CCSC Consortium for Computing Sciences in Colleges

Southeastern Region

34th Annual Southeastern Conference

Student Research Contest

Extended Abstracts

January 22 and 23, 2021
University of North Carolina – Asheville
Asheville, North Carolina

Table of Contents

Keystroke Injection Detection and Prevention

Emory Lindsey
Georgia College & State University
Faculty Advisor: Dr. Gongbing Hong
Department of Computer Science and Information Systems

Keystroke injection is a hardware attack vector by which a device, usually in the form of a USB thumb drive, is used to type a sequence of keystrokes at relatively fast speeds. During device enumeration, such devices are typically enumerated as HID keyboard devices. Due to the absence of a standard for device authentication, the USB protocol causes a USB bus to blindly trust any connected device regarding its advertised capabilities [1]. Therefore, an injection device can easily trick the host machine into believing that it is simply a USB keyboard. The purpose of this research is to design a simple detection mechanism for keystroke injection attacks using Linux as the target platform. There are many ways to approach this problem and such approaches differ in terms of complexity. The approach taken here is relatively simple; however, as mentioned later, the detection model discussed below was found to be successful in a practical manner.

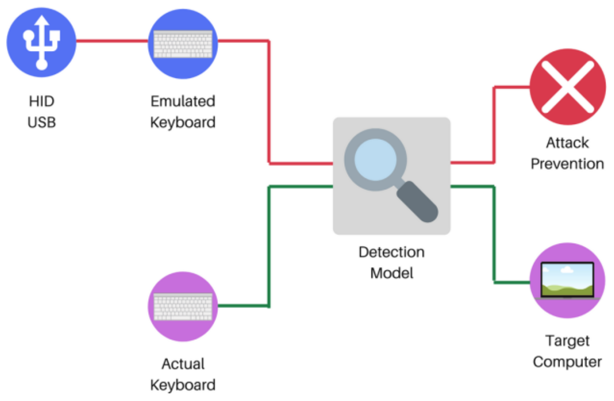


Figure 1: Visual Representation of Detection/Prevention Model

As depicted above in Figure 1, the detection model monitors all connected devices that are classified as keyboard devices. As shown above, any normal activity is simply passed to the target computer and any activity that is flagged as malicious will trigger the attack prevention mechanism, which will disconnect all present keyboard devices. In regards to implementation, a simple kernel module handles the process of capturing keystroke events, calculating the keystroke separation timing, denoted as s, between two subsequent keystrokes, and sending the timing data to user space using the netlink socket API as mentioned by Neuner [1]. The userland application is then responsible for the detection and prevention of injection attacks. Since human beings cannot obtain a value of s less than 80 ms [2], the detection threshold for the models is s = 80 ms. In an effort to reduce false positives, the userland application calculates the average separation timings, s_{avg}, across every four keystroke events. As mentioned before, computers often trust the keyboard input that is received. However, despite this seemingly large vulnerability, successful execution requires some degree of social engineering. Due to this requirement, keystroke injection devices are often programmed to perform injection at inhuman speeds in an effort to reduce the amount of time needed to perform the attack. These two truths lessen the burden of detecting and stopping this form of attack. Of course, in theory, an attacker could modify the keystroke separation to be greater than or equal to 80 ms; however, an attacker usually will not have the luxury of time and so they will most likely opt for a smaller keystroke separation setting such as $s = 10$ ms. The detection model found in this research assumes that the attacker will opt for smaller keystroke separation timings instead of choosing timings greater than or equal to 80 ms.

As a means of testing the effectiveness of the detection model, two injection devices were used. The first was a Raspberry Pi Zero W with a custom OS image (P4wnP1 A.L.O.A.) that supports injection with various values of s. The second, more inconspicuous device, was a Bash Bunny from Hak5. During primary testing with the Raspberry Pi Zero W, a series of tests were run for $s \cong 10$ ms, 20 ms, 30 ms, …, 80 ms. The value of s and the number of allowed keystrokes, denoted as k, were found to be inversely proportional for values of $s \cong 10$ ms, 20 ms, …, 50 ms with the maximum value of $k \cong 30.9$ and the minimum value being $k \cong 11.5$. There was a significant number of false negatives for values of $s \cong 60$ ms, 70 ms, and 80 ms that likely resulted from the decision to take the average value of s for four keystrokes and compare this value to the detection threshold of 80 ms. During secondary testing with the Bash Bunny, the detection model showed significant promise with an average value of $k = 10.8$, with $s \cong 10$ ms. The results of the primary and secondary tests have two significant implications regarding the detection model implemented in this research. Firstly, a value of $k \cong 30.9$ might be alarming, but it is important to remember that this includes both whitespace characters found within the payload and the keystrokes needed to launch a terminal environment within a given OS. In addition, some popular Linux-based reverse shell payloads [3] are themselves longer than 30 characters. Second, an attacker is more likely to choose a Bash Bunny for a social engineering engagement due to its inconspicuous nature. Therefore, due to the promising results of 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

Nick Roy
Faculty Mentor: Dr. Hosam Alamleh
Department of Computer Science
University of North Carolina, Wilmington

In the past decade we have noticed an increase in the number and usage of unmanned aerial vehicles which are commonly known as drones. Today, many civilian drones are equipped with cameras and are used by a wide range of entities such as the government, corporations, individuals, and others. This creates a privacy issue as it became easy to use drones to intrude upon people's privacy and take pictures or record videos. There have been increased efforts to control drones and prevent them from flying in restricted areas. This research proposes a methodology to achieve that.

Many of the advanced drones today are equipped with a Global Positioning Systems (GPS) chip. This is done for the purpose of keeping track of the drone location and have the drone automatically to return to the launch coordinates in case it goes out of the drone remote controller range or loses the signal from the remote controller. This research proposes a technique to ground GPS-enabled drones by broadcasting manipulated GPS signals. A GPS-enabled system uses signals broadcasted by satellites to calculate its location. However, in today's civilian GPS there are no source authentication techniques implemented, which means that current GPS systems do not check if the received broadcasted signals are coming from the satellite or another source. In this research, we analyze the possibility of locally broadcasting manipulated GPS messages for the purpose of grounding GPS-enabled civilian drones and calculate the coordinates where the drone was launched from. This is done by developing a closed-loop algorithm that adjusts the locally broadcasted manipulated GPS signals based on targeted drone direction of motion.

Zero-effort Mobile Device Attendance Application

Aidan Shene, Jake Aldridge, Dr. Hosam Alamleh

University of North Carolina Wilmington Computer Science Department, Congdon Hall 2010

Smartphones can sense several types of signals over the air using different radio frequency technologies (e.g., Wi-Fi, Bluetooth, cellular signals, etc.). Furthermore, smartphones receive broadcast messages from transmitting entities (e.g., network access points, cellular phone towers, etc.) and can measure the received signal strength from these entities. Broadcast messages carry the information needed in case a smartphone chooses to establish communication. We believe that these signals can be utilized in the context of students' classroom attendance tracking, specifically because they could provide an indication of the location of a user's device. The proposed system aims to have students' smartphones in the classroom to generate a "location proof" based on the radio frequency fingerprints scanned by the user's device could then be used to verify users' locations.

In the proposed attendance tracking system, there are two types of accounts: student accounts, and instructor accounts. Both account types scan for different ambient signals and use them to generate a location proof. Then, in the application server, machine learning is used to build a classifier that determines room-level proximity between the students' and the instructor devices. We believe this work is important because attendance tracking is essential in education systems as attendance information is important to optimize the educational process. Moreover, the proposed system helps to preserve class time by automating this important process.

In this research, we explore the possibility of using different radio frequency technologies. This is important because the operating systems of different smartphones allow for varying levels of access to signal information and measurements, which is done for privacy purposes. Due to innate respect for user privacy, and the inherent legal risks, the app will only collect data once the explicit approval of the user has been given by only collect user data once they have pressed a specific button. This type of technology has many other potential uses. Broadening the scope from classroom attendance, this app could be used for meetings, or large events. It also could be expanded to check the number of devices in a specific area. 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 for a device that routinely monitors how many devices are in an area and when changes occur.

Network-Level Cloud Threats Project Proposal

Hannah McSwain
Department of Computer Science
University of North Georgia

Faculty Mentor: Dr. Ahmad Ghafarian

Abstract

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. The primary tool used for testing will be Nmap and Microsoft Baseline Security Analyzer to test AWS port and sniffing vulnerabilities on the network level.

Technology Needed

- Amazon Web Services Cloud Computing account
- AWS Microsoft Windows Server 2016 AMI
- Oracle VM Virtual Box
- MSEdge Windows 10 Virtual Machine
- Kali Linux Virtual Machine
- Nmap
- Microsoft Baseline Security Analyzer

References

1. R, Charanya & Murugaiyan, Aramudhan & Mohan, K. & Sampath, Nithya. (2013). Levels of Security Issues in Cloud Computing. International Journal of Engineering and Technology. 5. 1912-1920.
2. Qaisar, Sara & Khawaja, K.. (2012). Cloud computing: Network/security threats and countermeasures. Interdisciplinary Journal of Contemporary Research in Business. 3. 1323-1329.

Permissioned Authority Based Blockchain Voting with Honestvote

Author: Jacob Neubaum

Mentors: Dr. Ngo and Dr. Liu Cui

Computer Science Department of West Chester University

1 Abstract

This research explores the application of blockchain technology for electronic voting systems. In order to do this, I spent over a year leading the design and engineering of a custom permissioned blockchain for voting, called Honestvote, which was engineered and deployed as a network of interconnected docker nodes. It has currently been used for small elections such as conducting surveys. The project is available at honestvote.io.

2 Introduction

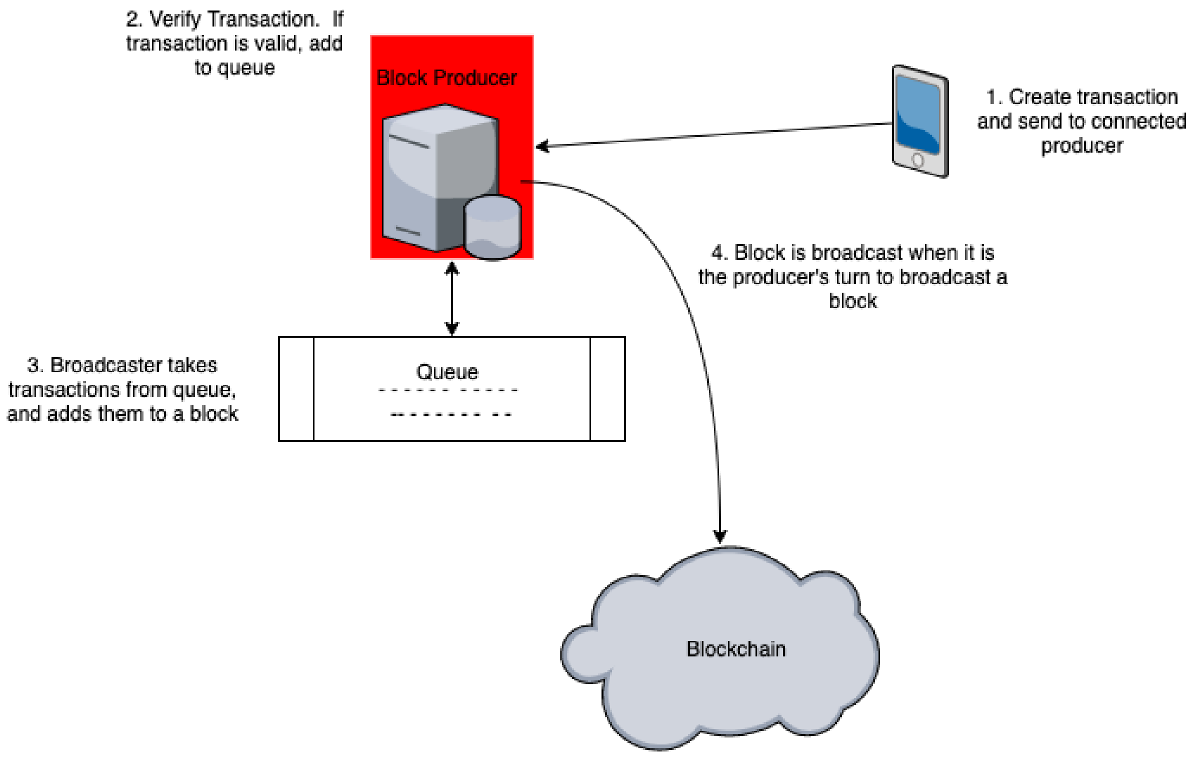
Honestvote rigorously fights against fraud by using blockchain technology as a distributed ledger and relying on a unique implementation of the PoA (Proof of Authority) consensus algorithm to make sure that new votes are not tampered with and that the total history of elections on the network is preserved. Digital signatures ensure that voters are able to trace back their vote at any time to verify that it was not tampered with and made to the correct candidate. Traceable ring signatures are deployed to maintain anonymity of voters and prevent the double spending problem (bad actors cannot vote more than once).

3 How it works

An institution, *e.g. county or church*, that wants to conduct an election on the Honestvote network must join the network as a trusted authority. This requires that they pay a fixed one time fee to the network which deters DDoS attacks and network spam. Secondly, the institution that requests to become a trusted authority must create a custom .txt DNS record to prove ownership of a domain. Finally, a randomly rotating number of existing trusted authorities are selected as auditors and vote on whether or not this new institution should be allowed into the network. If an institution is admitted by the network, they are allowed to declare elections on the Honestvote network, but are also required to perform consensus of election results on the Honestvote network. If they ever lie about election data, past or present, their permissions on the network are revoked and their reputation as a trusted institution is publicly tarnished. The theory is that reputable institutions have a very strong incentive not to tamper with elections because the entire world would instantly be made aware and their reputation would be permanently destroyed.

Election declarations are broadcast to the Honestvote network as transactions by a trusted authority. Declaring an election requires paying a network fee which is dependent on the amount of registrations that are allowed. Each voter requesting registration is issued a registration transaction on the network by the trusted authority that declared the election. Vote transactions are signed with a ring signature to mask the identity of the voter and then they are broadcast to the Honestvote network. Voters are able to trace back their vote on the chain history, but the identity of the voter is never revealed. The total election history is permanently stored on the blockchain and available for all interested parties, including the broader public to validate results. Voters are able to conveniently access the Honestvote blockchain from the web via their mobile device or laptop. Polling places can set up a device for individuals that might not have access to such a device or lack technical skills.

4 Diagram


2. Verify Transaction. If transaction is valid, add to queue
Block Producer
1. Create transaction and send to connected producer
4. Block is broadcast when it is the producer's turn to broadcast a block
3. Broadcaster takes transactions from queue, and adds them to a block
Queue
Blockchain

5 Conclusions

Recently, there has been a lot of controversy over whether or not current voting systems in the U.S. are secure, whether or not U.S. elections have been tampered with, and whether or not ballots are making it to the intended location [1]. Honestvote appears to be a huge improvement over conventional voting systems at ensuring this security. Hopefully, Honestvote, or a system like it, is able to be implemented for U.S. elections in the near future.

References

Lucas Ropek. (2020, November 16). *The 'Most Secure' U.S. Election Was Not Without Problems* Retrieved from https://www.govtech.com/elections/The-Most-Secure-US-Election-Was-Not-Without-Problems.html (2020, December 1).

Abstract: Interactive Fire Spread Simulation

Andrew Droubay[1]
Faculty Mentor: Dr. Anil Shende[1]
Department of Mathematics, Computer Science and Physics
December 2020

Roanoke College

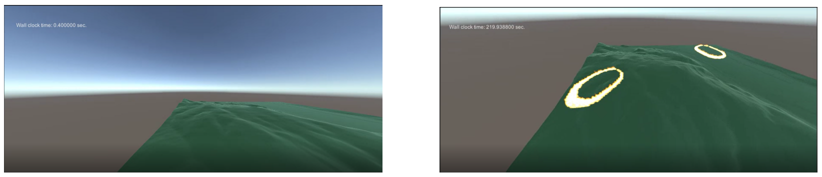
1 Introduction

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 firefighters from across the country deploy to battle fires that react in an unfamiliar fashion. A brush fire in California spreads differently than a forest fire in Virginia. Without experiencing these changes, they can be unprepared for the fire they face, at times resulting in severe injury or loss of life.

However, there is a way to understand this behavior without first-hand experience. The United States Forest Service (USFS) has developed a physics-based mathematical model that predicts fire spread behavior. This model has several levels of complexity, beginning at level-set 1, which gives basic data on the time it takes fire to reach a map cell, and continuing past level-set 4, which includes particle cloud data. In order to better train forest firefighters, this project will use the Unity game engine and the USFS model to create an interactive virtual reality (VR) simulation that will allow users to experience and interact with a scientifically-accurate forest fire. This will give them the ability to cheaply and safely train in diverse environments.

2 Related Work

VR is increasingly used as a powerful method of training in the many fields. It provides a way to practice precision motor control and to develop an understanding of an unfamiliar environment. It is prominetly used in the medical field and is significantly effective at reducing errors in difficult surgeries [1]. Notably, a 1997 study used VR as a training method for firefighters performing search and rescue operations in an unfamiliar environment. The study indicated that the VR training gave a significant improvement in search time when compared to no training [2]. Despite being conducted over 20 years ago, this study shows the promise of VR in relation to fire training.


Wall clock time: 0.400000 sec.
Wall clock time: 219.938800 sec.

(a) Landscape (b) Landscape with Fireline

Fig. 1: Aerial View

3 Our Work

Current work uses the Unity game Engine to read in a file containing information about the landscape, fuels and fire locations. Then, Unity calls the fire models, and reads the output. It then displays this output over time as a fireline. Currently, firefighters can put on a VR headset and controllers to enter and interact with the terrain. Within the simulator, they have a driptorch that they can use to directly start a fire. This interaction triggers Unity to write a new input file for the fire model, and runs the fire models in parallel to the running simulation. Once the model finishes, Unity displays the resultant fireline. More work is in process to enable more dynamic firefighting, but the current framework is promising.

References

1. Ahlberg, G., Enochsson, L., Gallagher, A.G., Hedman, L., Hogman, C., McClusky 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 surgery **193**(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
1017 Academic Way, Tallahassee, FL 32304
mcinnest@cs.fsu.edu, crmills@fsu.edu

1 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.

2 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, and defect prediction among many others [5, 10, 13]. Unfortunately, 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 [8, 14, 2]. Therefore, even if a firm is able to establish traceability for a mission critical system, the effectiveness of that information is potentially only temporary. To address this situation, previous studies have attempted to either completely or partially automate the process of establishing traceability links between system components [4]. 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). Preliminary results of using this technique are promising and show higher recall and precision than its contemporaries [12, 11].

3 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. Together, these datasets represent 755 documents and 22346 potential links, 9.46% of which are valid and 90.54% are invalid. Table 1 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.

3.1 Methodology

Our evaluation consists of a preprocessing, training, and validation phase.

3.1.1 Preprocessing

Document text was first tokenized using NLTK's [3] punkt tokenizer. Then documents were cleaned by removing stopwords using NLTK's built in stopwords list augmented with a collection of Java and C++ keywords. Additionally, whitespace, punctuation, and purely numeric tokens were removed from each document. Finally, a Porter Stemmer [3] was applied to all remaining terms. It is important to note that this stemmer was chosen because it language invariant and our datasets contain both English and Italian words. Each dataset consists of two sets of artifacts of different types (e.g., code classes and use cases), and this same process was applied to documents in both of those sets.

Table 1: DATASETS USED IN THE EVALUATION

System	Invalid Links	Valid Links	Artifact Types†
Albergate	882	53 (5.67%)	UC, CC
eAnci	7091	554 (7.24%)	UC, CC
eTour	6363	365 (5.43%)	UC, CC
iTrust	1493	58 (3.74%)	UC, CC
MODIS	890	41 (4.40%)	HighR, LowR
SMOS	3512	1044 (22.91%)	UC, CC
Total	**20231**	**2115 (9.46%)**	

† HighR = High−level Requirements, LowR = Low−level Requirements, UC = Use Cases, CC = Code Classes

Once the data was 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. Therefore, each *metadocument* contains data from a unique pair of potentially related documents in the system, each of these pairs is then a potential traceability link by construction. Further, the text of each metadocument was randomly shuffled using the Numpy's [7] shuffle method, in order to reduce potential bias from token arrangement when setting the sequence length. Finally, each metadocument in our aggregated dataset 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. This information was then used to evaluate model predictions.

As shown in Table 1, class imbalance was present in all datasets due to the large number of metadocuments (i.e., artifact pairs). As one might expect, for a given system there are far fewer metadaocuments representing valid links than invalid ones. To mitigate negative effects of class imbalance on model performance, each dataset was balanced using Imbalanced-Learn's [9] SMOTE implementation. For our evaluation, we continued synthetic metadocument generation until an equal number of valid and invalid links was reached within each dataset.

3.1.2 Training and Validation

In order to construct a minimally complex proof of concept for the approach, a shallow Tensorflow [1] 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. Based on initial positive results on the SMOS dataset, we kept these hyperparameters and performed 50 trials of 10 fold cross validation with 15 epochs per fold on each dataset. Shuffled stratification via scikit-learn's [6] StratifiedShuffleSplit implementation was used to sample the dataset in each fold to again minimize selection bias.

4 Results

Table 2: EVALUATION 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

5 Conclusions and Future Work

Table 2 shows the results of our evaluation. 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. Therefore, given our precision and recall results, it is clear that word embeddings used within a shallow network can successfully perform metadocument classification for the systems under study. Future work will focus on improving this model by further optimizing its hyperparameters and evaluating the model's generalizability. Moreover, we plan to investigate minimizing the start-up cost associated with employing our model to make it appealing to industry partners with greenfield systems. 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.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Giuliano Antoniol, C Casazza, and Aniello Cimitile. Traceability recovery by modeling programmer behavior. In *Proceedings of the Seventh Working Conference on Reverse Engineering*, pages 240–247. IEEE, 2000.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[4] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.

[5] Elke Bouillon, Patrick Mäder, and Ilka Philippow. A survey on usage scenarios for requirements traceability in practice. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 158–173. Springer, 2013.

[6] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[7] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[8] Laurence James. Automatic requirements specification update processing from a requirements management tool perspective. In *Proceedings of the International Conference and Workshop on Engineering of Computer-Based Systems*, pages 2–9. IEEE, 1997.

[9] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[10] Patrick Mäder and Alexander Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, 20(2):413–441, 2015.

[11] Chris Mills, Javier Escobar-Avila, Aditya Bhattacharya, Grigoriy Kondyukov, Shayok Chakraborty, and Sonia Haiduc. Tracing with less data: Active learning for classification-based traceability link recovery. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 103–113. IEEE, 2019.

[12] Chris Mills, Javier Escobar-Avila, and Sonia Haiduc. Automatic traceability maintenance via machine learning classification. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 369–380. IEEE, 2018.

[13] Patrick Rempel and Patrick Mader. Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering*, 2016.

[14] Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer. Scenarios in system development: current practice. *IEEE Software*, 15(2):34–45, 1998.

Implementing a Video Game Artificial Intelligence with Deep Reinforcement Learning

Matthew Cox
Faculty Mentor: Gilliean Lee
Department of Mathematics and Computing
Lander University
320 Stanley Avenue, Greenwood, SC 29649
matthew.cox1@lander.edu, glee@lander.edu

Introduction

The development of artificial intelligence (A.I) to perform tasks without human supervision for both academic and commercial use has always been a popular research topic and has had many advancements over the years. It is important to understand the different technologies and methods to properly implement and showcase the practical applications of A.I. One such method of implementing A.I. is Deep Learning, which is a subfield of machine learning that uses artificial neural networks to simulate learning by the human brain. For our research, we created an A.I. to play the Atari game Space Invaders by using a method of Deep Learning called Deep Reinforcement Learning that allows the A.I. to learn from its interactions by performing actions in order to maximize rewards without being explicitly taught about its environment. This Deep Reinforcement Learning algorithm utilizes the libraries TensorFlow 2.1, OpenAI Gym, and TF-Agents to train the A.I. with the goals of achieving the highest score possible and showing how the score changes depending on changes to the hyperparameters and number of iterations.

Deep Reinforcement Learning Development

Optimizing Rewards

With Reinforcement Learning, the program's agent makes observations of its environment and takes actions and receives either rewards or penalties for these actions. Any action that helps the agent accomplish its goal rewards it, but any action that offers no benefit or impedes the agent's will penalize it, or in the case of a game with a score, the score will be used as its rewards. The algorithm that determines which action the agent should take is called a policy, and the optimal policy that maximizes its rewards is unknown in 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 to traverse. Another problem 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, to store previous experiences is used to solve these problems.

Implementing Deep Q-Learning

The A.I. should only use relevant states it has learned, instead of randomly iterating through all possible actions and game states, and to accomplish this we implement a Deep Q-Network that approximates a Q-Value for each possible action for a state and save data as state, action, reward, and next state. We implemented three variants of a Deep Q-Network—DQN, Double DQN, and Dueling DQN—to measure each variant's effect on the overall performance on the A.I. using the open-source TF-Agents library with TensorFlow and OpenAI Gym.

Training and Testing the Agent

Deep Learning is a computationally expensive process, so it is inefficient to use a home computer. Instead, we received a Google Cloud Platform research grant to use their cloud Machine Learning servers. We set up the

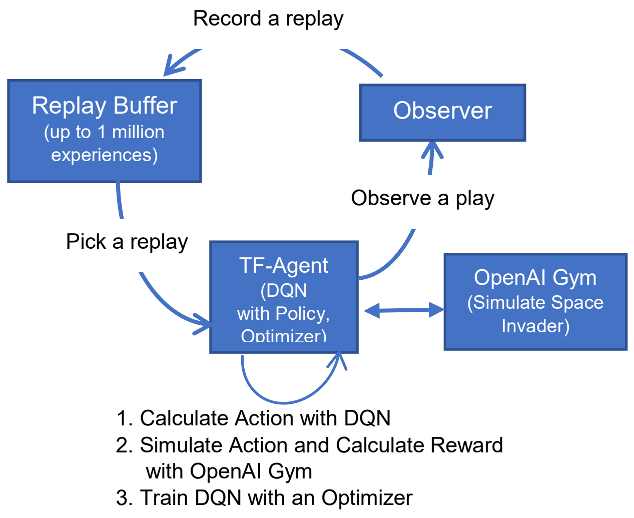

Record a replay
Replay Buffer
(up to 1 million experiences)
Observer
Observe a play
Pick a replay
TF-Agent
(DQN with Policy, Optimizer)
OpenAI Gym
(Simulate Space Invader)
1. Calculate Action with DQN
2. Simulate Action and Calculate Reward with OpenAI Gym
3. Train DQN with an Optimizer

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

environment in Jupyter Notebook using a Google Cloud GPU and installed the libraries. With the environment set up, we applied the Deep Reinforcement Learning algorithm along with a driver and replay buffer to collect experiences and display results with a logger. During the process of learning, we made changes to hyperparameters such as the type of optimizer and its hyperparameters—learning rate, decay rate, and momentum—, the size of the replay buffer, and the neural network's parameters to determine which network variant and hyperparameters allowed the agent to achieve the highest score in the least amount of training time. The overall mechanism of the training environment is shown in Figure 1.

Results

We successfully set up the environment and trained the A.I. using DQN, Double DQN, and Dueling DQN. However, TF-Agents did not fully support Dueling DQN so we implemented Dueling DQN on our own, which took some time. We changed several hyperparameters of the algorithms and trained the A.I. for up to 200,000 steps each and from our data, we saw that most changes to the hyperparameters did not make a significant impact on the performance of the A.I., but it showed that Double DQN managed to perform better than both DQN and Dueling DQN to achieve an average high score of around 400 points, with its highest scores at around 1000 at about half a minute in one episode. We also observed that the AI developed new strategies as well as exploited learned strategies to achieve a higher score, such as hiding behind cover and hitting the purple drone that occasionally flies by near the top to gain extra points, showing the learning process of the AI.

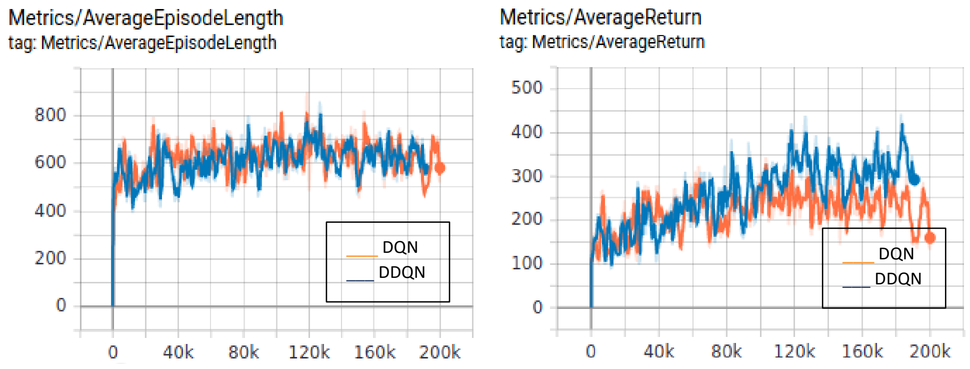


Figure 2. Comparison of Average Episode Length and Return/Reward vs. Training Steps on DQN and DDQN

Conclusions

We can conclude from these results that the A.I. has successfully undergone the process of learning and—at least up to 200,000 steps—Double DQN is a better algorithm to use than DQN and Dueling DQN and changes to hyperparameters make little short-term difference. Different algorithms and an increased number of training steps did not seem to significantly affect the average length of an episode, which is a sequence of states, actions, and rewards that ends with a terminal state such as losing a life. It's expected to perform better with a greater number of training steps. It would be possible to apply the algorithm to most simple video game environments, such as games played on Atari, but more advanced algorithms like Proximal Policy Optimization would be required for more complicated games. Likewise, an A.I. would have to be trained to play these games for a longer period with more resources to perform at a basic level.

References

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd ed.,* O'Reilly Media, Inc.
Thomas, A. (2019, November 15). *Atari Space Invaders and Dueling Q RL in TensorFlow 2.* Adventures in Machine Learning. https://adventuresinmachinelearning.com/atari-space-invaders-dueling-q/
Mnih, V. et. al. (2013). *Playing Atari with Deep Reinforcement Learning.* arXiv:1312.5602v1 [cs.LG]
Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* MIT Press.
TensorFlow. (n.d.). *TensorFlow*. https://www.tensorflow.org/
Brockman, G., et. al. (2016). *OpenAI Gym*. arXiv:1606.01540 [cs.LG]
Guadarrama, S., et al. (2018). *TF-Agents: A library for Reinforcement Learning in TensorFlow.* GitHub repository. https://github.com/tensorflow/agents

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

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

Extended Abstract

Falls are the third leading cause of unintentional injuries for individuals ages 18–35 years according to the CDC; although there are many studies for falls in the elderly population, the causes and circumstances of falls for younger adults are understudied. 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.

In this study, we used machine learning techniques to examine patterns in the kinematic data captured in a controlled laboratory environment for signals of an impending fall. This information is correlated with real-life fall potentials as assessed by a daily, 16-week survey.

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. All subjects signed a consent form approved by the local Institutional Review Board [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. The obstacle height was set at 25% of the subject's leg length. The obstacle was composed of a Masonite board, painted flat black and designed to tip if contacted (similar to a hurdle). Subjects were not told that obstacle contacts were of interest. The starting point was set to ensure the subjects would cross the obstacle with the right leg first (lead foot). 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. In the daily survey, participants received a daily email asking if they slipped, tripped, or fell in the past 24 hours. If a slip, trip, or fall was not reported, the survey was concluded. The survey data continued over a 16-week period (112 days). 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'.

We performed principal component analysis to examine the interrelations among our set of variables. Then, we tested some classification algorithms based on that analysis. After statistical analysis on those selected algorithms, we chose the model that has the best performance.

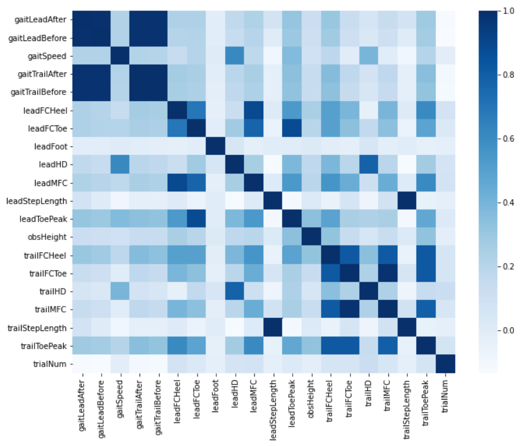


Figure 1: Correlation matrix with heatmap graph.

Feature selection has been done to remove irrelevant data and the features that are less important in our dataset. We have used Feature Importance, and Correlation Matrix with Heatmap as in Figure 1. We removed features that have a correlation value greater than or equal to 0.9, which are: ’gaitTrailBefore’, ’gaitTrailAfter’, ’gaitLeadBefore’,’trailStepLength’, and ’trailMFC.’ Afterwards, hyperparameter tuning has been used to find the optimal hyperparameters for our model parameters.

Finally, 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 .

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.

We have compared the proposed algorithms by their statistical results to identify the best algorithm for our data. We experimented changing the number of features for each of the chosen algorithms. Then, we chose the best number of features for each individual classifier. In Table 1 we wrote these in the ’Features’ column. We applied hyperparameter tuning in an effort to increase performance for all techniques. As a result, an increase across all techniques was observed; the best classification algorithms in terms of their precision are Gradient Boosting, Random Forest Classifier and Decision Tree Classifier.

Table 1: Classification Algorithms Results with Selected Number of Feature and with Hyperparameter Tuning

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%

After we applied machine learning algorithms, our findings shows that all of the chosen data features have an important role on classifying if a subject is faller or non-faller. This machine learning method that predicts falls may identify ”at-risk” individuals. Thus, obstacle crossing behavior can be generalized to falls in the real world for young adults.

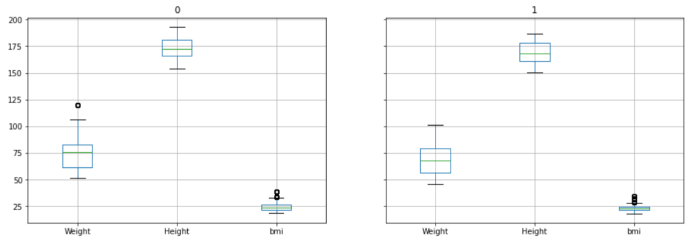


Figure 2: Faller (1) and non-faller (0) participants weight, height and BMI examination illustrated as box plots.

To observe what determines a subject to be a faller based on the given survey data, we calculated each subject’s BMI. We compared the weight, height and BMI for fallers and non-fallers by a box plot as in Figure 2. From the 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.

According to a prior study of retrospective fallers, only 55% of fallers were categorized as overweight or obese” [2]. The results of our survey data shows that only 21% of fallers were categorized as overweight or obese and 45.7% of non-fallers were categorized as overweight or obese. We believe that the participant’s age could factor into those results since in that study they have participants who were around 70 years old. Furthermore, participants were students in a kinesiology major, which may be a more active population given their study focus.

References

[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.

[2] H. Qiu, R. Z. U. Rehman, X. Yu, and S. Xiong, “Application of wearable inertial sensors and a new test battery for distinguishing retrospective fallers from non-fallers among community-dwelling older people,” *Scientific reports*, vol. 8, no. 1, pp. 1–10, 2018.

Development of a Data Anonymizing Tool to Make Private Data Available for Analysis

Scott Gangloff, Gilliean Lee

Department of Mathematics and Computing Lander

University

320 Stanley Avenue, Greenwood, SC 29649

scott.gangloff@lander.edu, glee@lander.edu

Introduction

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.

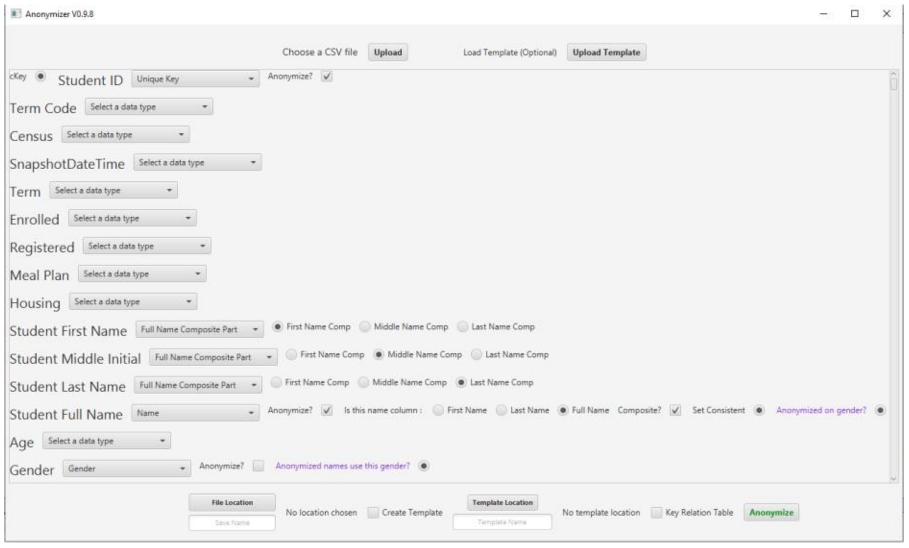
Development

Dependencies

The Anonymizer is written in Java using JavaFX and utilizes the Maven [1] architecture to effectively use dependencies. Dependencies are third party plugins that function as an extra tool that can be used when programming. In our case, the Anonymizer utilized the open-source OpenCSV [2] dependency, which is a tool that can read and write to CSV files faster, and with fewer lines of code.

User Interface (UI)

The UI enables a user to import a selected CSV file, and then immediately populates a scrollable pane containing the headers to columns of the CSV. Each header then has a combo box allowing the user to specify what kind of data is in that column. There are 5 data types, Unique Key, Name, Full Name Composite Part, and Numeric. A column can be marked as Unique Key datatype to designate it as a primary key field, which is useful when there are multiple rows in the dataset that have the same unique key as another row. Name type column can also be anonymized on gender if there is a gender column. For example, a male name will be anonymized into a different male name. This is useful so the anonymized dataset appears as an authentic dataset. Names are chosen from a source of 100,000 unique names organized by gender. The 'Full Name Composite Part' datatype allows a full name to be split into other columns, if applicable.

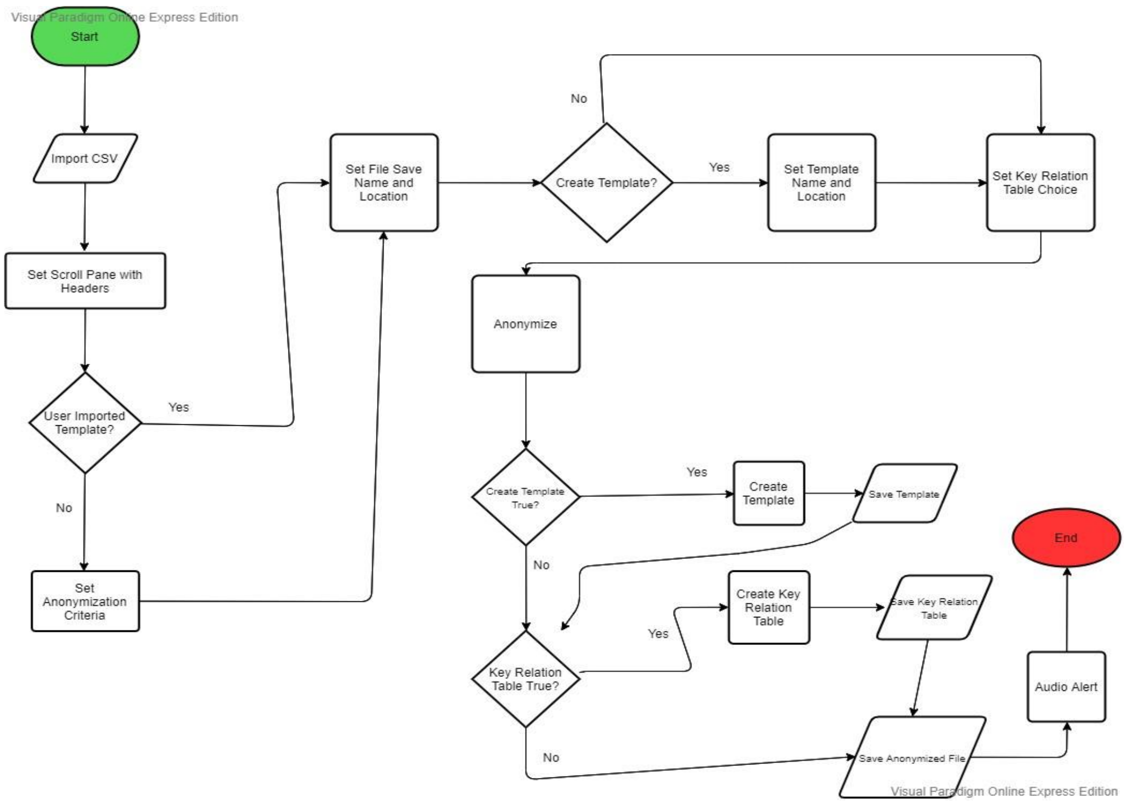


cKey ● Student ID: Unique Key; Anonymize? ✓
Term Code: Select a data type
Census: Select a data type
SnapshotDateTime: Select a data type
Term: Select a data type
Enrolled: Select a data type
Registered: Select a data type
Meal Plan: Select a data type
Housing: Select a data type
Student First Name: Full Name Composite Part; [✓ First Name Comp] Middle Name Comp Last Name Comp
Student Middle Initial: Full Name Composite Part; First Name Comp [✓ Middle Name Comp] Last Name Comp
Student Last Name: Full Name Composite Part; First Name Comp Middle Name Comp [✓ Last Name Comp]
Student Full Name: Name; Anonymize? ✓ Is this name column : First Name Last Name [✓ Full Name] Composite? ✓ Set Consistent ● Anonymized on gender? ●
Age: Select a data type
Gender: Gender; Anonymize? ☐ Anonymized names use this gender? ●
No location chosen ☐ Create Template No template location ☐ Key Relation Table

Figure 1 : Interface with a loaded sample student dataset.

There are extra features other than exporting an anonymized CSV file. Save Template feature enables a user to save and share their anonymization settings, so they do not have to fill out the same criteria again for other CSV files with the same headers.

There is also the option of creating a 'Key Relation Table' as a separate file. This table contains the original keys followed by the alias keys and is useful for administrators to identify what certain keys have been anonymized. Shown in Figure 1 shows a UI after importing a CSV file with anonymization setting partially filled out. Shown in figure 2 is a workflow of the Anonymizer process.


Visual Paradigm Online Express Edition
Start
Import CSV
Set Scroll Pane with Headers
User Imported Template?
Yes
No
Set Anonymization Criteria
Set File Save Name and Location
Create Template?
No
Yes
Set Template Name and Location
Set Key Relation Table Choice
Anonymize
Create Template True?
Yes
Create Template
Save Template
No
Key Relation Table True?
Yes
Create Key Relation Table
Save Key Relation Table
No
Save Anonymized File
Audio Alert
End
Visual Paradigm Online Express Edition

<u>Execution</u>

We needed to keep execution time in consideration when building this software for efficiency. When handling data, sometimes the datasets are very large. In our situation, the student dataset had almost 50,000 rows and over 100 columns, so the program needed to be efficient. Currently, 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

The software we developed anonymizes datasets with PII and it has created opens avenues for research opportunities to use such datasets, especially for machine learning, other data sciences, and the development of applications that deal with sensitive data. In the future, we plan to make the application more robust by supporting more datatypes and giving more options on datatype criteria.

We would like to thank Mr. Matt Braaten at Lander University's Presidential Office for the research opportunity.

References:

[1] "Maven", Apache Maven, https://maven.apache.org/what-is-maven.html

[2] "OpenCSV", OpenCSV, http://opencsv.sourceforge.net/.