shows the summary of notations. Our DMHP adopts the Dirichlet process to determine the prior distribution of user modes. Each mode has its own temporal dynamics in terms of activity type and time. We adopt one set of marked Hawkes processes to describe each mode. To some extent, we can take each user mode as one dimension and all of the existing user modes construct a *multi-dimensional space*. From this point of view, in contrast with the hierarchical RNN in Chapter 6, a detection framework with two dimensions (scales), DMHP is a mode-based multi-scale detection model with unlimited dimensions. For each activity in the streaming sequence, we calculate a combined likelihood score based on the current K modes we have at hand and use the score to predict whether the activity is malicious.

7.3.1 Marked Hawkes Process with Prior

In this section, we develop a marked Hawkes process model with Dirichlet priors to capture both *time* and *type* information.

Time. We adopt Hawkes process to model the *time* information. Given an observed activity stream $(t_1, a_1), \ldots, (t_{n-1}, a_{n-1})$, for an upcoming activity (t_n, a_n) with an unknown mode assignment z_n, based on Eq. 6.4, the occurrence likelihood associated with *time* is measured by

$$p(t_n|z_n, t_{1:n-1}) = \lambda^*_{z_n}(t_n) \cdot exp\Big(- \int_{t_{n-1}}^{t_n} \lambda^*_{z_n}(\tau) d\tau \Big), \tag{7.2}$$

and its intensity value is as follow

$$\lambda^*_{z_n}(t_n) = \lambda_0 + \sum_{t_i < t_n} \gamma_{z_n}(t_n, t_i) \mathbb{1}[z_i = z_n], \tag{7.3}$$

where z_i refers to the mode assigned to the i-th activity and $\gamma_{z_n}(t_n, t_i)$ is the triggering function of mode z_n with the form

$$\gamma_{z_n}(t_n, t_i) = \sum_{l=1}^{L} \alpha^l_{z_n} \cdot \kappa(\tau_l, t_n - t_i), \tag{7.4}$$

$$\boldsymbol{\alpha}_{z_n} \sim Dir(\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_L^{'}), \tag{7.5}$$

where $\kappa(\cdot)$ is the base kernel function, $\boldsymbol{\alpha}_{z_n} \in \mathbb{R}^L$ is the weight vector for L different time scale kernels, $\boldsymbol{\alpha}^{'} = (\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_L^{'}) \in \mathbb{R}^L$ are the corresponding hyperparameters of Dirichlet prior, and τ_l is the typical reference time points.

Type. We take use of the categorical distribution with a Dirichlet prior to model the

upcoming activity type a_n underlying mode z_n

$$p(a_n|z_n, y_{1:n-1}) \sim Cat(\boldsymbol{\theta}_{z_n}), \tag{7.6}$$

$$\boldsymbol{\theta}_{z_n} \sim Dir(\theta_1^{'}, \theta_2^{'}, \ldots, \theta_D^{'}), \tag{7.7}$$

where $\boldsymbol{\theta}_{z_n} \in \mathbb{R}^D$ is the parameter vector for categorical distribution with mode z_n and $\boldsymbol{\theta'} = (\theta_1^{'}, \theta_2^{'}, \ldots, \theta_D^{'}) \in \mathbb{R}^D$ is the corresponding hyperparameters of Dirichlet prior. Then, based on the historical activity types, by the Dirichlet-Categorical conjugate relation, the occurrence likelihood associated with *type* can be concretely represented as

$$p(a_n = d|z_n, y_{1:n-1}) = \frac{C^{a_n=d,z_n} + \theta_{a_n=d}^{'}}{C^{z_n} + \sum_{i=1}^{D} \theta_i^{'}}, \tag{7.8}$$

where $C^{a_n=d,z_n}$ is the number of activities with type d in mode z_n excluding the current activity, $\theta_{a_n=d}^{'}$ is the d-th element of hyperparameter vector $\boldsymbol{\theta'}$ that is related to activity type a_n, and C^{z_n} is the total number of activities in mode z_n excluding the current activity.

Time & Type. Given the history of past activities, the conditional density function that the upcoming activity will happen at time t_n with type a_n can have different forms. In practice, we typically choose some simple factorized formulation to reduce the excessive complications caused by jointly and explicitly modeling the time and type information. In our paper, by Eq.7.2 and 7.8, the joint occurrence likelihood of the upcoming activity (t_n, a_n) under user mode z_n provided by marked Hawkes process is formulated as

$$p(t_n, a_n|z_n, t_{1:n-1}, a_{1:n-1}) = p(a_n|z_n, a_{1:n-1}) \cdot p(t_n|z_n, t_{1:n-1}). \tag{7.9}$$

7.3.2 Dirichlet Marked Hawkes Process

Our proposed Dirichlet Marked Hawkes Process (DMHP) model is one mixture model and consists of a set of marked Hawkes processes with prior. As a non-parametric Bayesian model, DMHP can be taken as a generative model. Its equivalent generative process for one user-specific activity stream can be described as follows.

1. Initialize parameters: base intensity λ_0, hyperparameters for Dirichlet priors, $(\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_L^{'})$ and $(\theta_1^{'}, \theta_2^{'}, \ldots, \theta_D^{'})$

2. Draw α_1 from $\mathrm{Dir}(\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_L^{'})$ and θ_1 from $\mathrm{Dir}(\theta_1^{'}, \theta_2^{'}, \ldots, \theta_D^{'})$

3. Draw t_1 from $\mathrm{Poisson}(\lambda_0)$ and $a_1 \sim \mathrm{Cat}(\theta_1)$

4. Assign $K \leftarrow 1$ and $z_1 \leftarrow K$

5. For $n > 1$:

   a. Draw time $t_n > t_{n-1}$ for the new activity from $\mathrm{Poisson}(\lambda_0 + \sum_{i=1}^{n-1} \gamma_{z_i}(t_n, t_i))$

   b. Draw the mode z_n for the new activity

$$p(z_n = k | t_{1:n}) = \begin{cases} \frac{\lambda^*_{z_n=k}(t_n)}{\lambda_0 + \sum_{i=1}^{n-1} \gamma_{z_i}(t_n, t_i)}, & k = 1, \cdots, K \\ \frac{\lambda_0}{\lambda_0 + \sum_{i=1}^{n-1} \gamma_{z_i}(t_n, t_i)}, & k = K+1 \end{cases} \tag{7.10}$$

   If $z_n = K{+}1$, draw $\boldsymbol{\alpha}_{z_n}$ from $\mathrm{Dir}(\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_L^{'})$, draw $\boldsymbol{\theta}_{z_n}$ from $\mathrm{Dir}(\theta_1^{'}, \theta_2^{'}, \ldots, \theta_D^{'})$, $K \leftarrow K + 1$.

   c. Draw type for the new activity

$$a_n | z_n, a_{1:n-1} \sim \sum_d \frac{C^{a_n=d, z_n} + \theta^{'}_{a_n=d}}{C^{z_n} + \sum_{i=1}^{D} \theta_i^{'}} \cdot \delta_d, \tag{7.11}$$

where δ_d is a point mass centred at activity type d.

In the above process, K is a mode counter to record the total number of modes we have until now, k refers to some certain mode, and z_n represents the mode assignment of the n-th activity. In step 5a, we first draw time t_n for the new activity by a Poisson process parameterized by an intensity value that is derived from the historical activities and current user modes. The generation of the activity timing in the process can be viewed as the superposition of a Poisson process λ_0 and multiple Hawkes processes. In step 5b, we draw the mode z_n for the new activity with the probability mass function (shown in Eq. 7.10) that is constructed by Eq. 7.3. If it is a new mode, we draw its weight vector $\boldsymbol{\alpha}_{z_n}$ for the L time scale kernels, $\boldsymbol{\theta}_{z_n}$ to parameterize the categorical distribution and then increase K by one. In step 5c, we draw type for the new activity with the mass function constructed by Eq. 7.8.

7.3.3 Insider Malicious Activity Detection

In insider threat detection, our goal is to evaluate whether or not the most recently observed activity (t_n, a_n) is anomalous on the fly. The above Dirichlet marked Hawkes process gives a basic idea on how a sequence of samples $\{(t_n, a_n)\}$ is generated. In particular, we show in Eq. 7.10 of step 5b how to calculate the probability of mode z_n given the time sequence, i.e., $p(z_n = k|t_{1:n})$. In step 5c, we further show in Eq. 7.11 how to generate a_n given mode z_n and the previous type sequence $a_{1:n-1}$. This generative process gives us idea on how to distribute the most recent observed activity (t_n, a_n) to user modes in a probabilistic way. Formally, we have

$$p(z_n = k|t_{1:n}, a_{1:n}) \sim p(a_n|z_n = k, a_{1:n-1}) \cdot p(z_n = k|t_{1:n}). \tag{7.12}$$

We then derive a combined likelihood score for the most recent activity (t_n, a_n) given the previously observed activity stream $(t_1, a_1), \ldots, (t_{n-1}, a_{n-1})$. The score is based on current K modes and is formulated as

$$s(t_n, a_n) = \sum_{k=1}^{K} \underbrace{p(t_n, a_n | z_n = k, t_{1:n-1}, a_{1:n-1})}_{\text{intra-mode likelihood}} \cdot \underbrace{p(z_n = k | t_{1:n}, a_{1:n})}_{\text{mixture weight}}. \tag{7.13}$$

Note that $p(z_n = k|t_{1:n}, a_{1:n})$ measures the mixture weight of (t_n, a_n) from mode k given the whole activity sequence until (t_n, a_n) and $p(t_n, a_n|z_n = k, t_{1:n-1}, a_{1:n-1})$ is the probability of activity (t_n, a_n) occurring inside the given mode k. In our work, we assume *time* and *type* are conditionally independent given the user mode z_n and apply Eq. 7.9 to reduce the excessive complications.

Algorithm 2 shows the framework of our insider activity detection. We assume the first activity (t_1, a_1) is normal. For each activity (t_n, a_n) s.t. $n > 1$, lines 3-5 illustrate whether a new mode is needed for the upcoming activity. Lines 6-9 are the loop to iterate all of modes we have at hand to calculate the corresponding likelihood scores under specific modes, the mixture weights, and the final combined likelihood score. In particular, line 7 is for the computation of likelihood given a specific mode k; line 7 is for calculating the mixture weights over all the modes. Then, a combined likelihood score of the activity is derived in line 9. Given a predefined threshold ϵ, lines 10-15 are to evaluate whether activity (t_n, a_n) is malicious or not: if the combined likelihood score is below ϵ, it is labeled as an insider activity; otherwise, it is normal.

Algorithm 2: Insider Threat Detection via Dirichlet Marked Hawkes Process

Inputs : Activity stream $(t_1, a_1), \ldots, (t_n, a_n), \ldots$, concentration factors λ_0, $(\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_L^{'})$, $(\theta_1^{'}, \theta_2^{'}, \ldots, \theta_D^{'})$, and insider likelihood threshold ϵ.

Outputs: Real-time insider activity detection report $\mathcal{Y}$: 1 represents the insider activity and 0 indicates the normal one.

$\mathcal{Y} \leftarrow list[0]$; $K \leftarrow 1$

foreach *each activity* $(t_n, a_n), n = 2, \ldots$ **do**

  Draw $r \sim$ Uniform(0,1) **if** $r < \frac{\lambda_0}{\lambda_0 + \sum_{i=1}^{n-1} \gamma_{z_i}(t_n, t_i)}$ **then**

    Generate a new mode by Eq. 7.10 $K \leftarrow K + 1$

  end

  foreach $k \in \{1, \ldots, K\}$ **do**

    Compute the likelihood of (t_n, a_n) in mode k by Eq. 7.9 Compute the weight of mode k for (t_n, a_n) by Eq. 7.12

  end

  Compute the combined likelihood score $s(t_n, a_n)$ by Eq. 7.13

  if $s(t_n, a_n) < \epsilon$ **then**

    $\mathcal{Y}$.insert(1)

  end

  else

    $\mathcal{Y}$.insert(0)

  end

end

return $\mathcal{Y}$

Our DMHP detection framework consists of two components, *time* and *type*. For an ablation evaluation, we have listed three variants of our proposed model based on different input information:

1. **Time&type** model takes both *time* and *type* information as input and it is exactly depicted in Algorithm 2.

2. **Type** model only uses *type* information as input. To formulate this model, we need do

the following adaptions based on Algorithm 2:

(a) In line 7, to derive likelihood score, we replace Eq. 7.9 with Eq. 7.8.

(b) In line 7, to calculate the mixture weights, we rewrite Eq. 7.12 as $p(z_n = k|t_{1:n}, a_{1:n}) \sim p(z_n = k|a_{1:n})$.

3. **Time** model only uses activity *time* information as input. Given Algorithm 2, we need do the following adaptions:

(a) In line 7, to fix likelihood score, we replace Eq. 7.9 with Eq. 7.2.

(b) In line 7, for the computation of mixture weights, we rewrite Eq. 7.12 as $p(z_n = k|t_{1:n}, a_{1:n}) \sim p(z_n = k|t_{1:n})$.

Given an observed activity stream $(t_1, a_1), \ldots, (t_{n-1}, a_{n-1})$, our goal is to judge whether the most recent activity (t_n, a_n) is abnormal via a combined likelihood score. In this *on-the-fly* detection, one challenge is how to efficiently derive the latent variables, *user mode* z_n, from the posterior distribution $p(z_n = k|t_{1:n}, a_{1:n})$ (Eq. 7.12). To address this issue, we adopt the idea of [93] that reuses the past samples from $p(z_{1:n-1}|t_{1:n-1}, a_{1:n-1})$. Specifically, we apply the sequential Monte Carlo algorithm and take advantage of a set of $\mathcal{C}$ particles, each of which represents a hypothesis of the latent mode and is sampled from a proposal distribution $q(z_{1:n}|a_{1:n}, t_{1:n})$, to sequentially approximate the $p(z_{1:n}|a_{1:n}, t_{1:n})$. Moreover, it would be very important that, for streaming data, the expected time cost of sampling the *user mode* z_n and updating the triggering kernel parameters αs should not grow with the amount of observed activities. We set up a predefined window size and only consider the activities within the observed window and neglect the ones outside the window.

Given an activity sequence, we apply a Sequential Monte Carlo (SMC) to infer the latent variables, i.e., user modes, in our DMHP framework. Generally speaking, the inference

algorithm consists of two parts: *sampling user modes* and *updating triggering kernel*.

7.3.4 Sampling User Mode

Sequential Monte Carlo is used to get an approximation for the posterior $p(z_n|t_{1:n}, a_{1:n})$ in which a number of particles are maintained that each of particles refers to an instance of latent variable (i.e., a hidden mode) and has a weight to tell how well this instance can interpret the data. The weight of particle c at timestamp n goes as follows

$$\phi_n^c = \frac{p(z_{1:n}|a_{1:n}, t_{1:n})}{q(z_{1:n}|a_{1:n}, t_{1:n})}, \tag{7.14}$$

where $c \in \{1, \ldots, \mathcal{C}\}$ and $\mathcal{C}$ is the total number of particles; $p(\cdot)$ is the true posterior distribution, and $q(\cdot)$ is the corresponding proposal distribution.

To minimize the variance of the resulting particle weight, we take the posterior distribution $p(z_n|z_{1:n-1}, a_{1:n}, t_{1:n})$ as the proposal distribution $q(\cdot)$. Then, given the weight of particle c at timestamp $n-1$, *i.e.* ϕ_{n-1}^c, the unnormalized weight ϕ_n^c can be recursively updated by

$$\phi_n^c = \phi_{n-1}^c \cdot p(a_n|z_n^c, a_{1:n-1}). \tag{7.15}$$

7.3.5 Updating Triggering Kernel

For a stream of activities $(t_1, a_1), \ldots, (t_n, a_n)$, with current user activity (t_n, a_n), time parameter $\boldsymbol{\alpha}_{z_n}$ should be updated before the next activity is coming. Following the work [93], a set of samples $\boldsymbol{\alpha}$s are drawn from a Dirichlet distribution, $\{\boldsymbol{\alpha}_{z_n}^i\}_{i=1}^N \sim Dir(\alpha_1', \alpha_2', \ldots, \alpha_L')$ where N is the sampling size and these $\boldsymbol{\alpha}$s are used to obtain an estimation of the time parameter via maximizing the likelihood of activity streaming in terms of *time* information (Eq. 6.5).

Algorithm 3: Inference algorithm of the DMHP

```
1  Initialize φ_1^c to 1/C for all c ∈ {1...C} foreach each activity timing t_n, n = 1, 2, ... do
2      foreach c ∈ {1, ..., C} do
3          if one activity type a_n occurs at the time t_n then
4              sample mode z_n with Eq. 7.12 update triggering kernel parameter α by the sampling
                method stated above. update particle weights with Eq. 7.15
5          end
6      end
7      Normalize particle weights if ||φ_n||_2^{-2} < threshold then
8          resample particles;
9      end
10 end
```

Algorithm 3 shows the sequential Monte Carlo framework for mode sampling and triggering kernel updating. Line 1 initializes the particles and, for one specific activity, lines 3-9 iterate all of particles to sample the mode and update the time parameter: line 5 is for the mode sampling, line 6 aims to trigger kernel updating, and line 7 is to update the particle weights. In addition, lines 11-13 show the rules for particle resampling after the particle weight normalization (line 10).

7.4 Experiments

7.4.1 Experiment Setup

Dataset. We adopt the CERT Insider Threat Dataset [62]. To describe employees' action behavior, instead of the five original coarse-grained categories for activities, we apply fine-grained categories to describe the employees' operations (Table 6.2).

The CERT dataset contains 3995 benign employees and 5 insiders. For each user, it records activities from January 2010 to June 2011. On average, the number of activities for each employee is around 40000. We preprocess the original dataset by joining all the log files,

grouping them by each employee, and then sorting the activities of each employee based on the recorded timestamps. We include all 5 insiders and randomly select 5 benign employees in our experiment. Table 7.3 shows statistics for 5 insiders, the number of activities of each insider in column 2 and the number of malicious activities of each insider in column 3. We can see that malicious activities take up a very small percentage of all activities: ACM2278 with 0.0701% (22/31370), CDE1846 with 0.3549% (134/37754), CMP2946 with 0.3920% (243/61989), MBG3183 with 0.0094% (4/42438), and PLJ1771 with 0.0858% (18/20964). It is challenging to detect effective insider threat detection algorithm due to the extremely low percentage of malicious activities.

In our experiment, we treat each employee's activities as one sequence, each of which is sorted by the activity occurrence time, and we have 10 streams which are from 5 insiders above and 5 normal users randomly selected. We feed them into our proposed DMHP algorithm, one after another, and then obtain the combined likelihood score of each activity for each specific employee dynamically. Given an empirical threshold ϵ, each activity in the stream can be judged by the real-time combined likelihood score: it is labeled as an insider activity if the corresponding combined likelihood score is less than τ; otherwise, it is labeled as a normal activity.

Hyperparameters. In all our experiments, we set the base intensity of Hawkes process $\lambda_0 = 0.1$, the concentration parameter of Dirichlet prior for Hawkes process $(\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_L^{'})$ and that for categorical distribution $(\theta_1^{'}, \theta_2^{'}, \ldots, \theta_D^{'})$, the number of RBF Kernels $L = 7$. The typical reference time points τ_l for 7 kernels are 3, 7, 11, 24, 48, 96, 192 hours with the corresponding bandwidths 1, 5, 7, 12, 24, 24, 24, respectively. We do not specify the exact value of likelihood threshold ϵ; instead we rank each activity based on its likelihood value and report the number of real malicious activities in the certain percentage of ranked

activities.

Baselines. Our DMHP *time&type* uses both type and time information in the modeling. In our evaluation, we compare it with five baselines, DMHP *time*, DMHP *type*, Isolation Forest [89], Local Outlier Factor [94], and ADeMS [95]. DMHP *time* only uses time information as input and DMHP *type* only uses activity type information. Hence the comparison with these two baselines can be considered as the ablation study. Isolation Forest and Local Outlier Factor are two typical unsupervised anomaly detection methods. Isolation Forest detects anomalies based on the concept of isolation without employing distance or density metrics whereas Local Outlier Factor identifies anomalies by measuring the local deviation of a given data point with respect to its neighbors. ADeMS is one of recent state-of-the-art streaming-based anomaly detection algorithms. ADeMS maintains a set of few orthogonal vectors to capture the prior from the historical non-anomalous data points and use a reconstruction error test to evaluate the anomaly of upcoming data. On the implementation, we take use of the corresponding scikit-learn packages for Isolation Forest and Local Outlier Factor, and the GitHub implementation for ADeMS. To adapt the input of baselines, for CERT dataset, we encode each activity by a 42-D vector in which 24-D is for time and 18-D is for activity types.

Training & testing. The activities in the first one and a half months are used to train the proposed model and baselines. We think the dynamic models would stabilize after training with activities of the first 1.5 months. All the rest activities (16.5 months) are employed for evaluation. In all models, we also treat all *WWW_visit* activities as normal as *WWW_visit* activities trivially contribute to the insider threat detection.

Table 7.2: Category-specific AUC values for compared models

	Logon	Email	Http	Device	File
Time&type	**0.9592**	**0.9314**	**0.9844**	**0.9535**	**0.9585**
Type	0.6931	0.4931	0.7616	0.6884	0.7907
Time	0.4982	0.5970	0.4405	0.4392	0.4738
Local outlier factor	0.7622	0.6811	0.8188	0.6194	0.8831
Isolation Forest	0.8352	0.9260	0.8180	0.8462	0.8797
ADeMS	0.9437	0.9204	0.9533	0.9334	0.9457

Table 7.3: Statistics of insiders: the total number of activities and the number of true malicious activities located in the top 15% of total activities with the lowest combined likelihood scores.

Insider	Total activity #	Total malicious activity #	*Time model*	*Type model*	*Time & Type* model	*Isolation forest*	*Local outlier factor*	*ADeMS*
ACM2278	31370	22	0	22	**22**	5	7	22
CDE1846	37754	134	26	76	**134**	134	83	130
CMP2946	61989	94	13	23	**91**	48	25	72
MBG3183	42438	4	0	4	**4**	0	4	4
PLJ1771	20964	15	0	9	**15**	11	6	11

7.4.2 Malicious Insider Activity Detection

Figure 7.1 shows the receiver operating characteristic (ROC) curves and the corresponding area under the ROC curve (AUC) values of our DMHP models (*time&type*, *type*, *time*), Local Outlier Factor, and Isolation Forest. We can see that our DMHP *time&type* achieves the best accuracy with the AUC value of 0.9509, which is significantly higher than *time* (AUC 0.5031) and *type* (AUC 0.6530). The reason behind this can summarized as follows. First, our *time* model is a self-exciting Hawkes process that the occurrence likelihood of a new event increases due to the just occurred events (Eq. 6.6); therefore, if series of malicious activities occur consecutively in a very short period of time, e.g., the malicious activities of *ACM2278* shown in the Table 7.4, our *time* model tends to provide high likelihoods for these malicious activities. Second, in general, our *type* model is a frequency-based model that it tends to assign a high (low) likelihood to the activity type with a high (low) frequency. Notably, in CERT dataset, some of malicious activities are of high-frequency, e.g., *email-in*,

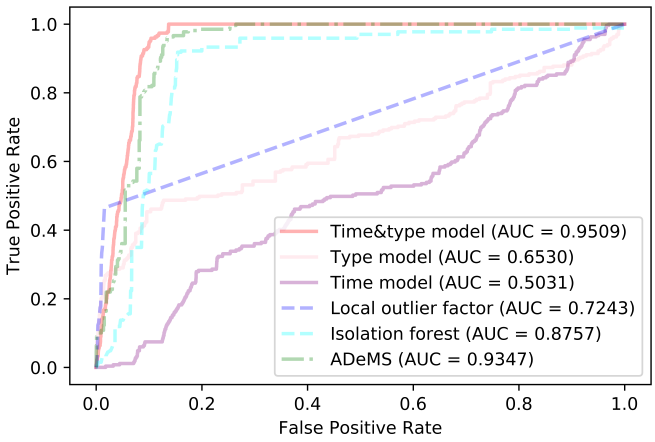


Figure 7.1: ROC for insider malicious activity detection

email-out or *website visit*, which makes the *type* model difficult to achieve high accuracy. However, as shown in Figure 7.3, our *time&type* model, which combines the *time* and *type* models, assigns a smaller likelihood to the malicious activity. This demonstrates the advantage of combining both activity type and time information in sequence modeling for insider threat detection. In addition, our DMHP *time&type* also significantly outperforms Local Outlier Factor (AUC 0.7243), Isolation Forest (AUC 0.8757), and ADeMS (AUC 0.9347).

For a comprehensive evaluation, we report in Table 7.2 the corresponding AUC values from all compared models for each of the five *log* categories, i.e., *logon*, *email*, *http*, *device*, and *file*.

We can observe that *time&type* obtains the highest AUC values in all five categories, *logon* 0.9592, *email* 0.9314, *http* 0.9844, *device* 0.9535, and *file* 0.9585. For baselines, Isolation Forest and ADeMS both perform well on all of activity types, while Local Outlier Factor achieves good AUC values in *http* and *file* but poor AUC values in *logon*, *email* and *device*. We also observe that DMHP with time information alone does not achieve good performance across all five categories.

We also show in Table 7.3 a detailed comparison of model performance for each

insider. We calculate the number of true malicious activities in the top 15% activities with the lowest combined likelihood scores from each model. As shown in columns 2-3, the malicious activities take up a very small percentage of activities of insiders. Moreover, those malicious activities are often hidden among normal activities and occur in a long period. Our *time&type* achieves highest recalls for all insiders except *CMP2946*. Concretely, it captures 22 real malicious activities for *ACM2278* with recall $22/22 = 100\%$, 128 real malicious activities for *CDE1846* with recall $134/134 = 100\%$, 91 real malicious activities for *CMP2946* with recall $91/94 = 96.80\%$, 4 real malicious activities for *MBG3183* with recall $4/4 = 100\%$, 15 real malicious activities for *PLJ1771* with recall $15/15 = 100\%$. Generally, ADeMs also performs well in the malicious activity detection via likelihood score ranking. Isolation Forest achieves its highest recall 100% for *CDE1846*. However, it obtains fairly low recall values for other insiders, e.g., *ACM2278* 22.72% and *MBG3183* 0%. This is because malicious activities in *ACM2278* and *MBG3183* are heavily associated with *http* and Isolation Forest seems to be not good at capturing *http* behavior pattern, which is demonstrated by the comparably lower AUC in Table 7.2.

7.4.3 Case Study

In accordance with Table 7.3, the high recall values provided by DMHP *time&type* model in the top 15% activities lead us to a hypothesis that DMHP *time&type* has a tendency to assign low (high) combined likelihood scores to the malicious (normal) activities and this point can be supported by the comparison of empirical distributions between normal and malicious activities shown in Figure 7.4.3 and 7.4.3. To further testify our hypothesis and provide a clear picture for malicious activity detection, we focus on one insider, *ACM2278*, as a case study.

With the scenario design of CRET dataset, we learn that *ACM2278* refers to *an insider who did not previously use removable drives or work after hours begins logging in after hours, using a removable drive, and uploading data to wikileaks.org and then, leaves the organization shortly thereafter*. There are 22 malicious activities carried by *ACM2278*, which form two concrete instances of insider behavior pattern: one instance is from activities 1 to 12 and another is from activities 13 to 22. For reproducibility purpose, we list them in Table 7.4. In both attack instances, the insider logs on the company's computer after work, connects the removable drives to computer, uploads data, and finally ends at the next day's morning before the working hour.

Table 7.4: Malicious insider activities of *ACM2278* (Activities 1-12 and 13-22 form two attack instances).

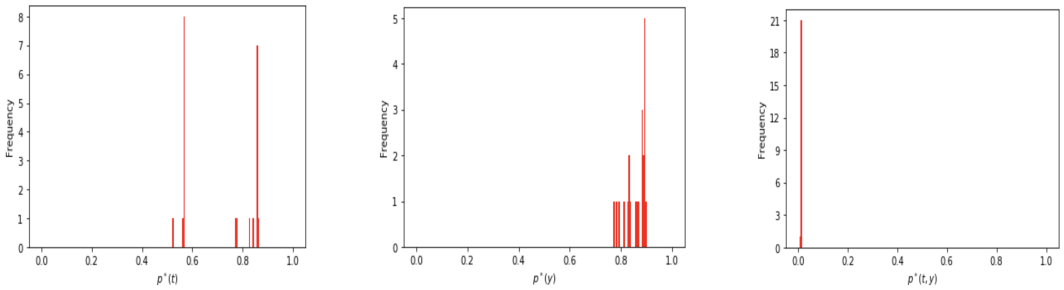
Activity ID	Activity event	
1	2010-08-18 21:47:42	Weekday_Logon_After
2	2010-08-18 22:59:20	Connect_After
3	2010-08-19 01:34:19	WebsiteUpload
4	2010-08-19 01:34:19	File_jpg
5	2010-08-19 01:37:20	WebsiteUpload
6	2010-08-19 01:37:20	File_jpg
7	2010-08-19 01:38:10	WebsiteUpload
8	2010-08-19 01:38:10	File_txt
9	2010-08-19 01:46:04	WebsiteUpload
10	2010-08-19 01:46:04	File_zip
11	2010-08-19 05:23:05	Disconnect
12	2010-08-19 06:10:59	Logoff
13	2010-08-24 01:02:58	Weekday_Logon_After
14	2010-08-24 03:24:16	Connect_After
15	2010-08-24 03:48:51	WebsiteUpload
16	2010-08-24 03:48:51	File_jpg
17	2010-08-24 03:43:48	WebsiteUpload
18	2010-08-24 03:43:48	File_doc
19	2010-08-24 03:34:21	WebsiteUpload
20	2010-08-24 03:34:21	File_txt
21	2010-08-24 04:15:32	Disconnect
22	2010-08-24 04:20:39	Logoff

Table 7.5 shows the performance of insider threat detection provided by all models for *ACM2278*. We report the recall value calculated in the top 3% percent of activities with

Table 7.5: Malicious insider activities detected by different models for *ACM2278* in top 3% percent of activities with the lowest combined likelihood scores

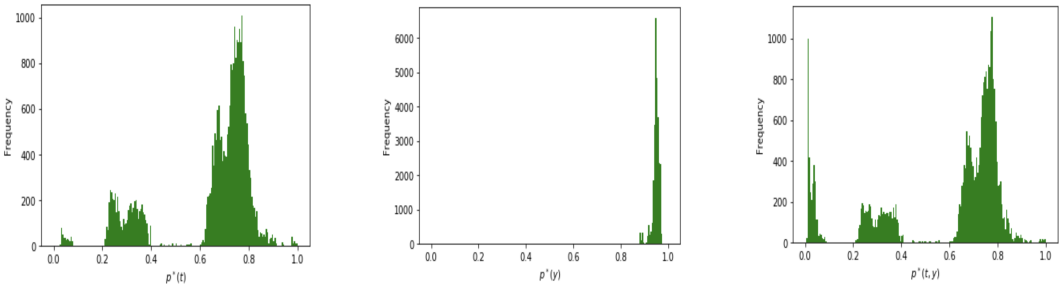
	# of activities	Detected activity IDs
Time model	0	-
Type model	19	1-6, 8, 10-17,19-22
Time&type model	22	1-22
Isolation Forest	1	12
Local Outlier Factor	7	2, 11, 12, 14, 17, 21, 22
ADeMS	22	1-22

the lowest combined likelihood scores from each model. In accordance with Table 7.5, we can see that DMHP *time&type* model and ADeMS both perform well and they detect all 22 malicious insider activities in the top 3% percent of alerted activities; *type* model performs better than the other two baselines with recall $19/22 = 86.36\%$ while *time* and Isolation Forest perform poor.



(a) Time (b) Type (c) Time & Type

Figure 7.2: Histogram of combined likelihood scores for malicious activities of *ACM2278*



(a) Time (b) Type (c) Time & Type

Figure 7.3: Histogram of combined likelihood scores for normal activities of *ACM2278*

Table 6.2 shows the 18 fine-grained categories in our evaluation and each category is

one activity type. Figure 7.2 shows the histogram of combined likelihood scores for malicious activities of *ACM2278*: (7.4.3), (7.4.3) and (7.4.3) refer to the corresponding histogram of DMHP with *time*, *type* and *time&type* as input, respectively. Figure 7.4.3 shows that the combined likelihood scores of malicious activities mainly lie in two intervals $[0.5, 0.6]$ and $[0.7, 0.9]$ from *time*. Similarly, we show in Figure 7.3 the histogram of combine likelihood scores for normal activities, in which (7.4.3), (7.4.3) and (7.4.3) indicate the corresponding histogram with *time*, *type* and *time* & *type* as input.

In Figure 7.4.3, we can see the empirical distribution of normal activities roughly has two peaks with 0.3 and 0.7 as center. In this case, combine likelihood scores of malicious and normal activities are interleaved together and it is hard to find a judge threshold to separate them clearly, which results in a low AUC for *time* model shown in Figure 7.1. Figure 7.4.3 shows that the combine likelihood scores of insider-threat malicious activities for *type* ranges between 0.7 and 0.9, while the ones of normal activities mainly spread from 0.9 to 1.0 in accordance with Figure 7.4.3. Exactly, we can see that *type* model can generally separate malicious and normal activities, which results in a good recall for *ACM2278* shown in Table 7.3. From Figures 7.4.3 and 7.4.3, we can clearly observe that the combined likelihood scores of malicious activities from *time&type* model concentrates on the extreme left part of the empirical distribution of normal activities. As a result, *time&type* model produces a high AUC value as shown in Figures 7.1. In addition, we can see ADeMS also performs well with 22 activities included in top 3% percent.

7.4.4 Wikipedia Vandal Detection

To further evaluate the performance of our DMHP model, we conduct a different task, i.e., detecting vandalism on Wikipedia on the UMDWikipedia dataset [11]. In Wikipedia,

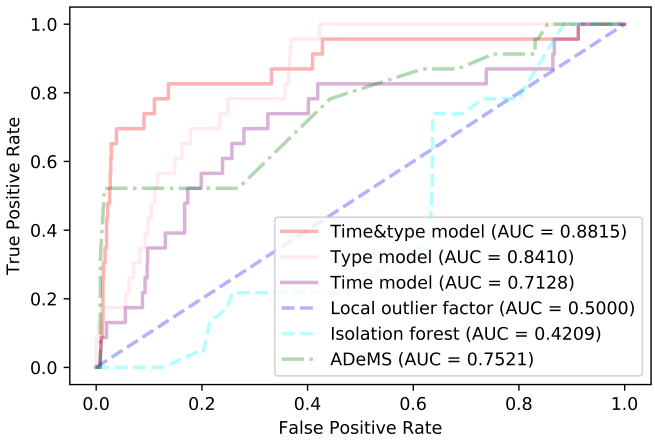


Figure 7.4: ROC curve of malicious edit detection

Table 7.6: The performance of malicious activity and hidden user detection with various λ_0 values on page "List of metro systems", in which AMI refers to 'Adjusted Mutual Information' and NMI denotes 'Normalized Mutual Information'

λ_0	# of predicted modes	AUC	AMI	V-measure	Homogeneity	NMI
0.00001	10	0.8440	0.4446	0.6287	1.0000	0.6771
0.001	16	0.8464	0.3146	0.5001	1.0000	0.5774
0.01	16	0.8464	0.3146	0.5001	1.0000	0.5774
0.1	20	0.8815	0.2352	0.4075	1.0000	0.5058
0.2	22	0.8031	0.2163	0.3840	1.0000	0.4875
0.3	32	0.6719	0.1464	0.2887	1.0000	0.4107

each article page is edited by users including certified editors and volunteers. Each edit contains various attributes such as page ID, title, time, categories, page type, revert status, and content. Because Wikipedia is based on crowdsourcing mechanism and applies the freedom-to-edit model (i.e., any user can edit any article), it leads to a rapid growth of vandalism (deliberate attempts to damage or compromise integrity). Those vandals who commit acts of vandalism can be considered as insiders in the community of Wikipedia contributors.

Note that user edits in Wikipedia are in principle similar to user activities in insider threat detection that are recorded in log files.

In this experiment, we focus on three pages, "List of metro systems", "Star Trek Into Darkness", and "St. Louis Cardinals". For each page, we collect all edits from all of

its contributors (including vandals). Each page has more than 300 edits and at least three malicious edits. Specifically, if an edit is reverted by bots or certified editors, we consider it malicious. We define two types of activities based on whether the edit is on a content page or a meta page. The reason we treat the edits of each page (rather than of a contributor) as a sequence is to have the ground truth of user modes. In this setting, the number of user modes is the same as the number of contributors and the edits from the same contributor can be considered as activities under a user mode in DMHP. To adapt the input of baselines, we embed each activity by a 26-D vector with 24-D for time and 2-D for activity types.

Experiment Results. Figure 7.4 shows the ROC curve of malicious edit detection. We can observe that our DMHP *time&type* significantly outperforms baselines in terms of AUC on malicious edit detection. Similar to results on the CERT dataset, with combining activity types and time information, *time&type* achieves higher AUC value than the model using activity type or time alone. We also observe that, in constrast with CERT dataset, baselines (especially Isolation Forest) achieve poor performances on Wikipedia dataset. This is because Isolation Forest is based on attribute-wise analysis so that it tends to fail when the distribution for anomalous points becomes less discriminative, e.g., if the anomalous and non-anomalous points share similar attribute range or distribution [96]. Compared with CRET dataset, activity type information in Wikipedia dataset is subtle with only choices of *meta page* or not, which, to some extent, is not sufficient to discriminate whether or not an activity is malicious.

We further select the page "List of metro systems" for our case study. The edit sequence consists of 396 edits from 6 contributors, among which there are 20 malicious edits. Table 7.6 shows the performance of malicious edit detection with various λ_0 values. Our DMHP *time&type* achieves good performance on detecting malicious edits on the page with

AUC=0.8815 when $\lambda_0 = 0.1$. A further investigation shows that our *time&type* model can detect 12 out of 20 malicious edits in 10% of total edits with the lowest combined likelihood scores.

Sensitivity of Hyperparameters We focus on the question whether the detection performance of our DMHP is sensitive to hyperparameters. From Eq. 7.10, we can see that the base intensity λ_0 mainly determines whether a new *mode* is created or not for an upcoming activity. In other words, λ_0 affects the number of *modes* which may further affect the detection performance. We can observe from columns 1-2 in Table 7.6 that the number of *modes* increases with the increment of λ_0. This is because, as shown in Eq. 7.10, the bigger λ_0 is, the higher probability of creating a new *cluster* will be. Concretely, the empirical quantity relation between the number of *modes* and λ_0 follows the rule: as λ_0 approaches to 0, the number of *modes* goes to 1; it goes to the total number of activities in the stream as λ_0 approaches to $+\infty$ [97].

To disclose the relationship between the number of *modes* and the detection accuracy, we report the changes of AUC values with variation of λ_0 in column 3 of Table 7.6. We can see when λ_0 ranges from 0.00001 to 0.1, the AUC values do not change much although the number of predicted hidden users (*modes*) increases accordingly. This is because, as λ_0 is 0.00001, the number of predicted users is 10 that is larger than the ground-truth number of real contributors. In other words, the model tries to distinguish different editing styles in a fine-grained manner even if these edits are actually from the same user. As a result, the AUC values slightly increase from 0.8440 to 0.8815. However, if the λ_0 keeps increasing, we can observe a significant reduction of AUC values. This is because the number of the predicted user is too large compared with ground-truth, which is out of a reasonable range. In this case, the model has already been overfitted, so it is very important to choose an appropriate

value for λ_0 which is sensitive.

To study why the number of predicted hidden users (modes) does not affect much the detection performance when λ_0 ranges from 0.00001 to 0.1, we further adopt four clustering metrics, Adjusted Mutual Information, V-measure, Homogeneity, and Normalized Mutual Information, to evaluate the performance of our *time&type* model on the cluster result. Note that we have the ground truth about which contributor (*mode*) each edit is from for UMDWikipedia data. From Table 7.6, we can observe that with the increasing of the predicted number of *modes*, the values of all clustering metrics except *Homogeneity* decrease. However, the Homogeneity scores are always 1 with different λ_0 values, which indicates that each mode contains edits from one single user. This explains why the detection performance in terms of AUC is insensitive to the parameter value of the base intensity λ_0 in the range of $(0.00001, 0.1)$.

7.5 Summary

In this chapter, we have developed the Dirichlet Marked Hawkes Process (DMHP) framework for insider threat detection. Our experiments on two datasets show that DMHP can assign low likelihoods to malicious activities and achieve a good performance in terms of AUC.

8 Conclusions and Future Work

In this chapter, we summarize our works and, based on the observed results and performance analysis, propose several potential research directions on dynamic fraud detection.

8.1 Conclusions

Around dynamic fraud detection, in this dissertation, the works are delivered by the clues: (1) how to build a machine learning model with only positive samples, (2) how to avail few negative samples to improve the discrimination capability of the trained model, (3) how to utilize the late-response labels for fraud early detection and (4) how to develop a multi-scale fraud detection method to capture the subtle and time-evolving misbehavior. Aiming to solve the above challenges, we have done the works as follows.

In Chapter 3, our proposed method OCAN works for one-class fraud detection as only benign users are available during the training phase. During training, OCAN adopts LSTM-Autoencoder to learn benign user representations, and then uses the benign user representations to train a complementary GAN model. The generator of complementary GAN can generate complementary benign user representations that are in the low-density regions of real benign user representations, while the discriminator is trained to distinguish the real and complementary benign users. After training, the discriminator is able to detect malicious users which are outside the regions of benign users.

In Chapter 4, we propose a novel framework combining the idea of self-supervised pre-training and metric-based few-shot learning to detect insiders. First, our framework is pre-trained on a large amount of user activity data to obtain prior knowledge about user

behaviors. Then, it is fine-tuned to derive a similarity function which is to give a higher similarity score between the oncoming insider and the observed insiders.

In Chapter 5, we propose a survival analysis based fraud early detection model, SAFE, which maps dynamic user activities to survival probabilities that are guaranteed to be monotonically decreasing along time. SAFE adopts recurrent neural network (RNN) to handle user activity sequences and directly outputs hazard values at each timestamp, and then, survival probability derived from hazard values is deployed to achieve consistent predictions. Because we only observe the user suspended time instead of the fraudulent activity time in the training data, we revise the loss function of the regular survival model to achieve fraud early detection.

In Chapter 6, we propose a hierarchical neural temporal point process model by combining the temporal point processes and recurrent neural networks for insider threat detection. Our model is capable of capturing a general nonlinear dependency over the history of all activities by the two-level structure that effectively models activity times, activity types, session durations, and session intervals information.

In Chapter 7, we present a Dirichlet Marked Hawkes Process (DMHP) to detect malicious activities from insiders in real-time. DMHP combines the Dirichlet process and marked Hawkes processes to model the sequence of user activities. Dirichlet process is capable of detecting unbounded user modes (patterns) of an infinite user activities, while for each detected user mode, one set of marked Hawkes processes is adopted to model user activities from time and activity type information so that different user modes are modeled by different sets of marked Hawkes processes. To achieve real-time malicious insider activity detection, the likelihood of occurrence of the most recent activity calculated from the DMHP is adopted as a score to measure the maliciousness of the activity. Since the majority of user activities

are benign, those activities with low likelihoods are labeled as malicious activities.

8.2 Future Work

Based on the works which have been done, we propose multiple potential future works with a focus on the following aspects.

Unavailability of labeled data. In this direction, we have conducted two works: a one-class adversarial net for only positive samples and a few-shot learning model for the extremely imbalanced dataset with few negative samples. We are thinking and attempting to: (1) generate more high-quality negative samples to improve the discrimination capability of the few-shot learning model in the fine-tuning phase; (2) with few negative samples, introduce few-shot learning into one-class adversarial net to aid the latter to describe a more clear decision boundary. Recently, some related studies have occurred [98, 99, 100].

Lack of high-quality labels. To address this issue, we have developed a neural survival analysis model for fraud early detection with the late-response labels. However, it is still limited in one aspect: pivoting on the platform's suspicion actions we observed, SAFE can take the early detection in general but it still can not quantify the extent to which, in training phase, the records should be detected in advance. Therefore, it could be a potential research point to accurately formulate how many timestamps before the event point should be considered in the training.

Multi-scale detection frameworks. Hierarchical RNN takes advantage of Hawkes process to capture the time information. One limitation is that as a self-exciting model, it can not fit the scenario: the most recent event decreases the probability of event occurrence rather than increasing. Therefore, it will be an interesting research point how to combine self-exciting process and self-correcting process [54] to comprehensively describe the time-

evolving behavior for fraud detection.

DMHP framework can seamlessly integrate more information about user activities. For example, we can incorporate topic modeling in DMHP to model the content information such as the emails or surfed Web pages. We plan to extend our framework to capture those content and contextual information in our future work. We also emphasize that our introduced *user mode* in DMHP can help accurately model a user's activity sequence because in practice one user often has multiple roles and different working patterns. In our future work, we will conduct more studies to learn how various parameter settings in DMHP (e.g., base intensity, concentration parameters, reference time) affect the number of *modes* and the detection accuracy. In our DMHP models, we introduce the use of a user specified threshold to judge whether the most recent activity is malicious or not. In fact, it is difficult to specify the appropriate threshold value since the threshold may be different for different users and should also dynamically change along the time. We plan to study how to adjust the threshold values dynamically based on the capacity of examining the alerted fraudulent activities each day.

Efficient Implementation. We have not taken much discussion on scalability for the proposed frameworks, and one of main reasons is due to the scale of input data. Instead of large-scale data, the evaluation datasets in this dissertation are only collections in a specific short period of time. Efficient implementation definitely is a key criteria for dynamic fraud detection. Therefore, it would be an important research direction how to improve the efficiency of the proposed frameworks in a real productive environment. Concretely, if RNN is leveraged to adapt the input of sequential data, we can consider a sliding window mechanism and just take into account the most recent records. Also, as for DMHP, we could explore different posterior approximations for a higher implementation efficiency.

Bibliography

[1] D. Chaffey, "Global social media research summary 2020," https://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research, 2020, [Online; accessed 2020-04-17].

[2] Bitdefender, "Stakker, nude photos and beheadings:top facebook scams and malware attacks in 2014," https://hotforsecurity.bitdefender.com/blog/stalkers-nude-photos-and-beheadings-top-facebook-scams-and-malware-attacks-in-\\2014-11080.html, 2014, [Online; accessed 2020-04-15].

[3] L. Coles-Kemp and M. Theoharidou, *Insider Threat and Information Security Management*, 07 2010, vol. 49, pp. 45–71.

[4] CSO, C. D. of SRI-CMU, and ForcePoint, "2018 u.s. state of cybercrime," Tech. Rep., 2018.

[5] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *DMKD*, vol. 29, no. 3, pp. 626–688, 2015.

[6] M. Jiang, P. Cui, A. Beutel, C. Faloutsos, and S. Yang, "Catchsync: Catching synchronized behavior in large directed graphs," in *KDD*, 2014.

[7] Q. Cao, X. Yang, J. Yu, and C. Palow, "Uncovering large groups of active malicious accounts in online social networks," in *CCS*, 2014.

[8] X. Ying, X. Wu, and D. Barbará, "Spectrum based fraud detection in social networks," in *ICDE*, 2011, pp. 912–923.

[9] S. Kumar and N. Shah, "False information on web and social media: A survey," *arXiv:1804.08559 [cs]*, 2018. [Online]. Available: http://arxiv.org/abs/1804.08559

[10] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *CEAS*, 2010.

[11] S. Kumar, F. Spezzano, and V. Subrahmanian, "Vews: A wikipedia vandal early warning system," in *KDD*, 2015, pp. 607–616.

[12] S. Yuan, P. Zheng, X. Wu, and Y. Xiang, "Wikipedia vandal early detection: from user behavior to user embedding," in *ECML/PKDD*, 2017.

[13] E. A. Manzoor, S. Momeni, V. N. Venkatakrishnan, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *KDD*, 2016.

[14] L. Wu, X. Wu, A. Lu, and Z.-H. Zhou, "A spectral approach to detecting subtle anomalies in graphs," *Journal of Intelligent Information Systems*, vol. 41, 10 2013.

[15] S. Yuan, X. Wu, J. Li, and A. Lu, "Spectrum-based deep neural networks for fraud detection," 2017.

[16] L. Wu, J. Li, X. Hu, and H. Liu, "Gleaning wisdom from the past: Early detection of emerging rumors in social media," in *SDM*, 2017, pp. 99–107. [Online]. Available: https://asu.pure.elsevier.com/en/publications/gleaning-wisdom-from-the-past-early-detection-of-emerging-rumors-

[17] Z. Zhao, P. Resnick, and Q. Mei, "Enquiring minds: Early detection of rumors in social media from enquiry posts," in *WWW*, 2015, pp. 1395–1405. [Online]. Available: https://doi.org/10.1145/2736277.2741637

[18] L. Liu, O. D. Vel, Q. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 2, pp. 1397–1417, 2018.

[19] A. Sanzgiri and D. Dasgupta, "Classification of insider threat detection techniques," in *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, 2016.

[20] T. Rashid, I. Agrafiotis, and J. R. Nurse, "A new take on detecting insider threats: Exploring the use of hidden markov models," in *MIST*, 2016.

[21] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," in *AI for Cyber Security Workshop*, 2017.

[22] D. C. Le and A. N. Zincir-Heywood, "Evaluating insider threat detection workflow using supervised and unsupervised learning," in *SPW*, 2018.

[23] S. S. Khan and M. G. Madden, "One-class classification: taxonomy of study and review of techniques," *The Knowledge Engineering Review*, vol. 29, no. 3, pp. 345–374, 2014.

[24] M. A. F. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.

[25] D. M. J. Tax and R. P. W. Duin, "Support vector data description," *Machine Learning*, vol. 54, no. 1, pp. 45–66, 2004.

[26] L. M. Manevitz and M. Yousef, "One-class svms for document classification," *JMLR*, vol. 2, no. Dec, pp. 139–154, 2001.

[27] D. M. J. Tax and R. P. W. Duin, "Uniform object generation for optimizing one-class classifiers," *JMLR*, vol. 2, no. Dec, pp. 155–173, 2001.

[28] M. Kemmler, E. Rodner, E.-S. Wacker, and J. Denzler, "One-class classification with gaussian processes," *Pattern Recognition*, vol. 46, no. 12, pp. 3507–3518, 2013.

[29] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[30] Y. Wang, Q. Yao, J. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," 2019.

[31] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, 2015.

[32] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," in *Advances in neural information processing systems*, 2016.

[33] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *CVPR*, 2017.

[34] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*, 2016, pp. 1842–1850.

[35] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.

[36] P. Wang, Y. Li, and C. K. Reddy, "Machine learning for survival analysis: A survey," 2017. [Online]. Available: https://arxiv.org/abs/1708.04649

[37] B. Liu, Y. Li, Z. Sun, S. Ghosh, and K. Ng, "Early prediction of diabetes complications from electronic health records: A multi-task survival analysis approach," in *AAAI*, 2018.

[38] R. Ranganath, A. Perotte, N. Elhadad, and D. Blei, "Deep survival analysis," in *2016 Machine Learning and Healthcare Conference*, 2016. [Online]. Available: http://arxiv.org/abs/1608.02158

[39] C.-N. Yu, R. Greiner, H.-C. Lin, and V. Baracos, "Learning patient-specific cancer survival distributions as a sequence of dependent regressors," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 1845–1853.

[40] S. Ameri, M. J. Fard, R. B. Chinnam, and C. K. Reddy, "Survival analysis based framework for early prediction of student dropouts," in *CIKM*, 2016, pp. 903–912.

[41] H. Jing and A. J. Smola, "Neural survival recommender," in *CIKM*, 2017, pp. 515–524. [Online]. Available: http://doi.acm.org/10.1145/3018661.3018719

[42] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent marked temporal point processes: Embedding event history to vector," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[43] N. Barbieri, F. Silvestri, and M. Lalmas, "Improving post-click user engagement on native ads via survival analysis," in *WWW*, 2016, pp. 761–770. [Online]. Available: https://doi.org/10.1145/2872427.2883092

[44] G. Yang, Y. Cai, and C. K. Reddy, "Spatio-temporal check-in time prediction with recurrent neural network based survival analysis," in *IJCAI*, 2018.

[45] D. R. Cox, "Regression models and life-tables," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 34, no. 2, pp. 187–220, 1972. [Online]. Available: https://www.jstor.org/stable/2985181

[46] A. M. Alaa and M. van der Schaar, "Deep multi-task gaussian processes for survival analysis with competing risks," in *NIPS*, 2017.

[47] E. Martinsson, "Wtte-rnn: Weibull time to event recurrent neural network," Master Thesis, University of Gothenburg, Sweden, 2016.

[48] M. Luck, T. Sylvain, H. Cardinal, A. Lodi, and Y. Bengio, "Deep learning for patient-specific kidney graft survival analysis," *arXiv:1705.10245 [cs, stat]*, 2017. [Online]. Available: http://arxiv.org/abs/1705.10245

[49] J. L. Katzman, U. Shaham, A. Cloninger, J. Bates, T. Jiang, and Y. Kluger, "Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network," *BMC Medical Research Methodology*, vol. 18, no. 1, p. 24, 2018. [Online]. Available: https://doi.org/10.1186/s12874-018-0482-1

[50] C. Lee, W. R. Zame, J. Yoon, and M. van der Schaar, "Deephit: A deep learning approach to survival analysis with competing risks - semantic scholar," in *AAAI*, 2018.

[51] P. Chapfuwa, C. Tao, C. Li, C. Page, B. Goldstein, L. Carin, and R. Henao, "Adversarial time-to-event modeling," in *ICML*, 2018. [Online]. Available: http://arxiv.org/abs/1804.03184

[52] E. Biganzoli, P. Boracchi, L. Mariani, and E. Marubini, "Feed forward neural networks for the analysis of censored survival data: a partial logistic regression approach." *Statistics in medicine*, vol. 17 10, pp. 1169–86, 1998.

[53] G. L. Grob, Â. Cardoso, C. H. B. Liu, D. A. Little, and B. P. Chamberlain, "A recurrent neural network survival model: Predicting web user return time," *CoRR*, vol. abs/1807.04098, 2018. [Online]. Available: http://arxiv.org/abs/1807.04098

[54] J. G. Rasmussen, "Lecture notes: Temporal point processes and the conditional intensity function," *arXiv:1806.00221 [stat]*, 2018.

[55] Q. Zhao, M. A. Erdogdu, H. Y. He, A. Rajaraman, and J. Leskovec, "Seismic: A self-exciting point process model for predicting tweet popularity," in *KDD*, 2015.

[56] A. Reinhart, "A review of self-exciting spatio-temporal point processes and their applications," *arXiv:1708.02647 [stat]*, 2017.

[57] M. Farajtabar, "Point process modeling and optimization of social networks," Ph.D. dissertation, 2018.

[58] H. Mei and J. Eisner, "The neural hawkes process: A neurally self-modulating multivariate point process," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[59] S. Xiao, M. Farajtabar, X. Ye, J. Yan, L. Song, and H. Zha, "Wasserstein learning of deep generative point process models," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[60] H. Zha, J. Yan, X. Liu, L. Shi, and C. Li, "Improving maximum likelihood estimation of temporal point process via discriminative and adversarial learning," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018.

[61] S. Li, S. Xiao, S. Zhu, N. Du, Y. Xie, and L. Song, "Learning temporal point processes via reinforcement learning," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, 2018.

[62] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," in *IEEE Security and Privacy Workshops*, 2013.

[63] J. Cheng, M. Bernstein, C. Danescu-Niculescu-Mizil, and J. Leskovec, "Anyone can become a troll: Causes of trolling behavior in online discussions," in *CSCW*, 2017, pp. 1217–1230.

[64] S. Kumar, J. Cheng, J. Leskovec, and V. S. Subrahmanian, "An army of me: Sockpuppets in online discussion communities," in *WWW*, 2017, pp. 857–866.

[65] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," in *NIPS*, 2014.

[66] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv:1511.06434 [cs]*, 2015.

[67] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using lstms," in *ICML*, 2015.

[68] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. Salakhutdinov, "Good semi-supervised learning that requires a bad gan," in *NIPS*, 2017.

[69] P. Zheng, S. Yuan, X. Wu, J. Li, and A. Lu, "One-class adversarial nets for fraud detection," 2018.

[70] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv:1406.1078 [cs, stat]*, 2014.

[71] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *arXiv:1606.03498 [cs]*, 2016.

[72] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," *arXiv:1609.03126 [cs, stat]*, 2016.

[73] L. Schoneveld, "Semi-supervised learning with generative adversarial networks," Ph.D. dissertation, 2017.

[74] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[75] D. Costa, M. Albrethsen, M. Collins, S. Perl, G. Silowash, and D. Spooner, "An insider threat indicator ontology," in *CMU*, 2016.

[76] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv:1810.04805 [cs]*, 2018.

[77] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," in *NIPS*, 2017.

[78] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[79] T. Mikolov, G. Corrado, K. Chen, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013.

[80] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.

[81] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," in *SPW*, 2013.

[82] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *ICML*, 2015.

[83] J. P. Klein and M. L. Moeschberger, *Survival analysis: techniques for censored and truncated data*. Springer Science & Business Media, 2006.

[84] P. Zheng, S. Yuan, and X. Wu, "SAFE: A neural survival analysis model for fraud early detection," *CoRR*, vol. abs/1809.04683, 2018. [Online]. Available: http://arxiv.org/abs/1809.04683

[85] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[86] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[87] S. Yuan, P. Zheng, X. Wu, and Q. Li, "Insider threat detection via hierarchical neural temporal point processes," 2019.

[88] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002.

[89] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008.

[90] H. Eldardiry, E. Bart, J. Liu, J. Hanley, B. Price, and O. Brdiczka, "Multi-domain information fusion for insider threat detection," in *2013 IEEE Security and Privacy Workshops*. IEEE, 2013, pp. 45–51.

[91] M. B. Salem, S. Hershkop, and S. J. Stolfo, "A survey of insider attack detection research," in *Insider Attack and Cyber Security: Beyond the Hacker*, 2008.

[92] T. E. Senator, H. G. Goldberg, A. Memory, and et al., "Detecting insider threats in a real corporate database of computer usage activity," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013.

[93] N. Du, M. Farajtabar, A. Ahmed, A. J. Smola, and L. Song, "Dirichlet-hawkes processes with applications to clustering continuous-time document streams," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 219–228.

[94] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Loop: local outlier probabilities," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1649–1652.

[95] H. Huang and S. Kasiviswanathan, "Streaming anomaly detection using randomized matrix sketching," vol. 9, 01 2016.

[96] H. Huang, H. Qin, S. Yoo, and D. Yu, "A new anomaly detection algorithm based on quantum mechanics," in *2012 IEEE 12th International Conference on Data Mining*, 2012, pp. 900–905.

[97] M. Gormley, "The dirichlet process (dp) and dp mixture models," 2016.

[98] R. ZHANG, T. Che, Z. Ghahramani, Y. Bengio, and Y. Song, "Metagan: An adversarial approach to few-shot learning," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 2365–2374. [Online]. Available: http://papers.nips.cc/paper/7504-metagan-an-adversarial-approach-to-few-shot-learning.pdf

[99] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," 2018.

[100] E. Zakharov, A. Shysheya, E. Burkov, and V. Lempitsky, "Few-shot adversarial learning of realistic neural talking head models," 2019.