Proceedings of the 38th Annual Conference of The Pennsylvania Association of Computer Science and Information Science Educators

March 24th - 25th, 2023

Hosted by:
East Stroudsburg University

PACISE 2023 – Conference Schedule
East Stroudsburg University

Friday, March 24th

Time	Event	
2:00PM – 5:45PM	**Check In – SciTech Bldg 1st Floor Lobby**	
3:00PM – 6:00PM	**Student Presentations – Undergraduate** **SciTech Room 356** **Chairs: Jeyaprakash Chelladurai/Dale Parson**	**Student Presentations – Graduate** **SciTech Room 355** **Chair: Minhaz Chowdhury**
3:00 – 3:30PM	Position Information with Neural Network – Joshua Lewis	Classification of Risk Level for Morbidity and Cesarean Sections Based on Materials Risk Factor – Sara Trabelsi
3:30 – 4:00PM	Comparative Study of Outlier Detection Techniques for Credit Card Fraud – Rachel B. Shirey	Comparative Study of Multivariate Time Series Forecasting Algorithms – Bradley Betts
4:00 – 4:30PM	**Poster Session – SciTech 3rd Floor Lobby**	
4:30 – 5:00PM	MalloT: Scalable and Real-time Malware Traffic Detection for IoT Networks – Ethan Weitkamp	University Chatbot: A journey into cloud-native development – Brian Montecinos-Velazquez
5:00 – 5:30PM	Machine Learning Techniques to Improve Users' Music Listening Experiences – Samantha J. Noggle	Rust: A deep dive – Joseph Cutrone
5:30 – 6:00PM	Combining Unsupervised and Supervised Learning for Credit Card Fraud Detection – Briar Sauble	**N/A**
6:00 – 7:15PM	**Dinner – Dansbury Commons**	
7:30 – 8:30PM	**Keynote Address** **SciTech Bldg Auditorium (First Floor – Room 117)** **Title: Vulnerabilities, Fairy Tales and Persistence** **Speaker: Mr. Douglas McKee** **Welcoming Remarks: President Kenneth Long**	
8:30PM -???	**Programming Contest Setup and Practice** **SciTech Rooms 137/138**	

PACISE 2023 – Conference Schedule
East Stroudsburg University

Saturday, March 25th

Time	Event		
8:30 – 10:00AM	**Check In – SciTech Bldg 1st Floor Lobby**		
8:30 – 10:00AM	**PACISE Executive Board Meeting** **SciTech – 3rd Floor Computer Science Conference Room**		
	Paper Presentations – Faculty **SciTech Room 356** **Chairs: Chad Hogg/Liu Cui**	**Special Presentations** **SciTech Room 355** **Chairs: Dongsheng Che/** **Stephanie Schwartz**	**Programming Contest** **SciTech Rooms 137/138**
8:30 – 9:00AM	**N/A**	**N/A**	**Team Check In**
9:00 – 9:30AM	On the Impossibility of Directly Proving or Disproving the Collatz Conjecture Using a Computer Program – Brandon Packard	Developing Mobile Learning Application based on Adaptive UIs – Kwang Lee	**Contest In-Progress** **9:00AM – 11:30AM**
9:30 – 10:00AM	Modelling a Microsurgical Suture in Unity – Justin Stevens	Cathalla - A fighting game built in Godot – Iain Turner	
10:00 – 10:15AM	**Break**		
10:15 – 10:45AM	Analyzing Game data from the board game Tsuro – Brandon Packard	IoT and Blockchain Integration: Is Blockchain a panacea? – Naresh Adhikari	
10:45 – 11:15AM	Improving students' programming skills by implementing a University-wide programming contest – Liu Cui	3D Printing of Customized Facemasks and PPE – Jack Leidemann	
11:15 – 11:45PM	Object-oriented creative coding for digital art students – Dale Parson	Quaternions In the Use of Three-Dimensional Rotation – Lukas Goodman	
12:00 – 1:00PM	**Lunch – Dansbury Commons**		
1:00 – 2:00PM	**General Meeting – SciTech Auditorium** Discussions Award Presentations Announcements Adjourn		

PACISE 2023 – Conference Schedule
East Stroudsburg University

KEYNOTE SPEAKER

Douglas McKee is a Principal Engineer and the Director of Vulnerability Research for Trellix Advanced Research Center, where he and his team focus on finding new vulnerabilities in both software and hardware.

Douglas has an extensive background in vulnerability research, penetration testing, reverse engineering, malware analysis and forensics, and throughout his career has provided software exploitation training to many audiences, including law enforcement.

Doug is a regular speaker at industry conferences such as DEF CON, Hardware.IO and RSA, and his research is regularly featured in publications with broad readership including Politico, Bleeping Computer, Security Boulevard, Venture Beat, CSO, Politico Morning eHealth, Tech Republic, and Axios. (**Source**: https://www.esu.edu/pacise/)

BEST PAPER AWARDS

Best Faculty Paper:

Modelling a Microsurgical Suture in Unity
Justin Stevens , Chad Hogg , Evan Hanzelman , Brian Smith , Joseph Sassani
Millersville University

Best Graduate Paper:

University Chatbot: *A journey into Cloud-Native Development*
Brian Montecinos-Velazquez, Dominic Pisano, Jared Miller, Harrison Kahl, Nicholas Hines, Tyler Profitt, Dominic Spampinato, and Linh B. Ngo
West Chester University of Pennsylvania, Department of Computer Science

Best Undergrad Paper:

MalIoT: *Scalable and Real-time Malware Traffic Detection for IoT Networks*
Ethan Weitkamp, Yusuke Satani, Adam Omundsen, Jingwen Wang, Peilong Li
Elizabethtown College, Computer Science Department

PROGRAMMING COMPETITION WINNERS

First Place:

Millersville University (Team Name: Millersville-1)

- Trever Bender
- Marshall Feng
- Samantha Noggle

Second Place:

West Chester University (Team Name: Byte Me)

- Mary Bauman
- Nolan Prochnau
- James May

Third Place:

East Stroudsburg University (Team Name: PT-1)

- Jonathan Hillanbrand
- Roshe Ford
- Daniel Sorrentino

IMPROVING STUDENTS' PROGRAMMING SKILLS BY IMPLEMENTING A UNIVERSITY-WIDE PROGRAMMING CONTEST

Jongwook Kim, Si Chen, Liu Cui
West Chester University
{jkim2, schen, lcui}@wcupa.edu

ABSTRACT

Programming is becoming the new literacy; a fundamental skill that everyone should learn. One of the essential skills for programmers is the ability to learn a set of well-defined instructions (i.e., algorithm), and apply them to other problems. To help students improve their programming skills, we provide a university-wide programming contest that is designed by students and for students. We hire student judges and help them create contest problems on an online judging (OJ) system, as well as organize the competition event. In this paper, we explain our programming contest's format and rules that we designed for undergraduate students, and discuss how efficiently and effectively we operate the programming contest by utilizing our OJ system.

1 Introduction

Programming is a fundamental skill required for most areas. Many departments request students to have basic programming skills. For example, physics, chemistry, and mathematics students use programs for data analysis. Business school students write programs to better plan their project. Even music school students use programs to compose symphonies. Many information technology (IT) companies also use programming questions at the first round of a job interview to check applicants' capability of logical thinking and problem-solving skills.

With years of teaching computer science courses at colleges, we have noticed a large gap between programming skills that students can learn from course assignments and those that they need for a technical interview or at work. Many factors contribute to this gap. First, programming requires a wide range of skills and knowledge to solve a problem. Assignments alone are far from enough. Second, problems vary a lot, and the course assignments cannot cover all or even most of them. Third, interview is a timed process, which is different from assignments where students are not constrained with time and can find help in many places. For those reasons, students need more programming experiences with solving non-trivial problems that can stress their capability of finding an optimal solution among infinitely many.

To bridge the gap, we provided continuous extra curriculum activities for students to sharpen their programming skills. One of them is a university-wide programming contest for students who are less-experienced in programming. A unique part of our contest, called West Chester University Programming Contest (WCPC) [1], is that it is designed by students and for students. We (faculty) recruit student judges who create contest problems with test cases, guide them in managing our online judging (OJ) system [2], and help them prepare the competition event. In this paper, we explain our programming contest's format and rules that we designed for undergraduate students, and discuss how efficiently and effectively we operate the programming contests by utilizing an OJ system.

2 Programming Contest Rules and Format

Most computer science departments are running a competitive programming student club for preparing students for nation-wide programming competitions like International Collegiate Programming Contest (ICPC) [3]. Unfortunately, not many students are benefited from the club activities since they mainly target on a small number of high-level students, not the majority of students who need improvements on their programming skills. We adopted many of the ICPC's format and rules but designed our programming contest for the "first time" contestants who are less-experienced in programming.

2.1 Level of Programming Questions

Unlike most programming competitions, we make the level of programming questions range from "very easy" to "easy" comparing to LeetCode [4] problems and require only limited programming experience. The main reason for doing so is that we aim to motivate under-performing students to put more time and effort in programming and encourage them to participate in more forthcoming competitions. We do rank contestants based on (i) the number of problems they solved correctly and (ii) total time they spent on all accepted submissions. However, the purpose of counting the number of correct submissions is not for us (faculty) to judge students' programming skills or rank them based on their performance. Our goal is to encourage many more average students to challenge easy questions and improve their computer science problem solving skills while they prepare for the contests.

For that reason, we allow students to bring any printed material they wish, including books, language reference manuals, code printouts, previous contest problems and solutions, and they are provided with sample questions from previous contests. The only restriction is that external electronic data (e.g., Internet, USB flash drives) is disallowed during a contest. In section 4.1, we discuss that our approach is supported by that the correct submission rate significantly increased between contests in March and October 2022.

2.2 Rewarding Student Judges and Contestants

A unique part of our contests is that they are designed by students and for students. Our student judges create programming problems for contests, validate submissions with using with test cases, manage our OJ system, and prepare the overall competition event under faculty's guidance. The opportunity to serve as a student judge is selectively given to only those who have completed all introductory programming courses and the data structures & algorithms course with a letter grade B+ or above. Their service is rewarded $100 per competition.

Contestants are also rewarded with prizes for their participation. First, they receive $5 amazon gift card for each solved problem regardless of ranking. Second, students can add their programming competition experience on their resume with another line. Free pizza and soft drinks are provided during the competition.

3 Online Judging System

The implementation of university-wide programming contests is a crucial aspect in enhancing students' programming skills and self-efficacy. In order to achieve the maximum benefits of these events, it is imperative to have a well-functioning OJ system. Our OJ system [2] offers students real-time feedback on their submissions, allowing for the tracking of their progress and identification of areas for improvement. Additionally, the OJ system tests the edge cases of students' submissions, ensuring the robustness of their solutions and avoiding costly dead-loops and incorrect implementations.

Having control over the problems and associated settings through the use of an in-house OJ system guarantees fairness and allows for customization to meet the specific needs of the university. This versatile tool can be utilized for a range of programming-related assessments, including practices, competitions, homework assignments, and club activities. The analysis of student submissions and performance through our OJ system provides valuable insights, which can be used to develop targeted interventions to support students' growth and enhance their programming skills.

Moreover, the OJ system offers numerous advantages over traditional paper-based programming competitions. It eliminates manual grading and ensures that all solutions are thoroughly tested using a set of predefined test cases. Students receive real-time feedback on their submissions including the specific test cases that their solution failed, allowing for a better understanding of the requirements and constraints of each problem. The OJ system also provides a level playing field for all students, as it eliminates the possibility of human error in grading and ensures that all solutions are evaluated using the same set of test cases.

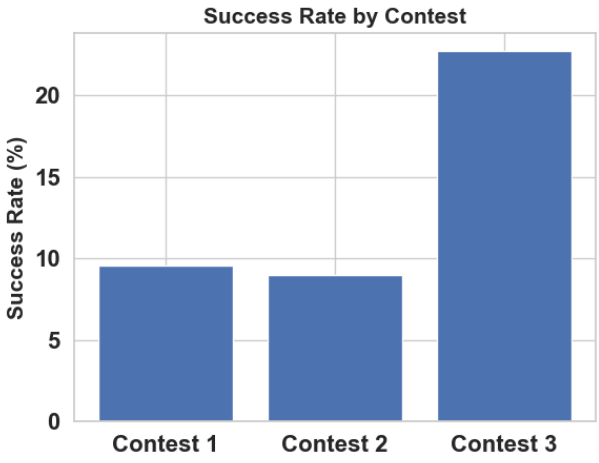

Figure 1: Average Success Submission Rate by Contest.

Using the OJ system, in general, each department can also easily tailor programming competitions and practices to meet their own requirements. In this way, not only computer science students, but all students can practice programming either on campus or through remote log in.

4 Evaluation of Submission Data

We present an evaluation of the student submission data collected from our OJ system. With a total of 507 submissions (from a total of 55 contestants), we aim to gain insights into the performance of the students and the quality of the problems in the contest. To achieve this, we have selected three key metrics – *success rate per contest, error analysis, and success rate per problem* – which provide a comprehensive understanding of the submissions and their outcomes. These metrics will be used to analyze the student submissions and draw conclusions about the effectiveness of the contest in improving the students' programming skills. In the following sections, we will discuss in detail the methodology used to calculate these metrics and the results obtained from the analysis of the student submission data.

4.1 Success Rate Per Contest

This metric evaluates the percentage of students who received correct answers for their submissions. It is calculated using the following equation:

$$Success\ rate = \frac{\#\ of\ correct\ submissions}{Total\ \#\ of\ submissions}$$

The results of our analysis are presented in Figure.1. As depicted in the figure, the average success rate for Contest 1 and Contest 2 was below 10%. This can be attributed to the students' unfamiliarity with the OJ system and its rules. In the early stages, students may have required multiple attempts to become familiar with the system and contest regulations, including the time penalty system which follows the classic ICPC rules. To address this issue, clear contest rules were established prior to the third contest, including a clarification that each rejected submission would result in a 20-minute penalty. As a result, the success rate increased to over 20% in the third contest. It is important to note that the success rate is also affected by the difficulty level of each contest. Although efforts were made to maintain a consistent level of difficulty, the difficulty level of each question will be further analyzed in a subsequent section.

4.2 Error Analysis

This metric assesses the types of errors that students made in their submissions, including *"Wrong Answer," "Compilation Error," and "Runtime Error."* The error analysis is calculated using the following equation:

$$Error\ analysis = \frac{\#of\ submissions\ with\ a\ specific\ error}{Total\ \#\ of\ submissions}$$

As demonstrated in Figure 2, the most frequently observed error in the student submissions was *"Compile Error,"* accounting for 46.15% of all submissions in our data. This error is attributed to either syntax errors in the code or logical errors. Given that the contest targets less-experienced students, it is not surprising that a higher proportion of submissions are affected by compile errors. The second most common error was *"Wrong Answer,"* which constituted 29.98% of all submissions. This error occurs when the student's code is compilable but does not produce the correct results. In some cases, the code may pass certain test cases but fails to provide the correct output for edge cases. The third most common error was *"Runtime Error,"* accounting for 9.27% of submissions. This error indicates that the student's code takes an excessive amount of time to execute or utilizes more resources than allowed. In entry-level contest questions, this error is often the result of infinite loops or logical errors in the code. Note that from Figure 2 we can also learn that the average success rate is 14.60%.

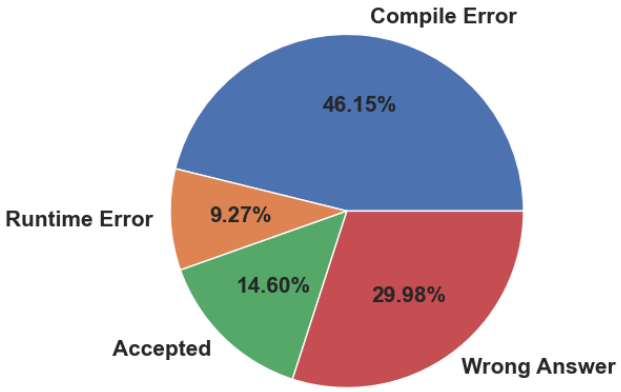

Figure 2: Percentages of Different Error Types.

4.3 Success Rate Per Problem

The success rate for a problem can be calculated as follows:

$$Success\ rate_{problem} = \frac{(\#\ of\ correct\ submissions_{problem})}{(Total\ \#\ of\ submissions_{problem})}$$

The success rate for each problem was utilized to determine the difficulty level of each problem. Table 1 lists the programming problems that we used for each contest.

Contest #	Problem Names
1	Phone Number Team Number
2	Word Composition Excel Column Number Maximize Sum of Array
3	Fudging Finances Passcode Roman Numeral

Table 1: List of Programming Problems.

As shown in Figure 3, the problem with the highest success rate was "Excel Column Number," with a rate exceeding 65%. This problem asked students to translate a column title as it appears in an Excel sheet to its corresponding column number. The problem with the lowest success rate was "Word Composition," with a rate less than 5%. Both of these problems were from Contest 2 and "Word Composition" was the first question of the contest. This may have contributed to the low success rate, as students may have attempted to solve the more challenging first question and not had sufficient time for the easier questions. To address this issue, future programming contests will have the easiest question set as the first question, allowing students to build confidence and prepare for more challenging questions.

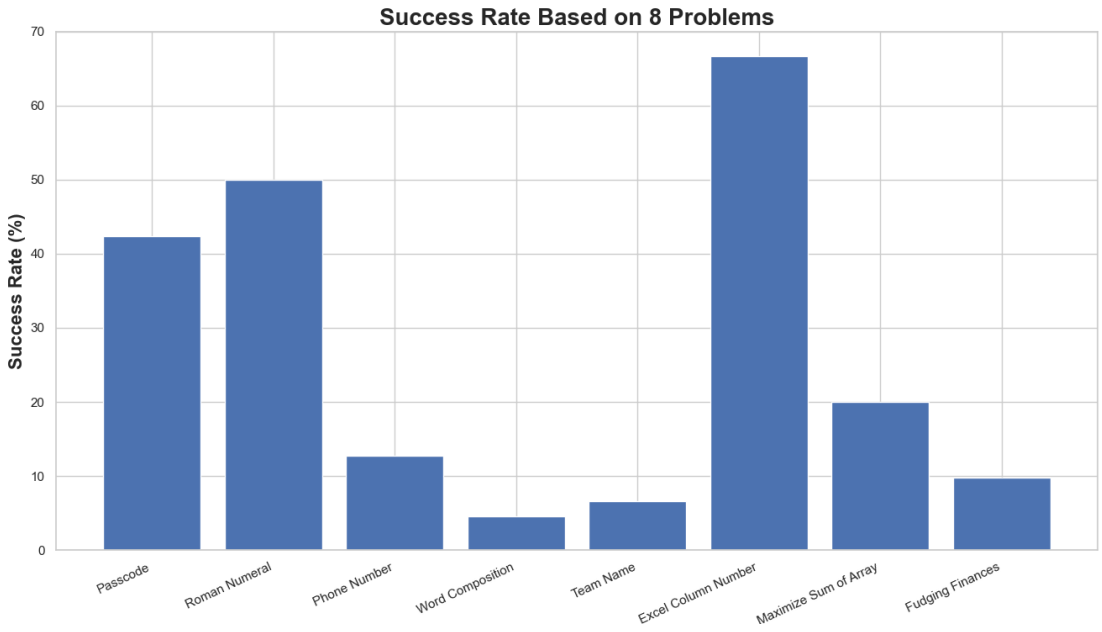

Figure 3: Success Rate of Each Contest Problem.

5 Follow-up Surveys

We conducted two surveys, one for contestants, and another for student judges after the third contest. Four students among seventeen contestants and two student judges out of three participated in the surveys.

5.1 Contestants Survey and Results

Below lists five survey questions we asked to the contestants and the results.

Q1. Please let us know your WCPC experience (Satisfied, Average, Not Satisfied)

1. Overall experience (75% satisfied, 25% average)
2. Coding environment (100% satisfied)
3. Food (100% satisfied)

Q2. How do you evaluate the difficulty of problems (Easy, Medium, Hard)

1. Problem 1 Fudging Finances (50% Easy, 50% Medium)
2. Problem 2 Passcode (75% Easy, 25% Medium)
3. Problem 3 Roman Number (25% Easy, 50% Medium, 25% Hard)

Q3. How frequently do you want to participate in WCPC? (Once a year, Once per semester, Twice per semester)

25% once a year, 75% twice per semester

Q4. What is the best part of WCPC?

> It was a great way to test my java skills and figure out what I need to work on.

> WCPC makes me brush up on my Java. I haven't used Java for many of my classes, and this gives me a good reason to review. There's nothing to lose – free pizza, potential to win gift cards, and I get better at programming.

Q5. Do you have any suggestions/comments for WCPC?

> Please have as many as possible! I wouldn't change anything. Great work by the professors and students that set it up.

5.2 Student Judges Survey and Results

We also performed a survey of student judges using another set of five questions. Below lists the questions with results.

Q1. Please briefly describe your experience in being a student judge.

I enjoyed being a judge. It was fun to draw on my competitive programming experience to create questions and to help facilitate the contests. Also it wasn't too time-consuming or challenging.

Way more time has been required filling out forms and driving places than doing the actual work.

Q2. In your opinion, what is the best part of WCPC?

Writing the problems/input! It's fun work.

Probably its accessibility/beginner-friendliness. I think the WCPC problems are at least approachable for a majority of CS majors, and the growing repository of problems on the OJ system is a great resource for underclassmen looking for more practice and beginners to competitive programming. I have referred a couple people looking to get started with competitive programming to the OJ system. I'd say for most CS students, the problems and the event itself are probably a more accessible alternative to leetcode or other programming contests.

Q3. Do you have any suggestions on how we improve WCPC?

1. In future contests I think there should be 5 minutes at the beginning to formally go over the rules. After the contest I found out that a number of people didn't know there was a time penalty for incorrect submissions, or that they could use an IDE/editor other than the built-in one.
2. It would be ideal if judges who haven't been employed by the college before didn't have to go through the background check process and employment paperwork (reclassify it as stipend, scholarship, etc.?) in total this took a few hours, including
3. separate appointments with HR, fingerprinting, etc. – honestly felt like complete overkill for this position/payment.

Don't make us do hours of paperwork and clearances on top of doing the actual work, it means the pay significantly lower compared to hours worked.

5.3 Discussion

The survey results advocate our idea of recruiting students for programming competition using OJ system. First, all contestants who participated in the survey are satisfied with the coding environment, which reinforces the benefit of using OJ system. Second, contestants want more competition, two per semester, and student judges enjoy the experience of creating problems and test cases for beginner friendliness problems. These findings further support our idea of recruiting students as judges. Two competitions per semester is very difficult if not impossible for faculty members to hold. Student judges make it possible while having fun. Moreover, students find out what they need to work on, which is the whole point of this programming competition.

To implement student judges' suggestions, we will go over the rules with contestants in the competition in March 2023. Further, the OJ system is open for students to try sample problems and problems from previous competitions.

We also see the dissatisfied part from student judges, mainly on the hiring process. We worked with Human Resource and Payroll to facilitate the process, however, most of the clearance and forms are required by the university.

6 Conclusion

Programming is a necessary capability for everyone in the next 10 years, just like reading and writing. Programming skills help students across majors since computer-based solutions such as data analysis and web services are essential to all areas. We designed a university-wide programming contest for students less-experienced in programming. By participating in the contests, students not only exercise problem-solving skills, but also gain self-efficacy, which provides them the confidence and motivation in solving problems in interviews and external competitions, as well as at work. Our online judging system provides an efficient, effective, and fair way of conducting programming contests compared to traditional paper-based competitions. It offers students a platform to hone their programming skills and gain self-efficacy, while providing valuable data for analysis and improvement. The implementation of a university-wide programming contest and the accompanying online judging system hold immense potential for the development of students' programming abilities and the creation of innovative and impactful projects.

References

[1] West Chester Univeristy Programming Contest (WCPC), https://sites.google.com/view/wcpc.

[2] WCPC Online Judge System (OJS), http://wcpc.fun.

[3] International Collegiate Programming Contest (ICPC), https://icpc.global.

[4] Leetcode, https://leetcode.com.

ANALYZING GAME DATA FROM THE BOARD GAME TSURO

Brandon Packard
CU-GAME, Pennsylvania Western University
bpackard@pennwest.edu

ABSTRACT

The board game Tsuro is about building paths on a grid and trying to avoid being removed from that grid. It has fairly simple mechanics that lead to surprisingly complex behavior. In this work, we create an artificial intelligence to play the game, and then run it for 1,000,000 rounds of an 8-player game. We then analyze the data gathered during this process to draw some interesting (and in some cases unexpected) conclusions about the game and the board on which it is played.

1 Introduction

Tsuro: The Game of the Path, often just called Tsuro, is a board game released by Calliope Games, falling in the category of "easy to learn, hard to master". Some work has been done on implementing this game in code [1], but as far as we are aware the game has never been brought into academia, so there is no real related work. To this end we created a game named FailRoad that is aesthetically and programmatically original, but follows the same gameplay mechanics as Tsuro, which allows us to draw conclusions about Tsuro itself. We then created an artificial intelligence (AI) to play the game, and analyzed various aspects of the game using a data analytics approach. In this work, we will share our discoveries.

1.1 Tsuro

Tsuro is a board game referred to as "The game of paths". A video description of the game is available here [2] – with the difference that our version allows players to move themselves off the board at any time, which places larger emphasis on the competency of the AI. We will also discuss the game rules in-depth in this section. In this game for 2-8 players, each tile has 4 sides, each of which has 2 entry/exit points where the tile will connect to other tiles on the board, as shown in Figure 1. Each tile contains 4 different paths connecting the entry/exit points. When considering rotations as the same tile, there are 35 unique tiles.

The game takes place on a 6x6 grid. Initially, the grid starts with no tiles on it, so the only available positions are the entry/exit points on the outside of the grid. Each player picks one of these starting points for their token. Play then starts with the oldest player, and then moves clockwise. Each player gets 3 tiles in their hand. They must play a tile adjacent to their current position, but can play any of their 3 tiles rotated in any of the 4 directions, so a total of 12 possible moves. After each play, they draw a new tile if possible.

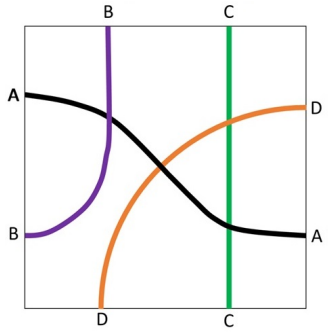

Figure 1: A single tile from the game of Tsuro. Each tile features 4 paths that connect the 8 entry points, as labeled by matching letters.

The astute reader may have noticed that if a player gets eliminated with tiles in their hand, that's less tiles that are available to play on the grid. As such, any player who is eliminated has all tiles from their hand shuffled back into the tile "deck". This does lead to a complication however. It is possible that the next person supposed to place a tile does not have any tiles in their hand, but that there may be tiles in the future after players get eliminated! To handle this, there is a special tile called the Dragon Tile. When a player is supposed to be next to play a tile but no tiles are available, they receive the dragon tile. Play proceeds as normal, except that nobody is allowed to draw tiles from the deck, even if tiles are added back into it, until the player with the dragon tile gets to draw. After that, drawing resumes as before, until such time (if any) that the situation arises again.

Once the player has played a tile, they follow the path from their current position on the tile to the ending position (for example, if a player was at the left position labeled A in Figure 1, they would move to the right position labeled A). If there is another tile adjacent to where they end up, they take the appropriate path on that tile as well. This process repeats until the player ends up in a position without a new path to take. If this position is on the edge of the grid, the player is removed

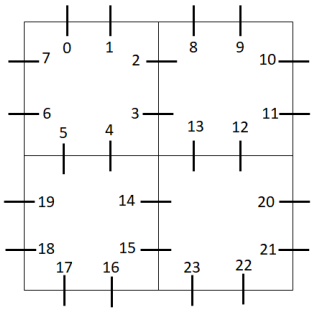

Figure 2: Illustration of how player positions, or the nodes of our graph, are labeled based on a 2x2 tile grid. Each position in the array lists the 8 node values for that board square. Note that there are overlaps between tiles, so the top left tile adds 8 new values, then the top right adds 6, the bottom left adds 6, and the bottom right only adds 4, for a total of 24 nodes.

from the game. If the position is anywhere else, the player remains in the game. If any other players are adjacent to the tile the player has played, they follow the same process. In this way, a single move can affect not only yourself, but also some of the other players as well.

Once only one player is left in the game, they are considered the winner. However, as the players move around the board, there are a couple of unique situations. It is possible that two players will collide when taking their paths. If this happens, both players are removed from the game (if they are the last two players, this results in no winner). You may have noticed that there are 36 spaces on a 6x6 grid, but only 35 tiles. If all 35 tiles are played, any players who manage to survive are considered the winners (in theory, this could lead to as many as 8 players winning, but this would be incredibly unlikely).

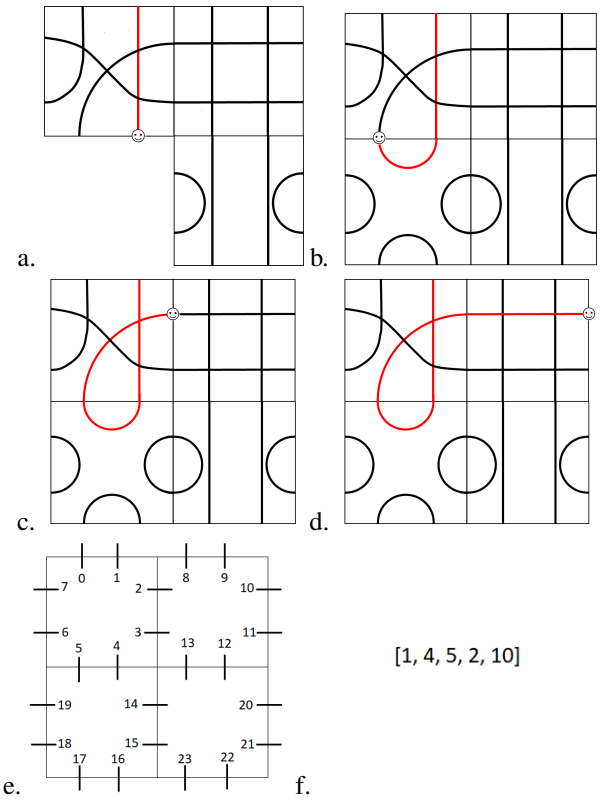

Figure 3: Depiction of player movement when a new tile is placed. The small smiley represents the player, and the red line represents their movement so far. Image *a* is before the player's move, and the images from *b* to *d* show the progression of the player when they play the bottom left tile. Assuming the node values in *e*, the player's path through the graph would be *f*.

1.2 Terms and Definitions

To help avoid confusion, several of the terms we use throughout this paper are listed below along with the definitions we are using for them.

- Tile: a single tile of the Tsuro game
- Tile Grid: The 6x6 game board onto which tiles are placed
- Node: A place on the board in which a player can be, since they are always on the edge of a tile. A visual of this concept can be seen in Figure 2.
- Path: The path that a player moves along in the game, consisting of nodes
- Round: A full play of the game from start to finish
- Ranking/Final Ranking: What position the player finished the round in (1st, 2nd, etc.)
- Passive Share - When a player is on the same tile grid square as at least one other player, and will be moved by one of the other players
- Active Share - When a player is on the same tile grid square as at least one other player, but will be the first of the players sharing the tile to move, thereby controlling where they and the other players will go.

Given the terms above, let's see what happens in a typical move of the game. Figure 3 illustrates what happens when a *tile* is placed on a 2x2 portion of the *tile grid*, as well as the numerical *path* that the player would take through the *nodes*. This demonstrates where the complexity of the game comes in, because one move can put you (and your opponents) in a very different place than you were previously, especially in the later part of the game.

The rest of this work is as follows. Since there is no real related work to our own that we are aware of, we will next be describing our experimental setup in Section 2. After that, we will desribe some Summary Statistics about the game and then some Summary Statistics about the winners in Sections 3 and 4. Next, we will do a more detailed analysis of several areas of the game and gameboard in Section 5. Finally, we end with conclusions and ideas for future work in Section 6.

2 Experimental Setup

The first step in collecting data from the game was to create the game itself. To do this, we needed to maintain both a 2D list (for the tiles) and a graph data structure (for the nodes and paths between them). Further implementation details are outside of the scope of this work, but we would be happy to discuss them with any interested parties. Our creation, named "FailRoad" can be seen in Figure 4. Although both the visuals and code are unique and original, it follows the same game mechanics as Tsuro and can therefore be used to analyze it.

The second step was to create an (AI) that can play the game automatically. The AI didn't need to be perfect, but it needed to have sufficiently complex behavior so that we would be able to derive information about the game from having the AI play it. The details of the Train AI used are discussed in Section 2.1.

Finally, we set up some code to gather data, and had the AI play the game. In this case, we used 8 players, which is the maximum allowed in the game. We chose 8 players in hopes that more players interacting would lead to more interesting conclusions. It is also imperative to note that we ran the game for 1,000,000 rounds, with randomly assigned starting positions and tile-deck orderings, to gather as much data as we could and minimize the chance any results were random.

2.1 Tsuro AI

In order to get meaningful results, we had to have an AI that could play the game relatively well (for, example it's hard to draw conclusions about the game if every AI moves randomly). As such, we created an AI with several prioritized tiers of behavior. Note that we refer to the placement of a tile in the player's hand in any of the 4 rotations as a single move. Intuitively, the AI does the following, in order of priority from most to least:

- If any move results in the player winning the game, take that move.
- Of all possible sets of 2 moves (one move this turn, one move the next turn based on the current board and the player's current hand)....
  - Avoid "sharing" the next tile played with any other players
  - Remove as many players from the game as possible
  - Try to get as close to the center as possible

The detailed process is shown in Algorithm 1. Note that there is still a bit of randomness in the AI, as it breaks ties between equally promising moves randomly. It should also be noted that although this algorithm comes with no guarantees about optimality, but in practice it tends to perform very well based on visual observations (it is certainly much better than we are at playing the game). As such, we believe that this AI is good enough to use to discuss various aspects of the game.

Algorithm 1: Algorithm which is followed in order to choose a move by the Tsuro AI. The main priority is to keep the player alive, but attempts are also made to eliminate the other players when possible.

```
function makeAIMove()
If any move results in the player winning the game, return that move
Set currentSet to all possible sets of 2 moves (one move this turn, one
    move the next turn based on the current board and the player's current
    hand).
if Any moves in currentSet avoid ending up on the same tile as another
    player then
    Set currentSet to a new set with just those moves
end
if Any moves in currentSet remove other players from the game then
    Set currentSet to a new set with only the moves that remove other
        players (maximizing the number of players removed)
end
Set currentSet to a new set with only the moves that get the player as
    close to the center as possible
Return a random move from all of the moves still in currentSet (This
    will be the first tile of the 2 tile set that is selected).
```

3 Summary Statistics

In this section, we are going to discuss some of the general statistics for the game, based on the data we collected and analyzed. First, let's discuss the number of tiles played in each game (across all players). The minimum number of tiles played was 22, and the maximum was 35 (which is the maximum possible in the game). More interesting, however, is that the average number of tiles played per game is 34.022. For the average to be more than 34, many games must go to the maximum of 35 tiles. This is especially true when you consider that there are shorter games like 22 in the mix that would reduce the average. As such, most rounds of the game are very long. This is solid evidence that our AI is quite good at playing the game, as in our experience human-played games (at least at our skill level) rarely get to the max of 35 tiles.

Next, let's look at the number of winners per game. Remember that in theory, the number of winners can go from 0 to 8. The minimum number of winners was 0, and the maximum number of winners was 5. As large numbers of winners would be incredibly unlikely, especially when the AI will try to remove each other from the game, these numbers are reasonable (in fact, we are surprised there were 1078 games with 4 winners and exactly one game with 5 winners!). The average number of winners per game is a very interesting 0.947. The average being less than 1 indicates that there were more games with 0 winners than games with at least one winner. Considering each round is an even match in terms of skill (because all 8 players are being controlled by an identical AI), and that it's not always possible to land on the only empty spot by playing the last tile, this also makes sense.

Next, let's look at the length of the players' paths. Recall that the path is how many nodes on which the player traveled throughout the course of the round. There are 168 unique nodes on a Tsuro board. Since 48 of those are on the outer edge, and only 2 of those can be included in any player's path (the node where the player starts and the node where

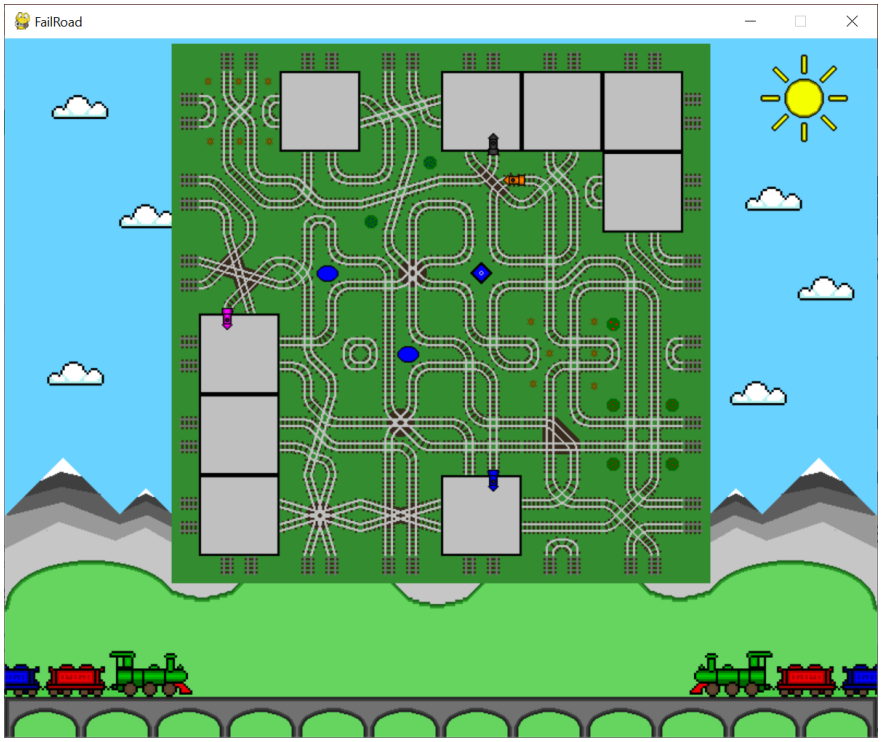

Figure 4: An image of our FailRoad game, with the same mechanics as the game Tsuro but original art and code. Note that the orange train is currently moving along the track in this image, and is not at its final resting position.

the player ends), the theoretical maximum path size would be 122. That being said, with other complications such as it being likely that no tile configuration can hit anywhere near all the nodes, the true maximum is likely much lower. The theoretical minimum for a path is just 2 nodes, if the player is removed from the game by another player before moving away from their starting position. Unsurprisingly, the minimum path in our data was in fact 2. The maximum path in our data was 50 nodes long, which is quite a trek around the board. The average length of a path was 11.238. This seems to indicate that most players take a relatively small path, but in reality there are *many* paths that are only a couple of nodes long, and then many paths that are longer.

Another interesting piece of data is how many players each player was able to remove from the board. The minimum possible would be 0, if the player isn't directly responsible for any opponents going off the board, and the maximum would be 7 if they were directly responsible for all of their opponents going off the board. In our data, the minimum number of removals was 0 for each player, and the maximum was 6 for all players except the last player (who only got 5 at most, likely because they move last and have less opportunities to remove their opponents). The average number of removals per player was 0.595, which means that each player, on average, removes about 1 player per 2 rounds of the game.

Finally, it may be of interest to note that the number of players who didn't get to play a single tile was 0.076 on average. This means that each player was removed from the game before even getting to make a single move around 7.5% of the time. As shown below, the later a player moves the more likely they are to get removed from the game, with it never happening to Player 1 since they move first.

[0.000, 0.020, 0.035, 0.056, 0.082, 0.113, 0.136, 0.160]

In the next section, we will be looking at similar statistics, but only for the winner(s) of the game.

4 Winner Statistics

First, let's take a look at the number of tiles that the winner(s) played. Each winner played a minimum of 3 tiles per round and a maximum of 14 tiles per round, with an average of 5.670 tiles per round. Since the overall number of tiles played each round is just over 34, that means that each winning player plays about 1/6 of the total tiles for that round, on average. This is because as players get removed from the game, the other players will get to play more tiles, assuming the game doesn't end first.

Next, let's examine the path length of the winners. Winner paths varied from a minimum of 3 nodes to a maximum of 42 notes, for an average length of 9.030. This is very interesting, as the average length of players overall was 11.238. These data points together seem to indicate that winning play-

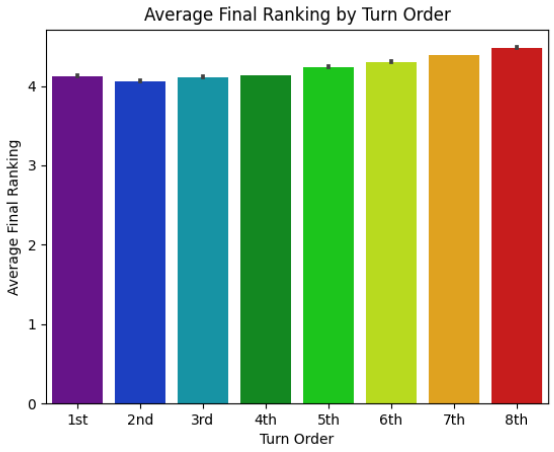

Figure 5: A bar graph showing the average ranking each player ends up with, based on their move order. Closer to 1 means they were closer to winning.

ers make moves that result in less movement. This makes sense since their AI will make them, in general, try to stick to the middle instead of moving to the edges. This means that, when possible, they will place tiles that create shorter paths in order to try to stay in the center of the board.

Finally, let's examine how many removals each winner was able to manage. Winners gained from 0 to 6 kills, with an average of 0.735 removals per winner per round. This is about 23.5% higher than the overall average, showing that winners remove more of their opponents than non-winners.

5 Game Board Analysis

Now that we've seen some summary statistics for both the player population as a whole and the winners, we will perform an in-depth analysis of the game board, to show how various aspects of the board and game affect the winner.

5.1 Turn Order Analysis

First, let us analyze which ranking each player ended up with, based on their turn order. Figure 5 shows a bar chart of the average rank for each player, with values closer to 1 meaning they are closer to winning. Please note that although the values are very close, they are statistically significantly different (with a P-value of 10^{-12} or lower), due to this data being averaged over 1,000,000 rounds.

Perhaps unsurprisingly, players who moved later attained a worse ranking overall. This makes sense, since the later the move the more opportunities there are for other players to remove them from the game or at least exert some control over where they move. That being said, the difference is not a large one, with the average ranking for each player falling between

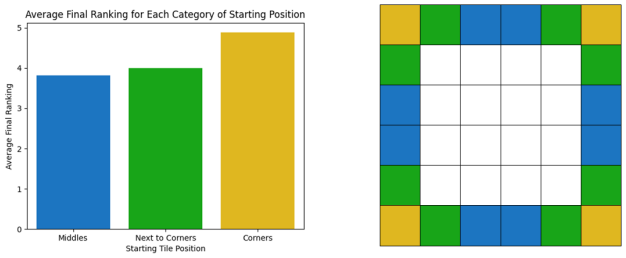

Figure 6: Left: A bar graph showing the average ranking each player ends up with, based on what type of tile in which they start. Right: A color coded visual of where each type of tile is located on the board.

4 and about 4.5. For context, if we were to just repeatedly and randomly assign each player a ranking between 1 and 8, we would expect each player to average out to a ranking of about 4.5. In other words, some players do better than random, but not much - indicating the order in which players move does not make a huge difference on their final rankings!

You may also notice that the best-ranking player is the player that moved second, not the player that moved first. This is a very interesting result, as the first player, on average: followed a longer path, had less active shares (see Section 5.3 for a description of active shares), survived more tiles, played more tiles, and removed more players than the second player. In fact, the only part of our data that could even possibly explain why the second player did better is that they did have more passive shares. As discussed in Section 5.3, more passive shares leads (somewhat unintuitively) to a better ranking on average and in general. As the difference between the first player and the second player in terms of final rank is very small, only 0.06 of a ranking on average, that difference is very likely explained by the extra passive shares.

5.2 Starting Node Analysis

Next, let's look at how players do on average versus their starting tile. There are 20 possible starting tiles. By applying rotational and mirror symmetry to the board, we can see that there are really only 3 types of tiles at which to start - those in the corner, those next to the corner, and those in the middle. Figure 6 provides both the average ranking for each of these 3 types and a color-coded visual of the positions themselves on the board.

As you can see, there is a small (but statistically significant) difference between the next to corner starting tiles and the middle starting tiles, meaning that there's not a big difference between starting in those two types of tiles. However, there is a large gap between those two types and the corner. In fact, starting in a corner puts a player, on average, almost a full ranking behind those who do not start in corners. This shows that corners are the worst place to start in, and given a human-played game where we can pick our starting location, that we should probably avoid starting in any corners. In our opinion, there are two main reasons for this. First, corners are the only

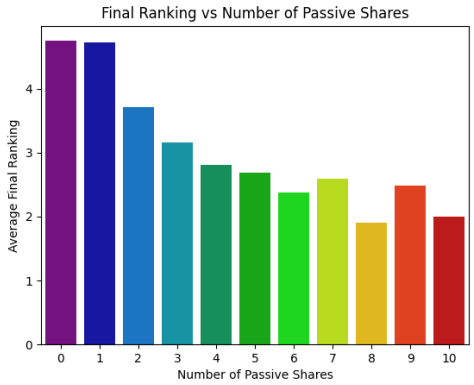

Figure 7: A bar chart of average final ranking versus the number of passive shares the player had during that game.

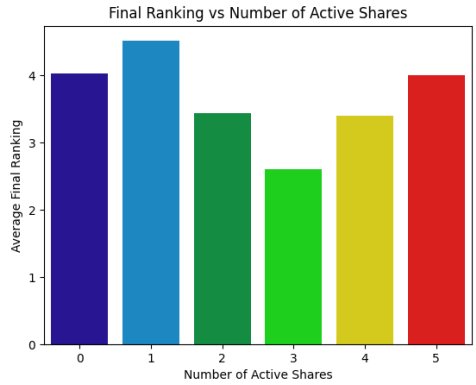

Figure 8: A bar chart of average final ranking versus the number of active shares the player had dring that game.

starting tiles where an opponent that moves before you could potentially knock you out of the game before you even get a chance to move (assuming the player wouldn't also knock themselves out in the first move, which our AIs never would). Second, the middle is intuitively much "safer" than the edges, at least at the start and middle of game, because there are more safe moves and paths in the middle than there are on the edges (at the end of the game, there are so many paths directly off the board that the middle is also very dangerous). These factors together likely explain why starting in a corner is so dangerous, and leads to a worse ranking on average.

5.3 Sharing Tiles Analysis

In Tsuro, we refer to a "share" as a situation in which more than 1 player is "on" the same grid location (but not node - recall each grid location has 8 nodes attached to it). That is to say, when any one of those players makes a move by placing down a tile, all players "sharing" that grid location will move according to the tile. Extending this, an active share is a share where the player we are examining will get the next move out of all players sharing, meaning they will have control to move themselves and those they are sharing with. A passive share is a share where the player we are examining will *not* get the next move, meaning that one of the other players will have the opportunity to move the one we are examining.

Intuitively, active shares are desirable, because you control where you and your opponents end up next, and passive shares are highly undesirable, because you are guaranteed that someone else will move you, possibly off the board or into a worse situation. However, when we crunch the numbers, the results are a bit surprising.

Figures 7 and 8 show the average final ranking for players relative to the number of passive and active shares they had during the game, respectively. First, let's examine the passive shares. The first thing you may notice about this chart is that overall, as the number of passive shares goes up, the average ranking actually *improves*. But if passive shares take away the player's control, how is this possible? To understand this, let's think about how the game evolves. At the start, unless the player is in a corner, they are very safe. As play progresses, players tend to move towards the center (as discussed more in Section 5.4). So in the early and middle game, we have essentially two major possibilities for each player: they are either 1. in the center with many other players, or 2. on the edge with few or no other players. In this situation, scenario 1 is actually much safer! Being in the middle gives more options of where to move, and even when other players move you, it's hard for them to remove you from the game. Conversely, being on the edge is dangerous because it limits your motion and makes it much easier for other players to remove you from the game. As such, players with more passive shares tended to be in the middle where many players clump together, which is usually safer (until the end of the game when everything is dangerous). Players with less passive shares tended to be on the edges and corners more, where it is more dangerous.

There are a couple of other data points that are worth mentioning. First, 7 and 9 break the trend and have a worse rank than those around them. A more in depth analysis would be needed to know the exact reasons for this, but it's likely due to the fact that passive shares sometimes get "chained". That is, Player A is sharing with Player B. Player B makes a move and moves Player A to the same tile as Player C, who now also gets to move Player A. This would likely happen only in the middle to end of the round. As such, it could just be that 7 and 9 tend to be where a chain of passive shares is more likely to remove players from the game, where 8 and 10 are more likely for the player to get control again without being removed from the game. It's also important to note that although the lower number of shares are statistically significant from each other, out of 1,000,000 games * 8 players only 2 instances of 10 passive shares occurred, so more data may be needed to "smooth out" the higher end of the curve and draw more meaningful conclusions about that part of the game.

The other data point worth mentioning is the 0 passive shares data point. To get 0 passive shares, it means the player has never ran into any other player. To never encounter another

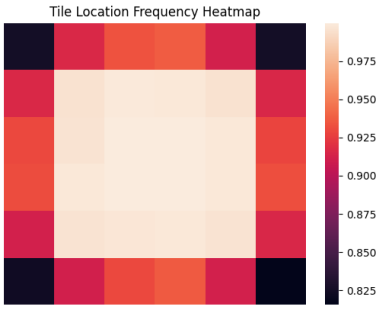

Figure 9: A heatmap of how often a tile was played on a particular grid location, expressed as a proportion of games that a tile was placed there at some point during the game. Lighter colors are more used locations, and darker colors are less used locations.

player, they must never go to the center, since that is where all the players attempt to go. However, since they would also be trying to get to the center, they must be trapped on the edge, and must remove themselves from the game eventually. As our AI never removed itself from the game unless it had no choice, it is likely that other players ended up placing tiles that impeded its progress without ever being on the same tile.

Next, let's look at the final ranking compared to the number of active shares (Figure 8). The first thing we should mention is that 5 active shares was included for completeness but only had 1 instance out of the 1,000,000 games * 8 players and therefore shouldn't be used to draw any conclusions. Four active shares had 1,263 out of 8,000,000 possibilities, so we are fairly confident saying this is an outlier as well, and would need more data to draw any conclusions. Overall, it seems like more active shares should lead to a better ranking, and we can see this is true for 1,2, and 3 active shares.

That just leaves 0 active shares to consider. Zero active shares means that the player will never get a potential chance to remove another player from the game. If you think back to the AI that we use, players will sacrifice getting closer to the center if they can remove another player from the game. It is highly possible that since 0 active shares never get a chance to remove another player, they focus more on getting to the center of the board, where it is safer. On the other hand, those with 1 active share sometimes sacrifice board position for an early removal, thus performing slightly worse in the end.

5.4 Tile Location Analysis

Finally, let us examine what areas of the board were more commonly used. Figure 9 shows a heatmap of where tiles were played on the board over all 1,000,000 rounds. Lighter squares were used more often, and darker squares were used less often. It can be quickly seen that the locations in the middle were used the most, the locations in the corners were used the least, and the other locations on the edge were used somewhere in-between. Each corner was used in about 82.5% of games, and each edge was used for around 91 - 93% of games. The tiles in the middle were used in virtually every game, with the bottom right of the 2x2 square in the very center being used in 999,999 / 1,000,000 games.

This makes sense, since our AI has a rule that tries to move it to the center. However, we believe it's more than just an AI bias. In many board games (such as chess), controlling the center of the board is very important. In Tsuro, the center of the board is the safest place to be for most of the game, because there are the least possibilities to be sent off the board, since that would require being sent along 2-3 tiles instead of just 1 like the edges. As such, most competent AIs would tend towards the center as well.

In addition to this, the 16 squares in the middle have 4 possible directions of movement (without immediately going off the board) - up, right, down, left. The edges only have 3, and the corners only have 2. When you consider that any adjacent tiles in play could prevent you from moving in a direction, being on the edges or in the corners is very dangerous, so most AI (and in our experience, most human players) would tend to migrate towards the center in early game. As such, we believe this heatmap is representative of the game itself, and is a good indicator of where action tends to happen on the board.

6 Conclusions

In conclusion, Tsuro is a game that is easy to play, but difficult to play well. In this work, we first recreated the game, then created an AI to play it. We ran 8 different players with this AI for 1,000,000 rounds of the game to get data to analyze.

We found that most games with our AIs go almost to the maximum possible length. We also found that although there can be multiple winners of a round, there was on average less than 1 winner per round, and that winners play more tiles, take shorter paths, and remove more of their opponents.

We then moved on to a more detailed and general analysis of the game. First, we found that overall, players who move later do slightly worse on average. Next, we found that corners were the worst places to start, and that being closer to the middle was better. After that we saw that increases in both types of shares led to better final rankings. Finally, we saw that corners were used the least often, followed by edges, and that the middle of the board was used in almost every game.

Our hope is that this in-depth analysis of the game situations and board provide a starting point for those looking to create an AI or just improve their own skill. Ideas for future work include analyzing different AIs or different numbers of players (8 was chosen because it seemed like it would give us the most interesting data, but 2 or 3 could offer its own insights). Finally, creating a machine learner, such as a reinforcement learner, and running it against itself and our AI could help lead to even more interesting results in the future, and is the most promising avenue for future work.

References

[1] S. Pyper, Python tsuro, https://github.com/sophia-pyper/python-tsuro, 2018, accessed: 2022-01-24.

[2] Watch_It_Played, Tsuro - how to play, https://www.youtube.com/watch?v=cxpQDmkdEQY, 2013, accessed: 2023-03-15.

RUST: A DEEP DIVE

Joseph Cutrone
West Chester University
joe.contact@pm.me

ABSTRACT

Rust is a fairly new programming language created by a Mozilla employee, Graydon Hoare, in 2006. It was created with the intention to be a safer and more efficient C/C++ alternative. Rust is a very polarizing language with many devout followers and likewise, many people who simply do not like it. Within this independent study, there will be a general summary of Rust's history, a description of the benefits Rust has over C/C++, and lastly a detailed project where a bash shell is written in Rust. This deep dive will hopefully give a reader a better understanding of the language itself and why it could, and should, be implement it into their professional workflows.

1 Introduction

1.1 Rust - What is it?

Rust is a general purpose programming language that has begun to be implemented in place of C/C++. Rust emphasizes type safety, concurrency and performance. Compared to C/C++, Rust enforces memory safety without the need for a garbage collector or reference counting. Rust does this through its borrow checker which tracks objects lifetime and variable scope for every reference during compilation. Rust has become a very popular language in systems programming because of this feature. Rust started as a personal project in 2006 from developer Graydon Hoare. [1] In 2010, Rust was officially announced by Mozilla. Rust drastically evolved over the next few years, introducing: classes, polymorphism, destructors, and traits to provide inheritance. The first stable Rust release (1.0) was pushed in 2015.[2; 1] Due to layoffs, caused by the COVID-19 Pandemic, the Rust Core Team announced plans for a Rust foundation to offset the layoffs affects. In February, 2021 the Rust Foundation was announced by its five backing companies: Amazon Web Services, Huawei, Microsoft, Mozilla, and Google.

1.2 Adoption

The Rust language has been steadily growing with developers and companies since its initial release. This is evident through its consistent winning of the Stack Overflow "most loved programming language" [3] contest from 2016 through 2022. Major software companies have also started developing using rust, including Discord, Amazon, Meta, Microsoft, and Alphabet. Mozilla, a founding party in Rust, uses Rust for their parallel browser engine and "Quantum" which is a collection of projects used to improve Mozilla's "Gecko" engine. There is a "Rust for Linux" patch series, started in 2021, for adding Rust support to the Linux kernel. Discord uses Rust for video encoding and parts of it's backend. Microsoft Azure uses Rust for artificial intelligence and IoT devices. In April, one of the founding companies, Google announced support for Rust within the Android Open Source Project.[1]

Rust Foundation and Governance Teams: Rust is ran by two separate entities: the Rust Foundation and Governance teams. The Rust Foundation is a non profit organization who's purpose is supporting the Rust project legally. The foundation manages trademark and infrastructure. It was founded February 2021 by AWS, Huawei, Microsoft, Mozilla, and Alphabet. The other entity running rust is Governance teams. These teams are responsible for the developement of Rust. The Core Team manages overall direction of the project and other teams leadership. Language team designs new language features, and the Compiler team develops the compiler and optimizes it.

1.3 Rust's Features

Macros: Rust allows for the use of two types of macros: Declarative, and Procedural. Declarative macros use pattern matching to determine expansion. On the other hand, procedural macros use functions compiled before any other components to run and modify the compilers input.[4] These macros are generally more powerful tools than declarative macros but are also much more complex to maintain and create. Procedural macros have three sub sections: Function-like, Attribute, and Derive. Each of which have their own functionalities and declaration styles.

Memory Management and Safety: Rust does not use automated garbage collection. Memory is managed through initialization. Rust provides low overhead deterministic management resources.[5; 2] Values associate on the stack by default and dynamic allocations must be explicit. Rust prevents

dangling pointers and undefined behaviour by verifying validity at compile time. Rust also has immutable and mutable references. Rust is also designed to be memory safe. It doesn't permit null pointers, dangling pointers (as previously mentioned), or data races. Data can be initialized only through a fixed set of forms, which require input to be initialized. Unsafe code can be passed through the restrictions using the unsafe keyword, however this is clearly an intentional bypass of the safeties Rust provides out of the box.[4]

Trait Objects: Rust traits use static dispatch which means the types of all values are known at compile time. However, Rust uses trait objects to provide the user the ability to use duck typing.[4]

C/C++ Integration: Rust has the ability to call code written in C and for C to call Rust's code. This is through the Rust Foreign Function Interface (FFI). Rust allows something similarr with C++ using a library called CXX.[6] Rust's structs can be given a '#[repr(C)]' attribute to force the same layout as C to help with this process.

Cargo: Cargo is a powerful build system and package manager that comes with the Rust language. Cargo can compile, download, distribute, and upload *crates* (packages) from the official registry.[6; 5] By default, these crates come from a web sourced repository. However, users can setup their own repositories using git or local files.

Rustfmt: Rustfmt is a code formatter built into Rust that changes indentation and whitespace in a developers code. This is a feature that helps developers keep a uniform coding standard. [4]

Clippy: Clippy is a built-in linting tool which improves readability and performance of code. Clippy is named after the Microsoft Office feature and it has more than 400 rules. [4]

System Calls: Rust allows a user to call system calls such as:

```
read, write, open, close, exec, wait,
    fork, kill, and exit
```

[7]

1.4 Performance

Rust's creation was in an effort to make a language as efficient and portable as C/C++ while being safer than those languages. [4] Due to the lack of garbage collection, Rust can achieve this goal of being more efficient than other memory-safe languages. Rust has two modes for developing: safe and unsafe. Safe is the default mode in which Rust was intended to be used. In the unsafe mode, the person developing the code is responsible for keeping the code safe.[4] Rust's features are optimized at compile time and cost nothing to a program's runtime. Rust also uses LLVM, meaning all the performance boosts associated with LLVM come with Rust as well. Compared to other languages, Rust has no memory overhead for abstractions. [8]

2 Rust Compared To The Competition

Rust's vast lists of features sets the language apart from its older counterparts. This section will discuss some of Rust's features in depth and how they compare to older programming languages.

Testing: Testing code is usually a tedious task for most other languages, however, Rust has a modern approach to tests. For other languages, a user would have to create a test function that calls the function being tested and compares expected versus actual outputs. In Rust, typing '#[test]' above a function will mark a function for testing.[4] This will allow a user to type 'cargo test' and check if the function is working.[5] Placing these tests outside of the source directory will make them integration tests and have them run without needing access to source code. [8]

Documentation:
Using:

```
'\\\' instead of '\\'
```

will include all functions into the codes documentation. This feature tends to lead to better code documentation. The better a code's documentation, the more digest-able the code will be to other developers. As more developers become familiar with Rust code, there will be more developers that begin to use it for their projects. Converting developers to using Rust is important for the growth and longevity of the language. As seen on stackoverflow, Rust is certainly one of the highest converting languages in recent times![3]

Safety: Another reason Rust could be chosen over C or C++ is its safety through its design. This default level of safety also makes Rust more user friendly than these less modern languages. One major safety change over C and C++ is checking pointers at compile time.[4; 8] During compilation, the Rust compiler ensures that every value is only assigned to one parent at a time. Each parent is then responsible for clearing the resource. The compiler also ensures that pointers cannot outlive parent. Therefore, if a parent, or variable, is dropped then so is the pointer. The compiler will use the borrow-checker to ensure no hanging pointers.[5] All types are also by default immutable in Rust. All of these features help protect against common safety mistakes developers make when coding in C and C++.

Thread Safety: Another modern feature, touted as one of Rusts primary features that sets it apart from older languages, is the built in thread safety. Rust variables know if it is safe to cross thread boundaries and they know if its safe for values to be given to other threads or to be accessed by two threads at the same time.[8] Thread safety eliminates the worry of thread racing and overwriting references. This also applies to references. Pointers can only have one mutable reference or

any number of immutable references to any value of a code.

Lack Of Null Pointers: Rust also has no null pointers. Any reference is guaranteed to not be null and Rust uses this fact to optimize. The compiler forces a user to handle errors and null values.[2] How the user deals with these errors is up to the user, but Rust wont let any of these errors slip through the cracks. Rust also uses '?' notation to catch and call errors if there is one, or it will simply set a value to the completed try.[4; 2] For example:

```
let x = "This String Should Be Split".modify()?;
```

This will run x through the modify function, whatever the user codes that function to do, and if successful x will be set to that value. If unsuccessful the error will be returned.

Rust and C/C++ Together: As mentioned in the previous section, Rust can call C and C++ code without hurting efficiency. This helps programmers implement Rust code into older repositories using C or C++.

Community: The Rust community is very modernized compared to other programming language communities. Rust is thoroughly and precisely documented. Rust forums have many members helping each other solve issues with their code. External packages that can be brought in through the package manager are plentiful.[6] This vast number of user created libraries and packages only add to Rusts versatility! At the time of writing this paper, crates.io reports 23,806,288,169 downloads of user uploaded crates! [6] . On Rust's official site, you can further see how community driven and accessible the language is. The official site links to 3 chat services: Discord, Teams, Zulip. The site also links to two separate forums: one for users to discuss with eachother, and one for the development teams to discuss with Rust users. This large community support system is not found within other languages. It is hard to find even official websites for languages such as C or C++. It is even more difficult to find officially endorsed avenues for developers to discuss the language, its development, and help each other solve coding problems.

3 Learning Rust vs. C

From a students perspective, Rust and C have very different learning curves. The format of C code is much closer to Java than Rust. Java is a language learned by many beginners for its ease of use and accessibility. Therefore, C is much closer to the language many beginner developers and students know. This gives C the upper hand over Rust for beginners who will be much more comfortable writing C code. Rust requires much more time to learn initial coding standards due to always needing to define pointers and types, and many other features that Rust requires a user to use in an effort to keep code safe. This makes the initial learning curve much higher. However, where Rust starts to take the edge over C is when a developer is working on a much more complex problem where errors are more prevalent. Rust's safety features will lessen debugging time for these types of projects.

3.1 Project: Shell in Rust

The final section of this study will show a project done in Rust to create a user bash shell. The differences between Rust and C will be pointed out within each step.

Project

Goals
1. Create a custom shell that can run in interactive mode or batch mode. Interactive mode functions as a typical shell and is basically an infinite loop looking for commands to run, or the break statement 'exit'. Batch mode will intake a file with multiple commands and run them all.
2. Find paths to executable commands. For example the 'ls' will be looked for in /usr/bin.
3. Implement the commands 'exit', 'path', and 'cd'. 'exit' ends the user's shell. 'path' can be used to add different paths to the search path of the shell. 'cd' is a command used to change directories in the shell.
4. Handle errors. Using rusts 'Ok' and 'Err' features, attempt to catch an error in the code that would otherwise break the project.

Project

Goals 1 & 4

```
if args.len() > 1
{
    let fpath = Path::new(&args[1]);
    let mut _file = "";
    let mut fil_con = String::new();
    match File::open(fpath)
    {
        Ok( mut _file) => {let _ = _file.read_to_string(&mut fil_con);},
        Err(_) => {

            exit(0);
        }
    }

    let lines: Vec<&str> = fil_con.split("\n").collect();
    for x in lines
    {
        let mut temp: Vec<&str> = x.split(" ").collect();
        let name = temp[0];
        temp.remove(0);
        run_com(name.to_string(), temp, &mut path);
    }
    run_com("exit".to_string(), [].to_vec(), &mut path);
}
else
{
    loop
    {
        let mut user_string = String::new();
        stdin().read_line(&mut user_string).unwrap();


        let mut temp: Vec<&str> = user_string.trim().split(" ").collect();
        let name = temp[0];
        temp.remove(0);
        run_com(name.to_string(), temp, &mut path);

    }
}
```

The variable 'args' is collected from the 'args' passed with the Rust code when it is ran. If the 'args' vector has more than one item in it then it is trying to be ran in batch mode. This section of the 'if else' statement also covers goal *4* of this project which catches and handles errors using Rusts error catching. Next, if the shell is being ran in batch mode, the code will split the file by every new line character. Each line is then run using the 'run_com'. If the shell code is run without any arguments then it is being ran in interactive mode. The loop infinitely accepts anything entered into the shell. The code parses the input and then sends it to the 'run_com' function. Within this code segment there is use of 'Vectors'. Vectors are a major improvement over C arrays. Vectors are dynamic meaning their memory allocation shrinks and grows with need. In C, an array needs to be a fixed size or can be resized using 'malloc'. C arrays are also *dangerous* because they can lead to many pointer errors and overflow errors. Rust's safety features prevent all of this.

Goal 3

```
fn run_com(name:String, args:Vec<&str>, path:&mut Vec<String>)
{
    match name.as_str()
    {
        "cd" =>
        {
            env::set_current_dir(args[0]).unwrap();
        },
        "path" =>
        {
            path.clear();
            for x in args
            {
                path.push(x.to_string())
            }
        },
        "exit" =>
        {
            exit(0);
        },
        _ =>
        {
            if name.len() > 0 {Command::new(binaryname(name, path)).args(args).spawn().unwrap().wait().unwrap();}

        },
    }
}
```

The 'run_com' function accepts a string from the shell's input. If the user input is one of the three functions 'cd', 'path', or 'exit' then the shell will execute the custom functions created for this project. The projects 'cd' uses a system call to set the current directory to whatever argument the user enters. If this function was written in C, the code could be passed invalid arguments and break the C code. To counter-act this you would need to write a way of catching the error and returning exiting the code without breaking it. In Rust, the error would be caught and wouldn't break the code. The project's path pushes the user entered path to the path vector. If a user enters 'exit' the code calls 'exit(0)', ending the looping interactive shell or the batch file. The final match of the match statement is the catchall. This final match takes any user input and searches if it's name exists among the path files that the user specifies to the shell.

Goal 2

```
fn binaryname(name: String, path:&mut Vec<String>) -> String
{
    if name.len() != 0
    {
        for x in path
        {
            let check_p = format!("{}/{}",x,name);
            if Path::new(&check_p).exists()
            {
                return check_p;
            }
        }

    }
    return String::new();
}
```

The 'binaryname' function is called when an unrecognized command is entered into the shell. The function takes the user input and checks if any functions of a matching name exist in the path files specified for the shell. The code loops through all the paths in the 'path' vector. If a command with a matching name exists in one of the paths, the command is executed.

Code

```rust
use std::fs::File;
use std::io::prelude::*;
use std::env;
use std::path::Path;
use std::process::*;
use std::io::*;

fn main() {
    let args: Vec<_> = env::args().collect();
    let mut path: Vec<String> = ["/bin".to_string(), "/usr/bin".to_string()].to_vec();


    if args.len() > 1
    {
        let fpath = Path::new(&args[1]);
        let mut _file = "";
        let mut fil_con = String::new();
        match File::open(fpath)
        {
            Ok( mut _file) => {let _ = _file.read_to_string(&mut fil_con);},
            Err(_) => {

                exit(0);
            }
        }

        let lines: Vec<&str> = fil_con.split("\n").collect();
        for x in lines
        {
            let mut temp: Vec<&str> = x.split(" ").collect();
            let name = temp[0];
            temp.remove(0);
            run_com(name.to_string(), temp, &mut path);
        }
        run_com("exit".to_string(), [].to_vec(), &mut path);
    }
    else
    {
        loop
        {
            let mut user_string = String::new();
            stdin().read_line(&mut user_string).unwrap();


            let mut temp: Vec<&str> = user_string.trim().split(" ").collect();
            let name = temp[0];
            temp.remove(0);
            run_com(name.to_string(), temp, &mut path);

        }
    }
}
```

```rust
fn run_com(name:String, args:Vec<&str>, path:&mut Vec<String>)
{
    match name.as_str()
    {
        "cd" =>
        {
            env::set_current_dir(args[0]).unwrap();
        },
        "path" =>
        {
            path.clear();
            for x in args
            {
                path.push(x.to_string())
            }
        },
        "exit" =>
        {
            exit(0);
        },
        _ =>
        {
            if name.len() > 0 {Command::new(binaryname(name, path)).args(args).spawn().unwrap().wait().unwrap();}

        },
    }
}

fn binaryname(name: String, path:&mut Vec<String>) -> String
{
    if name.len() != 0
    {
        for x in path
        {
            let check_p = format!("{}/{}",x,name);
            if Path::new(&check_p).exists()
            {
                return check_p;
            }
        }

    }
    return String::new();
}
```

References

[1] W. Editors, Rust (programming language), 2022.

[2] R. Foundation, The RustC Book, 2022.

[3] S. O. Team, Annual Developer Survey, 2022.

[4] R. H. Arpaci-Dusseau, A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces* (Arpaci-Dusseau Books, 2018), 1.00 edition.

[5] R. Foundation, The Cargo Book, 2022.

[6] crates-io Team, crates.io, 2022.

[7] R. Foundation, The Rust Standard Library, 2022.

[8] D. J. Gjengset, Considering Rust, 2020.

POSITION INFORMATION WITH NEURAL NETWORKS

Joshua Lewis
Dr. Girard
jl0189@ship.edu and cdgira@ship.edu
Shippensburg University

ABSTRACT
Spatial information is not commonly used in neural networks. This is because as location values increase in value, this triggers a stronger response from the neural network, when this may not be the desired outcome. This research looks at how the way spatial information is presented affects the accuracy of the neural network. Three different ways are used to represent the information: a grid of all possible positions, an abbreviated model of (x,y) positions, and an (x,y) coordinate. The overall accuracy of the model under each of these conditions will be tested and compared to one another.

KEY WORDS
Neural Network, Spatial Location

1 Introduction

Neural networks are used in many areas due to their ability to generate correct output from data they haven't been trained on. J Steven Perry says in an online tutorial that "Neural networks are particularly well-suited for a class of problems known as pattern recognition [3]." Neural networks excel at pattern recognition because they are very good at eliminating bias that humans can have, and do not get tired like people do; seeing the same pattern over and over does not 'bore' the network[1,3,7,8]. However, neural networks are not often used in regards to spatial information. This paper will explore the idea of using neural networks with spatial information and testing how well it performs.

2 Neural Networks

Neural networks are networks of artificial neurons, see Figure 2.1, that can learn. The learning process is: the network will evaluate the correctness of its output, either by a self-check or with help of a human being, and then, depending on how close or how far the output is from the correct answer, it will adjust itself by either a small or large amount. The process of how the learning occurs will be described more in depth further into the review [1,2,3,6,7,8,9].

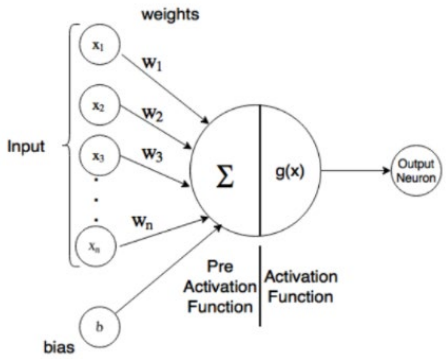

Figure 2.1 - An Artificial Neuron [9]

2.1 How Neural Nets Work

Neural networks can be broken down into two structural parts, neurons and layers, then into two procedural parts, forward propagation and back propagation. Neurons within a neural network are connected to other neurons and have a weight associated with the connection. Each neuron holds information that is transmitted to the other neurons it is connected to, see figure 2.2. The direction information flows depends on if forward or back propagation is occurring [3,6,17].

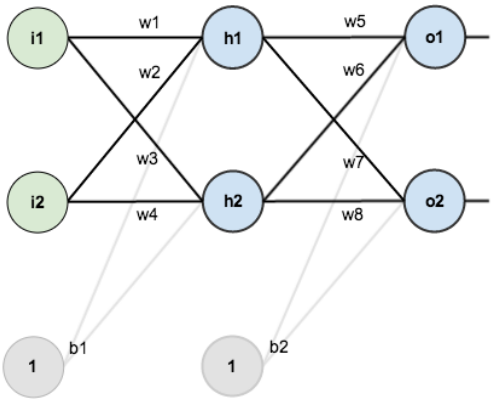

Fig 2.2 Connected Neural Network [11]

Each layer of the neural network can have 1 or more neurons associated with it. There are always at least 3 layers within a neural network: one input layer, one output layer, and one or more hidden layers between the other two layers, see figure 2.2. The input layer will take in numerical values pertaining to the situation being evaluated which will act as the input neuron's value. From there, each neuron in the first hidden layer will sum the value of each input neuron times the weight of the connection. After all of the summation has finished, each neuron's value may have an activator function applied to it, that generates the value for that neuron [2,3,6,9,10,11,12,17].

- Sigmoid

  It maps the input (x axis) to values between 0 and 1.

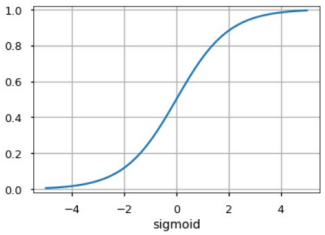

Fig 2.3 Sigmoid function [6]

- Tanh

  It is similar to the sigmoid function butmaps the input to values between -1 and 1.

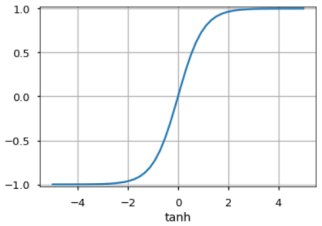

Fig 2.4 tanh function [6]

- Rectified Linear Unit (ReLU)

  It allows only positive values to pass through it. The negative values are mapped to zero.

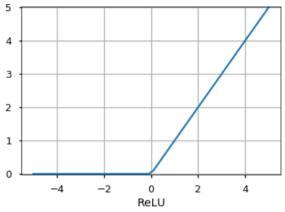

Fig 2.5 ReLU function [6]

An activator function commonly is a function that outputs a value between zero and one, or a value between negative one and positive one. Examples of these activator functions can be seen in figures 2.3 - 2.5 [3,6,9,11,12,17]. Sometimes before the activator function is applied, a bias is added in to change the value by a set amount for that layer [3,11,12,17]. Often a bias is added to try and deal with an offset issue with the input values to improve the learning process of the neural network [5,9,11,12,17]. This cycle will continue until the network has run through all the layers within it [2,3,6,9,10,11,12,17].

This entire process of the network going from the input layer to the output layer is called forward propagation [3,6,9,11,12,17]. When the output layer's neurons have been evaluated, a decision will be made based on the result. This process happens every time a new input is given to the network and everything is computed again [2,10,17].

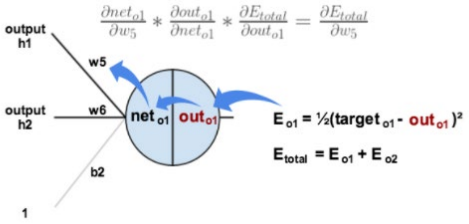

Fig 2.6 Backpropagation example [11]

2.2 Learning Process

The other procedural part of the neural network, called back propagation, is the process of the network improving itself to become more accurate in its responses, see Figure 2.6. This process starts at the output layer and works its way backward through the layers as it self-evaluates. This process begins by evaluating the output, which is done either by the network evaluating itself, or by an outside source, such as a human being, saying how close the network was to the correct answer. Once the network knows how far off it was from the goal, it will calculate the error to know numerically how far off it was from the target, see equation 1 [4,11,14,15].

$$(1)\ E = output - target \quad [11]$$

The connected weights to that output neuron are then adjusted based on the activation function used, the learning rate, and the amount it contributed to the output value using equations 2 and 3. Equations 2 and 3 assume no use of a bias value.

$$(2)\ adjOut = output * (1 - output) \quad [11]$$
$$(3)\ w_1 = w_1 + \eta * in_1 * adjOut * E \quad [11]$$

For equation 2, output is the result produced by the output neuron. Equation 2 assumes we used the sigmoid activation function, so we are using the derivative of that function to reverse the process that got us the output value. For equation 3 it is updating the weight, w_1, for the connection to neuron 1 in the previous layer. The value in_1 is the output value from neuron 1 in the previous layer. The value η is the learning rate value [11]. The learning rate dictates how much of an impact each learning iteration will have [4,11,14,15].

$$(4)\ E_{total} = \sum_m^1 (output_n - target_n) \quad [11]$$

The total error is used when updating the weights between the hidden layer and the input layer or between hidden layers. Normally the error, E, is summed for each output neuron, see equation 4, to compute what the total error is for the network. Then, the weights are adjusted that

connect these layers the same as with the weights between the final hidden layer and the output layer by substituting E_{total} for E in equation 3 [4,11,14,15].

With back propagation, it can be applied every cycle of forward propagation, or it can be applied after multiple forward propagations. In either case, this process is effectively repeated until the desired rate of error has been reached within the network [4,11,14,15].

2.2 Representing Spatial Information

This project focuses on the use of neural networks with regards to position data. There are very limited resources and examples of this in use because this is not an area in which neural networks excel. In a paper by Davidson et al[1], neural networks are used for finding the location of instruments using radio waves to find the position of a particular instrument. One key contribution of their work was passing in map information into the neural network. They noted previous approaches of using map information did not involve a neural network. They used a grid based approach where the map was provided as a binary image and there was an input neuron for each pixel of the image [1]. While accurate, this approach does not scale well.

This issue of encoding location data runs into the problem of representing certain types of values to a neural network. If we simply encode the x location as a single value for the network this is called label encoding. Unfortunately, this type of encoding has the neural network thinking the value of 15 for x it is three times stronger than a value of 5 for x. However, with the grid approach from Davidson et al, they are using a one-hot approach. In a one-hot approach you have each node represent a specific value. For example, for the values 1-3, one node would be the value for 1, another 2, and a third 3. While it means more nodes for the neural network it helps it better understand the meaning of the information. [20, 21]

We propose evaluating three approaches to representing spatial information in a two-dimensional grid. For the first approach, called Grid, we will take the one-hot encoding approach of Davidson et al and provide a neuron for each possible location on the grid. For example, a 10 x 10 grid would require 100 neurons. The second approach will also be a one-hot encoding approach, called Abbreviated. In Abbreviated each coordinate value is broken down by ones, tens, hundreds, etc… Then for each type of value (e.g. ones, tens, …) we will provide 10 neurons representing the values 0 to 9. As such a 10 x 10 grid would require 20 neurons, 10 for X and 10 for Y. For the final approach, called Chord, we will just use label encoding with one neuron for the X value and one for the Y value.

3 Primary Objective

Compare how the input layer model of a neural network affects the accuracy to learn to pass a ball from a stationary location to a moving object. (2.5 person weeks over 1 semester).

The Null hypothesis is there will be no difference in accuracy between the approaches. We hypothesize that the Grid layout will have the best accuracy. Additionally, we hypothesize that the Chord approach will have the worst accuracy.

4 The Scenario

The neural network will learn how to pass a ball to a moving person called the receiver in a 100x100 sized grid. Provided as input to the neural network are the location of the receiver, speed of the receiver, and a unit vector representing the direction of the receiver. The starting position of the receiver will be given to the network via one of the three ways that were spoken about previously, Grid, Abbreviated, or Chord method. There is one input neuron for the speed of the receiver that ranges from 0.25 to 1.0. There is also one input neuron for the X value of the unit directional vector that ranges from -1 to 1. Lastly, there is an input neuron for the Y value of the unit directional vector that ranges from -1 to 1.

Constants that are present in the system are the speed of the ball and the position of the sending player, called the sender. The ball is a constant speed, set to 2, so it is always faster than the receiver, ensuring there is always a solution for the ball reaching the receiver. The sender is always placed in the middle of the grid (49,49).

The Grid method has a total of 10,003 input nodes: 10,000 for the location, 2 for receiver direction, and 1 for receiver speed. The Abbreviated method has a total of 43 input nodes: 40 for the location, 2 for receiver direction, and 1 for receiver speed. Finally, the Chord method has 5 input nodes: 2 for location, 2 for receiver direction, and 1 for receiver speed. There are two output neurons that represent the unit vector direction to kick the ball. The ball is assumed kicked at time 0 and that the speed of the receiver is constant. The accuracy of the network is measured based on how far off the unit vector was from the correct value using equations 5 and 6.

$$(5)\ Err_{total} = \sum(|(X_n - X_{act})| + |(Y_n - Y_{act})|)$$

$$(6)\ Acc = Err_{total}/m$$

In equation 5 X_n is the result for the X unit vector for test run n, same for Y_n with respect to Y. X_{act} and Y_{act} are the correct values. In equation 6 the total error is then divided by the total number of examples tested, represented as m, to give the overall accuracy.

4.1 Examples

Figures 4.1 and 4.2 show an example. Both figures show the same scenarios with different outputs from the network,

figure 4.1 is the correct output and figure 4.2 is an incorrect output. Ten sets of 10,000 of these scenarios are randomly generated for training and testing purposes.

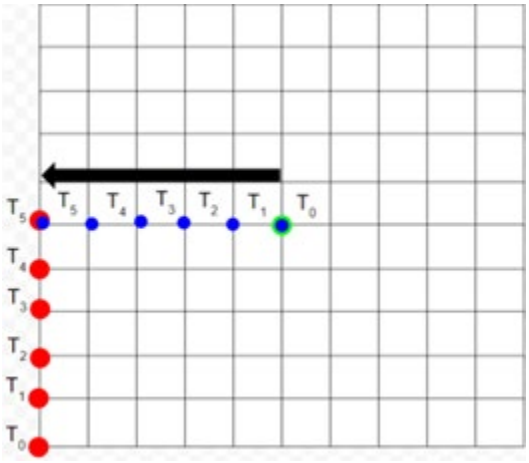

Figure 4.1 Correct example

In Figures 4.1 and 4.2 the red dot represents the receiver, the green dot represents the sender, and the blue dot the ball. In Figure 4.1 the receiver is moving in the direction of 0,-1 from location 0,10 at a speed of 1. The ball is moving from location 5,5 in the direction of -1,0 at a speed of 1. The ball and sender meet at location 0,5.

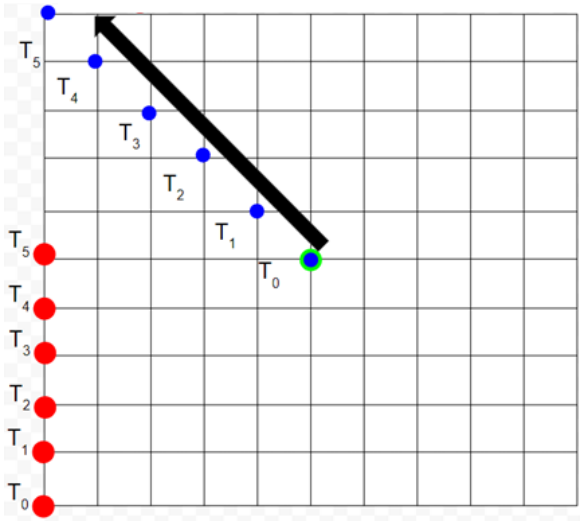

Figure 4.2 Incorrect example

In Figure 4.2 the receiver is again moving in the direction of 0,-1 from location 0,10 at a speed of 1. The ball however is moving at a speed of 1.41 from location 5,5 in the direction of -0.707,-0.707. The ball ends up at location 0,0 while the sender is at location 0,5 and they do not meet.

5 Solution Description

The Java programming language was utilized to implement a neural network with one hidden layer. It did not utilize a bias and used the sigmoid as the activation function. The learning rate for the neural network was set to 0.05.

A dataset was generated by creating 100,000 random scenarios, broken up into 10 sets. When a scenario was generated, it was validated to ensure it had a solution and fit the parameters expected by the neural network. Both the scenario and solution were stored for training and testing.

For each treatment a neural network was generated with random weights and then trained on 9 sets of the dataset and tested on the remaining set. This was repeated 10 times for each treatment to ensure less chance of any treatment getting lucky on the randomly generated weights.

Input Representation		
Grid	Abbrev	Chord
X	X	X

Table 5.1 Block Design

6 Results

Approach	**Avg Error Rise (Y)**	**Std Dev Rise (Y)**
Grid	0.1109	0.01201
Abbreviated	0.08237	0.01099
Chord	0.1518	0.001117

Table 6.1 Accuracy Results - Rise

T-test results:
Grid vs. Abbreviated: 17.44
Grid vs Chord: 33.74
Abb vs Chord: 62.54

Approach	**Avg Error Run (X)**	**Std Dev Run (X)**
Grid	0.1234	0.006026
Abbreviated	0.09924	0.01096
Chord	0.3576	0.001516

Table 6.2 Accuracy Results - Run

T-test results:
Grid vs. Abbreviated: 19.22
Grid vs Chord: 232.34
Abb vs Chord: 375.02

Based on the T-scores we can say with confidence that Abbreviated preformed the best, then Grid, and lastly Chord. It is worth additional testing for why Chord had strong Rise accuracy, but poor Run accuracy. Additionally, while it is positive that Abbreviated performed better than Grid, it is surprising. As such, it is worth additional testing on Grid verses Abbreviated to ensure no issues with training/evaluation.

7 Conclusion

Based on the results of the experiment it can be concluded that the hypothesis for the Chord approach was correct. However, the hypothesis for the Grid and Abbreviated approaches was not. In addition to being more accurate than Grid, the Abbreviated method also relies on much fewer nodes. As the dimensions of the area grow the number of nodes for Grid grows by n^2 for 2D and n^3 for 3D. However, for Abbreviated the growth rate is only $2 * \log_{10}n * 10$ for 2D and $3 * \log_{10}n * 10$ for 3D. Additionally, as the number of objects being tracked increases with Abbreviated the location is easily attached to additional information (e.g. speed and direction) of the object. For Grid you would potentially need a grid per object.

References:

[1] H. Bergkvist, P. Davidsson, and P. Exner, Positioning with Map Matching using Deep Neural Networks, *MobiQuitous 2020 - 17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2020.

[2] 3Blue1Brown, Build an AI from Scratch | Neural Network Java, *YouTube*, 25-Jun-2021. [Online]. Available: https://www.youtube.com/watch?v=cpYRGNT41Go

[3] Perry, Steven J., Create an artificial neural network using the Neuroph Java framework, *IBM Developer*. Jan 8, 2018. [Online]. Available: https://developer.ibm.com/tutorials/cc-artificial-neural-networks-neuroph-machine-learning

[4] J. Deus, Implementing an Artificial Neural Network in Pure Java (No external dependencies)., *Medium*, 01-Sep-2020. [Online]. Available: https://medium.com/coinmonks/implementing-an-artificial-neural-network-in-pure-java-no-external-dependencies-975749a3811

[5] Lukasz Gebel, Why We Need Bias in Neural Networks, Medium, 23-Jan-2022. [Online]. Available: https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98

[6] V. Gupta, Understanding Feedforward Neural Networks, *LearnOpenCV*, 20-Apr-2021. [Online]. Available: https://learnopencv.com/understanding-feedforward-neural-networks/

[7] M. Ibrahim, M. Louie, C. Modarres, and J. Paisley, Global Explanations of Neural Networks, *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 2019.

[8] D. M. F. Izidio, A. P. D. A. Ferreira, and E. N. D. S. Barros, Towards better generalization in WLAN positioning systems with genetic algorithms and neural networks, *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019.

[9] A. Malhotra, Tutorial on Feedforward Neural Network - Part 1, *Medium*, 02-Feb-2018. [Online]. Available: https://medium.com/@akankshamalhotra24/tutorial-on-feedforward-neural-network-part-1-659eeff574c3

[10] S. Mall and S. Chakraverty, Multi Layer Versus Functional Link Single Layer Neural Network for Solving Nonlinear Singular Initial Value Problems, *Proceedings of the Third International Symposium on Women in Computing and Informatics - WCI '15*, 2015.

[11] Mazur, A Step by Step Backpropagation Example, Matt Mazur, 15-Feb-2022. [Online]. Available: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[12] Finn Eggers, Neural networks tutorial: Fully Connected 1 [Java], *YouTube*, 26-May-2017. [Online]. Available: https://www.youtube.com/watch?v=d3OtgsGcMLw

[13] Finn Eggers, Neural networks tutorial: Fully Connected 4 [Java] - Feed Forward Implementation," *YouTube*, 21-Jun-2017. [Online]. Available: https://www.youtube.com/watch?v=6Gz8oVRzoAg

[14] V. Parmar, Blog: Building a simple neural net in Java, *SmallData*. [Online]. Available: https://smalldata.tech/blog/2016/05/03/building-a-simple-neural-net-in-java

[15] M. J. Piovoso and A. J. Owens, Neural network process control, *Proceedings of the conference on Analysis of neural network applications - ANNA '91*, 1991.

[16] R. G. Price and S. D. Goodwin, Synthesizing neural networks for learning in games, *Proceedings of the 2008 Conference on Future Play Research, Play, Share - Future Play '08*, 2008.

[17] S. Sonawane, Understanding and Implementing Neural Networks in Java from Scratch 💻, *Medium*, 17-Jul-2021. [Online]. Available:

https://towardsdatascience.com/understanding-and-implementing-neural-networks-in-java-from-scratch-61421bb6352c

[18] J. Violos, S. Tsanakas, M. Androutsopoulou, G. Palaiokrassas, and T. Varvarigou, Next Position Prediction using LSTM Neural Networks, *11th Hellenic Conference on Artificial Intelligence*, 2020.

[19] B. L. Yoon, Artificial neural network technology, *ACM SIGSMALL/PC Notes*, vol. 15, no. 3, pp. 3–16, 1989.

[20] J. L, Calculating an intercept course to a target with constant direction and velocity (in a 2-dimensional plane), *The source of all good bits*, July 17 2011. [Online]. Available: http://jaran.de/goodbits/2011/07/17/calculating-an-intercept-course-to-a-target-with-constant-direction-and-velocity-in-a-2-dimensional-plane /

[21] Tokuyama, Yusuke, Ryo Miki, Yukinobu Fukushima, Yuya Tarutani, and Tokumi Yokohira, Performance Evaluation of Feature Encoding Methods in Network Traffic Prediction Using Recurrent Neural Networks, *ICIET*, 2020.

[22] Shaikh, Rahil, Choosing the right Encoding method-Label vs OneHot Encoder, *Towards Data Science*. 2018.

MODELLING A MICROSURGICAL SUTURE IN UNITY

Justin Stevens[1] , Chad Hogg[1] , Evan Hanzelman[1] , Brian Smith[2] , Joseph Sassani[2],[3]
[1]Millersville University
[2]Simulation Systems
[3]Penn State School of Medicine
{jmsteve1, chad.hogg}@millersville.edu, evan.hanzelman@gmail.com, bsmith@simsys.us, jxs14@psu.edu

ABSTRACT

Virtual reality simulators can be an effective way to train and evaluate future surgeons, but require realistic modeling of physical objects such as bodily tissues, sutures, and instruments. We have found accurately modeling a suture to be especially difficult, and describe here several approaches that we have attempted within the Unity game engine.

1 Introduction

Virtual reality simulators provide a cost-effective and safe way for surgeons to practice and perfect their craft, and to have their skills evaluated. We are attempting to build such a simulator, initially for ophthalmologic microsurgery, in the Unity 3D engine. So far we have succeeded in creating custom peripherals similar to real surgical instruments and sensing their location and orientation in space. An off-the-shelf 3D video system allows the user to see virtual representations of these instruments in a virtual world, in which maintain the same position and orientation as the peripherals do in the physical world. The goal of our simulator is that users will be able to use these virtual forceps to grasp a simulated suture to stitch together simulated simulated body tissues and tie knots, with the knot-tying as a particular point of emphasis.

The most challenging aspect of this simulation has been modeling a suture that behaves similarly enough to real sutures that the virtual experience will be meaningfully similar to the physical experience. For now, our suture is composed of a series of nodes connected by joints. We have found ways to configure these nodes and joints that produce a marginally acceptable model, but are continuing to attempt to improve it. The paper discusses the various challenges in creating a realistic model and the parameter values that we have found to work best.

2 Background and Related Work

Microsurgical suturing, or microsuturing, is performed by ophthalmologists in procedures such as scleral laceration repair and keratoplasty. It is one of the most technically demanding surgical skills, involving the precise placement of microsutures in sub-millimeter sized anatomy using only hand-held instruments. Consequently, microsuturing requires years of training and practice to master. In some cases, new technology has replaced the need for microsuturing. For example, cataract surgery is now generally performed via a self-sealing sutureless wound. However, procedures involving trauma repair, for example, still depend upon microsuturing, and it remains a critical part of the ophthalmologist's toolkit. Moreover, the reduction in the use of suturing during cataract procedures has decreased resident experience with this vital skill.

Therefore, the trend away from microsuturing creates a risk for those patients who must have a procedure that requires it. In routine clinical practice, ophthalmologists and ophthalmology residents now have fewer opportunities to perform "live" microsuturing than they once did, and in the absence of this frequent practice, skills can become rusty. In some cases, a resident may not acquire enough microsuturing practice during his or her ophthalmology training to gain proficiency, and therefore that resident may not be capable of operating independently or safely.

Unfortunately, opportunities to practice microsuturing outside the operating theater are limited. An optimal training method should (1) have high fidelity (i.e. realism or face validity), so that the rehearsed skills translate to real-world performance, and (2) be convenient to deliver, so that it can be implemented with meaningful frequency. These two attributes are often juxtaposed, and microsurgery training models rarely provide the type and frequency of practice needed for trainees to gain, maintain, and document proficiency [1; 2].

Current methods of microsuturing instruction include practice on tissue substitutes such as surgical tubing or 3D printed models; practice on cadaveric tissue and dead animal parts; and practice on live, anesthetized animals [3; 4; 5]. The inexpensive substitutes usually lack the fidelity necessary to mimic real-world conditions, whereas the more expensive biological models are low throughput, require regulatory oversight and specialized facilities, and cannot be repeated with meaningful frequency. While there are some validated measures of microsurgical skill [6], overall competency is judged largely by subjective evaluation. None of

these training methods offers objective assessment of performance, which is needed both for focused didactic feedback and for standards-based competency benchmarking. Thus, they require the presence of a trained observer to document and to score the trainee's performance.

At this time of extreme sensitivity to cost and quality in healthcare, improved training methods are a simple yet often overlooked way to drive efficiency. For microsuturing in particular, good technique is expected to contribute to better outcomes and shorter operating times. Therefore, the challenge we seek to address is how high-fidelity microsuturing instruction can be delivered frequently, expediently, and inexpensively.

Virtual reality-based simulation has begun to transform the delivery of surgical training. The advantages are plentiful: simulation exercises can be made to be very realistic, yet they are easily deployed in an office or computer lab; every aspect of a simulation can be measured, analyzed, and documented to evidence learning progress; multiple scenarios can be constructed to provide practice on varying clinical presentations; and scenarios can be re-run over and over, at zero marginal cost per repetition, until proficiency is achieved. Currently, sophisticated surgical simulators exist for numerous specialties and techniques [7; 8; 9; 10; 11; 12], with the most mature applications in laparoscopy and robot-assisted surgery. Within ophthalmology, we know of two mature simulators, the Eyesi Surgical Simulator [13; 14] and the HelpMeSee Eye Surgery Simulator [15; 16], that have had a significant impact on preparing surgeons for cataract and vitreoretinal surgery. However, no simulator has been widely accepted for microsuturing.

Creating a simulator entirely from scratch allows the most flexibility and control, but would make the project very large and complex. We have chosen instead to use Unity, a modern 3D game engine that provides various tools and components for modeling phenomena in virtual 3D worlds. Several other research projects have also built surgical simulators of varying types on top of Unity [17; 18; 19; 20]. None of these projects model knot-tying in sutures.

3 Unity Components

Unity allows the programmer to create Game Objects, which represent things that exist in the virtual 3D world. By default, Game Objects contain only a Transform component, which describes the object's position, rotation, and scale within the world. Many other types of pre-built components can be added to Game Objects and configured as desired. For example, a Mesh component makes an object visible and controls its appearance.

A Collider component describes the volume of space filled by the Game Object, and be used to detect intersections with the volumes of other Game Objects that also have Colliders on them. A Rigid Body component connects a Game Object to the physics engine, causing it to respond appropriately to forces such as gravity and collisions with other objects that also have Rigid Bodies. Joint components connect Rigid Bodies together, causing forces applied to one to propagate to others, and constraining the types of rotations and movements that are possible between the Rigid Bodies. Unity currently offers Character, Configurable, Fixed, Hinge, and Spring joints. The Character, Fixed, Hinge, and Spring joints are all special cases of Configurable Joints since they can all be modeled using the Configurable Joint with certain parameters. Physics Material components further allow the programmer to control how Game Objects interact with the physics system by setting properties such as the Game Object's coefficients of static and dynamic friction.

Game Objects can be created, and components added or removed from them and reconfigured both through a world editor interface and through scripts that run dynamically as the simulation is in progress [21].

4 Modeling Requirements

Our simulation requires three types of objects: a tissue subtrate, two pairs of forceps, and a suture. Prospective surgeons practicing inside the simulation will be expected to use the forceps to grasp the suture, pull it through the tissue subtrate, and tie a knot with the two ends of the suture.

The only interesting behavior that the tissue substrate must have is allowing the end of the suture to pass through it and pull the rest of the suture along with it. We have a simple substrate model with holes that seems sufficient for our purposes, at least for an initial simulator.

The forceps are made up of many parts, each with Meshes and Colliders and Rigid Bodies that are intended to appear and behave like real surgical instruments. They are the virtual representations of custom peripherals we created that allow the positions and rotations of physical objects the user is holding to be observed and updated in the virtual world in real-time.

The most challenging object to model has been the suture, and we have not yet found a solution that is entirely satisfactory. A real suture used in microsurgery behaves similar to fishing line, though it is much smaller. It should be flexible enough to easily bend when force is applied to it, yet rigid enough that it attempts to return to an entirely relaxed position in the absence of any forces. When forceps tug on an end of a suture, the force should be applied evenly across the length of the suture, both deforming and moving it. When a suture has been tied into a knot, the force of friction must be sufficient to prevent the knot from unraveling.

5 Suture Models

We have chosen to model the suture as a long chain of discrete Game Objects, each with Capsule-shaped Colliders and appropriate Rigid Bodies and Physics Materials, with Configurable Joints connecting each node to the next. Configurable Joints were chosen to give the developers the most freedom when creating the joints, since they offer everything the other joints have and more. When force is applied to any node, that force is distributed through the joints to the other nodes. These nodes connected by joints create a chain-like structure shown in Figure 1.

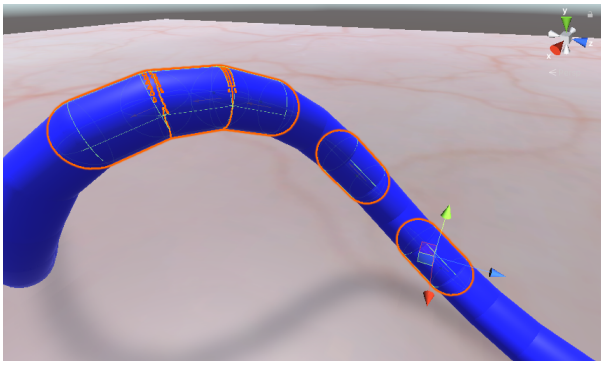

Figure 1: From left to right, three consecutive suture nodes, then two other nonconsecutive nodes.

Currently, this suture is being modeled using 128 individual nodes with 127 separate Configurable Joints connecting them. A sequence of more numerous but smaller nodes might allow better behavior, but increases the computational load on the physics engine, which must respond immediately to changes in the system. When these nodes are given the ideal parameters, which we have yet to discover, it may be able to perfectly model a real-world suture. We have begun to find ourselves in a dilemma wondering if we have yet to discover the ideal parameters for the Colliders / Rigid Bodies / Physics Materials on each node and the Configurable Joints between them. Or have we reached the limitation of our this type of structure and need to find an entirely different model?

So far we have been operating from the assumption that we have not yet found the ideal parameters. We have, however, found that changes in these parameters do materially affect the suture's behavior and have been able to move in the direction of desired behavior. Currently, inexperienced users are occasionally able to use the virtual forceps to grasp the simulated suture and tie a knot in it. Knowing this it is more than likely that those with suturing experience will be able to consistently tie knots, though this still needs to be tested on the current model. In addition to this, while tying the knot and manipulating the suture under normal circumstances, the suture remains relatively stable whereas past models would have joints break and/or the suture would explode. This occurs when Unity's physics engine cannot find a way to bring two subsequent nodes back together and the entire suture flies off the screen trying to find the equilibrium point. Finally, the current model interacts well with the forceps that the user uses to manipulate the suture's position and rotation. Previous models, which were made more rigid by adjusting the spring force on the joint, would periodically rebound or bounce away from the forceps when the user attempted to grasp a portion of the suture. While this still can happen in the current model, the likelihood of it happening has decreased considerably and when it does happen, the rebound is typically not as noticeable or as strong as in past models.

While all the points made above are definitely a step in the right direction, the current model is still missing important characteristics of a real-world suture. Most notably, the model does not contain the rigidity and overall feel that a suture in the real world exhibits. An actual suture has certain characteristics and rigidity that is often compared to that of a human hair or fishing line. The current suture, with its lack of rigidity, behaves more like cooked spaghetti. Even though it is too floppy, this actually allows for an easier knot tying experience when compared to previous models. However, when a knot is tied, the suture does not stay in place and it starts to untangle itself. While this can be fixed by increasing the friction on the nodes' Physics Material, this makes it harder to tie a knot and makes it more likely to explode. In addition to this, there are a few bugs with the current model that break the immersion of a real suturing environment. As noted above, the model can explode while a real suture would simply break into two pieces when too much force is applied. Also, when the user grasps a node in between a joint, the Unity physics engine cannot find the equilibrium point for those joined nodes, since the forceps are in the way. This causes the suture to spasm until the user ungrasps it and makes it difficult to grasp the other side of the suture with the other forceps due to the spasm. In extreme circumstances, where the joints are under too much stress, this bug will cause the suture to explode. The last known bug, which is mentioned above, is the rebound bug, in which the suture moves away from the forceps while attempting to grasp the suture. While this bug has been minimized and its effects are mostly negligible in the current model, finding a solution to this bug and all others previously mentioned is necessary for creating an immersive suturing experience.

6 Parameter Variations and Observed Effects

During the process of tweaking parameters of various entities related to the suture, we began to document how changes to different parameters affect the suture's behavior and the positives and negatives that come with these changes. Oftentimes when tweaking parameters, the model felt more like a real suture, but it introduced new bugs or made existing bugs more common and/or had a worse effect when they occurred. Unfortunately, the opposite was also found to be true.

Figure 2 shows just a few of the parameters of a Configurable Joint, with the values that we found to work best. The most common parameter we tweaked was the Angular Y Limit and

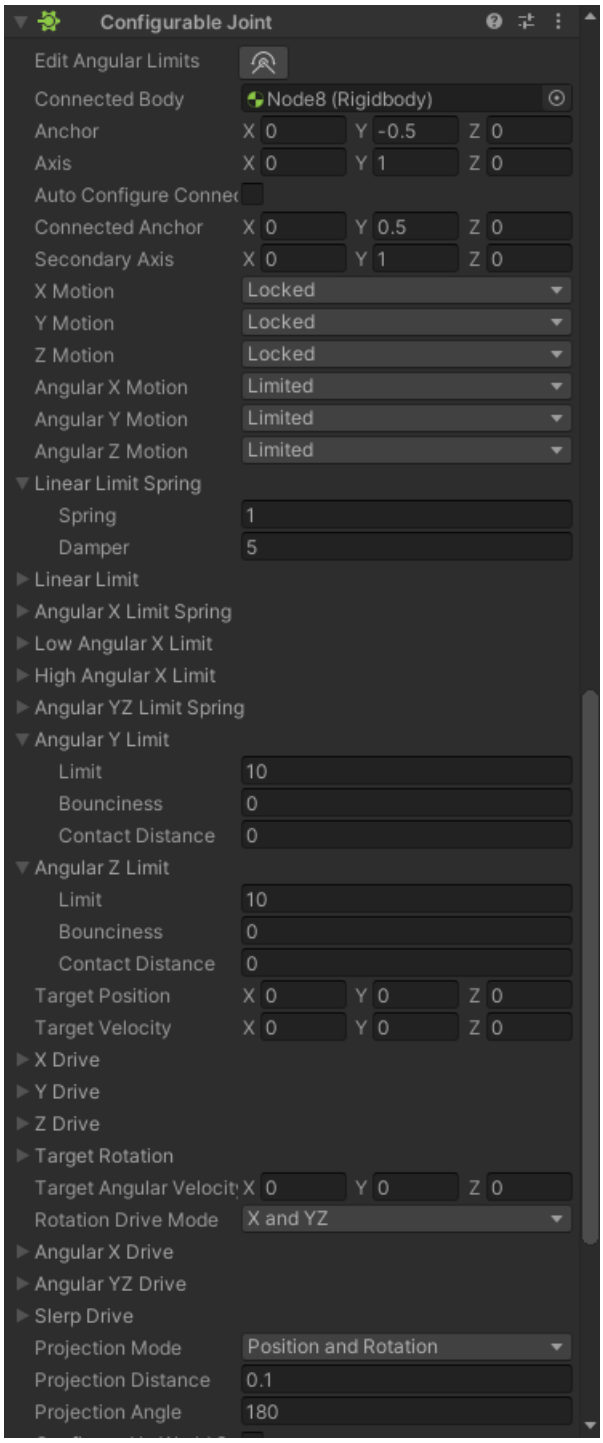

Figure 2: Configurable Joint parameters.

the Angular X Limit on the nodes' Configurable Joint. This value determined how far a joint was allowed to bend before the spring force would begin acting on the Rigid Bodies. One would think setting this parameter to 0 and the spring force to a very high value would make the joint stiff and not bend, but this was not the case. It did make the suture more rigid, but only to a certain degree that was not nearly as rigid as that of a real suture. With what we know, this was caused by some external force still being acted on the joint that Unity deemed stronger than the spring force. The external force is gravity acting on each node and the forces that the forceps apply on the suture while pulling on it. We ultimately set the angular limits to 10 degrees since too high of a value allowed the suture to be too flexible and anything less made the suture hard to bend, which introduced more stress on the joints. More stress created a higher likelihood of the suture exploding.

After we found an agreed upon value for the angular limits, we further experimented with the linear limit spring on the Configurable Joint. The associated value dictates the strength of the forces that the joint applies to the connected node when it leaves the angular limit. The spring value had to be set such that the spring was not too strong, so that there would not be a lot of stress on the joint when bending it and tying it in a knot, while at the same time the joint had to be rigid. With numerous testing, the value of 1 was found to be the best balance of the two. The damper value has been tricky to calibrate. This value softens the force of the spring to eliminate jittering. A high value completely removes the jittering effect, but makes the spring force far too weak. A low value does nothing to fix the jittering effect of the suture. Through testing we arrived at a value of 5, but this value may need to be further tested.

Figure 2 shows a variety of other parameters that can also be tweaked to perfect the joint. These values have been briefly experimented with and thus far have proved to have little to no effect compared to the parameters listed above or changed the joint in a way that was not desired. For this reason, these values will not be examined in detail. Furthermore, the Colliders, Rigid Bodies, and Physics Materials have parameters that can be adjusted as well. In future models, these parameters will be further tested.

7 Future Work

There are many other components to the simulator that need to be completed or improved before it will be ready for use by students. Focusing on the model of the suture for now, we have many parameters left to consider. We have also thought that it may be necessary for these parameters to change dynamically based on properties such as the number of other nodes in proximity or whether or not this particular node has been grasped. If we cannot achieve a sufficiently realistic model through any tweaking of these parameters we may abandon our nodes-and-joints model in favor of something else entirely, such as a deformable mesh.

8 Conclusion

Unity has proven itself to be an effective, but overwhelmingly flexible, system for modeling 3D phenomena. Modeling a microsurgical suture is especially difficult because it must balance flexibility and rigidity, distribute forces over its entire surface, and respond realistically to minute forces. We believe that a series of nodes connected through Configurable Joints is the most promising technique for building such a model, and have been able to fine-tune it by experimenting with many of the parameters that are provided by that Unity component.

References

[1] S. Ramachandran, A. M. Ghanem, S. R. Myers, Assessment of microsurgery competency – where are we now?, *Microsurgery*, *33*(5):(2013), 406–415.

[2] R. M. Studinger, M. M. Bradford, I. T. Jackson, Microsurgical training: is it adequate for the operating room?, *Eur. J. Plast. Surg.*, *28*(2):(2005), 91–93.

[3] A. M. Ghanem, N. Hachach-Haram, C. C. M. Leung, S. R. Myers, A systematic review of evidence for education and training interventions in microsurgery, *Arch. Plast. Surg.*, *40*(4):(2013), 312–319.

[4] C. C. M. Leung, A. M. Ghanem, P. Tos, M. Ionnac, S. Froschauer, S. R. Myers, Towards a global understanding and standardisation of education and training in microsurgery, *Arch. Plast. Surg.*, *40*(4):(2013), 304–311.

[5] M. Singh, N. Ziolkowski, S. Ramachandran, S. R. Myers, A. M. Ghanem, Development of a five-day microsurgery simulation training course: a cost analysis, *Arch. Plast. Surg.*, *41*(3):(2014), 213–217.

[6] D. Dumestre, J. K. Yeung, C. Temple-Oberle, Evidence-based microsurgical skills acquisition series part 2: validated assessment instruments – a systematic review, *J. Surg. Educ.*, *72*(1):(2015), 80–89.

[7] K.-S. Choi, S.-H. Chan, W.-M. Pang, Virtual suturing simulation based on commodity physics engine for medial learning, *J. Med. Syst.*, *36*(3):(2012), 1781–1793.

[8] S. Haque, S. Srinivasan, A meta-analysis of the training effectiveness of virtual reality surgical simulators, *IEEE Trans. Inf. Technol. Biomed.*, *10*(1):(2006), 51–58.

[9] H. Kazemi, J. K. Rappel, T. Poston, B. H. Lim, E. Burdet, C. L. Teo, Assessing suturing techniques using a virtual reality surgical simulator, *Microsurgery*, *30*(6):(2010), 479–486.

[10] I. Kozak, P. Banerjee, J. Luo, C. Luciano, Virtual reality simulator for vitreoretinal surgery using integrated OCT data, *Clin. Ophthalmol.*, *8*:(2014), 669–672.

[11] G. S. Ruthenbeck, K. J. Reynolds, Virtual reality surgical simulator software development tools, *J. Simul.*, *7*:(2013), 101–108.

[12] K. Triantafyllou, L. D. Lazaridis, G. D. Dimitriadis, Virtual reality simulators for gastrointestinal endoscopy training, *World J. Gastrointest. Endosc.*, *6*(1):(2014), 6–12.

[13] D. S.-C. Ng, Z. Sun, A. L. Young, S. T.-C. Ko, J. K.-H. Lok, T. Y.-Y. Lai, S. Sikder, C. C. Tham, Impact of virtual reality simulation on learning barriers of phacoemulsification perceived by residents, *Clin. Ophthalmol.*, *12*:(2018), 885–893.

[14] M. Radia, M. Arunakirinathan, D. Sibley, A guide to eyes: ophthalmic simulators, *Bull. R. Coll. Surg. Eng.*, *100*(4):(2018), 169–171.

[15] J. R. Broyles, P. Glick, J. Hu, Y.-W. Lim, Cataract blindness and simulation-based training for cataract surgeons, *Rand Health Q.*, *3*(1).

[16] A. Singh, G. H. Strauss, High-fidelity cataract surgery simulation and third world blindness, *Surg. Innov.*, *22*(2):(2015), 189–193.

[17] J. Zhang, Y. Lyu, Y. Wang, Y. Nie, X. Yang, J. Zhang, J. Cheng, Development of laparoscopic cholecystectomy simulator based on unity game engine, in *CVMP'18*, pages 1–9 (2018).

[18] K. Fan, A. Marzullo, N. Pasini, A. Rota, M. Pecorella, G. Ferrigno, E. D. Momi, A unity-based da vinci robot simulator for surgical training, in *BioRob'22*, pages 1–6 (2022).

[19] F. Zhang, Z. Sun, T. Wang, Brain modeling for surgical training on the basis of unity 3d, in *ISAIR'22*, pages 1–8 (2022).

[20] L. Khan, Y.-J. Choi, M. Hong, Cutting simulation in unity 3d using position based dynamics with various refinement levels, *Electronics*, *11*(14).

[21] *Unity User Manual*.

COMPARATIVE STUDY OF OUTLIER DETECTION TECHNIQUES FOR CREDIT CARD FRAUD

Claude Pierre Louis, Rachel Shirey, and Dr. Raed Seetan
Slippery Rock University
Slippery Rock, PA 16057
{cvp1002, rbs1005, raed.seetan}@sru.edu

ABSTRACT
Many machine learning techniques have been developed to detect outliers within datasets. These techniques are used to detect whether activity on a credit card is fraudulent or legitimate. The primary purpose of this study is to compare three of these well-known outlier detection machine learning techniques on a dataset with fraudulent credit card activity within. Isolation Forest, One-class Support Vector Machines (SVM), and Local Outlier Factor (LOF) Algorithms were investigated, and the results were analyzed. To evaluate the performance of these three machine learning techniques, we used a dataset consisting of more than 280,000 credit card transactions. The results of this research study showed while Local Outlier Factor Algorithm and One-class Support Vector Machines Algorithm both had better results for precision and recall than Isolation Forest Algorithm, the only algorithm of the three that was specific enough was the Isolation Forest Algorithm. With a specificity rate of 80 percent, Isolation Forest Algorithm did what Local Outlier Factor Algorithm and One-class Support Vector Machines Algorithm could not do–which was to properly identify the true negatives. It is for the aforementioned reasons that the Isolation Forest produced the most reliable results with acceptable rates of precision, recall, specificity, and sensitivity.

KEY WORDS
Outlier Detection, Credit Card Fraud, Isolation Forest, Support Vector Machines, Local Outlier Factor.

1. Introduction

The usage of credit cards in United States is higher than it has ever been, with more than 1 billion credit cards in use [1]. In 2018, there were 130,928 credit card fraud reports recorded in the United States [1]. Not only were there so many credit card fraud reports, but there was also a total of $24.26 billion lost to payments on credit card fraud worldwide with a prediction of global losses to grow by another $10 billion over the next three years [1]. Credit card companies and banks use outlier detection techniques in order to find outliers, or data objects that have deviated significantly from the other objects [2]. However, according to an Experian report, only 54 percent of businesses stated that they were "somewhat confident" in their ability to detect fraudulent activity on time [1]. Additionally, only 40 percent stated that they felt "very confident" about preventing these incidents [1]. The United States makes up 38.6 percent of the world's reported payment card fraud losses even though the US only generates 29 percent of total global purchases [1].

This research study serves to offer greater understanding about existing methods for outlier detection. This study is important especially in the United States where the proportion of the United States' portion of the world's reported payment card fraud losses is too high compared to the total global purchases.

Previous studies have investigated one of these machine learning techniques at a time but have not compared to see which technique works the best for credit card fraud detection.

In this study we investigate the performance of three machine learning techniques: Isolation Forest Algorithm, One-class Support Vector Machines (SVM) Algorithm, and Local Outlier Factor (LOF) Algorithm to perform credit card fraud detection using a dataset with a total of 284,807 credit card transactions.

Isolation Forest Algorithm is a very effective tree-based algorithm that detects both outlier and novelty detection in high-dimensional data. This algorithm works to isolate observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. In the Isolation Forest Algorithm, the contamination hyperparameter is the most important value but is also an unknown value [3]. This hyperparameter represents the proportion of outliers in the dataset and the values range from 0 to 0.5, with the default value being 0.1. If there is reason to believe there will be many outliers in the data, the contamination can be set to a larger value. However, this unknown value is a major limitation [4].

One-class SVM Algorithm is an unsupervised machine learning algorithm that can be used for novelty detection. This algorithm is very sensitive to outliers which makes it the best option to use for novelty detection when the data is not very polluted with outliers. One-class SVM can also be applied to high-dimensional data sets and does not have any underlying assumption in the distribution of the data [3].

Local Outlier Factor Algorithm is also an unsupervised machine learning algorithm and was originally created for outlier detection, however, can also be used for novelty detection. This algorithm also works well with high-dimensional datasets. With LOF, the local density deviation of a given data point is computed with respect to the neighboring data points. The samples with a substantially lower density are considered outliers [3].

The remainder of this paper is structured as follows: Section 2 discusses the related works. Section 3 presents our methodology. Section 4 discusses the results of the study. Section 5 concludes our study. Lastly, Section 6 provides our recommendations.

2. Related Work

Anomaly Detection in Credit Card Transactions Using Machine Learning [5]

Meenu et al. worked on this project in 2020. In their study, the researchers aimed to develop an automated classifier that can detect fraudulent credit card transactions with high efficiency [5]. The performance was evaluated by using precision and recall.

The researchers used the Isolation Forest algorithm and H2O.ai, which supports the most widely used supervised and unsupervised machine learning algorithms.

The test data used for this study was found on Kaggle. This dataset includes approximately 500 fraudulent transactions, and 284,300 reported legitimate transactions, which makes it a highly imbalanced dataset [5].

The researchers concluded that the technique they used was successful in its ability to distinguish anomalies and inliers by creating several decision trees for every data point and was effective with a fraud detection model observed to be 98.7 percent [5].

Outlier Detection Credit Card Transactions Using Local Outlier Factor Algorithm (LOF) [6]

Sugidamayatno and Lelono worked on this study in 2019 and looked at utilizing the Local Outlier Factor Algorithm to detect outliers in a dataset [6]. This study used a much smaller dataset consisting of only 10 transactions.

The researchers found that the LOF Algorithm resulted in the highest level of accuracy, recall, and precision and 96 percent, 98 percent, and 9 percent, respectively [6].

A Comparison of Outlier Detection Algorithms for Machine Learning [7]

H. Escalante researched six methods for outlier detection in this research paper written in 2014. The study looked at the following methods: distance based, distance K-based, statistical, kernel-based, v-SVM, and one-class SVM [7]. The researcher used datasets from the UCI repository and added outliers and noise to properly test the algorithms.

The kernel-based novelty detection approach was found to be the best performer, as it showed perfect performance four times [7]. On the other hand, the v-SVM was found to be the worst method as it only detected a few of the true outliers.

Additionally, to note, the one-class SVM and the distance-based methods detected almost 100 percent of the outliers but had an incredibly high rate of false positives [7].

Based upon the aforementioned reasons, this study concluded that the kernel-based novelty detection method is also effective and very simple [7].

A Survey on Outlier Detection Techniques for Credit Card Fraud Detection [8]

Pawar et al. worked on this research study in 2014 and they investigated Principal Component Analysis (PCA) to detect an outlier [8].

The dataset that was used is available on UCI Machine Learning Repository and is standard German Credit Card Fraud dataset [8]. The PCA technique was also tested with two-dimensional synthetic data where 100 data instances were generated, consisting of 80 normal instances and 20 outliers [8].

The researchers concluded that the Principal Component Analysis is suitable for credit card fraud detection [8].

3. Methodology

3.1 Dataset

The selected dataset for this work was downloaded from Kaggle.com and contains credit card transactions made by European cardholders in September 2013. The dataset has a total of 284,807 transactions that occurred in a span of two days. There are 492 frauds present in the dataset, which represent 0.172% of all the transactions recorded.

The data contains numerical values that have undergone dimensionality-reduction by principal component analysis. As a real dataset, the reduction step was crucial to make it more manageable and maintain the confidentiality of the raw data values. Credit card fraud detection relies heavily on outlier detection methods, thus the choice of this dataset.

The data consists of 31 attributes. The attribute V1, which is the first one, contains the time of the transaction and V31 contains two numerals, 0 and 1, identifying regular transactions and fraud respectively. The three algorithms will be tested with this dataset using the Weka Tool, version 3.8.6.

3.2 Preprocessing & Analysis Preparation

The preprocessing involved reducing the size of the data set. As we know, scaling is a major problem in data cleaning, and it was challenging to analyze almost 300,000 instances in Weka Tool.

After running a few tests with the program crashing multiple times and displaying 'low memory' as the cause it became apparent that the data size would need to be reduced in order to be able to test the algorithms.

The first part of the preprocessing was done using Microsoft Excel, since the data was converted into a comma-separated value (.csv) file. The data was filtered using the 30th attribute 'Amount' to remove all transactions that were less than 100 dollars. Completing this task reduced the number of instances to 57,385, which represent about twenty percent (20%) of the original data set.

All the data was numerical, which is not a suitable format for the algorithms we selected. In order to solve this issue, the 'class' attribute was changed from numerical to nominal by using the unsupervised attribute 'Numerical To Nominal' Filter in Weka Tool.
Additionally, an extra column named 'label' was added to the dataset, as part of a requirement for One-class SVM algorithm.

It is important to note that the data is highly imbalanced, with the preprocessed data set containing only 130 instances classified as fraud.

3.3 Analysis Method

The following algorithms were used to analyze the data: Local Outlier Factor, Isolation Forest and One-class Support Vector Machine.

The Local Outlier Factor is a classifier that applies an algorithm to compute an outlier score for each instance in the data.

The Isolation Forest is a tree-based classifier designed for anomaly detection. This algorithm works by analyzing how far a data point is from the rest of the data, rather than modeling the normal points.

The One-class Support Vector Machine is an unsupervised model for outlier detection. It works by learning the boundary for the normal data points, then it identifies the data outside of the border as anomalies.

The above algorithms were applied to the preprocessed dataset separately with a 10 folds Cross-Validation. The results were exported to Microsoft Excel for analysis and comparison using precision, recall, sensitivity and specificity rates.

Algorithm	Type	Weka Attribute
LOF	Schema	weka.classifiers.misc.LOF -min 10 -max 40 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\"" -num-slots 1
	Filter	creditcard_shrunk-weka.filters.unsupervised.attribute.NumericToNominal-Rlast
One Class SVM	Schema	weka.classifiers.functions.LibSVM -S 2 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model / -seed 1
	Filter	creditcard_shrunk-weka.filters.unsupervised.attribute.NumericToNominal-Rlast
Isolation Forest	Schema	weka.classifiers.misc.IsolationForest -I 100 -N 256 -S 1
	Filter	creditcard_shrunk-weka.filters.unsupervised.attribute.NumericToNominal-Rlast

Table 1. Weka Schemas and Filters

4. Results

The Support Vector Machine (SVM) had the best precision and recall, with a mean rate of 99.8% and 99.8% respectively. The algorithm was able to correctly classify 99.8% of the instances. The Isolation Forest algorithm correctly classified 94.7% of the instances and had a mean precision of 99.7%, and a mean recall rate of 94.7%. The Local Outlier Factor (LOF) algorithm correctly identified 99.7% of the instances and had a mean precision of 99.5% and a mean recall of 99.8%.

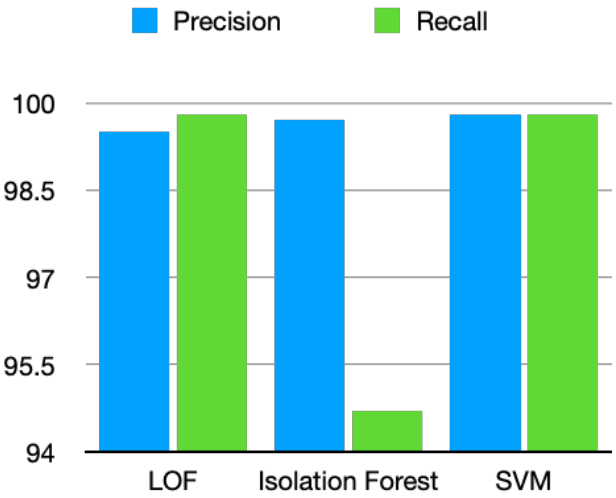

Figure 1. LOF, Isolation Forest and SVM precision and recall rate in percentage

The SVM and LOF algorithms perform better instance identification than the Isolation Forest. However, all three algorithms have about the same precision. The Isolation Forest had the lowest rate of recall.

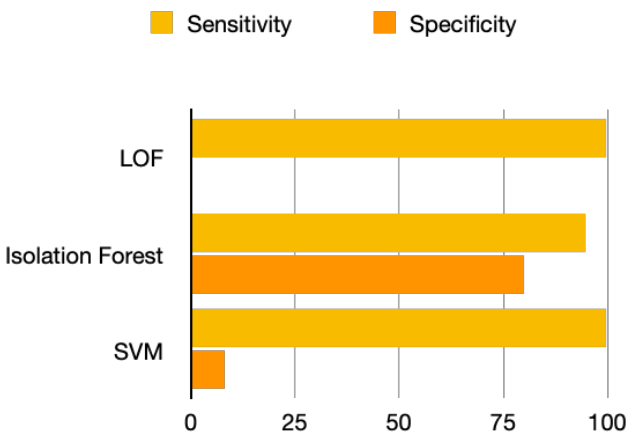

Figure 2. Sensitivity and Specificity of LOF, Isolation Forest and SVM in percentage

The specificity and sensitivity of the algorithms varied greatly. The sensitivity was 99.98%, 94.7%, and 100% for LOF, Isolation Forest and SVM respectively. However, only the Isolation Forest algorithm was specific enough, with a rate of 80%. The confusion matrix of the LOF and SVM algorithms could not properly identify the true negatives, resulting in incorrectly classifying many instances.

```
=== Confusion Matrix ===

     a     b   <-- classified as
 54235  3020 |     a = 0 = normal
    26   104 |     b = 1 = fraudulent
```

Figure 3. Isolation Forest Confusion Matrix

```
=== Confusion Matrix ===

     a     b   <-- classified as
 57255     0 |     a = 0 = normal
   120    10 |     b = 1 = fraudulent
```

Figure 4. Support Vector Machine Confusion Matrix

```
=== Confusion Matrix ===

     a     b   <-- classified as
 57243    12 |     a = 0 = normal
   130     0 |     b = 1 = fraudulent
```

Figure 5. Local Outlier Factor Confusion Matrix

The dataset used was huge and highly unbalanced and this may explain why we obtained these results. Out of the three algorithms, the Isolation Forest produced the most reliable results, that is an acceptable rate of precision, recall, specificity and sensitivity.

5. Conclusion

This study demonstrates and compares the capabilities of three different machine learning techniques to find outliers in datasets. The results of this research study show that the Support Vector Machine Algorithm had the best precision and recall, with a mean rate of 99.8 percent for both precision and recall. Correct classification of the instances was 99.8 percent, as well. Local Outlier Factor Algorithm came in second best with correct identification of instances at 99.7 percent, mean precision at 995 percent, and mean recall at 99.8 percent. The least effective method was the Isolation Forest Algorithm with 94.7 percent correctly classified instances, 99.7 percent mean precision, and 94.7 percent mean recall.

The sensitivity of the Local Outlier Factor, Isolation Forest, and One-class Support Vector Machines were 99.98 percent, 94.7 percent, and 100 percent, respectively. However, the only algorithm that was specific enough was the Isolation Forest algorithm. Isolation Forest had a rate of 80%, while the Local Outlier Factor and One-class Support Vector Machines were unable to properly identify the true negatives, which led to incorrect classification of many instances.

6. Recommendations

It is essential to consider the size of a dataset and the computational power of the machine that will be used to run the algorithm. Scalability and efficiency are known challenges of data mining and analysis. For example, one of the three algorithms selected for this study ran for over twenty-four hours before producing a result. Furthermore, the program used to run an algorithm may be a limiting factor, thus somehow influencing the results of an analysis. Therefore, further studies may compare the results of one algorithm applied to the same dataset using R, Python, and Weka Tool.

References:

[1] N. Cveticanin, “Credit Card Fraud Statistics: What Are the Odds?,” dataprot.net, 02 Nov 2022. [Online]. Available: https://dataprot.net/statistics/credit-card-fraud-statistics/ [Accessed 21 Nov 2022].

[2] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, Third Edition. Waltham, MA: Elsevier, Inc, 2012.

[3] R. Pramoditha, “4 Machine learning techniques for outlier detection in Python,” towardsdatascience.com, 27 Mar 2021. [Online]. Available: https://towardsdatascience.com/4-machine-learning-techniques-for-outlier-detection-in-python-21e9cfacb81d. [Accessed 08 Nov 2022].

[4] R. Pramoditha, “Two outlier detection techniques you should know in 2021,” towardsdatascience.com 20 Mar 2021. [Online]. Available: https://towardsdatascience.com/two-outlier-detection-techniques-you-should-know-in-2021-1454bef89331. [Accessed 09 Nov 2022]

[5] Meenu, Gupta, S., Patel, S., Kumar, S., & Chauhan, G. (2020, September 23). Anomaly detection in credit card transactions using Machine Learning. SSRN. Retrieved November 15, 2022, from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3670230

[6] Sugidamayatno, S., & Lelono, D. (2109). Outlier detection credit card transactions using local outlier factor algorithm (LOF). IJCCS (Indonesian Journal of Computing and Cybernetics Systems). Retrieved November 15, 2022, from https://jurnal.ugm.ac.id/ijccs/article/view/46561

[7] H. Jair Escalante, “A Comparison of Outlier Detection Algorithms for Machine Learning,” Programming and Computer Software, January 2005. Available: https://www.researchgate.net/profile/Hugo-Jair-Escalante/publication/228728521_A_comparison_of_outlier_detection_algorithms_for_machine_learning/links/0912f50b777c20ab5e000000/A-comparison-of-outlier-detection-algorithms-for-machine-learning.pdf. [Accessed 08 Nov 2022].

[8] Pawar, A.D., Kalavadekar, P.N., & Tambe, S.N. (2014). A Survey on Outlier Detection Techniques for Credit Card Fraud Detection. IOSR Journal of Computer Science. Retrieved 01 Dec 2022, from https://www.iosrjournals.org/iosr-jce/papers/Vol16-issue2/Version-6/H016264448.pdf

COMBINING UNSUPERVISED AND SUPERVISED LEARNING FOR CREDIT CARD FRAUD DETECTION

Briar Sauble
Undergraduate of Department of Computer Science, Millersville University
bjsauble@millersville.edu

ABSTRACT

Due to the rapid growth of e-commerce, finding a proper method to detect credit card fraud becomes more important than ever. Supervised learning methods can find patterns of credit card transactions but can miss novel patterns of fraud and consumer patterns that have yet to be trained upon. Unsupervised learning, in contrast, can be used to find anomalies instead of specific patterns but is limited due to variability caused by how different components are weighed. In this paper, we develop a way to combine unsupervised learning with weights discovered by prior supervised learning to improve credit card fraud detection. We use a decision tree to find the importance of different components of a credit card transaction in relation to fraud. This importance is then used to determine weights for distance used with the k-means clustering algorithm. To test our weighted k-means clustering algorithm, a dataset consisting of synthetic credit card transactions with over five hundred thousand transactions is used. Our algorithm improves clustering accuracy to 98.51%, compared to the normal k-means clustering accuracy of 87.92%. Moreover, the banking industry wants its models to be clearly interpreted. So, we discover a simple soft clustering algorithm based on our hybrid method.

KEY WORDS

Credit card, Fraud detection, K-means, C4.5 Decision tree, Clustering, Anomaly detection

1. Introduction

Credit card use has exponentially increased in the modern age due to its ease of use in transactions in digital and physical retail spaces. However, this corresponding increase has subsequently led similarly large losses due to the potential for credit card fraud. Out of 320k registered reports on fraud, as noted by the Federal Trade Commission (FTC), 133k of these related to credit card fraud [1]. The immense amount of losses because of fraud has brought as demand for efficient ways of detecting credit card fraud.

To aid in this search, machine learning algorithms provide a way to automate the process of data analysis to calculate a likelihood of a given transaction being credit card fraud. Two primary paradigms exist, typically falling between unsupervised and supervised learning. For credit card anomaly detection, unsupervised learning is used on unlabeled datasets, clustering information based on the patterns it finds. Supervised learning, in contrast, uses of labeled datasets to train algorithms that to classify data or predict outcomes accurately. Both supervised learning and unsupervised learning hold significant limitations—unsupervised learning lacks the knowledge of patterns that may be known and trained on by a supervised learning method, while supervised learning depends on the existence of labeled data as well as the accuracy of the data it is trained on, which may not always be feasible for credit card fraud analysis [2]. Research on comparing different machine learning techniques for credit card fraud detection can be found in [3] and [4].

Previous research has investigated the possibility of utilizing different unsupervised and supervised machine learning algorithms. These have primarily used synthetic datasets, due to the difficulty of acquiring real-world credit card fraud data. Methods using fuzzy clustering and neutral networks on synthetic datasets have proven to obtain a 93.90% true positives [5]. Other research has investigated semi-supervised methods, such as Dynamic Incremental Semi-Supervised Fuzzy C-Means (DISSFCM), to find results using both unlabeled and labeled data for precision levels around 70% [2]. In contrast, utilization of split algorithms, with a K-means unsupervised algorithm mixed with a supervised algorithm has led to promising results, with the use of random forests leading accuracies as high as 99.9% [6]. Prior research suggests an incentive to analyze a mixture of unsupervised and supervised learning for future research. More related research can be found in [7].

In this paper, we propose a hybrid method of determining credit card fraud using a C4.5 decision tree alongside K-means clustering. We use a decision tree to find the importance of different components of a credit card transaction in relation to fraud. This importance is then used to determine weights to modify the Euclidean distance of a k-means clustering algorithm. To test our weighted k-means clustering algorithm, a dataset consisting of synthetic credit card transactions with about one million and eight thousand transactions is used. Our algorithm improves the fraud detection accuracy to 95.35%, compared to the normal k-means clustering accuracy of 90.50%.

Furthermore, we develop a soft clustering algorithm based on the proposed method so the predictions can be easily adjusted using a threshold. This is more applicable to the banking industry since they need to adjust the predictions according to the current economic situation. Also, they want their models to be easily interpreted and meet the audit requirements. Our soft clustering algorithm is simple and easy to interpret, unlike many existing soft clustering algorithms.

2. Background

In this section, we introduce the machine learning techniques used in this paper. Unless referenced otherwise, the descriptions and formulas referenced within this research are derived from [8].

2.1. Decision Tree

Decision trees are a supervised classification methodology that uses a hierarchical leaf structure that split off to express all the possible choices for a set of data. Splits occur based upon split criterion, which have quantifications of split quantity that are dependent on the algorithm that is used. The C4.5 algorithm used in this research utilizes an entropy measure split. The entropy *E(S)* for a given set *S* is computed based on the class distribution $p_1 \dots p_k$ as follows:

$$E(S) = -\sum_{j=1}^{k} p_j log_2 (p_j) \tag{1}$$

The C4.5 algorithm, when determining its split criteria, uses the difference in entropy to find the information gain, with the highest information gain being utilized to split at a given point. It acquires this by subtracting *E(S)* by the overall entropy for an r-way split of set S, as follows:

$$EntropySplit(S \to S_1 \dots S_r) = \sum_{i=1}^{r} \frac{|S_i|}{|S|} E(S_i) \tag{2}$$

The information gain found is then divided by a normalization factor for the sake of adjusting for a varying number of categorical values. The $EntropySplit(S \to S_1 \dots S_r)$ for a given element of S, such as S_1, can then later be used as a feature importance for a dataset.

2.2. K-Means

K-means is an unsupervised learning algorithm that requires the repeated determination of centroids of clusters, and the determination of similarity between the centroids and the original data points. The algorithm iteratively determines the centroids of clusters, and then assign data points to their closest centroid. At a higher level, these steps remain the same for categorical data. However, the specifics of both steps are affected by the categorical data representation as follows:

1. Centroid of a data set: All representative-based algorithms require the determination of a central representative of a set of objects. In the case of numerical data, this is achieved very naturally by averaging.

2. Calculating similarity to centroids: The most common way to determine the similarity is to compute the Euclidean distance between a data point and the centroids.

The Euclidean distance formula determines the space between two points within a dataset, following the formula, with q and p representing two points in space, n representing the number of dimensions, and q_i and p_i representing two vectors of a given space:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{3}$$

A weighted distance follows a similar formula, but for each Euclidean vector $(q_i - p_i)^2$ a predetermined weight w_i is applied as a product, with each set of vectors having its own weight.

2.3. Soft Clustering

Soft clustering, or fuzzy clustering, does not use binary values for determining whether a specific data point will belong to a given cluster. Instead, it assigns a probability chance that a datapoint fits within a given cluster, with a value closer to a given centroid being given a greater probability. An instance of a soft clustering algorithm is the Fuzzy C-means (FCM), used in previous credit card analysis to determine a suspicion score within given clusters [2]. Further soft clustering algorithms have been researched elsewhere [9], proving useful in machine learning research.

3. Proposed Hybrid Learning Model

3.1. Dataset

In this paper, a previously simulated fraud dataset generated by Brandon Harris's Sparkov Data Generation tool [10] is utilized to deal with the inability to acquire real world transactions for use in data analysis. It covers the transactions of 1000 synthetic credit card users,

performing transactions within a pool of 800 merchants, split across two datasets—a training dataset of 1,296,675 transactions, and a testing dataset of 555,719 transactions.

Each transaction contains information on the transaction's date and time, the credit card number of the customer, the merchant's name, category and location in longitude and latitude, the customer's name split between first and last name, the customer's gender, the customer's location split between state, zip, latitude, and longitude, the customer's city population, job, and date of birth, a transaction number, and a target class that informs whether or not a transaction is fraudulent.

3.2. Preparation of the Dataset

To parse and perform operations on the data, the credit card fraud dataset's testing and training portions were input into the scikit-learn Python library [11] as separate data frames. To prepare data for clustering, we have to scale forms of non-continuous non-quantitative data so an appropriate distance measurement can be obtained. The qualitative data of the dataset includes the date and time of a transaction, identifiers of the merchant through name and category, and identifiers of the credit card holder through first/lane name, gender, street address, and job. These features provide characteristic data of the individuals involved within a given transaction. As this data cannot be scaled for clustering without holding a numeric value, the LabelEncoder from the scikit-learn Python library is used to force the information to be numeric, in a range from 0 to *n*-1, with n being the number of categories..

The data also contains various quantitative features, which can be divided based upon their properties of either being continuous or discrete. The index of the transaction, as well as identifiers such as the transaction and credit are number, city population, and ZIP code of the credit card holder are discrete values. In contrast, the amount of the transaction, split latitude and longitude data for the credit card holder and merchant, the city population of the credit card holder, and the UNIX time of the transaction are all considered continuous features. For the sake of this research, both discrete and continuous values are treated the same in terms of scaling.

For better parsing, scikit-learn's RobustScaler was utilized on each of the values, removing the median and scaling based on the quartile range for later transformation. This standardization of the dataset allows for easier interpretation of the dataset, as well as providing values that can be interpreted for distance.

3.3. Experimental Process

For the sake of testing the hybrid learning model, a decision tree was created based on the previously defined features based upon the training set, and its feature importance values $S_1 \dots S_r$ as defined in (2) are interpreted to generate list of weights that could be used for a weighted K-means clustering algorithm. Each of the weights derived from the feature importance are interpreted through their range of 0 (not relevant) to 1 (relevant), Features below a given weight of 0.001 are cut from the clustering analysis due to a lack of relevance.

Each of these weights are then applied to a k-means clustering algorithm modified to utilize the weights in a weighted Euclidean distance algorithm, applied to the split test dataset. For example, given a feature that holds a value of 6 and a centroid that holds the value of 4, a standard Euclidean distance calculation of $\sqrt{(4-2)^2}$ would lead to the result of 2. However, the addition of a given weight 0.5 would modify the distance to be $\sqrt{0.5(4-2)^2}$, or about 1.414. Values that hold less significance will hold a shorter distance from given clusters, lessening their impact on determining which cluster a given datapoint will belong to.

Two clusters are generated for the sake of the experiment—one being defined as "non-fraud", and another "fraud". It is assumed that data will tend to lean towards being non-fraud, so the cluster that holds the most data will be interpreted as non-fraud, and the one with the lower portion of data will be interpreted as potential fraud data. Data that is accurately found in the "non-fraud" cluster is counted towards positive accuracy, which is reported in comparison towards a non-weighted K-means clustering using the same dataset.

Following previous research that analyzed credit card fraud [2], we utilized the following metrics to determine the predictive ability of our hybrid classifier on the target class of credit card fraud:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2\,\frac{Recall * Precision}{Recall + Precision} \tag{4}$$

TP, or true positive, refers to an instance of accurately predicted instances credit card fraud—in contrast, FP, or false positive, refers to an incorrect prediction of a fraud. TN and FN refer to true negative and false negative, and provide similar classifiers for the negative class, in this instance being a non-fraud transaction.

A high precision score ensures that fraudulent detections are not being detected on nonfraudulent data, while a high

recall score ensures a low risk of fraudulent transactions getting through as nonfraudulent. The F1 score gives the harmonic mean of both values. We prioritize for our hybrid classification model to have a low recall score, as it is assumed that false detections of fraud would have lower impact on a customer than fraudulent activity going undetected.

3.4. Results

Through using the decision tree's feature importance statistics, a list of weights was found as follows, rounded to the third decimal place: [0.002, 0.346, 0.061, 0.068, 0.058, 0.096, 0.172, 0.100, 0.097], corresponding to the credit card number, transaction amount, ZIP code, latitude and longitude of customer, the city population, the UNIX time, and the merchant latitude and longitude. Some features had less significance in terms of affecting how a decision tree would interpret results, but four out of the nine features hold a significance relevance of at least 0.10. The decision tree itself held an accuracy of 95.78% when trained upon the training set.

Initial non-weighted K-means clustering achieved an accuracy of 87.92%, and later weighted K-means clustering produced an improved accuracy of 98.51%. This change suggests significance in adding weights to the features of the dataset. Further analysis would be needed to interpret whether similar accuracy increases could be replicated in similar datasets, and if the arbitrary weights used hold any significance due to the large number of elements that were deemed unfitting to hold any significant weight.

When analyzed, the model achieved a precision score of 0.740, a recall score of 0.895, and a F1 score of 0.810. Compared to previous research, our hybrid model did not reach equivalent scores to the hybrid models previously published in [6], containing scores for these three statistics for K-means based models at around 99%. However, in comparison to the scores calculated for DISSFCM in [2], both abilities to prevent false alarms have been improved, with only the precision having instances of being lower than the highest values of around 75%. Due to differing circumstances across studies with varying artificial datasets, there is challenge in providing direct comparison between studies. Despite this, the hybrid model was able to achieve positive results in accurately determining fraud.

4. Proposed Hybrid Soft Clustering Model

4.1. Motivation

Although previous results may suggest further investigation into hard clustering methods, "hard" clustering methods like K-means force results into solid conclusions. This may not be feasible for a financial institution, as acting on the results of the machine learning may over-emphasize false positives. Meanwhile, financial institutions need to adjust their predictions according to the current economic situation and they need their models to be interpretable [12]. Therefore, there is motivation to instead investigate potential soft clustering algorithms, as their percentage can be more easily interpreted and acted upon based on its result.

However, there are challenges with soft clustering algorithms that make them difficult for real-world use in the banking industry. If ensemble methods are used for generating percentages of a transaction, they can become too complex to be easily interpreted for real-world applications [2]. This leaves room in modern data mining research to make an easily interpretable model that is accurate while simultaneously being readable enough for real-world applications.

4.2. Model Proposal

A potential soft clustering model for credit card fraud detection could be derived based on the proposed hybrid method presented in Section 3. For this model to work, a set of training data and a set of testing data are needed. Then the model can be built in the following steps.

1. Use the set of training data to build the hybrid learning model presented in Section 3. After this step, the training data should be clustered into k clusters.

2. For any datapoint in the set of testing data, compute the weighted distance d_i between the datapoint and the centroid of the ith cluster for all $1 \leq i \leq k$.

3. The probability p_i for the datapoint to belong to the ith cluster is computed as:

$$p_i = 1 - \frac{d_i}{\sum_{i=1}^{k} d_i} \tag{5}$$

For example, for a trained model with 2 clusters (k=2), if the distance between a new datapoint and the centroid of the 1st cluster is 0, the probability for this datapoint to belong to the 1st cluster $p_1 = 1 - 0 = 1$. The probability for this datapoint to belong to the 2nd cluster $p_2 = 1 - \frac{d_2}{d_2} = 0$. If the distance between a new datapoint and the centroid of the 1st cluster is the same as the distance between the datapoint and the centroid of the 2nd cluster, the probability of for this datapoint to belong to either cluster is $1 - \frac{d_1}{2*d_1} = 0.5$.

5. Future Work

As noted within Section 4, we intend to explore soft clustering methods to further see if applying weights has any significant impact in improving results or having any use in credit card fraud analysis. Ideally, the simplified model will improve upon previously established soft clustering efforts, but further research will be necessary in order to investigate whether or not improvements or meaning can be found from this model. Using the hybrid soft clustering model, it may be possible through future research to detect a consistent threshold from a given non-fraud or fraud cluster that will provide accurate results across differing sets of credit card fraud data.

Despite the increase in accuracy, the hybrid learning model also leaves room for further analysis. As the features currently in place have low impact when modified with weights, it can be interpreted that a model that excluded these values would be little different than one that weights them minimally. Further research would be needed to determine this. Additionally, the inclusion of new features, such as interpreting the distance between the credit card holder's home and the location of their transaction, could possibly bring more useful features for more accurate data analysis.

Other limitations of the data set, such as the lack of real-world data, demand future compensation by further analysis upon other simulated credit card fraud datasets. These datasets may differ significantly in the information provided due to their source, which will require future standardization of features for future research to depend upon.

6. Conclusion

To work around the consequences of both unsupervised and supervised learning, we proposed two hybrid learning algorithms to try and accurately predict credit card fraud. By using the feature importance of a decision tree to create weights to apply to the distances of a modified K-means algorithm, we were able to improve an initial accuracy of 87.92% to one of 98.51%. This increase signifies some relevance in the use of weights, but further research is necessary to confirm whether this increase in accuracy holds any significant use for credit card anomaly detection. Furthermore, research is still necessary to determine the worth of the hybrid soft clustering model. The initial proposed algorithm does show promise in providing results, but future analysis will help to further elucidate its purpose and potential with aiding financial institutions in determining credit card fraud.

References:

[1] V.N. Dornadula, S. Geetha, Credit card fraud detection using machine learning algorithms, *Procedia Comput. Sci.*, 165, 2019, 631–641.

[2] G. Casalino, G. Castellano, & C. Mencar, Credit card fraud detection by dynamic incremental semi-supervised fuzzy clustering, *Proceedings of the 2019 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology (EUSFLAT 2019)*, 2019, 198-204.

[3] J. O. Awoyemi, A. O. Adetunmbi & S. A. Oluwadare, "Credit card fraud detection using machine learning techniques: A comparative analysis," *2017 International Conference on Computing Networking and Informatics (ICCNI)*, Lagos, Nigeria, 2017, 1-9.

[4] S. Khatri, A. Arora & A. P. Agrawal, Supervised Machine Learning Algorithms for Credit Card Fraud Detection: A Comparison, *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Noida, India, 2020, 680-683.

[5] T. Behera, S. Panigrahi, Credit Card Fraud Detection: A Hybrid Approach Using Fuzzy Clustering & Neural Network, *2015 Second International Conference on Advances in Computing and Communication Engineering (ICACCE)*, 494-499.

[6] S. Jain, N. Verma, R. Ahmed, A. Tayal, & H. Rathore, Credit card fraud detection using K-means combined with supervised learning, *Hybrid Intelligent Systems*, 420, 2022, 262–272.

[7] A.O. Adewumi, A.A. Akinyelu, A survey of machine-learning and nature-inspired based credit card fraud detection techniques. *Int J Syst Assur Eng Manag 8 (Suppl 2)*, 937–953 (2017). https://doi.org/10.1007/s13198-016-0551-y

[8] C. C. Aggarwal, *Data Mining: The textbook* (New York City, NY: Springer Cham, 2015).

[9] E. H. Ruspini, J. C. Bezdek & J. M. Keller, Fuzzy Clustering: A Historical Perspective, *IEEE Computational Intelligence Magazine*, 14(1), 2019, 45-55

[10] K. Shenoy, Credit Card Transactions Fraud Dataset, *Kaggle*. [Online]. Available: https://www.kaggle.com/datasets/kartik2112/fraud-detection.

[11] F. Pedregosa et al., Scikit-learn: Machine Learning in Python, *JMLR 12*, 2011, 2825-2830.

[12] A. Ghosal, A. Nandy, A.K. Das, S. Goswami & M. Panday, A Short Review on Different Clustering Techniques and Their Applications. In: Mandal, J., Bhattacharya, D. (eds) *Emerging Technology in Modelling and Graphics. Advances in Intelligent Systems and Computing*, 937, Springer, Singapore.

UNIVERSITY CHATBOT: A JOURNEY INTO CLOUD-NATIVE DEVELOPMENT

Brian Montecinos-Velazquez, Dominic Pisano, Jared Miller, Harrison Kahl,
Nicholas Hines, Tyler Profitt, Dominic Spampinato, and Linh B. Ngo
West Chester University of Pennsylvania, Department of Computer Science
BM935325@wcupa.edu, DP963106@wcupa.edu, JM909119@wcupa.edu, HK869465@wcupa.edu,
NH906322@wcupa.edu, TP894110@wcupa.edu, DS946528@wcupa.edu, lngo@wcupa.edu

ABSTRACT

Machine learning (ML) chatbot applications are an increasingly popular enterprise solution. Research and development of ML chatbots can be restrictive due to high resource costs. The development of sophisticated chatbots can be facilitated through a cloud-based framework for deploying chatbot infrastructure. A Kubernetes framework for cloud computing education [1] is adapted for chatbot deployments. The framework emphasizes the utility of cloud computing concepts such as containerization, orchestration, and CI/CD (Continuous Integration/Continuous Delivery) for ML applications. Further, cloud security best-practices are followed in order to elevate deployments to industry-level operations. The University Chatbot is a prototype project demonstrating the capability of the framework. Through this chatbot, users are able to make queries in natural language regarding campus events. Chatbot deployments follow a full- stack workflow. A front-end component implements a natural language chat interface for user interaction. The back-end involves a ML chatbot application that can process user input and query a database populated with domain-specific information.

1 Introduction

Chatbots enable users to interact with services through natural language conversation. Such applications have become a ubiquitous solution for automating customer support in enterprise settings [2]. Typically, chatbots are limited to organization-specific queries. Level of sophistication can range from tree-based dialogue flow to the implementation of Machine Learning (ML) models for Natural Language Processing (NLP). Recent innovations have broadened the utility of chatbots to that of sophisticated, general-purpose assistants. One notable example is ChatGPT [3]. Within a few months of its introduction, ChatGPT has garnered attention for its potential to disrupt both academic and industry settings [4]. As further advancements are made, the integration of chatbots into day-to- day operations will become more critical to meet expectations. However, the development of sophisticated chatbots is prohibitive, given the complexity and resource intensity. This work presents an effort to address this issue through the design of a cloud-native framework for deploying chatbot infrastructure.

The University Chatbot originated as a learning opportunity into the field of cloud computing. Its motivation being to highlight the utility of cloud-native development for complex, full-stack applications, including ML chatbots. Development relied on cloud computing concepts such as containerization, container orchestration, and CI/CD (Continuous Integration/Continuous Delivery). Through this approach, we developed a scalable, secure, and accessible solution that can be re-conceptualized as an adaptable framework for future chatbot development.

The framework is based on a standard chatbot's workflow, with five components: a UI (User Interface), a NLP (Natural Language Processing) server, an Actions API server, a Database, and a web crawler for data extraction (Webscraper). A prototype for this framework was developed with a wide array of open-source tools and deployed over an academic cloud. An initial version of the framework allowed for each application component to be containerized through Docker and orchestrated within a Kubernetes cluster. This version had several drawbacks, including a lack of consideration for cloud security and manual configuration of cluster services. A second version was developed to address these concerns, introducing automated deployments through CI/CD pipeline integration and a suite of security features.

The remainder of this paper is organized as follows. Section 2 presents an overview of cloud computing concepts, chatbot development practices, and previous work related to cloud-native enablement of ML services. Section 3 describes the development environment, the design of the framework, and the development process of a prototype. Section 4 evaluates utility of the framework. Section 5 concludes the paper and discusses possibilities for future work.

2 Literature Survey

Over the past decade, the fields of cloud computing and ML have seen increasing, industry-wide adoption. Given this rise, a substantial amount of research has been carried out into each field independently and how they can interact. We present a literature survey in order to properly contextualize our work. The scope of our survey includes chatbot development practices, cloud computing concepts, and previous work

specific to cloud-native enablement of ML services.

2.1 Chatbot Development

Chatbots present users with a conversational interface where they can make domain-specific queries in natural language. According to [2], a ML chatbot's workflow is as follows. Users interact with the chatbot through a conversational UI. Their input is forwarded to the chatbot's NLP unit, which processes the input and crafts an appropriate response - as seen in Figure 1. First, a NLU (Natural Language Understanding) model parses out the message's intent and extracts entities. An intent maps to a set of predefined actions that a chatbot can execute. Entities are data points extracted to allow for actions to be more specific to user input. A dialogue manager maps the intent to an action within the chatbot's domain. Actions can be API calls or requests to external resources necessary to craft an appropriate response. A dedicated actions server can be deployed independently of the NLP unit to carry out requests. An external resource can include a database for storage of training data or records to be accessed by users. A Natural Language Generator uses data returned by an action to craft an appropriate response to the user in natural language. NLU model training can be resource intensive, both in computational power and storage capacity. Training datasets must be large and expansive enough to generate an effective model. Through the cloud, these costs can be alleviated.

2.2 Cloud-native Development

As global computing demand increases, so does the need for reliable, scalable cloud solutions. Through containerization, application workflows can be modularized into independent components handling distinct workloads. Docker is a leading, open-source container platform [5]. Within containers, applications are packaged into lightweight, portable units along with their dependencies and execution environment. This is valuable for applications with a large number of dependencies, such as ML projects. In contrast to hypervisor-based deployment methods, containers can be rapidly deployed across different host environments with minimal overhead. The efficiency of Docker containers is due to the layered approach for building container images. This approach saves computation effort and network bandwidth, as existing layers in an environment do not need to be rebuilt or loaded from a registry. Container portability is critical for dynamic cloud environments, where applications may need to be migrated or redeployed in varying infrastructure contexts.

As an application's complexity and workload increases, the management of many contianers becomes cumbersome. Kubernetes is an open-source, container orchestration system introduced in 2014 [6]. Since its introduction, Kubernetes has become a ubiquitous cloud solution due to its ability to automate the deployment, scaling, and management of containers [7]. Through Kubernetes, containers are hosted as pods within a cluster of compute nodes. This allows for features

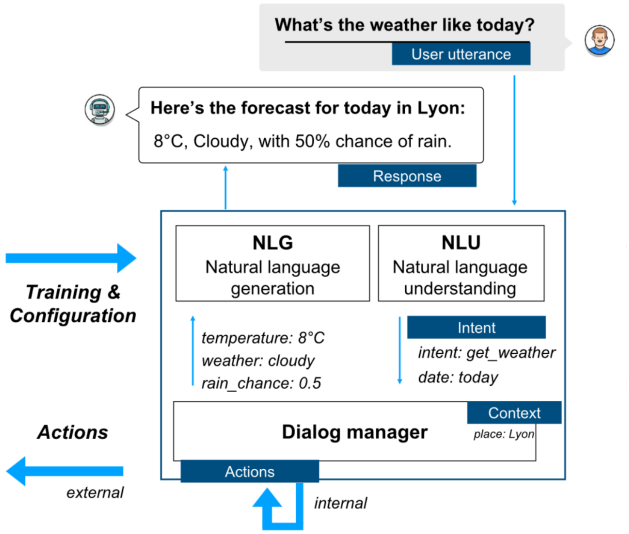

Figure 1: Diagram of chatbot NLP unit, borrowed from [2]

such as automated scaling, networking, and load balancing of resources. Pod replication allows for services to be scaled up or down to meet varying workload needs.

Application development lifecycles can require a set of repetitive tasks for building and deploying updates. Manual execution of these tasks can be a bottleneck in development efficiency. Through CI/CD pipelines, the building, testing, and deployment of applications can be automated. In a CI step, changes committed to a version control repository trigger the automated building and testing of code. This allows for builds to be validated prior to deployment, reducing the risk of errors causing service disruptions. Once validated, the build is passed to a CD step to automatically deploy across target environments. Automating these steps allows for more efficient delivery of application updates to end users. Jenkins is an open source automation server which can be configured to implement CI/CD pipelines [8]. A Jenkins server can be integrated within a Kubernetes cluster to automate container deployment tasks. By leveraging these concepts and tools, our framework enables the development of reliable applications.

Maintaining a secure cloud environment is an increasingly critical aspect of cloud-native development. For instance, chatbot operations may require users to provide or access sensitive information. Implementing security measures is necessary for maintaining reliable services. Securing chatbot services requires user authentication and end-to-end encryption of traffic [9]. On the infrastructure level, there exists an extensive set of best-practices to follow for Kubernetes deployments [10]. Role-Based Access Control (RBAC) ensures internal systems are only accessible to authorized users or services. In RBAC, the management of user privileges is abstracted through roles. Roles map to a set of system privileges. In industry settings, RBAC is further enhanced through Identity management, where users are thoroughly authenti-

cated prior to role assignment [11]. Access to the cluster can be further restricted through network-wide and container-specific security policies. Distinct workloads can be isolated into unique name spaces, or virtual sub-clusters, to safeguard critical resources. SSL/TLS support enables end-to-end encryption between pods for secure network traffic. The integration of CI/CD pipelines requires further security consideration, given the privileged level of access that pipelines require. According to [12], pipeline deployments should be authorized through RBAC policies. Sensitive configuration values can be secured behind a secrets manager. As part of the build validation, pipelines can automate vulnerability checking of applications by scanning dependencies. Implementing these security practices serves to reduce the attack surface of chatbot deployments through our framework.

2.3 Chatbots in the Cloud

Development of ML projects can be prohibitive due to their resource intensity, both computationally and storage-wise. Thus, cloud deployments are an attractive solution. Early proponents of cloud-based ML services highlight the utility of automated configurations and efficient resource scaling with hypervisor-based solutions [13], [14]. Later work introduced the implementation of containers to improve ML deployments. In [15], a Kubernetes platform for ML education is proposed as an improvement over hypervisor-based methods. Benefits cited include more rapid deployments and improved scalablility for large, multi-tenant operations. Further work serves to illustrate the industry-wide adoption of Docker- and Kubernetes-specific deployments for ML workloads [16], [17] - including work seeking to enhance the architecture [18]. [19] presents a study of emerging common practices for deploying ML projects through Docker. Considering the complexity of ML applications, CI/CD pipelines serve to make development life cycles more manageable. Even so, the unique needs of ML projects require an enhancement of existing CI/CD strategies, as illustrated in [20], [21].

As stated previously, ChatGPT is a recent example of chatbot technology which has received wide recognition for its advanced capabilities [4]. The exact infrastructure logistics for the development of ChatGPT are not publicly documented. In [22], OpenAI outlines that previous GPT models required a Kubernetes deployment scaled up to 7,500 nodes. Their workloads consist of dynamic ML projects which are updated frequently. Operations rely on Kubernetes automated scaling and networking utilities to allow rapid, steady operations [22]. Serving as a case study into the robustness of a Kubernetes deployments for chatbots, this example highlights the relevance of our work. We explore chatbot development at a smaller scale using industry-standard technologies and practices - including containerization, CI/CD, and security aspects. Our work elucidates the logistics of an increasingly important technology and provides a foundation for future study.

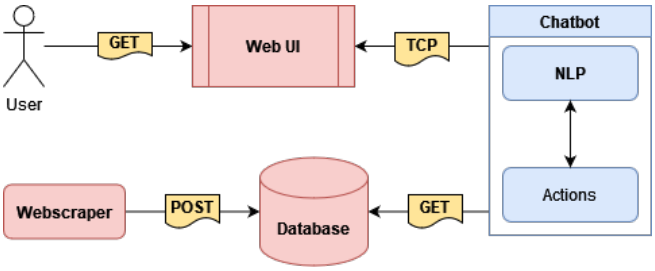

Figure 2: Chatbot workflow diagram

3 Development

A template of our framework is outlined in Figure 2. Considering a chatbot's typical workflow, our design features five components: a front-end Web User Interface (Web UI), a NLP server, an Actions server, a Database, and a Webscraper. A typical process flow involves a user making a query in natural language through the Web UI. This query is forwarded to the NLP server. Once the NLU unit has processed the intent behind the user's input, the NLP server makes a request to the Actions server. The Actions server carries out the request through API calls. In this example, it queries the Database and returns a set of records to the NLP server. The server can then craft and send a response back to the user through the Web UI. The Webscraper executes independently to populate the Database with data collected from domain-specific websites.

In order to display the utility of the framework, a prototype chatbot was developed [23]. The University Chatbot follows the workflow outlined above, allowing users to make domain-specific queries about university events. The implementation is summarized as follows. The Web UI is a standard chat interface built with JavaScript and Socket.IO. Site files are hosted in an Nginx container to handle web traffic. The NLP and Actions servers are both developed as part of the Rasa platform for chatbot development [24]. A MySQL database instance stores records accessible to the Actions server. The Webscraper is a Python script which populates the Database.

Several chatbot development tools exist to meet the diverse needs of enterprise settings. Major cloud vendors, such as Azure and AWS, provide their own ML bot services. These are typically expensive, managed services used to expedite the development process [25]. Selecting the appropriate tool to meet development needs involves several factors, including potential website integration, deployment methods, and pricing models [26]. Rasa is an open-source platform for chatbot development [24]. Through Rasa, the generation of training data is standardized for convenient NLU model creation. This involves defining sample data within a predefined structure of YAML files. An nlu.yaml file contains a list of intents with detailed examples of possible user messages. Regular expressions, lookup tables, and entity definitions can be used to create rich examples for training. Further, dialogue management is trained by defining sets of stories and rules. Stories outline

typical dialogue paths that a user may follow. Rules ensure the NLU model maps an appropriate response to certain intents. A domain.yaml file defines the scope of data to train the model with. Training data is passed through a training pipeline to output sophisticated NLU models A key benefit of Rasa is the availability of NLU pipeline templates and pre-existing training data sets. These features allow for a functional model to be trained small amount of domain-specific data. This preliminary model can be made accessible to users, allowing for realistic data to be collected to train improved models [27]. Such an iterative process fits a CI/CD life cycle, where incremental improvements are continuously delivered end users through a secure, automated pipeline. Rasa also maintains official Docker images for NLP and Actions servers, streamlining the process of containerization and orchestration.

The following sections detail the development environment and describe the status of the prototype at each version of the framework. Version 1 allowed for Kubernetes deployment but lacked CI/CD pipeline integration. Version 2 is a security-enhanced Kubernetes deployment with full CI/CD pipeline integration.

3.1 Environment

Our work extends an existing Kubernetes framework for cloud computing education [1]. This framework is designed to support complex, full-stack applications within a Kubernetes cluster. Projects within this framework are deployed on CloudLab, an academic cloud. Founded by the National Science Foundation in 2014, CloudLab was built to provide researchers with a robust cloud environment for computing research [28]. Within CloudLab, infrastructure is deployed as time-constrained experiments. Experiments are configured through a profile written in Python. Through this profile, startup scripts can be leveraged to configure a multi- node Kubernetes cluster within a provisioned experiment. The first version of this framework relied on Bash scripting to deploy a Jenkins server within the cluster [29]. Further configuration of Jenkins, required a manual step through the Jenkins UI at the start of each CloudLab experiment. A second version of the framework introduced automated configuration and deployment of the Jenkins server through Helm charts. With Helm, configuration values are stored in reusable YAML files to simplify deployment [30].

Further, this second version introduced several security features such as ingress control, TLS (Transport Layer Security) certification, and access control [31]. Security policies are defined to restrict ingress traffic. As outlined in Figure 3, an ingress controller is deployed within the cluster to route traffic to internal services and attach certifications. An Nginx server on the cluster's head node routes traffic from standard ports 80/443 for HTTP/HTTPS traffic to the ingress controller. This is necessary as the default port range for Kubernetes deployments is [30000, 32767], which may be restricted by user-end firewalls. The ingress controller attaches TLS

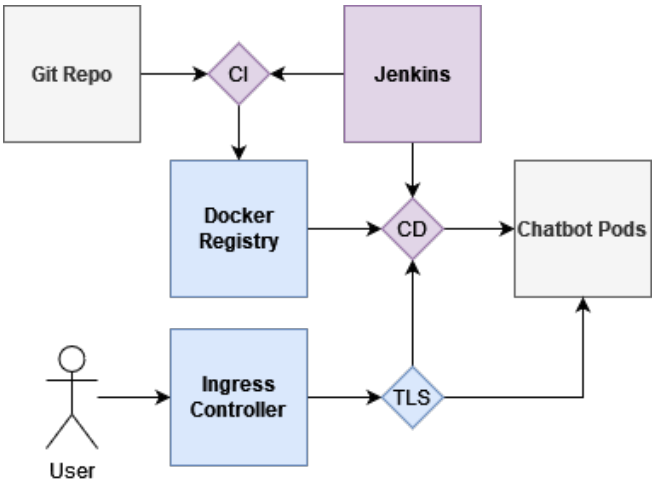

Figure 3: Security-enhanced framework

certificates to ingress objects, thus enabling HTTPS traffic. A certificate manager is deployed within the cluster to manage certificates signed by LetsEncrypt, an open certificate authority [32]. These verified certificates are made available to internal services and trusted by internet browsers on the user-end. Container images are stored in a private Docker registry. This eliminates the need to maintain a public, external registry through DockerHub, a popular Docker registry service. Deploying the private registry within the cluster follows security best practices, as it reduces ingress and egress traffic required for internal services. Lastly, RBAC authorization is enabled within the cluster to control access to resources. Particularly, the Jenkins server now requires a service account to authorize CI/CD pipeline execution over cluster resources.

3.2 Version 1: Kubernetes Deployment

At the end of the first phase of development, the prototype was incomplete [33]. While the back-end was functional for a base case's workflow, the front-end Web UI was not developed to a functional state. Each component was successfully containerized and deployable within the Kubernetes cluster. Although this version of the framework included Jenkins, the manual configuration step impeded the implementation of CI/CD pipelines. Instead, manual deployment of resources was enabled with Docker-Compose to build images and Bash scripting to deploy pods. The status of each component we outlined in Figure 2 is detailed as follows.

The Web UI was composed of an HTML file mimicking a university site with a popup chat interface. The development of a Socket.IO chat interface to enable communication between the site and chatbot was unsuccessful. Socket.IO was the preferred method due to built-in integration with Rasa and WebSocket encryption. This failure was attributed to the prioritization of the back-end workflow within the time constraint of this development period. The nonfunctional Web UI was deployed as an Nginx container made accessible to external

traffic directly through a NodePort protocol. Through this protocol, a port within the Kubernetes default range is exposed for external traffic.

The Chatbot was functional as a prototype. The NLP server was able to parse intent from user inputs and make the appropriate request to the Actions server.The Actions server was able to query the Database and return data to the NLP server. Considering that the Web UI was not functional, testing required running a Python script within the NLP server container to launch a command line chat interface. Success was measured based on the Chatbot's capability to handle a base case of user input requiring a request to the Database. For this base case, a user can request information on upcoming campus events in natural language. The NLP server parses out the intent, "event_search", and maps it to the appropriate action. In the Actions server, a python file defines a set of actions in the form of classes through the Rasa SDK (Software Development Kit). Each server is containerized separately and deployed jointly as a multi- container pod within the cluster. The NLP server trains an NLU model and configures connection to the Web UI. The Actions server is built with the Rasa SDK for Python. Rasa SDK simplifies the process of receiving requests from and sending replies to the NLP server. Beyond that, Python is used to carry out API calls needed to fulfill requests. For our use case, dependencies for MySQL database connectivity are loaded in the container. When the Actions server receives a request to perform an "event_search", a function queries the database for a set of records containing event information. This information is dispatched to the NLP server to be returned to the user. This base case was implemented successfully, but further functionality was not developed.

The Webscraper was composed of a Python script to populate the Database with data collected from specific university domains related to campus events. This script relied on several dependencies for web crawling, data extraction, and MySQL database management. The script was containerized and deployed as a Kubernetes job to execute a single time.

The Database was a standard Kubernetes deployment of MySQL. It stored university-related data accessible to the Actions server. The deployment pulled directly from the official MySQL image and required a persistent volume for storage within the Kubernetes cluster. Database credentials were hardcoded within configuration files.

3.3 Version 2: Security-enhanced Deployment with CI/CD

The second phase of our development with the security-enhanced framework is ongoing. Within the new framework, the development process of the Chatbot [23] has improved significantly. Critically, the use of Helm to automate the deployment the Jenkins server facilitated CI/CD integration. The development process now benefits from full CI/CD pipeline integration, making the continued development of the Chatbot more efficient. Development of individual components is streamlined. We follow a branch-based CI/CD integration, where separate pipelines deploy each component independently. This allows for component updates to be delivered without affecting the availability of other components. As outlined in Figure 3, the CI step of each pipeline is triggered when changes are committed to the specific branch of a Git repository. This step uses Docker commands to build new images and store them in the private Docker registry. Access to the registry is managed through signed certificates from the certificate manager. In the CD step, the pipeline pulls images from the private registry and deploys containers to the cluster. With this integration, several improvements have been made. Components now benefit from dynamic configuration through parameter substitution in pipeline runs. Within the Linux environment of the CloudLab experiment, this involves the Unix command, SED, for string replacement within configuration files. This simplifies the process of migrating, or redeploying, to new CloudLab experiments by removing the need to manually update hard-coded configuration values.

Component improvements are ongoing. A Socket.IO chat application for the Web UI is soon to be realized. The introduction of ingress control and TLS certification is a more secure and convenient solution compared to the NodePort protocol. While Socket.IO secures the communication between the Web UI and the NLP server, HTTPS ensures communication between the user's browser and the Web UI is secure. Moreover, the ability to define host names routed to each service through the ingress controller simplifies the internal networking architecture. Improvements to the Chatbot include entity extraction and updated training data within the NLP server. Deployment of the NLP and Actions containers will be separated to independent pods to improve service reliability. The Webscraper is being adapted into a continuous service to update the Database with new information from university sites or to repopulate the Database in case of severe service disruption. Access to the Database is secured through parameter substitution and secrets management. The application of RBAC for Database authorization rather than passwords is being explored. Feature ideas for improving the prototype include defining actions for querying granular event information, allowing for personalization of the Chatbot through user accounts, and expanding the Chatbot's domain to provide expanded university utilities. At the framework level, inspired by the difficulty in developing a Web UI component, future work can explore to development a modularized solution that can facilitate the integration of chatbot deployments into existing front-end sites.

4 Discussion

Development with the initial version of the framework was considerably slow. Early development focused on research into chatbot development patterns and cloud computing principles. The adoption of containerization facilitated the design process, as chatbot workloads were readily mapped to containerized components. The priority of this development

period was a successful Kubernetes deployment with a functional back-end workflow for basic use cases. Much of this development period over-relied on manual deployments with the help of Bash scripting and Docker-Compose. Configuration values were hard-coded into files, making redeployment across CloudLab experiments a tedious, error-prone process. The development of the NLP server, in particular, was hindered by lengthy model training steps prior to deployment. Integrating CI/CD practices early in this development period would have alleviated these issues.

The security-enhanced framework is a significant improvement over the initial version. Implementing automation services and security best practices allows for development to be in line with industry standards. This is the key contribution of our work. The framework provides a solid foundation for future work that integrates a variety of concepts critical to real-world applications. Although the prototype has not been developed to the level of a full-fledged chatbot solution, the experience serves to highlight the flaws and benefits of the framework. A containerized approach makes the framework extensible across environment contexts. Each component is interchangeable with alternate implementation technologies to meet differing needs. Kubernetes-based solutions for ML applications are a well-established standard [16], [17], [18]. This robustness is further enhanced by the consideration for security best-practices, both for general Kubernetes deployments and those specific to chatbots [10], [9]. The integration of CI/CD for chatbots, or ML projects in general, is an ongoing area of research [20]. Our current integration brought noticeable improvement to the development life cycle of the University Chatbot prototype. Future work would benefit from adapting pipelines to meet ML-specific workload needs, such as automated testing for NLU models [21].

5 Conclusion

The current framework provides a solid foundation for future research into cloud-native development of ML chatbots. Our framework integrates containerization, orchestration, CI/CD, and cloud security practices to inform an extendable process for chatbot deployment. The University Chatbot prototype developed demonstrates the utility of the framework for deploying reliable, adaptable, and secure chatbots. Further, as development continues, new requirements can inform reassessment. The framework can be expanded in to meet demands as they arise. At the current stage of development, future improvements to be made include, but are not limited to, the following:

- The inclusion of automated testing of NLU models through Jenkins CI/CD pipelines.
- The expansion of RBAC authorization to specify granular authorization for internal service interactions.
- The development of a context-agnostic, front-end solution that can bridge the integration of chatbot deployments into existing front-end sites.

Such improvements would extend the utility and adaptability of our framework for research and, potentially, industry-level needs. Key to our work is the availability provided by the use of open-source tools over an academic cloud provider. As the prominence of chatbot technologies continues to rise, the need for accessible, robust options for research and education will also rise.

References

[1] A. Reppert, B. Montecinos-Velazquez, H. Kahl, R. Reid, D. Rivas, D. Spampinato, H. Zhong, L. B. Ngo, A kubernetes framework for learning cloud native development, *Journal of Computing Sciences in Colleges*.

[2] M. Baez, F. Daniel, F. Casati, B. Benatallah, Chatbot integration in few patterns, *IEEE Internet Computing*, *25*(03):(2021), 52–59.

[3] Chatgpt: Optimizing language models for dialogue, 2023.

[4] B. Guo, X. Zhang, Z. Wang, M. Jiang, J. Nie, Y. Ding, J. Yue, Y. Wu, How close is chatgpt to human experts? comparison corpus, evaluation, and detection, 2023.

[5] Docker documentation, https://docker.com/, 2023.

[6] Kubernetes documentation, https://kubernetes.io/.

[7] C. Miyachi, The rise of kubernetes, in *2021 Cloud Continuum*, pages 1–5 (IEEE Computer Society, Los Alamitos, CA, USA, 2021).

[8] Jenkins, https://www.jenkins.io/, 2014.

[9] A. Eltahir, H. Abdulla, J. Platos, V. Snasel, Review of chatbot security systems, in *2022 26th International Conference on Circuits, Systems, Communications and Computers (CSCC)*, pages 167–178 (IEEE Computer Society, Los Alamitos, CA, USA, 2022).

[10] M. I. Shamim, F. A. Bhuiyan, A. Rahman, Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices, in *2020 IEEE Secure Development (SecDev)*, pages 58–64 (IEEE Computer Society, Los Alamitos, CA, USA, 2020).

[11] A. W. Services, Policies and permissions in iam, https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

[12] P. Bajpai, A. Lewis, Secure development workflows in ci/cd pipelines, in *2022 IEEE Secure Development Conference (SecDev)*, pages 65–66 (IEEE Computer Society, Los Alamitos, CA, USA, 2022).

[13] Y. Takabe, M. Uehara, Rapid deployment for machine learning in educational cloud, in *2013 16th International Conference on Network-Based Information Systems (NBiS)*, pages 372–376 (IEEE Computer Society, Los Alamitos, CA, USA, 2013).

[14] L. Padro, J. Turmo, Textserver: Cloud-based multilingual natural language processing, in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*,

pages 1636–1639 (IEEE Computer Society, Los Alamitos, CA, USA, 2015).

[15] C.-H. Lee, Z. Li, X. Lu, T. Chen, S. Yang, C. Wu, Multi-tenant machine learning platform based on kubernetes, in *Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence*, ICCAI '20, page 5–12 (Association for Computing Machinery, New York, NY, USA, 2020).

[16] Y. Huang, K. cai, R. Zong, Y. Mao, Design and implementation of an edge computing platform architecture using docker and kubernetes for machine learning, in *Proceedings of the 3rd International Conference on High Performance Compilation, Computing and Communications*, HP3C '19, page 29–32 (Association for Computing Machinery, New York, NY, USA, 2019), ISBN 9781450366380.

[17] D. Buchaca, J. Berral, C. Wang, A. Youssef, Proactive container auto-scaling for cloud native machine learning services, in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 475–479 (IEEE Computer Society, Los Alamitos, CA, USA, 2020).

[18] A. Kanso, E. Palencia, K. Patra, J. Shan, M. Chao, X. Wei, T. Cai, K. Chen, S. Qiao, Designing a kubernetes operator for machine learning applications, in *Proceedings of the Seventh International Workshop on Container Technologies and Container Clouds*, WoC '21, page 7–12 (Association for Computing Machinery, New York, NY, USA, 2021).

[19] M. Openja, F. Majidi, F. Khomh, B. Chembakottu, H. Li, Studying the practices of deploying machine learning projects on docker, in *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, EASE '22, page 190–200 (Association for Computing Machinery, New York, NY, USA, 2022).

[20] S. Garg, P. Pundir, G. Rathee, P. Gupta, S. Garg, S. Ahlawat, On continuous integration / continuous delivery for automated deployment of machine learning models using mlops, in *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 25–28 (IEEE Computer Society, Los Alamitos, CA, USA, 2021).

[21] B. Karlaš, M. Interlandi, C. Renggli, W. Wu, C. Zhang, D. Mukunthu Iyappan Babu, J. Edwards, C. Lauren, A. Xu, M. Weimer, Building continuous integration services for machine learning, in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 2407–2415 (Association for Computing Machinery, New York, NY, USA, 2020).

[22] B. Chess, E. Sigler, Scaling kubernetes to 7,500 nodes, 2021.

[23] B. Montecinos-Velazquez, University Chatbot, https://github.com/bmv0161/csc603-project.git/, 2022.

[24] Rasa documentation, https://rasa.com/, 2023.

[25] S. Packowski, A. Lakhana, Using ibm watson cloud services to build natural language processing solutions to leverage chat tools, in *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*, CASCON '17, page 211–218 (IBM Corp., USA, 2017).

[26] S. Perez-Soler, S. Juarez-Puerta, E. Guerra, J. de Lara, Choosing a chatbot development tool, *IEEE Software*, *38*(04):(2021), 94–103.

[27] K. N. Lam, N. N. Le, J. Kalita, Building a chatbot on a closed domain using rasa, in *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, NLPIR 2020, page 144–148 (Association for Computing Machinery, New York, NY, USA, 2021).

[28] R. Ricci, E. Eide, C. Team, Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications, *; login:: the magazine of USENIX & SAGE*, *39*(6):(2014), 36–38.

[29] L. B. Ngo, Introduction to cloud computing, https://github.com/class-master/csc468, 2022.

[30] Helm documentation, https://helm.sh/docs/.

[31] Kubernetes framework: Containerized registry, https://github.com/AustinMReppert/csc603cloud/tree/k8s-helm/, 2022.

[32] Let's Encrypt, https://letsencrypt.org, 2022.

[33] anonymous, University Chatbot, https://github.com/bmv0161/csc468-final/, 2022.

MalIoT: Scalable and Real-time Malware Traffic Detection for IoT Networks

Ethan Weitkamp, Yusuke Satani, Adam Omundsen, Jingwen Wang, Peilong Li
Elizabethtown College, Computer Science Department
weitkampe, sataniy, omundsena, wangjingwen, lip @etown.edu

ABSTRACT

The machine learning approach is vital in Internet of Things (IoT) malware traffic detection due to its ability to keep pace with the ever-evolving nature of malware. Machine learning algorithms can quickly and accurately analyze the vast amount of data produced by IoT devices, allowing for the real-time identification of malicious network traffic. The system can handle the exponential growth of IoT devices thanks to the usage of distributed systems like Apache Kafka and Apache Spark, and Intel's oneAPI software stack accelerates model inference speed, making it a useful tool for real-time malware traffic detection. These technologies work together to create a system that can give scalable performance and high accuracy, making it a crucial tool for defending against cyber threats in smart communities and medical institutions.

1 Introduction

Since the emergence of the Internet of Things (IoT), the problem of malware attacks on IoT devices has remained a constant problem. Technology improvements have caused the difficulty to increase over time, both in terms of the volume and the variety of the attacks [1]. According to research, malicious attacks on IoT devices have significantly increased recently. Zscaler's research shows that during the pandemic in 2022, malware attacks on IoT devices linked to business networks have risen by 700% [2]. The development of the internet, social networks, smartphones, and IoT devices have allowed bad actors to produce malware that is more advanced than ever before.

To protect against online attacks and maintain network stability, real-time and scalable malware detection solutions are required to identify and stop malware traffic in IoT networks. Due to its capacity to automatically detect and react to emerging malware threats in real-time, machine learning (ML) is a key strategy in IoT malware traffic detection [3], [4], [5]. ML models can swiftly scan this data to identify and prevent malware activity because IoT devices create enormous amounts of data that make it challenging to detect malware using conventional approaches. Because ML techniques are flexible and adaptable, it is possible to continuously upgrade the models to detect new varieties of malware as they appear. Additionally, ML models have the capacity to learn from enormous volumes of data and recognize patterns in the data that can point to malware activity, enabling them to detect malware of previously unidentified forms.

However, current machine learning-based solutions for detecting malware in network traffic often struggle to handle the growing number of IoT devices and detect malicious traffic with low latency. To address these issues, this paper proposes a scalable end-to-end network traffic analysis system that can detect malware in real-time. The system utilizes distributed systems such as Apache Kafka and Apache Spark, allowing for efficient scalability as the number of IoT devices grows. Furthermore, the use of Intel's oneAPI software stack for both machine learning and deep learning models has been shown to improve the inference speed of the model threefold. Specifically, there are three main contributions to this paper. (1) We seek to overcome the class imbalance and model over-fitting issues from the prior works by enriching the "IoT-23" dataset and adding a more diversified dataset named "ToN_IoT". (2) We accelerate the malware traffic detection inference speed with the use of Intel's oneAPI software. (3) We boost the system scalability and responsiveness by implementing a big data platform that includes Apache Kafka as the streaming engine and Apache Spark as the data processor.

The organization of this paper is as follows. Section §2 summarizes the previous works on IoT malware detection. Section §3 depicts the overall design of the data pipeline. Section §4 evaluates the performance of the methods. Lastly, Section §5 concludes the paper.

2 Related Works

One of the traditional approaches for detecting malware traffic is through the use of signature-based detection methods [6], [7], which use a database of known malware signatures to identify and block traffic that matches these signatures. While this method can be effective in detecting known malware, one of its biggest limitations is its incapability to detect unknown attacks, and it does not cover the detection of large-scale attacks.

To solve this problem, network behavior-based detection is

proposed for malware traffic detection [8]. Behavioral modeling methods focus on identifying and blocking malicious traffic based on its behavior rather than its signature [9], [10], [11]. This approach is more effective at detecting new or unknown malware as it is not reliant on a database of known malware signatures. However, it can be more resource-intensive and may produce false positives. Most existing studies in this area have limited scope, focusing on specific malware types, such as Bots, or on particular attack types, such as DoS, and anomalies in specific protocols or network layers [12]. Another limitation of traditional detection solutions is that the conventional solution depends on one network premises; this cannot detect attacks that originated at different network premises. Another problem is there is the case that devices might still operate for a long time, even after infection.

Another technique used in malware traffic detection is sandboxing, which involves creating a virtual environment in which to analyze and test suspected malware [13], [14]. This allows analysts to safely observe the behavior of the malware and determine its potential threats.

Machine learning methods are introduced in the previous works for malware traffic detection [15]. Kaluphahana et al. [16] proposed Adept, a security framework that detects bots attack and classifies them into attack stages across space and time. Adept utilizes alert-level and pattern-level information to classify the type and stages of attacks into categorical classification by using machine learning models, k-nearest neighbor, random forest, and support vector machine. Moreover, an end-to-end monitoring system RTC was proposed to identify new threats by manually extracting features from different protocols and network layer traffic data [17]. However, machine learning methods rely heavily on manual feature extraction, which is a time-consuming and resource-intensive process.

Since deep learning methods are efficient in feature extractions and automatic learning, the research on malware traffic attack detection is shifting from machine learning to deep learning [18], [19], [20], [21]. Sahu et al. [22] present a security framework for IoT attack detection using a hybrid Deep Learning model with two stages. Specifically, a CNN model first learns the features from the IoT network traffic, then the feature representation generated from the previous step is used as the input of an LSTM model for attack detection. However, whenever the network is scaled up by adding additional IoT devices, then an additional CNN module should be accompanied by the connected network switch.

Table 1 shows the performance of machine learning and deep learning models in classifying large-scale offensive accesses within IoT systems. Goyal et al. [23] observe that SVM and ANN models generate similar accuracy scores, while the precision score of ANN model outperforms SVM by 4.3%. Long-Short-Term Memory (LSTM) is used to determine whether an attack or a benign attack can be detected from a relatively small amount of data. Liang et al. [24] conclude that the SVM model has an accuracy score of 88.18%, but the recall and F1 scores are below 50. On the other hand, the LSTM model has an accuracy score of 99.98%, and both recall and F1 are close to 100.

Table 1: Model Accuracy in Surveys

Reference	Model	Accuracy	Precision	Recall	F1
Goyal et al. [23]	ANN	99.74	95.99	100	97.95
	SVM	99.86	91.98	100	95.82
Liang et al. [24]	LSTM	-	99.98	100	99.99
	SVM	-	88.18	45.43	59.97

3 Design

In this section, we present the design of the MalIoT system by describing the overall data pipeline, data collection and generation, offline model training, and online model inference.

3.1 Overview to the Data Pipeline

The overall data pipeline is illustrated in Fig. 1. The information originates from IoT devices within a local network, which transmits their network activity through a smart gateway that houses the subsequent steps in the process. Initially, the network activity is intercepted by a software sniffer Bro on the smart gateway, which generates PCAP files from the received data. These PCAP files are subsequently dispatched to a Kafka producer, which transmits the data to a Kafka topic. The Kafka producer ensures that the pipeline can be scaled to accommodate additional IoT devices on the network. The data is then ingested from the Kafka topic using Spark streaming, which maintains the pipeline's scalability for any number of IoT devices. The data can then be utilized for the purposes of retraining ML or DL models, or for conducting an online inference on the network activity using previously trained ML or DL models.

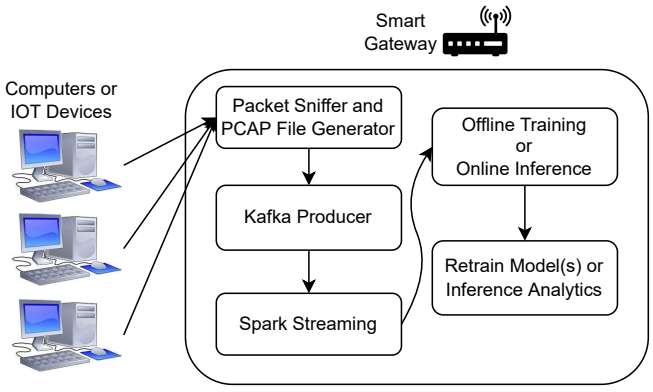

Figure 1: Overall Design Pipeline

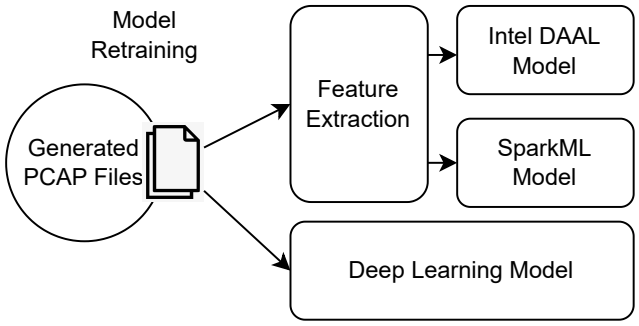

Figure 2: Model Training Process

3.2 Data Generation

To train various models, we need an initial dataset and a method to retrain the models using fresh data to adjust to new types of attacks. The retraining process is illustrated in Figure 2. When PCAP files are produced from the packet sniffer, they can subsequently be used to retrain the models with the real-time network traffic on the network. We employed the IoT-23 Dataset and TON_IoT Dataset as our starting datasets for training the original models. We tested the accuracy of machine learning and deep learning models using various feature subsets provided by the dataset and opted to use most of the features, excluding the host and recipient IP addresses and ports. These attributes didn't boost model accuracy, as they should not be linked with a malicious or benign attack and would merely overfit our models. Once the foundational models are established using the existing dataset, they can be retrained as needed using the newly generated PCAP files. For machine learning models, the desired features must be extracted from the PCAP files before retraining, as well as any other data preprocessing procedures.

Datasets: Aposemat IoT-23 datasets [25] are captured from the network traffic of Internet of Things(IoT) devices. The dataset was first released in January 2020 and covers captures from 2018 to 2019. The IoT-23 datasets include 20 instances of malware captured from IoT devices, along with 3 instances of benign IoT device traffic.

ToN_IoT datasets [26] are new generations of Internet of Things (IoT) and Industrial IoT (IIoT) datasets captured from a realistic and large-scale network with heterogeneous data sources. It consists of datasets from IoT and IIoT sensors, Operating systems datasets of Windows 7 and 10, Operating systems datasets of Ubuntu 14 and 18 TLS, and Network traffic datasets.

The basic information of each dataset is listed in Table 2. IoT-23 datasets consist of 325,307,990 captures from different IoT network traffics, including 294,449,255 malware captures executed in infected IoT devices and 30,858,735 captures from benign IoT device traffic. In the ToN_IoT datasets, the Windows 7 dataset has 28366 records and 132 features, the Windows 10 dataset consists of 35,975 records and 124 features, and the Network dataset has 21,978,632 records and 42 features. To generate the final synthesized training dataset, we integrate TON_IoT and IoT-23 together and then supplement the synthesized dataset with extra benign traffic from the CIC IoT datasets [27] to balance the classification categories.

Table 2: Dataset Information

Dataset	IoT-23	ToN_IoT
Benign	30,858,735	300,000
Anomaly	294,449,255	161,043
Total	325,307,990	461.043
# of features	23	45
# of IoT devices	3	9

Features: The common features shared in between the two datasets are listed in Table 3. In the experiments, we utilized two types of feature selection for our machine learning models: 1) the "Full Feature Set", which contains all features except for timestamp and unique identifier, and 2) the "De-identified Feature Set", which removes the TCP/UDP IP address and port information of both the originating and responding endpoints. We removed this information to avoid potential biases in the machine learning models when classifying malicious traffic based on IP addresses and ports.

Table 3: IoT-23 and TON_IoT Features Selection

IoT-23 Feature	TON_IoT Feature	Description
ts	ts	Timestamp of connection
id.orig_h	src_ip	originator's IP address
id.orig_p	src_port	originator's TCP/UDP port
id.resp_h	dst_ip	responder's IP address
id.resp_p	dst_port	responder's TCP/UDP port
proto	proto	transport layer protocol of connection
service	service	service
duration	duration	duration
orig_bytes	src_bytes	originator's payload bytes
resp_bytes	dst_bytes	responder's payload bytes
conn_state	conn_state	connection state
missed_bytes	missed_bytes	missed_bytes
orig_pkts	src_pkts	number of ORIG packets
orig_ip_bytes	src_ip_bytes	number of ORIG IP bytes
resp_pkts	dst_pkts	number of RESP packets
resp_ip_bytes	dst_ip_bytes	number of RESP IP bytes

3.3 Offline Model Training

As time progresses, new malicious attacks emerge and are designed. What was once considered secure must adapt to maintain its security in the present and future. This is especially true in the field of machine learning. To ensure our predictive models remain secure, they must be trained on more up-to-date data. To support this, we have integrated a means of retraining our various models with our own generated data. The data passing through the pipeline is saved in a format that enables easy retraining of our models at a later point in time. Therefore, we need not rely solely on existing datasets to keep our models up to date.

We evaluated five machine learning models and five deep learning models. The machine learning model classifiers include random forest, decision tree, logistic regression, linear SVC, and Gaussian-NB. The deep learning models consist

of artificial neural networks (ANN), 1D convolutional neural networks (1DCNN), two-dimensional CNN (2DCNN), long short-term memory (LSTM), and a combination of CNN and LSTM. Each model possesses unique strengths and benefits that prompted our decision to evaluate them. Regarding the hyperparameters of our deep learning models, we used ReLU as the activation function for all of the ANN and CNN models, except for the LSTM model, which uses 'tanh' as the activation function and sigmoid as the recurrent activation function. The ANN only uses one layer, while all of the other models use two layers. More details about our model hyperparameters can be found in Table 4.

Table 4: Deep Learning Hyper-Parameters

Parameter	Value
learning_rate	1e-3
decay_rate	1e-5
dropout_rate	0.5
dense_units	128
n_batch	100
n_epoch	1
filters	filters
kernel_size	4
strides	1
CNN_layers	2
clf_reg	1e-5

3.4 Online Inference

After the network traffic is converted into PCAP files and processed by Spark streaming, our previously trained model(s) can make an inference on the data. As mentioned previously, we have trained five types of machine learning models and five types of deep learning models.

Upon entering the Spark streaming stage of the pipeline, all new data is evaluated by one of the trained models based on the data's timestamp. Machine learning models generally require longer preprocessing times since feature extraction is necessary, but inference/prediction times are quick as these models are not very complex. On the other hand, deep learning models have less preprocessing time but longer inference times due to their complexity. These models tend to be more accurate on complex data, like that found in our pipeline. At present, the pipeline concludes once an inference is made on the network traffic, but the resulting inferences may be used as desired.

4 Evaluation

This section of the paper evaluates the performance of our proposed system. Specifically, we will talk about the specifications of the testbed, and the accuracy and timing performance of each machine learning and deep learning model.

4.1 Experiment Testbed

We summarize the system specifications and software versions for the testbed in Table 5. We use built-in Python tools in order to collect our data, like training time, inference time, accuracy, and CPU usage.

Table 5: Experiment Platform Specification

Item	Specification
CPU	Intel(R) Core(TM) i9-9920X CPU @ 3.50GHz
GPU	GTX 2080 Ti with 11 GB DDR6 Memory
Memory	64 GB DDR4 @ 3600MHz
Storage	2TB SSD
Host OS	Ubuntu 18.04 LTS
Tensorflow	2.7
Apache Spark	3.0.1
Apache Kafka	2.6.0
Accelerator	Intel DAAL v2020.1

4.2 Model Accuracy Comparison

We calculated the prediction accuracy from machine learning and deep learning models. In the machine learning inference section, we design five machine learning models (Random-forest, Decision-Trees, Logistic-Regression, Linear-SVC, and Gaussian-NB) and two feature sets, the full feature set, and the de-identified feature set.

Figure 3 shows the outcomes of machine learning models. In addition, Figure 4 shows the outcomes from five deep learning models, Artificial Neural Network, Convolutional Neural Network, Convolutional Neural Network_2D, Long-Short-Term-Memory, and a combination of Convolutional Neural Network and Long-Short-Term-Memory.

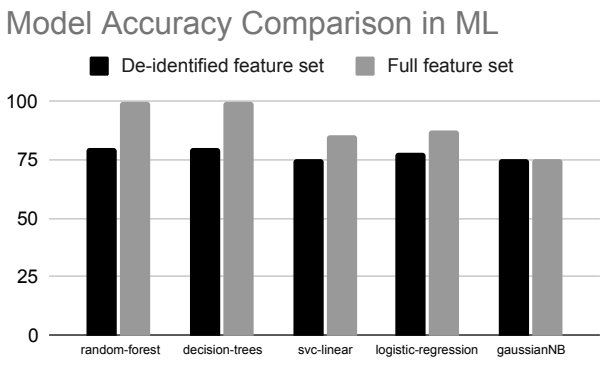

Figure 3: Model Accuracy Comparison in ML

In the machine learning experiments, generally full feature set marked a higher score. The first two models, random-forest and decision-trees, almost reached 100%. The score of the de-identified feature set is 75% on the whole.

In the deep learning section, the first three models were marked as almost 100%. LSTM and LSTM+CNN, on the other hand, scored about 75%.

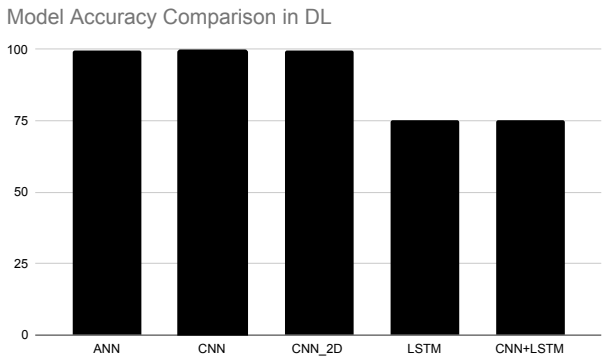

Figure 4: Model Accuracy Comparison in DL

Table 6: Inference Time for Machine Learning Models on Two Different Feature Sets

	De-identified feature set (ms)	**Full feature set (ms)**
Random-Forest	33.18	34.20
Decision-Trees	10.40	10.78
Logistic-Regression	10.30	10.71
SVC-Linear	10.30	10.68
Gaussian-NB	10.48	10.87

4.3 Inference Time Comparison

We measured the time of average time it takes for inference per each line from CSV files. As same with before experiment, we tested machine learning and deep learning. Figure 5 Shows the results from machine learning models, and Figure 6 Shows the results from deep learning models

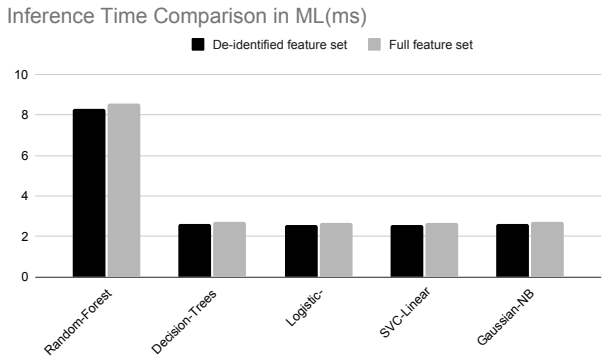

Figure 5: Inference Time Comparison in ML (ms)

This experiment shows that the random-forest model takes longer than other models. The reason seems that random-forest runs decision trees in parallel. The other models make inference per line around 2.6ms. In addition, the full feature set requires a little longer time to predict than the De-identified feature set.

Table 7: Inference Time for Deep Learning Models

	Time per row (ms)
ANN	0.056
CNN	0.118
CNN2D	0.102
LSTM	0.378
CNN+LSTM	0.226

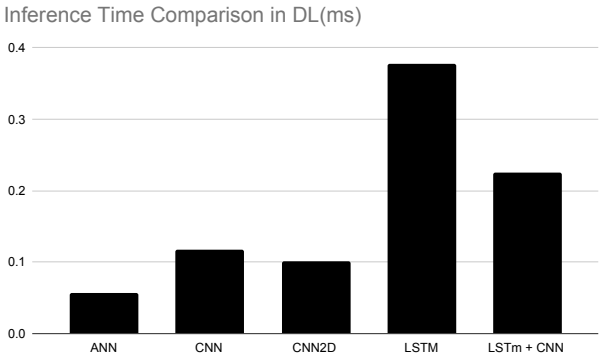

Figure 6: Inference Time Comparison in DL (ms)

This experiment shows that all deep learning models make inferences much faster than machine learning models. LSTM models take a little longer than other models, but only 0.4 ms per line

4.4 Scalability

Another key component of our research is scalability. We measured the change in inference time depending on the number of supporting devices. In the test, we added the number of devices from one to nine. Figure 7 Shows the outcome of the trial.

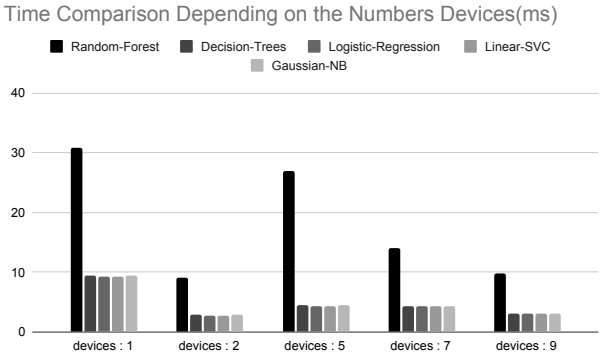

Figure 7: Time Comparison Depending on the Numbers Devices(ms)

The experiment shows the scalability of our model. Generally, the inference time per line decreases as the number of connected devices gradually increases. Random-forest takes longer to predict than other models, the same as the previous experiment.

5 Conclusion

In our project, we were able to achieve the objectives of real-time and scalable malicious network traffic detection by utilizing Apache Kafka producer and Apache Spark. Our models were trained on an enriched dataset that comprises IoT-23 and TON_IoT which contains millions of network flows with information on both malicious and benign network traffic. Upon preprocessing the data, we were able to perform inferences on it using the various models we developed. This process could be automated on a smart gateway to enable real-time detection on a local network. We have made the source code for this project available on GitHub for future research purposes. It can be accessed via the following link: https://github.com/BlueJayADAL/NetSec.

Acknowledgment

This work is supported in part by a grant from Intel Corporation, and a grant from the Summer Scholarship, Creative Arts and Research Projects (SCARP) Program of Elizabethtown College.

References

[1] D. Gibert, C. Mateu, J. Planes, The rise of machine learning for detection and classification of malware: Research developments, trends and challenges, *Journal of Network and Computer Applications*, *153*:(2020), 102526.

[2] Zscaler study confirms iot devices are a major source of security compromise, reinforces need for zero trust security.

[3] C. Jindal, C. Salls, H. Aghakhani, K. Long, C. Kruegel, G. Vigna, Neurlux: dynamic malware analysis without feature engineering, in *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 444–455 (2019).

[4] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, S. Venkatraman, Robust intelligent malware detection using deep learning, *IEEE Access*, *7*:(2019), 46717–46738.

[5] Y. Fang, Y. Gao, F. Jing, L. Zhang, Android malware familial classification based on dex file section features, *IEEE Access*, *8*:(2020), 10614–10627.

[6] S. Hajiheidari, K. Wakil, M. Badri, N. J. Navimipour, Intrusion detection systems in the internet of things: A comprehensive investigation, *Computer Networks*, *160*:(2019), 165–191, ISSN 1389-1286, doi:https://doi.org/10.1016/j.comnet.2019.05.014.

[7] M. Masdari, H. Khezri, A survey and taxonomy of the fuzzy signature-based intrusion detection systems, *Applied Soft Computing*, *92*:(2020), 106301.

[8] S. Nari, A. A. Ghorbani, Automated malware classification based on network behavior, *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 642–647.

[9] Ö. Aslan, M. Ozkan-Okay, D. Gupta, Intelligent behavior-based malware detection system on cloud computing environment, *IEEE Access*, *9*:(2021), 83252–83271.

[10] H. S. Galal, Y. B. Mahdy, M. A. Atiea, Behavior-based features model for malware detection, *Journal of Computer Virology and Hacking Techniques*, *12*(2):(2016), 59–67.

[11] W. Liu, P. Ren, K. Liu, H.-x. Duan, Behavior-based malware analysis and detection, pages 39–42.

[12] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. A. Kemmerer, C. Krügel, G. Vigna, Your botnet is my botnet: analysis of a botnet takeover, in *Conference on Computer and Communications Security* (2009).

[13] N. Miramirkhani, M. P. Appini, N. Nikiforakis, M. Polychronakis, Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts, in *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1009–1024 (IEEE, 2017).

[14] S. Liu, P. Feng, S. Wang, K. Sun, J. Cao, Enhancing malware analysis sandboxes with emulated user behavior, *Computers & Security*, *115*:(2022), 102613.

[15] O. Barut, M. Grohotolski, C. DiLeo, Y. Luo, P. Li, T. Zhang, Machine learning based malware detection on encrypted traffic: A comprehensive performance study, in *7th International Conference on Networking, Systems and Security*, 7th NSysS 2020, page 45–55 (Association for Computing Machinery, New York, NY, USA, 2020), ISBN 9781450389051, doi:10.1145/3428363.3428365.

[16] K. L. K. Sudheera, D. M. Divakaran, R. P. Singh, M. Gurusamy, Adept: Detection and identification of correlated attack stages in iot networks, *IEEE Internet of Things Journal*, *8*(8):(2021), 6591–6607.

[17] D. Bekerman, B. Shapira, L. Rokach, A. Bar, Unknown malware detection using network traffic classification, *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 134–142.

[18] O. Barut, Y. Luo, T. Zhang, W. Li, P. Li, Multi-task hierarchical learning based network traffic analytics, in *ICC 2021 - IEEE International Conference on Communications*, pages 1–6 (2021), doi:10.1109/ICC42927.2021.9500546.

[19] O. Barut, T. Zhang, P. Li, Weakly supervised learning for network traffic classification, in *NAS 2022 - IEEE International Conference on Networking, Architecture and Storage*, NAS '22, pages 1–4 (Philadelphia, PA, USA, 2022).

[20] O. Barut, Y. Luo, P. Li, T. Zhang, R1dit: Privacy-preserving malware traffic classification with attention-based neural networks, *IEEE Transactions on Network and Service Management*, pages 1–1, doi:10.1109/TNSM.2022.3211254.

[21] T.-L. Huoh, Y. Luo, P. Li, T. Zhang, Flow-based encrypted network traffic classification with graph neural networks, *IEEE Transactions on Network and Service Management*, pages 1–1, doi:10.1109/TNSM.2022.3227500.

[22] A. K. Sahu, S. Sharma, M. Tanveer, R. Raja, Internet of things attack detection using hybrid deep learning model, *Computer Communications*, *176*:(2021), 146–154, ISSN 1873703X, doi:10.1016/j.comcom.2021.05.024.

[23] M. Goyal, I. Sahoo, G. Geethakumari, Http botnet detection in iot devices using network traffic analysis, in *2019 International Conference on Recent Advances in Energy-efficient Computing and Communication (ICRAECC)*, pages 1–6 (IEEE, 2019).

[24] X. Liang, T. Znati, A long short-term memory enabled framework for ddos detection, in *2019 IEEE global communications conference (GLOBECOM)*, pages 1–6 (IEEE, 2019).

[25] S. Garcia, A. Parmisano, M. J. Erquiaga, Iot-23: A labeled dataset with malicious and benign iot network traffic (version 1.0.0) [data set], doi:http://doi.org/10.5281/zenodo.4743746.

[26] N. Moustafa, A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets, *Sustainable Cities and Society*, *72*:(2021), 102994.

[27] S. Dadkhah, H. Mahdikhani, P. K. Danso, A. Zohourian, K. A. Truong, A. A. Ghorbani, Towards the development of a realistic multidimensional iot profiling dataset, *2022 19th Annual International Conference on Privacy, Security & Trust (PST)*, pages 1–11.

MACHINE LEARNING TECHNIQUES TO IMPROVE USERS' MUSIC LISTENING EXPERIENCES

Samantha Noggle and Stephanie Schwartz
Millersville University
{sjnoggle, stephanie.schwartz}@millersville.edu

ABSTRACT

Recommender systems play a crucial role in delivering a personalized user experience. This is especially true for music streaming services like Spotify, which aim to curate music based on listeners' preferences to maintain engagement and stay ahead in a competitive market. This paper proposes a model designed to predict whether or not a user will skip a song based on information available within the user's current listening session. A dataset provided by Spotify for a machine learning challenge is utilized and augmented with features that we propose to help measure the "distance" between a song and songs that have been either listened to or skipped within the same session. We demonstrate that using our model, a prediction can be made on whether a user will skip the final song in their session that is 78.23% accurate. An analysis of feature importance is also presented to better understand which factors might be most influential in predicting this user behavior. The ability to predict whether or not a song will be skipped or played could be utilized in future work to dynamically impact song selection within a listening session to minimize the number of songs being skipped by the user.

1 Introduction

Music streaming platforms implement extensive and complex recommender systems in order to predict exactly what a user wants to listen to. Giving relevant recommendations is extremely important as it improves a user's listening experience and overall satisfaction. In a competitive market with many services vying for each user's time and money, the ability to accurately predict what an individual user wants becomes a critical factor in the success of a music streaming service such as Spotify, Apple Music, or YouTube Music. In this paper, we will focus on Spotify since we are utilizing a dataset provided by the service for a machine learning challenge [1].

Given the importance of its recommendations to its success, Spotify has naturally devoted considerable resources to creating its own multi-armed bandit model for content recommendation. A multi-armed bandit algorithm is a reinforcement learning technique named after the one-armed bandit slot machines at casinos [2]. The agent is tasked with multiple "arms" to pull or, rather, choices to make, each with unknown rewards. The "arms" in this case are different songs to recommend, and the rewards are user interactions with the song chosen, such as a "skip" or a "like". Spotify's model is called Bandits for Recommendations as Treatments (BaRT), and it uses three main functions to make the most accurate song suggestions for users: natural language processing (NLP), collaborative filtering, and raw audio analysis. Natural language processing is used to parse lyrics, user-created playlist names, and web-crawled data from media about a given song. This information helps to categorize tracks by their content and how listeners and critics perceive them. Collaborative filtering is a powerful technique that compares the listening history of similar users. So, if a user's listening history is very similar to user X, they will likely enjoy the tracks that user X has liked that they have not listened to yet.

Finally, Spotify analyzes the raw audio of each track by first generating a time-frequency representation of its audio frames. This is then used as input to a convolutional neural network (CNN) as seen in Figure 1. CNNs are typically used with visual data, but for this case, it is examining raw audio instead of pixels. From this model, audio features are output such as key, tempo, danceability, and loudness. This method provides a deep understanding of each song despite having a lack of user data in the case of new or unpopular music [3].

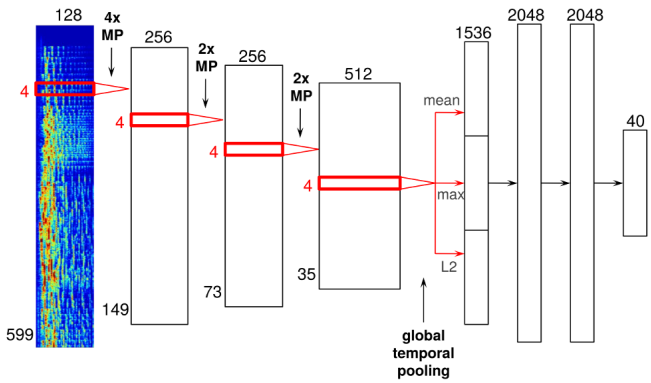

Figure 1: A convolutional neural network architecture used by Spotify

These techniques efficiently generate tailored playlists, personalize the home screen, and recommend new music.

When making a recommendation, systems often face the *exploration vs exploitation trade-off* [4]. That is, when a user faces a decision about what to listen to or watch next, they may opt to *explore* new content or *exploit* their existing knowledge of what they know and enjoy. This trade-off occurs at multiple levels in recommendation systems. For example, a system could opt to try to find out more about the user's preferences by presenting them with music the system is not sure they'll like (exploring the user's preferences) or the system could exploit its knowledge about the similarity of a song to others that the user already enjoys. A user could also explicitly opt into exploration by selecting "Discovery Mode" or they might decide to play a tried and true playlist of their own creation.

Given the nature of exploration, both on the part of the user and the system, it is natural that a user will be presented with some songs that they simply do not like and will opt to skip. However, an analysis of user listening behavior showed that an astounding 48.6% of songs on Spotify are skipped before being played until the end and 24.14% are skipped within the first five seconds [5]. This is, of course, more than just users skipping a song that they are unfamiliar with and do not enjoy. It is common for users to skip through the playlists that they have created themselves (and obviously already enjoy) to find what they are most in the mood for during a particular listening session. It is this level of behavior that we are interested in understanding. In this situation, the user's general music taste is irrelevant because they are listening to a collection they have already proclaimed to enjoy. But they are in a particular mood, skipping through songs that they like, but do not want to listen to at the moment. Given this behavior, the most relevant information (and therefore the information we wish to exploit) is limited to the specific session. Our research question, therefore, is whether we can use the session data to accurately predict whether or not a user will skip a particular song. If we can, then this ability could then be used to queue songs that the user actually wants to hear at that moment. Additionally, we wanted to gain a better understanding of which features are most important in guiding these predictions.

2 Related Work

Recommender systems for session-based data must use substantially less information than other methods. Instead of having a user's long-running history, only a few of their interactions with the platform are available. There is no user profile to pull information from. For the purpose of fine-tuning a shuffle algorithm on a session-by-session basis, it is ideal that only the current session's data is used for that decision.

Recurrent neural networks (RNN) are a successful and popular solution to the task of sequence-based predictions [6, 7]. Previous solutions involved item-to-item recommendations by simply recommending similar items, but an RNN is able to model an entire session and achieve greater success. For our purposes, we wanted to be able to have more transparency and insight into which features were most important in predicting whether or not a song would be skipped than a neural network would readily allow. For this reason, we did not use an RNN, though it is undoubtedly worth investigating in future work.

Many high-performing solutions to Spotify's Sequential Skip Challenge [1] (explained in section 4) utilized more than one model to make a prediction. Hansen et al. [8] achieved 2nd place in the challenge by using two RNNs. One RNN was tasked with encoding the first half of a session, which was used as input to a second RNN that makes predictions. Bères et al. [9] and Ferraro et al. [10] reached 10th and 14th place respectively by combining the output of multiple boosting trees.

3 Approach

In order to predict whether a listener will skip a song, a random forest classifier was trained to label each final song in a session as being skipped or not. Highly accurate solutions already exist [6–9, 11] that use a variety of deep learning methods however, random forests were chosen for this task because they allow the overall importance of each feature to be measured. Random forests also reduce overfitting by aggregating the predictions of multiple decision trees. Each decision tree in the ensemble uses a random set of features and a different subset of training data [12].

In 2019, Spotify in collaboration with The ACM International Conference on Web Search and Data Mining (WSDM) released a dataset of roughly 130 million listening sessions. It was released as part of a challenge hosted by AIcrowd [1] to predict if users will skip a track or not based on their behavior within the first half of a given session. During the challenge, the training data was comprised of the first half of the session and the test data (withheld from the participants) was the second half of the session. The challenge had ended before our research project but the training data was still available and useful for exploring our research questions. In the next section, we discuss the data in detail, including the preprocessing and feature engineering that we performed.

4 Data

Our dataset of 200,000 sessions was a subset of the training data from the Spotify Sequential Skip Challenge described above. Since we were using the original training data as both our training and test datasets, we treated the first half of the sessions (the original training data) as an entire session and made it the goal of our model to predict whether the last song in a session would be skipped or not.

For each track in a listening session, there are two categories of features available in the original data: 1) track features and 2) metadata features. The track features (discussed in Section 4.2) include those output by the CNN shown in Figure 1 given the track's time-frequency representation while

the metadata features provide context for the session (as discussed in Section 4.3). We also engineered a number of our own features to help to examine our hypotheses (Section 4.4). Because one of our goals for this project was to understand the importance of various features and categories of features in predicting whether a song will be skipped in a session, we include an analysis of feature importance as we discuss the data in the following sections.

4.1 Preprocessing

We treat each session as a row in our dataset. On average, each session contains 15 tracks. For each track, the data includes variables called "skip_1", "skip_2", and "skip_3" which tell how long a song was listened to before being skipped. In our experiment, we ignored these fields and use a boolean "not_skipped" feature instead. For the last track in a session, the "not_skipped" value was what was learned and predicted by our model.

The session datasets were first stripped of the majority of their metadata. Information regarding how each track ended was removed as it implicitly indicates whether songs have been skipped. The only additional information that was kept was data that could be generalized across the entire session such as the date, hour of the day, and if the user is a premium Spotify user. Then, the last song of each session was isolated into a separate dataset to prevent any of the target song's features from being included in the training process.

Along with the metadata features, each song within a session of the original dataset contains a track ID of the song. The track ID can be used to locate the track features of songs in a separate file. An 8-dimensional acoustic vector that represents the track is also provided. The only preprocessing required for the track features was to change the "Mode" column to numerical values of 1 and 0 rather than "Major" and "Minor", and to normalize the data. These changes were necessary in order to treat the features as a vector for distance calculations.

4.2 Track Features

Each track in a session has 21 numerical features that describe the song and its acoustic structure.

- Duration
- Release Year
- U.S. Pop. Estimate
- Acousticness
- Beat Strength
- Bounciness
- Danceability
- Dynamic Range Mean
- Energy
- Flatness
- Instrumentalness
- Key
- Liveness
- Loudness
- Mechanism
- Mode
- Organism
- Speechiness
- Tempo
- Time signature
- Valence

A majority of these features, specifically duration, release year, key, mode, and tempo are exactly what they seem. However, Spotify includes a few values that are not so straightforward. According to Spotify's API, *danceability* tells if a track is suitable for being danced to based on its tempo, beat strength, and rhythm stability. A track's *valence* tells how positive and euphoric a track is. For example, 'Toxic' by Britney Spears is rated as 0.924, while Lana Del Rey's 'Summertime Sadness' scores a 0.22. Also, *liveness* conveys how likely it is that the track was recorded live, with an audience present. The 8-dimensional acoustic vector is also given, though each value is quite abstract as it represents the song's structure at a lower level.

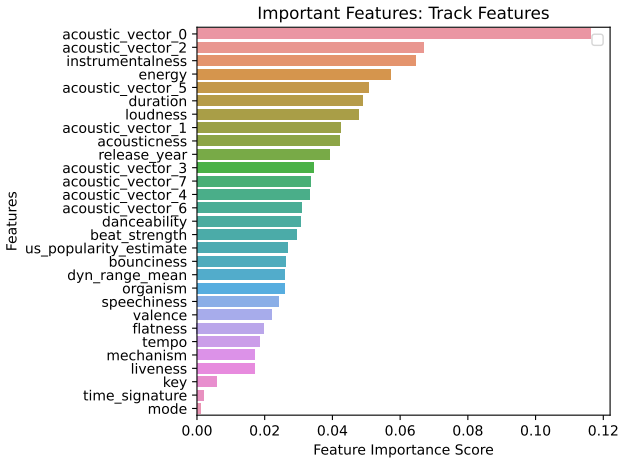

Figure 2: Ranking of the features of a model only trained on Spotify track features

A baseline model was trained to predict if the final song of a session will be skipped based only on its track features. Figure 2 shows how predictive each of these features were. When making a prediction solely on the structure of a given song, the model may be learning which songs fade out into interludes, or contain a few seconds of silence at the end. The acoustic vector features are too abstract to understand exactly which quality of the track it represents, but duration as a feature is easy to justify. The longer a song lasts, the more time there is for a user to skip it. This model was only 53.67% accurate at predicting whether the final song would be skipped in a session, so the track features as raw data only perform roughly as well as a random guess.

4.3 Metadata

Additionally, the data for each session includes features providing context for the session. This context can provide information about factors that have been shown to impact user behavior such as the fact that users have been shown to skip less often at times of the day when they are not paying attention to the music (for example, when they are asleep or at work) [5]. The hours in between those typically less attentive times contain more skips. Similarly, users skip more during

weekends than weekdays possibly because they are able to pay more attention to their music [5]. The following four features were included in our dataset:

- Month
- Hour of day
- Day of the week
- Spotify premium user

Note that the metadata features exist at the track level in the original dataset but that we generalize them to the session level so that we can attempt to capture some of the behavior described above (like skipping more on weekends). There were some features for which this generalization did not make sense such as how a track ended, if there was a pause before a track, and the type of collection the user is listening to. The first two very obviously apply to a track rather than a session. But the type of collection that a user was listening to may also vary across tracks in a single session.

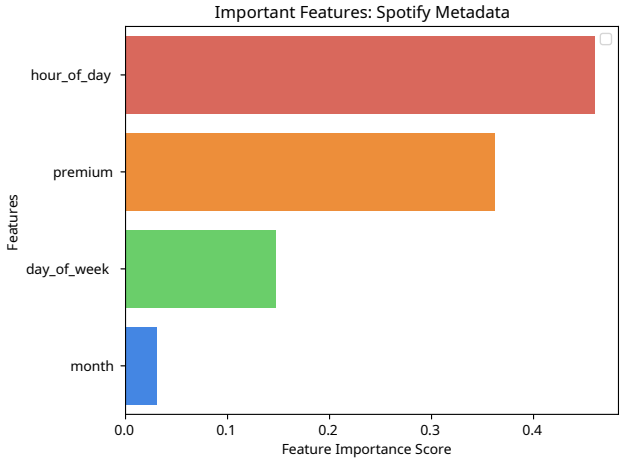

Figure 3: Ranking of the features of a model only trained on Spotify metadata

In order to get a sense of the relative importance of each of these features, we trained our model using only this metadata. Figure 3 shows the Feature Importance Scores of each of the four features. The fact that *hour_of_day* and *day_of_week* are more important than *month* seems to support our hypothesis (and others' observations [5]) about there being regular times when users skip more or fewer songs. Whether or not a user is a *premium* Spotify user impacts how many skips they are permitted per hour, so it is not surprising that this feature is relatively important in this set. The accuracy of the model trained only on this data is 59.55%. It is interesting that this model performed better overall than the model trained only on the Spotify track features, and shows that skipping behavior may be more closely related to overall behavior patterns than to individual song attributes.

4.4 Feature Engineering

Our primary hypothesis in this project is that user behavior exhibits a degree of consistency within a session. Based on this premise, it is posited that listeners are likely to skip songs that are similar to those they have previously skipped, while also being inclined to listen to songs that are similar to those they have already listened to within the same session. In order to test this hypothesis, two categories of additional features were created regarding 1) the "closeness" of the last song to the previous tracks in a session and 2) contextual features from the session.

Song Similarity Metrics

To evaluate the similarity of songs within a session, we represent the 21 track features in combination with the supplied 8-dimensional acoustic vector as a 29-dimensional vector, which we compare using linear algebra. Because we were unsure which distance measures would be most effective, we calculated several different distance metrics, augmented our dataset, and examined the resulting random forests to learn more about the behavior of these metrics.

The idea of using feature vectors was inspired by the way Spotify compares its millions of tracks by representing them as vectors in a high-dimensional space. To facilitate this process, Spotify's developers created a library called ANNOY, which stands for "Approximate Nearest Neighbors Oh Yeah" [13]. ANNOY is designed to handle large-scale data sets efficiently and uses a tree structure for this purpose. The data is split recursively into smaller sections using randomly placed hyperplanes, with each node in the binary tree representing a part of the data. The two children of each node represent a further subdivision of the parent data. By randomly generating multiple trees out of the data, the nearest neighbors can be found using the union of all trees. This allows ANNOY to perform approximate nearest neighbor searches quickly and accurately.

Spotify uses ANNOY to make specific recommendations for a user based on their previous favorites, playlists, etc. Our problem is more specific, given that we want to model the user's behavior in a single session to find out what they're interested in listening to at that moment. We hypothesized that if a song is similar (or "close") to other songs the user has listened to in a session, it is more likely that they will want to listen to the song. Similarly, if a song is closer to other songs the user has skipped in this session, the user is more likely to skip the song. There are many different ways to mathematically express "closeness" between vectors, so thirteen different features were proposed and calculated so that we could evaluate their relative effectiveness in a random forest model.

The first distance calculation used was *Euclidean distance* which is defined as the square root of the sum of the squared differences between the elements of two vectors [14]. It is a common metric that measures the straight-line distance between two points in N-dimensional space. *Manhattan dis-*

tance is another metric that measures the distance between two vectors. It is the sum of the absolute differences between the elements of both vectors [15]. Instead of a straight line, this calculation gives the distance by simulating movement along a grid, only being able to move horizontally or vertically. We also utilized the *angle between two vectors*, calculated using the cosine of the angle between the vectors, which is the dot product of the vectors divided by the product of their magnitudes [16]. The smaller the angle is, the more similar the vectors are.

Each of these metrics was used to find the distance between the final song (the one we are trying to predict whether the user will skip) and the most recently played song, the most recently skipped song, the average played song vector, and the average skipped song vector. The average vectors were utilized to represent the aggregate of the previously played or skipped songs in the session as a single vector. In addition to these metrics, a *k-d tree* [17] was used to find the nearest neighbor of the last song played.

Table 1: Distance Metrics

Name	Description
AvPlay	Distance between the final song and the non-skipped average vector, calculated with Euclidean, Manhattan, or Angle
AvSkip	Distance between the final song and the skipped average vector, calculated with Euclidean, Manhattan, or Angle
LastPlay	Distance between the final song and the last non-skipped song, calculated with Euclidean, Manhattan, or Angle
LastSkip	Distance between the final song and the last skipped song, calculated with Euclidean, Manhattan, or Angle
neighborSkipped	A boolean indicating whether the final song's nearest neighbor was skipped

Table 1 lists the distance metrics that were calculated and added to the dataset for each session. Given that the first four metrics were calculated with Euclidean, Manhattan, and Angle, this resulted in twelve separate features. With neighborSkipped, there were thirteen total features.

Notice that we are trying to capture and understand the relative importance of both the aggregate behavior over the session (the averages of played and skipped songs) and the most recent behavior (the last skipped or played song). For sessions in which all songs were only skipped or played, the distance between the final song and that empty category was recorded as -1.

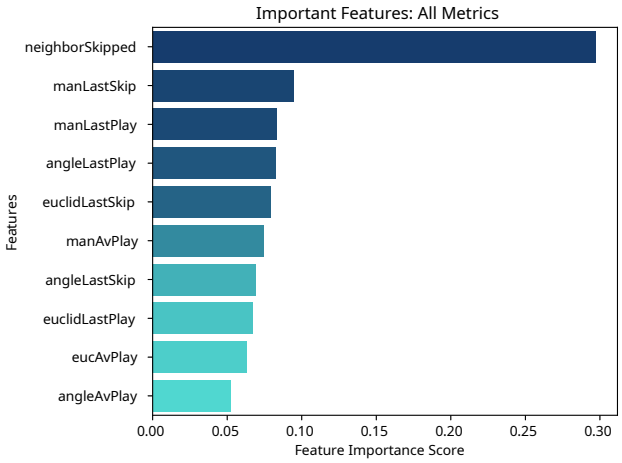

Figure 4: Top 10 most important features of a model trained on similarity metrics. Features not shown ranked $\leq .05$

When considering these similarity metrics, we knew that there would be a high degree of correlation between the Euclidean, Manhattan, and Angle features but we wanted to explore the results from the random forest to see if one category of distance measure tended to perform better overall. We utilized all thirteen of the features collectively in order to train a random forest model. When examining the importance of the features as shown in Figure 4, the Manhattan metrics ranked the highest relative to the Euclidean and Angle metrics. This aligns with the findings of Aggarwal et al. [18] who demonstrate that in high dimensions, Manhattan is preferable compared to Euclidean.

Given this analysis, we utilized the Manhattan metrics (*manAvPlay*, *manAvSkip*, *manLastPlay*, and *manLastSkip*), along with *neighborSkipped*, in our final dataset. In terms of feature importance within this set, the results are shown in Figure 5. Here it makes sense that the similarity to the aggregation of skipped songs would be the least important metric since we would anticipate a wide variety of songs could be skipped within a session, whereas we hypothesize that played songs would have more common attributes/similarity. A random forest trained using only these five features achieved a 67.57% accuracy, indicating that the feature engineering we did in using a vector space to abstract some of the context of the session in terms of the types of music being listened to and skipped had a positive impact.

Contextual Features

We also calculated two contextual features to include in the dataset. These features were intended to provide additional information beyond the distance metrics used to compare the songs within the session. These features are shown in Table 2.

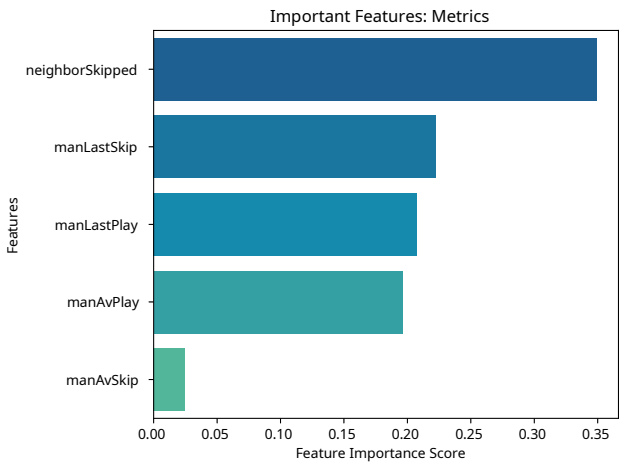

Figure 5: Ranking of the features of a model only trained on final metrics

Table 2: The two contextual features that were added and their descriptions

Name	Description
percent_skipped	The percentage of songs in the session that the user has skipped
prevSongPlayed	If the user skipped or played the previous song

We hypothesized that the percentage of songs a user has skipped in their session might indicate their current level of engagement with their music, as discussed in Section 4.3. A user skipping a majority of the songs that have been played suggests that they are engaged and paying attention and are likely to skip again, only fully listening to songs they really want to hear. Similarly, if the user skipped the most recently played track, this might indicate that they are currently active and paying attention at that moment (even if they were not skipping much earlier) and are more likely to skip again.

As with the other categories of metrics, we trained a random forest using just these two features. In terms of importance, both ranked highly with *percent_skipped* having a score of 53.3 and *prevSongPlayed* having a score of 46.7. Given that there were only two features being utilized, we did not have expectations of high accuracy with this model. However, it was quite surprising that the random forest accurately predicted whether a user would skip the last song in a session 76.01% of the time. We will discuss our ideas about possible reasons for this in the next section.

5 Results

After preprocessing and feature engineering were complete, each row of our dataset (representing a session) contained 5 metric features, 2 contextual features, 4 metadata features, 21 original track features, and an 8-dimensional acoustic vector totaling 40 features and the target value of whether or not the final song was skipped. We then split the 200,000 sessions into a training set containing 70% of the sessions and a test set containing 30% of the sessions.

The random forest models were implemented in Python and utilized the random forest classifier from the sklearn library. The final ensemble was made up of 100 trees, each with a maximum depth of 6. In order to understand the role of different features and feature categories in the model's success, several random forests were created, each using different sets of features. The results of these models were presented as we discussed each category of feature, but are summarized for comparison here, along with the results of our final models.

Table 3: Random forest results using different groups of features.

Source	Categories	Accuracy
Spotify	Track Features	53.67%
	Metadata	59.55%
	Track Features & Metadata	59.81%
Engineered	Metrics	67.57%
	Contextual Data	76.01%
	Metrics & Contextual Data	78.26%
All Data		77.69%
Final Model		78.23%

As a baseline, the first iteration only used the 29 original track features of the final song in a session. In this case, the context of a session is ignored completely. The model had to rely on the structure of a single song in order to decide if it will be skipped or not. It has been shown by Montecchio et al. [19] that a user's behavior (skipping or not) can be connected to the musical structure of a track, but it is clear that the track features in this dataset are not allowing us to capture this. This model is the least complex and had 53.67% accuracy as shown in Table 3. Then, using only certain metadata from Spotify as described in section 4.3, an accuracy of 59.55% was reached. Combining these groups of data from Spotify did not meaningfully improve the random forest's predictions.

In order to gauge the effectiveness of our engineered features, they were each used in isolation. Metrics alone achieved 67.57%, and Contextual Data, despite having only two features, was even more accurate at 76.01%. It is evident that the track features are more effective when used to try to reason about the musical context of a session than when a single track's features are used without this context (as in Section 4.2). When these two groups of features (Metrics and Contextual Data) were combined, the model achieved an accuracy of 78.26%. It was surprising to us that the similarity metrics representing musical context only yielded a small improvement in accuracy over the Contextual Data alone. This seems to indicate that the overall user pattern of behavior

(skipping or not skipping) may be more important than the user's musical mood.

Table 4: Confusion Matrix for Final Model

	Predicted Played	Predicted Skipped
Actual Played	67%	33%
Actual Skipped	14%	86%

Using both the original Spotify features and our newly engineered features in a model (shown as "All Data") resulted in slightly lower accuracy than when only the engineered features were used. This was an unexpected outcome, but we hypothesized that it was due to the near-random-guess performance of the Track Features as a whole. So our final model uses Metrics, Contextual Data, and Metadata, and achieves a predictive accuracy of 78.23%.

Because the classes being predicted (skipped and played) are roughly balanced in both our training and test data, we knew that there was some learning taking place to attain an accuracy of 78%. However, we were interested in the types of mistakes the model was making and where, potentially, improvements could be made. The confusion matrix presented in Table 4, shows that the performance of the model in predicting songs that are skipped is better than its performance in predicting the songs that will be played. This, combined with the small increase in accuracy from the inclusion of our engineered similarity Metrics with the Contextual Data leads us to believe that we are leaning too heavily on the user's general behavioral patterns (skipping vs not skipping) and that there is more exploration to be done in learning the type of music that the user is currently interested in hearing. This observation is further upheld in our analysis of feature importance in the next section.

5.1 Feature Importance

One of the goals of this research was to discover which features in the data were the most predictive. When using a random forest, it is possible to calculate which features contributed most to the final predictions. For this, Mean Decrease Impurity (MDI) was used to calculate feature importance. MDI measures the average decrease in the impurity of nodes when splitting the data on a particular feature. Nodes are impure when they contain data from different classes. If a feature has a high MDI score, it effectively splits data into more homogeneous groups, making the most progress toward a prediction. Using the Python libraries matplotlib and seaborn, visualizations of these scores were generated.

Figure 6 shows the importance of a track's acoustic features and metadata when making a prediction of whether a user will skip a song. The prominence of the *premium* feature is straightforward to understand. Users without Spotify premium have a limited amount of skips, and therefore will use them more sparingly. The rest of the metadata features are ranked lower than expected. The *month* feature was found to be not important at all. This can be attributed to the fact that the data was not equally distributed across all months, causing the feature to not be properly understood by the random forest. It seems to be the same situation for *day_of_week* as this feature also scored unexpectedly low.

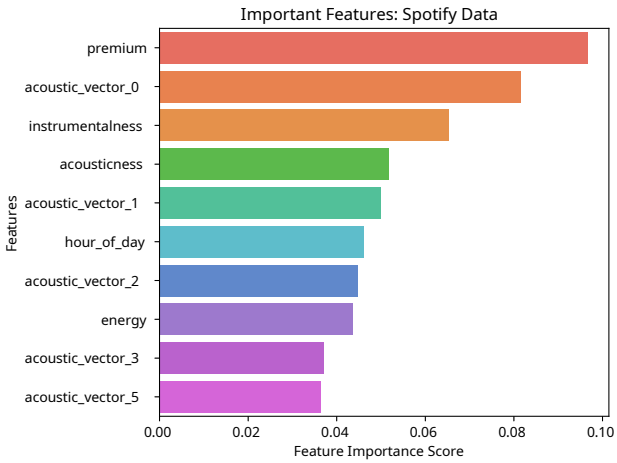

Figure 6: The top 10 most important features of a model trained on Spotify's data (Track Features and Metadata). Features not shown ranked $\leq .033$

Of our engineered features, the contextual data is extremely predictive. When combined with the metrics it dominates the rankings in Figure 7. If we had not run the metrics by themselves first, it would seem as if they are not predictive at all. According to the MDI scores, the contextual data is simply more relevant in making successful predictions in this model. As mentioned previously, this suggests that our future work should be concentrated on improving our metrics in order to be able to better recognize the songs that the user is most interested in listening to.

Focusing only on the metrics, *manLastSkip* and *manLastPlay* both rank higher than the distances between the average vectors. From this result, we can assume that the type of song a user skipped or played immediately before is more relevant than the type of songs they have skipped or played on average throughout their session. It is possible that because the average vectors represent every song that has been skipped or played in a session, as a user's session grows, these vectors will become more obscure and less relevant.

Finally, as shown in Figure 8, the engineered features greatly outperform those relating to Spotify metadata. As stated before, this group of features was not expected to do so poorly. Though, this outcome may be attributed to the lack of diversity of dates in the data used. It is interesting to note that the order of feature importance of the engineered features remained consistent between when they were used in isolation vs alongside metadata, reinforcing the fact that the metadata was barely influential.

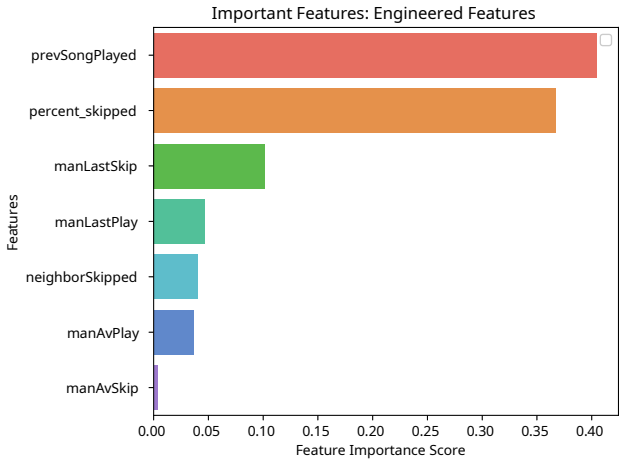

Figure 7: The most important features in a model trained on only our engineered features (Metrics, and Contextual data).

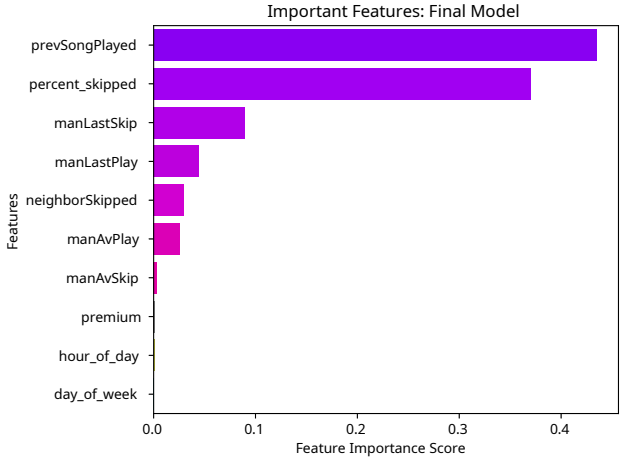

Figure 8: The top 10 most important features (no Track Features) used in the final model. Features not shown ranked $\leq .0001$

6 Future Work

Our goal is that this model could eventually be used not only to make a prediction but to choose songs from the playlist a user is listening to that they are least likely to skip. As the session progresses, and more songs are listened to, the model will have access to more information and be able to make better predictions in real-time. A simple way to accomplish this would be to classify every song yet to be played and use the model's confidence to rank which songs it is most certain will not be skipped. This, however, would have linear complexity as the size of a playlist grows. Instead, a random sample of songs from the playlist may be picked and ranked in order to keep up with the speed at which a user can skip songs.

Alternatively, a more complex model using a recurrent neural network similar to the solution of Hansen et al. [8] could be trained to compare the accuracy between artificial intelligence techniques. Our belief is that prediction accuracy might increase with an RNN, but our initial use of decision trees allowed for influential features to be identified and better understood.

If data was available containing more than one session for a user, habits could be learned over time to better anticipate the user's moods. Listening preferences of a user by time of day, day of the week, or even month could be learned. With this information, the system could adjust its choice of queued songs at the beginning of each session. So, if a user typically listens to slow and sad music on weekdays after 6:00 PM, when they shuffle their playlist at this time, the first songs that play are more likely to be slow and sad.

A challenge of this feature would be the balance between true randomization and tailored randomization. The learning aspect must be subtle enough to not negatively impact the user's experience. It is important to avoid locking the system into playing a specific genre of music at a particular time, as this would detract from the intended variety offered by a shuffle algorithm. To achieve seamless and virtually unnoticeable customization, it is crucial to find the proper balance between tailored and random songs.

7 Conclusion

In this paper, we have explored our research question of whether we can utilize session data to accurately predict whether or not a user will skip a particular song. We hypothesized that user "skipping" behavior could influence the overall pattern within a session and that users would be more likely to listen to similar songs within a session according to their mood. We have shown that it is indeed possible for a model to make these predictions accurately, though the most influential information was not necessarily as we hypothesized. When predicting whether a user will skip a song, the frequency of skipping and recent skipping behavior are more influential factors than the similarity of the song to previously skipped songs in our current model. In future work, we will explore whether we can more accurately predict whether a user will play a song by utilizing different metrics and strategies. However, our final model described here attained 78.23% accuracy and shows that user behavior can be leveraged to predict whether or not a song will be skipped in a given session. In the future, this model could be utilized to adapt queued songs according to the user's behavior and current preferences, resulting in fewer skips overall and an improved user experience.

References

[1] B. Brost, R. Mehrotra, T. Jehan, The music streaming sessions dataset, in *Proceedings of the 2019 Web Conference* (ACM, 2019).

[2] L. Weng, The multi-armed bandit problem and its solutions, *lilianweng.github.io*.

[3] S. Dieleman, Recommending music on spotify with deep learning, 2014.

[4] X. Wang, Y. Wang, D. Hsu, Y. Wang, Exploration in interactive personalized music recommendation: A reinforcement learning approach, 2013, doi:10.48550/ARXIV.1311.6355.

[5] P. Lamere, The skip, 2014.

[6] D. Jannach, M. Ludewig, When recurrent neural networks meet the neighborhood for session-based recommendation, in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 306–310 (Association for Computing Machinery, New York, NY, USA, 2017), ISBN 9781450346528, doi:10.1145/3109859.3109872.

[7] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk, Session-based recommendations with recurrent neural networks, 2015, doi:10.48550/ARXIV.1511.06939.

[8] C. Hansen, C. Hansen, S. Alstrup, J. G. Simonsen, C. Lioma, Modelling sequential music track skips using a multi-rnn approach, doi:10.48550/ARXIV.1903.08408.

[9] F. Béres, D. M. Kelen, A. Benczúr, et al., Sequential skip prediction using deep learning and ensembles.

[10] A. Ferraro, D. Bogdanov, X. Serra, Skip prediction using boosting trees based on acoustic features of tracks in sessions, doi:10.48550/ARXIV.1903.11833.

[11] F. Meggetto, C. Revie, J. Levine, Y. Moshfeghi, Why people skip music? on predicting music skips using deep reinforcement learning, *arXiv preprint arXiv:2301.03881*.

[12] L. Breiman, Random forests, *Machine Learning*, *45*(1):(2001), 5–32, doi:10.1023/a:1010933404324.

[13] E. Bernhardsson, Nearest neighbors and vector models – part 2 – algorithms and data structures, 2020.

[14] E. W. Weisstein, Euclidean metric.

[15] M. Barile, E. W. Weisstein, Taxicab metric.

[16] A. M. Helmenstine, Angle between two vectors and vector scalar product, 2020.

[17] B. Wheeless, A look into k-dimensional trees, 2021.

[18] C. C. Aggarwal, A. Hinneburg, D. A. Keim, On the surprising behavior of distance metrics in high dimensional space, in J. Van den Bussche, V. Vianu, editors, *Database Theory — ICDT 2001*, pages 420–434 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2001), ISBN 978-3-540-44503-6.

[19] N. Montecchio, P. Roy, F. Pachet, The skipping behavior of users of music streaming services and its relation to musical structure, *PLOS ONE*, *15*(9):(2020), e0239418, doi:10.1371/journal.pone.0239418.

OBJECT-ORIENTED CREATIVE CODING FOR DIGITAL ART STUDENTS

Dale E. Parson
Kutztown University of Pennsylvania
parson@kutztown.edu

ABSTRACT

An introductory course in creative graphical coding for both computer science and digital art majors need not be watered down. The availability of an excellent Java-based framework including an IDE, debugger, and high-level class and function library avoids potentially problematic mathematics and device-control demands by encapsulating them within the library. Students can add or modify a few lines of code and then run them from the IDE without explicit compilation steps or tool changes, immediately seeing the results of initial coding or bug fixes in the form of animated graphical objects. Incremental introduction of object-oriented mechanisms within the course such as reuse, function and class encapsulation, interface and implementation inheritance, and polymorphism, go hand-in-hand with incremental addition of features to an initially simple in-line program. Course enrollments are high, students succeed, and the opportunities for creative work are open ended.

KEY WORDS

Computer animation, creative coding, digital art.

1. Introduction

"Hybrids that can fluidly cross the chasm between technology and the arts are mutations in the academic system. Traditionally, universities create technology students or art students – but never mix the two sides of the equation in the same person. During the 1990s, the mutants that managed to defy this norm would either seek me out, or else I would reach out to find them myself. Bringing these unique people together was my primary passion, and that's how I came into contact with Casey Reas and Ben Fry." (Forward to Reference [1] by John Maeda, currently VP of Design and Artificial Intelligence at Microsoft.)

This experience paper grows out of 13 years of teaching interactive computer graphical design and creative multimedia programming to undergraduate students. The first 5 years consisted of teaching Photoshop and Illustrator composition in a general education course and teaching object-oriented Java programming to computer science and mathematics majors. In 2014 the author migrated the raster and vector graphics editing courses to two new creative coding classes using the Processing framework and media libraries [1-3], coded in Java. CSC120 *Introduction to Creative Graphical Coding* first ran in fall 2015. CSC220 *Object Oriented Multimedia Programming* followed in spring 2016. One to three sections of these courses have run during alternate semesters since then, attracting computer science and mathematics majors, and non-majors to the 100-level course.

In 2015 our College of Visual and Performing Arts formed a team including the author to plan for a new BFA in Applied Digital Arts (APD). By the spring 2018, third-semester APD students comprised about 80% of two sections of CSC120, a core course in their major, with many going on to CSC220 as a major elective in two of their specialization tracks. Many of these students had never anticipated programming. The sections that follow summarize both how object-oriented structures eased their transition into the world of coding, and pedagogy for accommodating non-traditional programming students.

2. Creative Graphical & Multimedia Coding

2.1 CSC120 *Introduction to Creative Graphical Coding*

Each student advances their own custom avatar through developmental stages in structure and behavior throughout the semester. The textbook's *Zoog* animated character serves as a template for this development [2], to be customized and extended by each student into a unique avatar. By semester's end we have used object-oriented mechanisms to make creation of an interacting society of avatars possible. The following subsections outline avatar developmental stages.

2.1.1 Project 1: In-line Visual Code

Figure 1 on the next page shows two display captures of a Project 1 mobile avatar, the Professor, in front of a campus building, moving among lampposts and trees. These are two virtual photographs from the running handout code that students must modify to create their own custom mobile avatars and stationary settings. For many art students who have never programmed before, the task of customization is at first daunting. Aspects of these images worth consideration follow.

- Students must use shape-displaying library functions such as line(), ellipse(), rect(), arc(), quad(), stroke() and fill() (for color), strokeWeight() (for stroke thickness), and two-dimensional display coordinates for placement.
- The Processing run-time framework invokes the custom draw() function at the framerate, typically 30 or 60 frames per second, to achieve smooth animation. By changing location coordinates within each call to draw(), the programmer moves objects at a smooth rate.
- The colors of the background and trees in Figure 1 vary from dark and floodlit-red in the top frame to daylit and green in the bottom frame. The author teaches parameters such as color as locations in an abstract space, in this case a color space, similar to the 2D x,y locations of placement of graphical primitives.
- Placing the avatar behind a lamppost in the bottom frame requires plotting the avatar's body parts before plotting the intervening lamppost within the code. There is no third, Z dimension in 2D graphics. Objects plotted later appear in front of objects plotted earlier at overlapping locations.

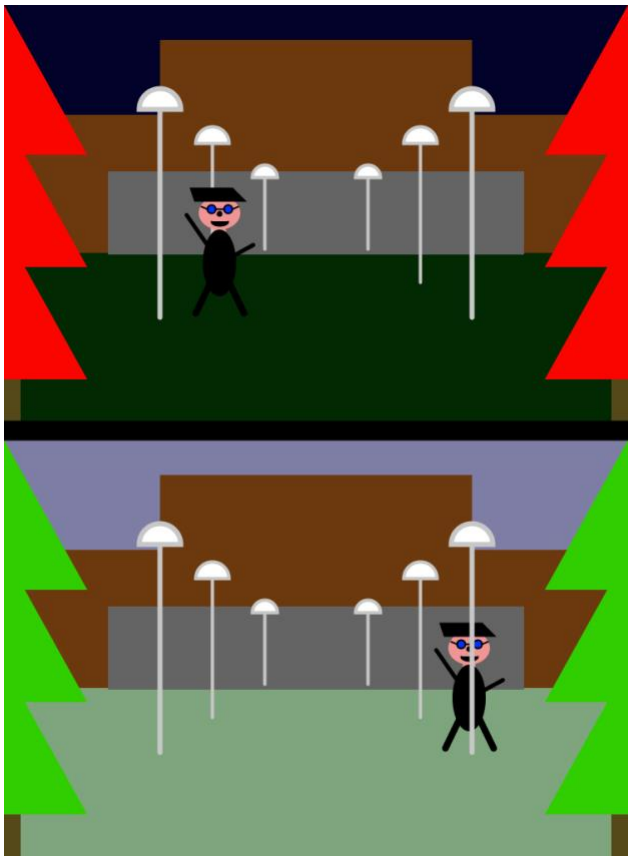

Figure 1: Snapshots of the Handout Code's Animation

This so-called *sketch* (Processing's term for a program) contains 4 global state variables used and updated within the periodic draw() function.

- int backgroundColor = 0 ; // cycle through grayscale
- int avatarX = 0 ; // Move avatar left-right-left in cycles.
- int rangeX = 0 ; // X from 0 to 2*width-1 for update
- int legX = 0 ; // Distance leg from X center [-10,10].

The author describes such a 2D scenario as a stage on which players move among the set and can disappear off-stage. This avatar moves behind the trees at the left and right.

Figure 2 shows a much more elaborate CSC120 Project 1 created by a music major with graphical design experience in the Blender design environment [4] but with no previous Processing or Java programming. The car bounces realistically and throws a shadow as it moves below the lamp, and the sun rises and sets in the background with the sky darkness tracking night and day.

Figure 2: A Student Mobile Avatar

Most initial student projects are much simpler than that of Figure 2. Most students are just starting to program. Projects do not strive for mathematical precision. Instead, the author encourages students to prototype placement of background objects and body parts, and to nudge them into place when they are slightly off. Students get visual feedback immediately when changing and running code. Even though the programming language is Java, the Processing Integrated Development Environment (IDE) hides the compilation step and highlights compile-time errors and warnings in the editor. Figure 3 shows the IDE highlighting a misspelled function name interactively. Misspelled variables, syntax errors, and most other compile-time problems are highlighted by behind-the-scenes source code analysis. Most run-time errors that crash a program highlight the offending line of code.

The most difficult bugs are those that move entire graphical objects off screen. Even scaling the entire display down so objects that are off-screen may appear on-screen does not always find them. In practice, no students seems to get into the disappearing object problem more than once per student.

One requirement imposed by the instructor that does not appear until late in the textbook [2] is that students must surround any mobile or complex graphical object display code such as the car with an opening pushMatrix() call, followed by translate(X,Y) to a x,y reference point within the object, and a closing popMatrix() call at the end. This translate(X,Y) call moves the 0,0 reference point of the 2D coordinate system from the upper left of the display to the conceptual center of the object. Once that center reference point is in place, the x,y locations of body parts are relative to that reference point, not relative to a global location. A body part can then be placed relative to the local 0,0. Furthermore, pushMatrix(), translate(), plotting, and popMatrix() calls can be nested so that a complicated body part can be placed relative to its center rather than to the reference point of the entire body. This coordinate locality of reference introduces students to the idea of a *stack* of reference points. Furthermore, any call to the 2D rotate() function centers around the innermost reference point, and scale() affects nested translate(X,Y) and object placement scaling. Nested translate(), rotate(), and scale() changes are discarded by a subsequent popMatrix() call. Recent versions of Processing have combined pushMatrix() and pushStyle() into a push() library function paired with a composite pop() function. The pushStyle() / popStyle() pair save and restore incoming color and other non-coordinate visual aspects similar to pushMatrix() / popMatrix() for coordinates. The next paragraphs give an example use of these functions. In class the instructor demonstrates these ideas by physically translating himself around the room and nested-translating displayed objects held in his hands.

The code of Figure 3 creates the display window of Figure 4. The framework-specified setup() function sets the size of the display window in pixels (width, height), the frame rate (frequency of periodic calls to draw() in times-per-second), and the color mode, in this case Hue-Saturation-Brightness (HSB). Argument 360 defines a color wheel (hue) that varies from 0 (red) through 360 (red) as illustrated in Figure 5. Color Saturation and Brightness range from 0% to 100%. The author came to learn that HSB with saturation and brightness of 100% was the most effective way to use the relatively low-saturation university planetarium projector, when an art student demonstrated that on the dome during a CSC120 group capstone project session.

The draw() function of Figure 3 runs periodically at the frame rate, 30 frames per second as configured in setup(), illustrating the library functions used to create Figure 4. Code comments complement this discussion. At draw() entry, coordinate 0,0 is at the left, upper corner of the display as is typical for graphics libraries.

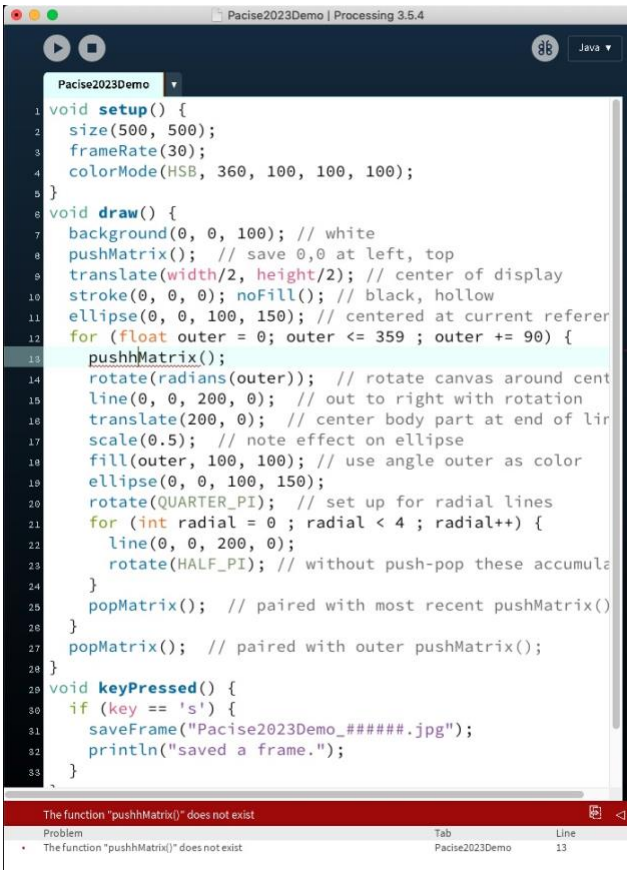

Figure 3: Processing's IDE

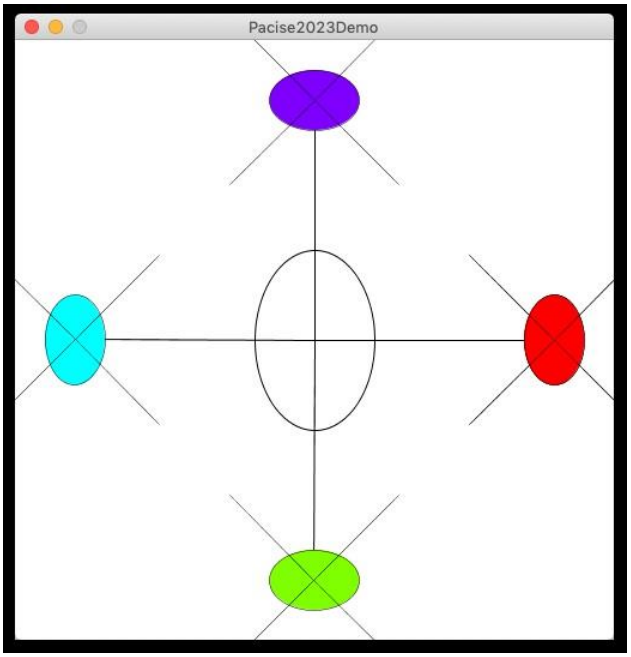

Figure 4: Display Generated by Figure 3's Code

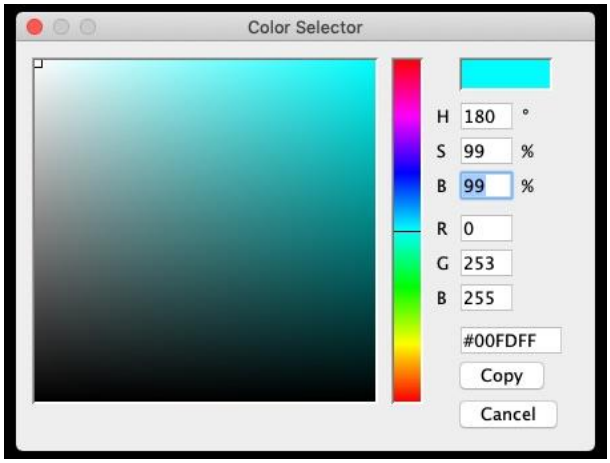

Figure 5: The IDE's Color Selector Tool

After setting the background to white (0% saturation – ignoring the hue – with 100% brightness), draw()'s first pushMatrix() call saves the incoming coordinate system. Translating to the halfway locations of the predefined system variables *width* and *height* (of the display) moves the 0,0 reference point to the display center. After drawing an ellipse with a width of 100 pixels and a height of 150 at the center (i.e., translate-relative location 0,0), draw() enters a for-loop iterating from 0 through 270 degrees in 90 degree steps in variable *outer*. At each step the loop starts with pushMatrix(), saving the loop-incoming 0,0 point at the display center. It then rotates the entire display an increment of 90 degrees, converted to radians. It draws a line from the 0,0 center point to the x=200,y=0 point relative to the preceding rotation. Figure 4 shows 4 lines radiating from the center due to this line() call at source code line 15. Next, translate(200, 0) moves the 0,0 reference point out to the end of that line. Calling scale(0.5) reduces the size of the ellipse at line 19 and the length of the lines at line 22 by half, even though they use the same sizes and coordinates used in lines 11 and 15. The visual universe has been scaled down. At line 18 fill(outer, 100, 100) uses the angle of global rotation to set the hue, with saturation and brightness of 100%. At line 20 rotate() applies the built-in constant of QUARTER_PI radians to offset upcoming lines by 45 degrees, and the inner loop draws 4 lines, each rotated from its predecessor by 90 degrees. Figure 4 shows the results. The HALF_PI rotations accumulate because there is no surrounding push()-pop() pair to undo them. Students can use the radians() function to convert degrees, or they can use radian constants and variables. The outer loop ends with a popMatrix() call that discards the translations, rotations, and scales that follow its partner pushMatrix() at the top of the loop, and the final popMatrix() re-establishes the global coordinate system coming into draw(). Processing has done a good job of modernizing the body-relative use of coordinates inherited from the LISP-derived Logo programming language [5,6] that, like Processing, originated in the MIT Media Lab.

2.1.2 Project 2: Behavior as Functions

Assignment 2 adds requirements for loops and creation of at least one function for the avatar called from draw() at the framerate. As usual, the author hands out example code along with the assignment specification that students must understand.

```
// THE display() FUNCTION DISPLAYS AN AVATAR
// BASED SOLELY ON ITS PARAMETER VALUES.
// DO NOT USE GLOBAL VARIABLES IN display().
// STEP 1: Write your version of this function:
void display(int avx, int avy, float avscale, int avwiggle) {
  // avy is this avatar's x location; avy is its y location;
  // avscale is its scaling factor; avwiggle wiggles parts.
  // Move avatar push, translate, scale, and pop code here,
  // but keep the avatar move code up in draw().
  // USE GEOMETRIC TRANSFORMS TO POSITION
  // THE AVATAR
  // All avatar-specific coordinates are relative to the
  // translated center of the avatar.
```

Here are the in-code instructions for writing a loop.

```
// I SATISFIED STEPS 5 (more avatars) & 7 (a loop)
// IN THIS LOOP.
// YOU CAN USE A LOOP TO DRAW SOMETHING
// OTHER THAN AN AVATAR, E.G., SOME PART OF
// SCENERY. I USED IT TO DRAW AVATARS. YOU
// MUST DRAW AT LEAST A SECOND AVATAR FOR
// STEP 5, BUT THE AVATAR DOES NOT HAVE TO
//  BE PART OF THE LOOP.
for (int clonex = width/4 ; clonex < width
          ; clonex += width/2) {
// start at width/4; adding width/2 comes back into the
// loop only a second time.
   display(clonex, height/8, 0.5, 16*legX);
}
```

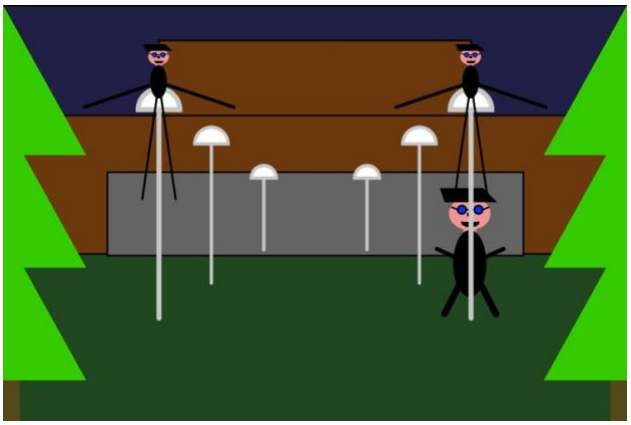

Figure 6: A Function Stamps Out Multiple Avatars

Students add a function definition and multiple invocations to stamp out multiple avatars. The author's handout implementation added only one more global variable for

limb movement speed. Other than that, the placement of the two additional avatars appearing in Figure 6 are at fixed locations with no mobility. The Professor avatars in Figure 6 wave their long arms but otherwise are immobile. It would be straightforward to add mobile avatars that move only in parallel with the original avatar without adding new location state variables. Their respective reference points would be at fixed offsets from the original, reference avatar.

2.1.3 Projects 3-4: Behavior + State as Classes

Project 3 creates a class from Project 2 global state variables, function display(), and the state update logic in draw() into a new move() function. Handout class Professor contains these state variables, move() display(), and a constructor that sets initial location, speed, and scale. Replacing the global state variables is an array of Professors. Project 3 introduces this array of objects and expands the use of loops to iterate over it for calls to the constructor using pseudo-random locations and an assortment of scaling factors as arguments. Construction occurs in Processing's setup() function, and a loop within draw() invokes display() and move() on each Professor object. Students must replace Professor with their own classes as extensions of appearance and behavior coded in Project 2. Example animated display is much like Figure 6, but now avatars can move independently from each other.

Project 4 introduces a Java interface and abstract class that initiate students into object-oriented design and implementation. **Interface Avatar**, supplied by the instructor, specifies the following class functions.

- display() is identical to Project 3, using state fields within each object to project graphics.
- move() is identical to Project 3, updating state variables such as location and speed.
- getX() and getY() return Avatar coordinates.
- shuffle() and forceshuffle() specify optional and mandatory random changes in location, triggered by keyboard commands, for jumping out of dead ends.
- getBoundingBox() returns the 4-tuple of upper left, lower right x,y coordinates of a 2D box enclosing the Avatar object.

Abstract class AvatarHelper, also supplied by the instructor, houses state variable fields common to all Avatar objects such as x,y location and speeds. This helper class also provides a collision detection and recovery function definition, to be used at the bottom of each concrete Avatar class move() function. Collision detection uses objects' getBoundingBox() function calls to detect overlapping Avatar objects.

In the handout code, concrete **class Professor** derives from AvatarHelper and implements the specifics of Professor.display() and Professor.move(). Display and movement logic are the same as in Project 3. Figure 7 shows the inheritance hierarchy of Project 4 handout code. Data fields and functions in **bold** are defined by Processing, with setup() and draw() written by the coder.

Project 4 uses an array of abstract Avatar object references that can house multiple Professor objects with independent location and speed state variables. There are 50 Professor objects in Figure 8. The transition from a behavioral display() function in Project 2 to a class that encapsulates each avatar's state in object fields and behavior in display() and move() functions is an incremental transition. The ease with which 50 Avatar objects can be constructed within a loop and stored in a polymorphic Avatar array is impressive to students.

Figures 7 and 8 also show immobile **Furniture** objects – the magenta line barriers – and a rotating yellow **Paddle** object in the center that also derive from interface Avatar and abstract class AvatarHelper, with their objects residing in the polymorphic Avatar array. All of these objects interact via collision detection and recovery in AvatarHelper, called at the bottom of the move() functions, with collision detection calling individual objects' getBoundingBox() function. Figure 8 shows the rectangular bounding box around each avatar, displayed during debugging of getBoundingBox() and collision detection / recovery.

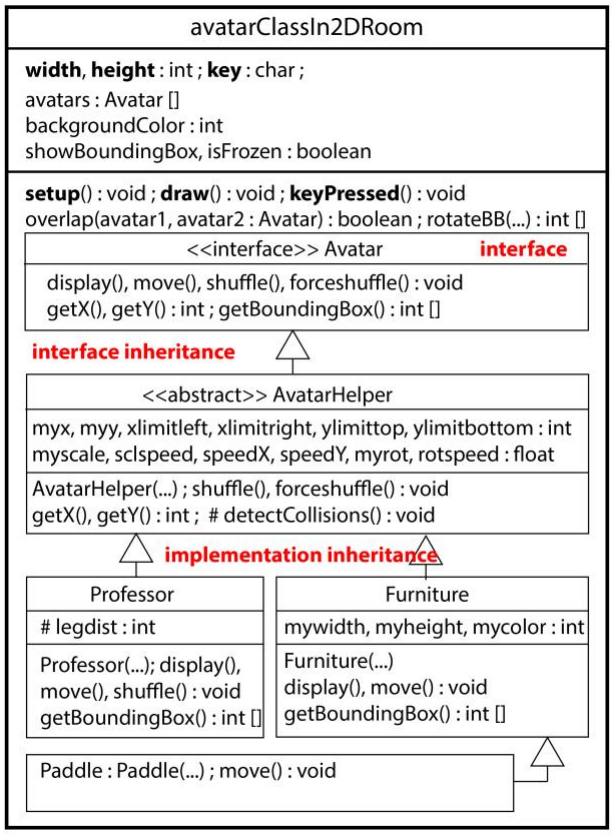

Figure 7: Project 4 Inheritance Hierarchy

Students must replace the Professor class with their own class, using display() and move() logic from Project 3 along with enhancements. They must also make enhancements to the Furniture and Paddle class appearance and behavior.

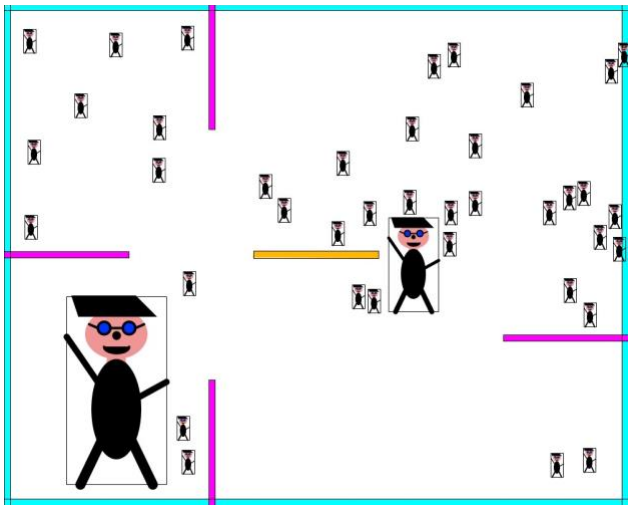

Figure 8 : Array of Avatars of Varying Classes.

Within the scope of four projects, novice programmers are exposed to the practical utility the following object-oriented concepts and mechanisms. Each project builds upon the prior, presents opportunities and requirements for student customization, and provides immediate feedback in terms of visual presentation.

- Code reuse is a key object-oriented concept. Processing supports library reuse through its graphical functions and classes.
- An interface specifies functions provided by subclasses.
- An abstract class provides data fields (variables) and functions (a.k.a. methods) that can be used by subclasses. You cannot construct an object of an abstract class. It is a helper class.
- Concrete classes are classes from which you can construct objects that model state in data fields (variables), and activities in functions.
- Polymorphism (many forms) means that subclasses of interfaces and base classes can take many forms, such as these concrete classes. Our use of an array of Avatar objects from varying classes illustrates polymorphism.

2.1.4 Project 5: A Group Project Capstone

Project 5 builds on the object-oriented infrastructure introduced in Projects 3 and 4 without adding new concepts. The primary addition is the coexistence of many student Avatars acting as paintbrushes. The draw() function does not erase previous frames via the background() library function. The author took student login ID strings, generated empty Avatar-derived class declarations named after those strings, and each student supplied custom display() and move() code for their Avatar paintbrush. The author also generated keyboard command interpretation for adding and removing student mobile paintbrushes. Figure 9 shows photos of three stages of student paintings on the university's planetarium dome during the interactive project demonstration. The polymorphism of the array of Avatar objects housing student Avatar concrete objects made this integration of student paintbrushes possible.

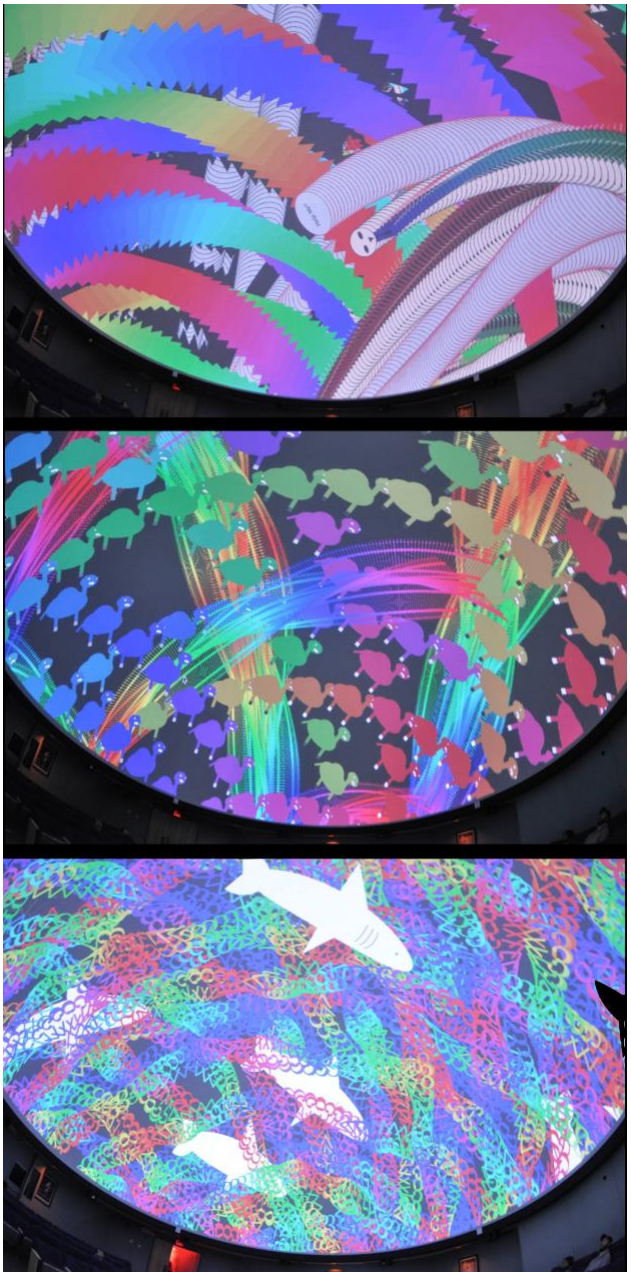

Figure 9: Student Avatar-Interface-Derived Objects

2.2 CSC220 *Object Oriented Multimedia Programming*

The object-oriented mechanisms and student coding efforts introduced in previous course projects open doors for extension and reimagination driven by student creativity and additional Processing library support. Figure 10 illustrates just two of the projects we have tackled in CSC220, the second-level course, namely 3D Avatar environments and photo-derived avatars.

Multimedia projects introduced in CSC220 include music and sound. Java supplies interfaces and classes for reading and writing packet-like MIDI (Musical Instrument Digital Interface) messages [7,8]. The Java library includes basic software-synthesized instruments and audio effects such as stereo balance, chorus, and reverb. Students extend MIDI-generating sketches through play & listen interaction with the MIDI instruments and effects.

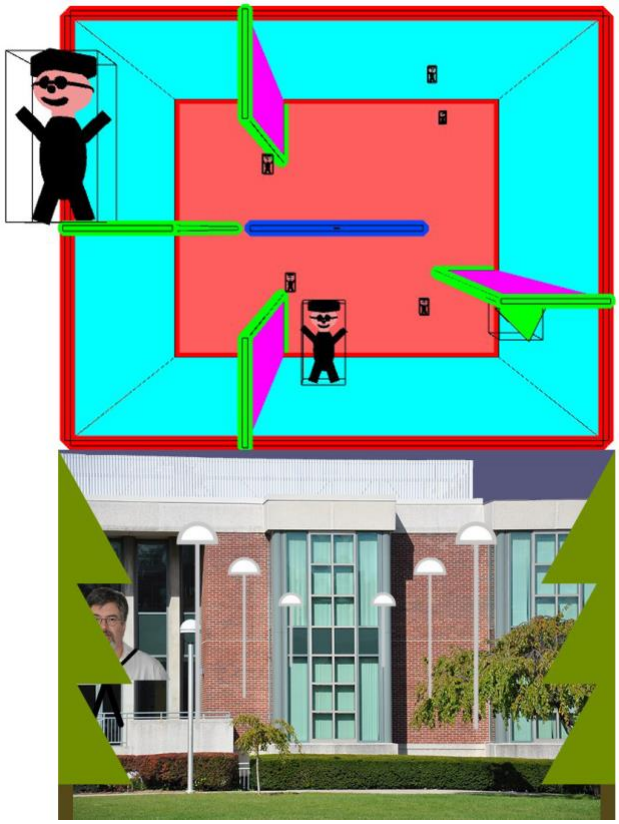

Figure 10: 3D Avatars and Photographic Avatars

The Android operating system for tablets and cell phones can run Processing code via cross-compilation from a laptop or PC [9]. There are a few added library functions and considerations of display size and aspect ratio. The author and students have connected Android client Processing sketches to Processing display servers via wireless networking [10]. Figure 11 shows a 3D, server-side Processing projection designed for our planetarium dome controlled by leaner 2D, client-side graphical remote controls of Figure 12 on laptops and Android tablets.

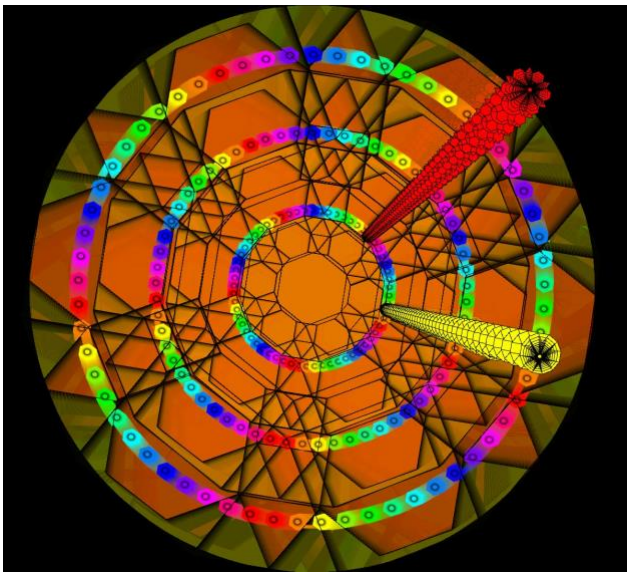

Figure 11: 3D Server Visual Music Projection

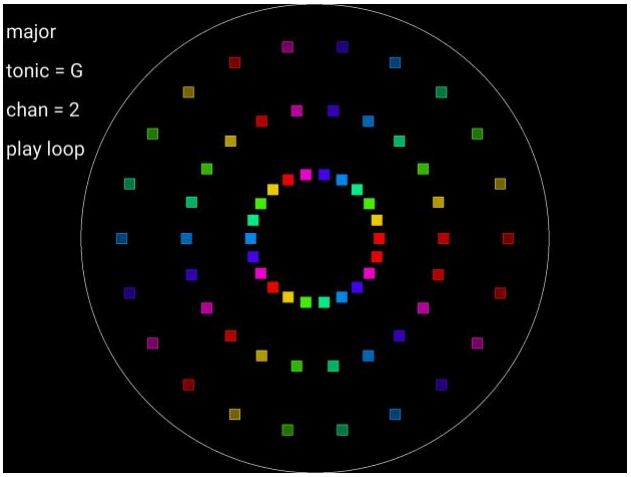

Figure 12: 2D Client Controller on a Laptop or Tablet

Recursion is another CSC220 topic. Figure 13 shows 4 screenshots of an interactive recursive program that includes user key commands for changing depth of recursion, animating rotation of shapes, and navigating the camera point-of-view through the 3D space. Figure 13 shows the display of the handout code. Students must customize the space-filling base shapes and add commands for the expansion and contraction of the lattice and the base shapes. Other recursive projects model plants and architectural structures.

The final CSC220 project for fall 2022 consisted of an interactive visual instrument. The approach is an improvisational video instrument coded by the instructor, and extended by each student who then conceived, practiced and performed a novel piece for exhibition using

their own photographs. Fifteen students have granted permission to have their work appear in the video [11]. The video piece is running in the university library throughout the spring 2023 semester. Figures 14 and 15 show two screen captures from the 34-minute video.

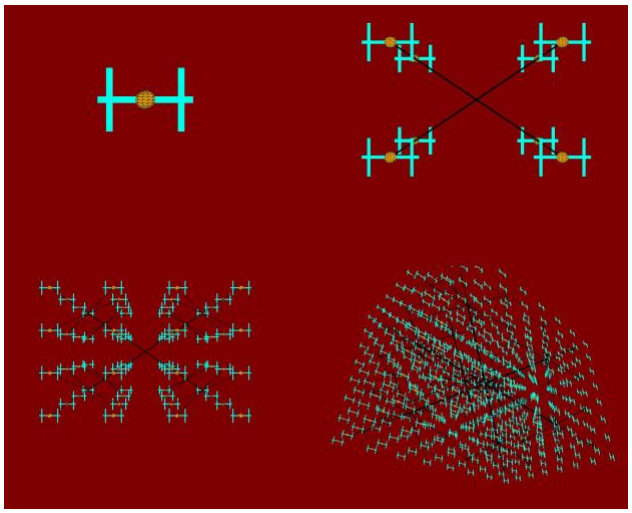

Figure 13: Stages of Interactive, Animated Recursion

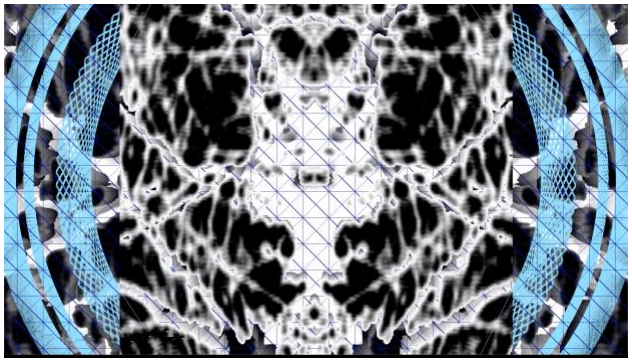

Figure 14: Screen Capture from Fall 2022 CSC220

Figure 15: Screen Capture from Fall 2022 CSC220

3. CSC120 and CSC220 Pedagogy

The author uses an approach adapted from 20 years as an industrial software developer, the last 10 of them as a lead system architect. The approach is to treat students as junior engineers and to supply a substantial project framework for most assignments that require students to read and understand code. Feedback from alumni confirm that this is what they spend much of their professional lives doing. In the projects listed above, the author supplied the Professor class, the Avatar interface and AvatarHelper abstract base class, and other framework code cited. Supplying substantial code teaches students how to read and understand code and supports ambitious projects.

Students customize projects in ways immediately clear when running their sketches. Not only does the primary requirement for custom work encourage creative thinking and exploration of Processing library capabilities, it also makes it possible to hand out one solution for a problem as the starting point. Cheating is much less possible, and late assignments due to illness or other legitimate reasons are not a problem because the instructor never hands out "the solution". There are N + 1 solutions for N students plus the instructor.

In stages of CSC120 instruction the instructor and students copy and paste code from the on-line textbook examples [2] into the Processing IDE instead of teaching from slides. We experiment with modifying and discussing those sketches. Class time consists largely of interacting with code and its animated execution.

In 2013 through 2015 the instructor and two student collaborators data mined the correlation of programming behavior to project grades for Java programming students [12,13], done with permission of those students. The following are the primary takeaways applied in CSC120 and CSC220.

- Start two weeks before the project deadline. Additional time is usually wasted. Two weeks give students time to determine whether the project is within easy reach or requires effort immediately. For most projects students can skip working for half of these days, but the two-week start is essential. The author has solved this problem by having an in-class work session at the next class after an assignment is handed out, to give students time to read and formulate questions. Students requested such sessions the first time the author skipped one. Both this practice and the copying and pasting of on-line textbook code examples mandate a lab classroom. The author also schedules an office hour immediately after class, when possible, to continue working with students having problems.
- Working fewer than 60 minutes per coding session does not correlate well with good project grades, presumably because the cognitive system does not

have time to fully engage. The author insists on a minimum of 80-minute classes.

- No more than 20% of work should occur at night. This one is impossible to enforce, but the author discusses it with students.

The immediate sensory feedback of a working or broken code change seen by running the sketch from the IDE is extremely valuable in tying student programming to sensory awareness. There is a positive affective dimension to seeing lively animations when compared to more conventional textual output from programs.

4. Conclusions

Our department's major CS I and II courses do not introduce classes until CS II and substantive inheritance until later. It is possible to introduce the object-oriented mechanisms of Figure 7 into a late-semester CSC120 project because these mechanisms relate so closely to what students need to do to get multiple avatars running around among obstacles on the screen. Development is incremental starting with the Project 1, so there are no large conceptual leaps. The instructor always hands out working code as a starting point, so there is code to inspect as questions arise. Digital art students who expressed concerns when entering CSC120 have done well, and many have gone on to take CSC220 and concentrate in one of the tracks for which it is a major elective. Enrollments are high, students succeed, and the opportunities for creative work are open ended.

4. Acknowledgements

Thanks go to Professor Josh Miller of the Communications Design Department (merged with Art in 2022) for collaborating with the author and his students in interdisciplinary team projects courses for many years. The courses acted as an incubator for multimedia software projects inspiring the author's work described here. Professor Miller worked to have CSC120 included as part of the core of the Applied Digital Arts major, with CSC220 serving as an elective. It has been very rewarding to help art majors acquire a love, or at least a liking, of programming.

Dr. Phill Reed, Professor of Astronomy at Kutztown, deserves much thanks for his work and patience in supporting the author, students, members of the faculty and public in making the university planetarium available for course lab sessions and many evening and multi-day visual music concerts. The Kutztown Planetarium has been an ideal immersive multimedia laboratory environment [14,15].

Dr. Lisa Frye, Chair of the Department of Computer Science and Information Technology, has been very supportive of the author's inclination to explore strange new worlds and to go boldly where no Kutztown CS professor has gone before in the areas of interdisciplinary courses in multimedia coding and data science. The job is still fun as the author approaches his 70th birthday.

Finally, students Eric Wolfe created the elaborate mobile avatar of Figure 2, and Paul Barton and Madison Schlott create the video segments of Figures 14 and 15 respectively, granting permission to use them in public displays [11]. Students have been every bit as much explorers in creative multimedia coding as the author.

References:

[1] C. Reas and B. Fry, *Processing, A Programming Handbook for Visual Designers and Artists*, Second Edition (Cambridge, MA: MIT Press, 2014).

[2] D. Shiffman, *Learning Processing, a beginner's guide to programming images, animation, and interaction*, Second Edition (Burlington, MA: Morgan Kaufmann, 2015), http://learningprocessing.com/.

[3] Processing home page, https://processing.org/.

[4] Blender home page, https://www.blender.org/.

[5] B. Harvey, *Computer Science Logo Style*, Volumes 1 through 3, MIT Press, Second Edition, 1997.

[6] S. Papert, *The Children's Machine: Rethinking School In The Age Of The Computer*, Revised edition, Basic Books, 1994.

[7] Java class library javax.sound.midi package, https://docs.oracle.com/javase/8/docs/api/index.html?javax/sound/midi/package-summary.html.

[8] J. Glatt, *MIDI Technical Fanatic's Brainwashing Center*, http://midi.teragonaudio.com/, tested February 2023.

[9] A. Colubri, *Processing for Android* (New York, NY: Apress Media, 2017).

[10]. D. Parson, "A Distributed Model-View-Controller Design Pattern for a Graphical Remote Control of a Multi-user Application". *Proceedings of the 35th Annual Spring Conference of the Pennsylvania Computer and Information Science Educators* (PACISE), West Chester University of PA, West Chester, PA, April 3-4, 2020. Presented on-line April 10, 2021 due to COVID cancellation of PACISE 2020.
https://research.library.kutztown.edu/cisfaculty/10/

[11]. "CSC220 student videos fall 2022 shared with permission", CSC220 Object-Oriented Multimedia Programming, Final Project, Copyright January 2023, Creative Commons Attribution 4.0. 1080P 33:53 mins:secs, by Justus Hamm, Christopher Lipovsky, Rebekah Underwood, Paul Barton, Amber Kulp, Daniel Gorman, Armando Velazquez Santiago, Kylee Hager, Jesse Quier, Madison Schlott, Luis Feliz, Marcello Feliciano, Michael Colandene, Palak Dilawari, Ryan

Livinghouse, and Dale E. Parson of Kutztown University of PA. https://youtube.com/watch?v=GbasCYtyFhQ&si=EnSIkaIECMiOmarE

[12] D. Parson and A. Seidel, "Mining Student Time Management Patterns in Programming Projects," *Proceedings of FECS'14: 2014 Intl. Conf. on Frontiers in CS & CE Education*, Las Vegas, NV, July 21 - 24, 2014.

[13] D. Parson, L. Bogumil, and A. Seidel, "Data Mining Temporal Work Patterns of Programming Student Populations". *Proceedings of the 30th Annual Spring Conference of the Pennsylvania Computer and Information Science Educators* (PACISE), Edinboro University of PA, Edinboro, PA, April 10-11, 2015.

[14] D. Parson, "A Circular Planetarium as a Spatial Visual Musical Instrument", white paper (2018), accepted for presentation at IMERSA SUMMIT 2019 of planetarium directors, content providers, immersive multimedia designers and coders, Columbus, OH, February, 2019. https://research.library.kutztown.edu/cisfaculty/1/

[15] D. Parson and P. Reed, "The Planetarium as a Musical Instrument," *Proceedings of the 2012 New Interfaces for Musical Expression Conference*, Ann Arbor, MI, May 20 - 23, 2012.

Comparative Study of Multivariate Time Series Forecasting Algorithms

Bradley Betts, Kirstin Crawford, Rachel Henigin, Raed Seetan
Slippery Rock University
bsb1006@sru.edu, kjc1016@sru.edu, reh1015@sru.edu, raed.seetan@sru.edu

ABSTRACT
The purpose of this research is to compare multivariate time series forecasting algorithms in order to provide insight on the most efficient algorithms when predicting outcomes. Datasets involving weather and stock information are evaluated through the use of the Vector AutoRegression (VAR), Long Short-Term Memory (LSTM) neural networks, and Facebook Prophet algorithms. Python and the R Programming Language are utilized to perform the algorithms and produce meaningful forecasts. The models are assessed through similarity, identification of correlation, and visualization performance as demonstrated with the Root Mean Squared Error (RMSE). This project is different from other related works that were researched because of their focus on univariate time series algorithms, or the use of said algorithms. Data cleaning is used to handle missing data, erroneous values, and outliers. The results show that Facebook Prophet outperforms VAR and LSTM neural networks for datasets of all sizes, and that LSTM neural networks perform better than VAR for larger datasets, while VAR performs better than LSTM neural networks for smaller datasets.

KEY WORDS
Vector AutoRegression, LSTM networks, Facebook Prophet.

1. Introduction

Time series analysis can be defined as a way of analyzing a continuance of data points that were accumulated over a period of time. Since a time series involves time-dependent variables, the analysis allows for the influential factors on those variables to become known. Univariate time series forecasting models are good for predicting outcomes for a single variable. However, multivariate time series forecasting analysis is more challenging, and predicts future values through identifying patterns with the multiple variables at hand. A multivariate time series has multiple time-dependent variables with each variable not only depending on the past values, but also on other variables within the dataset. This dependency is then used to forecast future values.

One of the most popular methods for multivariate time series forecasting is Vector AutoRegression (VAR). This method is defined as a way to assimilate the relationship between multiple variables as they transform over time. It allows for multivariate time series activity through the generalization of the single-variable autoregressive model. Another method that is widely used is called the Long Short-Term Memory (LSTM for short) networks. This model is an advanced recurrent neural network (RNN) that reduces the long-term dependency issues within the dataset. As the patterns within the data are separated by longer periods of time, LSTM becomes more and more useful. Also, Facebook Prophet is a multiple time series forecasting algorithm tool that estimates seasonality that has either linear or non-linear growth. The product of these algorithms result in multivariate time series conclusions that demonstrate predictive information valuable for businesses and research.

Depending on the desired forecasting outcomes, each algorithm is handled differently and provides different results. When comparing the models of multivariate time series algorithms, it is important to be aware of the types of errors, identification of correlation, and visual performance. In order to determine the best algorithm, the forecasting measures and procedures are classified through the use of analytical programming, resulting in optimal predictions. Multivariate time series algorithms can be evaluated and compared to figure out the best way to provide efficient forecasted results.

The purpose of this study is to compare a few different multivariate time series algorithms with different types of data by comparing the results of the forecast with the actual test results found in the original dataset. The specific metric chosen is the Root Means Squared Error (RMSE), due to its ease of readability. Each algorithm is graded based on their performance against each other in predicting the most accurate results for each dependent variable in the datasets. This is presented on a scale of 1-3, with 1 being the least accurate, and 3 being the most accurate of the algorithms using the RMSE measurement. This study will serve as a guide for what time series algorithm is best suited for analysts performing multivariate forecasting on their data.

Previous studies have put a focus either on comparative studies of univariate time series algorithms, an in-depth study of a single multivariate time series algorithm, or a comparative study comparing some univariate methods with a multivariate method. Studies testing the performances of differing multivariate methods are sparse. This study aims to fill in that gap in the current literature of multivariate time series forecasting.

The remainder of this paper is structured as follows: Section 2 discusses the related works, Section 3 presents the methodology used, Section 4 demonstrates the results of the study, Section 5 concludes the study, Section 6 offers suggestions for future studies, and Section 7 shows all the supplementary material.

2. Related Works

2.1 Univariate Forecasting

Phan [1] examined six methods for forecasting univariate meteorological data. These included: Simple Exponential Smoothing (SES), Seasonal-naive (Snaive), Seasonal-ARIMA (SARIMA), Bayesian Structural Time Series (BSTS), Imputation of Time Series Based on Dynamic Time Warping Functions (DTWBI), and Feed-Forward neural network (FFNN). In particular, Phan was largely concerned with comparing performance between linear and neural network models, as linear models may predict general trends, but fail to determine nonlinear features or those which exhibit quick change. All but one of the methods above are standard options, in regards to forecasting. DTWBI, however, was previously regarded as an NN-type classification algorithm. It was selected for testing due to its ability to capture the shape of dynamic data. With the six methods, experiments were conducted to determine which method best performs forecasting for specific cases and parameters. In assessment of model performance, it is essential to consider both the ability to accurately predict response variables and to model the shape of dynamic trends and seasonality. For univariate meteorological forecasting, Phan finds that FFNN produces the most accurate forecasting values, while DTWBI produces the best shape and dynamics.

Katsuri [2] conducted a study to determine an optimal method of prediction for Crude Palm Oil (CPO) price. The price of CPO is non-linear, non-stationary, and dynamic. Thus it is difficult to predict these values with standard linear techniques. Some linear-centric methods, like Box-Jenkin's Autoregressive Moving Average (ARMA), produce short-term predictions yet falter when forecasting long-run time series data. "Traditional shallow" neural networks are not suited for the non-linear, nonparametric, noisy, dynamic, complex, and chaotic nature of many real-world time series. Given these concerns, Katsuri compared three models: Artificial Neural Network (ANN), Long Short Term Memory (LSTM), and the Holt-Winter model. ANN is described as the most efficient NN approach, while Holt-Winter exists as a statistical smoothing method. LSTM, however, includes procedures for long-term learning as well as for efficiency, such as the incorporation of a 'forget' gate, to remove non-necessary information. Consequently, upon implementation, LSTM produced a Root Mean Square Error which is 48.45 less than Holt-Winter's and 75.44 less than ANN. With this, Katsura found that LSTM dramatically outperforms the ANN and Holt-Winter models, when predicting CPO price.

2.2 Multivariate Forecasting

Muzaffar [3] performed a study to compare the ability of algorithms to forecast load, given some meteorological data. As the expanse and complexity of power stations has increased, the difficulty in accurately predicting energy needs, given extreme weather phenomena, has increased as well. In response, the study is conducted by making comparisons between the increasingly popular, NN-type LSTM method and traditional forecasting methods, such as Autoregressive Moving Average (ARMA), Autoregressive Moving Integrated Average (ARIMA), and Autoregressive Moving Integrated Average with Explanatory Variables (ARIMAX). Given the context of the subject data, predictions are produced for timelines of 24 hours, 48 hours, 7 days, and 30 days. The study finds that LSTM outperforms the traditional methods in general and for short-term especially. Importantly, the LSTM model is capable of accommodating the addition of many significant variables, to improve prediction.

Jammalamadaka [4] developed a Multivariate Bayesian Time Series (MBSTS) model which utilizes online text mining to predict stock prices. The model was the first of its kind and suggested that there was more to discover about machine learning, data mining, and forecasting complex phenomena, such as stock prices. In particular, the model incorporated sentimental predictors, which are found to substantially contribute to the accuracy of predictions. The sentimental predictor, in the final iteration, classifies big company-focused news articles as having some polarity: *positive*, *negative*, or *neutral*. In earlier iterations, a sentimental predictor was used to classify social media posts, about tech giant companies, as having some emotion along the lines of: *anxiety*, *calmness*, *dislike*, *fear*, *liking*, *love*, *joy*, *sadness*, and *unknown*. While the polarity indicator was found to improve prediction, the emotion indicator was determined insignificant and omitted from future iterations. Further, the study evaluated the ability of sentimental indicators and compared the accuracy of MBSTS to other time series algorithms. Jammalamadaka tested Autoregressive Integrated Moving Average (ARIMA), Long Short-Term Memory (LSTM), and

MBSTS. For each algorithm, three models were constructed. These included models with sentimental predictors, without sentimental predictors, and with no predictors. The study concluded with two primary findings. First, those models with sentimental predictors outperformed the other models of its kind (no sentimental predictors and no predictors), across all algorithms. Second, the sentimental predictor-incorporated MBSTS model achieved the most accurate predictions.

2.3 Comparative Procedures and Measures

Iwok [5] compared the performance of univariate and multivariate time series, using five time series variables for Nigeria's gross domestic products. The study raises the pertinent question of whether multivariate time series outperforms its univariate counterpart. With advancements in machine learning and data mining, univariate forecasting seems limited, in comparison to that which is multivariate. Especially in regards to time series forecasting, univariate methods tend to fail in supplying information about interrelationships of variables. Multivariate methods primarily excel at utilizing information about interrelationships to optimize data processes such as classification, clustering, and forecasting. With this, it is fairly easy to assume that more is better. However, to test this common assumption, Iwok develops a univariate ARIMA model and multivariate VAR model. Model performance is determined by comparing, for each of the five time series variables, the Mean Square Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). Across every variable's statistics, the univariate ARIMA procedure indicates superiority. Hence, it is concluded that, for this context, the univariate model is the best. The finding draws pertinent attention to the common oversight of "simpler" methods. From this, Iwok concludes by recommending that comparative analysis be conducted to determine which method, univariate or multivariate, is best at predicting, given particular data and concerns.

In contrast, all previously mentioned studies perform model comparisons between models with the same number of response and explanatory variables. Rather, these other studies largely draw comparisons between statistical and neural network models. Phan [1] decidedly conducts the most thorough comparison, with six different tested algorithms. Additionally, four performance indicators are used for evaluation. These are similarity, Normalized Mean Absolute Error (NMAE), Root Mean Square Error (RMSE), and Fractional Bias (FB). And performance is compared across time series with various degrees of seasonality and trend. The study provides a structure for comparison of time series algorithms, as well as impressive results regarding the performance of DTWBI.

The remaining majority, [2], [3], [5], compare performance between three to four different algorithms. Jammalamadaka [4] is a slight exception to this, as additional iterations of models are constructed to determine pertinence of sentimental predictors. Additionally, these only consider mean error statistics, for performance evaluation.

2.4 Using Algorithms for Data Cleaning

Patil [7] discusses different data cleaning algorithms for data warehouse systems. Due to the demand for precise information regarding data collection and processing, data cleaning is an important step when analyzing data because it can increase the quality of the data while reducing the efforts in collecting it. Methods such as deduplication, substantiation, and house holding are explained to provide insight when dealing with data cleaning within data warehouse systems. A data cleaning algorithm detects inaccurate data and then corrects the errors and commissions through steps like checks, reviews, assessment, and validation of the data. This paper also examines the phases of data cleaning: data analysis, defining the transformation workflow and mapping rules, verification, transformation, and backflow of cleaned data. Data cleaning thus improves the quality of the data by reducing the inconsistency and assists the users or experts with standardizing the data.

In the next research paper, J.M.Z.H [8] discusses the topic of data cleaning using a machine learning paradigm for big data analytics. A big challenge in big data analytics is detecting and repairing dirty data and if this challenge is not met, there can be inaccurate data remaining which can cause unpredictable conclusions. The survey paper dives into data quality troubles, different cleaning tools available, and the challenges that arise when cleaning big data. A comparison of commercialized data quality management tools shows the vendor, data cleaning product, website to obtain this product, and the likes and dislikes of this product from a compilation of survey results from customers. Some of these data cleaning tools that were compared are Infosphere QualityStage from IBM, Oracle Enterprise Data Quality from Oracle, Data Management from SAS, and SAP Data Services from SAP. It was concluded that machine learning algorithms could replace data science jobs in the future because of the fast evolution of big data and the increased availability in accessing programming tools like R and Python. Machine learning applications improve over time since they learn through the addition of more data thus showing how important data cleaning is when dealing with big data analytics.

Wang [9] conducts a thorough examination of time series data cleaning techniques and reviews methods for each type. This examination aims to solve problems presented in the cleaning of time series data. Specifically, there are four types of errors in time series data- continuous, single

big, single small, and translational. Ignorance toward these errors can lend to inefficiencies in time series models. However, the cleaning of time series data is difficult, given large amounts of data and errors, complex causes for errors, continuous generation and storage, and the minimum modification principle. To attempt to handle these problems and difficulties, Wang examines four types of techniques. These are smoothing based, constraint based, statistics based, and anomaly detection. Via testing, concludes that, in general, smoothing techniques require a lesser time demand, but may distort results given the ease of changing originally correct data. Constraint based methods are found to be generally too-strict for time series cleaning, as these are based on equality relationships; however, initial anomaly detection offers some promise of resolution, as these could treat outliers before implementing strict constraint based cleaning algorithms. With a variety of adaptive statistical techniques, these types tend to fare better for time series cleaning. With these methods, algorithms can be chosen based on desired or optimal sample sizes, outcome probabilities, and relationships between outcomes, granting more-accommodating results that do not greatly bias data or change correct values. Unsurprisingly, finds that anomaly detection algorithms play an important role in time series cleaning, as these support the principle of minimum modification and may improve the success of further cleaning techniques with preliminary anomaly handling. Conclusively, the research conducted leads to encouraging application and development of these algorithms to improve time series cleaning, with careful consideration of the specific data, causes for errors, and the desired results.

When discussing the preprocessing, cleaning, and analysis of data, larger datasets prove to be problematic, since they can often be unreliable, unscalable, and very slow to work with. Triguero [10] made this process more efficient for big data by using the k-nearest neighbor (k-NN) algorithm to turn big data into what he dubbed “smart data”. This algorithm can reduce the size of data by removing redundant data, and correct data imperfections seen from noisy and missing data. Apache Spark Packages were also analyzed and compared with one another on their accuracy, reduction of data, and runtime of the algorithms in question. Although all k-NN algorithms improve upon big data, it was found that the most efficient algorithm to use is situational for each dataset in question.

3. Methodology

3.1 Datasets

The first dataset, which was acquired from Kaggle, contains weather data from the city of Delhi, India. From January 1st, 2013 to April 24th, 2017. The mean temperature (in Celsius), humidity (grams of water vapor per cubic meter volume of air), wind speed (in kilometers per hour), and mean pressure (in atmospheric pressure) was tracked using the Weather Underground API, and are quantified as numeric features. The data comes pre-split into a training and test set, with 1,462 and 114 observations respectively.

The second dataset comes from the UCI Machine Learning Repository. Much like the first dataset, the data is allocated daily, with the exception being the lack of weekends, due to the data pertaining to stocks from within the public stock market. This dataset was created to test the efficiency of convolutional neural networks (CNN) for automatic feature selection and market prediction through the closing values of five different stocks: DJI, NASDAQ, NYSE, RUSSELL, and S&P 500. Alongside the five dependent values, there are also a variety of features split into groups. These groups include: technical indicators of the closing value of the dependent stock, the treasury bill’s secondary market rate, change in market yield on U.S. Treasury securities, relative change in prices of commodities, returns of different world indices, exchange rates between the U.S. dollar and other currencies, returns of U.S. companies, and returns of futures. All data was gathered from December 31st, 2009 to November 15th, 2017. Each dependent closing value for the five stocks has its own dataset containing these values, resulting in each containing 84 features and 1,984 records. Because most of the features are the same across each of the five datasets, they were combined into one dataset, with the technical indicators of each dependent variable also being removed. This brought the features down to 74.

When looking at the collection of datasets, those based on predicting weather were found to have no blank values, while the stocks had several missing values. Data cleaning will be discussed in detail in the next section.

3.2 Preprocessing and Transformations

3.2.1 Cleaning Blank Values

The weather dataset contained no blank values to handle. However, the stocks data had 2,692 missing values across many variables. To handle these, Weka was utilized to fill missing values by using the unsupervised procedure, ReplaceMissingValues. This replaced each missing value with the feature’s mean value, which was acceptable since the values that were missing were from features representative of changes in quantitative values, (I.e., prices of different stocks, commodities, futures, etc.). The data was cleaned in this manner over using algorithms due to the simplicity of the data used in this study, and low portion of missing data.

3.2.2 Dimensionality Reduction

As the weather dataset has a small number of features, there was no need to reduce the number of dimensions for that dataset. Dimensionality reduction of the stocks dataset was deemed necessary due to its larger number of dimensions, and therefore was split into three steps: manually removing features that demonstrated ≥ 0.80 correlation with other independent variables, running stepwise regression, and performing principal component analysis (PCA).

After cleaning the stocks dataset, it was found that attempting to run stepwise regression to decide important variables was impossible, due to the AIC being negative infinity. This is a result of the data being overfitted when including every variable. To remedy this problem, the data was normalized, and several correlation matrices were run on subsets of the features (for the sake of readability). Completing this left 37 independent variables that saw no heavy correlation between each other.

Fortunately, the AIC was no longer negative infinity, so stepwise regression was feasible with what remained. These regressions were performed on each dependent variable individually to see if any of the dependent variables exhibit unique variance from select independent variables, or if they are explained by similar features. The forward selection resulted in no change in the feature selection. Meanwhile, both the backward elimination, and bidirectional elimination methods, resulted in anywhere from nine to twelve features. Fortunately, every variable exhibited in these were the same. The twelve variables that were kept are as follows: *DGS10*, *GBP*, *CNY*, *GE*, *MSFT*, *SSEC*, *TE6*, *DE4*, *CTB3M*, *CTB6M*, *copper.F*, and *silver.F*. Explanations on these variables can be found in the corresponding paper [6]. Upon investigating the results of these models, the value allocated to the *CNY* (exchange rate between the U.S. dollar and the Chinese yuan) regressor was significantly higher than all the others, including the intercept. Though this is not indicative of the U.S. and Chinese economies having a causational relationship, further investigation into their stocks could help explain any possible correlations seen between the two economic superpowers of the world.

Performing the PCA on the stocks dataset (with five different analyses for each dependent variable) did not result in any significant reductions of the dataset. When interpreting the results of a PCA, the first few principal components that explain at least 80% of the variance are kept. This ratio is increased to 90% if other analyses are to be performed following the PCA. For 90% of the variance to be accounted for, 9 principal components would need to be selected of the 12. As shown in Figure 1, this lack of significant results in conducting a PCA on the stocks dataset is the result of the data being mutually orthogonal, meaning that there are no redundancies in the dataset. However, this means that dimensionality reduction cannot be performed any further. The PCA was also run before stepwise regression, with similar results. This concludes that there were no redundancies in the dataset, and that the minimum number of features necessary to best explain the variation of the stocks datasets are the twelve mentioned earlier in this section.

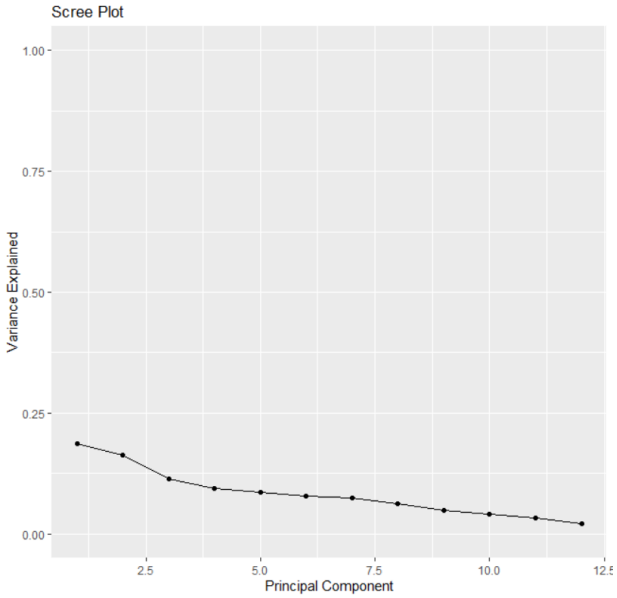

Fig. 1 Principal Component Analysis Scree Plot of the Stocks Dataset

3.2.3 Erroneous Values and Outliers

With the information on both datasets available online, there were some erroneous values found in the weather dataset, but no significant outliers (barring blatant errors seen in the weather data) in both. The egregious errors that appeared were for some values of mean pressure. For example, a -3 in atm is physically impossible as 0 atm is a vacuum. These values were corrected by researching the weather data of Delhi, India on *timeanddate* during the day the erroneous value was recorded. On this site, the pressure was measured seven to eight times across three hour intervals in inches of mercury (“Hg). Each value was corrected with the average of these values and converted to atmospheric pressure (atm). The weather data no longer experienced outliers as a result of these corrections. Outliers for the stocks dataset are kept as many features exhibit a balanced ratio of high and low outliers, but are accounted for in the next section on transforming the data.

3.2.4 Transformations

Some of the algorithms being performed in this study require the data to be normalized and normally distributed. Henceforth, it is important to check said data for these assumptions. The weather data exhibits normal distribution for each of the variables, but the stocks data

has two variables which have slight negative skew: *DE4* and *CTB6M*. *DE4's* data is mostly positive, with very few that are negative. To solve this, the data can all be increased by some constant that is slightly larger than the absolute value of the smallest value (to account for the possibility that future data may be slightly lower than the current minimum value). Then, those values can go through log transformation, which results in normal distribution. For *CTB6M*, a lot of values are negative, so this can be solved by taking the absolute value of the variable, and then put them through log transformation. However, when running this, a large amount of values become negative infinity, which are considered to be NA values. As such, the original *CTB6M* value is used, and not the transformed one. The effects of these changes are demonstrated below in Figure 2.

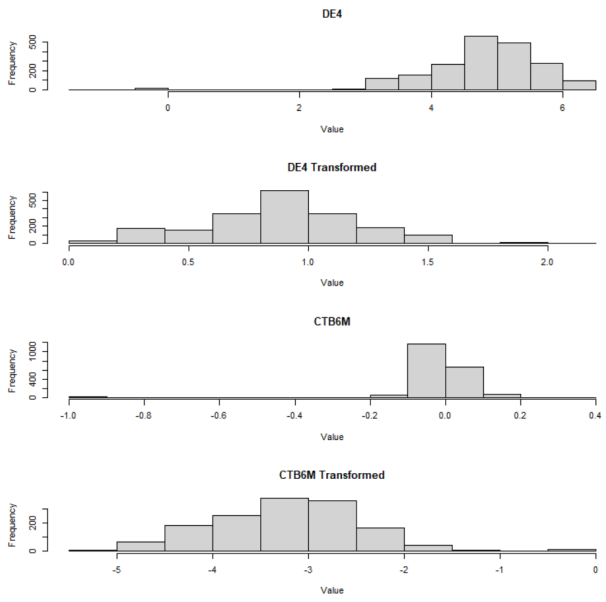

Fig. 2 Histograms of the Original and Transformed DE4 and CTB6M Features

3.3 Analysis Methods

For this study, the following three algorithms were used: Vector AutoRegression (VAR), Long Short-Term Memory (LSTM) neural networks, and Facebook's Prophet algorithm. For the sake of ease, the VAR and LSTM neural networks were programmed in R, and Facebook Prophet was programmed in Python (R's version of prophet had no existing documentation on its usage for multivariate time series forecasting).

VAR seeks to capture the relationship between multiple quantities as they change over time. This makes the model act as if each feature is its own dependent variable, so it will not require data to be normally distributed, but will still need normalized and lagged data. There are several diagnostics to run for the model of this algorithm that are possible with the vars package in the R Programming language:

1. The Phillips Perron test to determine stationarity of the features (which occurs before the creation of the model)
2. Determining an optimal number of lags
3. A serial test for autocorrelation
4. The ARCH test for heteroscedasticity
5. A normality test for distribution of residuals
6. Stability test for presence of structural breaks
7. Granger causality testing to see if each time series is useful in forecasting the others
8. Impulse response functions (IRFs) to see how shocks in each feature affects one another
9. Forecast error variance decomposition (FEVD) to see what errors in each variable are caused by

Once the diagnostics are performed, the forecast can be analyzed and checked for accuracy based on the root mean squared error (RMSE). The lower the RMSE value, the more accurate the generated results are. For the weather dataset, the optimal number of lags for the VAR model was $p = 3$ based on the Schwarz Criterion (SC(n)) for lag selection, the order of integration is $d = 1$, and for the stocks dataset, $p = 2$ and $d = 1$.

In accordance with the LSTM neural networks, the data will need to be made stationary, lagged, and normalized for optimal accuracy in forecasting. The input is then put into a three-dimensional array and the model is compiled using the keras package available in R.

Meanwhile, Facebook Prophet is extremely easy to use and only requires the data to be separated into training and test sets, and then input into the model. Prophet is based on the sklearn model API, and only requires the user to placate each independent feature as a regressor, the algorithm will do the rest. Forecasting (both univariate and multivariate) with Prophet is built in a model using the smooth line of $y(t) = g(t) + s(t) + h(t) + e$, where the variables are representative of overall growth trend, seasonality, holiday effect, and error term (or residual) respectively. Users can then fine tune the forecasts based on their domain knowledge. Prophet is available in both R and Python, but for this study, Python's prophet package was chosen due R's lack of multivariate time series forecasting for their version of Prophet. Every algorithm is assessed based on their calculated RMSE and compared alongside each other.

4. Results

Facebook Prophet produces the best forecasting models by far. However, the LSTM neural networks and VAR algorithm show different results based on the size of the dataset. VAR performed better than LSTM neural networks for the weather dataset, while the inverse took effect for the stocks dataset. LSTM neural networks tend to perform better with more features to work with, so it makes sense that for the most part, LSTM outperformed

VAR on the stocks dataset. The neural network actually performed almost on par with Facebook Prophet for the *RUSSELL_Close* feature, and worse than VAR for the *NASDAQ_Close* feature. The RMSE values for each algorithm are listed in Table 1.

Table 1
Algorithm RMSE Values

	Mean Temp	DJI	NASDAQ	NYSE	RUSSELL	S AND P
VAR	5.14	19185.72	5447.23	10914.70	1267.09	2212.89
LSTM	6.59	3102.52	11354.14	1485.76	238.63	432.74
Prophet	***3.89***	***1647.71***	***1434.35***	***972.39***	***238.26***	***278.07***

When ranking the algorithm's performance, Facebook Prophet is unconditionally the best of the bunch, earning a total of 18 points. LSTM is only slightly better than VAR, acquiring 10 and 8 points respectively. This ranking is demonstrated in Table 1.

Table 2
Algorithm Performances

	Mean Temp	DJI	NASDAQ	NYSE	RUSSELL	S AND P	Total
VAR	2	1	2	1	1	1	8
LSTM	1	2	1	2	2	2	10
Prophet	3	3	3	3	3	3	18

From this research, Facebook Prophet is the best algorithm to use for multivariate time series analysis, no matter the size of the data. Not only due to its consistently high accuracy of predicted results, but its simplicity to implement. However, when choosing between VAR and the LSTM network, it is dependent on the size of the dataset. If the dataset is smaller and doesn't have a high number of dimensions, VAR is the best bet. On the contrary, if the dataset is larger with more dimensions, the LSTM network is the better choice.

The visualizations of the *meantemp* feature from the weather dataset and *DJI_Close* feature from the stock dataset are shown for each algorithm in Figures 3-8 below.

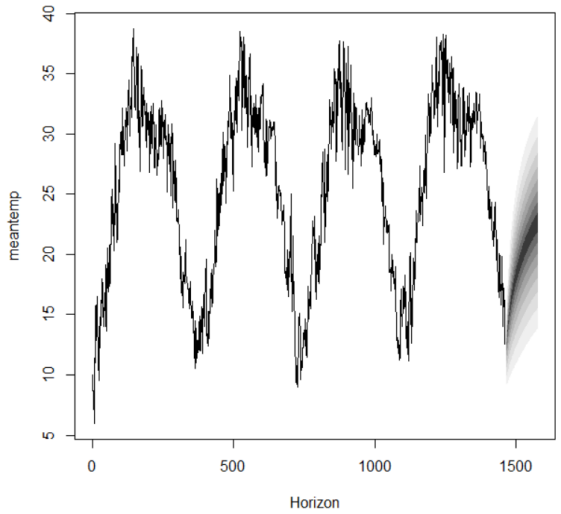

Fig. 3 Weather VAR Model Fan Chart Time Series

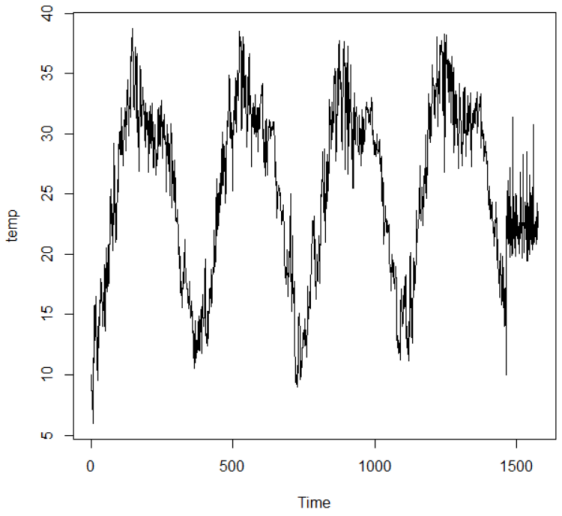

Fig. 4 Weather LSTM Network Model Time Series

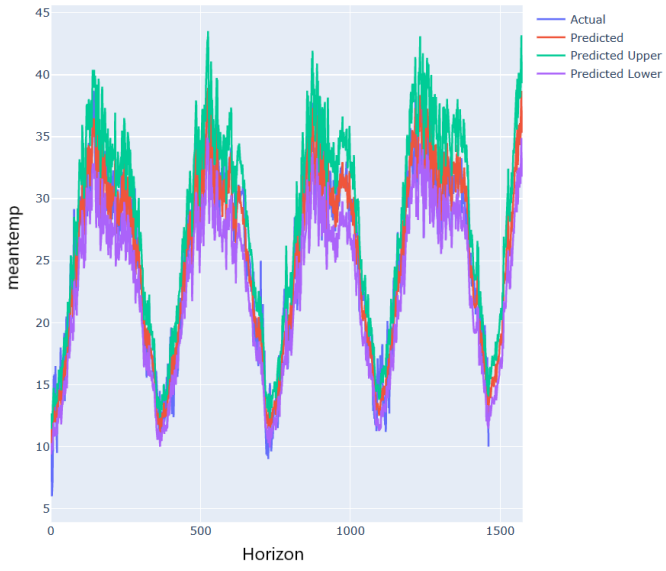

Fig. 5 Weather Facebook Prophet Model Time Series

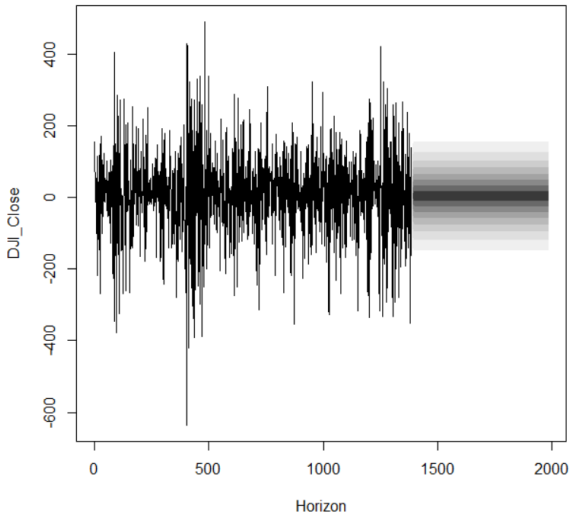

Fig. 6 Stock VAR Model Fan Chart Time Series

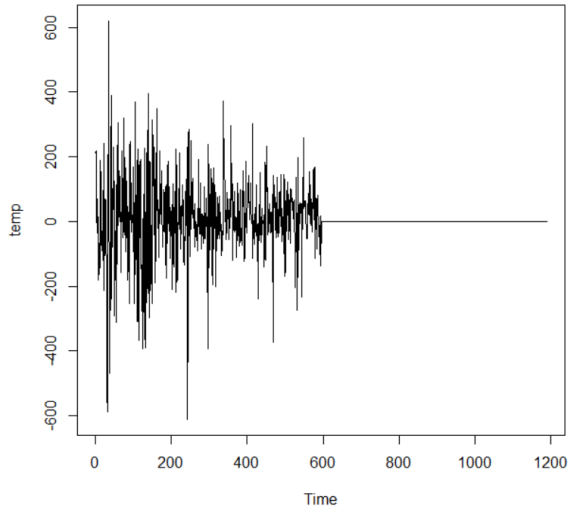

Fig. 7 Stock LSTM Network Model Time Series

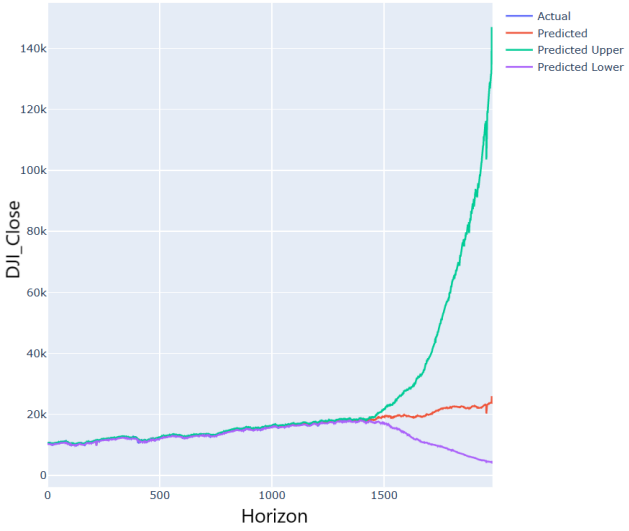

Fig. 8 Stock Facebook Prophet Model Time Series

It should be noted that each of the dependent variables that were analyzed are on a different scale from one another. For example, *meantemp* from the weather dataset saw values ranging from 6 to 38.71, while *DJI_Close* from the stocks dataset ranges from 9,686 to 23,563. So, with the RMSE for *DJI_Close* being 1,647.713, that doesn't make it significantly worse than *meantemp*, which had an RMSE of 3.889. The variable will usually perform similarly with each algorithm to each other, though exceptions do exist from this study as demonstrated by *NASDAQ_Close* performing much poorer than the other features of the stocks dataset for the LSTM neural network, but much better for the VAR algorithm.

5. Conclusion

This study demonstrates and compares the viability of a few multivariate time series forecasting algorithms. Facebook Prophet was unquestionably the best algorithm as it outperformed VAR, which performs better with smaller datasets, and LSTM neural networks, which performs better with larger datasets. The results seen for VAR and LSTM neural networks are as expected as they follow similar trends in performance from previous studies. Facebook Prophet was able to best account for the seasonal trend of data, while the other two lacked slightly in this regard for different reasons. VAR understood the seasonality, but its strength was not fully taken into regard by the model. Meanwhile, LSTM identified the trend, but failed to implement it with seasonality.

6. Future Work

A further investigation of how these algorithms perform is warranted through the means of more datasets to test them on. The datasets didn't allow forecasting to be tested for longer than a few months to around two years, due to the small amount of records both datasets contained. Datasets with prolonged periods of time should be analyzed to test the efficiency of these algorithms over prolonged periods of time. Additionally, there are certainly more multivariate time series forecasting algorithms that could be tested as well, like the Gated Recurrent Units (GRU) and CNN neural networks, Multivariate Bayesian Structural Time Series (MBSTS) algorithm, and Google's Temporal Fusion Transformer to name a few. There are also variants of the VAR algorithm to investigate as well, like the Vector-Integrated AutoRegressive Moving Average (VARIMA) algorithm.

Some of these algorithms are available in both Python and R, so a study could also be run with much larger datasets to test the efficiency of these algorithms in different programming languages.

As was discussed in the dimensionality reduction section of this paper, a study comparing and correlating various economies around the world to one another over the course of several years is also another path forward from this study, and would prove beneficial to understanding and predicting shifts in economic tides.

With how well the algorithms performed at predicting future values, further research into implementing such algorithms for data cleaning is another topic of contention to work towards in the future. Data cleaning, as a field of research within the data science community, is still young, with not a lot of attention given to it, despite considerably being the most important part of the data analysis process. A program or package created in a programming language, like R or Python, with the objective of cleaning data could prove useful to ease this process, while also demonstrating the effectiveness of these algorithms with more realistic data. However, it should be important to note the algorithms performed in this research were made for time series data, so a variety of different methods of data prediction should be used in such a program (I.e, regression, classification, forecasting, etc.).

There also seemed to be a lack of other research done with the datasets to compare the results in this research to. Research that does exist, does not contain the algorithms performed here. When more research is done on these datasets with VAR, LSTM, and/or Facebook Prophet, the results from this study can be compared accordingly.

References:

[1] T.-T.-H. Phan, É. Poisson, and A. Bigand, "Comparative Study on Univariate Forecasting Methods for Meteorological Time Series," *2018 26th European Signal Processing Conference (EUSIPCO)*, 2018, [Online]. Available: https://www.semanticscholar.org/paper/Comparative-Study-on-Univariate-Forecasting-Methods-Phan-Poisson/f746c3237891c6fac8d4d489d0e8d523d06af8f6

[2] K. Kasturi, "A Comparative Study on Univariate Time Series based Crude Palm Oil Price Prediction Model using Machine Learning Algorithms," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 4, pp. 5802–5806, Aug. 2020, doi: 10.30534/ijatcse/2020/238942020.

[3] S. Muzaffar and A. Afshari, "Short-Term Load Forecasts Using LSTM Networks," *Energy Procedia*, vol. 158, pp. 2922–2927, Feb. 2019, doi: 10.1016/j.egypro.2019.01.952.

[4] S. R. Jammalamadaka, J. Qiu, and N. Ning, "Predicting a stock portfolio with the multivariate bayesian structural time series model: Do news or emotions matter?," *International Journal of Artificial Intelligence*, vol. 17, no. 2, pp. 81–104, 2019, [Online]. Available: https://escholarship.org/uc/item/47m0302b#main

[5] I. A. Iwok, A. S. Okpe, A Comparative Study between Univariate and Multivariate Linear Stationary Time Series Models, *American Journal of Mathematics and Statistics*, Vol. 6 No. 5, 2016, pp. 203-212. doi: 10.5923/j.ajms.20160605.02.

[6] E. Hoseinzade and S. Haratizadeh, "CNNpred: CNN-based stock market prediction using a diverse set of variables," *Expert Systems with Applications*, vol. 129, pp. 273–285, Sep. 2019, doi: 10.1016/j.eswa.2019.03.029.

[7] R. Patil and R. Kulkarni, "A Review of Data Cleaning Algorithms for Data Warehouse Systems," *International Journal of Computer Science and Information Technologies*, vol. 3, no. 5, pp. 5212–5214, 2012.

[8] J. M. Z. H *et al.*, "A Survey on Cleaning Dirty Data Using Machine Learning Paradigm for Big Data Analytics," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 10, no. 3, p. 1234, Jun. 2018, doi: 10.11591/ijeecs.v10.i3.pp1234-1243.

[9] X. Wang and C. Wang, "Time Series Data Cleaning: A Survey," in *IEEE Access*, vol. 8, pp. 1866-1881, 2020, doi: 10.1109/ACCESS.2019.2962152.

[10] I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, and F. Herrera, "Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 2, p. e1289, Nov. 2018, doi: 10.1002/widm.1289.

CLASSIFICATION OF RISK LEVEL FOR MORBIDITY AND CESAREAN SECTIONS BASED ON MATERNAL RISK FACTORS

Madison Langer, Sara Trabelsi, Lauren Harding, and Raed Seetan
Department of Computer Science, Slippery Rock University, Slippery Rock, PA, 16057
{mal1033, sxt1041, lch1012, raed.seetan}@sru.edu

ABSTRACT

Maternal risk factors are analyzed to predict the susceptibility of morbidity or receiving a cesarean section. The goal of this system is to identify maternal risk factors that would put pregnant women at higher risk for morbidity and cesarean sections. Three classification models were investigated over two datasets, containing data from 1,014 and 80 patients respectively. The models were used to analyze the maternal risk factors that increase the likelihood of maternal mortality and rate of cesarean section. The data consisted of vital signs and age of the patients. The results showed that the Random Forest algorithm performed better than both Bayes Net and Attribute Selected Classifier in terms of accuracy and specificity. The accuracy was reportedly 84.5% in the Maternal Health Risk dataset, which is the highest overall, and 62.5% in the Caesarian Section Classification dataset. In terms of average accuracy, Random Forest performed the best with an accuracy of 73.5% between both datasets. Specificity for the Caesarian Section Classification dataset (73.9%) was the highest using Random Forest. Results also showed that with the Maternal Health Risk dataset, both high (96.2%) and low-risk categories (92.9%) had the highest specificity.

KEY WORDS

Maternal Risk Factors, Cesarean Section, Morbidity, Random Forest, Attribute Selected Classifier, Bayes Net.

1. Introduction

Maternal morbidity refers to any short- or long-term health problems that occur as a result of pregnancy or childbirth. It can manifest in different ways and in some cases can lead to maternal mortality [1]. In the U.S. alone, over 60,000 women are affected every year [2]. Older maternal age, abnormal blood pressure, glucose levels, and any heart conditions that may arise are all factors that can contribute to maternal morbidity [3]. Some of these factors can also be determinative of whether a cesarean section will be performed [4,5]. While generally a safe procedure, delivery by a cesarean section is associated with a higher risk of maternal morbidity compared to a vaginal delivery [6]. Not all risk factors will increase the chance of a cesarean section delivery or maternal morbidity, but it would be crucial to the patient and to health care providers to know if any of the attributes of the pregnancy would be significant enough to cause any health problems during or after the pregnancy. This can prevent maternal mortality and even fetal mortality. The purpose of this study is to identify the algorithm that works the best for classifying how vulnerable a woman is to maternal morbidity and a cesarean section delivery based on health risk factors in order for preventative measures to be taken by healthcare providers to reduce the chance of maternal morbidity.

Existing studies on this topic were limited in their research based on their choice of algorithms. Many of the same algorithms were utilized in the studies previously conducted. This study tests Random Forest, in addition to two other algorithms, Attribute Selected Classifier and Bayes Net, that were not seen in previous literature. Other studies did not address specificity in their results, which will be analyzed in this study.

Previous studies addressing risk level of morbidity based on maternal risk factors as well as cesarean sections have utilized classification analysis. Classification analysis was used to study maternal risk factors and their relationship between risk of morbidity and cesarean sections with the most commonly used algorithms being Random Forest, Support Vector Machine and Naive Bayes. One dataset used in these previous studies was collected using wearable sensors and a questionnaire that was reviewed by medical doctors at different hospitals and maternity clinics. Another dataset used in the previous studies was collected through patient records at a healthcare center.

In this study the performance of three algorithms on these datasets are evaluated: 1) Bayes Net 2) Attribute Selected Classifier and 3) Random Forest. The use of two other algorithms can further broaden the perspective of which algorithms are best for classification. By classifying this data our goal would be to identify patients' risk factors, which can then be used to build an early detection model used in medical practice to implement preventative measures and mitigate negative outcomes in this patient population. Attribute Selected Classifier is a class for running an arbitrary classifier on data that has been reduced through attribute selection. Bayes Net is a base class for a Bayes Network classifier that provides data structures and facilities common to Bayes Network learning algorithms.

Random Forest is a class for constructing random forests, which are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest [7].

The remainder of this paper is organized as follows: Section 2 introduces our main new contribution, Section 3 introduces related work, Section 4 describes the methodology, Section 5 discusses the results of the study, Section 6 concludes our study, and Section 7 provides additional recommendations and areas for further study.

2. New Contribution

The goal of this system is to correctly identify maternal risk factors that would put pregnant women at higher risk for morbidity and cesarean sections, allowing providers to better care for these patients and decrease the complications associated with these risk factors.

By implementing an early detection risk assessment tool that can accurately identify factors that would put pregnant women at higher risk for morbidity, providers could take early action and potentially mitigate negative outcomes. The model could also identify factors that put pregnant women at higher risk for cesarean sections. Oftentimes, factors that increase risk for morbidity in pregnant women overlap with factors that would put them at risk for having a cesarean section delivery. This model could be integrated into an organization's Electronic Health Record (EHR) and would extrapolate the risk factor information from an existing patient's chart. By doing this, the data could be continually updated with each office or hospital visit made by the patient. The model would place women into categories of risk level similar to the Maternal Health Risk dataset used in this study. The model would pull data from the chart such as age, vital signs (heart rate, blood pressure), gravidity and parity, current gestational age, lab results including blood glucose, history of preterm deliveries or miscarriages, history of previous cesarean sections, and any number of other factors that could impact the risk level of the pregnancy or whether that patient could have a cesarean section. After compiling the data, the model would operate under thresholds determined and set by medical doctors to then place these patients into risk levels. When placed in low, medium, or high-risk categories clinicians would be aware of this risk level and could better provide care and educate their patients depending on their risks.

One action that can be taken as a result of a patient being flagged as medium or high-risk is implementing continuous monitoring. Monitoring vital signs such as blood pressure, temperature, and heart rate can quickly determine whether or not a woman should seek medical attention. By electronically recording their vital signs at home, patients and their providers can be alerted. When providers are notified, they could instruct their patients to report to their office or a higher level of care immediately if the recorded data is out of range. This system would increase safety and reduce preventable complications or even death of pregnant women and their neonates.

This model could also be used on an organizational level for the proper allocation of resources towards high and medium-risk patients. For example, if an organization has a large amount of high and medium risk patients that need more frequent bloodwork, ultrasounds, other forms of testing, or office visits, additional funding can be given to this area at that given time. During periods of large amounts of low-risk patients, the resources could be allocated to other areas. This could save them money in the long term. This could be especially important for healthcare organizations that are smaller and have limited funding and personnel.

People living in rural areas especially are known to have trouble with accessibility to healthcare as there are not many hospitals located in those areas. One concern that had risen a few years ago is that there are several rural areas that did not have obstetrician and gynecological services that had training on performing cesarean sections leaving many women in those areas at risk of maternal morbidity and mortality [8]. There have been plans to increase training on cesarean sections for family physicians, but that could take a while to implement [8]. With the use of classifier algorithms to identify how at risk the women are for cesarean sections or maternal morbidity, it gives the women a chance to find a physician nearby that is comfortable performing cesarean sections or find accommodations closer to the hospital when their due date is near. This could significantly reduce the maternal morbidity rates and perinatal mortality in rural areas.

3. Related Work

There have been several studies in the healthcare domain that have used data mining techniques to predict delivery by cesarean section and risk level in pregnant women.

For example, in one study classifiers were applied to the Caesarian Section Classification dataset that was utilized in this study. Six different algorithms were tested, including Naive Bayes, Support Vector Machine, K-Nearest Neighbor, OneR, J48, and Random Forest. The study found that Naive Bayes was able to correctly classify unseen cases and was the most accurate algorithm at 65%. However, all of the algorithms tested did not have large differences in accuracy. This dataset used was based on real-life data, while other similar studies utilized training sets. This study also utilized the cross-validation approach to ensure more realistic and reliable results. [9]

Another study also used the Caesarian Section Classification dataset utilized in this study. The study

aimed to create a healthcare operational decision-making system using machine learning classifiers. The goal was to assist doctors in making the best decisions for a given patient, specifically to identify whether a cesarean section surgery should be performed. They chose to test the system on cesarean sections as the surgery helps to save woman and child, and as it is the most commonly performed obstetric surgery in the world. Five different algorithms were tested, including Support Vector Machine, Random Forest, Naive Bayes, K-Nearest Neighbor, and Logistic Regression. The study found that K-nearest Neighbors and Random Forest were the most accurate at 95%. [10]

Another predictive model monitored pregnant women by analyzing their health data and identifying their risk intensity level. The researchers of this study collected the maternal health data that is being used in this study. Their focus was the development of wearable sensors to monitor pregnant patients and identify patients at risk even when they are in a remote location. The study's concern was in improving maternal health and reducing maternal and child mortality as these rates have not yet declined to meet the goal set by the United Nations. They tested eight different algorithms, including Decision Tree, Random Forest, Support Vector Machine, Sequential Minimal Optimization, Logistic Regression, Naive Bayes, IBk, and Logistic Model Tree. The study found that Decision Tree (98.51%) was the most accurate with Random Forest (98.42%) and IBk (98.32%) as close seconds. [11]

Another study used the Pima Indian Diabetes dataset, a diabetes dataset for women that was filtered and categorized according to maternal risk level. This study similarly sought to monitor pregnant women with wearable sensors in order to notify the women, their families, and respective doctors of their condition. They tested five different algorithms, including Sequential Minimal Optimization, IBk, Logistic Model Tree, Naive Bayes, and Logistic Regression. The study found that Logistic Model Tree was the most accurate with a rate of 97.96%. [12]

This study is different from the previous literature as it will test not only Random Forest, seen in previous works, but it will test two other algorithms, Attribute Selected Classifier and Bayes Net. These algorithms were not seen in previous literature. Other studies did not address specificity in their results, which will be analyzed in this study.

4. Methodology

4.1 Dataset

There are two datasets used in this study. The Caesarian Section Classification dataset [13] contains information on 80 pregnant women. This data was collected from a healthcare center in Tabriz, Iran through patient records. It ranks specific attributes of their pregnancy such as their age, delivery number, delivery time, blood pressure and whether they delivered vaginally or by cesarean section. There was a range of ages among the women with the youngest being 18 years old and the oldest being 40 years old. The delivery time was classified as zero, one, and two for timely, premature, and latecomer. The women's blood pressure was ranked 2 for high, 1 for normal, and 0 for low. Any heart problems were classified with a 0 for apt and a 1 for inept.

The Maternal Health Risk dataset [14] contains a sample of 1014 women in Dhaka and Khulna, Bangladesh. This data was obtained through the use of wearable sensors as well as using a questionnaire at different hospitals and maternity clinics. This dataset contains the ages of the women, their systolic and diastolic blood pressure values, their blood glucose level, their heart rates, and the predicted risk intensity level during their pregnancy. The identification of risk level was determined by medical doctors who reviewed the questionnaires.

The datasets were analyzed using Weka [15] version 3.6.14. Bayes Net, Attribute Selected Classifier, and Random Forest were utilized and compared. Accuracy, sensitivity, and specificity were used to compare the results. These measures will be discussed in greater detail in section 4.3.

4.2 Preprocessing & Analysis Preparation

Before the Maternal Health Risk dataset can be classified it needs to be preprocessed to show the best results. There was a slight class imbalance between the high-risk and low-risk cases. Before running the class balancer filter on Weka, looking at the dataset revealed that there may be some cases of duplication in the data set that would need to be removed in order for the classifier to produce more accurate results. After the duplicates were removed, the class balancer filter was applied.

Since there were only 80 instances for the Caesarian Section Classification dataset, it was much easier to analyze. With so few instances, there was also worry that there would be duplicates that would reduce the number, but after viewing the data there did not seem to be anything wrong with it. There was a slight class imbalance between cases which was resolved with the Class Balancer filter on Weka.

When using the Bayes Net algorithm to classify both datasets, some parameters were modified. For the Maternal Health Risk dataset the estimator was changed to simulated annealing which works well for a larger dataset. Debugging was also switched to true so that any errors can be removed for the best possible results. The simulated annealing estimator did not work well for the Caesarian Section Classification data set possibly due to it having a smaller sample size. The BMA estimator was chosen for the Caesarian Section Classification dataset as it provided the best results for the small sample size.

The Attribute Selected Classifier had the most modification in terms of parameters. The classifier chosen for both datasets was J48 so that data could be seen categorically and continuously. The search parameter was changed to ranker as it ranks attributes by their own individual evaluation. This paired with the symmetrical uncertainty attribute evaluator gave the best results for the Maternal Health Risk dataset. The same parameters were kept for the cesarean data except the evaluator was changed to the classifier subset evaluator as it gave better results possibly due to the difference in sample sizes.

The Random Forest algorithm parameters were kept at default settings as any changes reduced the accuracy, precision, and specificity. The only change made was the debugging parameter was turned on to identify and remove any errors in the classifying process.

4.3 Analysis Methods

The following three algorithms were used: 1) Attribute Selected Classifier 2) Bayes Net and 3) Random Forest. Attribute Selected Classifier is a class for running an arbitrary classifier on data that has been reduced through attribute selection. Bayes Net is a base class for a Bayes Network classifier that provides data structures and facilities common to Bayes Network learning algorithms. Random Forest is a class for constructing random forests, which are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest [4]

For each algorithm, 10-fold cross validation and Percentage Split of 66% (Split) were applied on the two datasets.

The results of the three algorithms on the two datasets were analyzed and compared. This study compares the following results:

1. Accuracy, which is the percentage of correctly classified instances. This was obtained from the Weka output.
2. Sensitivity, which is the proportion of positive instances that were correctly identified. This is calculated using the formula: TP / (TP + FN), where TP is the number of true positives and FN is the number of false negatives.
3. Specificity, which is the proportion of negative instances that were correctly identified. This is calculated using the formula: TN / (TN + FP), where TN is the number of true negatives and FP is the number of false positives.

Table 1
Weka Scheme & Filters

Algorithm	Type	Weka Attribute
Bayes Net	Scheme	weka.classifiers.bayes.BayesNet -D -Q weka.classifiers.bayes.net.search.local.SimulatedAnnealing -- -A 10.0 -U 10000 -D 0.999 -R 1 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator -- -A 0.5
	Filter	weka.filters.supervised.instance.ClassBalancer-num-intervals10
Attribute Selected Classifier	Scheme	weka.classifiers.meta.AttributeSelectedClassifier -E "weka.attributeSelection.CfsSubsetEval -P 1 -E 1" -S "weka.attributeSelection.BestFirst -D 1 -N 5" -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
	Filter	weka.filters.unsupervised.attribute.Remove-V-R5,4,3,1-2,6
Random Forest	Scheme	weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
	Filter	Default settings

This study will identify the algorithm that best classifies both datasets according to accuracy, sensitivity, and specificity.

5. Results

As shown in Figure 1, Random Forest provided the most accuracy with the Maternal Health Risk dataset with 84.5% accuracy. Bayes Net provided the most accuracy with the Caesarian Section Classification dataset with 67.3% accuracy. Attribute Selected Classifier performed the poorest in this metric, with 78.6% accuracy for the Maternal Health Risk dataset and only 57.6% accuracy for the Caesarian Section Classification dataset.

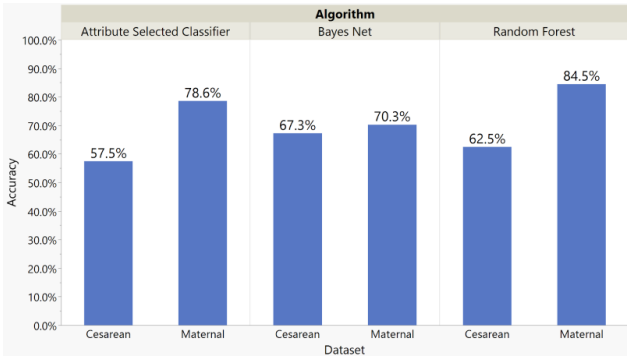

Fig 1. Average accuracy of the Caesarian Section Classification and Maternal Health Risk datasets across Attribute Selected Classifier, Bayes Nets, and Random Forest

In regard to sensitivity, Figure 2 shows that Bayes Net correctly classified the positive instances (no cesarean section) 64.7% of the time with the Caesarian Section Classification dataset. Random Forest performed the poorest in this metric at 47.1%.

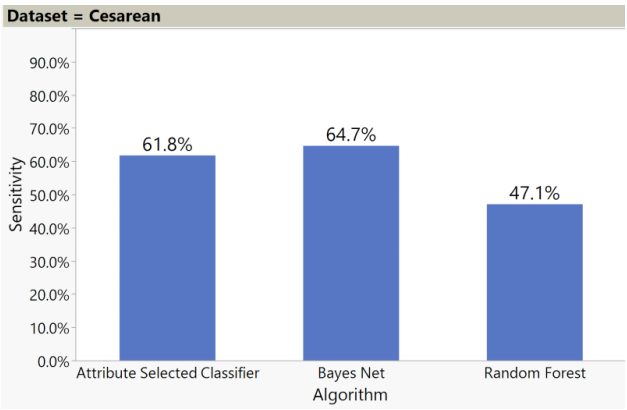

Fig 2. Average sensitivity of the Caesarian Section Classification dataset across Attribute Selected Classifier, Bayes Nets, and Random Forest

Because the Maternal Health Risk dataset is a multi-class dataset and cannot use binary classification, the sensitivities of each individual risk level were calculated. Figure 3 shows that with the Maternal Health Risk dataset, Random Forest correctly classified the positive instances greater than 81.0% of the time in the low, mid, and high-risk categories. High-risk patients, arguably our most important risk level to be classified correctly, were correctly classified 90.1% of the time. Bayes Net placed mid-risk patients in the mid-risk category, remarkably, only 38.4% of the time. A closer look at the Confusion Matrix shows that mid-risk patients were mostly being incorrectly placed in the low-risk category. This metric is greatly problematic as our goal is to identify at-risk patients and would not want patients being misclassified as low-risk as this would underestimate their risk level.

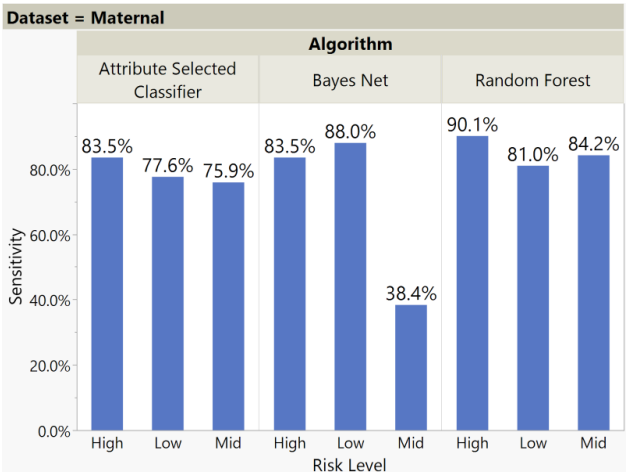

Fig 3. Average sensitivity of risk levels in the Maternal Health Risk dataset across Attribute Selected Classifier, Bayes Nets, and Random Forest

In regard to specificity, Figure 4 shows that Random Forest performed the best in this metric with the Caesarian Section Classification dataset and correctly classified the negative instances (cesarean section) correctly 73.9% of the time.

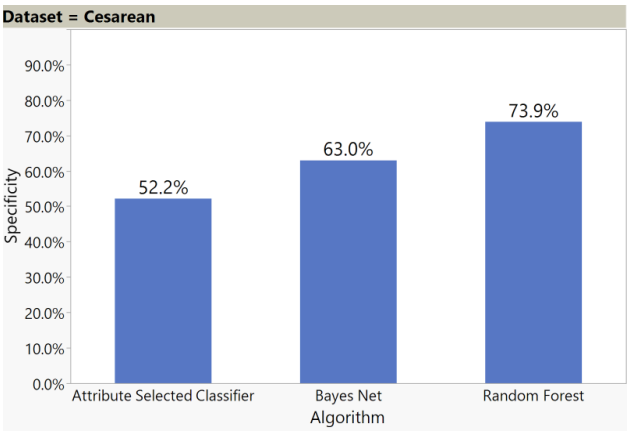

Fig 4. Average specificity of the Caesarian Section Classification dataset across Attribute Selected Classifier, Bayes Nets, and Random Forest

Because the Maternal Health Risk dataset is a multi-class dataset and cannot use binary classification, the specificities of each individual risk level were calculated. Figure 5 shows that Random Forest correctly classified the negative instances greater than 87.3% of the time in the low, mid, and high-risk categories. Bayes Net performs the poorest in this metric: 70.1% of the time the algorithm predicts that the patient should not be low-risk and is correct, meaning that 29.9% of the time the algorithm predicts that the patient would not be low-risk but is. This would be problematic in practice as this is overestimating risk level. This could lead to over allocation of resources by the healthcare organization towards patients that are thought to be mid or high-risk but are in fact low-risk, such as additional testing, bloodwork, or office visits that are not clinically indicated. This would not only save the organization money, but also cut down on unnecessary medical expenses for the patient.

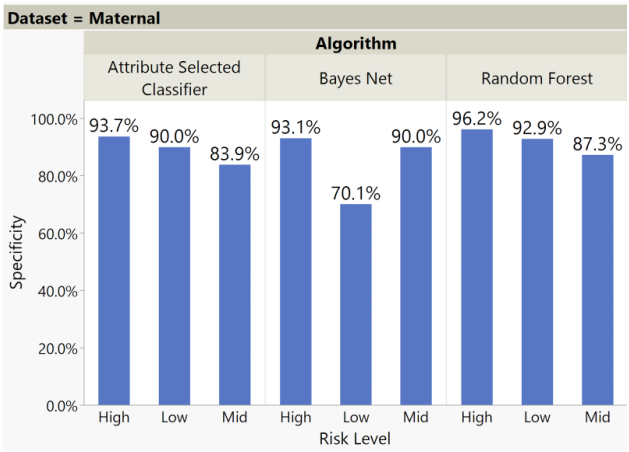

Fig 5. Average specificity of the risk levels in the Maternal Health Risk dataset across Attribute Selected Classifier, Bayes Nets, and Random Forest

6. Conclusion

This study demonstrates the effectiveness of Random Forest in classifying pregnant patients into risk levels and whether or not patients received a cesarean section. The Random Forest algorithm achieved an accuracy rate of 84.5% with the Maternal Health Risk dataset and 62.5% with the Caesarian Section Classification dataset, or an average accuracy of 73.5% between both datasets. It correctly classified positive instances of the high-risk class 90.1% of the time, mid-risk class 84.2% of the time, and low-risk 81% of the time. It correctly classified negative instances of the high-risk class 96.2% of the time, mid-risk class 87.3% of the time, and low-risk class 92.9% of the time. It correctly classified having a cesarean section 73.9% of the time. The only metric in which this algorithm did not perform well was classifying not having a cesarean section at 47.1%.

In this study different algorithms were tested and analyzed them with metrics not previously seen in other literature. Our results ultimately showed that Random Forest performed the best out of our three algorithms, similar to what some previous studies found.

In studying these algorithms and finding the high rates of accuracy, sensitivity, and specificity in our datasets run with Random Forest, it allows us to accurately identify pregnant patients who are at risk for morbidity and having a cesarean section. This algorithm performs well in these metrics, making it feasible to implement an early detection risk assessment tool that would operate complementary to the oversight of providers.

7. Future Work

Further recommendations that would benefit this area of study in the future include implementing datasets with larger study groups. For our dataset on Cesarean Sections, the study group consisted of only 80 participants. This issue led to lower accuracy rates for this specific dataset.

Another additional area of study would be to include other algorithms not previously tested in related literature. Another possible recommendation for future study is to investigate and implement more attributes, or risk factors, that are significant in predicting maternal morbidity or cesarean sections into our study. With more attributes larger amounts of data can be analyzed. By analyzing Big Data to create a new algorithm, the best possible results will be provided.

References:

[1] Maternal mortality. World Health Organization. (n.d.). Accessed November 4, 2022. https://www.who.int/news-room/fact-sheets/detail/maternal-mortality.

[2] Maternal mortality. World Health Organization. (n.d.). Accessed November 4, 2022. https://www.who.int/news-room/fact-sheets/detail/maternal-mortality.

[3] U.S. Department of Health and Human Services. (2020, May 14). What factors increase the risk of maternal morbidity and mortality? National Institute of Child Health and Human Development. Accessed November 4, 2022. https://www.nichd.nih.gov/health/topics/maternal-morbidity-mortality/conditioninfo/factors.

[4] Gestational diabetes and pregnancy. Centers for Disease Control and Prevention. (2022). Accessed November 4, 2022. https://www.cdc.gov/pregnancy/diabetes-gestational.html.

[5] Rydahl, E., Declercq, E., Juhl, M., & Maimburg, R. D. (2019). Cesarean section on a rise-Does advanced maternal age explain the increase? A population register-based study. *PloS one*, 14(1), e0210655. https://doi.org/10.1371/journal.pone.0210655.

[6] Leonard, S. A., Main, E. K., & Carmichael, S. L. (2019). The contribution of maternal characteristics and cesarean delivery to an increasing trend of severe maternal morbidity. *BMC pregnancy and childbirth*, 19(1), 16. https://doi.org/10.1186/s12884-018-2169-3.

[7] Breiman, L. Random Forests. Machine Learning 45,5–32(2001). https://doi.org/10.1023/A:1010933404324

[8] Tong, S. T., Eden, A. R., Morgan, Z. J., Bazemore, A. W., & Peterson, L. E. (2021). The Essential Role of Family Physicians in Providing Cesarean Sections in Rural Communities. *Journal of the American Board of Family Medicine: JABFM*, *34*(1),10–11. https://doi.org/10.3122/jabfm.2021.01.200132

[9] Jamjoom, Mona. (2020). Data Mining in Healthcare to Predict Cesarean Delivery Operations using a Real Dataset. 20-26. 10.5220/0010366700200026.

[10] Amin, Muhammad & Ali, Amir. (2017). Performance Evaluation of Supervised Machine Learning Classifiers for Predicting Healthcare Operational Decisions. 10.13140/RG.2.2.26371.25127.

[11] Ahmed, Marzia & Kashem, Mohammod. (2021). IoT Based Risk Level Prediction Model For Maternal Health Care In The Context Of Bangladesh. 10.1109/STI50764.2020.9350320.

[12] Ahmed, Marzia & Kashem, Mohammod & Rahman, Mostafijur & Khatun, Sabira. (2020). Review and Analysis of Risk Factor of Maternal Health in Remote Area Using the Internet of Things (IoT). 10.1007/978-981-15-2317-5_30.

[13] Caesarian Section Classification Dataset Data Set, https://archive.ics.uci.edu/ml/datasets/Caesarian+Section+Classification+Dataset, 02.11.2018.

[14] Maternal Health Risk Data Set Data Set, https://archive.ics.uci.edu/ml/datasets/Maternal+Health+Risk+Data+Set, 31.12.2020.

[15] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

ON THE IMPOSSIBILITY OF DIRECTLY PROVING OR DISPROVING THE COLLATZ CONJECTURE USING A COMPUTER PROGRAM

Brandon Packard
CU-GAME, Pennsylvania Western University
bpackard@pennwest.edu

ABSTRACT

The Collatz Conjecture is an unsolved problem in mathematics which has plagued mathematicians for decades. Its intuitive and seemingly simple formula belies incredibly complex behavior, and no attempt to prove or disprove it has been successful. With the creation of more powerful computers and more accessible programming languages, many have turned to attempting to write computer programs to prove or disprove this conjecture. In this paper, we show that directly proving or disproving the Collatz Conjecture via a computer program not only hasn't been done, but is in fact an impossible task for modern computers.

1 Introduction

The Collatz Conjecture has been an unsolved problem in mathematics for decades. This conjecture will be explained in more detail in Section 3, but it basically states the following: Given a positive whole number, repeatedly either multiply it by 3 and add 1 (if it's odd) or divide it by 2 (if it's even), and your number will eventually arrive at 1. Many people have tried to prove or disprove it, but its seemingly simple formula belies its incredibly complex behavior. It has been stated that modern mathematics "is not ready for this sort of problem" [1], and many a budding mathematics researcher has been gently steered away from this problem.

With the explosion in popularity of computers and programming languages, a new strategy has arisen for solving the problem - the use of computer programs. Computer programs have been used to test quintillions of numbers to attempt to find a solution. Computers have been used to solve many problems and improve almost every aspect of our everyday lives. However, there is a fatal flaw in trying to directly solve the Collatz Conjecture with a computer program.

In this paper, we will demonstrate that writing a program to prove or disprove the Collatz Conjecture is undecidable – that is, a computer program can't directly solve the problem. This isn't to say that tools can't be made to help with a solution, because programs can help test assumptions, find patterns, and so on. It also doesn't mean that the Collatz Conjecture itself is undecidable, but rather that directly solving it with a program is simply not possible.

The rest of this paper is organized as follows. First, we will discuss some of the related work that has been done on the Collatz Conjecture in the field in Section 2. Next, we will describe the conjecture in depth in Section 3, followed by a description of the Halting Problem in Section 4, an important ingredient for our discussion. We will then show in Section 5 that none of the potential ways to prove or disprove the Collatz Conjecture are feasible to do on a computer, even given infinite memory and processor time. Finally in Section 6, we will end the paper with some conclusions and implications for trying to solve this problem.

2 Related Work

There have been dozens if not hundreds of publications on the Collatz Conjecture, far too many to cover here. Instead, we will focus on publications which use programs as part of the process to either prove, disprove, or garner more information about this problem. First, Honda, Ito, and Nakano created a program to verify that various numbers would return to 1. Their approach was able to verify 1.31^{12} 64-bit numbers per second, for a total of 2^{30} but they were able not able to prove or disprove the conjecture [2]. Others have used computer programs to verify much higher numbers, up to 2^{68}, with the same results of them all eventually returning to 1 [3].

In a fairly novel approach, Perez attempts to prove that the Collatz Conjecture is false, by writing a program that finds integers with ever-greater stopping times. He reasons that since the algorithm does not halt and continues to find larger stopping times, there must be a number which takes infinite time to stop [4]. This is an interesting insight, but as discussed in Section 4, we cannot know if such an algorithm will halt.

Also, Gurbaxani attempted to create different programs for different modifications of the Collatz Conjecture, with the hope of both finding other interesting problems and giving some insight into whether or not the conjecture is true [5]. Mirkowaska and Salwicki [6] use programming to attempt to show that the Collatz Conjecture itself is undecidable, but do not examine the undecidability of creating a program to directly solve it. Other computational techniques, such as linear search and data visualization, have also been used for the purpose of garnering more information about this problem [6].

3 The Collatz Conjecture

The Collatz Conjecture is a well-known and currently unsolved program in Mathematics. It is also known by many other names, such as the 3n+1 problem, the Ulam Conjecture, and the Syracuse Problem, to name a few. Regardless of what you call it, the Collatz Conjecture is a seemingly simple problem with unexpectedly complex behavior that has eluded any attempts by amateur and professional mathematicians to prove it true or false for decades.

Algorithm 1: Process for the Collatz Conjecture.

```
function Collatz(n)
while True do
    if n is odd then
        n = 3n + 1
    end
    else
        n = n / 2
    end
end
```

As seen in Algorithm 1, the process/formula for the Collatz Conjecture is quite straightforward. Given some number *n*, we check if it is even or odd. If it is odd, we get a new number by multiplying *n* by 3 and adding 1. If it is even, we get a new number by dividing *n* by 2. We then repeat this process forever. The Collatz Conjecture states that for every positive integer, repeating this process over and over (as illustrated by the *while True*) will eventually result in repeating the numbers 1, 4, 2 infinitely. For example, let's start with 13 and see what happens:

- 13 is odd, so 3*13 + 1 $\Rightarrow$ 40
- 40 is even, so 40/2 $\Rightarrow$ 20
- 20 is even, so 20/2 $\Rightarrow$ 10
- 10 is even, so 10/2 $\Rightarrow$ 5
- 5 is odd, so 3*5 + 1 $\Rightarrow$ 16
- 16 is even, so 16/2 $\Rightarrow$ 8
- 8 is even, so 8/2 $\Rightarrow$ 4
- 4 is even, so 4/2 $\Rightarrow$ 2
- 2 is even, so 2/2 $\Rightarrow$ 1
- 1 is odd, so 3*1 + 1 $\Rightarrow$ 4
- 4 is even, so 4/2 $\Rightarrow$ 2
- 2 is even, so 2/2 $\Rightarrow$ 1
- {1,4,2} repeats forever

This $\{1,4,2\}$ loop is often simplified to the statement that $Collatz(n)$ eventually arrives at the number 1.

We can refer to showing that the Collatz Conjecture is either True or False (proving or disproving it) as "solving" the problem. In order to solve it, one of these three things would need to be achieved:

1. Prove that all positive integers eventually arrive at 1 when the formula is repeatedly applied to them.
2. Prove that there is some positive integer s which will grow infinitely when the formula is repeatedly applied to it.
3. Prove there is some positive integer s which results in a loop that does not include 1 when the formula is repeatedly applied to it.

It has been verified via computer programs that this conjecture holds true for all positive integers up to at least 2^{68} (that is, roughly 295 quintillion) [3], and many attempts to prove or disprove the Collatz Conjecture use programming in some way, as discussed in Section 2. Although potentially useful tools can be created via programming, there is one major obstacle to directly solving the Collatz Conjecture using a program – the Halting Problem.

4 The Halting Problem

With some programs, we can know for sure that they halt, or eventually terminate and stop running. For example, a program that does nothing but adds two numbers together is guaranteed to halt. However, this is not true in the general case. The Halting Problem is the problem of determining, given some program and some input to that program, whether the program will ever halt, or just continue to run forever.

As proven by Alan Turing, this problem is *undecidable* [7]. That is to say, it is impossible to know whether programs will stop running or not in the general case. We don't typically encounter this problem in our own code, since we purposely write most programs to eventually terminate, specifically avoiding infinite loops and other similar issues. However, when making programs to model real-world scenarios, such as mathematical properties of certain numbers, we can't always be sure that they will terminate.

In fact, the Halting Problem being undecidable directly leads to it being impossible for there to be an algorithm which decides whether a particular statement about natural numbers is true or not. This is because if we had a particular input and a particular program, and a proposition about whether or not it would halt, it could be converted into an equivalent statement about natural numbers. Then, if an algorithm could find whether or not that statement about natural numbers is true, it could also determine whether the original program halts, thus violating that the Halting Problem is undecidable. This is a key insight for the upcoming discussion.

5 Proving that The Collatz Conjecture is Undecidable Using A Computer Program

So how does this relate back to the Collatz Conjecture? Recall that in order to solve the problem, we either need to prove that it's true for all positive integers, prove that a number grows infinitely, or prove that a number results in a loop which doesn't include the number 1.

5.1 Initial Thoughts from Intuition

Let's start by looking at these three possible solutions from an intuitive standpoint. In order to prove that the Collatz Conjecture holds true for all positive integers using a computer program, we would need to test all positive integers. As there are an infinite number of them, such a program would never finish, even with unlimited resources.

Similarly, let's say that we find a value of n that does in fact grow forever. In order to verify our solution via a computer program, we would have to run the program forever. As we cannot do this, and in fact cannot even know if a program *is* running forever, we cannot verify the solution.

Finally, to determine if a value of n just loops forever (without arriving at 1), we could also need to run the program forever - the loop could consist of infinite numbers, or just too many numbers to parse in a feasible amount of time, and we have no idea how far towards infinity a potential solution number may lie. In addition, we are unable to verify that a number does *not* loop forever, as this would require knowing if it *does* grow forever (if it doesn't loop, it must return to 1 or grow forever). At the very least, mathematicians have calculated that such a loop must consist of at least 186 *billion* numbers [3], and we know it is not in the first 2^{68} of them. As such, even if there is a solution number that starts a loop, directly finding one programmatically seems incredibly improbable.

Let us now leave intuition behind and prove that none of these three solutions are feasible via computer program.

5.2 Proving we Can't Prove the Collatz Conjecture

The proof for determining that we can't prove the Collatz Conjecture is true via a program is quite straightforward. In order to prove it is true, we have to write a program which shows it is true for all positive integers - of which there are an infinite amount. Even if we use shortcuts to knock out some numbers, such as not bothering to test any powers of 2 since those are just repeatedly divided by 2 until they hit 1, it still leaves an infinite number of positive integers. Additionally, infinitely large numbers cannot even be stored on a computer, much less used in mathematical calculations. As such, no matter how much memory or processing time we have, we can never test every single possible positive integer. Therefore, writing a program which verifies that the Collatz Conjecture holds true for all positive integers is not possible.

5.3 Proving we Can't Prove That a Number Grows Forever

We assert that determining if a number grows forever is an impossible task for a computer program. In order to prove this, we will use the classic method of Proof by Contradiction, specifically by reducing the Halting Problem to this problem and showing that if we solve the latter, we must be able to solve the former. First, let's assume that we have created a function, *grows(n)*, that for any positive integer n returns true if it grows forever when applying the formula in the Collatz Conjecture to it, and false if it eventually reaches 1 instead.

Algorithm 2: The HaltTest function, which will run infinitely whenever grows(n) returns true, and immediately halt whenever when grows(n) returns false.

```
function HaltTest(n)
if grows(n) == True then
    while True do
    end
end
else
    return
end
```

Using this function, we can then construct *HaltTest(n)*, the algorithm shown in Algorithm 2. Basically, HaltTest calls grows(n) using the same value, n, that it was given. If grows(n) returns true, HaltTest then infinitely loops, meaning it never halts. On the other hand, if grows(n) returns false, HaltTest immediately halts instead. This is our *reduction* from the Halting Problem to the problem of finding a number that grows infinitely according to the formula used in the Collatz Conjecture. Therefore, if grows(n) is able to determine if a number grows forever, it is also able to solve the problem of whether or not the program will halt.

That is to say, assuming the existence of the function grows(n), we can determine whether or not the original program halts, *thereby solving the halting problem*. Since the Halting Problem is undecidable, this means that we have a contradiction, and the grows(n) function cannot exist! As such, creating a program to successfully test whether or not a given positive integer grows infinitely *must be impossible*, so verifying that a potential solution to the Collatz Conjecture does so is also impossible.

5.4 Proving we Can't Prove That a Number Loops Forever

We assert that determining if a number loops forever without including the number 1 is also an impossible task for a computer program. In order to prove this, we will again use the classic method of Proof by Contradiction, specifically by reducing the Halting Problem to this problem as well. In fact, the proof for this is virtually the same as when trying to find a number that can grow forever, but we will walk through it for completeness. First, let's assume that we have created a function, *loops(n)*, that for any positive integer n returns true if applying the formula in the Collatz Conjecture to it creates a loop of numbers not including 1, and false if it eventually arrives at 1 instead.

Using this function, we can construct *HaltTest2(n)*, the algorithm shown in Algorithm 3. Basically, HaltTest2() calls

Algorithm 3: The HaltTest2 function, which will run infinitely whenever loops(n) returns true, and immediately halt whenever when loops(n) returns false.

```
function HaltTest2(n)
if loops(n) == True then
    while True do
    end
end
else
    return
end
```

loops(n) using the same value, n, that it was given. If loops(n) returns true, HaltTest2() then infinitely loops, meaning it never halts. On the other hand, if loops(n) returns false, HaltTest2() immediately halts. Notice that this means we have created a reduction from the Halting Problem once again – if loops(n) solves the problem of determining if a number loops without arriving at 1, it also solves the problem of whether or not the program will terminate.

That is to say, assuming the existence of the function loops(n), we can determine whether or not the original program halts, *thereby solving the halting problem*. Since the Halting Problem is undecidable, this means that we have a contradiction and the loops(n) function cannot exist! Therefore, creating a program to successfully test whether or not a given positive integer loops infinitely *must also be impossible*. Also note that we cannot verify that a number does not loop infinitely in all cases, since it may grow forever, which offers more evidence that loops(n) cannot exist.

5.5 Wrapping it all Up

As we have seen, it is impossible to write a computer program to find any of the three solutions provided in Section 3 – verifying all positive integers arrive at 1, finding a positive integer which grows forever, or finding a loop that does not arrive at 1. It logically follows that it is impossible to solve the Collatz Conjecture directly using only a computer program.

One potential counterargument to this is that we *can* determine if grows(n) or loops(n) is valid for some values – after all, this has essentially been done for all values from 1 to 2^{68}, as previously mentioned. This parallels the fact that we *can* determine if some programs will halt. However, just as we can determine if some programs halt but not all programs, we can determine if some values would return true or false for grows(n) or loops(n) – but not *all* values. To confirm this, let's think about this in a bit more depth.

First, let us consider grows(n). It is very important to note that if a number is found for which grows(n) returns true, there must be an *infinite* number of numbers for which grow returns true! To see this, assume a number x is found that grows forever. If x is odd, that means that $3x + 1$ must also

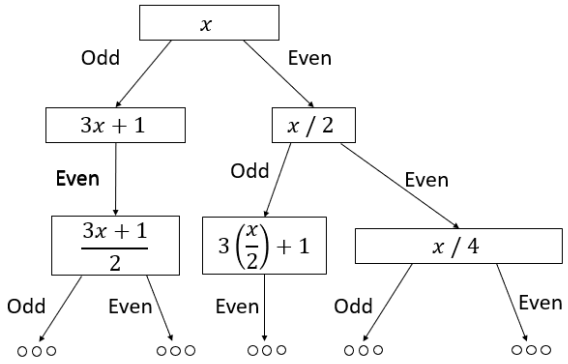

Figure 1: A tree showing the numbers that must also grow infinitely if some number x grows infinitely, depending on whether each number is even or odd. Note that if a number is odd, causing us to do 3x+1 on it, the resulting number will always be even.

.

grow forever. If x is even, that means that $x/2$ must also grow forever. This then repeats for the new number, as shown in Figure 1. As such, there are an infinite number of numbers which must be verified for grows(n) to return true for any number, which takes us back to the program never halting!

Also, let us consider loops(n). Recall that loops(n) can determine if *any* positive integer will loop without arriving at a value of 1. Also recall that mathematicians have calculated that if a loop exists, it must consist of *at least* 186 billion numbers. In theory, that means that we might only need to run loops(n) for 186 billion different values – which could be feasible. However, in order to only run loops for a value that is a solution, *the solution must already be known*. Otherwise, as is the case here, we are looking at the set of all positive integers in general, which fits the Halting Problem criteria. After all, a function that determines if a positive integer loops without hitting 1 must be able to verify this for any positive integer. To this end, we need to realize that we cannot verify that a number does NOT loop in finite time - it could be growing infinitely instead, further showing creating this function is not possible!

Another counterargument comes from the idea of Fermat Primes. A Fermat Prime is a number of the form $2^{2^m} + 1$ that is prime. One might construct a function fermat(n) which determines if a number is a Fermat prime. If we plug in this function as we do with grows(n) and loops(n), it seems as those a solution to this must also be impossible. However, there are only 5 known Fermat primes (3,5,17,257, and 65537). Euler was able to show in 1732 that $m = 6$, which is 4,294,967,297, is not prime (it is evenly divisible by 641 and 6,700,417), and it is postulated that no higher number of the form $2^{2^m} + 1$ is prime either. What's important to note is that although finding all the factors of 4,294,967,297 was an incredibly hard problem back then, it is a computationally trivial problem today. In fact, it can be done in just 7 (and possibly less) lines of Python code today, and takes less than a second to compute.

As such, given our modern tools, it can be said that the idea that all non-negative integer values of m are prime has a *trivial counterexample*. The same is true for many other claims as well – all numbers are even, all numbers divisible by 3 are also divisible by 4, and so on. These problems are so relatively simple that they also have easy counterexamples. However, the Collatz Conjecture does not have a trivial counterexample (as evidenced by the community working on this for decades and having tested 2^{68} different numbers), with proving it false requiring potential clashes with infinity. This is a key difference that makes the Halting Problem apply to the Collatz Conjecture but not something like Fermat Primes.

6 Conclusion

In conclusion, the Collatz Conjecture has been worked on for decades, but a proof of whether it holds true or not has proven to be incredibly elusive. Many have turned to computer programs to try to solve the problem, often by trying to find a counterexample that makes the conjecture be false, but also verifying that it holds true for many numbers. In this work, we have shown that even with computers much more powerful than what we have now, creating a program that directly proves or disproves this conjecture is impossible. To verify that all positive integers arrive at 1, we would have to verify an infinite amount of numbers using a computer program, which is impossible because the program could never test every single number when there are an infinite amount of them. If we were able to create a function that determined if a positive integer grew forever or created a loop which doesn't arrive at 1, it would allow us to solve an impossible problem (the Halting Problem) since both of these functions could require an infinite amount of processor time. As such, directly solving this problem using a computer program is impossible.

Note that this conclusion does not solve the Collatz Conjecture or imply that it is undecidable - just that we cannot create a program to directly solve it. Mathematicians have previously used programs to look for patterns of numbers, which they then use to construct induction-based proofs. This has an interesting implication for solving the problem - although tools could still potentially be made using computer programming, any proof or disproof of the Collatz Conjecture will need to be provided from a purely mathematical standpoint, and not a programming one. We hope this insight into the Collatz Conjecture will help the mathematical community refocus its efforts and one day come up with a solution for this problem. It may also indicate that any statement requiring proof by induction to be proved true cannot directly be solved by a computer due to requiring the verification of infinite values, but a deeper dive into that idea is a task for another time.

References

[1] R. Guy, *Unsolved problems in number theory*, volume 1 (Springer Science & Business Media, 2004).

[2] T. Honda, Y. Ito, K. Nakano, Gpu-accelerated exhaustive verification of the collatz conjecture, *International Journal of Networking and Computing*, *7*(1):(2017), 69–85.

[3] Veritasium, The simplest math problem no one can solve - collatz conjecture, https://www.youtube.com/watch?v=094y1Z2wpJg, 2022, accessed: 2023-01-30.

[4] J. A. Perez, Collatz conjecture: Is it false?, *arXiv preprint arXiv:1708.04615*.

[5] B. M. Gurbaxani, An engineering and statistical look at the collatz (3n+ 1) conjecture, *arXiv preprint arXiv:2103.15554*.

[6] C. Koch, E. Sultanow, S. Cox, Divisions by two in collatz sequences: A data science approach, *Int. J. Pure Math. Sci*, *21*.

[7] A. Turing, Halting problem, *London Mathematical Society*.