FRAUD DETECTION THROUGH DEEP USER REPRESENTATIONS AND SINGLE CLASS LEARNING WITH GENERATIVE ADVERSARIAL NETWORKS

VIMAL JOSE

Thesis Report

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

LSTM……..	Long Short Term Memory
GAN….....	Generative Adversarial Networks
OCC……..……	One-Class Classification

CHAPTER 1

INTRODUCTION

As the reach and role of internet increases in our daily lives, the impact of internet frauds on our lives increases exponentially. With most of our transactions being made online – from transferring money to making friends and from acquiring knowledge to finding a romantic partner – there is no limit for the impact and effect of online frauds. And the same is applicable more so for businesses and establishments as internet frauds often result in monetary and non-monetary losses to businesses including their credibility and goodwill. A few examples for these are – the spread of fake news on social media, dissemination of fake information on knowledge sharing platforms like Wikipedia, fake profiles on social media platforms and fraudulent transactions in financial networks. Even though this has been well researched before, two of the major challenges that hold us back from eradicating them completely are:

1. The percentage of reported or identified malicious activity among the total user behavior is very small. For e.g., the reported % of fraud transactions in total payments in the US for 2016 is 0.0014%. [28][29]
2. Once a pattern of fraud is starting to be detected by the security systems, the attackers malicious agents change their pattern.
3. Waiting long enough to collect the required amount of data for detection often ends up being too late.

To specify the same in machine learning terms, this means the non-availability of enough data for the fraud classes is one of the key reasons why the current systems are unable to detect malicious activity early enough.

Another key point is that in most of the online malicious activities, it has been noticed that most of them contains repetitive factors – be it in terms of users, associations, history and repeated patterns of transactions. So, it can be said that knowing the history and the associations of the users / actors involved might be an important factor in detecting fraud. In other words, fraud detection is more effective if done at a user level than at a transaction level. So, it is important that fraud detection needs to be done in a stateful manner at the actor level. To specify the same in machine learning terminology, and ideal fraud detection algorithm should be a sequence classifier rather than a data point classifier.

Keeping these in mind, the goal this paper is to build a detection algorithm which can:

1. Detect without any data regarding the fraudulent activity beforehand.

2. Identify new patterns of fraud even when we have no previous data to learn from.
3. Make use of the history of an actor when available.

Based on these goals, the proposed approach in this paper is to build a one-class classifier (OCC) for non-fraud users. We'll be making use of an LSTM-autoencoder to convert the actor history, whenever available into user representations in a fixed dimensional feature space. These user representations will be used for generating the user representations of malicious users by a complimentary GAN generator and the GAN discriminator will be used as one of the final classifiers.

For testing these assumptions, we will be working on the Wikipedia edits dataset for identifying Wikipedia Vandalism. So the goal of this research would be to develop a system using LSTM auto encoder and a complimentary GAN to correctly identify wikipedia vandals based on the Wikipedia Edits dataset.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

Fraud detection is a field which has a substantial amount of research, both existing and ongoing. Different approaches including probabilistic, graph based, cluster based and embedding based approaches are already existing. But most of the approaches require at least a small amount of training data for the fraudulent class. Also, there is considerable amount of research available on novelty detection – which is another way to think about the issue.

As mentioned, in an OCC scenario, we only have one class available. So, to correctly learn the intrinsic geometry of the non-malicious class, it is important to represent the data in the best possible way, as the initial step. For this, we can use user representations and there is multiple existing research on this. Recently deep learning-based approaches have found success in this task [3] [4]. But in our case, since ours is sequential data, it is required for us to have an encoder which supports it. Research in [5] uses Long Short Term Memory Networks (M-LSTMS) for this task. However, it requires information about both the fraud and non-fraud classes to perform. For example, Adversarially Learned One-Class Classifier for Novelty Detection [6] uses an adversarial network to train an Encoder-Decoder network for making effective representations of the data. Since ours is a sequence representation, in this paper, we will be proposing an LSTM-autoencoder to generate the user representations.

Another advantage of our framework is that since we are using a complementary GAN generator to generate the fraud user representations, the discriminator will be able to predict the non-fraud users with better confidence. The theoretical decision boundaries for the non-fraud users will be better defined by using this technique.

2.1 Problem definition

Wikipedia is currently the world's largest encyclopaedia and is a source that supplies credible information to a large majority of the academic and non-academic population of the world. It has over 49M pages and 37M registered users. Since it is a completely crowd sourced platform for information dissemination, edits done by the users are key to the growth and maintenance of the platform. Till date there has been over 923M edits on the platform. With such large numbers and a large majority of the population using Wikipedia as a credible source of information, it is very much prone to vandalism through malicious edits which is defined by Wikipedia as "any addition, removal, or change of content, in a deliberate attempt to

compromise the integrity of Wikipedia". Such vandalism affects the credibility of the platform and it might be fatal if used to spread fake facts as well.

2.2 Related Research

The independent research that has been conducted in this space is quite sizeable. Based on the relativity to our approach, the existing literature can be compared based on Intent, Technique and Data requirements.

Fraud detection is a topic which has got considerable amount of research in the recent decade, especially after social networks, online news and reviews became common. And most these researches focus primarily on fake profiles on social media, misinformation in the form of news or fake reviews on websites like Amazon and Yelp.

For social media fake account detection, research in SynchroTrap [1] identified that the actions performed by fake accounts are somewhat synchronized even in different social network contexts. So [1] concentrated on the identification of malicious accounts by clustering these user accounts based on their loosely simultaneous actions over a predefined window of time. In other words, [1] was using user similarity and synchronicity of the actions for identifying malicious accounts and the results are quite commendable. Another similar research for the same intent was CATCHSYNC [2] which focused on anomaly detection by analyzing the connectivity patterns of these profiles using graph analysis. CATCHSYNC identified and utilized two of the key graph based patterns which are characteristic of fraudsters which were synchronized behavior and rare behavior. Synchronized behavior was defined as when suspicious nodes have very similar behavior patterns simultaneously or with a time lag. This is useful as many a times fraudulent users or profiles support each other in the form of upvotes or likes or social media followership, as they have the same goal. Rare behavior was defined as when the behavioral or connectivity patterns of profiles were differing considerably from the majority. CATCHSYNC introduced two new measures, variants of which have been used in subsequent research, called synchronicity and normality. Similar to our research, CATCHSYNC also didn't require any labelled data as it was completely based on unsupervised learning.

Research in [30] provides a comprehensive study across multiple aspects of fake information including the users or profiles which conducts these malicious activities and spreads them, the reasons why they are performing or spreading these activities and measuring the impact and characteristics of these activities across multiple dimensions. This provides an insight into why

and how the problem of fraud needs to be solved and how it is impacting the credibility of our information networks.

On a similar line, research in [31] focuses on developing a spectrum based detection framework to identify a specific malicious network attack style called Random Link Attacks (RLAs). Random Link Attacks are defined as when a malicious user creates an array of profiles or edits and makes them interact so as to improve the trust rating of the profiles by circular upvotes, interactions or votes. Once they have setup trust scores, these profiles will start interacting with or editing the real profiles and documents, so as to go undetected. [31] shows that these users can be identified with an analysis of their spectral coordinate characteristics. [32] also proposes a similar solution based on a combination of deep neural networks and spectral graph analysis for fraud detection on a signed graph with only a small set of labeled training data. It develops and evaluates two neural network architectures including a deep auto encoder and a convolutional neural network for fraud detection.

Research in [29] looks at the problem of fake reviews on sites like Yelp and Amazon. It uses both supervised and unsupervised learning and it uses both behavioral and linguistic features for detecting fake reviews. The key take away for our research from [29] is that it has been proven that behavioral features perform much better when compared to linguistic features. Even though this cannot be applied directly to our research as our problem statement is different, this gives us a direction that it might be better to give importance to behavioral features than textual or linguistic features. A similar research in [34] identifies several specific behaviors of review spammers and uses them to identify the spammers. In [34] they are trying to detect spammers based on their product or product group selection and how different their reviews are, from the average review of the product. In other words, this is similar to [2] even though here we are using topic similarity instead of synchronicity.

While most of these concentrated on utilizing linguistic clues for malicious intent, behavioral footprints and inter-relations between malicious agents, [35] proposed a novel approach called SPEAGLE that takes into account the linguistic data, timeseries data, numeric data such as ratings and relationship data into a single framework. In other words, they utilize the metadata, relational data, linguistics and agent details to identify fake agents or users as well as fake reviews. In addition to that, SPEAGLE was able to achieve comparable performance with other existing models with semi supervised learning approaches.

Based on training data requirements, we can say that there are supervised, semi supervised and unsupervised approaches available as per the current state of research, most of them being

supervised models. Supervised approaches are the ones which are currently the state of the art in the Wikipedia malicious edit detection use case.

2.3 Existing literature on Wikipedia Vandalism detection

Cluebot NG: Cluebot NG[7] was one of the first approaches in solving the issue of vandalism on Wikipedia. It was a simple and modular approach by combining the predictions made by a naïve Bayes classifier and an artificial neural network. While the naïve Bayes classifier had good precision since it was relying heavily on specific words which were almost only seen in vandal edits, it had low recall values due to the over dependence on vandal words. The predictions made by the naïve Bayes model were used as one of the features for the neural network, along with a considerable number of other statistical features. And the output of the ANN was used for as the final prediction. Cluebot NG was mostly reliant on linguistic features and other details of the edit action performed.

STiki: STiki[8, 36] was another tool which was made available to trusted users to detect vandalism, spam, and other types of unconstructive edits on Wikipedia articles. STiki was not an automated model, but a collaborative tool which showed suspected malicious activity to trusted end users. Whenever an edit is marked as vandalism by the user, STiki added that information to the training data for the next iteration. The key advancement STiki made in terms of vandalism detection was in the feature engineering part. It uses features including spatial and temporal properties unlike Cluebot NG. This includes simple features like edit time-of-day, revision comment length etc. and aggregated features which combine time decayed behavioral features (feedback from trusted users) to create reputation values for single-entities (user, article) and spatial groupings thereof (geographical region, content categories). Later, an updated version of STiki called Snuggle was rolled out to detect and manually tag vandal users and edits.

VEWS[10]: VEWS approach was a combination of two earlier approaches, namely Wikipedia Vandal Behavior (WVB) and Wikipedia Transition Probability Matrix (WTPM). Wikipedia Vandal Behavior approach utilized an innovative set of user editing patterns as features to classify, while Wikipedia Transition Probability Matrix approach utilized a combination of features obtained from a transition probability matrix and then reduced it using an auto-encoder built on neural networks to classify users as vandals and benign. VEWS outperformed both Cluebot and STiki with an 85% classification accuracy. VEWS also produced a new data set with about 33K Wikipedia users and 770K edits. Our research will be reusing this dataset for experiments.

WDVD[37]: One of the more accurate approaches to Wikipedia vandalism detection was WDVD. It utilized 27 features about an edit's content and 20 features about an edit's context. The 27 features about content included character level, word level and sentence level features. Context based features included reputation of the user, geographical location, reputation of article and metadata details. The classifier was built using random forests with multiple instance learning. Another key point that WDVD introduced was the concept of sessions – consecutive edits on the same article by the same user was termed as a single session. Since the edits done in a single session are generally inter-related, this concept of sessions significantly improved the performance of the classifier. The concept of edit sessions have been reused in our research for experiments.

All of these approaches are based on supervised learning and relies on the existence of labelled data.

2.4 Existing literature on Single Class Classification

One class classification or single class classification problems is a variation of novelty detection. This means that the data will contain information about only a single class and will be learning to predict whether a new data point belongs to that class or not. These types of classification algorithms are very useful for anomaly detection as the non-availability of anomalous data is one of the key pain points in such problems. In our research, we will be considering the Wikipedia Vandal detection as a one class classification (OCC) problem as that would enable us to solve it without having to rely on labelled data. One of the most popular OCC models are One class Support Vector Machines (OCSVM) [38, 39]. Another common type of OCC is one class nearest neighbor (OCNN). While OCSVM tries to predict the class of the data by modelling a hypersphere around the observed data points, OCNN predicts by calculating the distances between data points. Another popular OCC is one class Gaussian Process (OCGP). However, none of these models are completely unsupervised. They require either the parameters or thresholds to be set based on heuristics or domain knowledge for them to perform effectively.

In our research, we'll be using Generative Adversarial Networks (GANs) to generate data for training the classifier and we will be comparing our results with these classification methods.

2.5 Concepts used

2.5.1 Recurrent Neural Networks

One of the major issues that existed with normal feed forward neural networks is that doesn't have persistence. This means that normal feed forward neural networks treat each input data point as an independent data point. Because of this, neural networks were not useful in any problems which required an understanding of the context. And most of the more complex problems demands a good understanding of the context of the problem and an understanding of the sequence of information at any given point. An example for this is understanding the contents of a movie. If each frame is to be treated as independent frames, one would not be able to understand the story that is being shown in the movie. Another example could be any time series forecasting problem – if each data point is treated as independent, the feed forward neural network won't be able to understand the trends in a time series. This was a major disadvantage of using feed forward neural networks.

This is the issue that Recurrent Neural Networks resolves. Recurrent Neural Networks enable information to be persisted. Recurrent Neural Networks are networks with loops built into their structure, so that the information that passes through them is passed stage by stage, to enable persistence at each stage. A recurrence relation over timesteps is the reason behind their ability to do this. It can be summarised as:

$$S_t = f(S_{t-1} \cdot W_{rec} + X_t \cdot W_x)$$

Where k is the current timestep, St is the state at current time step, X_t is the inputs at current time step and W_{rec} and W_x being parameters of the model. This way, RNN can be mentioned as a state model with a feedback loop. Because of this recurrence relation, the state changes over time and with a delay of one time step, the feedback is transmitted back into the state. Thus, it is able to have a memory between different time steps because of the evolution of the state.

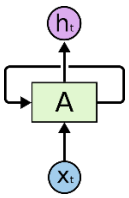


Figure 1[40]

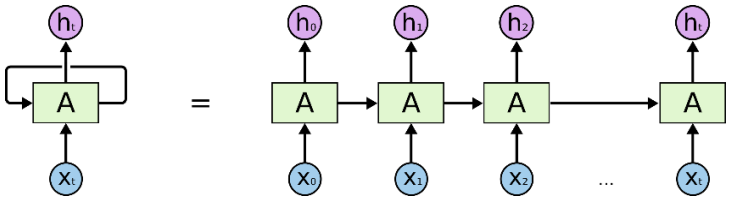


Figure 2[40]

Figure 1 is a simplified representation of a Recurrent Neural Network. It takes an input X_t and outputs h_t and because of the loop, it is able to persist information from one step to the next. To simplify it even further, it can be thought of as a chained repetition of the same network,

each transmitting a message to the its successor. In other words, Recurrent Neural Networks are a sequence of networks which sequentially passes information to its successor which is calculated based on the input at current time step and information from its predecessor.

Recurrent Neural Networks have been conditionally effective in a wide array of problems including language translation, video understanding, image captioning etc. But RNNs in its default form, has got a few disadvantages. With the structure that we discussed, RNNs are able to use past information with current information. But this is limited to the recent past. For example, in a language model, this works in the case of shorter sentences, but it will be problematic when the sentences are long.

Since the number of steps are generally limited in a RNN, the past information becomes forgotten in a few time steps. Because of this, RNNs have an issue with long term dependencies. When there are long term dependencies to be used, either we have to go for very long RNNs, which increases the complexity of the model or we have to go for any other technique.

The issues with RNNs have been explored in details in multiple different studies.

2.5.2 Long Short Term Memory Networks (LSTMs)

Long Short Term Memory Networks are a special case of RNNs designed specifically to solve the issue of long term dependencies. Currently most of the aforementioned use cases are being solved with LSTMs rather than vanilla RNNs.

The way LSTMs solve the long term dependency problem is by remembering information for prolonged periods of time by controlling the decay of information within the different steps of the network. By controlling the parameters, an LSTM will be able to identify both short and long term dependencies.

As we have already discussed, all RNNs comprise of sequentially arranged repeating modules. In vanilla RNNs, the structure of these modules are very simple in general. For e.g. a single tanh layer.

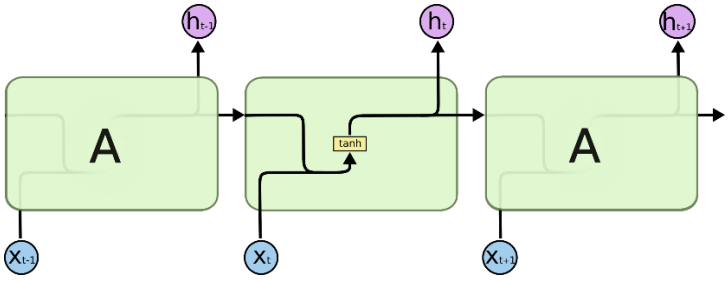


Figure 3 [41]

The repeating module in LSTMs have a much more complex structure. Instead of a single layer, generally it has four neural network layers which interact with each other to ensure information is remembered for longer periods of time.

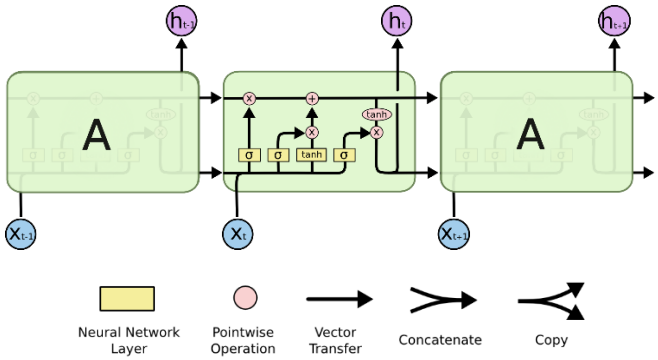


Figure 4 [41]

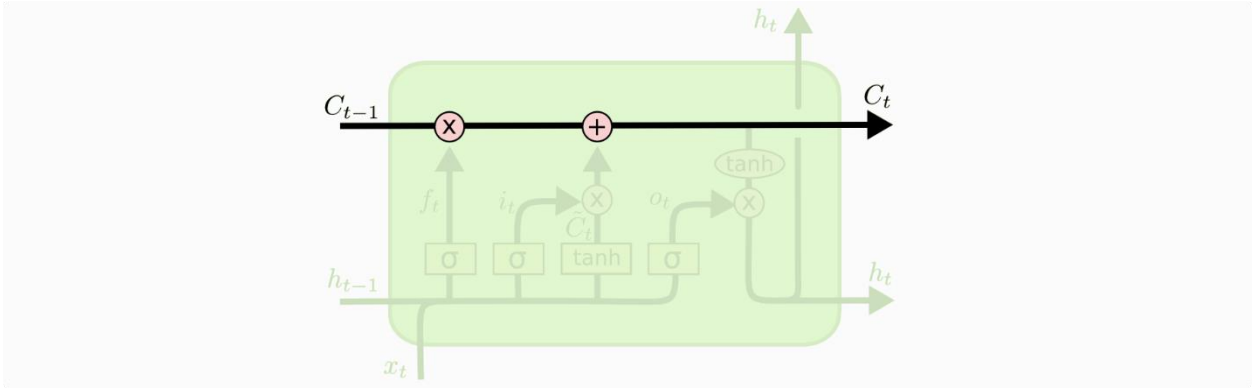


Figure 5 [41]

C_t is the cell state at step t.

x_t is the input at step t.

h_t is the output at step t.

Cell state goes from the first module until the last module through the entire chain. Each module presents two linear interactions to the cell state but it can flow through the entire chain without alterations easily as well, if required. Each module has the ability to optionally alter the cell state using structures called gates.

The first operation to perform in an LSTM is to identify what needs to be removed from the cell state. For this the sigmoid layer in the module checks x_t and h_{t-1} and outputs a number between 0 and 1 for each number in the previous cell state C_{t-1}. This output f_t determines whether to keep or discard the values from the previous cell state, 1 being 'completely keep' and 0 being 'completely discard'. In other words, in the first step the LSTM makes a decision on how much of the previous cell state needs to be removed from the current cell state.

Since we've already decided what to remove from the cell state, the next step is to decide what to add to the cell state. This happens in three parts, first the tanh layer computes a vector of the values needs to be added to the cell state, called the candidate values C_t, and the sigmoid input gate layer decides which of these values will be updated. In the third step, we calculate the product of the outputs from the sigmoid and the tanh layers and add it to the cell state C_{t-1}.
So in short, we multiply the previous cell state C_{t-1} with ft to discard the required parts of the previous cell state and then add the new updates by adding the product of the outputs of the tanh and sigmoid input gate layers.
The final step in LSTM is to decide what to output from the module. The output will be based on cell state, but LSTM performs a few operations before setting it as output. First the cell state passes through a tanh layer to bring the values between -1 and 1. Then, this is multiplied with a sigmoid of the concatenation of current input x_t and previous output h_{t-1}. This is done to make sure that we output only the parts which were intended.
This process repeats in each module at each time step. Thereby LSTMs are able to solve problems which requires an understanding of long term dependencies since they can theoretically remember information for an indefinite amount of time, if required.

2.5.3 Autoencoders

Auto encoders are neural networks trained to reproduce the input data as the output. These are special types of feed forward neural networks where the input is the same as the output. In other words, auto encoders attempt to learn an approximation of the identity function to make sure the output matches the input. It might seem trivial to approximate the identity function, but we make it challenging to the auto encoder by constraining the network, neural layer sizes and number of hidden layers. As the model is forced to keep only some of the original input in the intermediate layer, it is forced to prioritize which aspects of the input should be copied. This often results in the model learning the most useful and important properties of the data.
Auto encoders compress the input into a lower dimension and then attempt to reconstruct the original data from the compressed data. The intermediate representation, or the compressed form of the original data is often referred to as latent space representation. It is also referred to as the 'compression' or a 'summary' of the input or simply as the 'code'.
An autoencoder works in two phases – Encoding and Decoding. During the encoding phase, the original input is compressed into code. Code is the intermediate form of the input with which the decoder tries to recreate the original input. So there are three components to auto encoders, namely the encoder, code and the decoder.

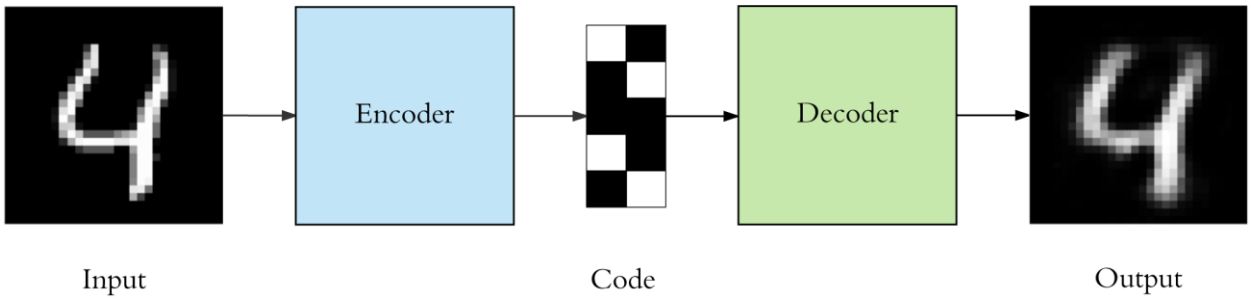


Figure 6 [42]

We required three things for creating an auto encoder: An encoding method, a decoding method and most importantly, a loss function for comparing the input and the output. The loss function will be an indicator for the amount of information loss happening within the auto encoder by comparing the original input fed into the encoder with the recreated output generated by the decoder. This will depend on the efficiency with which the code captures the salient features of the data.

The important properties of autoencoders are:

1. Specific to the data trained on: Since auto encoder tries to represent the information during training in the most efficient way possible within the code, it will be only be able to compress data which is similar to what is has been trained on. This is because auto encoders identify and learn specific features from the data so as to minimize the information loss. The same features might not be relevant for a different type of data.
2. Unsupervised: Auto encoders are primarily about learning data representations in the best way possible. We don't need any training data for training auto encoders as the loss function compares the inputs and outputs and uses it for adjusting the weights and biases. So auto encoders are considered unsupervised as they don't require labelled data for their training.
3. Lossy: Since autoencoders involve a decoding and an encoding phase and since the intermediate representation or the code is generally having lower dimensions than the input, there will be information loss between the input and output.

In terms of architecture, both the encoder and decoder are fully connected feed forward neural networks. The intermediate representation or the code is generally a single layer, whose dimensions will be less than the input and the output layers. The input and the output layers will have the same sizes as they are have to be compared with each other to calculate the loss. Before we start the training of the auto encoder the hyper parameter of code size needs to be set.

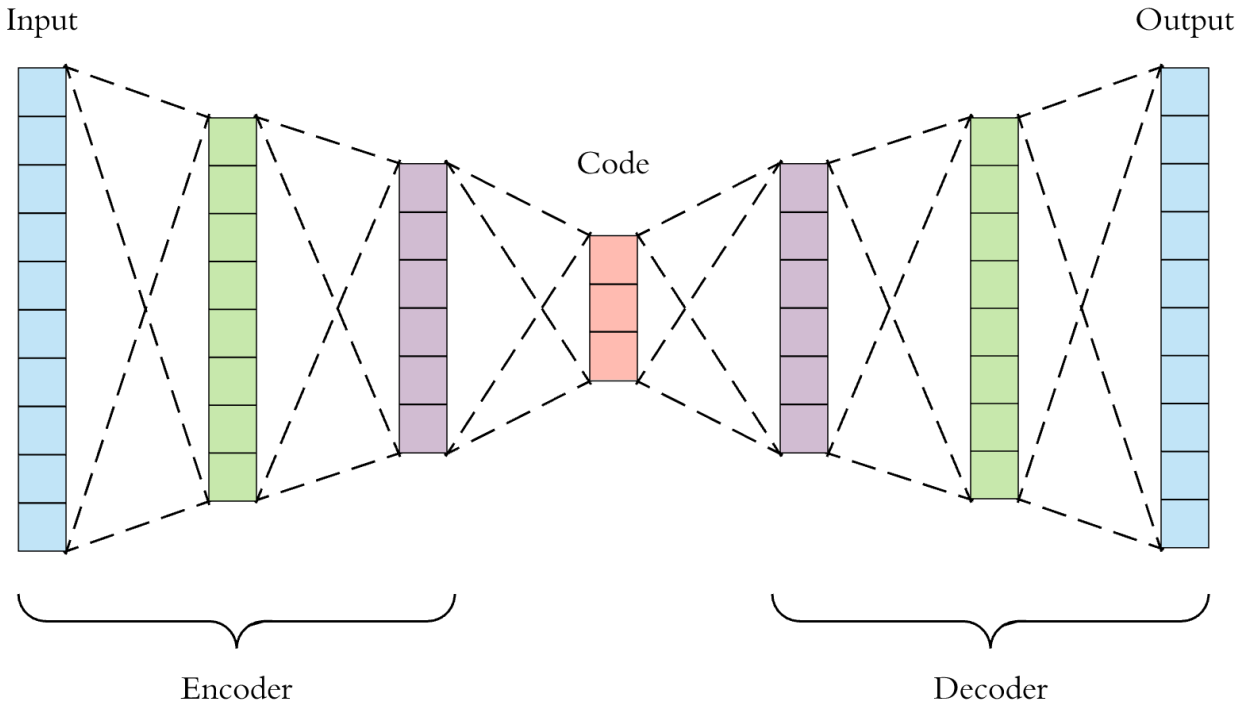


Figure 7 [42]

As seen from the visualization above, the input is first passed through the encoder, which is a full connected neural network. The encoder encodes the input and creates the code, which has a lower dimensionality than the input. The decoder, which is also a fully connected neural network, tries to recreate the input by using only the code. The dimensionality of inputs and outputs are the same as they have to be compared. In an ideal auto encoder, output will exactly be the same as the input. But this doesn't happen in general unless the size of the features to be learned from the input is less than the code size.

Auto encoders have four hyper parameters that needs to be set before training one – Code size, Number of layers, Number of nodes per layer and the loss function. For the loss function, we generally use mean squared error or binary cross entropy. By controlling these hyper parameters we'll be able to control the learning capacity of the auto encoder. If we oversize the autoencoder, it may start copying the input to the output without learning it and if we undersize it, it might not be able to learn efficiently because of the high losses.

2.5.4 LSTM Autoencoders

As we discussed, LSTM are RNNs that are specifically designed to support sequential data as input. LSTMs can learn the complexities of the time series input sequences and use memory to learn long input sequences. LSTMs can also be used to organize an Encoder-Decoder LSTM that enables the model to handle variable length input and output sequences. This Encoder-

Decoder architecture has been used for creating some of the state of the art models in sequence predictions including speech recognition, video understanding and language translation models. In the encoder-decoder LSTM architecture, the encoder LSTM takes in the input sequence and once the sequence is completely read, the output of the model will contain a fixed length representation of the complete input sequence. In other words, this is the learned representation of the complete input sequence. This is then passed to the decoder network which converts it into the output sequence step by step.

An LSTM Autoencoder is nothing but an implementation of an autoencoder specifically for sequential data using the Encoder-Decoder LSTM architecture. An Encoder-Decoder LSTM reads the dataset of input sequences one by one, encodes it and recreates it by decoding it. As in the case of autoencoders the performance of an LSTM autoencoder is also evaluated based on how well the decoder is able to recreate the input sequences. After training, once the autoencoder is able to achieve the desired accuracy in recreating the data, we can utilize the encoder model by discarding the decoder. This means we'll be feeding the input sequences as input to the encoder and will be taking the lower dimensional intermediate representation as the output of the model. This will enable us to perform fixed length encoding of input sequences by extracting the important features of the sequences. This fixed length representation can be used as an input for another learning model or as an efficient, compressed representation of the input sequence. Often times this approach is used for multi-variate time series problems to convert sequences into individual data points which can then be used for making predictions easily.

There are a few differences between regular LSTM networks and LSTM autoencoders. In LSTM autoencoders, all the layers return sequences, which means each layer is outputting a two dimensional array containing each timestep. In contrast, regular LSTMs have single dimensional encoded feature vectors as the result of any of the intermediate layers. The non-existence of this encoding vector is one of the key differences between a regular LSTM network and an LSTM auto encoder.

2.5.5 GAN – Generative Adversarial Networks

Generative adversarial networks or GAN is a neural network architecture in which two neural networks compete against each other by one of them generating synthetic samples of data that can be used to fool the other network and the other one trying to identify the synthetic data

from real data. Because of this competition, these are called adversarial networks. GANs are widely used in a variety of applications ranging from image generation to game playing agents. For a clear understanding of GANs, it is best to have a good understanding of generative and discriminative algorithms. Discriminative algorithms are essentially classification algorithms. That is, based on the features given, a discriminative algorithm will predict the label or class to which it belongs. So discriminative algorithms classify features into labels. They map features into labels and that is the only one task that they perform in a GAN. On the other hand, generative algorithms perform the exact opposite of that. They predict the features when the label is given. An example for discriminative and generative algorithms would be email classification. A discriminative algorithm predicts whether an email is spam or not based on the content of the email. On the other hand, a generative algorithm tries to predict the contents of the email given that the email is a spam. In other words, generative algorithms tries to predict the probability of features given the label or category. Generative algorithms can also be used as classifiers theoretically, but that is not a common use case.

The same information can be formalized as:

- Discriminative models learn the class boundaries.
- Generative models learn the individual class distributions.

In generative adversarial networks, the generator network generates synthetic instances of data while the task for the discriminator is to evaluate these instances for authenticity. The discriminator predicts whether each evaluated data instance belongs to the actual training data or if they are synthetically generated by the generator network. So the goal of the generator is to create the best synthetic data and the goal of the discriminator is to always correctly identify when the data is synthetic. The steps involved in a GAN which identifies MNIST digits would be:

1. The generator inputs any random numbers and outputs an image
2. The image generated by the generator is inputted to the discriminator along with real MNIST dataset images.
3. The discriminator predicts a probability score for each image for the image being authentic and not fake.

Using this technique, you essentially get a double feedback loop between the discriminator, generator and the real data. Both the generator and discriminator networks are trying to optimize opposing objective functions in a zero sum game. When the discriminator changes, the generator also changes accordingly. Their losses act in opposite directions.

CHAPTER 3

ANALYSIS

The data that will be used for this thesis will be the UMDWikipedia dataset, which consists the sequence of edits done by each user. The dataset contains the details of 770k edits from 17k non-vandals and 17k vandals. Some of the sequences are over 6000 steps long and can be considered outliers. These can be removed. For the purposes of this thesis we have limited the number of edits between 4 and 50 edits which comes to 11k vandals and 10k non-vandals. Each edit has got information about the following details: Meta-page or not, Speed of consecutive edits – less than 1 minute or not, Existence of the edit history of the page, whether or not the user's current edit would be reverted etc.

To correctly differentiate the edits made by vandals and benign users, this research utilizes two new datasets which are derived from the UMDWikipedia dataset.

1. **Edit Pair Dataset**: Each row of the edit pair dataset is of the form (u, p1, p2, t, feature set). u is the user id, t is the timestamp of the edit on p2 and (p1, p2) is a pair of pages that are subsequently edited by the user. If the user makes consecutive edits to the same page, p1 and p2 could denote the same page. Each row also contains the following features.
    a. Is p2 a metapage or not?
    b. Category for the time difference between the two edits p1 and p2:
        i. Very Fast Edit: less than 3 minutes
        ii. Fast Edit: Between 3 and 15 minutes.
        iii. Slow Edit: More than 15 minutes.
    c. Is p2 the first edit made by the user?
    d. If p2 has already been edited by the user in the past?
        i. If so, is p2 the same as p1. i.e. same edit on the same page?
        ii. Has there been a reversion for the users edit on p2 in the past?
        iii. The hyper-link distance between p1 and p2. The minimum number of links from p1 to p2 in the hyperlink graph for Wikipedia. These are categorized into:
            1. Equal to or less than 3
            2. More than 3
            3. Unreachable.
        iv. Category count similarity between p1 and p2.

These features basically describe the properties of page p1 with respect to p2.

2. **User Temporal Dataset:** This dataset captures a set of different temporal attributes of the user, including how the user was using Wikipedia for navigation and how fast the user was navigating. This is a chronological sequence of edits made by the user u in a format in which each consecutive combination of pages p1 and p2 corresponds to a row in this dataset. All the features described in the Edit pair dataset will also be available in this dataset as well.

3.1 Exploratory data analysis

Figure 8 shows the number of edits made vs the percentage of users for benign and vandal users. Figure 9 depicts the percent of re-edits vs the percentage of users for benign and vandal users. And Figure 10 shows the same for number of distinct pages edited by users.

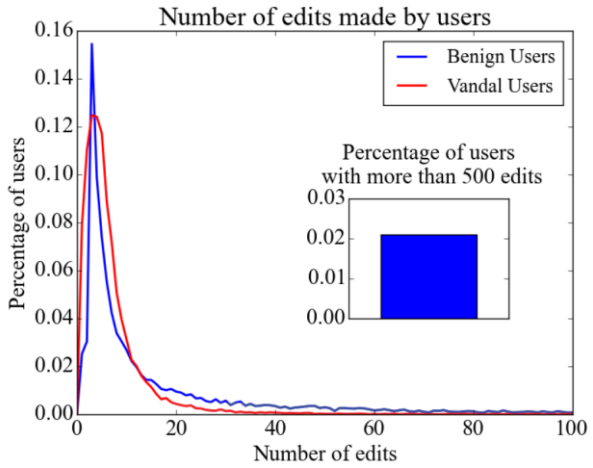


Figure 8

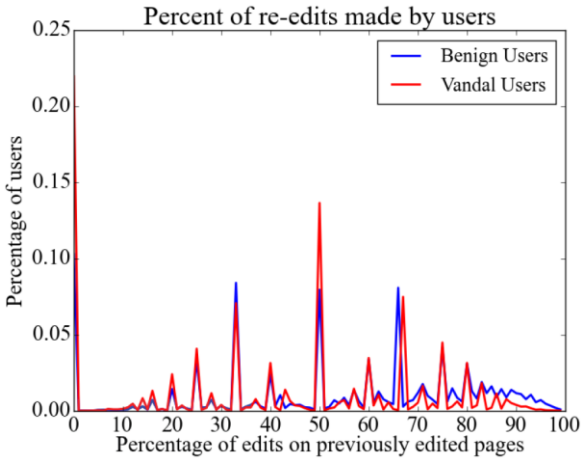


Figure 9

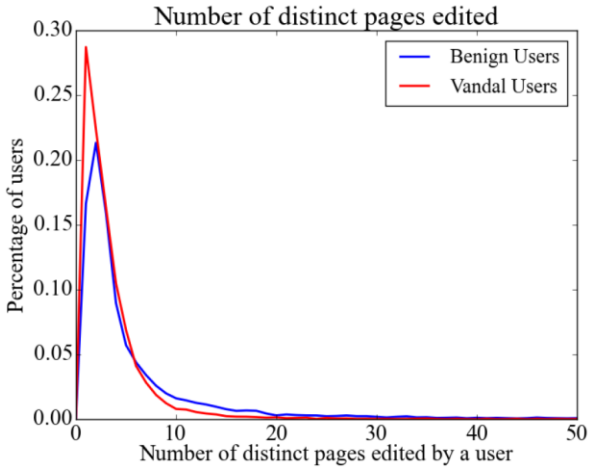


Figure 10

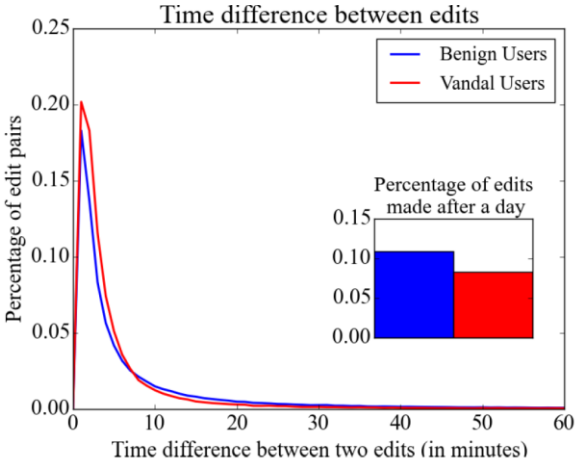


Figure 11

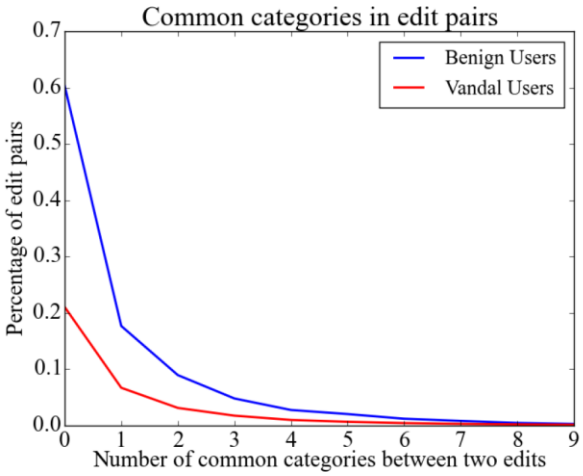


Figure 12

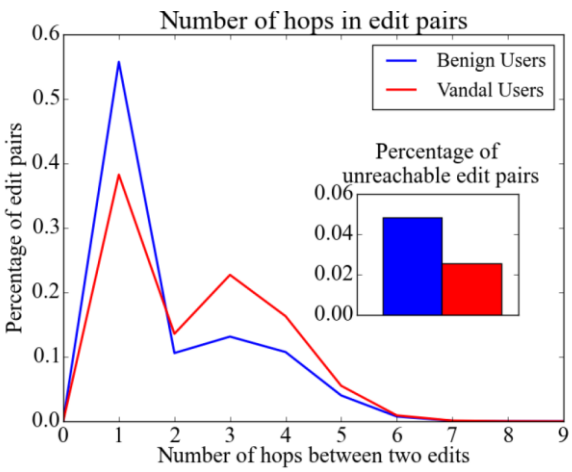
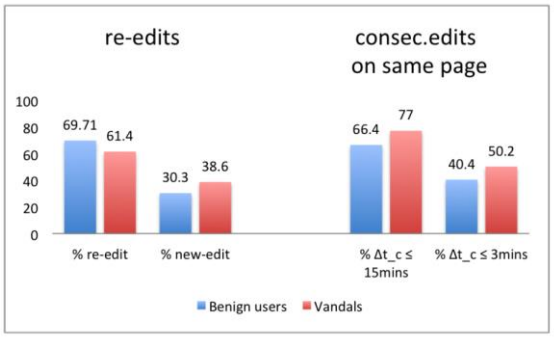


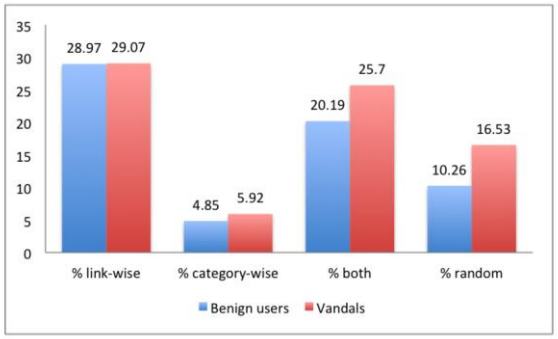
Figure 13

Figures 11 to 13 shows the percentage of edit pairs based on the time difference between edits, number of common categories and number of hops between the edit pairs for both vandal and benign users.



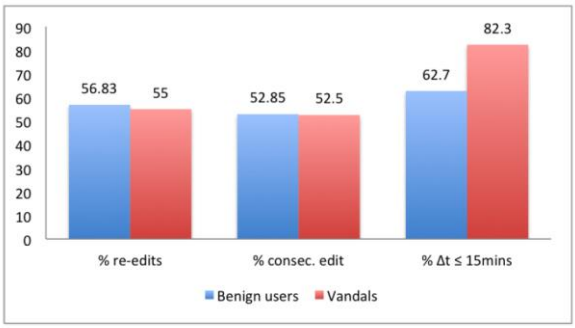
Type of edit and time data overall edits (Δt_c is the elapsed time between edits on the same page).

Figure 14



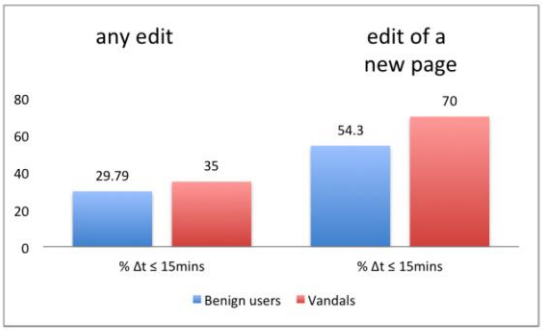
For a consecutive edit (p_1, p_2), how users surf from p_1 to the edit of a new page p_2.

Figure 15



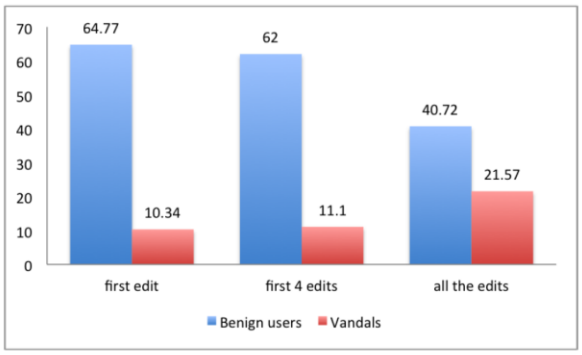
First 4 edits in users log.

Figure 16



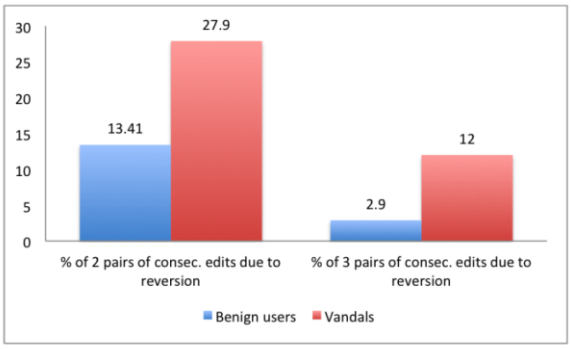
Edit time.

Figure 17



Meta-pages editing.

Figure 18



Edit war (involves reversion).

Figure 19

Figures 14 to 16 shows the similarities between vandals and benign users in terms of re-edits, navigation behavior and initial edits.

3.1.1 Differences between vandals and benign users

Figures 17 to 19 shows the differences between the behaviors of vandals and benign users. Some of the key points to be noted are:

- Vandals are faster at editing than benign users: Benign users only complete 29.79% of their edits in 15 minutes while vandals complete 35% of their edits within a time span of 15 minutes from the previous edit.
- Benign users take more time in editing a new page when compared to vandals: Within 15 minutes of their last edit itself, vandals complete 70% of their edits to a page unedited by them while the same for benign users is 54%. This is possibly because benign users take time to understand the article and has to check the given facts to make sure if it needs an edit or not. In a similar case, vandals don't have to think twice.
- Benign users edit much more meta pages when compared to the meta page edit behavior of vandals: The probability of a benign user editing also the meta page when making update to an article is 64.77%. At the same time, vandals seem to edit meta pages only 10.34% of the times. If we are to look at a user's first four edits, 62% of the edits by benign users are on meta pages, while this is only 11.11% for vandals. That being said, vandals does seem to edit more meta pages later but still not to the extent of benign users. If we look at the statistics for all the edits, the meta page share for benign users is 40.72% while it is 21.57% for vandals.
-

CHAPTER 4

DESIGN AND IMPLEMENTATION

Different approaches including probabilistic, graph based, cluster based and embedding based approaches are already existing in identifying vandalism on Wikipedia. But most of the approaches require at least a small amount of training data for the fraudulent class. Also, there is considerable amount of research available on novelty detection – which is another way to think about the issue. But none of the unsupervised approaches has resulted in a performance similar to the state of the art model performances. Our goal is to develop a framework which can be used for fraud detection involving user representation learning and single class classifications, in which we'll be solving one of the key issues of single class classifications, which is low confidence scores of predictions, by using a complimentary generative adversarial network for generating synthetic data for fraudulent class.

4.1 Approach

Our approach consists of two training steps. First step is to learn the non-vandal or benign user representations. For this purpose, an LSTM-Autoencoder is trained based on the non-vandal users and their edit sequences. As we have discussed in the last section, and LSTM-Autoencoder consists of an LSTM encoder and an LSTM decoder which takes part in this step. The idea behind this step is that the encoder will try to encode each of the inputs into user representations and the decoder will try to convert the representation back to the original input. Since reconstruction is the goal of this process, the encoder will learn to produce better feature rich representations and the decoder will learn to better reconstruct the original input with time. After training, the LSTM-Autoencoder will be able to capture the important features of the user activity sequences. And this trained network will be used later to generate the user representations of new data points.

The second step in our approach is to train a complimentary GAN. The discriminator of that network needs to be able to correctly identify vandals and non-vandals correctly from the input user representations. The objective of the generator in the complimentary GAN network is to generate user representations which would refer to vandal users. This is done by making sure that the generated samples by the discriminator are in the areas with low-density of non-vandal users. This is also based on the assumption that since the behaviour of vandals and non-vandals

differ, their user representation densities in different areas of the continuous feature space will also differ. This difference in behaviour is also observed during the exploratory data analysis. Overall, the training framework would look like:

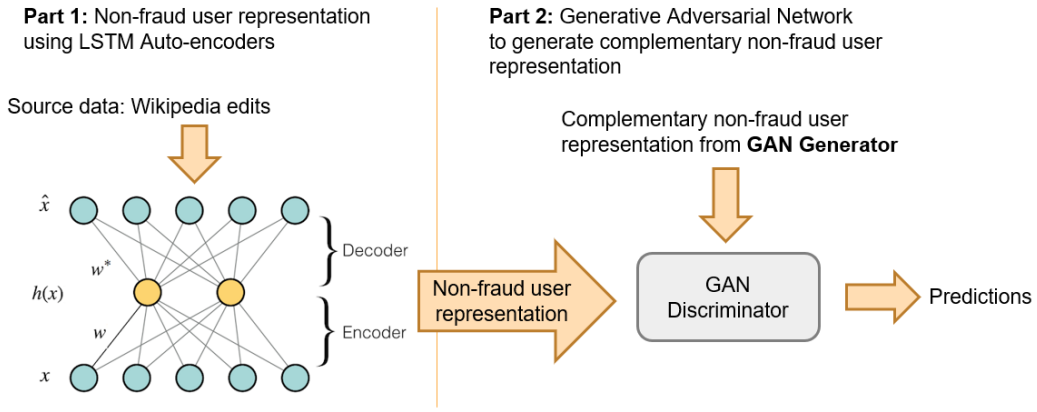


Figure 20: Proposed approach with two steps: user representation using LSTM-autoencoders and detection using GAN discriminator.

The simplified pseudo-code for the proposed approach is:

Inputs: Training data containing only benign users $M_{benign} = \{X_1, X_2, .. X_N\}$, Number of training epochs required for LSTM autoencoder $Epoch_{AE}$ and complimentary GAN $Epoch_{GAN}$

Outputs: LSTM Autoencoder and complimentary GAN models trained on $M_{benign.}$

1. Parameter initialization for LSTM autoencoder and complimentary GAN.
2. For each epoch until $Epoch_{AE}$ times:
    a. For each user in M_{benign}:
        i. Encode the user sequence using LSTM Encoder.
        ii. LSTM Decoder recreates the user sequence using the output of the LSTM Encoder.
        iii. Calculate loss function and optimize parameters for the LSTM Autoencoder.
3. For each user in M_{benign}:
    a. Compute the user representation of the user using the encoder from the LSTM auto encoder.
4. For each epoch until $Epoch_{GAN}$ times:
    a. For each benign user representation prepared in Step 3:

i. Train the generator to generate user representations in the low-density areas of the real benign user representation space.
ii. Train the discriminator to separate benign users and generated complementary samples.

5. Output trained LSTM auto encoder and complementary GAN.

4.1.1 LSTM-Autoencoder for User Representation

Encoding the users or user sequences into a continuous space is the first step in our approach. Each user has a sequence of activities and this can vary in length for different users. An example of the sequence that we are using for Wikipedia vandalism detection is the edit sequences of pages by the user. As different users might edit different number of pages, essentially the user sequence length is a variable length sequence. But this will be encoded into a fixed length user representation as the output of the LSTM autoencoder. This can be formalized as:
For a user u with T activities, the activity sequence will be:
$X_u = (x_1, \ldots, x_t, \ldots, x_T)$
Where x_t is the t-th activity vector.

Encoding: In the first phase, the LSTM encoder transforms the variable length sequence X_u into a fixed length user representation. This is the encoding phase. In this phase, t-th hidden vector of the LSTM encoder represents the details of the complete user sequence and is considered as the user representation of user u.

Decoding: In the second phase of the LSTM auto encoder, the decoder accepts the output of the encoder, which is the user representation of user u, as input and tries to recreate the original user sequence X_u. The recreation of the t-the hidden vector of the decoder gives the recreated user sequence X_{rec}.

The loss function will be based on the sum of the individual differences between each elements of the activity vector X_u and X_{rec}.

1. Complimentary GAN

A feed-forward neural network is used as the generator for the complimentary GAN and its outputs have the same dimensions as the user representation. Normally, the generators of

regular GANs are trained to generate fake benign user representations to be as close as possible with the distribution of the real benign users in the user space. But in our approach, the generator of the complimentary GAN learns a generative distribution which is similar to the complimentary distribution of the actual benign user representations.

The definition of the complementary distribution p* is:

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

Equation 1

ϵ is the definition of the threshold for high-density regions.

τ is the term for normalization to decide how dense should the generative distribution be, in comparison with the real benign user representation.

B_v is the user representation space.

p_{data} is the real benign user representation distribution

p* is the complementary user representation distribution

p_G is the generated fake benign user representation distribution

Loss function for the complementary GAN generator is based on the KL divergence between pG and p* and the feature matching loss [37] for making sure that the generated distribution is close to the complementary distribution and is constrained to the user representation space Bv. In other words, the generator of the complementary GAN tries to generate samples that are in the low density regions of real benign users. In the meantime, the discriminator of the complementary GAN is trained to identify and separate the benign users and generated complementary user representations.

Figure 21 illustrates the advantage of using a complimentary GAN instead of a regular GAN (figure only for illustration purposes)

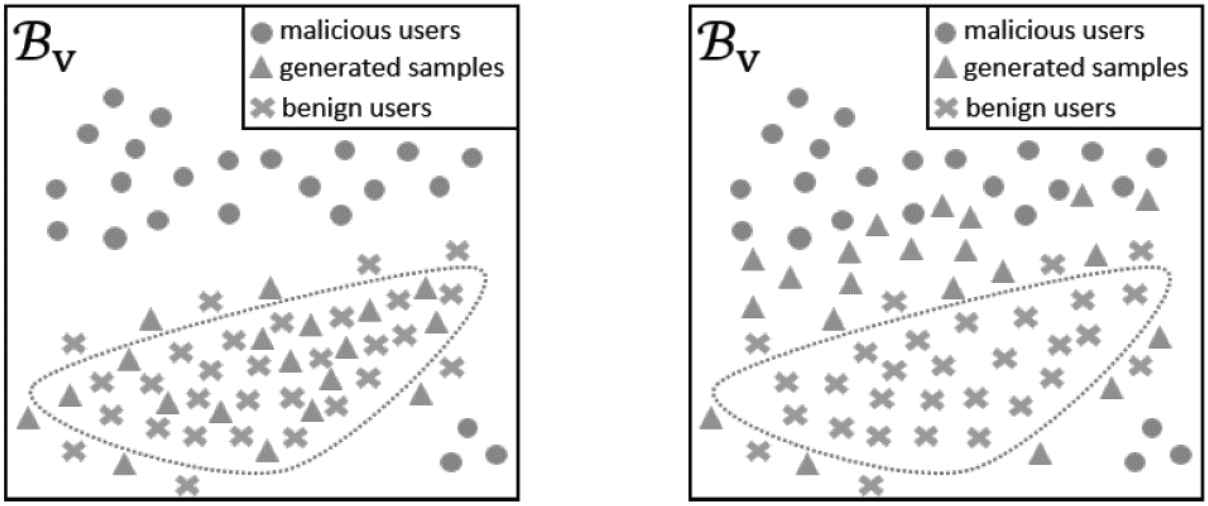


Figure 21

Even though there is similarity in a regular GAN and complementary GAN discriminators in terms of objective functions, the efficiency and confidence with which each of them detects malicious users are different by a large margin. As seen in Figure xxxx (a), in a regular GAN, the generated fake benign user representations and real benign user representations are very close to each other and are intermixed in some regions. This is because in a regular GAN, the generator tries to generate fake benign users as similar as possible to the real benign users. This results in low confidence for the discriminator predictions for real malicious users. Another issue with such a discriminator is that it might fail to identify new kinds of malicious users as the decision boundaries might not be well generalized.

At the same time, in Figure 21 (b), we see that the generated user representations are in the low density areas of the real benign users so that the decision boundaries are well generalized for areas with real benign users and the confidence for the identification of real and malicious users will be much higher than what we have in a regular GAN.

CHAPTER 5

EVALUATION

To evaluate our approach, we use a subset of the UMDWikipedia dataset [10] as described in the Chapter 2. The original UMDWikipedia dataset contains the details of 770K edits spanning 19 months which includes 17K vandals and 17k benign users. The dataset contains the details of the edits made by these users. The number of edits made by different users can vary quite widely. To avoid the outliers, we focus on those users with edit sequence lengths ranging from 4 to 50. 10528 benign users and 11495 vandals are left after this filtering. We reused the feature vector from VEWS research for this.

Our experiment will be compare our approach with other single class classification algorithms, with and without using the user representations. The comparison of results using user representation and without using user representations will give us an idea about if the usage of LSTM auto encoders in the approach is useful. And the comparison of other single class classifications with our trained discriminator will give us a view if the use of complementary GANs result in improved performance.

Hyperparameters used: We selected the dimension of the hidden layer for LSTM auto encoder as 200 and we set the training epochs for the LSTM auto encoder as 20. The discriminator for the complementary GAN is a feed forward network with 2 hidden layers with 100 and 50 dimensions each. The generator has an input layer of size 50 which takes noise as input. It also has one hidden layer with 100 dimensions. The input layer of the discriminator will have the same dimensions as the user representations and the input layer of the generator will take 50 dimensions of noise and the output layer of the complementary GAN generator will have the same dimensions as the user representations. The ideal size of the user representation was found to be 200 in our experiments. The training epoch of complementary GAN is 50. The threshold ϵ defined in Equation 14 is set as the 5-quantile probability of benign users as predicted by a pre-trained discriminator was used as the threshold ϵ. This was set based on trial and error.

Code: http://bit.ly/2SH4P3M

Baselines:

Our experiment will be compare it with other single class classification algorithms, with and without using the user representations. The baselines that we'll be using are:

1. **OCSVM:** One class SVM trains a support vector machine to identify a decision hypersphere encompassing the real benign users and everything outside this hypersphere will be tagged as the opposite class.
2. **OCNN:** One class nearest neighbours judges data points based on the distances to its nearest neighbours in the dataset and compare it with the average distance between those. Distant data points are classified as out of class. OCNN requires a small amount of labelled data for tuning the threshold distance. So cannot be said completely as a single class classification algorithm, but since this labelled data requirement is as small as 5%, we are still using it in our experiment.

We are using two methods for comparing these models to our approach

1. By using all the edit features as described in Chapter xxxx as the raw feature vector as input to the baseline models.
2. By using the user representations generated by the encoder from the LSTM auto encoder as input to the baseline models.

A set of randomly selected 7000 benign users is used as the training set. And a randomly selected list of 3000 vandals and 3000 benign users are used as the test set.

The results are as follows:

Input	Algorithm	Precision	Recall	F1	Accuracy
Raw Feature Vector	OCNN	0.5632	0.8789	0.6841	0.6019
	OCSVM	0.6849	**0.9752**	0.7702	0.7411
User representation	OCNN	0.8214	0.8092	0.8145	0.812
	OCSVM	0.6246	0.9592	0.7783	0.7249
	Our approach	**0.8946**	0.9305	**0.8968**	**0.8961**

Table 1

Here we see that our approach performs well in terms of Precision, F1 score and Accuracy. Also we can see that the usage of user representations greatly improves the performance of the baseline models, which means our usage of LSTM auto encoder for generating user representations is useful. This also means that the LSTM auto encoder is able to learn the significant details of the user and represent it efficiently. From our experiment we can clearly

see that both the phases – the LSTM auto encoder and the complementary GAN are resulting in improving the vandal identification capabilities in the Wikipedia vandalism detection problem.

CHAPTER 6

CONCLUSION

Through this thesis, till now, we've successfully planned and implemented a one-class classification approach for fraud detection when only benign user data is available, using LSTM-Autoencoders for user representation generation and complementary GANs for further classification learning. Since our complementary GAN approach generates users only in the low-density areas of the feature space, our detection confidence for non-vandal users can be improved when compared to other OCC approaches. We experimented our approach on a random subset of the UMDWikipedia dataset with extensive feature engineering and found that the proposed approach clearly outperforms the baseline models.

REFERENCES

[1] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. 2014. Uncovering Large Groups of Active Malicious Accounts in Online Social Networks. In CCS.

[2] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. CatchSync: Catching Synchronized Behavior in Large Directed Graphs. In KDD.

[3] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In NIPS

[4] David M. J. Tax and Robert P. W. Duin. 2001. Uniform Object Generation for Optimizing One-class Classifiers. JMLR 2, Dec (2001), 155–173.

[5] Shuhan Yuan, Panpan Zheng, Xintao Wu, and Yang Xiang. 2017. Wikipedia Vandal Early Detection: from User Behavior to User Embedding. In ECML/PKDD

[6] Sabokrou, Mohammad and Khalooei, Mohammad and Fathy, Mahmood and Adeli, Ehsan. 2018. Adversarially Learned One-Class Classifier for Novelty Detection. CVPR/2018

[7] Wikipedia contributors. 2010. ClueBot NG. (2010). http://en.wikipedia.org/wiki/User:ClueBot_NG.

[8] STiki: http://en.wikipedia.org/wiki/Wikipedia:STiki.

[9] Snuggle: http://en.wikipedia.org/wiki/Wikipedia:Snuggle.

[10] Srijan Kumar, Francesca Spezzano, and V.S. Subrahmanian. 2015. VEWS: A Wikipedia Vandal Early Warning System. In KDD. 607–616.

[11] S. M. Mola-Velasco, “Wikipedia vandalism detection through machine learning: Feature review and new proposals - lab report for pan at clef 2010.” in CLEF, 2010.

[12] A. G. West, S. Kannan, and I. Lee, “Detecting wikipedia vandalism via spatio-temporal analysis of revision metadata?” in EUROSEC, 2010, pp. 22–28.

[13] B. T. Adler, L. de Alfaro, and I. Pye, “Detecting wikipedia vandalism using wikitrust - lab report for PAN at CLEF 2010,” in CLEF, 2010.

[14] B. T. Adler, L. de Alfaro, S. M. Mola-Velasco, P. Rosso, and A. G. West, “Wikipedia vandalism detection: Combining natural language, metadata, and reputation features,” in CICLing, 2011, pp. 277–288.

[15] M. Potthast, B. Stein, and R. Gerling, “Automatic vandalism detection in wikipedia,” in ECIR, 2008, pp. 663–668.

[16] M. Sumbana, M. GonÃgalves, R. Silva, J. Almeida, and A. Veloso, “Automatic vandalism detection in wikipedia with active associative classification,” in TPDL, 2012, pp. 138–143.

[17] Shebuti Rayana and Leman Akoglu. 2015. Collective Opinion Spam Detection: Bridging Review Networks and Metadata. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’15). ACM, New York, NY, USA, 985–994.

[18] A Conrad Nied, Leo Stewart, Emma Spiro, and Kate Starbird. Alternative narratives of crisis events: Communities and social botnets engaged on social media. In Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing companion. ACM, 2017.

[19] Loosli, Ga¨elle, Canu, St´ephane, and Bottou, L´eon. Training invariant support vector machines using selective sampling. In Bottou, L´eon, Chapelle, Olivier, DeCoste, Dennis, and Weston, Jason (eds.), Large Scale Kernel Machines, pp. 301–320. MIT Press, Cambridge, MA., 2007. URL

[20] A. Kittur, B. Suh, B. A. Pendleton, and E. H. Chi, “He says, she says: Conflict and coordination in wikipedia,” in SIGCHI, 2007, pp. 453–462.

[21] G. Wu, D. Greene, B. Smyth, and P. Cunningham. Distortion as a validation criterion in the identification of suspicious reviews. Technical Report UCD-CSI-2010-04, University College Dublin, 2010.

[22] Zhou, L., Shi, Y., and Zhang, D. 2008. A Statistical Language Modeling Approach to Online Deception Detection. IEEE Transactions on Knowledge and Data Engineering.

[23] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. CopyCatch: Stopping Group Attacks by Spotting Lockstep Behavior in Social Networks. In Proceedings of the 22nd International Conference on World Wide Web (WWW), 2013.

[24] Joseph Rocca and Sebastian Rocca, Understanding Generative Adversarial Networks (GANs): https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

[25] Zihang Dai, Zhilin Yang, Fan Yang, William W. Cohen, and Ruslan Salakhutdinov. 2017. Good Semi-supervised Learning that Requires a Bad GAN. In NIPS.

[26] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density based algorithm for discovering clusters in large spatial databases with noise.. In KDD. 226–231.

[27] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde- Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. In NIPS.

[28] Federal Reserve Payments Study: https://www.federalreserve.gov/paymentsystems/2018-December-The-Federal-Reserve-Payments-Study.htm

[29] Consumer Sentinel network: https://www.ftc.gov/system/files/documents/reports/consumer-sentinel-network-data-book-january-december-2016/csn_cy-2016_data_book.pdf

[30] Srijan Kumar and Neil Shah. 2018. False Information on Web and Social Media: A Survey. arXiv:1804.08559 [cs] (2018).

[31] X. Ying, X. Wu, and D. Barbará. 2011. Spectrum based fraud detection in social networks. In ICDE. 912–923.

[32] Shuhan Yuan, Xintao Wu, Jun Li, and Aidong Lu. 2017. Spectrum-based deep neural networks for fraud detection. In CIKM.

[33] Arjun Mukherjee, Vivek Venkataraman, Bing Liu, and Natalie S. Glance. 2013. What Yelp Fake Review Filter Might Be Doing?. In ICWSM. 409–418.

[34] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. 2010. Detecting Product Review Spammers Using Rating Behaviors. In CIKM. 939–948.

[35] Shebuti Rayana and Leman Akoglu. 2015. Collective Opinion Spam Detection: Bridging Review Networks and Metadata. In Proceedings of the 21th ACM SIGKDD International

Conference on Knowledge Discovery and Data Mining (KDD ’15). ACM, New York, NY, USA, 985–994.

[36] Detecting Wikipedia Vandalism via Spatio-Temporal Analysis of Revision Metadata https://www.andrew-g-west.com/docs/wiki_eurosec_final.pdf

[37] Stefan Heindorf1, Martin Potthast, Benno Stein, Gregor Engels1 2013 Vandalism Detection in Wikidata

[38] Larry M. Manevitz and Malik Yousef. 2001. One-Class SVMs for Document Classification. JMLR 2, Dec (2001), 139–154.

[39] David M. J. Tax and Robert P. W. Duin. 2004. Support Vector Data Description. Machine Learning 54, 1 (2004), 45–66.

[40] RNN Implementation https://peterroelants.github.io/ posts/ rnn-implementation-part01 /

[41] Understanding LSTM Networks https://colah.github.io/ posts/ 2015-08-Understanding-LSTMs/

[42] Applied Deep Learning Part 3 https://towardsdatascience.com/ applied – deep – learning – part – 3 – autoencoders - 1c083af4d798

[43] B. T. Adler, L. de Alfaro, S. M. Mola-Velasco, P. Rosso, and A. G. West, “Wikipedia vandalism detection: Combining natural language, metadata, and reputation features,” in CICLing, 2011, pp. 277 – 288.

[44] O. Ferschke, J. Daxenberger, and I. Gurevych, “A survey of nlp methods and resources for analyzing the collaborative writing process in wikipedia,” in The People’s Web Meets NLP. Springer, 2013, pp. 121–160.

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

[46] M. Sumbana, M. GonÃgalves, R. Silva, J. Almeida, and A. Veloso, “Automatic vandalism detection in wikipedia with active associative classification,” in TPDL, 2012, pp. 138–143.

[47] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). ACM, New York, NY, USA, 895–904.

[48] Shebuti Rayana and Leman Akoglu. 2015. Collective Opinion Spam Detection: Bridging Review Networks and Metadata. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15). ACM, New York, NY, USA, 985–994.

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

[50] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval, pages 42–49, Berkeley, US, 1999. ACM Press, New York, US. URL http://www.cs.cmu.edu/ yiming/papers.yy/sigir99.ps