Keystroke Injection Detection and Mitigation

Emory Lindsey
Department of Computer Science and Information Systems
Georgia College & State University

Introduction

Keystroke injection is a hardware attack by which a USB device is used to type a sequence of keystrokes at inhuman speeds. Upon connection, a typical OS views such a device as a USB HID keyboard device. Since the USB protocol does not provide a standard for device authentication, USB buses and, by extension, operating systems often blindly trust connected devices regarding their advertised capabilities. The purpose of this research is to devise a simple detection mechanism that is capable of detecting and stopping a keystroke injection attack.

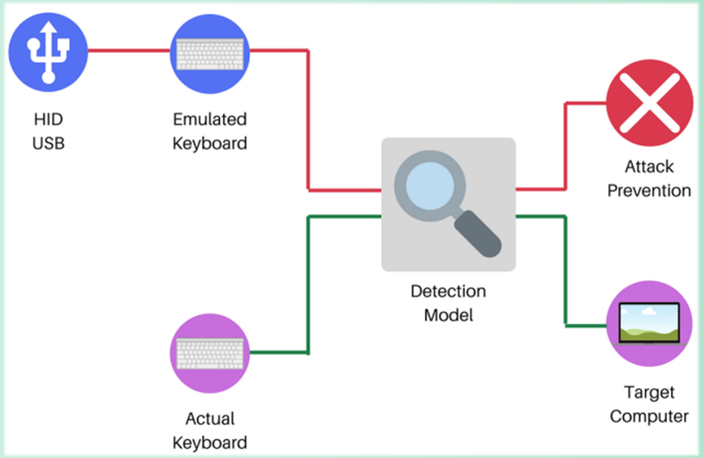


Figure 1. Visual Overview of Detection/Prevention Model

Materials

Keystroke Injection Devices:

1. Raspberry Pi Zero W (P4wnP1 A.L.O.A. image)
2. Bash Bunny

Host/Victim Machine:

Raspberry Pi 3 Model B+ (Raspbian image)

Methods

For primary testing, the Raspberry Pi Zero W was used to test a range of keystroke separation timings, denoted as *s*, where $s \approx 10$ ms, 20 ms, 30 ms, … 80 ms. For secondary testing, the Bash Bunny was used and was found to have a fixed *value* of $s \approx 10$ ms. Ten test runs were conducted for each distinct value of *s* and for each injection device used, recording the average number of keystrokes allowed for each value of *s*, denoted as *k*.

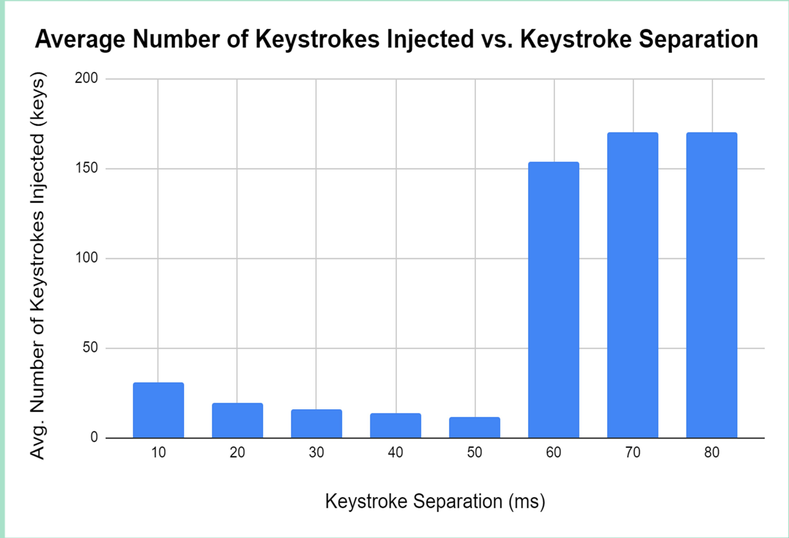


Figure 2. Primary test results (P4wnP1 A.L.O.A.)

Results

During primary testing, the value of *s* and *k* were found to be inversely proportional for values of $s \approx 10$ ms, 20 ms, …, 50 ms with the maximum value of $k \approx 30.9$ and the minimum value being $k \approx 11.5$. There was a significant number of false negatives for values of $s \approx 60$ ms, 70, and 80 ms. During secondary testing, the detection model allowed an average number of 10.8 keystrokes.

Conclusion

In order to effectively analyze the primary and secondary test results, a few factors must be taken into consideration. First, as mentioned in the Methods and Results section, the values for *s* are approximate values. This is the case due to the fact that the monitoring software occasionally calculated values slightly above/below the variable value for *s*. This fact coupled with how the detection model uses s_{avg} calculated over four keystroke events helps to explain the prevalence of false negatives as the value of *s* becomes close to the threshold of 80 ms. Secondly, a value of $k \approx 30.9$ might be alarming; however, this maximum value includes the whitespace characters found within the payload and any keystrokes needed to launch a terminal environment within a given operating system. In addition, many malicious Linux-based keystroke injection payloads are themselves longer than 30 characters. Finally, an attacker is more likely to choose a Bash Bunny for a social engineering engagement duce to its inconspicuous nature. Therefore, due to practical points mentioned above and the promising results found in secondary testing, the detection/prevention model was discovered to be effective in a practical sense.

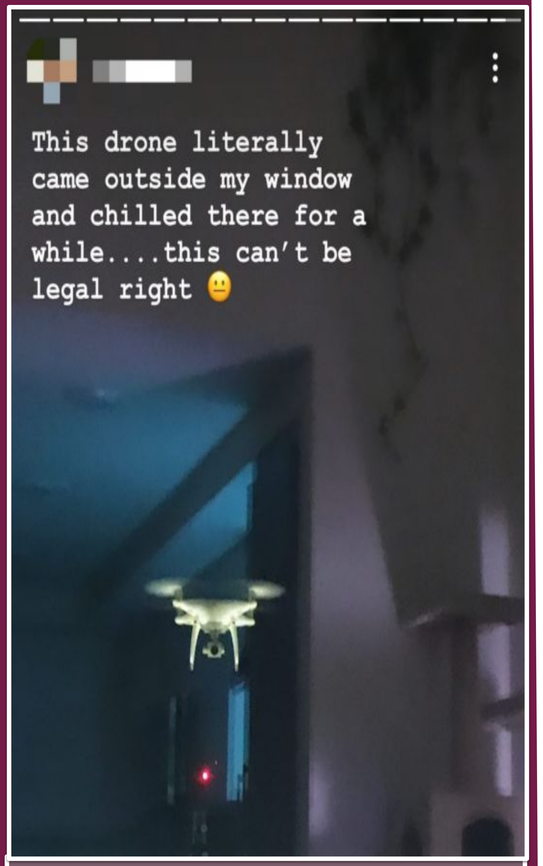
References

1. Neuner, S., Voyiatzis, A. G., Fotopoulos, S., Mulliner, C., & Weippl, E. R.: USBlock: Blocking USB-based Keypress Injection Attacks.(2018, July 10).

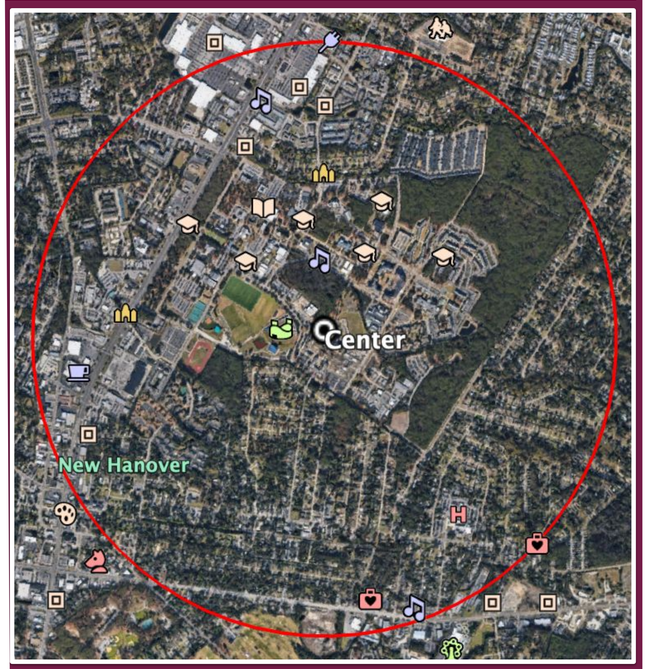
2. Umphress, D., Williams, G.: Identity verification through keyboard characteristics.International journal of man-machine studies 23(3), 263–273 (1985)

3. S. (2020, November 14). Reverse Shell Cheat Sheet. Retrieved December 16, 2020, from https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md

Finding the launch location of a drone

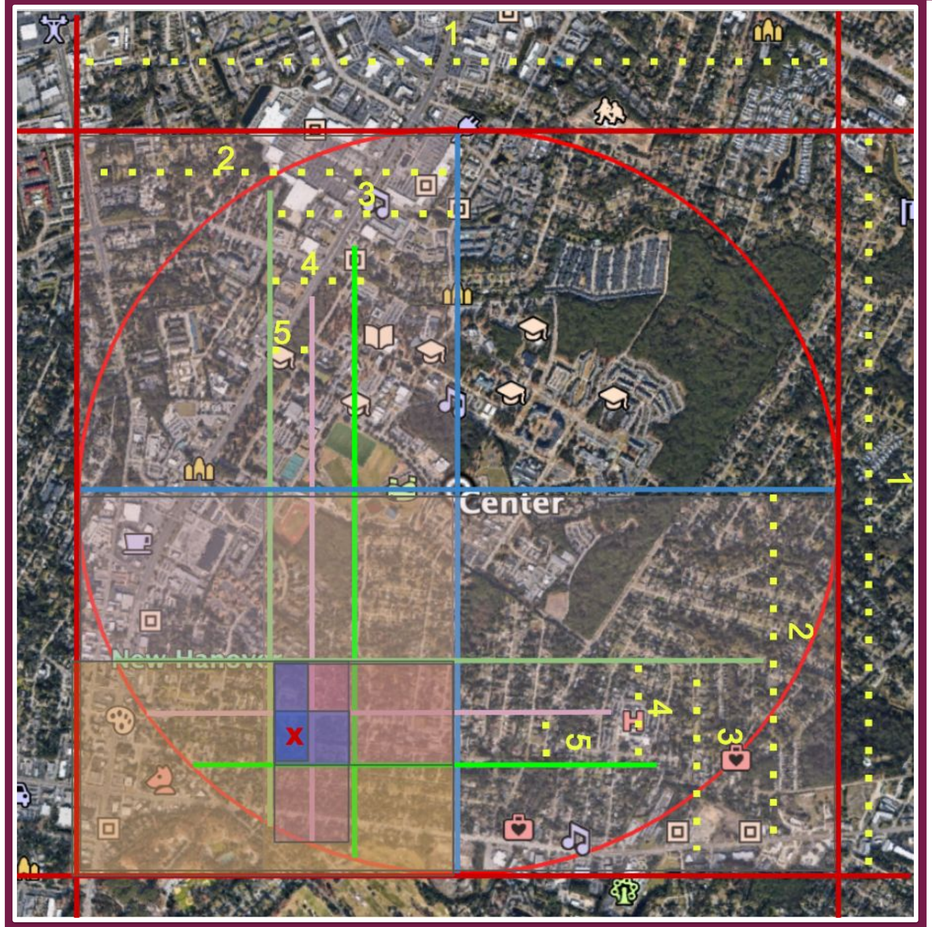


Today, we are witnessing an increase in the number of cases of drones flying into restricted areas or violating people's privacy. The most common methods today to deal with violating drones this are by jamming or simply shooting them down without collecting any information about where the drone was launched from. Our method targets GPS-enabled drones and not only grounds them but also calculates the launch coordinates by exploiting "Return to Home" function, which returns a drone to launch coordinates if it becomes out of reach. In the proposed method we use manipulated GPS broadcasts to ground a drone and find the launch coordinates

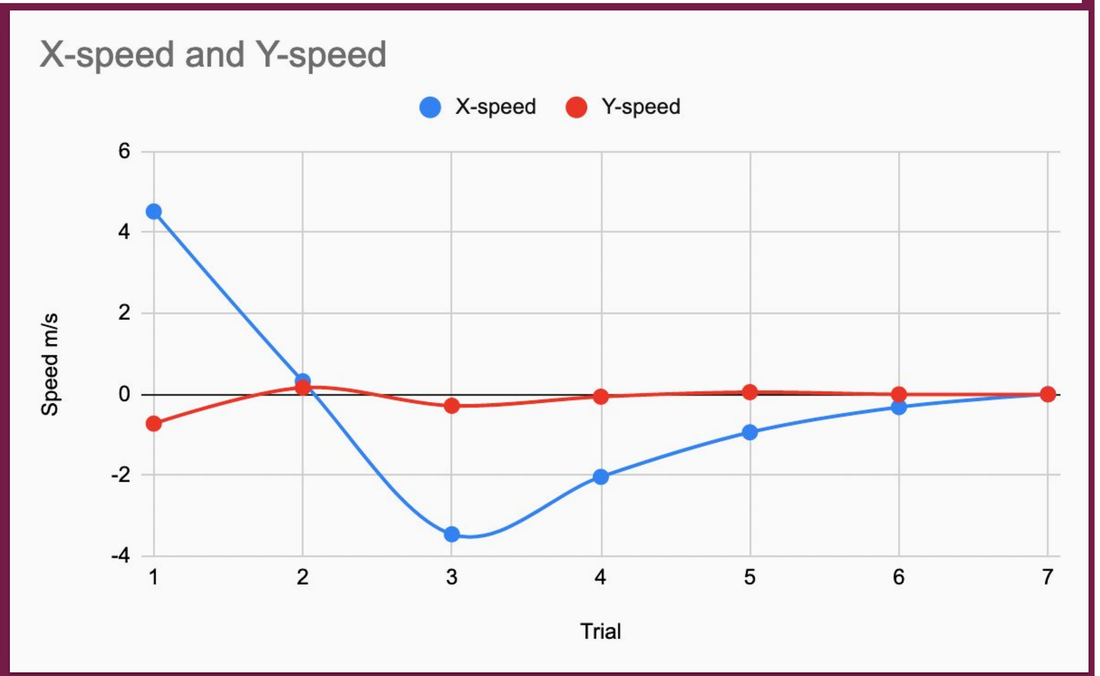


Remote controlled drones have limited range. At the first step in our algorithm is to find the boundaries of possible launch coordinates. In this simulation, 1 mile range is considered. This value is configurable.

After "Return to home" function is invoked, which can be forced by jamming. The next step is to broadcast manipulated GPS signal guesses in attempt to ground the drone. At the same time, the change of the x and y components of the drone's speed is observed. The broadcasted GPS guesses are generated following the bisection method. In a way after every broadcasted guess, the drone's response for this guess is utilized to close down on the drones launch location.



The bi-section algorithm looks for a sign change in order to guess the coordinates that can lead drone's speed to go to zero. When there is a sign change it means that we are closing down on the root. In our simulation, it took 7 trials to guess the coordinates that lead to to the x and y components to become 0. These are the launch coordinates



What is accomplished :

Proof of concept on how a software defined radio transmission system (HackRF was used) can be used to spoof GPS signals and how GPS enabled devices calculate spoofed GPS location.

Also, an algorithm is developed that finds the launch location by determining the guesses that will eventually converge to the solution

Next step:

To test the proposed algorithm utilizing software defined radio transmission system on drone's integrated control circuit.

Then finally test the system with a GPS enabled drone in action utilizing a software defined radio transmission system with directional antennas

By Nick Roy with Supervision of Dr. Hosam Alamleh

Opt-In Attendance Tracking

Aidan Shene, Jake Aldridge & Dr. Hosam Alamleh

University of North Carolina Wilmington
Department of Computer Science

Introduction

Attendance tracking is important for optimizing the educational process, and many different systems and technologies exist and have been implemented to do so. Current attendance tracking methods are often time consuming which wastes valuable class time, can require a lot of effort from the instructor or student, and may make it easy for students to falsify attendance. Our mobile app minimizes these issues by creating a quick, zero-effort system for attendance which requires user devices to be within range of the instructors, making attendance much more difficult to falsify. Using a "location proof" based off of the Received Signal Strength Indicator (RSSI) measured for the access points in the area, our system only requires instructors and students to activate the app so that this proof can be created. By determining the location of student devices only in reference to the instructor's device, the location privacy of the user is maintained.

Ethical Concerns and Privacy

Protection against electronic surveillance is a right afforded under the fourth amendment and the Electronic Communications Privacy Act (ECPA), the latter of which outlines how victims can sue individuals involved in unlawful intrusions for damages or equitable relief.

In recent years there have been more calls for police forces to make use of access point data, to use as evidence. While this poses great privacy concerns, there could be many applications that routinely monitor how many devices are in an area and when changes occur.

Due to the innate respect for users' privacy and the inherent legal complications regarding performing this type of user surveillance, we have opted for a different solution. Data is not collected until approval is explicitly granted by the user from the app on their device, making it required for each class attended.

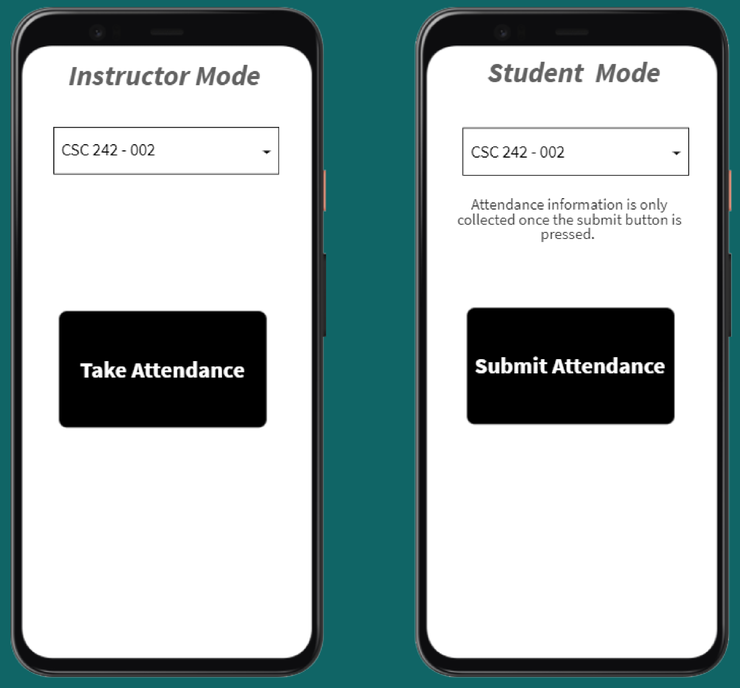


Figure 1. This mock-up showcases the main features of the app. Students and Instructors would have two different modes; the instructor's focused on collected attendance over a set time period and the student's centered around creating an individual location proof.

Wider Applications & Use Cases

The range of the radio frequency signals emitted from Wi-Fi access points is one of the limiting factors in applications of our research. It is feasible to determine whether a device is inside of a building, classroom, or in their immediate proximity.

In a more sizeable ecosystem, the feasibility begins to fade. If applied to a larger environment, more manual data collection is likely required to merit its use. This app has the potential to scale to a large number of devices. The process would remain the same regardless of the sample size, though the total time to classify each device change.

The purpose of our application is not necessarily to find exactly where a user is, but rather to definitively say where they are not — in the classroom. The scope of our research's applications is limited mostly to attendance based scenarios, whether for class, a meeting, or an event; however, it could be used to measure the number of users in a given location. Measurements collected could then be used to determine whether a building is at capacity.

Figure 2. Top 10 Access Points by MAC address versus average RSSI. The points labeled as "Reference" represent the RSSI from a fixed location in the classroom used as a basis of comparison for the other recorded locations — in and outside of the classroom, but in the same building.

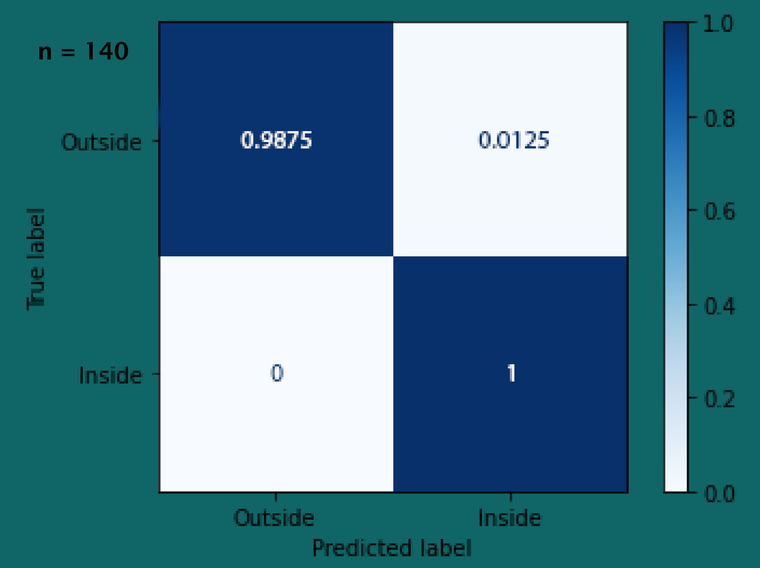


Figure 3. Predicted Location versus True Location. Average RSSI by Top 10 Access Points by MAC Address per location polled. Each sample represents an RSSI, which is used to predict whether the device is the inside or outside of the classroom. Similarity scores between a test data point and the reference dataset were calculated to determine if the test point was collected in or outside of the classroom.

Network Level Cloud Threats

Hannah McSwain
Department of Computer Science, University of North Georgia
Research Mentors: Dr. Ahmad Ghafarian

University of North Georgia

Introduction

Amazon Web Services is a leading provider of cloud computing technology. The company offers a wide variety of platforms to provide top tier scalability. As an alternative to on-premise web hosting, many companies are turning to AWS for cloud-based hosting solutions. The shift to cloud solutions raises the question of what security concerns are involved with cloud hosting alternatives. AWS emphasizes automation to promote ease of use. With the rise of new cloud computing technology, AWS network security settings should be evaluated. This project aims to conduct tests and analyses of network-level vulnerabilities on the Windows Server 2016 virtual machine powered by AWS.

Method

To best examine the Windows 2016 Server virtual machine's network security and functionality, tests were conducted beginning with the server's configuration. During the server setup process, time was devoted to analyzing each step to determine what procedures would be applied if the user chose standard configuration. In production environments where the servers would be used to host traffic, the administrator would need to devote time to ensure that the settings selected during this process were secure. Customization is entirely dependent on the intended use of the server, whether it be storage, testing, or web hosting. For the server in this experiment, standard storage and policies were applied because they were in the free tier. AWS offers several options when choosing an instance type. The tiers provide a variety of options for storage and advanced configuration. For this project, t2.micro was compatible with the free tier level, which included 1GB of memory, low to moderate network performance, and one virtual CPU.

Analyses

Analysis of the network level security of Microsoft's Windows Server 2016 was conducted using Microsoft Baseline Security Analyzer internally and Nmap running on a Kali Linux VM and MSEdge Windows 10 Flare VMs. The only concerning results produced from the Microsoft Baseline Security Analyzer were around IIS configuration, which had not been set up on the server since it was only intended for testing and not production. The scan revealed that automatic updates had not been configured. It would be essential to note that if an AWS AMI is being used to host web traffic; it would need to be configured after installation by the administrator. The scan revealed that updates need to be installed on the server, which was a surprising result to see that a new instance would not already have the most recent updates applied.

Nmap was the other primary tool used to analyze port vulnerability. The internal IP address for the server was 3.81.46.42, and the external IP address was 172.31.40.125. Nmap scans were conducted from within the Windows Server 2016 VM using the internal and external IP addresses. When the scan was conducted from the AMI VM with the external IP address, no open ports were found; however, nine open ports were found when the external IP address was used. This is to be expected since the scan was conducted internally on the system. The test of AWS security was also conducted by running a Nmap scan from an external Kali Linux and MSEdge Windows 10 VM. The results of scanning both external and internal IP addresses from the Linux VM produced similar results. When conducted in the Kali VM, no open ports were produced. The results reported that the host appeared to be down, but the target IP was blocking ping probes if it was up. Since the instance was running at the time of the scan, then it could be concluded that the AWS AMI was blocking the scan attempted by Nmap from Kali Linux.

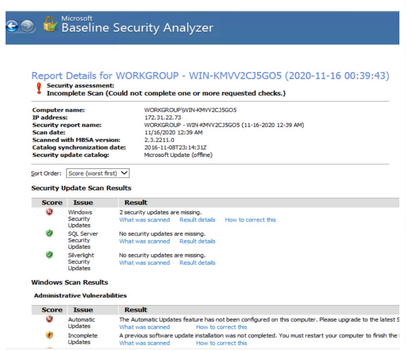
The next set of scanning through Nmap was conducted from an MSEdge Windows 10 VM running the Flare installation. The scan ran on the external IP inside the MSEdge Windows 10 VM showed that the machine was online, and port 3389/TCP ms-wbt-server was open. When the same scan was conducted on the external IP, it showed that the target IP was online, and 1000 ports were filtered but found no ports were open. A key point of analysis was found when configuring a server; the implications of standardized policies should be fully understood to know what has been applied.

Results

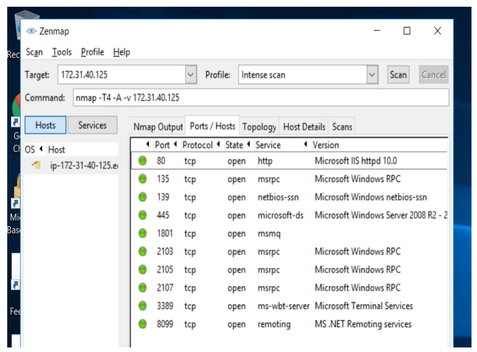
The results from the Microsoft Baseline Security Analyzer revealed that a particular configuration that is installed in the standard Windows Server 2016 AMI needs to be configured by the administrator. Although additional configuration is required, the scan conducted through Nmap did not produce results that would put into question the network level security of the VM.

Conclusions

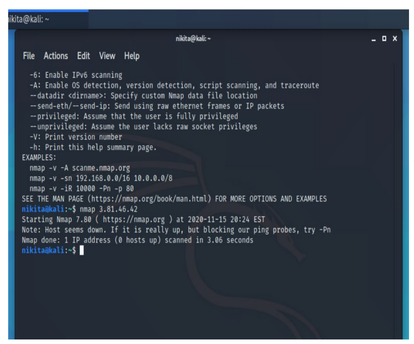
There were no results produced by the tests to indicate that the AWS hosted Windows Server 2016 has significant security vulnerabilities. The only major exposure that could impact any user would be the proper implementation of standardized security policies. When creating a new server instance, a user should know which procedures are being enforced on their account. With the positive impact that cloud hosting options brings, there is still concern about turning over on-premise hosting to a cloud alternative. Cloud hosting can certainly reduce cost since the payment model is based on services used; however, if factors such as the volume of page views can make cloud options more expensive than on-premise.



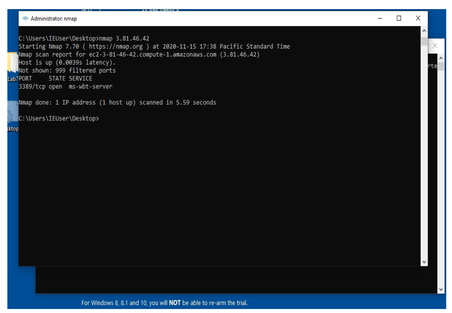
Microsoft Baseline Security Analyzer results provide information about the standard Windows Server 2016 AMI configuration.



Results of Nmap scan on internal IP (172.31.40.125) address conducted internally from Windows Server 2016 revealed that 9 ports were found open. When scan was conducted on the public IP address (3.81.46.42) no open ports were found.



Nmap scan of public and private IP addresses from Kali Linux Server both produced the same results. The scan reported that the host appeared to be down but if it was up then it was blocking ping probes.



Results from the scan inside the MSEdge Windows 10 VM on the public IP address (3.81.46.42) showed that the machine was online and port 3389/tcp ms-wbt-server was open. The internal IP (172.31.40.125) scan showed that all 1000 ports were filtered but found that no ports were open. Despite no open ports being found the scan did show that the target IP was online.

References

"Amazon Management Console ." *Https://Console.aws.amazon.com*, Amazon Wed Services.

Definition of Amazon Machine Image provided by Amazon Web Services.

Amazon Virtual Private Cloud. aws.amazon.com/vpc.

Nmap, nmap.org/.

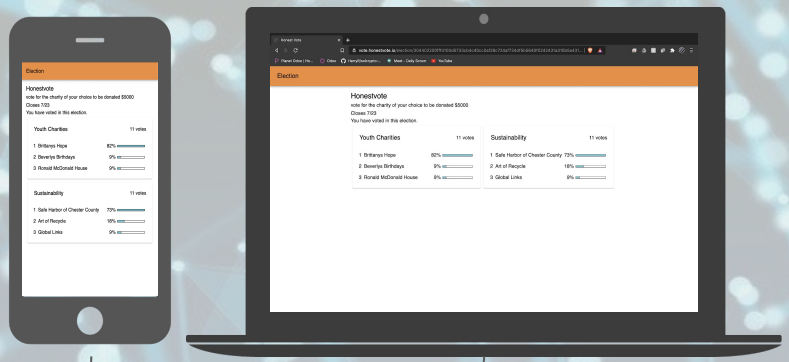
"What Is a Virtual Machine and How Does It Work: Microsoft Azure." *What Is a Virtual Machine and How Does It Work | Microsoft Azure*, azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/.

"What Is Cloud Computing? A Beginner's Guide: Microsoft Azure." *What Is Cloud Computing? A Beginner's Guide | Microsoft Azure*, azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

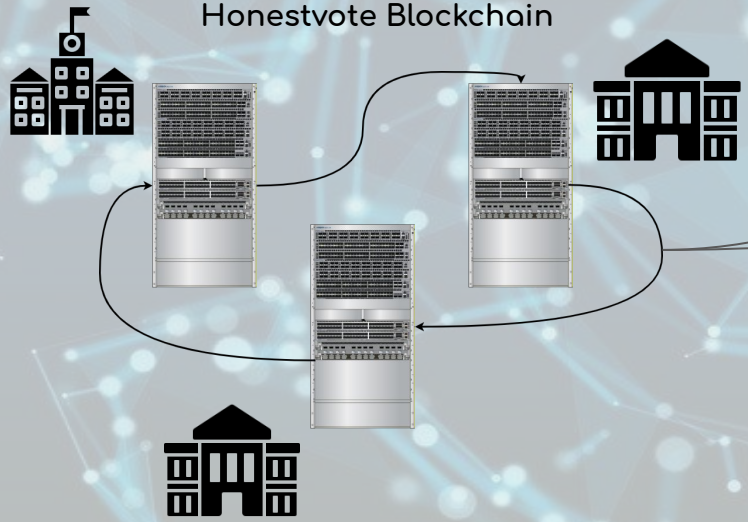
"What Is Cloud Computing? A Beginner's Guide: Microsoft Azure." *What Is Cloud Computing? A Beginner's Guide | Microsoft Azure*, azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

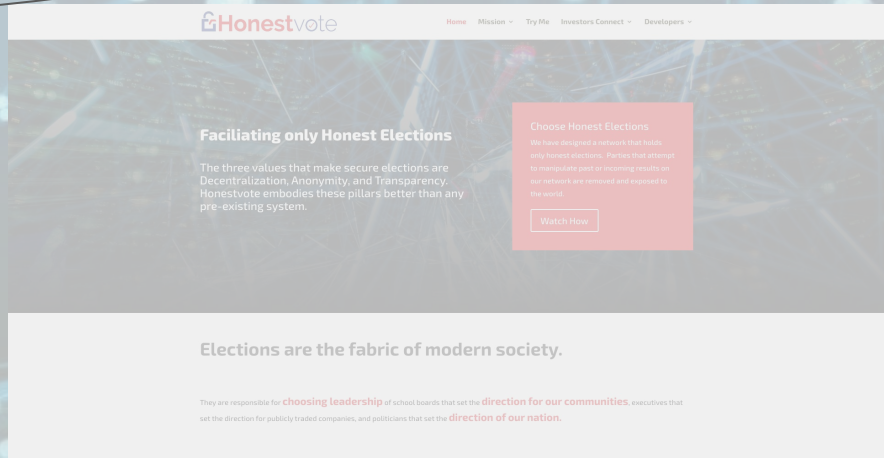
The first dedicated Proof of Authority (PoA) blockchcain network engineered to make voting safer

Client Application



Honestvote Blockchain





Honestvote

Using Unity to Develop an Interactive Fire Spread Simulation

Andrew Droubay, Dr. Anil Shende
Department of Mathematics, Computer Science, and Physics
Roanoke College

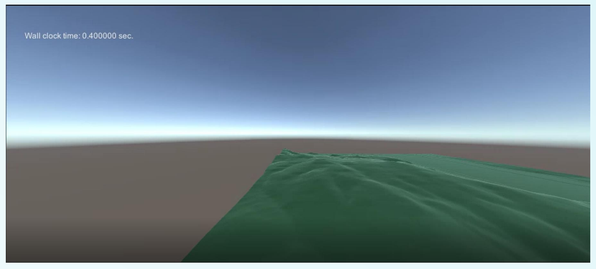
Introduction

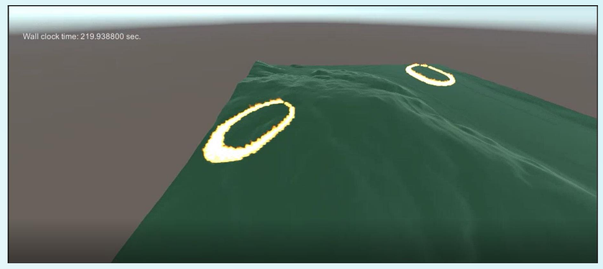
The range of ecological diversity managed by the United States Forest Service makes it difficult to train firefighters to work in many environments. Regional differences in fire spread behavior can result in increased hazards as unfamiliar firefighters deploy from across the country. To increase safety, this project aims to use fire spread models and the Unity game engine to create an interactive simulation that allows users to combat scientifically realistic fires in a digital environment using virtual reality (VR). The current prototype translates user control into an input read by the Forest Service's fire simulator and reads the output.

Materials

The simulator is built using the Unity Game Engine. Unity communicates with Forest Service's Wildland Fire Dynamics Simulator (WFDS) model. This communication places user input in WFDS, which can be read back to Unity. Because Unity is a game engine, this enables any standard VR headset to use and interact with the simulation.

Methodology





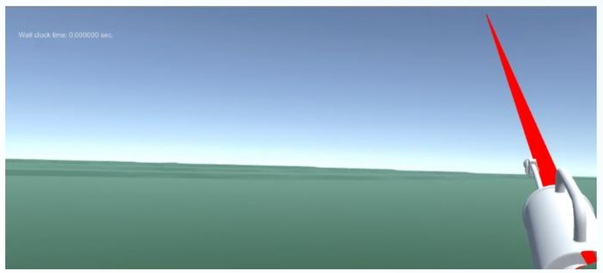
Aerial View of Fire Spread Over Terrain

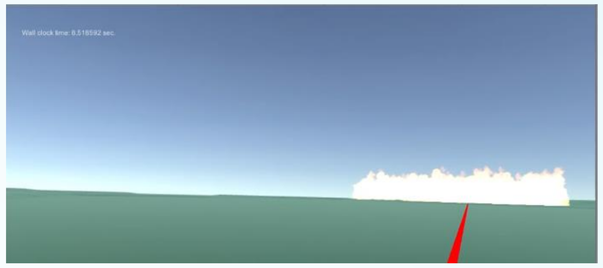
Feedback Loop

The Unity Game Engine runs scripts with C# code that recognizes user input and translates those points into data for the WFDS model. This data is then compiled to a new input file, which is sent to WFDS. The model then calculates the fire spread in parallel to the running simulation. Once the model finishes, the final data is sent to Unity, which will display it to the user.

Data Display

WFDS data is given as Time of Arrival (TOA) coordinates. Every terrain vertex has a value (in seconds) that indicates the time that the fire will reach that vertex. At that point, Unity creates a particle system representing a fire. Twenty seconds after a point is activated, the particle system is disabled as the fire fades. If this process is repeated for a series of points, it creates an accurate representation of how the fire spreads over time.





User View of Fire Ignition

Future Work

We have several different avenues for continuing this work. We want to upgrade the rendering engine, which should increase immersion and optimize the graphics. In addition, we plan to use more data from the WFDS model, such as heat release, wind and smoke clouds. We also plan to add other inputs, such as digging fire trenches, chopping trees, and different modes of player movement.

Most importantly, once the prototype has been thoroughly tested, it will be given to Forest Service firefighters to test its efficacy for fire training and comprehension.

References

Ahlberg, G., Enochsson, L., Gallagher, A.G., Hedman, L., Hogman, C., Mc-Clusky III, D.A., Ramel, S., Smith, C.D., Arvidsson, D.: Proficiency-based virtual reality training significantly reduces the error rate for residents during their first 10 laparoscopic cholecystectomies. The American journal of surgery193(6), 797–804(2007)2.

Bliss, J.P., Tidwell, P.D., Guest, M.A.: The effectiveness of virtual reality for administering spatial navigation training to firefighters. Presence: Teleoperators & Virtual Environments 6(1), 73–86 (1997)

Automating Software Traceability Link Recovery and Maintenance using Word Embeddings within a Shallow Neural Network

Marlan McInnes-Taylor, Faculty Mentor: Dr. Chris Mills
Florida State University Department of Computer Science

Abstract

In addition to code, software systems contain a multitude of files documenting features, known issues, legal requirements, etc. Software traceability is the ability to link related documents from these various sets through a process called *traceability link recovery*. While previous research has shown that established software traceability improves the quality of projects and makes them easier to maintain, establishing software traceability is often a manual, arduous, and error prone task. This research explores automating the process of software traceability link recovery using established techniques in machine learning. The results demonstrate that even with minimally tuned hyperparameters a shallow neural network can effectively predict which text-based artifacts within a software system are related to one another.

Introduction

Previous studies have shown that access to information explaining relationships between artifacts in a system leads to higher quality software and lower bug counts. This is largely credited to such information improving common software engineering tasks such as:

- program comprehension
- concept and bug localization
- defect prediction

The acquisition of traceability information is difficult and typically an afterthought during system construction. Consequently, hundreds or thousands of man hours can be spent after initial development manually inferring relationships between artifacts that are constantly in flux as the system changes during development and maintenance.

To address this situation, previous studies have attempted to either completely or partially automate the process of establishing traceability links between system components. In this work, we continue that research agenda by using neural networks to model *potential* links between text-based software artifacts and predict which are valid (i.e. two related documents) and which are invalid (i.e. two unrelated documents).

Materials and Methods

The training and test data were comprised of six datasets commonly used to validate approaches for automating traceability link recovery: Albergate, eAnci, eTour, iTrust, MODIS, and SMOS. The table below shows a breakdown of the data by project. Each of the projects involved in our evaluation is written in either Java or C++. The code used for parsing and cleaning the software artifacts is written in Python. TensorFlow 2.0 was used to train and evaluate the neural networks.

Summary of Datasets	
Total Artifacts	755
Potential Links	22346
Valid Links	9.46%
Invalid Links	90.54%

DATASETS USED IN THE EVALUATION

System	Total Artifacts	Invalid Links	Valid Links	Artifact Types *
Albergate	72	882	53 (5.67%)	UC, CC
eAnci	194	7091	554 (7.24%)	UC, CC
eTour	174	6363	365 (5.43%)	UC, CC
iTrust	80	1493	58 (3.74%)	UC, CC
MODIS	68	890	41 (4.40%)	HighR, LowR
SMOS	167	3512	1044 (22.91%)	UC, CC

* HighR = High-level Requirements, LowR = Low-level Requirements, UC = Use Cases, CC = Code Classes

Preprocessing

Document text was first tokenized then cleaned by removing stopwords, whitespace, punctuation, and purely numeric tokens. A Porter Stemmer was applied to all remaining terms. Once cleaned, we created *metadocuments* by concatenating all possible (order invariant) pairs of documents such that the documents in a pair belong to different sets of artifacts. Each metadocument contains data from a unique pair of potentially related documents in the system, and was labeled as either a valid link between two related documents or an invalid link between two unrelated ones as specified in each dataset's oracle file. As shown in the Datasets Table, class imbalance was present in all datasets. To mitigate negative effects of class imbalance on model performance, each dataset was balanced using the Synthetic Minority Over-sampling Technique (SMOTE), resulting in an equal number of valid and invalid links within each dataset.

Training and Validation

A shallow Tensorflow model was implemented to classify unlabeled metadocuments as valid or invalid. The model's first two layers perform text vectorization and transform the vectors using word embeddings. These vectors are then pooled before being passed into a 16 node dense layer followed by the output layer. We performed 50 trials of 10 fold cross validation with 15 epochs per fold on each dataset. Shuffled stratification was used to sample the dataset in each fold to minimize selection bias.

Results

System	Loss	Accuracy	Recall	Precision	F1Score
Albergate	0.28	0.95	0.91	0.98	0.94
eAnci	0.11	0.96	0.94	0.98	0.96
eTour	0.13	0.96	0.93	0.99	0.96
iTrust	0.17	0.97	0.95	0.99	0.97
MODIS	0.29	0.93	0.94	0.92	0.93
SMOS	0.33	0.87	0.75	0.98	0.85

Conclusions

Based on the results, it is clear that word embeddings used within a shallow network can successfully perform metadocument classification for the systems under study. Note that here we report recall, precision, and F1 score in addition to accuracy. This is because accuracy can be artificially inflated in imbalanced data. For example, in a dataset of 100 samples with 1 “false” label, a machine that always predicts “true” has high accuracy, but it's low recall illustrates the machines impracticality. These results act as a proof on concept, and first step towards creating a classifier which is useful outside of a purely research context.

Future Work

Our future work will focus on improving this model by further optimizing its hyperparameters and evaluating the model's generalizability. Initially, we plan to perform a series of experiments to identify the minimum data requirements using supportive techniques such as Active Learning and cross training. Furthermore, we will investigate how deepening the model’s architecture impacts performance in terms of our results metrics.

References

Please refer to submitted paper.

Implementing a Video Game Artificial Intelligence with Deep Reinforcement Learning

Matthew Cox
Faculty Mentor: Gilliean Lee
Lander University

Abstract

With recent advancements in artificial intelligence (A.I.), it is necessary to understand research and libraries related to Deep Learning to properly showcase methods and applications of Deep Learning with the latest technologies. Of these methods, we used Deep Reinforcement Learning and Deep Q-Networks with the libraries TensorFlow 2.1, OpenAI Gym, and TF-Agents to create an A.I. that performs actions to learn from its interactions and maximize reward without being explicitly taught about its environment. After setting up the environment and the data preparation pipeline, we tested the impact of different algorithms—DQN, Double DQN, and Dueling DQN—as well as changes to the algorithm's hyperparameters on the performance of the A.I. as it plays the Atari game Space Invaders and discovered Double DQN performed the best over 200,000 training steps with an average score of around 400 and a high score around 1000. Changes to optimizer hyperparameters had no significant impact.

Introduction

Artificial intelligence has always been a popular research topic for both academic and commercial use in the realms of robotics and game theory. Many different technologies and methods have been developed, and one of the most influential of these methods is Deep Learning, a subfield that machine learning that uses artificial neural networks to simulate learning by the human brain. By using a method of Deep Learning called Deep Reinforcement Learning that uses Deep Neural Networks with Q-Learning, a model-free algorithm, it is possible to create a functional A.I. to play video games with more rudimentary elements such Space Invaders on the Atari without human supervision. Reinforcement Learning by itself is a Machine Learning algorithm that focuses on maximizing cumulative rewards and minimizing penalties within an indefinite environment. Deep Reinforcement Learning only recently gained traction with the use of research from the company DeepMind, and with this research we were able to develop an A.I. that can play Atari games using a Deep Convolutional Neural Network. The A.I. uses a simulated Atari environment wrapped within the OpenAI Gym toolkit.

Deep Reinforcement Learning Development

Optimizing Rewards

With Reinforcement Learning, the program's agent makes observations of its environment and will either receive rewards or penalties for theses actions. Any action that helps the agent accomplish its objective will reward it, but any action that offers with no benefit or impedes the agent's progress will penalize it. When a game has a score, the score will be used as the agent's rewards. The algorithm that determines the agent's actions is called a policy, and the optimal policy to maximize rewards is unknown at the beginning. The A.I. must use a Q-Learning algorithm to determine the highest Q-value—which is the summation of its rewards from a set of possible actions the agent can take. However, Q-Learning by itself is inefficient, as it must retain all possible game states, possibly resulting in a larger matrix that is expensive to traverse. Another problem of Q-Learning by itself is that it choses random actions based on its current state, which is impractical for large neural networks. A combination of Deep Learning and Q-Learning along with a replay buffer, or experience replay, is used to solve these problems. The replay buffer will store data discovered from previous experiences in a large table. In our case, the replay buffer stores up to 1 million previous experiences.

Implementing Deep Q-Learning

Due to the sheer complexity of saving every state of the game and randomly iterating through possible actions based only on latest experiences, only relevant states the A.I. has learned should be used and saved. The A.I. accomplishes this by using a Deep Q-Network that will approximate a Q-Value for each possible action for a state and save data as [state, action, reward, next_state]. The Q-Learning algorithm to approximate the Q-value is as shown below.

$$Q(s,a) = r + \gamma \max Q(s',a')$$

s:state, a:action, r:reward, γ:discount factor, s':next state, a':next action

There are three main variants of a Deep Q-Network that the A.I. uses based on techniques developed by DeepMind to measure each variant's effect on the overall performance of the A.I.: DQN, Double DQN, and Dueling DQN. It is normally very time-consuming to implement and debug these algorithms from scratch, so we decided to use the open-source TF-Agents library that already provides DQN and Double DQN for use with TensorFlow and OpenAI Gym. Dueling DQN was not provided and so we had to develop that on our own as such it lacked the tools provided by TF-Agents.

Agent Training and Testing

To properly train the A.I., it is necessary to set up the environment and import the applicable libraries and toolkits. Deep Learning is a computational expensive process, so received a Google Cloud research grant to use their cloud Machine Learning servers instead of a home computer. We set up the environment in Jupyter Notebook using a Google Cloud GPU and installed the libraries. To start training the A.I., we first had to create the Deep Q-Network and apply the Deep Q-Learning algorithm and replay buffer along with a driver and observer to collect experiences from a given policy and display the results with a logger. The replay buffer would progressively record the data from previous experiences and use them to train the A.I. in further iterations. We then programmed the main training loop that trains the agent based on the given policy. During training, we made changes to the algorithm's hyperparameters — learning rate, decay rate, and momentum—as well as the size of the replay buffer to measure the difference in performance. The results logged were used to determine which Deep Q-Network variant and hyperparameters allowed the agent to achieve the highest score in a limited number of training steps. The overall mechanism of the training environment is shown in Figure 1.

We recorded both the average return and the average episode length of DQN and DDQN. The average return is the average score achieved over time, and the average episode length is how many steps on average were needed to reach a terminal state, in our case losing a life in Space Invaders.

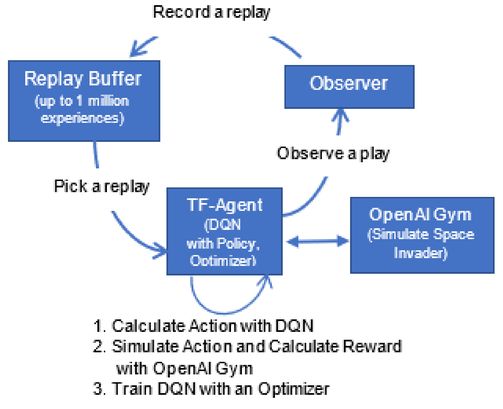


Figure 1. Mechanism of Deep Reinforcement Learning Environment with TF-Agent and OpenAI Gym

Results

We compared the performance impact of DQN, Double DQN, and Dueling DQN along with an experience replay when trained up to 200,000 training steps. When changing hyperparameters, we noticed no significant changes in performance. However, Double DQN performed better than both DQN and Dueling DQN in our tests. The average peaked at around 400 while the high score managed to peak around 1000, which was achieved in about half a minute in one episode. From observation, we noticed the A.I. developed strategies such as hiding behind cover and hitting the purple drone that flies by near the top to gain more points, showing the A.I. was learning how to gain more points in Space Invaders.

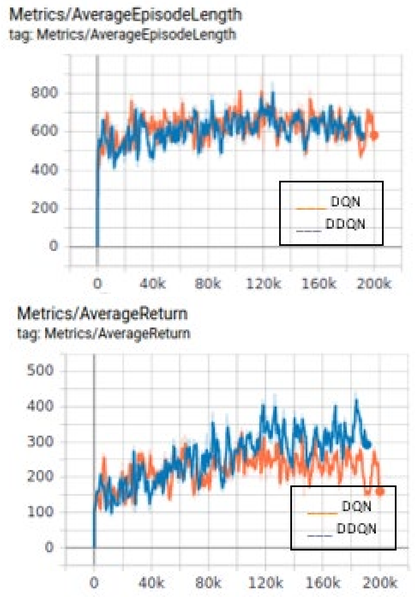


Figure 2. Comparison of Average Episode Length and Return/Reward vs. Training Steps on DQN and DDQN

Conclusion

We successfully developed a system to train the A.I. up to 200,000 training steps. From the results, Double DQN would perform better than DQN and Dueling DQN while changes to hyperparameters had no noticeable impact on performance. It would be possible to apply this system to other simple video game environments such as on Atari, but more advanced algorithms such as Proximal Policy Optimization would be needed for more complicated games like first-person shooters. The A.I. is expected to perform better with more training steps.

Predicting Fall-risk Factors with Artificial Intelligence During Adaptive Locomotion in Humans

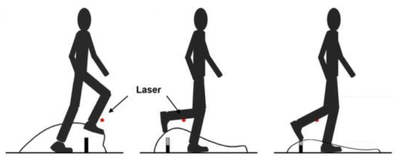
Nouran Alotaibi and Elif Sahin
Mentors: Dr. Gulustan Dogan, Dr. Michel J.H. Heijnen, and Dr. Karl Ricanek
Department of Computer Science
University of North Carolina Wilmington

Abstract

- This work is anchored by a novel locomotion dataset due to the population studied— young adults. This dataset is composed of 88 participants tracked over 150 runs.
- The goal of this study was to analyze the errors in foot placement, foot elevation, and leading factors that resulted in spontaneous contact with a fixed, visible obstacle in young, healthy adults.
- We used machine learning techniques to examine patterns in the kinematic data captured in a controlled laboratory environment for signals of an impending fall.

Dataset

- Eighty-eight undergraduate students (age 20.07 ± 0.7 years,range 18–22 years, 32 males; height 171.7 ± 9.5 cm; weight 72.9 ± 14.3 kg) participated in this study. Subjects were free from any impediments to normal locomotion and had normal or corrected-to-normal vision, as verified by self-report (1).
- Subjects walked at a self-selected pace on an 8-meter walkway, 150 times while stepping over an obstacle in the middle of the walkway.



Obstacle Contact

- Subjects were not told that obstacle contacts were of interest. At least 150 trials were collected per subject.
- Participants in this study were given a daily online survey to complete that recorded frequency and circumstances of slips, trips, and falls in the field.
- The survey data continued over a 16-week period. Sixty-two falls were reported by 38 participants (43%).
- We merged survey data and lab foot trajectory data to create the dataset. Twenty features have been created from the lab foot trajectory study. Faller/non-faller labels in our dataset have been created from the survey study.
- Based on survey data, participants who fell at least once during 16 weeks, were labeled as ‘Faller’. Participants who did not fall at all were labeled as ’Non-Faller’.

Methods

Python programming language was used at every stage of the study. Pandas, numpy, ggplot libraries, matplotlib and seaborn were used for data related operations. In addition, we ran all machine learning calculation experiments with the scikit-learn library on Google Colab. 20% of the data was used for test and 80% was used for training the models. Below are our methodology that we followed:

1) We performed principal component analysis to examine the interrelations among our set of variables as in Figure 1.

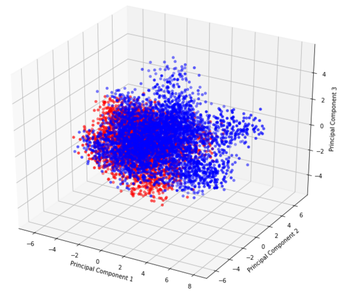


Figure 1: PCA 3D Projection

2) Feature selection has been done to remove irrelevant data and the features that are less important in our dataset.

3) We have used Feature Importance and Correlation Matrix with Heatmap as in Figure 2 and Figure 3.

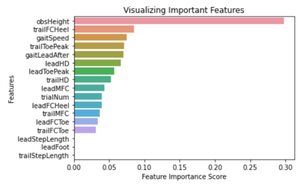


Figure 2: Feature importance

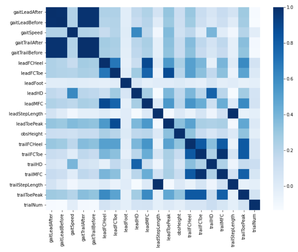


Figure 3: Correlation matrix

4) We removed features that have a correlation value greater than or equal to 0.9, which are: ’gaitTrailBefore’, ’gaitTrailAfter’, ’gaitLeadBefore’,’trailStepLength’, and ’trailMFC.’

5) Hyperparameter tuning has been used to find the optimal hyperparameters for our model parameters.

6) A model is created through the chosen machine learning techniques which are: AdaBoost, Gradient Boosting, Gaussian Naive Bayes, Decision Tree Classifier, KNN, Support Vector Machine (SVM) and Random Forest Classifier.

Results

The results showed us the overall accuracy of fallers and non-fallers as 99.09% (98.69% sensitivity and 99.37% specificity) when the Gradient Boosting algorithm was applied for classification.

Classifiers	Features	Accuracy	Precision	Sensitivity	Specificity	F1-Score	Classification Error
Logistic Regression	5	65.33%	60.00%	42.91%	79.19%	51.09%	34.67%
AdaBoost	15	93.21%	91.72%	91.60%	94.32%	91.66%	6.78%
Gradient Boosting	20	99.09%	99.08%	98.69%	99.37%	98.88%	0.91%
Gaussian Naive Bayes	10	67.95%	60.69%	60.37%	73.15%	60.53%	32.05%
Decision Tree Classifier	5	96.85%	95.47%	96.85%	96.85%	96.16%	3.15%
SVM (polynomial)	20	92.79%	92.36%	90.10%	94.70%	91.22%	7.21%
KNN	15	91.72%	88.86%	91.08%	92.16%	89.95%	8.28%
Random Forest Classifier	10	97.54%	98.65%	95.31%	99.10%	96.95%	2.46%

Conclusion

- Our findings shows that all of the chosen data features have an important role on classifying if a subject is faller or non-faller.
- Gradient Boosting algorithm may identify ”at-risk” individuals. Thus, obstacle crossing behavior can be generalized to falls in the real world for young adults.
- To observe what determines a subject to be a faller based on the given survey data, we calculated each subject’s BMI.
- From the below box plot we have found that the median weight, median height and median BMI for faller are respectively 71.00 (kg), 168.75 (cm), and 23.62 (kg/m2).
- For non-faller, the medians are respectively 75.55, 173.00, and 24.18.
- As a result of the above analysis, the fallers tend to weigh less, they are shorter, and hence their BMI is lower.

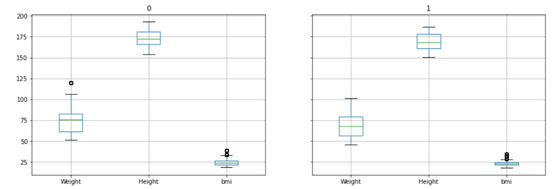


Figure 2: Faller (1) and non-faller (0) participants weight, height and BMI examination illustrated as box plots

Reference

[1] M. Heijnen, H. Johannes, and S. Rietdyk, ‘‘Falls in young adults: Perceived causes and environmental factors assessed with a daily online survey,’’ *Human movement science*, vol. 46, pp. 86–95, 2016.

Developing a Data Anonymizing Software to Make Private Data Available for Analysis

Scott Gangloff, Gilliean Lee
Lander University

Abstract

Data privacy is an important policy that is necessary to keep personal information from unauthorized access. Because of this, data is often kept locked, is available to only certain individuals, or can never be shared even when there are benefits of sharing without personally identifiable information (PII). Obtaining datasets for research is often hard to do. This can be an issue when an organization tries to research on datasets that include PII, and it frequently happens at Lander University's Institutional Research when they research Lander University's student datasets. A group of student developers could not work on dashboards and presentations with the student datasets due to their lack of security clearance. To solve this issue, we developed a tool that reads a data file, anonymizes certain sensitive data fields, and exports a resulting file that can be distributed and shared without worrying about security clearance. This tool, called the 'Anonymizer', allows users to import a CSV (Comma-Separated Values) data file, specify what features should be anonymized, the type of data each feature is, and relations between other features.

Introduction

The Anonymizer was initially created to anonymize Lander University's student data set. However, we decided to make the software more robust and compatible with almost any other dataset. To make this possible, there must be an array of datatype options to choose from. The user should be able to upload a CSV file of their choice, apply their chosen settings, process the data, and receive an anonymized dataset in the location of their choosing. The user should also have the option of saving a template containing the chosen data settings for the anonymized file. If the user ever must anonymize the same dataset again, they would not have to fill in all the settings again.

Development

Dependencies and Reading CSV Files

The Anonymizer utilizes the OpenCSV dependency to operate. OpenCSV is an open-source, third-party plugin that reads CSV files efficiently.

The Anonymize Process

Each column necessary for anonymization is saved into an array as a type of node. Nodes follow a hierarchical structure, with the type "Base Node" as the root. The Base Node has member data that each node must have, such as column index, column name, if it is anonymized, etc. For each different datatype (Name, Number, Key, etc.) there is a representative node that extends the base node. These datatype nodes have different member data that corresponds to what the user filled out in the UI. For example, the "Number Node" has member data for random percent, minimum value, decimal places, etc. The anonymizer navigates through the CSV file and anonymizes a field accordingly if the column index matches the node index. The anonymized value is then saved into a duplicate of the original dataset, and the new dataset is saved into the respective location when finished. Figure 1 shows the workflow of the anonymization process.

Name Anonymization

When a name is to be anonymized, the Anonymizer pulls a new name from a file of over 100,000 names. If the names are to be anonymized on gender, the anonymizer will only pull a name aligned with the same gender as the original name. One problem with the name generation was being limited to only 100,000 names. To combat this, the Anonymizer generates a random full name out of the first, middle, and last name columns in the name file. If the new name is unique, it will use that.

Optimization

Runtime was an important factor to keep in mind while developing the software. Datasets can oftentimes become very large, with hundreds of thousands of rows, and hundreds of columns, especially with datasets used for data science. The student dataset we worked with had 50,000 rows and over 200 columns. The code had to be as efficient as possible, whilst still including all the features. One way we made the code more efficient was by reading the entire uploaded CSV file into an array upon upload and referencing the array, rather than the CSV file itself. Array references are much faster than creating a reference to a CSV file.

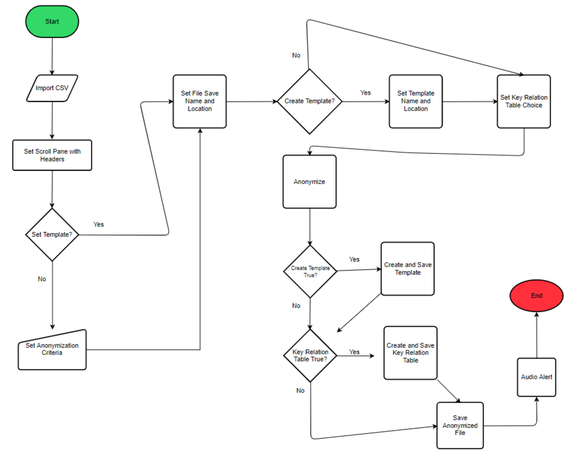


Figure 1. Anonymization Process

User Interface (UI)

Displaying Column Names

The center pane of the UI loads the column names of the CSV file after uploading. For each column name, there is a list of options the user can choose from. The user has the option of uploading a template after uploading their CSV file. This will disable the column names and settings process, as the settings are pre-chosen from the template. Figure 2 shows the UI with filled out settings.

Datatype Options

A list of data types is available for each column name. Upon choosing a data type, the UI dynamically loads criteria related to that option. For example, when choosing the 'Number' datatype, the UI loads fields for maximum value, minimum value, randomization percent, decimal places, etc. The array of choices related to each datatype provides the user with in-depth options for anonymizing. The user may also set which column is the key and whether other columns are consistent on that key. Datatypes consistent on a key will only anonymize once, and then remain consistent whenever the same duplicate key occurs again in the dataset. The datatype options available include Name, Name Composite, Number, Key, and Gender. The Name Composite data type is used for splitting a full name into other columns after anonymizing. Gender datatype is used if the user needs to anonymize names on gender. This means that male and female names will anonymize to another name of the same gender.

File Settings

On the bottom of the UI is a list of fields where the user can specify the name of the anonymized file, where it is saved, and if they would like to make a template or not. The "Translate Table" option was requested by an administrator.

Choosing this option keeps the original data in the anonymized file and mirrors the anonymized version after each row. This is useful for quick comparisons between original data and anonymized data. The "Key Relation Table"

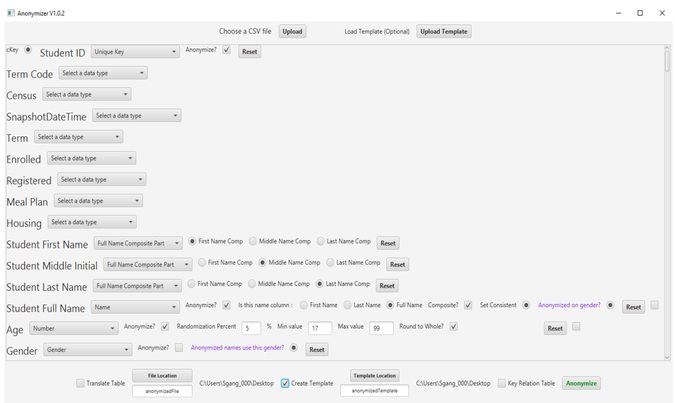


Figure 2. Anonymizer User Interface With Example Settings

Results/Execution

The Anonymizer currently works as intended with few known bugs. The Anonymizer can process a 50,000-row dataset in 3-4 minutes, or about 200 rows per second, on an Intel i7 3.4 GHz core. The application does not use multiple threads and utilizes just one core on a machine. Also, the Anonymizer uses about 250MB of RAM when handling datasets. The necessary RAM increases as the number of rows and columns in the dataset increases, albeit only by a marginal amount. The execution time can vary based on processor speed, overall dataset size, and the number of columns that are being anonymized in the dataset. The application also makes an audio cue when the anonymization process is completed.

Conclusion

We developed a data anonymizing tool that keeps data private while also preserving non-sensitive data to use for analysis. This software tool provides the user with an in-depth User Interface that allows them to specify how their data should be anonymized. Because of this tool, our institution can more easily share sensitive data with other departments, and to students for data science research.