Everglades Analysis Design Document

Catrett, Brian Epes, Chandler Hyppolite, Shauna
Krupa, Sebastian O'Quinn, Christian

April 27, 2020

Contents

Chapter 1

Executive Summary

1.1 Goals and Objectives

The goal of the project is to evaluate data from the Everglades game to collect advanced analytics of player agents to characterize behaviors that end up leading to wins. To accomplish this, analytic methods and algorithms will be utilized to evaluate game statistics such as territory control, unit differential, objective control, first turn advantage, the effects of "Fog of War", and many other aspects that exist within the game. In addition to these metrics, we will propose new metrics and evaluate their influence in leading to wins.

To generate new, relevant metrics, our original analysis of normal game play will be scrutinized and used to create AI scripts to test hypotheses. These scripts will provide additional data sets to be analyzed. Analyzing the new data set along with our original data set will provide insight into our original hypotheses thus allowing us to form new conclusions. These iterations of hypothesis testing will identify and characterize behaviors that lead to wins.

Using the characterized behaviors, a supervised machine learning algorithm will be developed. This algorithm will utilize the successful characteristics determined by AI scripts to predict the match outcome given the current state of the game. In addition to the current game state, the algorithm will need to successfully determine potential future moves that will affect the prediction of the match outcome. The algorithm should take into account at least one move in the future to give a more accurate prediction. Some game characteristics that should be considered

here would be the "Fog of War" disruptions, objective control changes and the likelihood of a given team losing pivotal units.

Finally, a real-time analytic engine will be developed to run alongside the match playback. Like the machine learning algorithm, the engine will need to predict the outcome given the current game states but in real time. The engine will need to put a stronger emphasis on potential moves given a game state and be able to accurately analyze the future effects of the loss of key units, the loss of objectives, and many other crucial factors in determining wins as the game develops. Additionally, the engine will be used to describe agent behaviors.

Chapter 2

Technical Content

2.1 Project Overview

We, along with three other groups of UCF Senior Design students, have been charged with developing, analyzing, and improving upon AI agents participating in the Lockheed Martin strategy game, EVERGLADES. The game is a turn based strategy game, played on a board consisting of connected nodes or territories. Each of the two AI agents participating in the game control an array of units with varying stats and attempt to achieve victory by capturing the opponent's base or eliminating the opposing units.

Our team will head the analysis portion of the project, while two teams focus on AI agent development, and the final team works on improving the playability and variability of the game. The amount of information generated by the interaction of the AI agents during gameplay is immense, and our main objective is to try to parse through the proverbial haystack to determine what is relevant and what is not.

The game of baseball is over a hundred years old, but was revolutionized by the use of Sabermetrics, which take previously overlooked statistics and use them to more accurately determine a player's affect on run scoring or team wins. Ideally, we would like to create comparable metrics for the EVERGLADES game, something that goes beyond the simple comparison between number of territories controlled or units remaining. This is not to say that these pieces of information are irrelevant, but it may be possible to combine them or expand upon them in a way that

makes them more indicative of agent success, such as how on-base percentage is more telling than batting average.

While our main focus will be on data analytics, we will also work in conjunction with the AI agent development teams to generate improved AI training capability. We will be able to take the agents they have produced and use them to test our hypotheses regarding what metrics indicate success, and then provide the AI agent development teams with feedback so that they might better understand what behaviors they should be rewarding. We can also develop new hypotheses based on how these teams have designed their agents. They may believe there is a different strategy or prioritize objectives differently to achieve success. This interchange of ideas will provide us with a rich and diverse data archive on which to perform our analysis.

2.2 Personal Motivations

Brian Catrett

What piqued my interest in the Everglades Analytic project is the data analysis or more importantly the creation of a real time analytic engine. I've always been drawn to data analytics and even in my previous career my ideal job was in data analysis. Currently one of my biggest interests and where I would like to land long term is Data Engineering. While Data Engineering focuses more on applications and harvesting big data I felt the analysis aspect would give insight into the field. In addition to this I have always been a competitive individual so developing AI scripts to test the results of our data analysis is intriguing.

Chandler Epes

I was hooked on the Everglades Analytics project when I saw the strategy game it is centered around. I have always been passionate about strategy games and coming up with optimal play patterns to help myself succeed. At the same time I have been interested in the recent development of high level AI for games such as Starcraft and DOTA 2 that are able to beat the best professional players in the world. This project seems like the perfect intersection of two of my passions and will be an excellent learning opportunity.

Sebastian Krupa

My main motivation for doing this project is to gain more knowledge on the subjects at hand. When the project was being explained during class, I immediately started thinking of ways I could transfer the relevant material into other projects I may want to work on in the future. I really liked how the material was presented and how there are other teams who are working together to make a proper project in the end. I haven't used Python or learned Machine Learning yet, but I'm very interested in learning and want to succeed on this project for myself and my groupmates.

Shauna Hyppolite

I became very interested in the EVERGLADES: Robot Behavior Analytics because of my interest in Machine Learning and the math behind it. I have been wanting to gain more knowledge in this particular field and with wanting to possibly go into grad school and specialize in Machine Learning, I felt this project would give me a great start. I have been wanting to get my hands on a machine learning project so that I can really grasp the different algorithms and how things work theoretically and mathematically to produce hypothetical results. I started to do my own supplementary learning in Machine Learning and I am excited to work on this project with my team.

Christian Read O'Quinn

What initially drew me towards the EVERGLADES project, specifically the Robot Behavior Analysis portion of the project, was the opportunity to broaden my knowledge base. I haven't had the opportunity to handle data analytics for a platform with the intricacy of the EVERGLADES simulation, nor have I had a chance to work in artificial intelligence development. Both fields, while new to me, have long been an interest of mine and I look forward to the challenge ahead. I am also taking on the responsibilities of project manager, which I hope will give me insight into what managing a team of bright developers might entail in the workplace. I believe my communication and interpersonal skills would be underutilized if I were to wind up as a software developer who just sits at his desk and types away all day. I think my best fit in a professional environment would be as a software architect or project manager, someone who can effectively

lead a team and utilize each member's strengths. I hope this experience will give me useful insight into my potential future career. Lastly, I am a competitive individual and I like to win. I believe that our team successfully finding relevant metrics for achieving victory will allow us to develop a formidable AI agent which will dominate all others.

2.3 Broader Impacts

The potential impacts of this project are diverse and far reaching. The AI development tools and techniques produced during the course of this project, if successful, could be used to train AIs to perform in any well defined environment like the EVERGLADES platform. Artificial Intelligence is being applied to increasingly complex problems and as machine learning capabilities and computing capacity increase, there's no telling where the ceiling is for machine intelligence.

IBM developed Watson to compete with the brightest minds on Jeopardy! and Google has created an AI to play Go, arguably the world's most intricate board game. Youtube is even moderated by AI driven censorship bots that determine what content is acceptable. Artificial Intelligence is everywhere and being able to effectively teach and analyze them is an invaluable skill.

In general the development of AI is affecting society on a day to day basis. In a world where an individual can ask Alexa for an answer instead of manually typing the question into a search engine it is reasonable to expect public engagement to continue to trend up in regards to AI and how STEM fields are affecting their everyday lives. Products featuring AI systems have literally made it into the hands of the vast majority of our population and the developments of these products are fast increasing. Products like our phones have become an extension of us and these devices are reaching the hands of younger and younger individuals. With exposure to this technology reaching our younger generations the expectations of their interest and how these products function should increase thus increasing participation of STEM fields of your future generations.

Legal, Ethical and Privacy Issues

One obvious potential ethical issue is the use of AI in warfare. Being that Lockheed Martin is contracted by U.S. military agencies, the improvements to any Artificial

Intelligence made in the course of this project could be applied to non-human combatants, such as drones. While we won't have any direct involvement in loss of human life, our research could be used in a way that results in that outcome. We do trust that Lockheed Martin and the U.S. government highly value the lives of all people and do everything possible to prevent conflict across the globe.

Another potential ethical concern is the use of AI for censorship. The amount of content being created in the world is at a historic level and it's impossible for it all to be analyzed through human labor. Artificial Intelligence can be used to help determine what follows a platforms guidelines and what doesn't but the Artificial Intelligence can be manipulated to marginalize a certain group of content creators. To prove a platform isn't intentionally manipulating their censorship AI, they would need to make the learning and decision making algorithms public. Access to this information, however, would allow nefarious users to subvert censorship by altering their content to avoid the filters.

2.4 Project Requirements and Specifications

2.4.1 Goal #1

Characterize Everglades match output to determine behaviors that most of- ten lead to wins.

Specification

Characteristics should be quantifiable and the logic for the characteristic should be clearly recorded in the documentation. A valid characteristic should be statistically significant in determining win rate. Invalid characteristics should be preserved to both avoid doing duplicate work across the team but also to be retested as our data sets and methods of observations mature.

Requirements

We have to find at least five valid characteristics. We hope that we will be able to easily surpass this goal and find a wide range of valid characteristics so we will

be able to gather as much information from a given game state as possible. This will make it much easier for our machine learning algorithm in Goal #4 to make accurate predictions.

2.4.2 Goal #2

Create AI scripts of targeting behavior to test hypothesis generated from characterization activity.

Specification

AI scripts will be made for specific characteristics defined in Goal #1. At least two AI scripts should be made for each characteristic, one that is defined to maximize that behavior and one to minimize that behavior. This is going to be used to see both how effective the characteristic can be when focusing on it while also observing if ignoring it can be potentially beneficial. This will hopefully allow us to observe the opportunity cost of certain characteristics / behavior.

Requirements

We must create an AI script for the characteristic we create, this will be used to determine if the characteristic is valid. An AI script must have a statistically significant difference in the desired characteristic over a standard random selection AI. This will ensure that we are getting accurate results about the effect of maximizing / minimizing that characteristic.

2.4.3 Goal #3

Propose new metrics to more accurately predict agent performance, similar to sabermetrics.

Specification

Metrics should be quantifiable and the logic for the metric should be clearly recorded in the documentation. These metrics should be based on historical data of AI agents, rather than their in game behaviors that defined the characteristics from Goal #1. A successful metric should be statistically significant in determining match outcome or accurate at estimating other match criteria such as score or node control.

Requirements

We have to find at least five valid metrics. We hope to find many more than that requirement though as having a larger range of metrics will allow for more angles of predicting future events. This will make it easier for us to predict outcomes and will prove invaluable for determining starting likelihood of win in goal #5.

2.4.4 Goal #4

Implement a supervised machine learning algorithm that predicts the outcome of a match given the current game state.

Specification

This algorithm should be designed to use successful characteristics defined in Goal #1 to identify important factors in the game state and use those to model and predict future game results.

Requirements

This algorithm should have an accuracy of at least 75% for game states after turn 100. Earlier game predictions are more lenient do to the ability for agents to stage comebacks, therefore for game states after turn 50 the algorithm must have an accuracy of at least 60%. Game States prior to these turn counts are likely

too volatile to accurately predict but we look forward to trying to push for high accuracy at all stages of the game.

2.4.5 Goal #5

Develop a real-time analysis engine that runs simultaneously with Everglades match playback. The engine should at minimum predict odds of winning and describe the agent behaviors.

Specification

This engine should use a combination of Goal #3 and Goal #4. It should determine the starting odds based on the strengths of each AI according to metrics defined in Goal #3 and then adjust those odds as the games progresses using an adjusted version of the algorithm from Goal #4 that account for both characteristics from Goal #1 as well as metrics about the AIs historical performances from Goal #5.

Requirements

This engine must be reasonably accurate (i.e. it should not predict that the loser has a 95% chance of winning the turn before they lose outside of extreme circumstances). It must determine a starting odds from historical data and then be able to adjust the likelihood of each agent winning as the game progresses based on their behaviors in the match.

2.5 Possible Features

Due to the scope of the data we are reviewing and the amount of characteristics and methods we intend to make it is imperative that we document our work consistently and thoroughly to make it easy for us to track our progress and review existing findings with new data. To this end we believe that using Jupyter Notebook would be ideal for this project. Jupyter notebook is a web-based development environment that allows for data and code to be displayed in a readable

and documented way. This is ideal for our purposes as it allows for us to make interactive documentation for our various characteristics, methods, and algorithms that actually display the results of those findings and how they work. It also allows for easy collaborative work on the documentation between the work by using version control and a hosted master copy.

One of the core challenges of this project is developing characteristics and metrics to measure, to do this we plan on using a mixture of our own intuition to find powerful play patterns as well as using machine learning to find trends within the data that point to effective AI behavior. Listed below are some ideas we have begun developing and a brief explanation of them.

- **Territory Control** - The percent of available territory an agent controls during the current turn. We predict that having control of more territory on the board means that you will have better control of the overall flow of the game and therefore a higher win rate.
- **Unit Differential** - The amount of units one agent has remaining minus the amount of units his opponent has remaining. We predict that having more units remaining that your opponent will result in more ability to control the game and therefore a higher win rate.
- **Fortress Control** - The percentage of time the agent has controlled the Fortresses nodes over the course of the game. We predict that having control of a fortress will allow for agents to be more flexible with their unit placements due to the defensive benefits for the units placed inside of the fortress allowing them to fend off stronger units than they normally would be able to. This should result in a higher win rate.
- **Watchtower Control** - The percentage of time the agent has controlled the Watchtower nodes over the course of the game. We believe the extra vision provided by the watchtower will be incredibly powerful as it allows the agent who controls it to avoid traps and dangerous movement into the fog of war. This should result in a higher win rate.
- **First Turn Advantage** - This is a simple comparison of win rate comparing when an agent goes first against agents going second. By determining if there is an advantage to going first or second, it may be possible for agents to determine how to adjust strategy based on turn order.
- **Unit Above Replacement** - Influenced by Sabermetrics statistic WAR (Wins above replacement) this evaluates specific unit contribution compared

to it's loadout/team. At a minimum evaluate what is the most useful unit.

- **First Unit Lost** - This is a simple comparison to see how often the first person to lose a unit loses the game. This would cause agents to play more defensively or stress the use of fortress nodes.
- **Aggressive vs Defensive Strategy** - Is there an advantage to be gained by pushing more aggressively towards the enemy position instead of defending your own? Is it wiser to fortify your position and let the enemy throw his forces at you until the turn limit is reached?
- **Movement Utilization** - Each agent can move seven of their twelve swarms per turn. The game does not recognize invalid moves and discards them, meaning inaccuracies in the AI decision making could result in missed opportunities. Alternatively, moving none of the swarms could potentially lead to the best outcome. Will complete usage of movement each turn reflect in how likely an agent is to succeed?
- **Territory Retention** - While an agent may succeed in capturing a territory, that action may expose the agent, causing the territory to be lost shortly thereafter. Could the rate at which territory is lost after being taken indicate an untenable strategy?
- **Unit Proximity** - Each unit type has different strengths and weaknesses. Utilizing combinations of units to cover for each others weaknesses and moving them together could provide a noticeable advantage. Different combinations of units may work better than others.

A challenge that must be faced early on with this project is determining how to best analyze and present the data. Most of our members lack expertise in the fields of data analysis and machine learning. We have been directed towards scikit-learn by our sponsor which is a Python package that is designed to aid in performing machine learning tasks as well as the OpenAI Gym for machine learning development. Scikit Learn will allow for us to try many different classification algorithms to find the ones that work best for our different datasets and measurements. For instance we could easily set up a script that runs nine different classifiers on a dataset and see which provides the best result. These results then need to be formatted in a way that is easily decipherable as well as appealing to the viewer.

Another interesting possibility is that the metrics that are most impactful in terms of determining if an agent will win or lose may change as the agents learn and become more proficient at the game. We may need to revisit previously discarded

hypotheses and retest them with more advanced agents or discard hypotheses that look promising but are becoming less fruitful. We could even see the emergence of a "metagame" where certain strategies become more or less effective based on how the agents are adjusting to each other. One agent might struggle to defeat another agent that most defeat easily but be able to dispatch all other opponents. There's really no telling what's going to occur.

2.6 Distribution of Work

Our project has been divided into three main sections so that work can be easily delegated amongst members of our team. The three sections are Database and Architecture, Data Analytics and Modeling, and Reinforcement and Supervised Learning development.

While each team member is assigned to one of these three sections, it is important that each team member be familiar with and have a solid understanding of the other portions of the project. For this reason, each team member assists with the other portions of the project that they are not assigned to. Each team member has also established a testing environment on their personal hardware, so that they can explore metrics independently.

- Database & Architecture
  - Chandler Epes
- Data Analytics & Modeling
  - Brian Catrett
  - Shauna Hyppolite
- Reinforcement & Supervised Learning
  - Sebastian Krupa
  - Christian Read O'Quinn

Brian Catrett

Brian Catrett is focusing on the Data Analytics & Modeling portion of the project, in conjunction with Shauna Hyppolite. He has familiarized himself with the rec-

ommended software, SciKit Learn and its affiliated tools, and has begun exploring how the data generated from the EVERGLADES matches can be most effectively modeled and visualized.

Brian also spearheaded the effort to establish testing environments on each team member's personal hardware. He was tasked with troubleshooting the initial version of the software that was provided by Lockheed Martin and produced a manual for setting up the testing environment for other team members to use.

Chandler Epes

Chandler Epes is primarily focused on the Database & Architecture side of the project. He is responsible for the collecting data, managing and designing the database, and creating datasets to be used in models and AI Agents. Chandler will be testing different database implementations and ways of manipulating the data in order to ensure a smooth workflow.

Alongside the database Chandler is also responsible for managing the server infrastructure for the project. He must make sure that that database is accessible for all users and easy to connect to when needed. Chandler is also in charge of managing data sets and general storage of our findings and documentation.

Shauna Hyppolite

Shauna Hyppolite is working on the Data Analytics & Modeling side of the project, partnering with Brian Catrett. She has become proficient in using SciKit Learn as well as Pandas and several other pieces of Data Analytics software so that she can most accurately represent the data generated during EVERGLADES matches.

Besides the Data Analytics and Modeling portion of the project, Shauna also participated in the establishment of the database, providing input on the architecture and implementation.

Lastly, Shauna is the point of contact between our team and our sponsors at Lockheed Martin. She handles communications between our team and their members and resolves any questions we may have about project requirements or the software provided.

Sebastian Krupa

Sebastian Krupa's main responsibilities were to research and begin focusing on development of Reinforcement Learning AI agents in collaboration with Christian Read O'Quinn. He was tasked with getting proficient with the different Reinforcement Learning Algorithms and to make the choice of which algorithm to be used when developing the agents.

Sebastian maintained contact with the other EVERGLADES groups to gain insight on their projects and to determine what algorithms they were going to work on for developing their own agents. Collaboration is a key focus of Sebastian's as, working together will help fill in for the flaws we each have.

Christian Read O'Quinn

Christian Read O'Quinn, who goes by Read, is primarily focused on the Reinforcement and Supervised Learning portion of the project. He has been tasked with cultivating an understanding of the AI development process so that it can be used in the adjustment of the EVERGLADES Agents as well as the production of the Real Time Analytic Engine for predicting win rate.

While the Reinforcement and Supervised Learning portion is his main focus, he assisted Chandler Epes in determining the structure of the database as well as how it should be implemented.

Lastly, Read is in charge Project Management. He organizes meetings and records minutes in addition to maintaining the Project Management tools being used, such as Trello and Discord.

2.7 Project Block Diagram

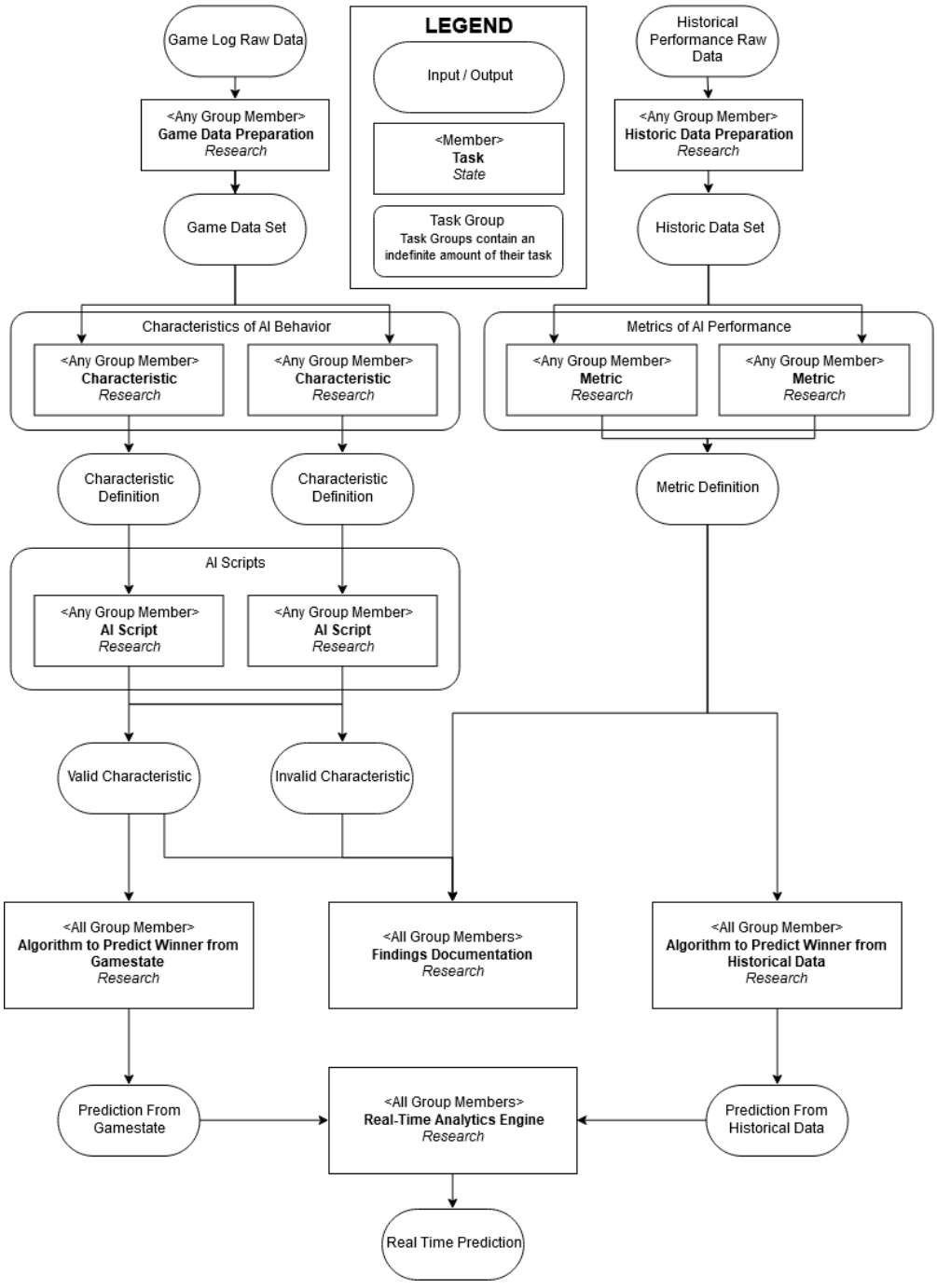


Figure 2.1: Project Block Diagram

Chapter 3

Research and Investigations

3.1 Game Mechanics

The manner in which gameplay is executed as outlined in the following document is accurate to the current iteration of the game as of December 2019. There exist plans to expand the variability and robustness of the EVERGLADES platform, which will be touched on as it pertains to each section of the following rules and mechanics section. These future plans will also be referenced in the in-depth explanation of the telemetry output files in section 3.3.

See the AI Developer's Guide, Telemetry Overview, and Everglades Project Update (July 2019) in the References section for further detail.

3.1.1 Overview and Objective

EVERGLADES is a synchronous turn-based strategy game played by two AI agents. The agents interact with the game environment by issuing movement commands once per turn. The movement instructions are then carried out by the gameplay environment which handles all other game mechanics including combat, capturing of territory, and accruing points.

Victory can be achieved in three ways: have the most points at the end of the 150 turn game, eliminate all units belonging to the opponent, or capture their Base node. Points are gained by destroying enemy units and holding territory on the

map.

3.1.2 Game Board

The game board is comprised of a series of connected nodes with each Agent's units starting on their Base node, located on their respective end of the playing field. The current game board, shown below, consists of 11 nodes, two base nodes and a 3x3 array of neutral nodes connecting the base nodes.

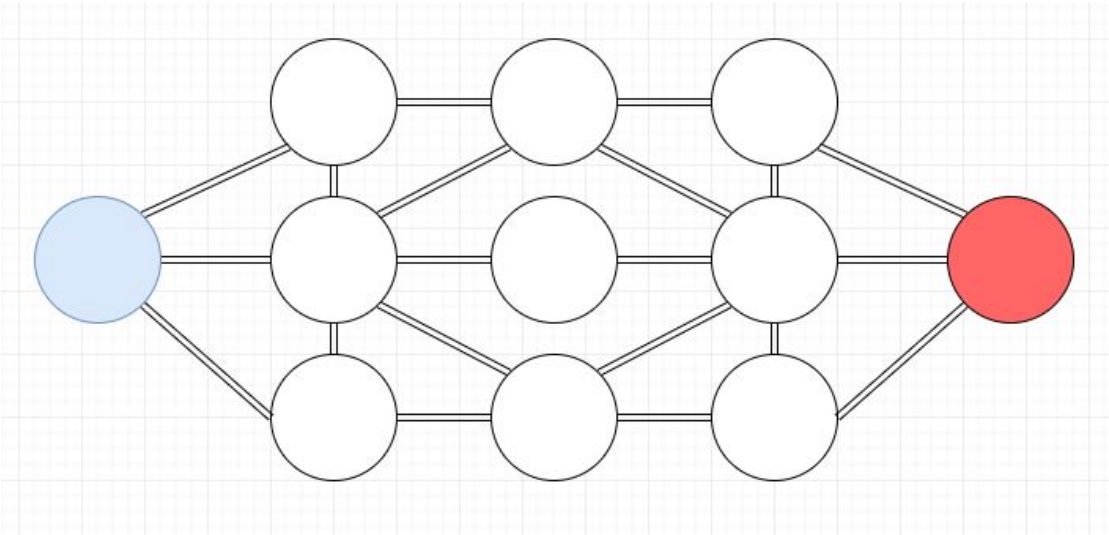


Figure 3.1: The Game Board.

There are also special node types called Fortresses and Watchtowers, each providing unique bonuses to the Agent that controls them. Fortresses provide additional defense bonuses for any allied unit in the area during combat, while Watchtowers provide vision of the territories adjacent to it.

In future, the game board will be variable, with different layouts of nodes and connections. The board may also be imbalanced, favoring one starting side over the other. As of now, all agents are being trained on the board shown above and all data collected is from matches played on said board.

3.1.3 Unit Types

Each agent is given a total of 100 units of three types: Strikers, Controllers and Tanks. These units are organized into 12 movable groups that the Agent can control.

Group	Loadout
Group A	8 Strikers
Group B	8 Controllers
Group C	8 Tanks
Group D	8 Strikers
Group E	8 Controllers
Group F	8 Tanks

Group	Loadout
Group G	8 Strikers
Group H	8 Controllers
Group I	8 Tanks
Group J	8 Strikers
Group K	8 Controllers
Group L	12 Tanks

Figure 3.2: Unit Group Distribution.

Units have 5 attributes that determine how able they are to perform certain tasks. These attributes are Health, Attack, Speed, Cost, and Capture Speed.

- Health determines how much damage a unit can sustain before it is destroyed
- Attack represents how much damage a unit deals during combat
- Speed represents how quickly a unit can move between nodes
- Cost is not used in this version of the game
- Capture Speed shows how quickly a unit will gain control of a node

Strikers, Controllers, and Tanks have the following attributes.

	Controller	Tank	Striker
Health	100	150	50
Attack Power	10	10	20
Speed	1	1	2
Cost	1	1	1
Capture Rate	5/turn	1/turn	1/turn

In future iterations of the game, an Agent will be able to "purchase" its own combination of units within a budget constraint and organize them into groups to best fit its strategy. There are also plans to add additional unit types with varying attribute levels and costs.

3.1.4 Movement

As previously stated, the only interaction the Agents playing the game have with the environment is issuing movement instructions once per turn. Each group of units starts the game in the "Ready to Move" state. When the environment receives the movement instructions, they are validated to make sure the group is in the "Ready to Move" state and is in a node adjacent to the node it's moving to. Once instructions have been validated the group begins moving to the new node. This group is now in the "Moving" state.

Moving between each node has a specific movement cost attributed to it. In the current version, this cost is uniform between all nodes. The Speed attribute determines how quickly a group of units can move from one node to the next. For example, if the movement cost from one node to the next is 10, and the group has a speed of 2 (Strikers), it will take 5 turns for the unit to move to the new node.

When a group is in transit, it cannot interact with the game environment by participating in combat or capturing territories. Groups can also move along the same connection as an opposing group at the same time. Connections operate more like two separate one-way routes, shown in green in Figure 3.3, as opposed to one single lane connection, shown in red.

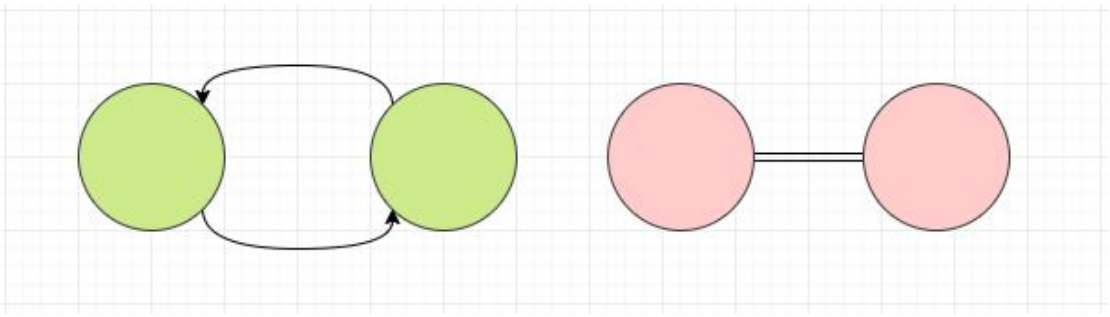


Figure 3.3: Movement Visualized.

Once a group has fully "paid" the movement cost associated with transitioning to the new node, it will be returned to the "Ready to Move" state and can receive further instruction from the Agent.

In future versions of the game, the movement cost between nodes will be varied and could even change between turns to simulate wind or environmental changes. The connections between nodes could also run only in one direction, preventing movement backwards.

3.1.5 Capturing Territory

Units capture nodes by remaining in that node for the turn. Currently, the standard node starts at a capture value of 0 and can increase up to 100 when fully captured by one side. Base nodes start at 200 capture points and have a maximum of 500 when fully captured.

Capturing nodes both gains points towards victory as well as provides a damage reduction bonus for any allied unit fighting in that captured node. The higher the capture amount of the node, the greater the bonus it will provide. The damage reduction ranges from 0% to 50% depending on the type of node and its capturing level.

The special nodes, Watchtowers and Fortresses also must be completely captured to provide their bonuses to units occupying them.

Nodes that are contested, containing units from both sides, cannot be captured. They will remain at the capture level they were at initially before becoming contested until only units from one side remain present.

3.1.6 Vision

Agents knowledge is limited by "fog of war", meaning they only have knowledge of certain parts of the playing field at certain times. Agents have knowledge of nodes that they control as well as any node occupied by a group under their control. Watchtowers also provide vision of adjacent nodes when they are controlled. This vision makes the agent aware of any opposing groups present as well as the capture value of the nodes.

In future, Agents will be able to have partial knowledge of a node as outlined in the Vision CSV file. This partial knowledge will allow them access to only part of the information available with full vision.

3.1.7 Combat

Combat only occurs when groups from opposing forces occupy the same node during a turn. The actual combat operates in a "revolutionary war" fashion, with all units lining up across from each other. Each unit deals damage based on their attack power, which can be reduced by defensive bonuses from node control or Fortresses.

Each unit in a group targets a unit in the enemy group at random and deals damage to it. If there are multiple groups available to target, one is chosen at random. This randomness to targeting provides a certain level of variance in combat. Damage cannot be healed and destroyed units cannot be replaced.

Units still deal full damage regardless of their remaining health and units can be dealt more than their remaining health in damage during combat. This potential to "overkill" a unit provides for interesting scenarios to analyze for generation of sabremetrics.

Let's take, for example, combat between two full health groups. One group of 8 Tanks will be fighting against one group of 8 Strikers on a node fully controlled by the Striker side. Tanks, with health of 150 and Attack Power of 10, would each do only 5 damage to whichever unit they target, due to the node control damage reduction bonus. Each Striker, with health of 50 and Attack Power of 20, would do full damage to the Tank units. This process is visualized in Figure 3.4.

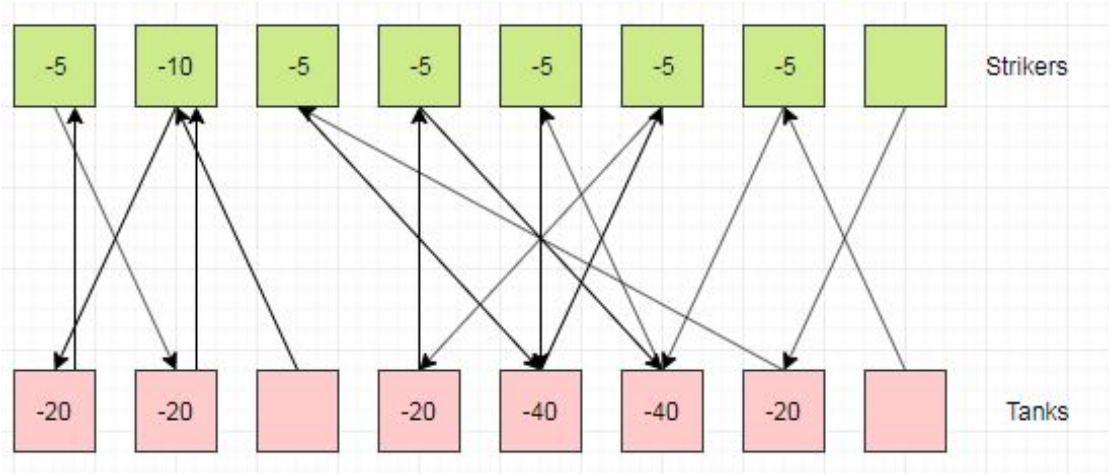


Figure 3.4: Combat Visualized.

3.2 Supervised Learning

The development of agents to play the EVERGLADES game will be handled strictly using Reinforcement Learning, but the training of the algorithm to estimate win rate in real time will be accomplished using Supervised Learning. These two processes differ greatly, most importantly in the types of problems they can be used to solve. As explained in Section 3.2, Reinforcement Learning allows the machine to explore problems that do not have a comprehensive data set to refer to. Reinforcement Learning also is not limited to making decisions based on actions taken and recorded previously. It can be used to go beyond what the human mind believes is optimal.

Supervised Learning, which will be explained more in depth in Section 3.6, lends itself better to problems that are attached to a comprehensive set of data, in which solutions can be found through regression or classification. Instead of a trial and error approach attempting to maximize reward, as in Reinforcement Learning, Supervised Learning improves by making decisions and then learning from them by being shown the correct answer afterwards.

For these reasons, the development of the Real Time Analytic Engine to predict the win rate of agents is better suited to a Supervised Learning solution.

3.3 Raw Data

Data is the core of our project. We can not analyze and come to conclusions if we do not have correct and easily accessible data. Therefore it is of the utmost importance that we create a database that is easy to work with and can accommodate all of our use cases. There are three sources of data that are produced by the EVERGLADES game system:

- Agent Feedback
- Match Telemetry
- Map Information

3.3.1 Agent Feedback

Agent Feedback is the feedback given to the Agents about the game state during the process of playing a match, its what the Agents use to make a decision on what to do next. It is passed to the Agents as an array with the following information:

```
Index 0: Turn Number
Indexes [1-4][5-8][...][41-44]: Node Information
   Node {1-11}:
      is fortress [0, 1]
      is watchtower [0, 1]
      percent controlled [-100-100]
      number of opponent units [0-100]
Indexes [45-49][50-54][...][100-104]: Group Information
   Group {1-12}:
      location as node number [1-11]
      class number [0-2](0: Controller, 1:Striker, 2:Tank)
      average group health [0-100]
      is in transit [0,1]
      number of units alive [0-100]
```

While Agent Feedback is important to understand for reasoning about how AI Agents make decisions for analysis purposes and for developing our own testing agents, we do not plan on recording any data of this type from the EVERGLADES system as it is a subset of information found in the Match Telemetry.

3.3.2 Match Telemetry

The Match Telemetry provide by the EVERGLADES server provides a variety of information stored as .csv files. The files all split into different folders based on their event types. The full list of events generated over the course of a match is as follows:

Telem_GAME_Scores occurs at the start of every turn.

Column Name	Description
0	Turn number
player1	Player 1 Score on this turn
player2	Player 2 Score on this turn
status	Current win status
focus	Camera Focus for playback

As the Camera Focus is only useful for the playback system we do not intend to track it in our Data Sets.

The potential win statuses include:

- 0: In Progress
- 1: Time Up
- 2: Base Capture
- 3: No More Units on either side (Mutually Assured Destruction)

Telem_GROUP_CombatUpdate occurs whenever combat occurs.

Column Name	Description
0	Turn number
player	Player who owns the units
node	Node combat is taking place on
groups	Array of GroupIDs that the units belong to
units	Array of UnitIDs that are participating in combat
health	Remaining health of the unit

Telem_GROUP_CreateNew occurs when a group splits into two. As this event does not occur in our version of the EVERGLADES system we do not need to track it or understand its structure.

Telem_GROUP_Disband occurs when a group is removed from play.

Column Name	Description
0	Turn number
player	ID of Player who owns the group
group	ID of the group that is being disbanded

Telem_GROUP_Initilization occurs at the start of the match to create each teams units.

Column Name	Description
0	Turn number
player	ID of Player who owns the group
group	ID of the group that is being created
node	ID of the node the group is spawning on
types	Type of the units in the group
start	Starting ID for the units in the group
count	The amount of units in the group

Telem_GROUP_Knowledge occurs each turn and shows the knowledge each player has of the groups.

Column Name	Description
0	Turn number
player	ID of the player whose knowledge is being described
unitTypes	Array of types of the units the player has knowledge of
unitCount	Array of amount of units in each group the player has knowledge of
status	Whether the units are stationary or in transit
node1	ID of the node the units are currently on or moving from
node2	ID of the node the units are moving to if they are moving

Telem_GROUP_MoveUpdate occurs whenever a group moves from one node to the next.

Column Name	Description
0	Turn number
player	ID of the player who owns the group
group	ID of the group
start	ID of the node the group is starting at
destination	ID of the node the group is moving to
status	Status of the groups move

A group will have one of these statuses:

- RDY_TO_MOVE - Move order has been issued by the Agent
- IN_TRANSIT - Group is moving between nodes
- ARRIVED - Group has arrived at its destination

Telem_GROUP_TransferUnits occurs whenever units move from one group to another. As this event does not occur in our version of the EVERGLADES system we do not need to track it or understand its structure.

Telem_NODE_ControlUpdate occurs whenever there is an update to the control value of a node.

Column Name	Description
0	Turn number
node	ID of the node
faction	ID of the player who is controlling the node
controlvalue	Amount of control the player has on the node
controlled	Whether the controlvalue is above the capture threshold

Telem_NODE_Knowledge occurs each turn and shows the knowledge each player has on each Node.

Column Name	Description
0	Turn number
player	ID of the player whose knowledge is being represented
nodes	Array of node IDs
knowledge	Whether the player has full, partial, or no knowledge of the node
percent	How close the node is to being fully controlled

Telem_PLAYER_Tags occurs at the end of the match and gives us information about who played in the game and how long it took.

Column Name	Description
0	Amount of turns the game took
player1	Script name used by player 1
player2	Script name used by player 2

3.3.3 Map Information

At the moment all matches in EVERGLADES are played on the same map. There does however exist infrastructure to use other maps and we believe that it is best to future proof our work by accounting for this capability in our database. The map is provided in JSON format and an example is seen below:

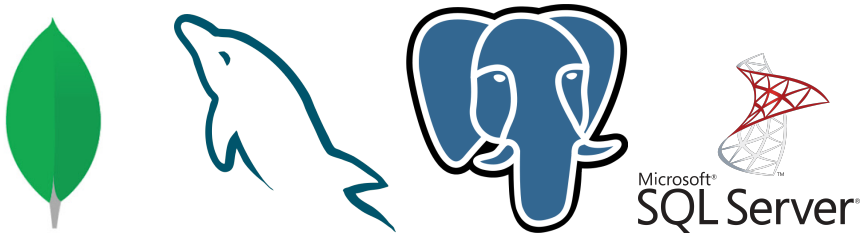
```
{
   "__type":"Map:#Everglades_MapJsonDef",
   "MapName":"Default",
   "nodes":[
      {
         "Connections":[
            {
               "ConnectedID":1,
               "Distance":5,
            },
            {
               "ConnectedID":2,
```

```
            "Distance":5,
         },
         {
            "ConnectedID":3,
            "Distance":5,
         }
      ],
      "ID":0,
      "Radius":10,
      "Resource":0,
      "StructureDefense":1,
      "NodeDefense":5,
      "TeamStart":0
   }
]
}
```

3.4 Data Preparation

3.4.1 Database Technologies

A large part of Data preparation involves how our data is stored. To research this we looked at both NoSQL databases and SQL databases and compared several different versions of SQL databases.



(a) mongoDB. (b) MySQL. (c) PostgreSQL. (d) SQL Server.

Figure 3.5: Database Technologies.

NoSQL Vs. SQL

During our initial survey of database technologies we did consider the option of a NoSQL Ddatabase such as mongoDB because we had heard of their uses in many modern applications. However we felt that a more traditional SQL database would be more effective for our project as our data is very structured and rigid in its nature. Therefore it is well suited to the tradition relational SQL style database.

SQL Databases

We look at the following implementations for SQL databases:

- MySQL
- PostgreSQL
- SQL Server

While all of these options would be usable for our project, there are enough differences between them where one would be the best option. SQL Server is not one we want to use due to its closed source nature that would drive our costs up and make the project more difficult to replicate compared to an open source solution. However we are still looking into whether we would prefer to use MySQL or PostgreSQL. We are planning on using MySQL primarily but also testing a parallel database using PostgreSQL for comparison purposes. One we get an idea of which is faster, especially once we begin to get a larger data set as more data rolls in we will then move fully to whichever is fastest.

3.5 Potential Models

To be able to predict game turn rewards and ultimately game outcomes, a combination of analytic tools should be considered. The telemetry output data can be best utilized by using supervised machine learning models. Some of the popular techniques, separated by category, are as follows:

- Regression:

  - Linear Regression
  - Polynomial Regression
  - Ridge Regression
  - Lasso Regression
- Classification:
  - K Nearest Numbers (KNN)
  - Logistic Regression
  - Neural Networks
  - Random Forest

This section will go into detail of these models and discuss their primary strengths, their short falls and differences that other models offer when trying to build classification and regression models.

3.5.1 Regression

Regression is a method of using data records to quantify relationships between the dependent variables and independent variables. This method allows the identification of which variables have an impact on certain areas of interest or the dependent variable. Our dependent variable being our hypothesis and our independent variables are the factors that are believed to be affecting it.

Linear Regression

Linear Regression is considered one of the simplest approaches for statistical analysis and data science. In fact it is considered to be the basis for more advanced models. The other Regressions techniques that will be discussed further below can be considered more advanced extensions of simple Linear Regression so it is important to understand it.

Linear Regression helps us determine the following:

- Is there a relationship between two variables?

- Is the relationship between these two variables linear?
- How strong is the relationship between the two variables?
- Which variable is the most significant?
- How accurate is our model when determining the effect of each variable?
- How accurately is our prediction of the target?

Below is the equation for a Linear Regression model with one feature or independent variable:

$$Y = \beta_0 + \beta_1 X \tag{3.1}$$

To make predictions with our model we must find the coefficients. To find these we will use the mean squared errors technique (or least squared method). This error is calculated with the following equation:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^{'})^2 \tag{3.2}$$

Squaring the error eliminates potential negative numbers which would decrease the sum of errors and would not produce a model that is a good fit. In addition to this, it will penalize large errors which also help in creating a better fit model.

After we have found the coefficients we must find our p-value which is used to accept or reject our hypothesis. The p-value will show if a variable is significantly affecting our hypothesis. In general a p-value that signifies a strong relationship will be below 0.05.

After, we determine if our model is a good fit by using residual standard error. Lower errors signify a model that is a better fit. The residual is the error that is not explained by the regression equation.

$$e_i = y_i - \hat{y}_i \tag{3.3}$$

In many circumstances multiple features will need to be taken into consideration when making a prediction. To do this we use multiple linear regression. In the

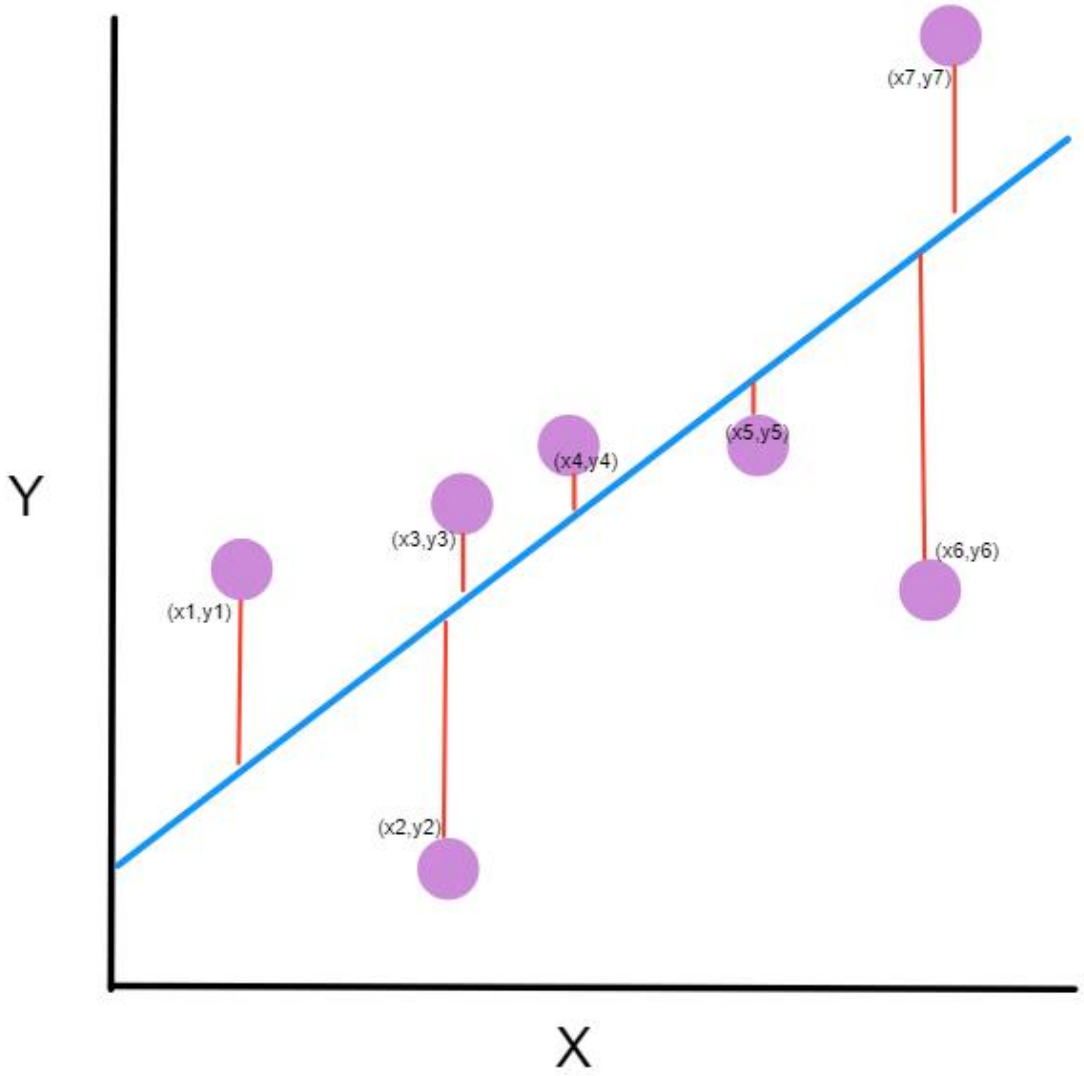


Figure 3.6: Graphical representation of using squared errors to map a regression line. *Source: freecodecamp.org*

equation below p is the number of features or independent variables. Using this method we will be using the F-Statistic instead of the p-factor. The F-Statistic represents the overall model instead of a single predictor like the p-value.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p \tag{3.4}$$

Polynomial Regression

Polynomial regression is best used in a scenario where linear regression under-fits the data or fails to capture the pattern in the data. Instead of using a linear model

like figure 3.x.1 we manipulate it into a model as follows:

$$Y = \theta_0 + \theta_1 x + \theta_2 x^2 \tag{3.5}$$

This allows the model to weight the coefficients. Using this equation will give a better fit to the data thus reducing our Root Mean Square Error. Using a Polynomial model allows the degree to be manipulated to find a curve that better fits the data set. Below is a comparison of a plot of data modeled with a linear regression line and two Polynomial lines of degree 2 and 3. What we see is as the degree in the model is increased, it fits the data better.

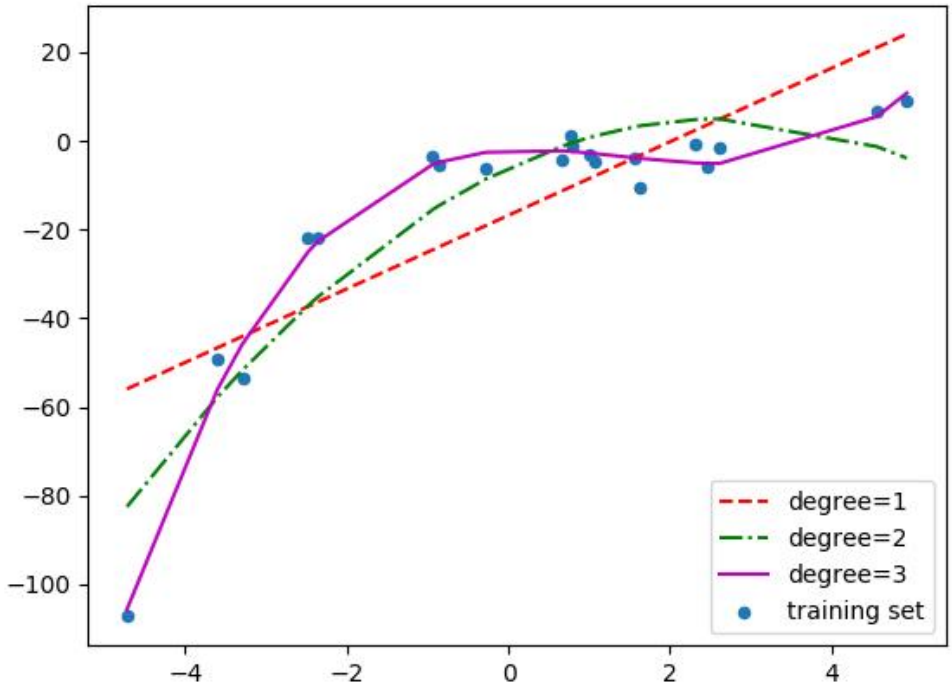


Figure 3.7: Compares a linear regression model compared to a polynomial model of degree 2 and degree 3. *Source: towardsdatascience.com*

One issue with increasing the degree is that eventually the model will be over-fitted if the degree is set too high. These models will be capturing the noise of the data and it is important to keep this in mind when it is being developed. In the scenario above the correct fit is the degree 3 model. To accomplish this we look for a model that has low variance and low bias. Figure 3.8 shows the difference between varying degree of variance and bias. The linear model has high bias and low variance, the degree 4 model has low bias and low variance and the degree 15

is over fitted which leads to low bias but high variance.

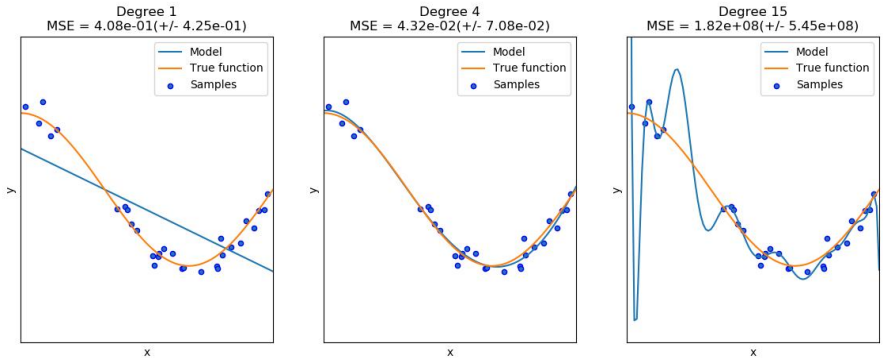


Figure 3.8: A comparison of varying degree of polynomial regression. *Source: Stackoverflow.com*

While Polynomial Regression might seem strictly better there are some disadvantages, such as potential outliers. These outliers are sensitive and the presence of one or two of these can negatively affect the analysis. In addition to this the tools of validation are much scarcer when compared to linear regression.

Ridge Regression

Ridge regression is best described as a variation of Linear Regression with the one major difference being it takes in consideration which variables are more important than others. Linear Regression uses Least Squares Method which does not weigh coefficients. This in turn produces a model with the Lowest Residual Sum of Squares, but having the lowest Residual Sum of squares may not be the most effective metric. Because of this Ridge Regression is a great consideration.

These weighted variables allow the use of a technique called L2 Regularization which introduces a penalty (the lambda term) to manipulate the model coefficient in a manner that produces better predictions. This cost function is as follows:

$$min(||Y - X\theta||_2^2 + \lambda||\theta||_1) \tag{3.6}$$

The desired outcome of testing various penalties on the model is to find one that reduces the Mean Square Error as much as possible. That is, the lower the penalty reduces the Mean Square Error the better fit the model is and as the penalty approaches zero the closer the Ridge Regression model approaches

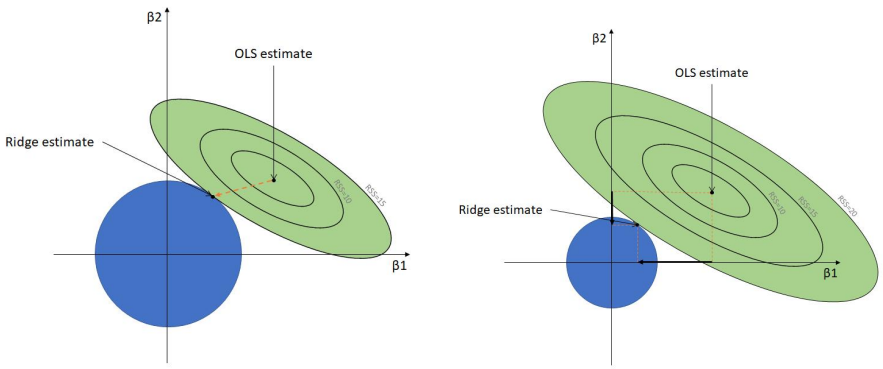


Figure 3.9: Graphical representation of the effects of the penalty adjustments on the RSS.

a Linear Regression model. It is generally accepted in data science that Ridge Regression out performs Linear Regression.

Lasso Regression

Lasso Regression is a model that uses shrinkage which is a technique where data is minimized to a central point, like the mean. This produces simpler models that take fewer parameters and is useful to help automate certain aspect of model selection like the elimination of parameters or variable selection.

To create these simpler models Lasso Regression also uses a Regularization technique similar to Ridge Regression but instead, Lasso Regression uses L1 Regularization. The differences between these two methods results in Lasso Regression models to be easier to interpret. L1 Regularization, like L2 Regularization, uses a penalty on the coefficients. The primary difference is L1 Regularization creates sparse models with less coefficients because some of the coefficients are reduced to zero, eliminating them entirely.

The cost function is similar to Ridge Regression and is as follows:

$$min(||Y - X\theta||_2^2 + \lambda||\theta||_1) \tag{3.7}$$

Lasso Regression is best used when there are a large number of independent variables. It allows these variables to be adjusted to zero which can reduce model complexity but this can also have adverse effects to the model if the variables are highly correlated which would over simplify the model. Since Lasso Regression

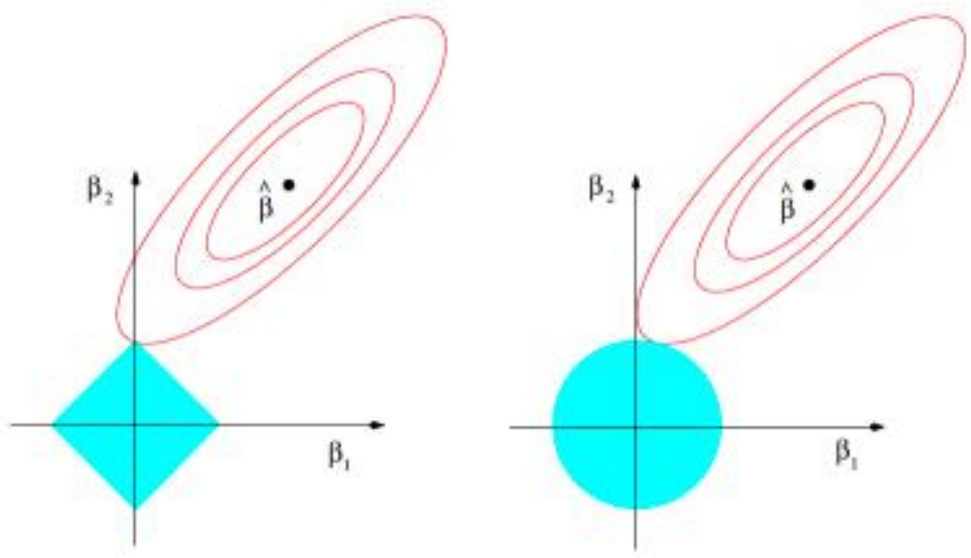


Figure 3.10: Lasso Regression compared to Ridge Regression. *Source: Medium.com*

shines over Rigid Regression when there are a lot of variables it is unlikely it will be of use when analyzing the output data.

3.5.2 Classification

Classification is used to determine a discrete output, for instance if something is true or untrue. Or classify a group of things, like flowers, based on their attributes. Or in the case of EVERGLADES Analytics did a player win or lose.

K-Nearest Neighbors(KNN)

KNN is widely used for classification in the Data Science industry. This is due to the fact that it is fast, easy to interpret and has high predictive power. KNN classifies data with the assumption similar things are near each other. It is used to classify a data point based on the data point's neighbors. That being said K is one of the most, if not the most, important parameter when using KNN.

This distance K is commonly determined by the Euclidean distance method although there are other methods such as Manhattan, Hamming and the Minkowski distance methods.

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \tag{3.8}$$

Some things to consider, as K approaches 1 it becomes less and less accurate. This creates high bias and gives an inaccurate estimate because the model will be too close to the training data. This also creates high variance which, as mentioned previously, captures a lot of noise. If K is increased to a high value we have a yet another negative affect. Larger values of K will have lower variance but produce higher bias.

Figure 3.11 shows how using $k = 3$ would classify our new example as green whereas $k = 7$ would classify it as red, emphasizing why the correct K value is important.

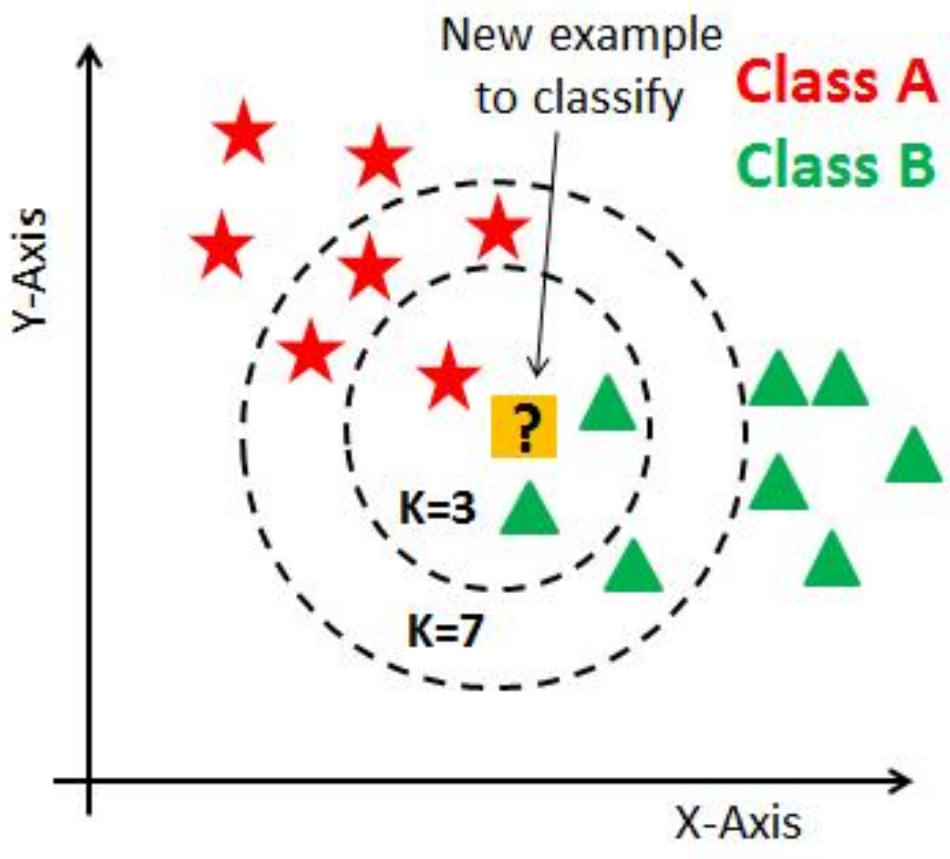


Figure 3.11: Shows the difference in K = 3 and K =7. *Source: datacamp.com*

Since choosing K is a major factor in a KNN model making the proper choice can make or break a model. One of the suggested ways of choosing K is cross-validation. Cross validation is taking a small subset of data from the training data

and using it to validate the model by selecting different values of K. Whichever value of K that produces the most accurate prediction of our training set should be used for the model. It is also common to use k as the square root of N where N is the number of samples.

There are a few issues with KNN model. First, the computation cost is high. This is largely due to calculating the distance between all of the training samples. The second is, as mentioned previously, finding a value of K that is not accurate will result in a less than ideal model.

Despite these issues there are plenty of pros for using this model and will play a role when analyzing the telemetry data output.

Logistic Regression

Logistic regression like KNN is used when the dependent variable is categorical. Unlike linear regression where continuous values are unbounded, Logistic regression uses strict ranges from 0 to 1. This is done by using the logistic function to squeeze the output of a linear function between 0 and 1. This results in the equation below.

$$P_i = E(y = 1|x_i) = \frac{e^z}{1 + e^z} = \frac{e^{\alpha+\beta_i x_i}}{1 + e^{\alpha+\beta_i x_i}} \tag{3.9}$$

This will produce a Sigmoid function similar to the one in Figure 3.12.

One way logistic regression differs from other classification models is it provides a prediction probability score of an outcome or event. So other classification models, like KNN will use the training data to give a prediction of a given outcome Logistic Regression gives a weight value of the likely hood of a given outcome. So Logistic Regression will make predictions like KNN by placing an outcome in particular class by analyzing if it is above or below 0.05 but in addition to this, the data can be further evaluated to see how definitive the placement was which gives more rationale to the prediction.

Some specific advantages of Logistic Regressions are:

- The predicted probabilities are gauged well

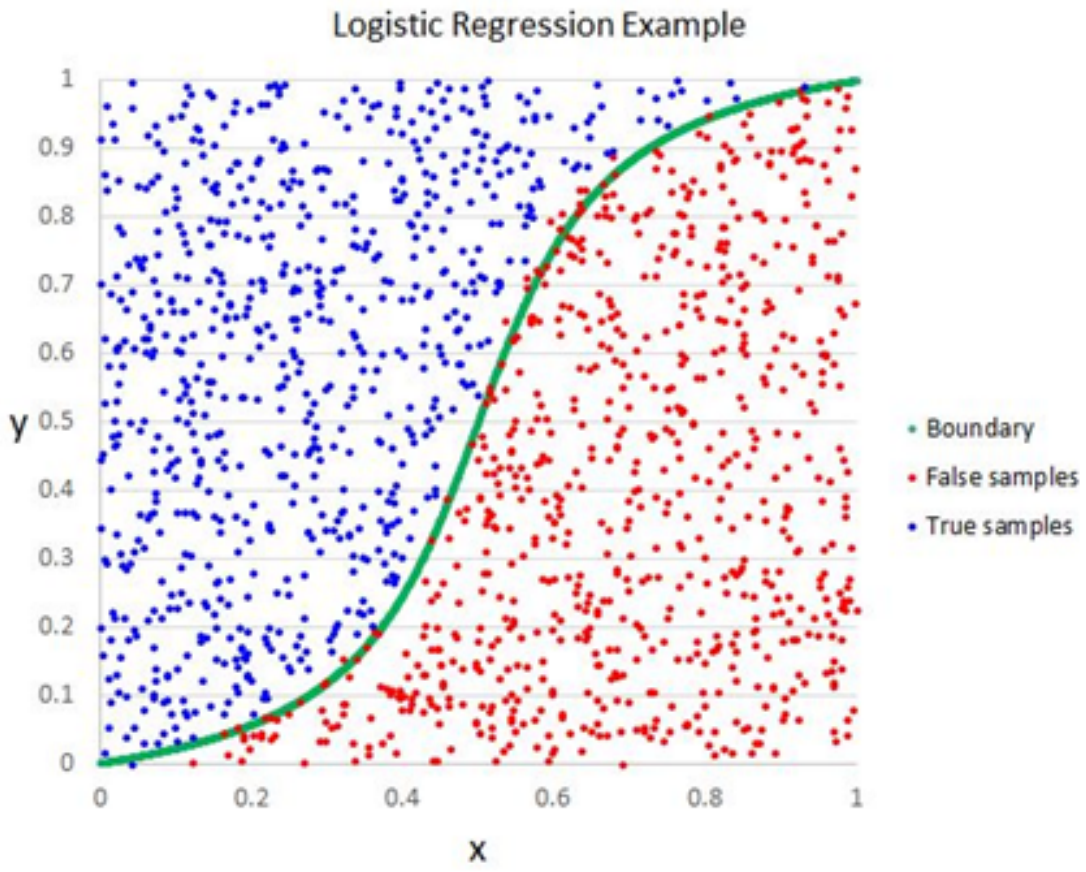


Figure 3.12: Logistic Regression Sigmoid Function.

- The input features do not need to be scaled
- It is easy to interpret
- There is very little or no tuning needed to receive accurate predictions
- It is efficient
- It does not require excessive amounts of computational resources
- It is easy to implement
- It is efficient to train

Some disadvantages that will need to be considered are:

- It is not one of the more powerful algorithms available
- It is sensitive to over-fitting
- Relies on properly identifying the significant independent variables

Another technique of Logistic Regression that may need to be considered is one-versus-all (OvA). OvA enables the training of up to 10 different binary classifiers. This could be useful when analyzing turn data since we will be able to analyze wins, losses and ties.

Support Vector Machines (SVM)

Support vectors are the data points that are closest to the decision boundary (or hyperplane) and influence the position and orientation of the hyperplane. They are the data points that are most difficult to classify. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

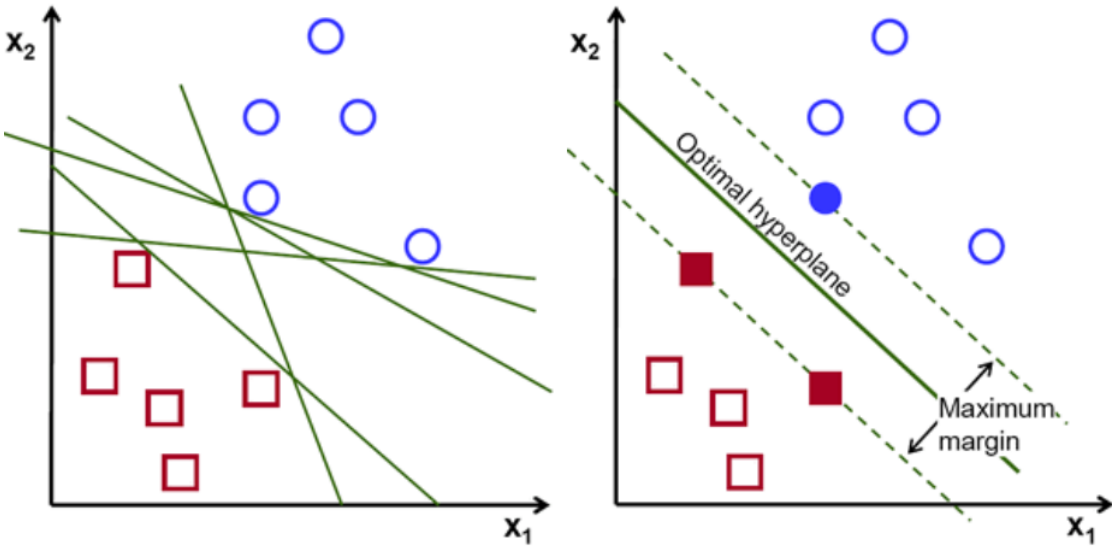


Figure 3.13: Support Vector Machine *Source: TowardsDataScience.com*

Support Vector Machines perform well high dimensional spaces, as it has the advantage of implementing the kernel trick. The kernel trick allows you to be able to transform the data into a required form.

SVM's are also memory efficient, as it only uses a subset of the training points as a decisive factor for classification. As versatile as SVM's are, they do not perform well when there is a lot of noise, or target overlap, in the data set and does not provide probability estimates, which are desirable in most classification problems.

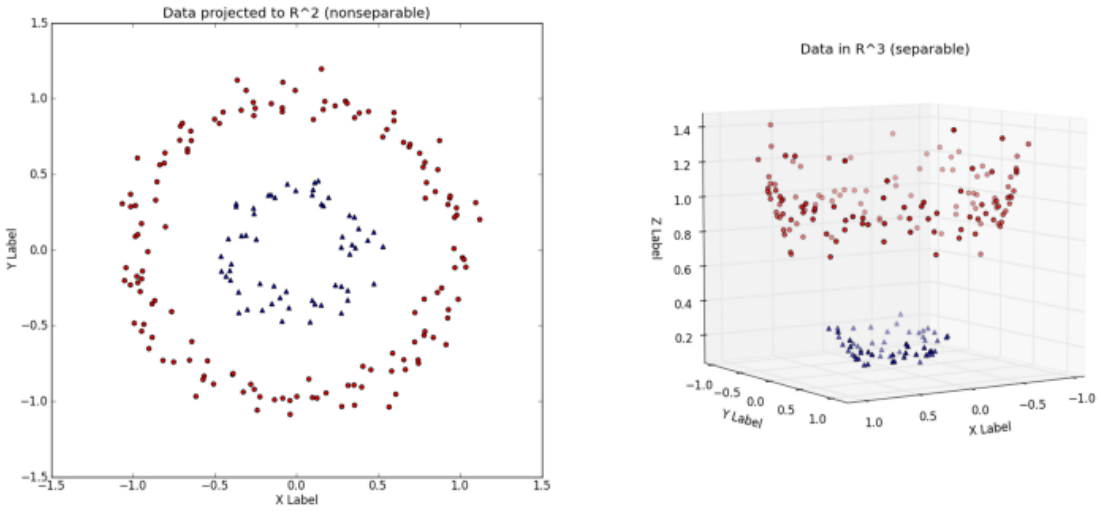


Figure 3.14: SVM using Polynomial Kernel *Source: StackExchange.com*

Random Forest Trees

As its name might suggest, Random Forest Trees uses decisions trees as its building blocks for determining which class an observation belongs too. The algorithm creates an arbitrary number of decision trees based on the data it is given. These trees are a training model used to predict a class by creating rules from the training data. The example Figure 3.15 is a simple representation of a single tree that determines if a person should buy or not to buy a vehicle.

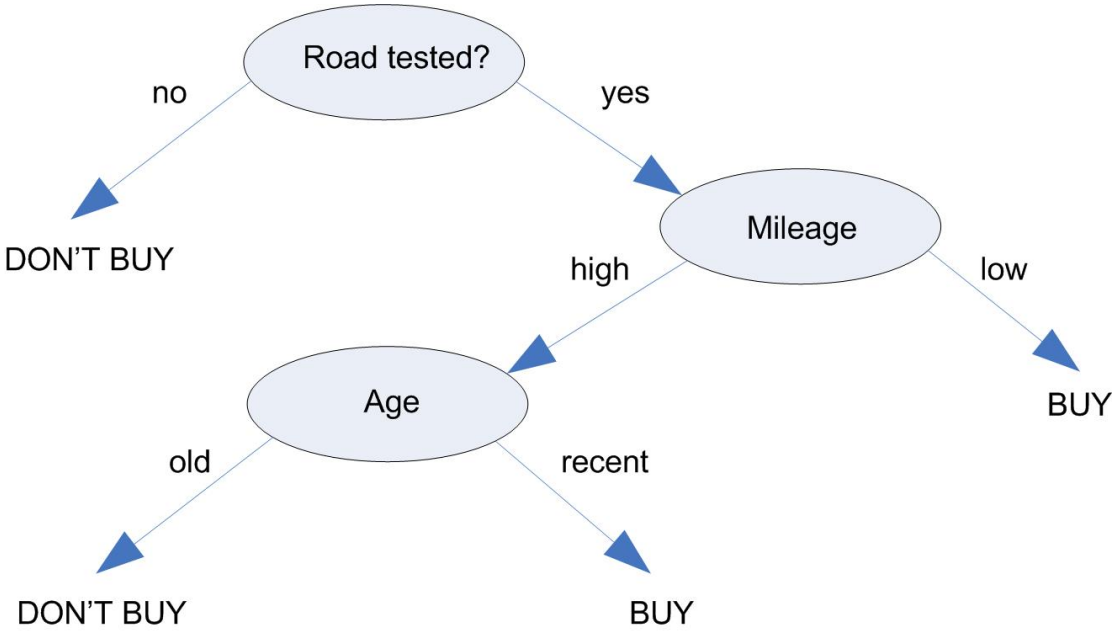


Figure 3.15: Decision tree for buying a car. *Source: IBM.com*

In decision trees the root nodes define the rules which are determined using infor-

mation from Gain and Gini Index Calculations but Random Forest take a different approach to determine nodes, it is done randomly.

Each of the trees is assigned a set of predictor values that are chosen independently and has the same distribution for all the trees in the forest. Each tree is able to produce a response based on the predictor values in the form of a class which is associated with the dependent variable. These trees vote for the most popular class. The Random Forest uses a margin function that determines the average vote for a given class. All of the averaged votes are compared to one another and the highest is the correct class. This method, like Neural Networks and Logistic Regression, provides us with a quantifiable measure from the predictions which gives a bit more insight of how certain the model was in making a decision.

There are some disadvantages with Random Forest though. One of them being decision trees tend to over fit data but this is remedied to an extent with the randomization of the rules in every tree. It is also known to give low prediction accuracy when compared to other machine learning algorithms and lastly, as the number of variables increase the complexity of the calculations increase.

Despite these disadvantages Random Trees still seem like an alternative to keep in consideration. They are easy to explain as they follow a very simple decision making process and they provide quantifiable measures for each prediction which helps with determining how accurate a model is.

3.6 Sci-Kit Learn

Choosing Scikit-learn as our library for utilizing machine learning modeling was a clear choice. Scikit-learn is a free open source library in Python that features various classification, regression, and clustering algorithms and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. NumPy is a Python library that add support for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. SciPy is also a free and open-source Python library used for scientific and technical computing. SciPy contains modules for optimization, linear algebra, and other tasks common in science and engineering. SciPy is built on top of NumPy.

Important features of scikit-learn:

- Simple and efficient tools for data mining and data analysis. It features various classification, regression, and clustering algorithms including support vectors machines, k-means, random forests, etc
- Accessible to everybody and reusable in various contexts
- Built on the top of NumPy, SciPy, and matplotlib

Before installing scikit-learn, ensure that you have the NumPy/SciPy stack installed. The stack includes:

- Numpy: Base n-dimensional array package
- SciPy: Fundamental library for scientific computing
- Matplotlib: Comprehensive 2D/3D plotting
- IPython (Jupyter Notebook): Enhanced interactive console
- Sympy: Symbolic mathematics
- Pandas: Data structures and analysis

A dataset is a collection of data. A dataset generally has two main components:

- Features: (inputs, attributes, etc) the variables of the data. They can be more than one and are represented by a matrix (commonly represented by 'X'). A list of all the features names is called feature_names
- Target: (response, label, output, etc) the output variable depending on the feature variables. We generally have a single response column and is represented by a vector (commonly represented by 'y'). All the possible values taken by a response vector are called target_names

Note the capital 'X' for the input variables and lowercase 'y' for output variables. Matrices are represented with capital letters and vectors with lowercase letters. We carry that convention into programming representation.

Scikit-learn comes loaded with a few example datasets like iris dataset for classification and Boston house prices dataset for regression. You can also load an

external dataset. For this purpose, we can use pandas library for easily loading and manipulating dataset.

In pandas, important data types are:

- Series: Series is a one-dimensional labeled array capable of holding any data type
- DataFrame: It is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It's the most commonly used pandas object

In this section, we will be using the example the Iris, Boston House Prices, and Diabetes datasets from scikit-learn to implement Logistic Regression, K Nearest Neighbors, Linear Regression, and Ridge/Lasso Regression using scikit-learn, numpy, scipy, and matplotlib.

3.6.1 Data in Scikit-Learn

The Data Matrix

Machine learning algorithms implemented in scikit-learn expect data to be stored in two-dimensional array or matrix. The arrays can either be numpy arrays. The size of the array is expected to be [n_samples, n_features]

- n_samples: The number of samples or observations. Each sample is an
- n_features: The number of features or distinct traits that can be used to describe each observation in a quantitative manner. Generally real-valued, but can be Boolean or discrete-valued in some cases

The number of features must be fixed in advance.

Every implementation of machine learning algorithm has the same basic components. You need to:

- Prepare the data

- Create the model
- Train the model
- Evaluate the model

Preparing the data is usually the most difficult part, as it involves not only collecting the data but also transforming that data into a format that can be utilized by a chosen algorithm. This can also involve dealing with missing values or corrupted/malformed data.

Creating the machine learning model is fairly straightforward when using a library like scikit-learn. There are typically only a few lines of code needed to instantiate a given machine learning algorithm. However, there are different arguments and parameters these algorithms take that will affect the model's accuracy.

Training the mode is also a rather straight forward when using scikit learn, as it also only takes a few lines of code. When training the model, however, the best approach is to split the data into a training set and test set. You will see later on why that is better than training and testing on the exact same data points.

3.6.2 A Simple Classification Example: The Iris Dataset

The Iris Dataset is a famous multivariate data set introduced by the British statistician and biologist Ronald Fisher in 1936. It consists of 50 samples from each of the three species of Iris (Iris setosa, Iris versicolor, Iris virginica). Four features were measured from each sample. The sepal length and width as well as the petal length and width, in centimeters.

Figure 3.16: Images from the Iris Dataset.

Loading the Iris Data with Scikit-Learn

Scikit-learn has a very straightforward set of data on these iris species. The data consist of the following:

- Features in the Iris dataset:
  - sepal length (cm)
  - sepal width (cm)
  - petal length (cm)
  - petal width (cm)
- Target classes to predict:
  - Setosa
  - Versicolor
  - Virginica

Scikit-learn allows for us to easily load the iris dataset through a library function `load_iris` and store it in our variable `iris`.

```
# import load_iris function from datasets module
from sklearn.datasets import load_iris
```

Figure 3.17: Importing the load_iris function.

```
# save "bunch" object containing iris dataset and its attributes
iris = load_iris()
```

Figure 3.18: Storing the results of the load_iris function into a variable.

Note: Import `sklearn` Note that `Scikit-learn` is imported as `sklearn`.

Once we have loaded the data into `iris` we are able to access the shape stored in the data set.

```
# check the shape of the features (first dimension = number of observations, second dimension = number of features)
print(iris.data.shape)
```

Figure 3.19: Output the shape of the data.

The result of this is the following:

```
(150, 4)
```

The `data` attribute of a dataset stores the actual data:

```
# print the iris data
print(iris.data)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

Figure 3.20: Output the full data array.

The information about the class of each sample is stored in the `target` attribute of the dataset:

```
# print integers representing the species of each observation
print(iris.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Figure 3.21: Output class information.

The names of the classes in the dataset are stored in the `target_names` attribute of the dataset, their position in the array represents the number used for them in the `target` attribute:

```
# print the encoding scheme for species: 0 = setosa, 1 = versicolor, 2 = virginica
print(iris.target_names)
```

Figure 3.22: Output the names of the classes.

The result of this is the following:

```
['setosa', 'versicolor', 'virginica']
```

The names of the features of the dataset are stored in `features_names`, the position in the array lines up with the values found in the `data` attribute:

The result of this is the following:

```
# print the name of the four features
print(iris.feature_names)
```

Figure 3.23: Output the names of the features.

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
    width (cm)']
```

Introducing the Scikit-learn estimator object

Every algorithm in `Scikit-learn` is an "estimator" object. Below, we grab the `LogisticRegression` model and fit the given data into that model, where `X` is the data array, and `y` is the class value for each data point.

```
# import Logistic Regression Class
from sklearn.linear_model import LogisticRegression

# instantiate the model
logreg = LogisticRegression()

# fit the data in the model
logreg.fit(X, y)
```

Figure 3.24: Fitting the data into a Logistic Regression model.

In this example, we are training on the entire data set and then will predict on new data. Another evaluation procedure is the train/test split, in which you take the entire data set and split it so that a portion is used to train the data and a portion is used to test. This way, we avoid overfitting the training data when training and testing on the same data.

```
# new observations
X_new = [[3, 5, 4, 2], [5, 4, 3, 2]]

# predict the response values for the observations in X
logreg.predict(X_new)
```

Figure 3.25: Testing on separate data from the training set.

The result of this is the following:

```
array([2, 0])
```

The values of the output represent the target integer for each new observation. The first data is predicted to be a virginica iris and the second is a setosa iris.

We can also train/test split the data, so that we are able to train and test on the

same data while reducing the possibility of overfitting. Advantages of train/test split:

- Models can be trained and tested on different data than the one used for training
- Target values are known for the test dataset and predictions can be evaluated
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance
- Reduces the chance of overfitting the training data

The train_test_split function takes several arguments:

- `X`, `y`: The feature matrix and target vector which needs to be split
- `test_size`: The ratio of test data to the given data
- `random_state`: If `random_state = some_number`, then you can guarantee that the split will always be the same. This is useful if you want reproducible results, for example, in testing for consistency in the documentation (so everyone gets the same results)

```
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=4)
```

Figure 3.26: Splitting a feature matrix and target vector into training and testing sets.

`test_size = 0.4` splits the data where the test is 40% of the data and 60% is used for training.

`random_state = 4` doesn't matter what the actual int is. Just matters that it's used every time to get the same split in the data.

```
# print the shapes of the new X objects
print(X_train.shape)
print(X_test.shape)
```

Figure 3.27: Printing the shapes of the new objects.

The result of this is the following:

```
(90, 4)
(60, 4)
```

From this point, it's very similar to the other to before, where we fit the data into the model then predict the X_test. So we train the `X_train`, `y_train` data set:

```
# train the model on the training set
logreg.fit(X_train, y_train)
```

Figure 3.28: Training a model using the training data split previously.

Now we get the compare the actual response values with the predicted response values:

```
# make predictions on the testing set
y_pred = logreg.predict(X_test)

# compare actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print(metrics.accuracy_score(y_test, y_pred))
```

Figure 3.29: Make predictions and score how the model did.

The result of this is the following:

```
0.933333333333333333
```

The closer the value is to 1, the more accurate the hypothesis/model is.

kNN: K-Nearest Neighbors

K nearest neighbors (kNN) is one of the simplest learning strategies: given a new, unknown observation, look up in your reference database which ones have the closest features and assign the predominant class. It assumes that similar things exist in close proximity. In other words, similar things are near to each other. kNN does not make any assumptions of the data distribution, therefore it is non-parametric.

kNN model works by taking a data point and looking at the 'k' closest labeled data points. The data point is then assigned the label of the majority of the k closest points.

Let us try the kNN model on our classification problem with `K = 1` by fitting our iris classification data to the model:

```
from sklearn import neighbors, datasets
knn = neighbors.KNeighborsClassifier(n_neighbors=1)
knn.fit(X, y)
```

Figure 3.30: Fit data using the kNN model.

We can now use our fitted model to try to predict what kind of iris has a 3cm x 5cm sepal and 4cm x 2cm petal:

```
print(iris.target_names[knn.predict([[3, 5, 4, 2]])])
```

Figure 3.31: Use the model to make a prediction from our input.

The result of this is the following:

```
['virginica']
```

We can generate a classification map for the KNN model trained on our dataset to help us visualise how the algorithm is making its decisions.

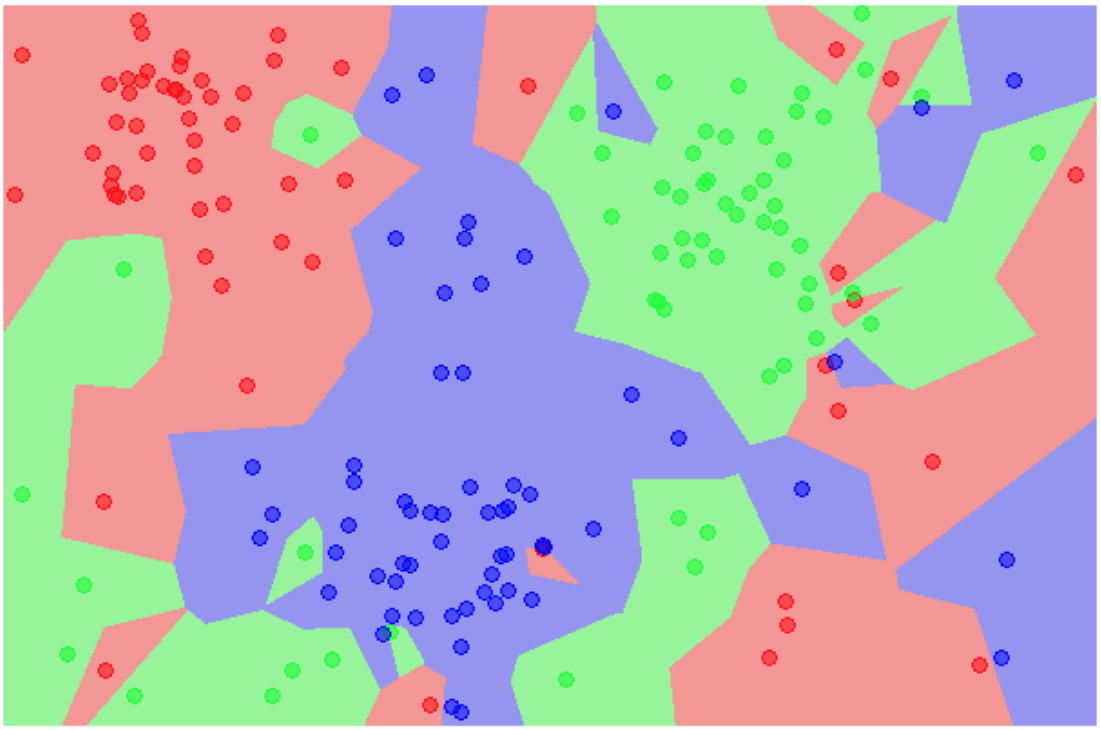


Figure 3.32: KNN classification map for k = 1.

Using a different value for K

The KNN model is highly dependent on the value of `K`, so lets try training the a KNN model on our dataset with a value of `K = 5`:

```
from sklearn import neighbors, datasets
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)
```

Figure 3.33: Fit data using the kNN model where k = 5.

We can then used this newly trained model where `K = 5` to make the same prediction we tried previously of what kind of iris has a 3cm x 5cm sepal and 4cm x 2cm petal:

```
print(iris.target_names[knn.predict([[3, 5, 4, 2]])])
```

Figure 3.34: Use the model to make a prediction from our input.

The result of this is the following:

```
['versicolor']
```

We can now generate a new classification map for the updated KNN model to see how the higher K value affects its classifying decisions.

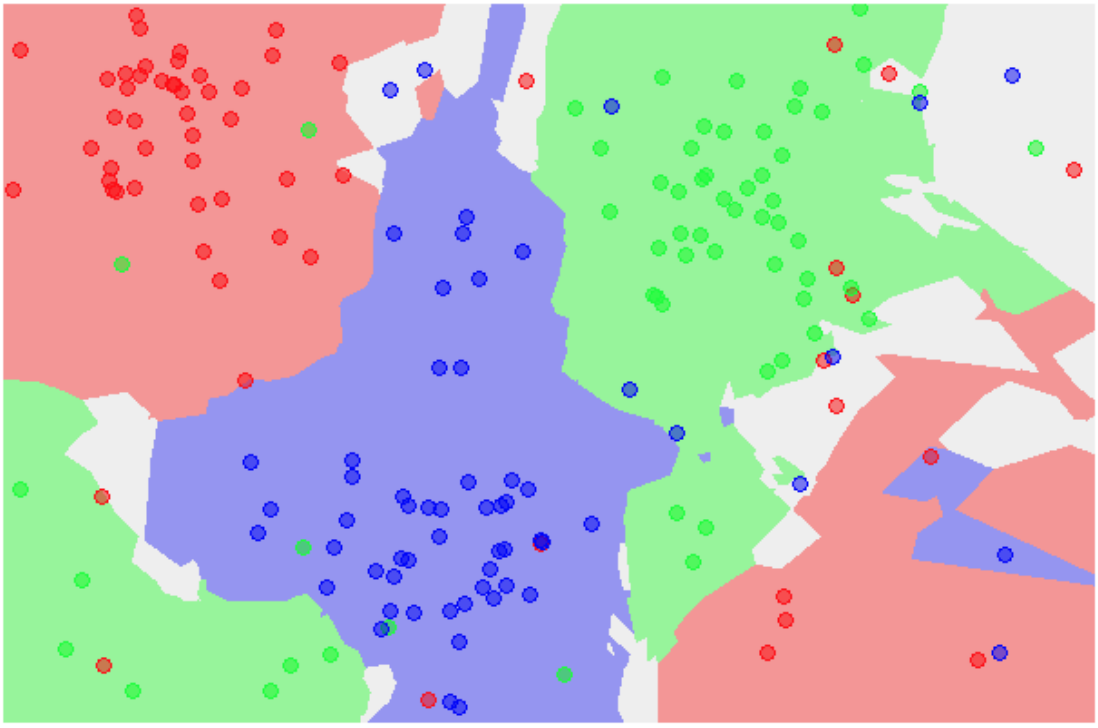


Figure 3.35: KNN classification map for k = 5.

So the question with KNN becomes about finding the best value of `K` for your dataset. One way to do is by simply training and testing the model with many possible values of `K` and comparing their accuracy. We can calculate accuracy by using the `metrics.accuracy_score` function with our test values and predicted values.

```
# try K=1 through K=25 and record testing accuracy
k_range = list(range(1, 26))
score = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    print("KNeighbors(n_neighbors= %s): %s" % (k, metrics.accuracy_score(y_test, y_pred)))
    score.append(metrics.accuracy_score(y_test, y_pred))
```

Figure 3.36: Test our prediction with a variety of K values.

The result of this is the following:

```
KNeighbors(n_neighbors= 1): 0.95
KNeighbors(n_neighbors= 2): 0.95
KNeighbors(n_neighbors= 3): 0.9666666666666667
KNeighbors(n_neighbors= 4): 0.9666666666666667
KNeighbors(n_neighbors= 5): 0.9666666666666667
KNeighbors(n_neighbors= 6): 0.9833333333333333
KNeighbors(n_neighbors= 7): 0.9833333333333333
KNeighbors(n_neighbors= 8): 0.9833333333333333
KNeighbors(n_neighbors= 9): 0.9833333333333333
KNeighbors(n_neighbors= 10): 0.9833333333333333
KNeighbors(n_neighbors= 11): 0.9833333333333333
KNeighbors(n_neighbors= 12): 0.9833333333333333
KNeighbors(n_neighbors= 13): 0.9833333333333333
KNeighbors(n_neighbors= 14): 0.9833333333333333
KNeighbors(n_neighbors= 15): 0.9833333333333333
KNeighbors(n_neighbors= 16): 0.9833333333333333
KNeighbors(n_neighbors= 17): 0.9833333333333333
KNeighbors(n_neighbors= 18): 0.9666666666666667
KNeighbors(n_neighbors= 19): 0.9833333333333333
KNeighbors(n_neighbors= 20): 0.9666666666666667
KNeighbors(n_neighbors= 21): 0.9666666666666667
KNeighbors(n_neighbors= 22): 0.9666666666666667
KNeighbors(n_neighbors= 23): 0.9666666666666667
KNeighbors(n_neighbors= 24): 0.95
KNeighbors(n_neighbors= 25): 0.95
```

We can now use `matplotlib` to make a simple plot of the relationship between the value of `K` and the testing accuracy.

```
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

Figure 3.37: Plot our accuracy values using matplotlib.

```
Text(0, 0.5, 'Testing Accuracy')
```

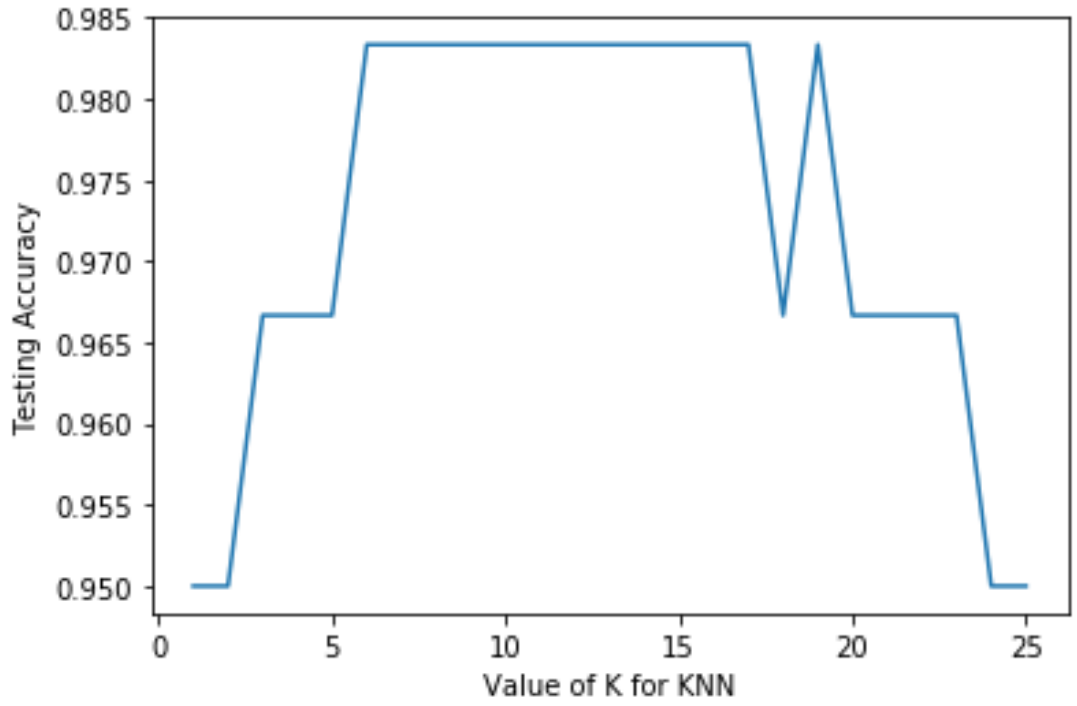


Figure 3.38: Plot of accuracy values for differing K values.

3.6.3 A Simple Linear Regression Example: Predicting House Prices

Now we'll use `Scikit-learn` to perform a simple linear regression on the housing data. In our example, we will be using the `LinearRegression` model on the Boston house prices set. This dataset records measurements of 13 attributes of housing markets around Boston, as well as the median price.

```
from sklearn.datasets import load_boston
data = load_boston()

# print the shape of the data
print(data.data.shape)
```

Figure 3.39: Load the dataset and print its shape.

The result of this is the following:

```
(506, 13)
```

```
print(data.DESCR)
```

Figure 3.40: Print the shape of the dataset's target.

The result of this is the following:

```
(506, )
```

As we can see, there are over 500 observations.

The DESCR variable has a long description of the dataset:

```
# print the shape of the target
print(data.target.shape)
```

Figure 3.41: Print the description of the dataset.

The result of this is the following:

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median
       Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM    per capita crime rate by town
        - ZN      proportion of residential land zoned for lots over
           25,000 sq.ft.
        - INDUS   proportion of non-retail business acres per town
        - CHAS    Charles River dummy variable (= 1 if tract bounds
           river; 0 otherwise)
        - NOX     nitric oxides concentration (parts per 10 million)
        - RM      average number of rooms per dwelling
        - AGE     proportion of owner-occupied units built prior to 1940
        - DIS     weighted distances to five Boston employment centres
```

```
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks
            by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at
    Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics &
    Management,
vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression
    diagnostics
...', Wiley, 1980. N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning
    papers that address regression
problems.
   .. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying
       Influential Data and Sources of Collinearity', Wiley, 1980.
       244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based
       Learning. In Proceedings on the Tenth International Conference of
       Machine Learning, 236-243, University of Massachusetts, Amherst.
       Morgan Kaufmann.
```

Sometimes it helps to quickly visualize pieces of the data using histograms, scatter plots, or other plot types. With matplotlib, let us show a histogram of the target

values: the median price in each neighborhood:

```
import matplotlib.pyplot as plt
plt.hist(data.target)
```

Figure 3.42: Plot a histogram of the dataset using matplotlib.

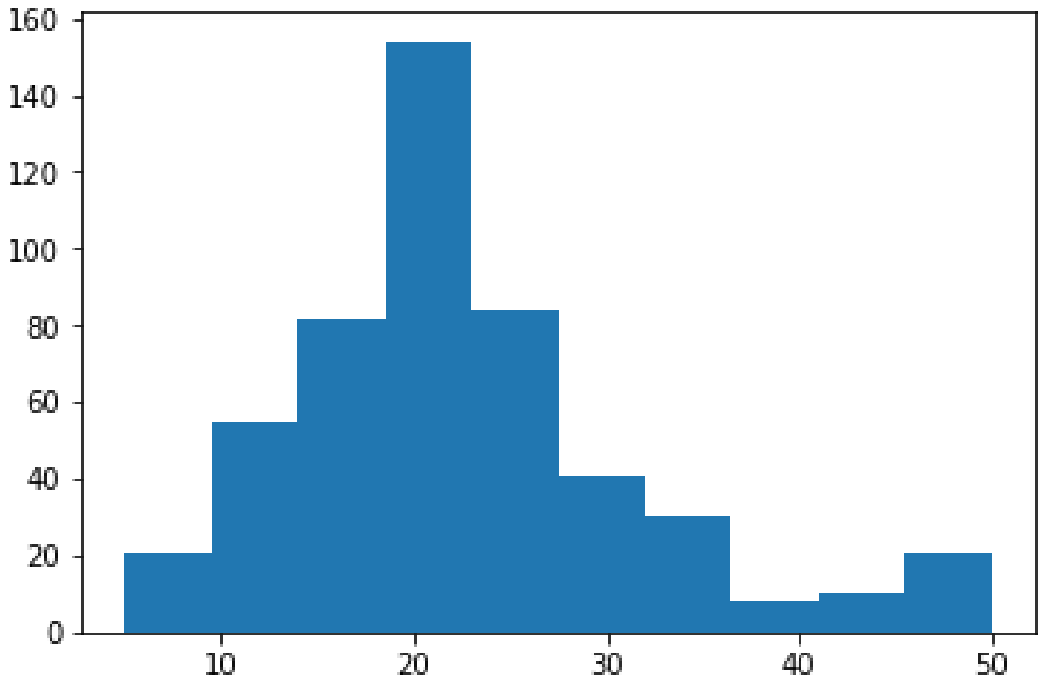


Figure 3.43: A histogram of the Boston Housing Dataset.

Now we'll use scikit-learn to perform a simple linear regression on the housing data. There are many possibilities of regressors to use. A particularly simple one is LinearRegression, which is basically a wrapper around an ordinary least squares calculation.

```
# split the housing data into a training set and test set
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target)

from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

Figure 3.44: Fit the training data to a Linear Regression model.

Now we use the X_test data to predict and the y_test data for the expected data.

```
predict = linreg.predict(X_test)
expect = y_test
```

Figure 3.45: Make predictions on our test data using the model.

Now we plot the data as a scatter plot.

```
plt.scatter(expect, predict)
```

Figure 3.46: Plot our predictions vs the target using matplotlib as a scatter plot.

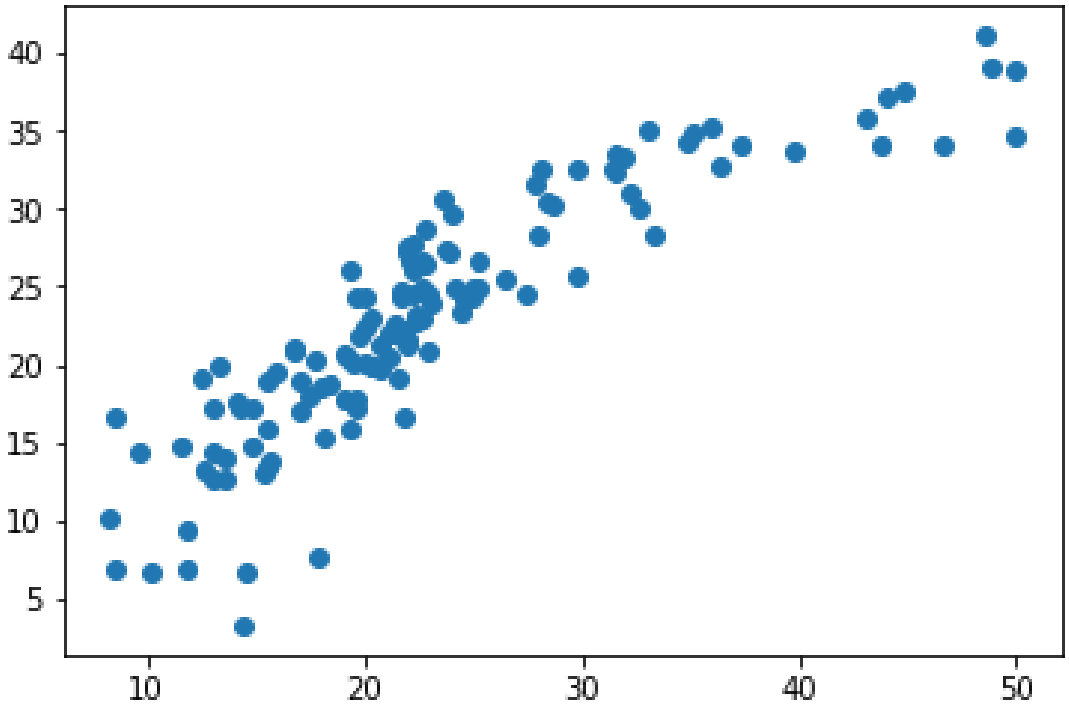


Figure 3.47: Scatter plot of expected value vs predicted.

Cross-validation consists in repetitively splitting the data in pairs of train and test sets, called 'folds'. Scikit-learn comes with a function to automatically compute score on all these folds. Here we'll do KFold with k = 5.

```
from sklearn.linear_model import Ridge, Lasso
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
from sklearn.model_selection import cross_val_score
cross_val_score(knn, X, y, cv=5)
```

Figure 3.48: Run a cross-validation with 5 folds.

This results in the following:

```
array([0., 0., 0., 0., 0.])
```

Ridge and Lasso Regression

Ridge regression is an extension of linear regression and addresses some of the problems of Ordinary Least Squares by modifying the loss function by imposing a

penalty on the size of the coefficients. The ridge coefficients minimize a penalized residual sum of squares.

Lasso Regression, or the Least Absolute Shrinkage and Selection Operator, is al a modification of linear regression. In Lasso, the loss function is modified to minimize the complexity of the model by limiting the sum of the absolute values of the model coefficients (also called the I1-norm)

We'll consider regularized linear models, such as Ridge Regression, which uses I2 regularization, and Lasso Regression, which uses I1 regularization. Choosing their regularization parameter is important.

Let us set these parameters on the Diabetes dataset, a simple regression problem. The diabetes data consists of 10 physiological variables (age, sex, weight, blood pressure) measure on 442 patients, and an indication of disease progression after one year:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_diabetes
data = load_diabetes()
X, y = data.data, data.target
print(X.shape)
```

Figure 3.49: Print the shape of the diabetes dataset.

This results in the following:

```
(442, 10)
```

```
from sklearn.linear_model import Ridge, Lasso
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
from sklearn.model_selection import cross_val_score

for Model in [Ridge, Lasso]:
    model = Model()
    print('%s: %s' % (Model.__name__, cross_val_score(model, X, y).mean()))
```

Figure 3.50: Cross validate the data on both Ridge and Lasso Regression.

This results in the following:

```
Ridge: 0.40942743830329875
Lasso: 0.35380008329932017
```

We compute the cross-validation score as a function of alpha, the strength of the regularization for Lasso and Ridge. We choose 20 values of alpha between 0.0001 and 1.

```
alphas = np.logspace(-3, -1, 30)

%matplotlib inline

for Model in [Lasso, Ridge]:
    scores = [cross_val_score(Model(alpha), X, y, cv=3).mean()
              for alpha in alphas]
    plt.plot(alphas, scores, label=Model.__name__)
    plt.xlabel('alpha')
    plt.ylabel('cross validation score')
    plt.legend();
```

Figure 3.51: Plot the cross validation scores of Ridge and Lasso as a function of alpha.

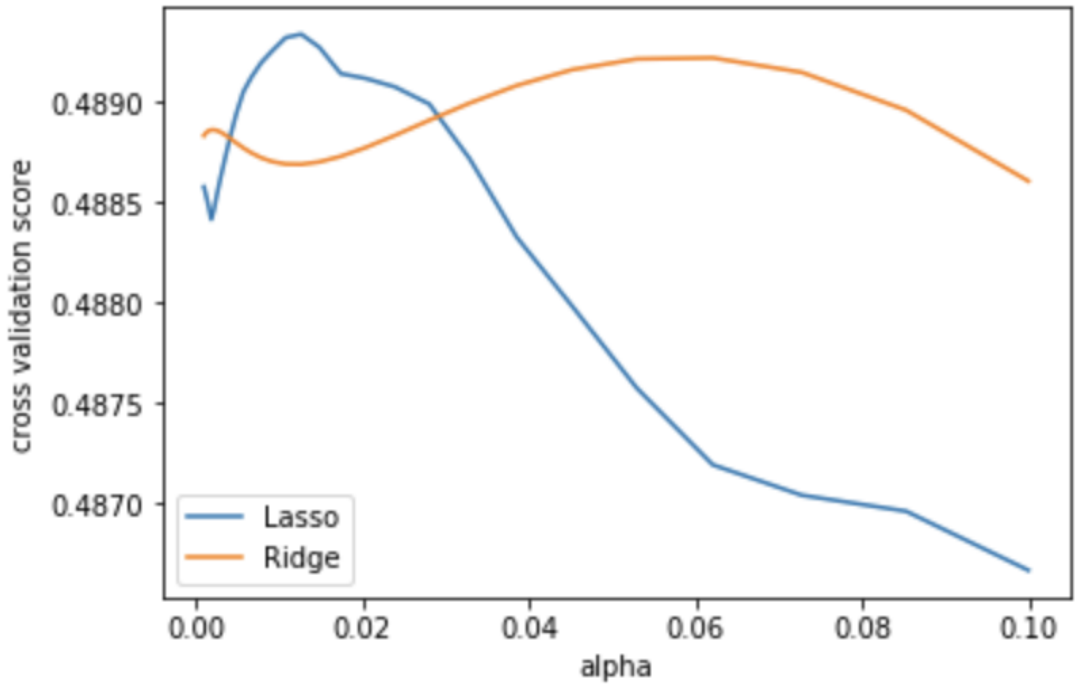


Figure 3.52: Line graph of performance of lasso vs ridge at different values of alpha.

Automatically Performing Grid Search

sklearn.model_selection.GridSearchCV is constructed with an estimator, as well as a dictionary of parameter values to be searched. We can find the optimal parameters this way:

```
from sklearn.model_selection import GridSearchCV
for Model in [Ridge, Lasso]:
    gscv = GridSearchCV(Model(), dict(alpha=alphas), cv=3).fit(X, y)
    print('%s: %s' % (Model.__name__, gscv.best_params_))
```

Figure 3.53: Calculate optimal values of alpha for ridge and lasso.

This results in the following:

```
Ridge: {'alpha': 0.06210169418915616}
```

```
Lasso: {'alpha': 0.01268961003167922}
```

3.6.4 Built-in Hyperparameter Search

For some models within scikit-learn, cross validation can be performed more efficiently on large datasets. In this case, a cross-validated version of the particular model is included. The cross-validated versions of Ridge and Lasso are RidgeCV and LassoCV, respectively. Parameter search on these estimators can be performed as follows:

```
from sklearn.linear_model import RidgeCV, LassoCV
for Model in [RidgeCV, LassoCV]:
    model = Model(alphas=alphas, cv=3).fit(X, y)
    print('%s: %s' % (Model.__name__, model.alpha_))
```

Figure 3.54: Perform a parameter search on our Ridge and Lasso models.

This results in the following:

```
RidgeCV: 0.06210169418915616
LassoCV: 0.01268961003167922
```

Now we can also measure the performance of these estimators. We have used data to set the hyperparameters, so we need to test on actually new data. We can do this by running cross_val_score() on our CV objects. Here we have 2 cross-validation loops going on, called 'nested cross validation':

```
for Model in [RidgeCV, LassoCV]:
    scores = cross_val_score(Model(alphas=alphas, cv=3), X, y, cv=3)
    print(Model.__name__, np.mean(scores))
```

Figure 3.55: Measure the performance of our estimators.

This results in the following:

```
RidgeCV 0.48916142454965533
LassoCV 0.4854913966864774
```

3.7 Unreal Engine

The EVERGLADES project is all built on top of a Unreal Engine game framework, as such we decided it would be important for us to get familiar with the software and see how we can potentially implement the final goal of adding a real-time view to the game to be able to see who is more likely to win based on the actions taken and the data we have analyzed.

Unreal Engine is written with C++, which allows us to be able to code up the real-time view using a programming language we are familiar with. Using C++ allows Unreal Engine to be very portable and allows for compatibility with many different platforms and hardware configurations. Unreal Engine implements an object-oriented approach to how the game project development works.

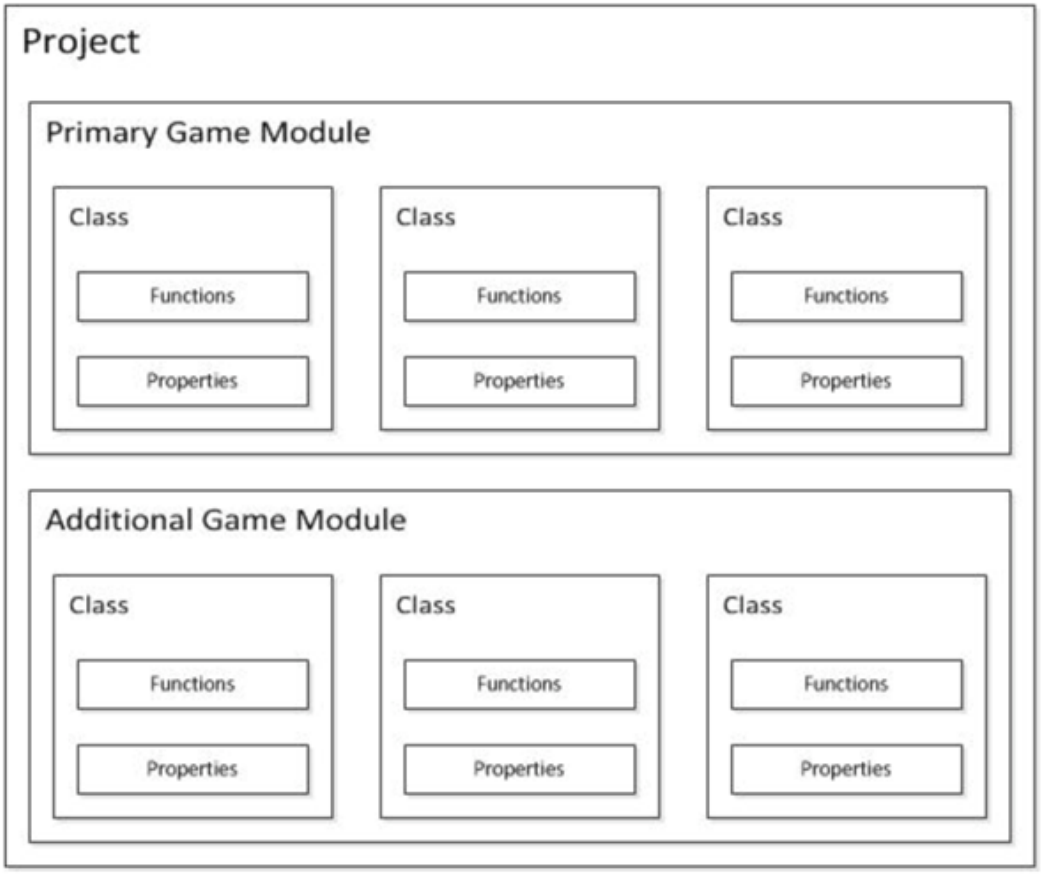


Figure 3.56: Project Design Structure in Unreal Engine.

This approach lets us have a greater understanding of how the game works with all of the entities and relationships between them that the EVERGLADES game has.

The EVERGLADES project uses Unreal Engine as a front end to consume the telemetry data that comes from the back end to render the gameplay to be more appealing for the user. While the game essentially is a 2D array of nodes, the Unreal Engine client converts this in a wonderful 3D environment without messing up the telemetry data in the progress. Any changes done to the front end will have no effect on the telemetry data as they exist in separate entities.

3.8 Docker

The EVERGLADES project heavily relies on docker just as much as it does the Unreal Engine. Docker at its core is a service that allows you to create OS-level virtual machines that can be used to deliver and develop software in packages called containers. These containers are isolated from each other and can store their own software, libraries, and other configuration files as needed. These containers don't only have to work in isolation, even if they can work as such, they can be used together through the many various channels that Docker allows for.

The main reason why Docker has been so widely adopted and is successful is for 3 major features that they have. Docker containers are portable and can be run on any hardware as long as it can run the Docker software. Docker containers are very efficient, in that they do not require the entire OS to be installed for a container to function, saving resources in the process. And lastly, Docker containers are safe to run as they have a high-level isolation to them and prevent anything from leaking out of them.

In our project, there are three main containers that exist: the game server, the gym environment, and the agent. The game server container stores the data related to running and compiling the actual game platform that is being used. The gym environment container takes the data from the game server container and formats it in a way that the agents can be able to use the data and to be able to make their moves. The agent container stores all the data that is related to the agent and is where the agent is stored and ran using the data being fed from the gym environment. There are two agents that run with the gym environment simultaneously, this allows for the turns in the game to go between agents until the game finishes.

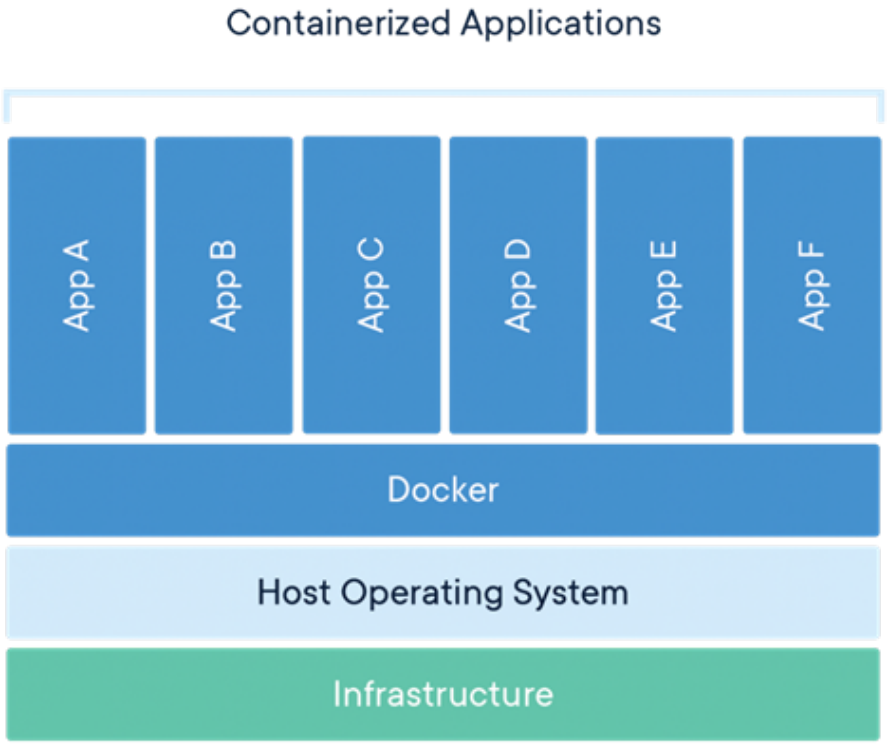


Figure 3.57: Docker Container Structure Overview.

1

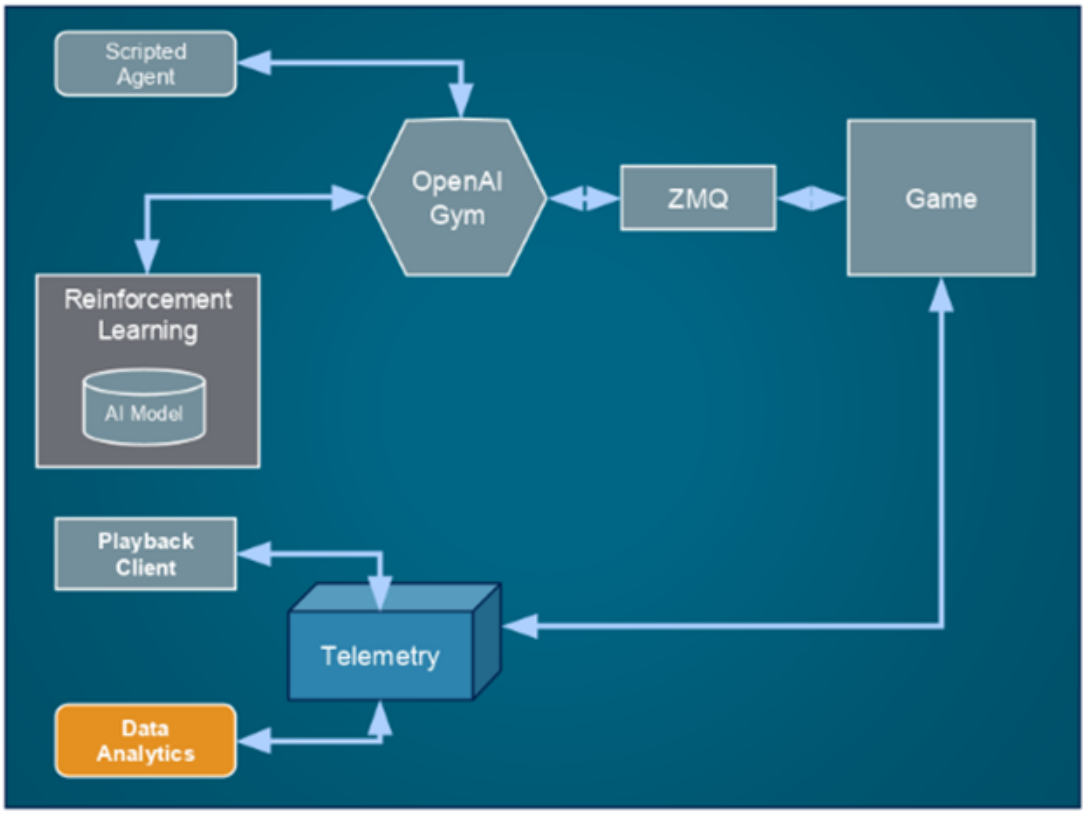


Figure 3.58: Everglades Project Platform.

3.9 Real World Implementations

Data Analytics is a very hot field within the area of Computer Science. Data Analytics is used in many different applications from Sports, Healthcare, and Business. While all of these areas seem to be different, they all have one major thing in common, data. These industries all produce data that needs to be looked at and analyzed, as such data analytics is very important.

Data Analysis in Sports

Data Analysis in sports can be similar to the example we used in our initial design document, baseball analytics. These analytics allow for scouts to get a better view of a player before even having to watch them play. The way the baseball analytics work very similarly to how data analysis can be used for other sports such as American Football and Football (Soccer).

Analytics can show how certain strikers fair when they have the ball at their feet in certain positions and can be used to predict in which way the player will shoot towards goal, or if they will make a pass to an oncoming supporting player. Analysis can work in many differing factors, such as scouting players to see if they will fit a style of play that the manager has in place, scouting the opposition for weaknesses in their game, and for training to make sure the players can understand their mistakes to a higher level. The video below shows a demo of how analytics can be used to assist players playing on the highest level of football. The data being collected in the video can show how effective a player can be without them having to be flashy such as Toni Kroos in this video example. Another example that exists is to see how effective a player would be at playing the sport, so below is another video that shows how effective one of the best players in the game is.

For American Football, analytics can be used to see how effective certain formations are based on the lineup you have at hand, such as choosing four verticals instead of something else. Similarly, to how football does the analysis, scouting the opposition is very important to American Football and is highly sought after. Before the draft many rookies go through what is called the draft combine, during this event the rookies are made to do tasks to show their strengths and abilities that help them with playing the sport. Data analytics have a large effect on the draft combine, by finding relationships between a rookie's ability with the ball and how strong they are to how successful the player will be for their team.

Figure 3.59: Video of Data Analysis on Soccer Gameplay.

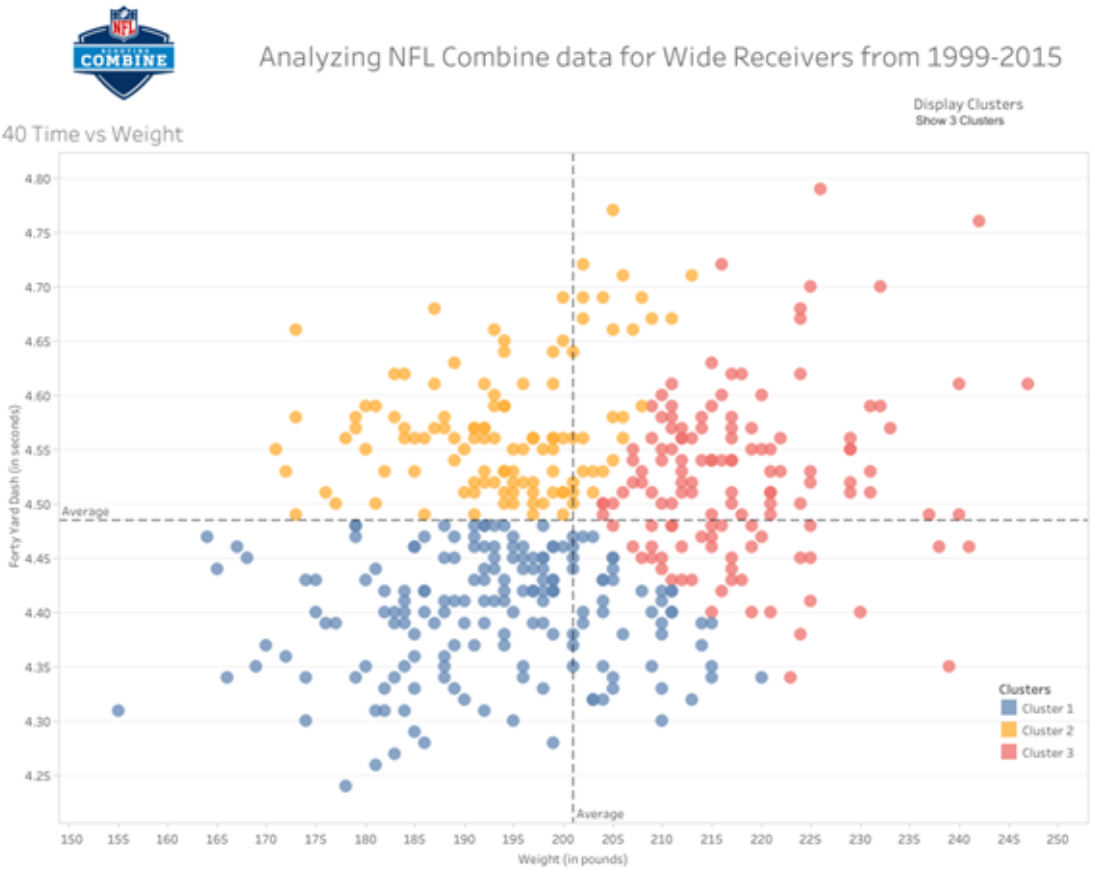


Figure 3.60: NFL Combine Data Analysis on Wide Receivers.

To put further emphasis on how analytics is important to sports in general. Data analytics can be used in preventative medicine to make sure that athletes are at peak condition to perform and have the ability and equipment that allows for them to do so.

Data Analysis in Insurance Markets

Just like how data analysis is useful for sports, it is very much useful to fight information asymmetry in a variety of industries. For example, to maximize revenue, health insurance firms are incentivized to price healthcare according to the amount of risk (expressed in potential healthcare costs) that each of their customers is prone to. However, health insurance is unlike most other sectors in the sense that it is very difficult for insurance firms to accurately price the cost of insurance for each customer. This is a result of the phenomenon known as information asymmetry, which occurs when one party has more or better information than the other. Information asymmetry occurs because, to avoid being targeted for higher insurance prices, customers typically do not disclose information that they believe would be used against them. Information asymmetry is a big issue because producers of a good could potentially provide too little or too much of the good, while consumers have no limit on what they can consume, up to a point where it is unnecessary. It is particularly problematic in the healthcare industry because it ultimately causes the industry to operate inefficiently, which is one of the main reasons for why the U.S. spends so much money on healthcare.

Data analysis provides an avenue to help diminish the effects of information asymmetry by providing firms the opportunity to collect information from their customers. In turn, this data is used to understand their behavior. Over time, predictive models become stronger and better able to assess risk of future customers.

Monitoring is a common method used by insurance companies to gather and analyze data from their customers. In the auto-insurance industry, some firms are introducing usage-based auto insurance using apps to their customers. In principle, the app would be able to detect the movement speed of the vehicle it is placed in, which could indicate risky behavior such as speeding habits. Progressive, one of many large U.S. auto insurance firms, offers customers a discount on their monthly payments through their "Snapshot" program, which tracks a user's driving metrics by installing an app on a given smart device, or by using a separate plug-in on the vehicle.

Figure 3.61: The Progressive "Snapshot" plug-in used to collect data from consumers.

The health insurance industry also has different strategies to engage in monitoring. Health insurance firms use monitoring to lower the healthcare costs of each customer through information campaigns and by encouraging healthy behaviors. Some providers, such as Group Health, use decision aids (informational videos that explain the differences between different treatments clearly) to encourage patients to choose the treatments that are simultaneously cost-efficient and medically effective, while discouraging unnecessary procedures. On the other hand, health insurance firms can also encourage attendance to fitness, nutrition, and wellness classes, which helps to encourage their customers to engage in healthier behaviors and consequently lower their risk. However, health insurance companies are starting to move in the direction of the car insurance industry; now, some workplace insurers are encouraging employees to wear Fitbit and other fitness tracking devices to qualify for a discount ranging from $100-2,000 off yearly insurance premiums. All in all, the intention of these incentives is to use monitoring as a means of gathering information to estimate the costs that customers could impose on them, and then price accordingly.

Figure 3.62: A fitbit used to gather data for insurance companies.

Data Analysis in Stocks

In addition to all the previous industries mentioned, data analysis is also heavily utilized in the stock market. The goal of every investor is to be able to predict when stocks with great potential will be at their lowest price, so that investors can maximize their revenue when that stock takes off. To achieve this goal, it is necessary to engage in stock trading with accurate and timely inputs. Human traders cannot conceivably keep track of more than a few stocks, and even then, humans cannot flawlessly perceive newer and upcoming stocks that may have potential in the future. Due to the magnitude of data that the stock market generates every day, as well as all the various sources that all this information comes from, even considering that a human could carry out thorough analysis is a foolish idea.

This is where data analysis comes in. Unlike financial analysis, data analysis is a more sophisticated way of predicting stock behavior. In the modern world, stock analyses need to incorporate variables that assess other external factors that affect stocks. These factors include the political environment, changes in consumer preferences, social and economic trends in the economy, government regulations, and countless others. It is increasingly important to account for these stocks because the stock market does not exist in a vacuum – anything could possibly impact share prices, and even go so far as to affect shares in entire industries.

Investors are able to utilize data analysis to their advantage via predictive mod-

els, which can incorporate any number of the previously mentioned factors. Furthermore, real-time data analytics can improve investing capabilities by engaging algorithmic analysis that provides time-sensitive insights that investors can take advantage of, which means they can get results on which stocks to invest in more and more quickly over time. By using data analysis, investors can better strategize on how to minimize risks while trading in the stock market.

3.10 Facilities and Equipment

The training of the AI agents and the storage of the telemetry files generated during said training require a reliable and appropriately sized system. The hardware available in the UCF Computer Science Senior Design lab is more than adequate to handle the needs of this project.

The lab has a total of 16 available work stations as well as a server for Virtual Machines and a Storage Server. The specifications for these pieces of hardware are as follows:

- 3x WorkStations
  - i7 4770 (3.4 GHz)
  - 16 GB RAM
- 3x GPU Machines
  - 1080 TI GPU
  - i7 8700 (4.6 GHz)
  - 16 GB RAM
  - SSD
- 5x Micros
  - i7 4785T (2.2 GHz)
  - 8 GB RAM
- 5x Fast Micros
  - i7 7700 (3.6 GHz)
  - 16 GB RAM

- VM Server
  - E5-2650 @ 2.3 GHz (20 cores)
  - 128 GB RAM
- Storage Server
  - E5-2620 @ 2.1 GHz (12 cores)
  - 64 GB RAM
  - 8 TB of storage for projects

All of the machines in the Senior Design Lab are running Windows 10 and have Office 2016 installed along with a wide array of programming and development software. There are also plenty a number of standalone GPUs that could be utilized if absolutely necessary.

In addition to the machines in the senior design lab, our team is utilizing our personal computers as environments to run agents against each other. This was especially helpful in the initial stages of the project when the specific mechanics of the game needed to be investigated and documented.

Our database was initially located on a personal computer as well. It was convenient as we worked to find the right balance between portability and making our database easily query-able. As the database scales, we will transition it to the servers provided in the Senior Design lab or even to a cloud based service as needed.

Chapter 4

Design

4.1 Testing Process

4.1.1 Modeling Procedures

Hypothesis Exploration

In our initial telemetry investigations, we explored two overall hypotheses. Experiment 4.4.2 investigates the theory that the player that loses the first few units has a direct effect on the outcome of the game and that the type of unit lost has a strong correlation with win/loss outcome. Experiment 4.4.1 investigates the theory that the player that moves from a defense node has a direct effect on the outcome of the game and the type of unit and the node that unit moves to has a strong correlation with win/loss outcome.

The goal of the models were to find any correlation between the first units lost and the affect it has on winning games as well as a correlation between the player of the first unit that leaves a node that has defense bonuses and the winner of the game

Data Access/Preparation

In our initial modeling, we used a python script to run the game 150 times to get enough telemetry data to run the machine learning models on. We then ran another python script to access each telemetry file and gather the data for usage in the pandas and Sci-Kit Learn modules in python. The scripts run through the data files of all the telemetry data and gathers the data requested into a summary file that is then converted to a csv file in the proper format for importing into pandas. The csv file stores the data in a two dimensional array [n samples, n features].

The python script is a preliminary data preparation program, as the data will be pulled from a MySQL database that will store all of the telemetry data in stage 2 of the project.

Once the database is set up we plan on having a highly rigid process for making datasets. Making a dataset will follow the following process:

1. Determine what data needs to be in the dataset
2. Write an SQL query to pull the dataset from the database
3. Store the dataset, label, and date it

It is incredibly important for us to be making solid datasets. This will provide consistency to our experiments by allowing us to make sure that we are performing them on the same sets of data. This also means that over the course of the project as we make more and more datasets, we will be able to test our models on newer and older data to verify the validity of our findings.

Hypothesis Testing - Model Building

Relevant Feature Variables

In our initial modeling we wanted to see how the first group lost affected the outcome of the game as well the first tank disband and the last to control a specific fortress node. We found relevant feature variables to be the players of the first, second, third and fourth unit lost, as well as the unit type.

In Experiment 4.3.1, we found the player of the first group lost to be relevant.

Variation of Independent Variables

In Experiment 4.3.1, we used some variation of feature variables, which paired the player with the first group lost along wit its unit type to see how that affected the outcome of the game.

Base Model Evaluation

We evaluate performance of the data using four base parameter models.

Classification Models:
Logistic Regression: C = 1
K-Nearest Neighbors: n-neighbors = 10
Random Forest Classifier: Max Depth = 2
Support Vector Machine: C = 1

We then pick the model that produces the highest performance scores for hyperparamater tuning using Grid Search CV

Grid Search Cross Validation Evaluation

Grid Search CV helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters. In addition to that, you can specify the number of times for the cross-validation for each set of hyperparameters.

Here are the few arguments that need to be defined while running Grid Search CV:

- estimator: The estimator of the model created
- params_grid: The hyperparameters you want to try
- scoring: The performance metric that you want to use

- cv: The number of cross-validation you have to try for each selected set of hyperparameters
- verbose: Can be set to 1 to get the detailed print out while the data is fit to Grid Search CV
- n_jobs: The number of processes you wish to run in parallel for this task. If it -1, it will use all available processors.

We will use Grid Search CV with 10 as the CV value on the model that produced the best performance to increase the performance metrics of the model.

Confusion Matrices

For each model a confusion matrix will be used to further determine how effective the model is performing. A confusion matrix is an n x n array of results from a testing set. It tallies the results based on the predictions that were made. A confusion matrix would group binary game predictions of Player 1 winning or Player 2 winning as follows:

- True Positives: When the model correctly predicts when Player 2 wins.
- True Negative: When the model correctly predicts Player 1 wins.
- False Positive: The model predicted Player 2 wins but Player 1 actually won.
- False Negative: The model predicted Player 1 won but Player 2 actually won.

In addition to the matrix itself it allows us to use other metrics to help determine if our model is effective. The other techniques the Confusion Matrix provides access to are:

- Specificity: How often is the model correct when Player 1 wins?
- Sensitivity (Recall): How often is the model correct when Player 2 wins?
- False Positive Rate: The rate of when the model categorizes Player 2 as the winner when player 1 was the actual winner.

- Precision: Represents how often the model is correct when Player 2 is predicted to win.
- Classification Accuracy: How often the model is correct overall based on the test data.
- Classification Error: How often the model is incorrect overall based on test data.

Learning Curve Evaluation

For the chosen model that produces the maximum accuracy and F1 Score from Grid Search CV, we will use a learning curve to evaluate the bias and variance in the model. The learning curve will plot the Number of Training Examples vs F1 Score.

In our models we are looking for low bias and low variance. We want low bias to avoid building a model that is too simple. a simple model performs poorly on training data, and it's extremely likely to repeat the poor performance on test data.

Similarly, we want low variance to avoid building an overly complex model. Such a model fits almost perfectly all the data points in the training set. Training data, however, generally contains noise and is only a sample from a much larger population. An overly complex model captures that noise. And when tested on out-of-sample data, the performance is usually poor. That's because the model learns the sample training data too well. It knows a lot about something and little about anything else.

Models that are high variance will overfit the training set and models are high bias will underfit the model. Since we can't have a model that achieves both low bias and low variance, we are striving for a trade-off.

Model Conclusion

For each model an overall conclusion will be made. Using the techniques previously stated a verdict will be formed on overall how effective was the model in testing the hypothesis. It will also compare the model to other, similar models to determine

if one should be considered over the other and lastly it will determine any key factors for the developing the AI Scripts.

4.1.2 AI Script Development

The AI Scripts were written by doing massive overhauls of two of the original agents that were provided with the project: All Cycle and Random Action. These two agents were the best documented of the ones that were supplied and were the ones we felt confident in being able to adjust. The first way that we did our overhauls for these agents was to assess what their general logic was trying to accomplish and see if there were ways in which we can help the agents translate it better into the game

4.1.3 Issues with Provided Agents

The major issues that we found in these agents were the following:

1. The agents were not checking if the move they are making is invalid or not.The logic behind the two agents is that they either choose the next node to go to as the All Cycle or to choose a random node to go to like in the Random Action. The flaw in this logic is that not every node has a connection to each other so there is a very high chance that the agents will choose to try to go to a node that will be reachable from the node that the unit it is on.This issue was considered major to us, because it causes the agents to waste turns that it could use to actually put up a challenge to the player/ custom agents.The way that we fixed this problem is to hard-code a shortest path for every node so that way the agents can determine what nodes they are connected to, in order to make smarter decisions on where to go.

2. The agents were not checking to see if a group of units were moving or not.Similar to why the first issue was considered major, this issue caused the agents to waste a significant amount of turns trying to move units that were in the process of moving to another node. The way that we fixed this problem was to utilize the observation array that is passed into the function that contains the data for all of the units and the nodes that are on the field. Using this array,we can detect if a unit group is moving or not and wrote a check to see if the unit was moving so that way another group could be selected instead.

3. The last major issue we found in the agents was that the agents were trying to move unit groups that had 0 hp. When a unit is considered dead they were not completely removed from the game, this issue once again causes wasted turns for the agents that could be better used trying to play the game. The way we fixed this bug was similar to how we fixed issue number 2, the observation array. The observation array also contains the health information for the unit groups, so we created a check to see if the unit group was dead or not before selecting them.

4.1.4 Testing Script Development

Once we fixed the base agents given to us, we then created our testing agent scripts. One of these was a set of scripts that took the re-vamped All Cycle agent and use it cycle to each node that was on the board. Depending on the node we could test how effective the strategy of rushing the base was for winning the game as quickly as possible. We also took these agents and implemented a timer like feature where they would play the game like usual and then activate their cycling paths to specific nodes, this gave us more information on when it was best to move units toward the base in the game than the basic re-vamped agent

4.1.5 Evaluate AI Scripts - Modeling

Once the AI Agents have been adjusted to target our desired metrics, the telemetry data produced from their testing can be used to evaluate how effectively our adjustments were made as well as the validity of the metric overall. This newly generated data set will be modeled using the same procedures outlined in Section 4.1.1, with emphasis placed on comparing the results to the initial, pre-modification data set. K Value Analysis, Cross Validation, and Confusion Matrices will be used as needed to better visual the data.

For each degree of adjustment implemented during training, modeling must be done to determine the ideal adjustment degree. Comparing each of these data sets with the original will allow us to draw conclusions regarding how strongly or delicately the adjusted Agents should be rewarded for pursuing the target metric to maximize the potential for success.

4.1.6 Overall Conclusion

Once the data generated during testing has been modeled, it will be possible to evaluate the relevancy of the metric being investigated and determine the level of impact it has on win rate. Our categorization of each metric after testing can be split into three potential outcomes

Validated

If it has been found that our target metric affects win rate in a demonstrable way that can be visualized using our modeling techniques, we can pronounce it as validated and list it as a SabreMetric. It can be given a descriptive identifier (ex. On Base Percentage in baseball) and used to help improve the development of future Agents. Even if the effect of the metric was opposite to the initial hypothesis proposed but is impactful, it can be validated. That is to say, if maximizing a metric was thought to increase win rate but actually dramatically decreased it, that metric can still be used.

Rejected

If a metric has been found to have little or no effect on the win rate of Agents, that metric will be rejected and not used in development of our win rate algorithm. Rejecting metrics still provides us with valuable insight, however. Knowing what does not affect the outcome of the game can be just as useful as knowing what does.

Requires Further Investigation

The final, and probably most frequent result of our investigations, will be a re-submission of the metric for further testing. The need for further exploration can be caused by many things. The modification of the agents could have impacted too many factors of gameplay for the results to be used to validate the specific target metric. The levels of reward that were tested could also be insufficient. For example, if we increased the reward for a certain action by 5%, 10%, and 15%, and the 15% increase had the most impact on win probability, we would need to test higher levels of increase to determine the optimal level.

4.1.7 Design Process Documentation

Each individual portion of the Design process has been illustrated throughout Chapter 4. In future, we believe that to make our findings more accessible to individuals outside of our team, a consistent and clear representation of our process should be generated. This template has been roughly outlined in Section 4.4.

4.2 Database

4.2.1 Database Design

Our Database design is based entirely around the data we covered in the Raw Data section. In that section we discussed how we were primarily interested in the Match Telemetry and Map JSON files. Map Information is pretty simple to track and parse as we can just store the name of the file it is contained in. Therefore our primary concern is parsing the information stored in the Match Telemetry files and figuring out the best way to store them for our purposes.

From a high level perspective the EVERGLADES system consists of two User Agents playing in a Game that involves capturing Nodes using Groups of Units. This description shows us the following core objects of the game:

- Games
- Agents
- Nodes
- Groups
- Units

This high level view lets us create the foundation of our database by creating tables that track basic information about each of these units. This can be seen in Figure 4.1. However this fails to capture the full picture, while this gives us reference points for each object it does not provide us with their state information throughout the course of the game and therefore fails to provide a complete picture

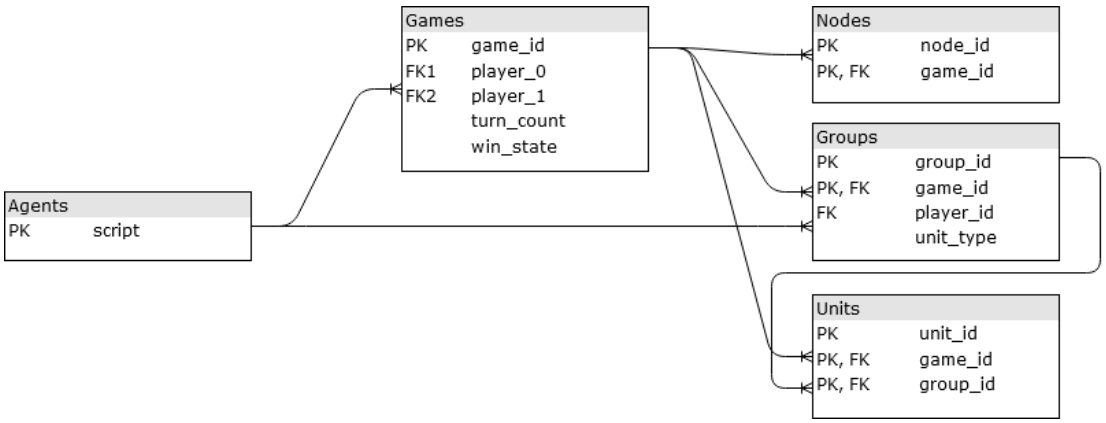


Figure 4.1: Basic Database ER Diagram.

of the data. This will lead to incorrect observations and cause us to come to false conclusions. To ensure that we maintain data integrity our database design should be able to take all of the telemetry files generated by game play. This starting point only covers the following telemetry files:

- Telem_PLAYER_Tags
- Telem_GROUP_Initilization

What we fail to capture in this high level overview is the actual details of the game objects and how they change over the course of the game. To track these changes we have to add extra tables that track those data points over time. These tables reference our initial object tables to allow for easy querying of specific data and to produce a clear relational structure. Our new database design with this taken into consideration can be seen in Figure 4.2.

This updated design allows for us to absorb all of the data provided by the telemetry files easily into the database in an efficient and logical manner. By tracking the events that occur on each object, we can identify the state of the game at any point by recreating it using those values. This design also allows for us to easily query specific events / changes in the game state to test whether they will be useful for identifying play patterns and other characteristics. The design consists of 12 tables:

1. Agents
2. Games

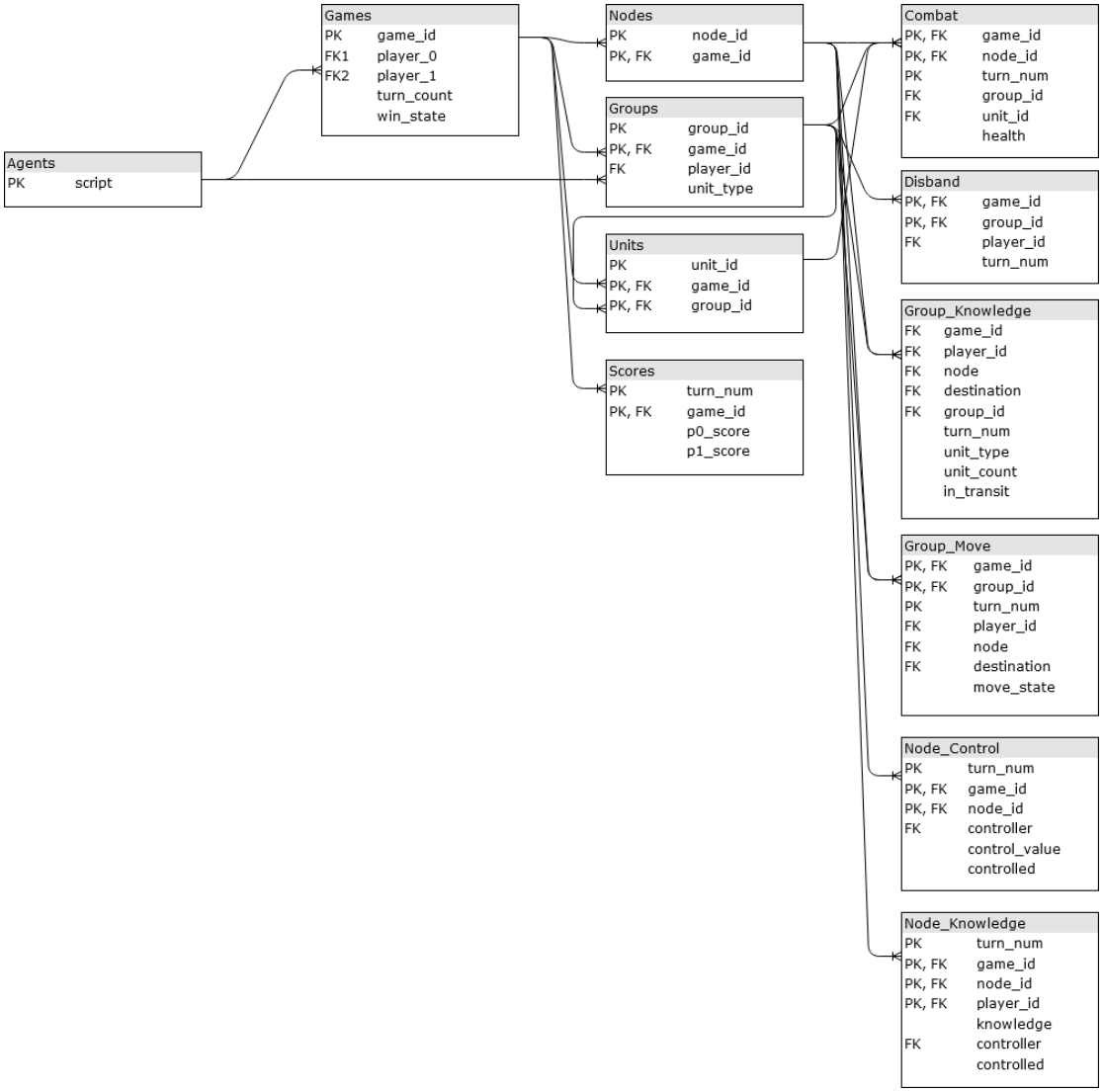


Figure 4.2: Complete Database ER Diagram.

3. Nodes
4. Groups
5. Units
6. Scores
7. Combat
8. Disband
9. Group_Knowledge
10. Group_Move
11. Node_Control

12. Node_Knowledge

An advantage of designing our database around the telemetry output is that it will make collaboration with other groups working on the EVERGLADES projects much easier. This is because all that will be necessary for collaboration will be to have them send us their telemetry files and we will immediately be able to store them in the database and begin identifying patterns in their agents. A potential goal for this would be to make a simple web portal where other groups could upload data from their agents and it would be automatically stored into the database.

Agents Table

The Agents table is rather simple. It tracks just the following:

- script: The filename of the script ran by the agent

We keep track of this because it allows us to look at performance of a specific agent over multiple games. This allows us to look at far more decisions made by an agent than if we just tracked a singular game. We can also use this to evaluate the performance of certain scripts vs other scripts, this type of analysis would allow us to see how an agent's playstyle changes against differing opponents.

Games Table

The Games table stores information about each individual game in the database. It tracks the following:

- game_id: The ID of this game, it is auto generated by the database
- player_0: The script used by player_0
- player_1: The script used by player_1
- turn_count: The amount of turns the game took before ending
- win_state: The way the game ended

This table is vitally important as it is what gives our data its key structure. It is important to understand the context where the data is from and that is exactly what each game describes. The information stored in how the game ended and how long it took is potentially incredibly useful, as it allows us to make datasets of games that ended early or games that ended in a draw and see if we can determine patterns that cause these.

Nodes Table

The Nodes table serves as a reference table for all the nodes in the game. It tracks just the following:

- game_id: The ID of the game this node is from
- node_id: The ID of the node, the ID is from the logic used by the EVERGLADES game

Having a table is beneficial over just directly referring to the node ids in the events as in the future different maps could have different node amounts. Therefore we can select games for datasets based on the amount of nodes they had.

Groups Table

The Groups table serves as a reference table for all the groups in the game. It tracks the following:

- game_id: The ID of the game this group is from
- group_id: The ID of the group, the ID is from the logic used by the EVERGLADES game
- player_id: The ID of the player the group belongs to
- unit_type: The type of unit the group is composed of

This is useful as it allows us to evaluate how well agents performed with each type of unit they controlled during the game. We can also compare how two agents used their units more or less effectively.

Units Table

The Units table serves as a reference table for all the units in the game. It tracks the following:

- game_id: The ID of the game this unit is from
- unit_id: The ID of this unit, the ID is from the logic used by EVERGLADES game
- group_id: The ID of the group this unit belongs to

Looking at individual unit usage is unlikely to be useful compared to group based statistics, however it is still important to store information about units to get a full understanding of how combat occurs and what its effects over the course of a game.

Scores Table

The Scores table simply records the score of each player on each turn of a match. It tracks the following:

- game_id: The ID of the game this score is from
- turn_num: The turn number these scores are from
- p0_score: The score of player 0
- p1_score: The score of player 1

Its simplest use is just looking at how each agents score changes over the course of the game and determining how impactful those score changes actually are to the end result.

Combat Table

The Combat table is one of the most important tables in the database. It has an entry for each time each individual unit takes damage in combat and tracks the following:

- game_id: The ID of the game this combat took place in
- node_id: The ID of the node this combat took place from
- turn_num: The turn number this combat took place on
- group_id: The ID of the group the unit involved belongs to
- unit_id: The ID of the unit involved in combat
- health: The remaining health the unit has after combat

We track combat by unit because the primary changes in combat happen on a per unit basis. It is also far more space efficient to look at a per unit basis than make a table that tracks all possible units. This is because combat can potentially contain only small amounts on units fighting, in which case a combat entry would have largely duplicate entries for those who did not fight. We can use the entries in this field to track which player is taking more or less damage over the course of the game. We can also rebuild combat from the table and use it to determine the effectiveness of certain units in certain match ups.

Disband Table

The Disband table records when a group disbands over the course of the game. It tracks the following:

- game_id: The ID of the game this happened in
- group_id: The ID of the group that is disbanding
- player_id: The ID of the player that the group belongs to
- turn_num: The turn number the group disbanded on

This table is incredibly useful as it gives us data on when agents groups are dying. We can then use this information to backtrack from and observe the situations that led to the groups having to disband. We can also track if it is more effective to fully commit units and have them disband early due to casualties or pull back and regroup with them to fight again later.

Group_Knowledge Table

The Group_Knowledge table tracks the knowledge an agent has on the location and movement of his opponents units. It records the following:

- game_id: ID of the game it takes place in
- player_id: ID of the player who has this knowledge
- node: The ID of the node the group is currently on
- destination: The ID of the node the group is traveling to
- group_id: ID of the group that the knowledge is of
- turn_num: The turn number the knowledge is from
- unit_type: The type of unit the knowledge is of
- unit_count: The amount of units in the group
- in_transit: Whether the group is moving to a destination or stationary

This table tracks very important information on the knowledge the agents have of the game state. This is incredibly useful to us as it allows us to see how decision making is affected by the level of knowledge available to the agent. This table is done as a per group basis as that is the logic used by the EVERGLADES telemetry and all units in a group move as one.

Group_Move Table

The Group_Move table tracks the movement of each group over the course of the game. It tracks the following:

- game_id: ID of the game them movement happens in
- group_id: ID of the group that is moving
- turn_num: The turn number the movement is occurring on
- player_id: The ID of the player the group belongs to

- node: The node the group is starting on
- destination: The node the group is travelling to
- move_state: The state of movement the group is currently in

Tracking the movement of groups over the course of the game is very important. It is important as it allows us to observe patterns in movement around the map. We can use this both to test whether there are certain optimal starting moves as well as checking if it is possible to be moving too much and not capturing nodes.

Node_Control Table

The Node_Control table tracks the control levels of the nodes over the course of the game. It tracks the following:

- game_id: ID of the game this record is from
- node_id: ID of the node whose control is changing
- turm_num: The turn number the control is changing on
- controller: ID of the player who controls the node
- control_value: The amount of control being exerted on the node
- controlled: Whether the control_value exceeds a node-dependant threshold that determines if it is actually controlled

Tracking the control level of nodes over the course of the game is very important. It allows us to see how highly an agent values capturing control of nodes to achieve victory over outright annihilating their opponents units. It also lets us see how efficiently agents capture new territory and how well they are using their units to do so. If one agent is moving all their units along a single path while the other spreads out more that is critical information that this table will help us capture.

Node_Knowledge Table

The Node_Knowledge table tracks the knowledge the agents have of each node over the course of the game. It tracks the following:

- game_id: ID of the game this record is from
- node_id: ID of the node the knowledge is on
- player_id: ID of the player who has the knowledge
- turn_num: Turn number this knowledge is from
- knowledge: Knowledge level the player has of the node
- controller: ID of the player who controls the node
- controlled: Whether the control_value exceeds a node-dependant threshold that determines if it is actually controlled

Tracking the node knowledge of the agents is incredibly important. The more information an agent has access to should lead to them making better and more informed decisions. Therefore this table will allow for us to track how much knowledge agents accrue over the course of the game. This can be used to determine if playing in a way that maximizes the amount of information will improve win rate for the agents. Other uses of this table would be to see if agents go after nodes that have been previously captured by the opponent in an effort to get them under their control.

4.2.2 Importing Data

As discussed in section 3.4.2 data from the EVERGLADES is stored in the following match telemetry .csv files:

- `Telem_GAME_Scores`
- `Telem_GROUP_CombatUpdate`
- `Telem_GROUP_Disband`
- `Telem_GROUP_Initilization`
- `Telem_GROUP_Knowledge`
- `Telem_GROUP_MoveUpdate`
- `Telem_NODE_ControlUpdate`

- `Telem_NODE_Knowledge`
- `Telem_PLAYER_Tags`

One of the primary motivations for our database design was to make it as easy as possible to import the data from these files. Due to our design importing the data is incredibly easy as many of the tables line up with the telemetry files.

Therefore we can make a python script that reads through a directory of the csv files and fills the information into predetermined SQL queries. Some examples of how some of the telemetry files translate to the tables are:

- `Telem_PLAYER_Tags` → Scripts Table
- `Telem_GAME_Scores` → Scores Table
- `Telem_GROUP_Disband` → Disband Table
- `Telem_GROUP_CombatUpdate` → Combat Table
- `Telem_GROUP_Knowledge` → Group_Knowledge Table
- `Telem_GROUP_Initilization` → Units Table & Groups Table
- `Telem_GROUP_MoveUpdate` → Group_Move Table
- `Telem_NODE_ControlUpdate` → Node_Control Table
- `Telem_NODE_Knowledge` → Node_Knowledge Table

This is also a great opportunity for collaboration with the other EVERGLADES group, by making it easy for us to accept any telemetry from the game, it will be very simple for them to share their telemetry with us. We are unsure exactly how we are going to do this, but one idea is to set up a website where they can upload their telemetry files and it will then import the data into the database and do some baseline analysis on their agents performance.

4.3 Key Findings

4.3.1 First Group Disband/Unit Type Evaluation

Overall Hypothesis:
The player with the first group disband has a direct effect on the outcome of the game and the type of units of that group has a strong effect on the win/loss outcome.

Overall Summary:
The goals of the following models are to find any correlation between the first group lost and its unit type with the affect it has on winning games. In addition, different agents will be introduced to see if they affect the win rate of the games and accuracy scores of the models.

Model #1

Feature Variables:
Player 1 Score
Player 2 Score
Win Type
Turn Group Lost Occurred
Unit Type of Group Lost
Player With First Group Lost

Target Variable:
Winner of the Game

Agents Used:
Player 1: Random Action Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest Trees
Support Vector Machines

Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on the each other as well as on the target.

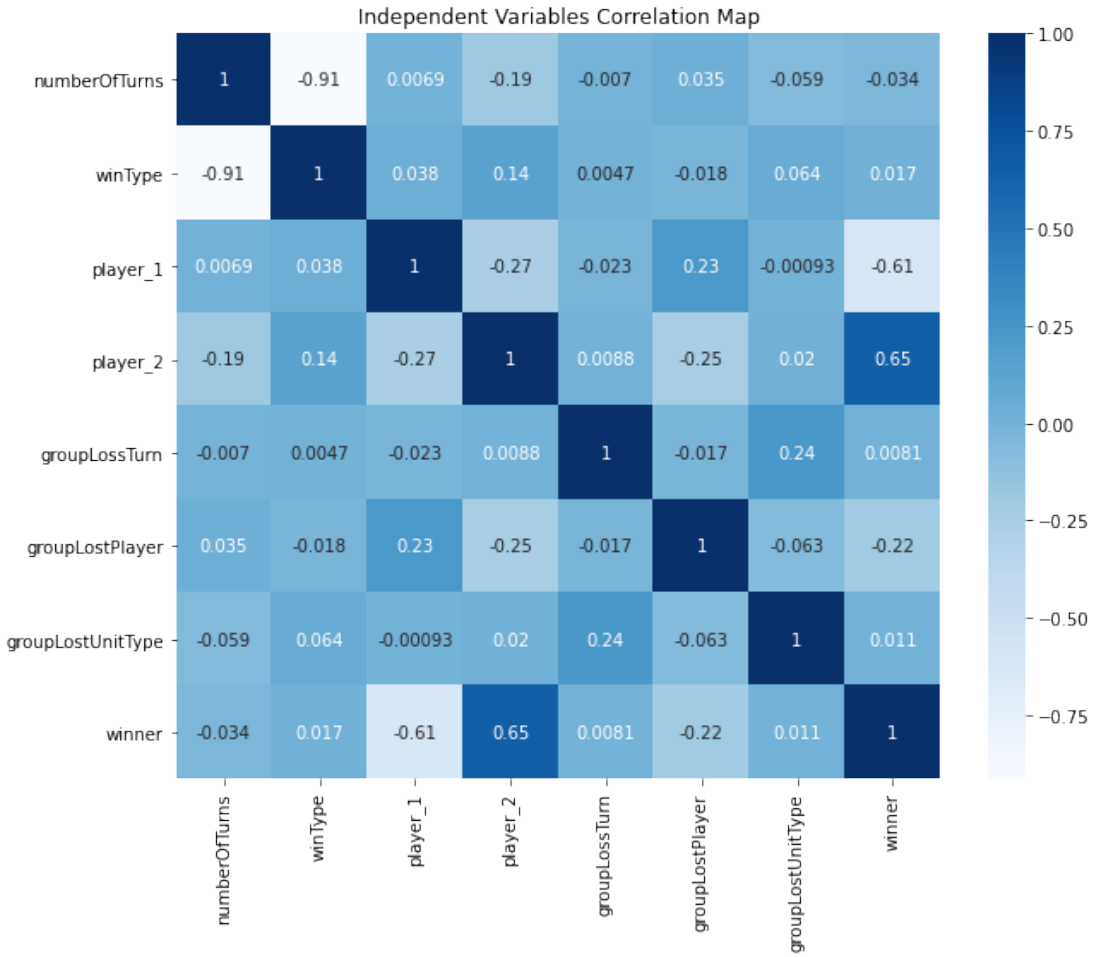


Figure 4.3: Independent Variable Correlation Map

Given these metrics, Number of Turns, Turn Group Lost Occurred, and Unit Type of Group Lost has the least impact on the Winner and will likely be omitted in future iterations.

Base Model Evaluations:
A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C = 1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

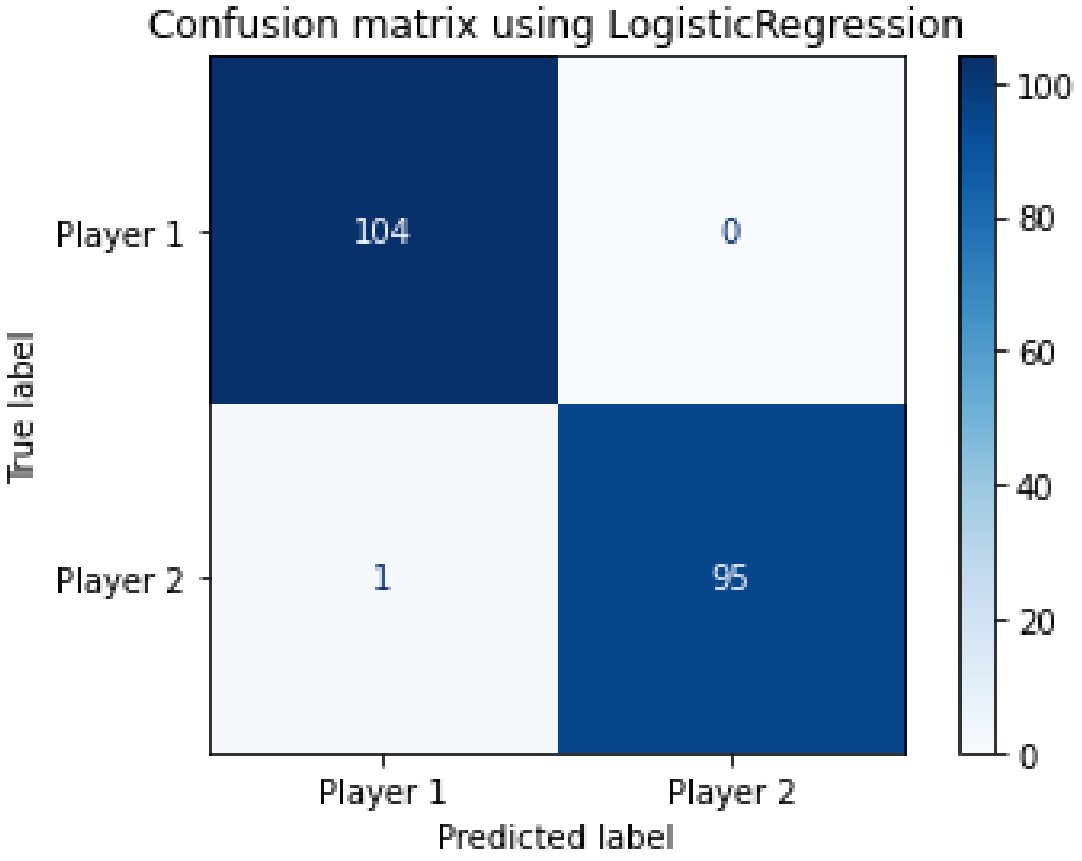
	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9950	0.9650	0.9300	0.9912
F1 Score	0.9952	0.9669	0.9330	0.9916

Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9900	0.9700	0.9450	0.9858
F1 Score	0.9905	0.9720	0.9474	0.9850

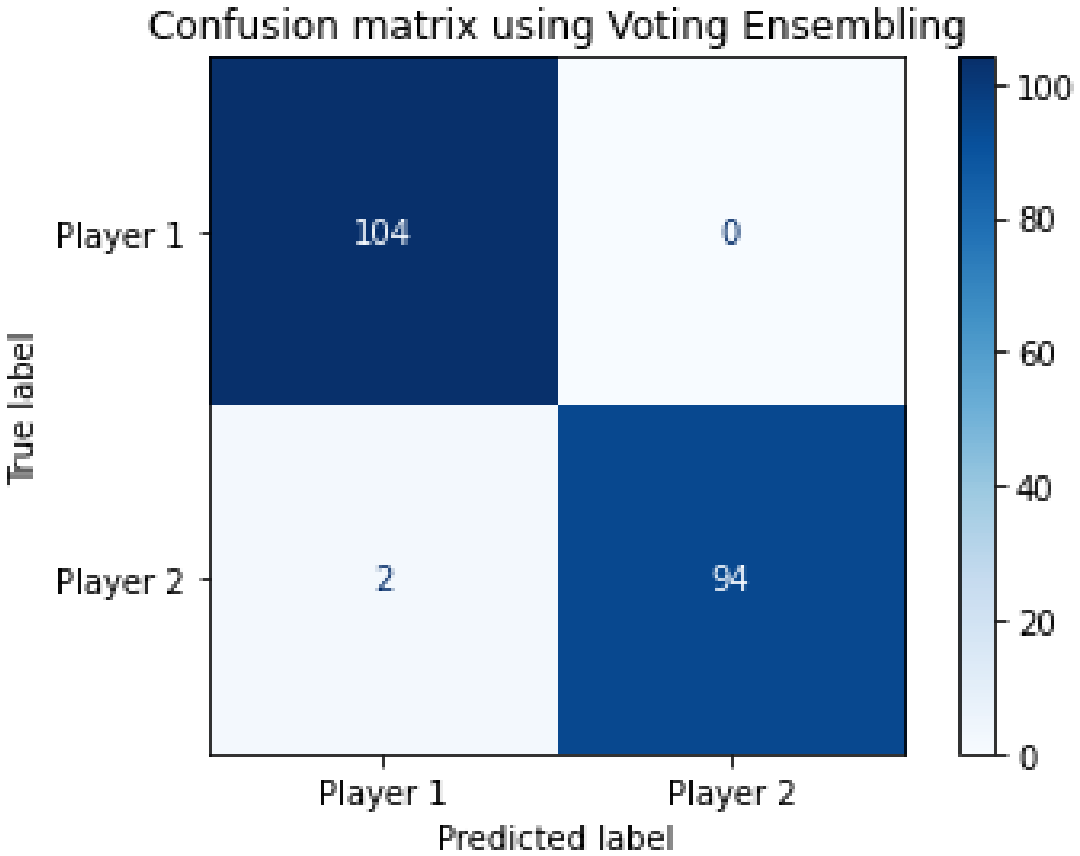
Based on the results, Logistic Regression had the highest F1-Score and Accuracy score for both the training set and test set. We will use Grid Search CV to tune the hyperparameters of the trained Logistic Regresion model to increase the accuracy scores.

Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value in Grid Search CV, C value of 5 (originally 1) produced the best result and yields the following scores when evaluating the accuracy and F1 Score for Logistic Regression, resulting in the following statistics:



Logistic Regression Accuracy Scores:
Classification Accuracy: 0.9950
Classification Error: 0.0050
Precision: 0.9905
Recall: 1.0000
F1-Score: 0.9952

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 0.9900
Classification Error: 0.0050
Precision: 0.9811
Recall: 1.0000
F1-Score: 0.9905

Based on the two models, Logistic Regression seems to be a better model, having an accuracy of 99.5%. The Recall, or how often the model is correct when Player 2 wins is 100% and the Precision, or how often the model is correct when Player 1 wins, is 99.05%, yielding an F1 score of 99.52%.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both Logistic Regression and Voting Ensemble produces the following learning curves:

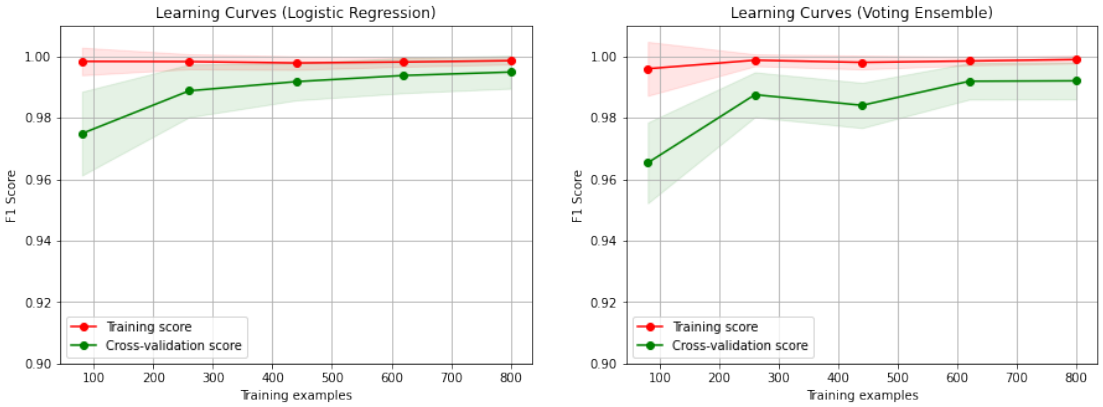


Figure 4.4: Learning Curves for Logistic Regression and Voting Ensemble

Although both curves seem to produce similar desired F1 Scores, the straight line produced by the training score seems to indicate that the models are highly over-fitting. Also, the learning curve representing the Logistic Regression model has converged, indicating that the model would not benefit from having more training examples.

Conclusion - Model #1:
The results from the models show that there is some correlation between the independent features and the dependent label. Logistic Regression seems to be the better model, as it produces slightly higher accuracy scores, although the learning curves indicate that there may be a bit of over-fitting happening within the model. That being said, there is still some reasonable argument that, if tested correctly or with different agents, the first player to have a group lost could an impact on the outcome of the game. The next model will attempt to refine the above model by removing low correlation features.

Model #2

Feature Variables:
Player 1 Score
Player 2 Score
Win Type
Player With First Group Lost

Target Variable:
Winner of the Game

Agents Used:
Player 1: Random Action Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest Trees
Support Vector Machines
Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on the each other as well as on the target.

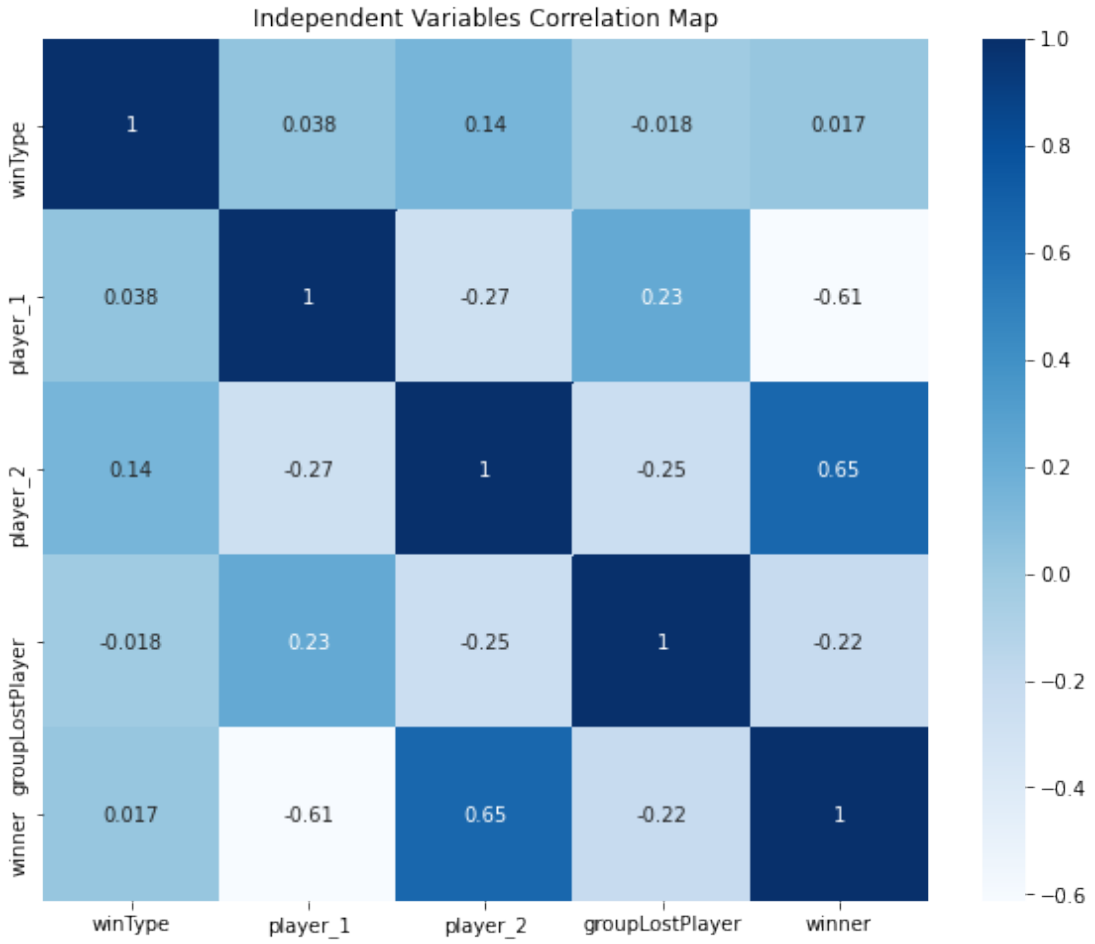


Figure 4.5: Independent Variable Correlation Map

Given these metrics, Win Type has the least impact on the Winner and will likely be omitted in future iterations.

Base Model Evaluations:

A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C=1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

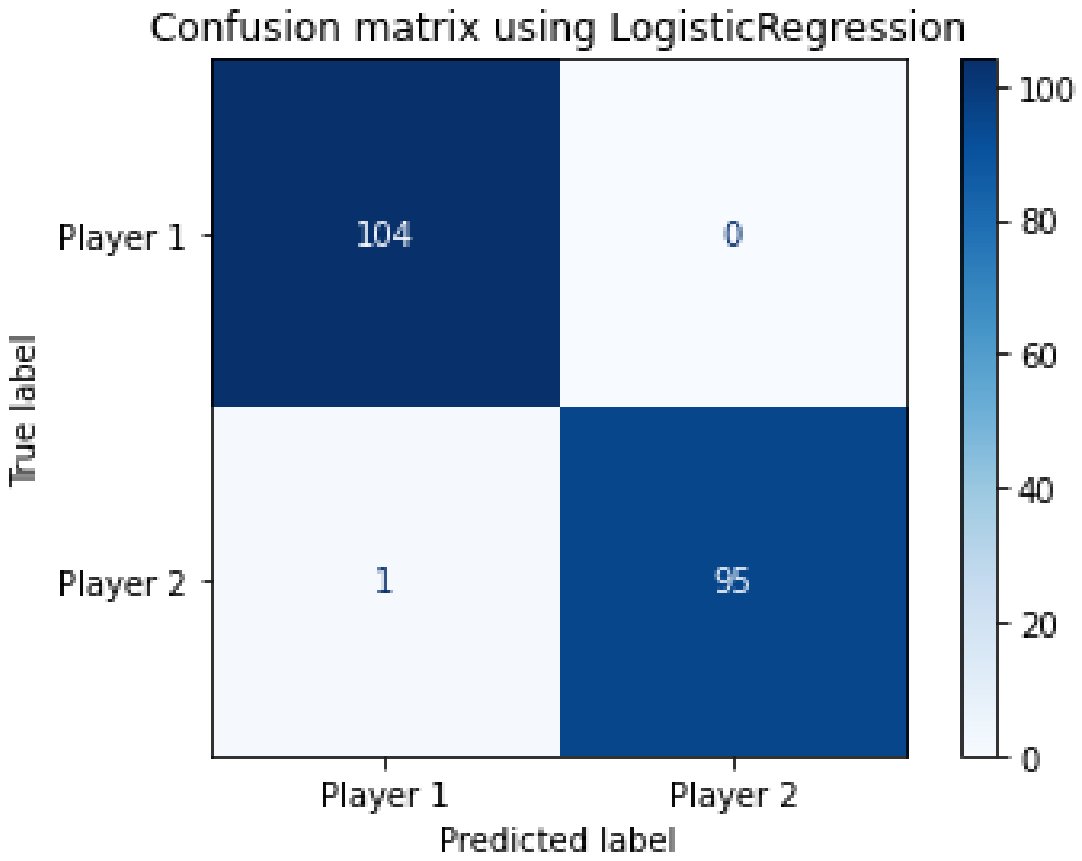
	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9975	0.9850	0.9475	0.9925
F1 Score	0.9976	0.9857	0.9504	0.9928

Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9950	0.9900	0.9550	0.9900
F1 Score	0.9952	0.9905	0.9573	0.9905

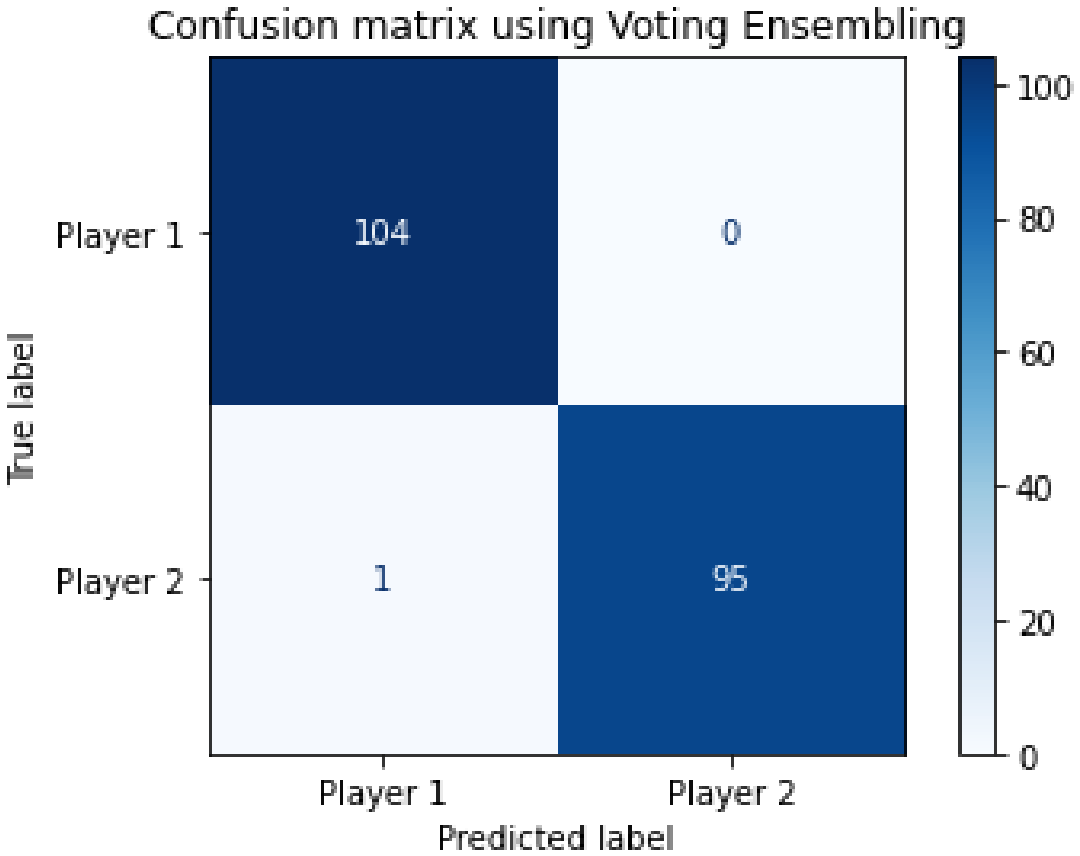
Based on the results, Logistic Regression has the highest F1-Score and Accuracy score for both the training set and test set. We will use Grid Search CV to tune the hyperparameters of the trained Logistic Regression model to. increase the accuracy scores.

Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value in Grid Search CV, C value of 5 (originally 1) produced the best result when evaluating the accuracy and F1 Score for Logistic Regression, resulting in the following statistics:



Logistic Regression Accuracy Scores:
Classification Accuracy: 0.9950
Classification Error: 0.0050
Precision: 0.9905
Recall: 1.0000
F1-Score: 0.9952

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 0.9900
Classification Error: 0.0050
Precision: 0.9811
Recall: 1.0000
F1-Score: 0.9905

Based on the two models, Logistic Regression seems to be a better model, having the same results as Model #1 with an accuracy score of 99.5% and F1-Score of 99.52%. It seems that changing the features used did not have an impact on the accuracy of the model.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both Logistic Regression and Voting Ensemble produces the following learning curves:

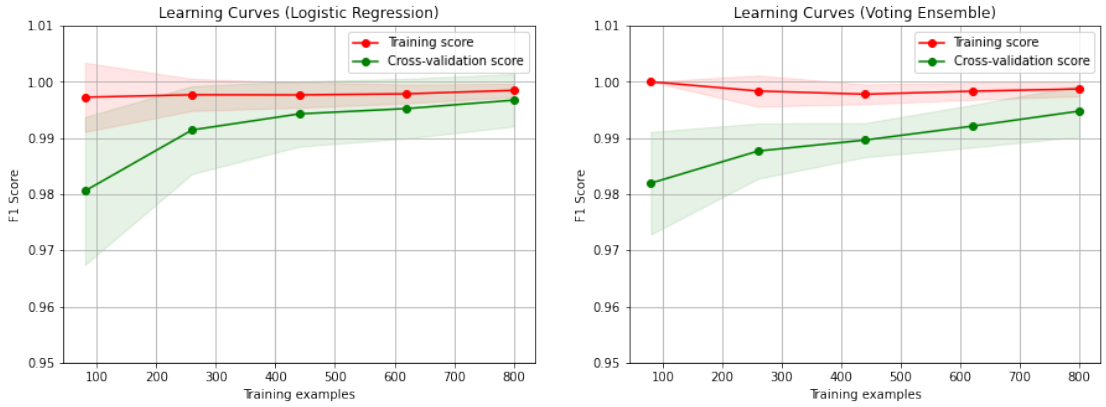


Figure 4.6: Learning Curves for Logistic Regression and Voting Ensemble

Although both curves seem to be producing similar desired F1 Scores, the straight line at near 100% produced by the training scores seems to indicate that the models are over-fitting. Also, the learning curve representing the Voting Ensemble model cross validation scores seems to be improving, as it is decreasing the variance as the number of training examples increase. The Voting Ensemble model may benefit from having more training examples.

Conclusion - Model #2:
The results from the models show that there is some correlation between the features and target variables. Logistic Regression seems to be the better model, as it produces slightly higher accuracy scores than the other models, although the learning curves indicate that there may be over-fitting happening within the model. There seems to be no impact on the accuracy scores with the removal of the features with low correlation to the winner, but there were some improvements with all the models evaluated, which shows that this model is better than Model #1. That being said, there is still some reasonable argument that, if tested correctly, the first player to have a group disband could have an impact on the outcome of the game. The next model will attempt to refine with above model using a different set of AI agents with the original set of variables to see if the

agent's behavior cause a difference in the results of if the first player to have a group disband and the unit type of that group.

Model #3

Feature Variables:
Player 1 Score
Player 2 Score
Win Type
Turn Group Lost Occurred
Unit Type of Group Lost
Player With First Group Lost

Target Variable:
Winner of the Game

Agents Used:
Player 1: Base Rush Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest
Support Vector Machines
Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on the each other as well as on the target.

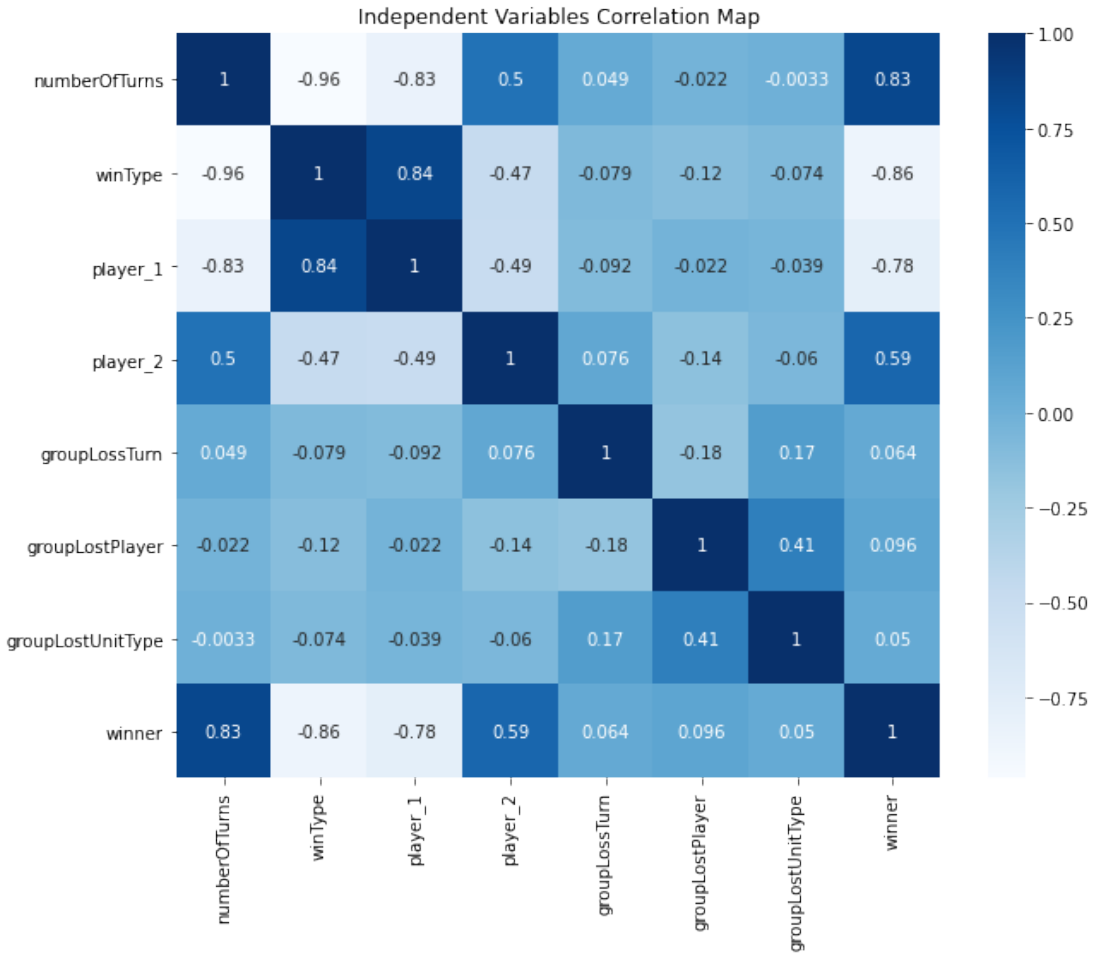


Figure 4.7: Independent Variable Correlation Map

Given these metrics, Number of Turns, Turn Group Lost Occurred, and Unit Type of Group Lost has the least impact on the Winner and will likely be omitted in future iterations.

Base Model Evaluations:

A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C=1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9992	0.9922	0.9797	0.9984
F1 Score	0.9996	0.9957	0.9887	0.9991

Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9983	0.9906	0.9906	0.9969
F1 Score	0.9969	0.9949	0.9948	0.9983

Based on the results, SVM had the highest F1-Score and Accuracy score for both the training set and test set. We will use Grid Search CV to tune the hyperparameters of the trained SVM model to increase the accuracy scores.

Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value in Grid Search CV, C value of 100 (originally 1) produced the best result when evaluating the accuracy and F1 Score for SVM, resulting in the following statistics:

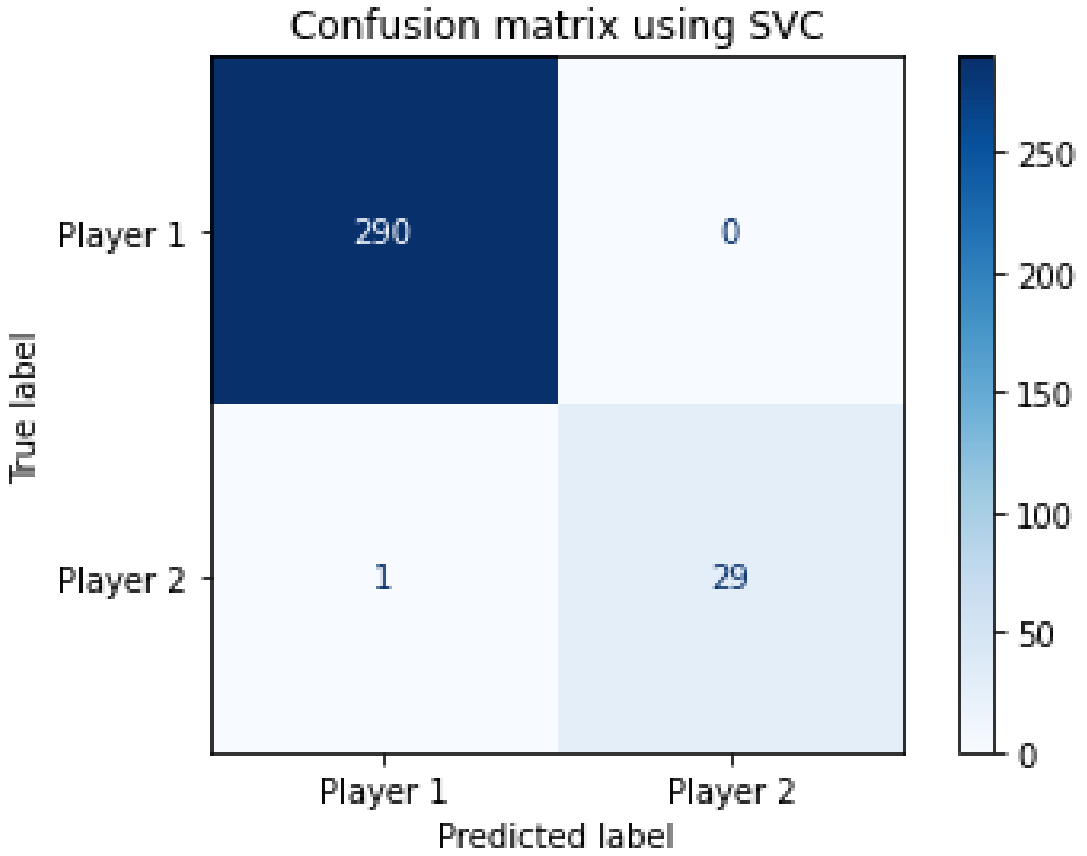
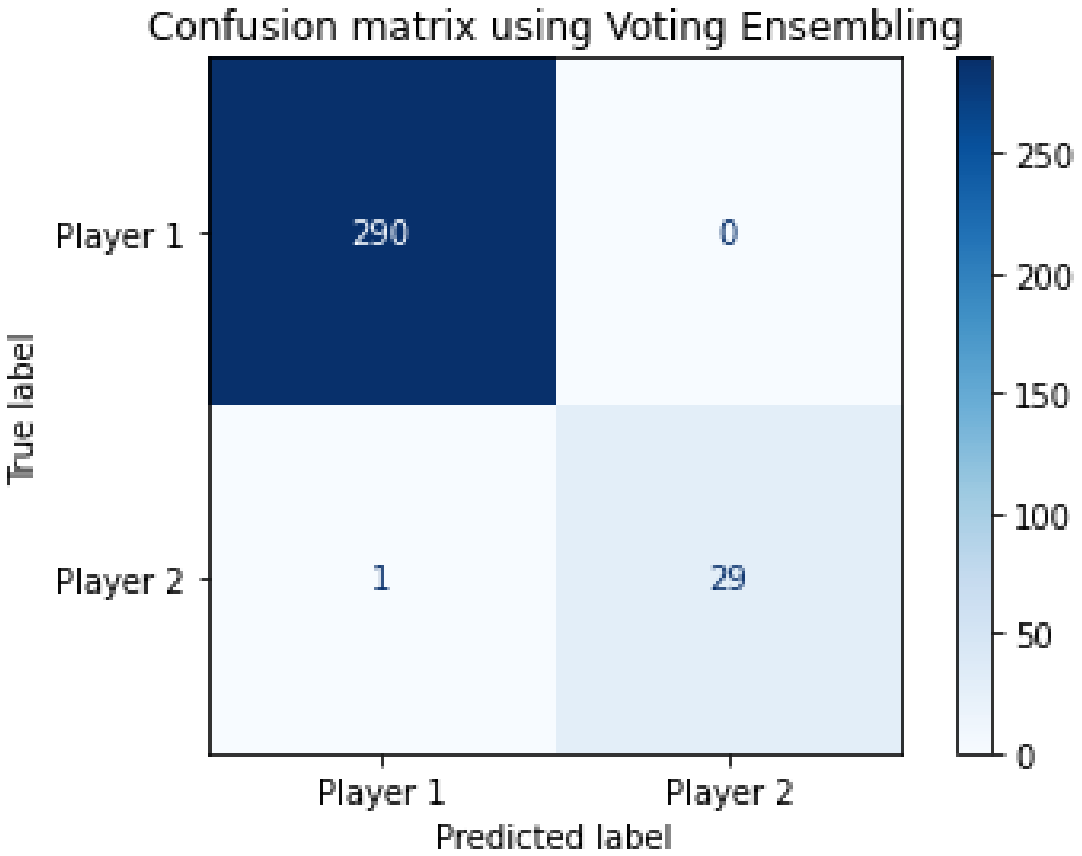


Figure 4.8: *SVC (Support Vector Classifier)*, another name for SVM

SVM Accuracy Scores:
Classification Accuracy: 0.9969
Classification Error: 0.0031
Precision: 0.9966
Recall: 1.0000
F1-Score: 0.9983

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 0.9900
Classification Error: 0.0100
Precision: 0.9811
Recall: 1.0000
F1-Score: 0.9905

Based on the two models, SVM seems to be a better model than Voting Ensemble, having an accuracy of 99.69%. The Recall, or how often the model is correct when Player 2 wins is 100% and the Precision, or how often the model is correct when Player 1 wins, is 99.05%, yielding an F1 score of 99.52%.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both SVM and Voting Ensemble produces the following learning curves:

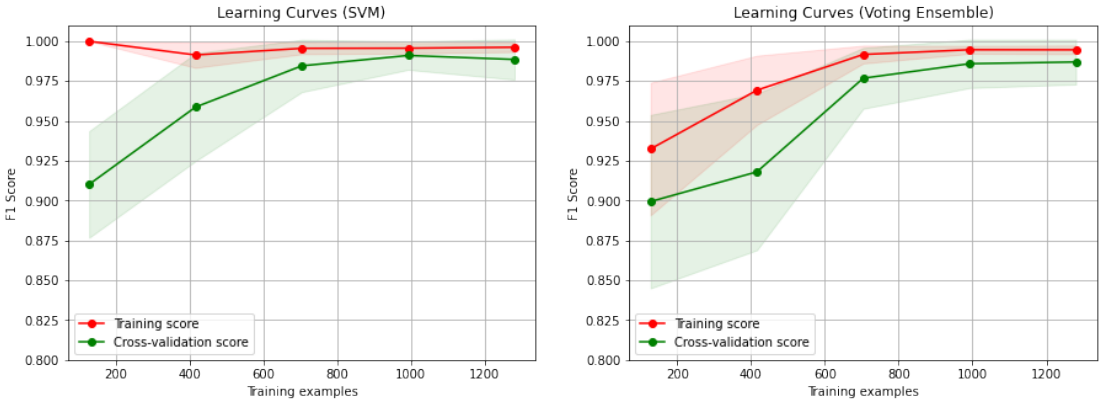


Figure 4.9: Learning Curves for SVM and Voting Ensemble

Both curves seem to be producing similar desired F1 Scores, although there may be over-fitting with the Support Vector Machine model, as it has an F1-Score near 100% regardless of the number of training examples. The Voting Ensemble may be the better model, as it seems to generalize the data more, although the model does high bias.

Conclusion - Model #3:
The results from the models show that there is some correlation between the feature and target variables. The Support Vector Machine model produced higher results, but after looking at the Learning Curve, may not be the best model compared to Voting Ensemble, as there is a bit of over-fitting on the model. That being said, using the Base Rush Agent not only increase the win rate for Player 1 (from 52.10% to 90.69%), but also we were able to increase the accuracy scores of the model by having a more stable agent. There is still some reasonable argument that the if tested correctly, the first player to have a group disband could have an impact on the outcome of the game. The next model will attempt to refine the above model by removing low correlation features.

Model #4

Independent Variables:
Player 1 Score
Player 2 Score
Win Type
Turn Group Lost Occurred
Unit Type of Group Lost
Player With First Group Lost

Dependent Variable:
Winner of the Game

Agents Used:
Player 1: Base Rush Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest
Support Vector Machines
Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on the each other as well as on the target.

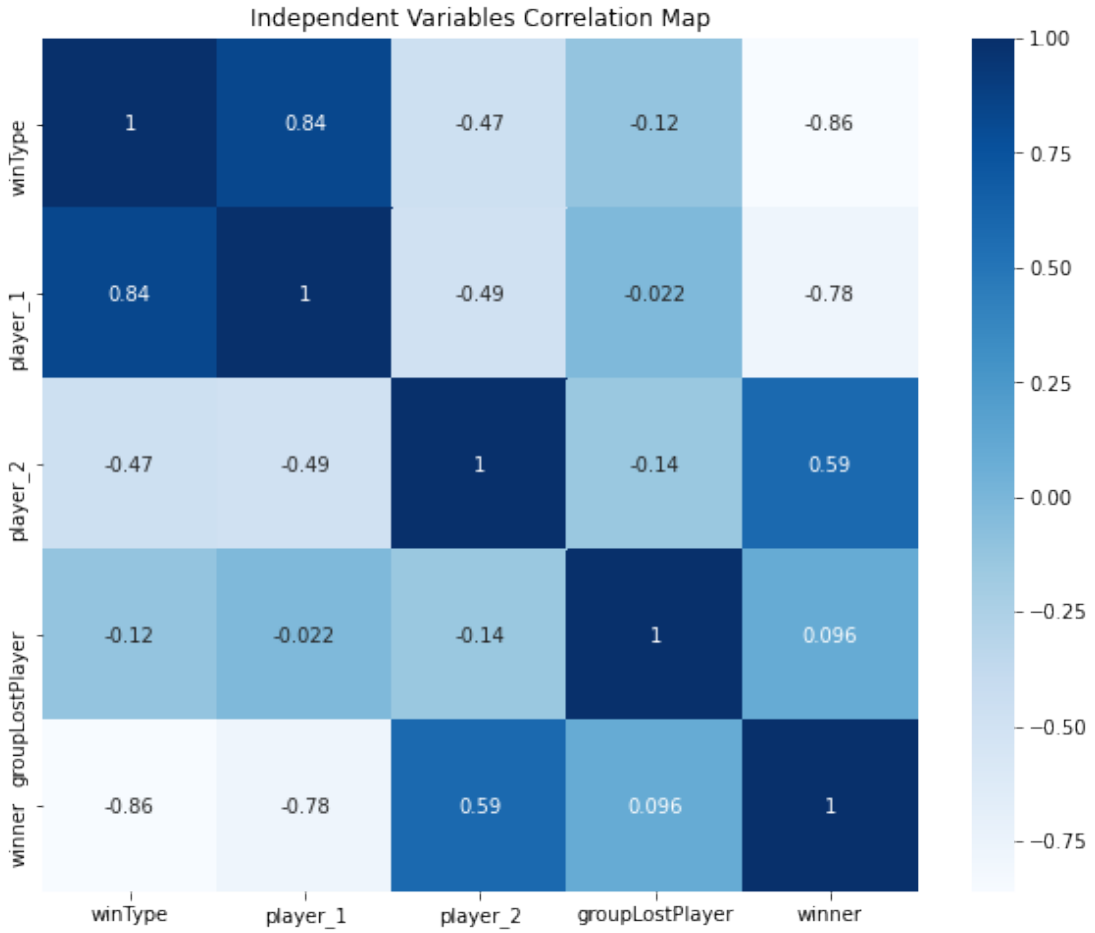


Figure 4.10: Independent Variable Correlation Map

Given these metrics, Number of Turns, Turn Group Lost Occurred, and Unit Type of Group Lost has the least impact on the Winner and will likely be omitted in future iterations.

Base Model Evaluations:
A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C=1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

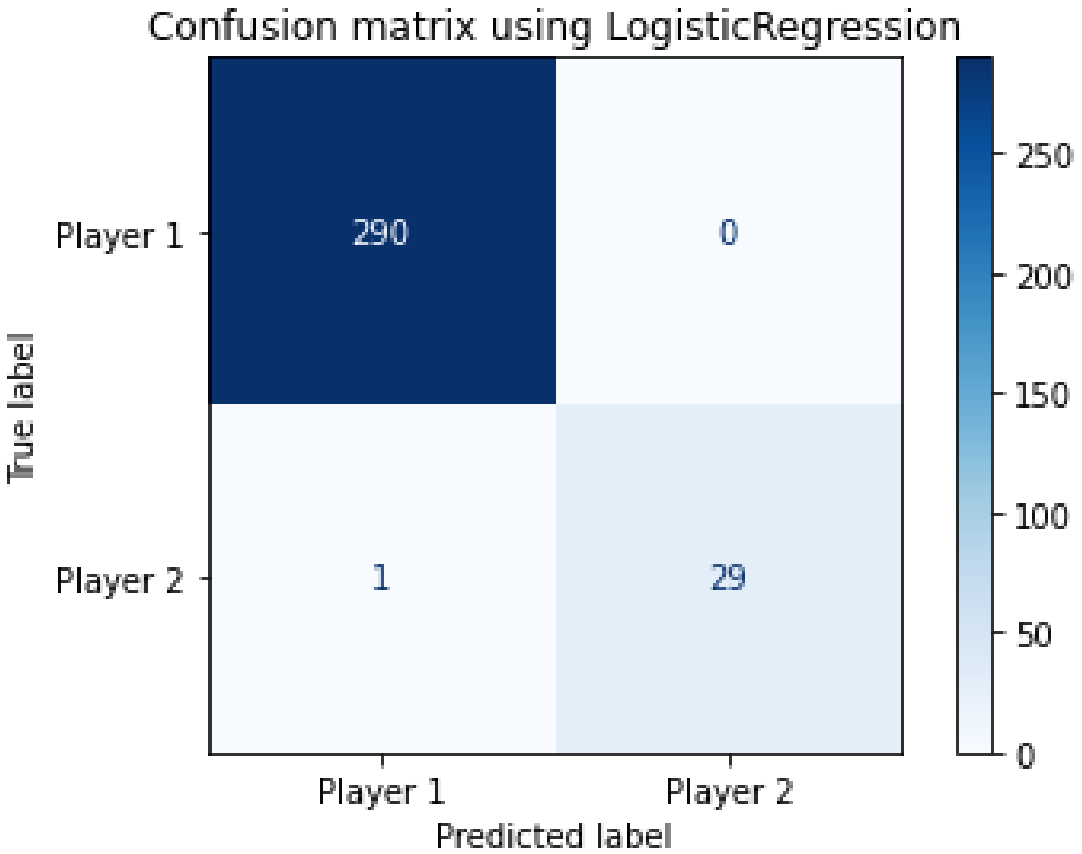
	Logistic Regression	KNN	Random Forest	SVM
Accuracy	1.0000	0.9977	0.9898	0.9984
F1 Score	1.0000	0.9987	0.9944	0.9991

Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9969	0.9969	0.9844	0.9969
F1 Score	0.9983	0.9983	0.9913	0.9983

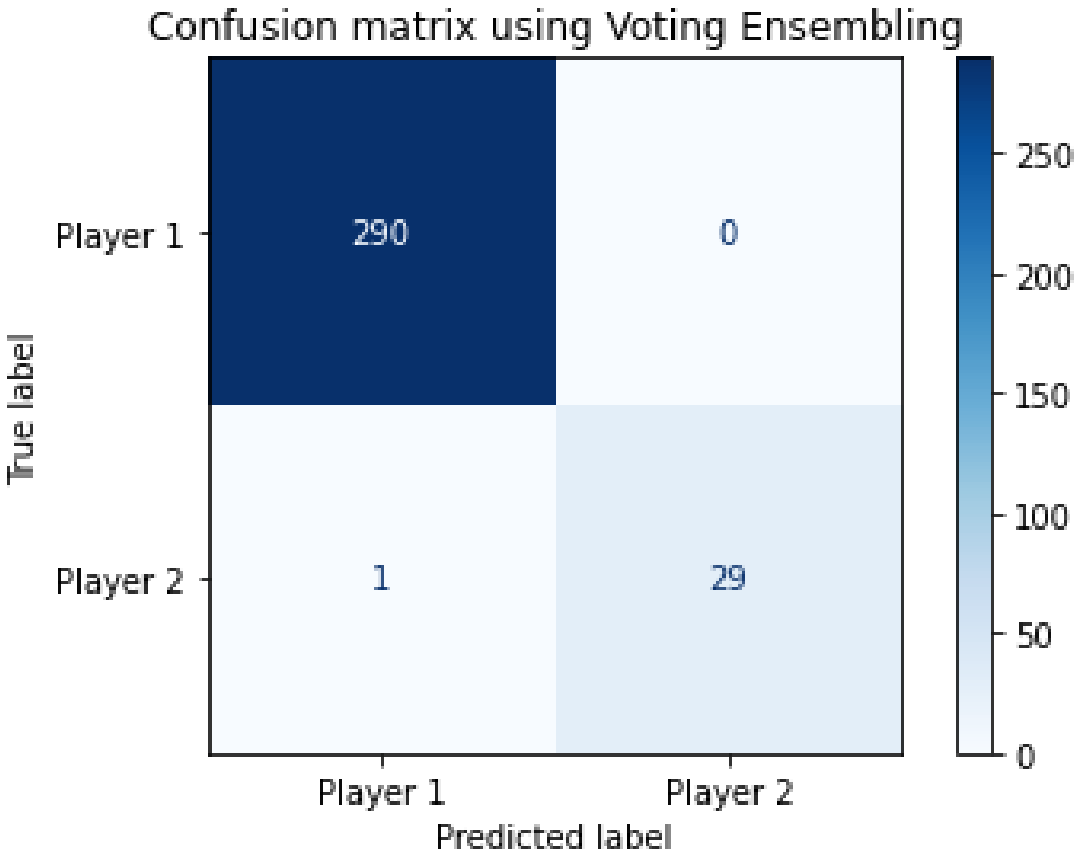
Removing the low correlation features resulted in Logistic Regression having the highest F1-Score for the training set and tied for highest with the test set. We will use Grid Search CV to tune the hyperparameters of the trained Logistic Regression model to increase the accuracy scores.

Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value in Grid Search CV, C value of 5 (originally 1) and L2 Regularization (originally L1 Regularization) produced the best result and when evaluating the accuracy and F1 Score for Logistic Regression, resulting in the following statistics:



Logistic Regression Accuracy Scores:
Classification Accuracy: 0.9969
Classification Error: 0.0031
Precision: 0.9966
Recall: 1.0000
F1-Score: 0.9983

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 0.9900
Classification Error: 0.0100
Precision: 0.9811
Recall: 1.0000
F1-Score: 0.9905

Based on the two models, Logistic Regression seems to be a better model, having an accuracy of 99.69%, the same results from Model #3 for SVM. The Recall, or how often the model is correct when Player 2 wins is 100% and the Precision, or how often the model is correct when Player 1 wins, is 99.05%, yielding an F1 score of 99.52%.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both Logistic Regression and Voting Ensemble produces the following learning curves:

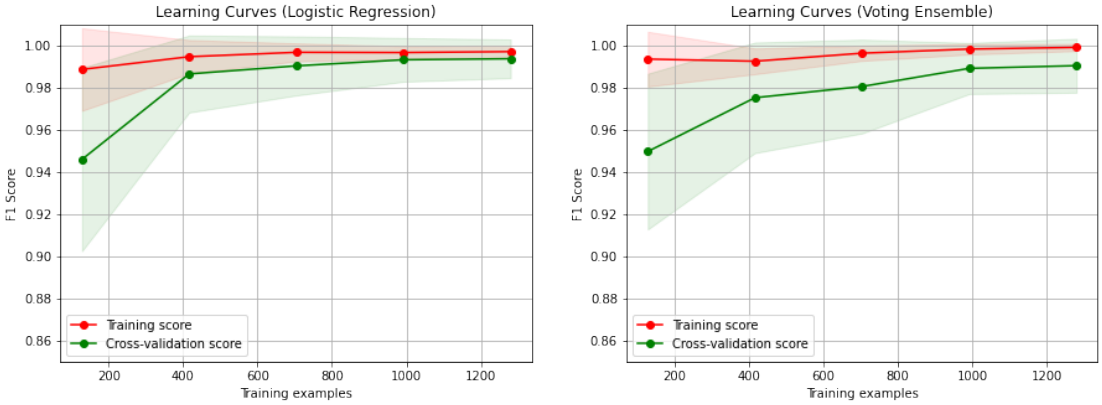


Figure 4.11: Learning Curves for Logistic Regression and Voting Ensemble

Both curves increased their scores from Model #3 and seem to be learning the data better. There still seems to be a bit of high bias and over fitting as the models are producing high F1-Scores with small amounts of training examples. The Voting Ensemble model may benefit from more training examples.

Conclusion - Model #4:
The results from the models show that there is some correlation between the independent and dependent variables. Logistic Regression produced 0.31% higher results than the model selected in Model #3 and produced higher results in the Learning Curve. Even so, there still may be a bit of over-fitting, as the Learning Curve shows high bias in the model. That being said, we were able to increase the accuracy scores of the model by removing independent variables with low correlation. There is still some reasonable argument that the if tested correctly, the first player to have a group disband could have an impact on the outcome of the game.

First Group Disband/ Unit Type Evaluation:
Based on the above models, one would conclude that losing the first or second

unit in a game is a strong indicator of who wins or loses. From the 1600 games of data, the above models were built from, Player 1 won 73% of the games when the first group lost was a group of Striker units. When evaluating the player that lost the first group in the game, Player 2 only won 7% of their games if they lost the first group and Player 1 of won 40% of the games when they lost the first group.

Analyzing the type of units lost was surprisingly ineffective. There were some expectations that the first group lost would have a strong correlation to win-loss ratio, but the models suggested otherwise. But one important piece of information that can be extrapolated from the data is the weakness of the Striker unit. The Strikers group was the first group type destroyed 86% of the time when Random Action Agents played against each other and 80% of the time when Random Action Agent was matched against the Base Rush Agent, which makes sense due to its defense disadvantage but mitigating this could be a reasonable strategy for a future agent. This data will be insightful when developing scripts and these models will be a great tool for building the Real Time Engine. One primary potential issue with this data is the type of agents used in the first two data. The agents were the random action agents provided with the project, which could be a primary issue with the Strikers. When placing the Random Action Agent against the Base Rush Agent, the Striker unit as the first unit lost did decrease by 6%, but with the Striker moving to a node with no support, it will likely be destroyed. Because of issues like these, smarter agents will need to be analyzed using similar models in this experiment.

4.3.2 First Player Tank Disband

Overall Hypothesis:
The player that has the first tank disband has a direct effect on the outcome of the game.

Overall Summary:
The goals of the following models is to find any correlation between the first player to have a tank disband and the effect it has on winning games.

Model #1

Independent Variables:
Player 1 Score
Player 2 Score
Win Type
Turn Count
Player of First Tank Disband

Dependent Variable:
Winner of the Game

Agents Used:
Player 1: Random Action Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest Trees
Support Vector Machines
Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on the each other as well as on the target.

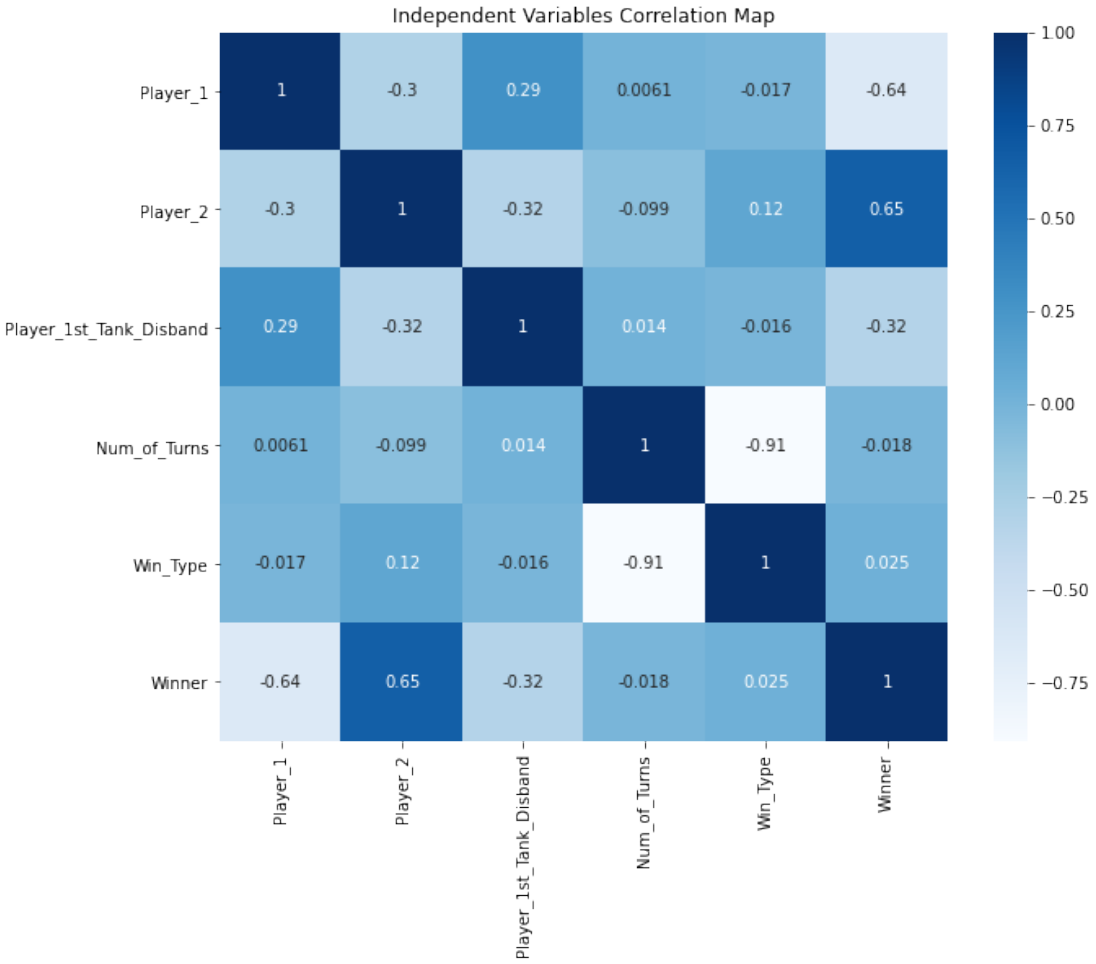


Figure 4.12: Independent Variable Correlation Map

Given these metrics, Number of Turns and Win Type has the least impact on the Winner and will likely be omitted in future iterations.

Base Model Evaluations:

A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C=1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

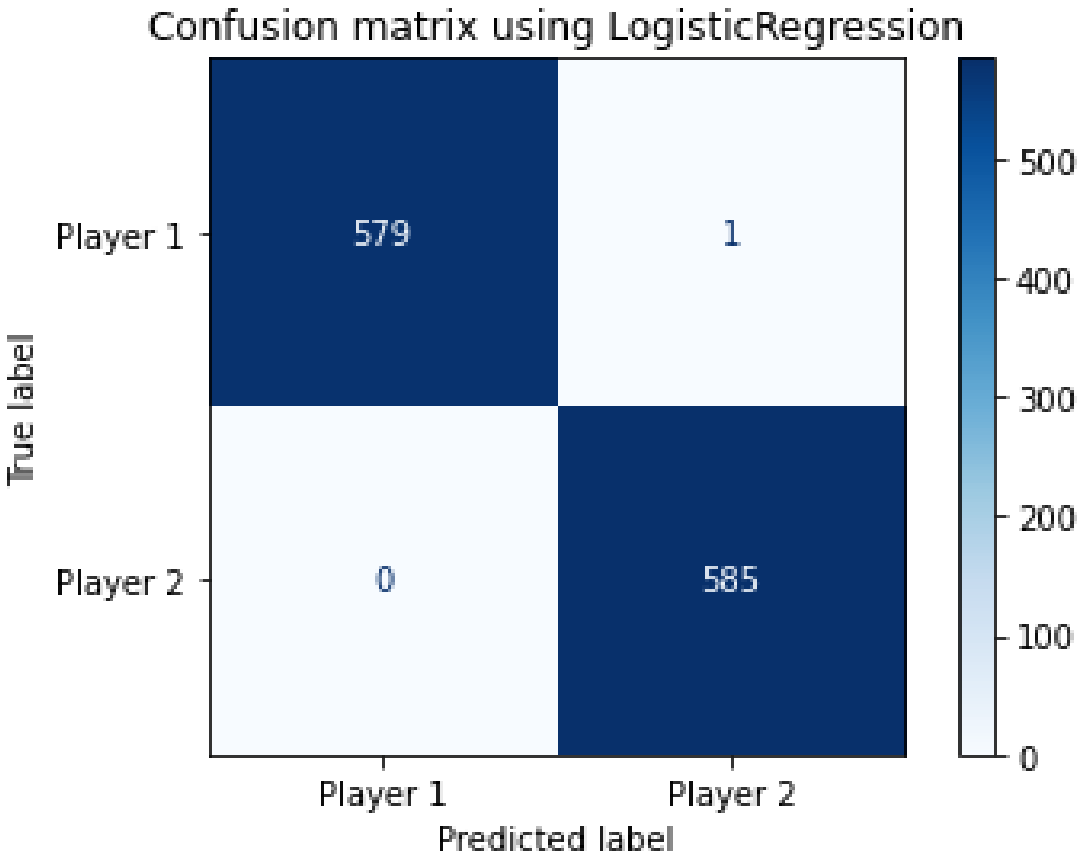
	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9994	0.9944	0.9375	0.9912
F1 Score	0.9994	0.9944	0.9964	0.9963

Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	1.0000	0.9966	0.9227	0.9983
F1 Score	1.0000	0.9966	0.9223	0.9983

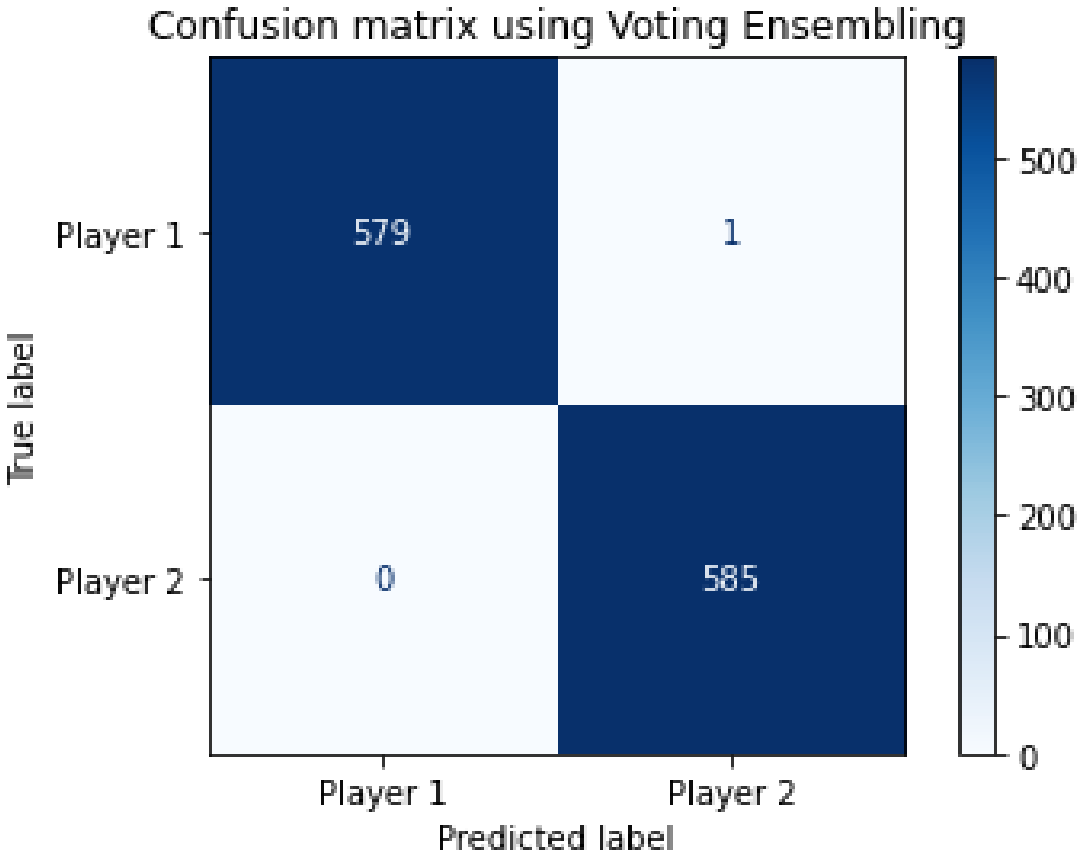
Based on the results, Logistic Regression had the highest F1-Score and Accuracy score for both the training set and test set. We will use Grid Search CV to tune the hyperparameters of the trained Logistic Regression model to increase the accuracy scores.

Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value in Grid Search CV, solver, which tries to find the parameter that minimizes the cost function, was tuned to 'liblinear', which produced the best result when evaluating the accuracy and F1 Score for Logistic Regression, resulting in the following statistics:



Logistic Regression Accuracy Scores:
Classification Accuracy: 0.9991
Classification Error: 0.0009
Precision: 1.0000
Recall: 0.9983
F1-Score: 0.9991

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 0.9991
Classification Error: 0.0009
Precision: 1.0000
Recall: 0.9983
F1-Score: 0.9991

Based on the two models, the two models are producing the same results, having an accuracy of 99.91%. The Recall, or how often the model is correct when Player 2 wins is 99.83% for both models and the Precision, or how often the model is correct when Player 1 wins, is 100% for both, yielding an F1 score of 99.91%.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both Logistic Regression and Voting Ensemble produces the following learning curves:

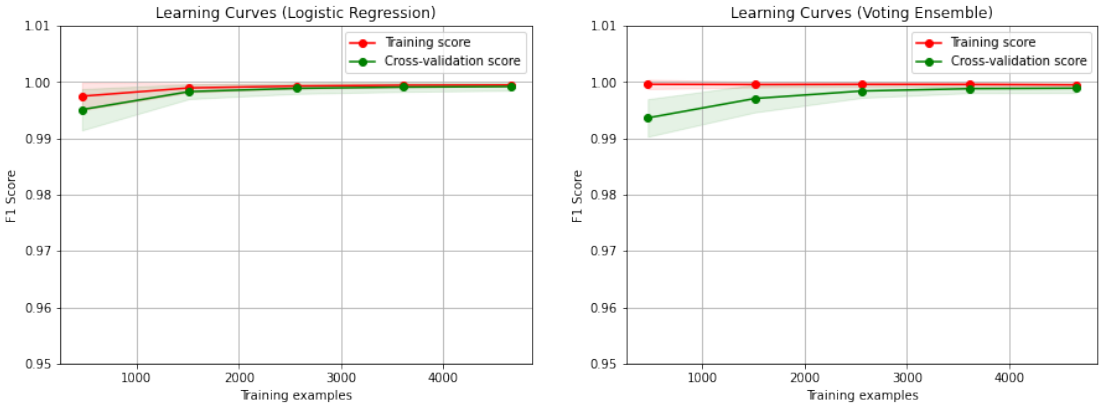


Figure 4.13: Learning Curves for Logistic Regression and Voting Ensemble

Both curves seem to be producing similar desired F1 Scores, though the Voting Ensemble model is over-fitting a little more than the Logistic Regression curve on the training set. Also, neither curve would benefit from more training examples, as they have both converged.

Conclusion - Model #1:
The results from the models show that there is some correlation between the independent features and the dependent label. Logistic Regression seems to be the better model, as it produces slightly higher accuracy scores, although the learning curves indicate that there may be a bit of over-fitting happening within the model. That being said, there is still some reasonable argument that the if tested correctly or with different agents, the first player to have a group disband could an impact on the outcome of the game. The next model will attempt to refine with above model by removing the features that have low correlation with the target variables.

Model #2

Independent Variables:
Player 1 Score
Player 2 Score
Player of First Tank Disband

Dependent Variable:
Winner of the Game

Agents Used:
Player 1: Random Action Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest Trees
Support Vector Machines
Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on the each other as well as on the target.

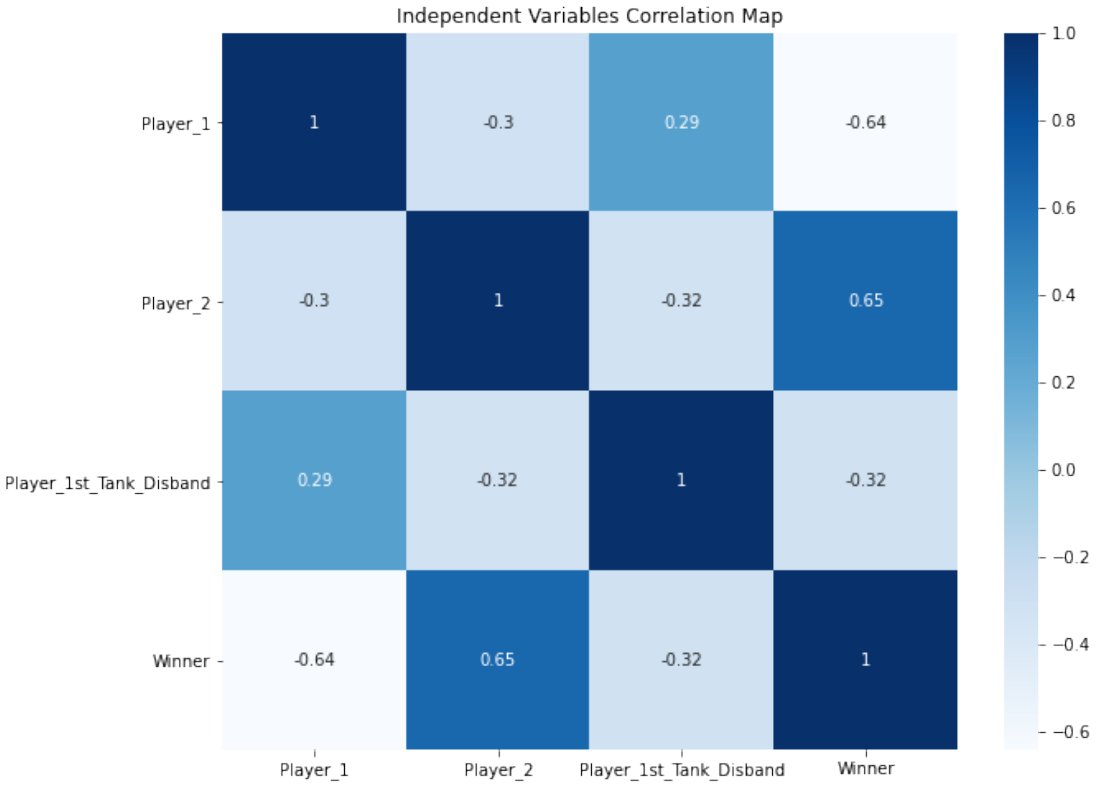


Figure 4.14: Independent Variable Correlation Map

Given these metrics, Number of Turns and Win Type has the least impact on the Winner and will likely be omitted in future iterations.

Base Model Evaluations:

A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C=1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9994	0.9948	0.9390	0.9961
F1 Score	0.9994	0.9948	0.9390	0.9961

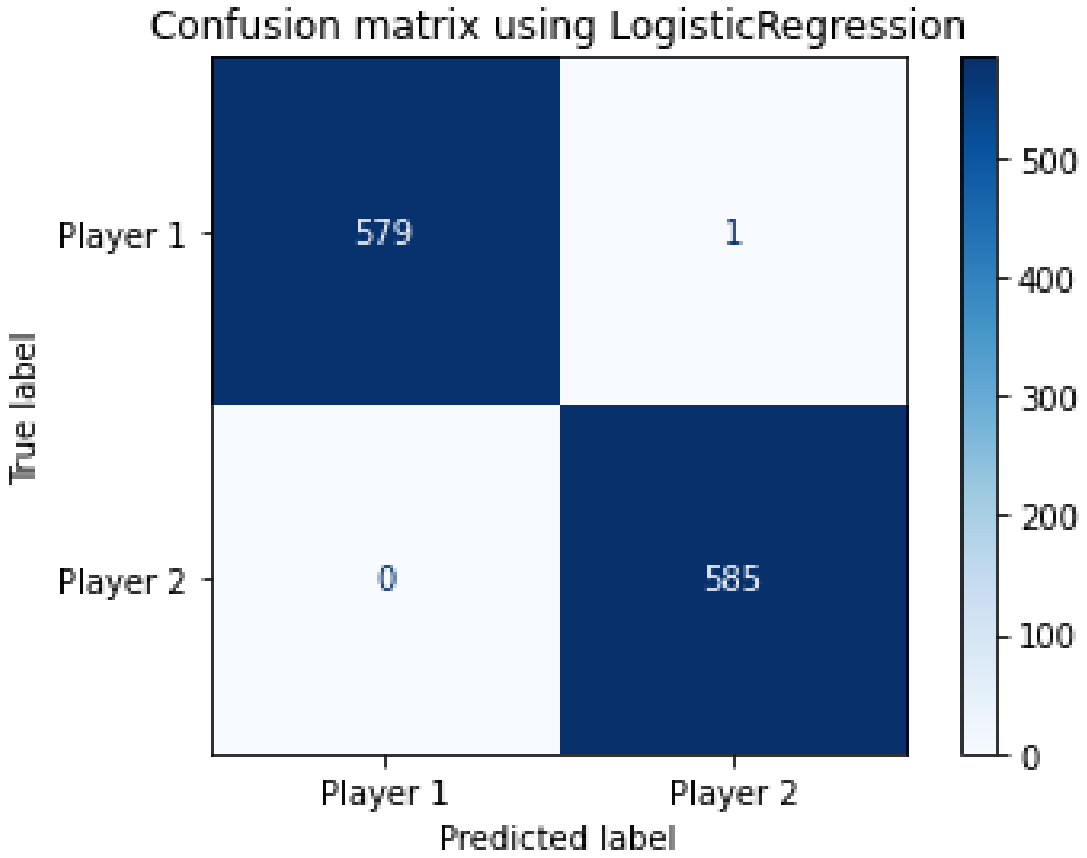
Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	1.0000	0.9966	0.9339	0.9974
F1 Score	1.0000	0.9966	0.9338	0.9974

Based on the results, Logistic Regression had the highest F1-Score and Accuracy score for both the training set and test set. We will use Grid Search CV to tune the

hyperparameters of the trained Logistic Regression model to increase the accuracy scores.

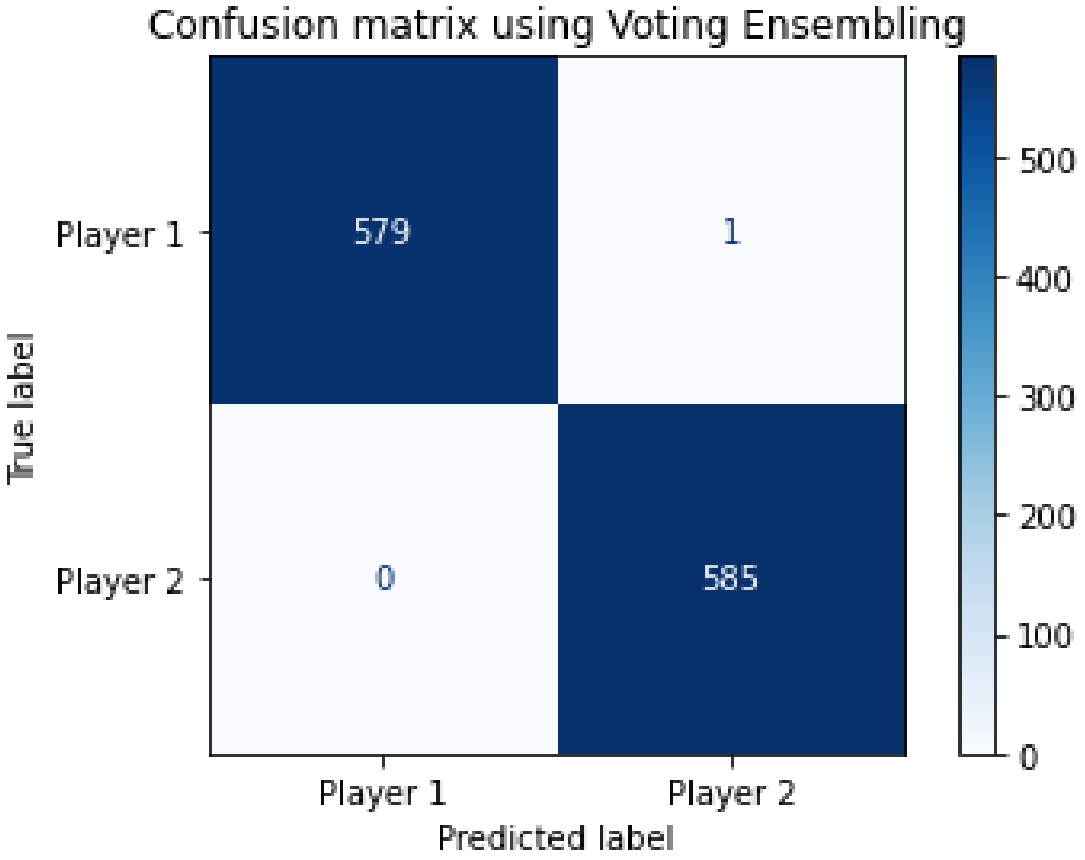
Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value in Grid Search CV, the solver for Logistic Regression, which tries to find the parameter that minimizes the cost function, was tuned to 'liblinear', resulting in the following statistics:



Logistic Regression Accuracy Scores:
Classification Accuracy: 0.9991
Classification Error: 0.0009
Precision: 1.0000
Recall: 0.9983
F1-Score: 0.9991

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support

Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 0.9991
Classification Error: 0.0009
Precision: 1.0000
Recall: 0.9983
F1-Score: 0.9991

There is no change in the results from the previous models used in the first iteration.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both Logistic Regression and Voting Ensemble produces the following learning curves:

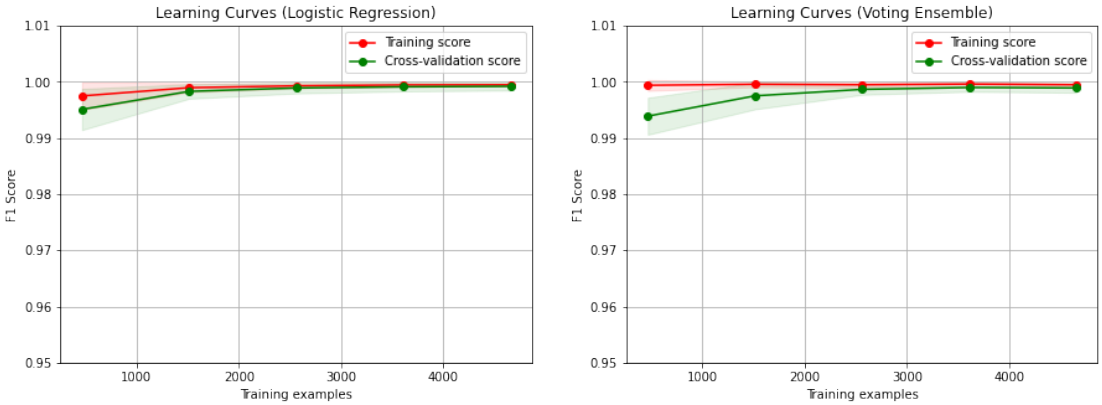


Figure 4.15: Learning Curves for Logistic Regression and Voting Ensemble

Both curves show no change in the results from the first iterations, which indicates that removing low correlation features has no affect on the learning curve for the models.

Conclusion - Model #2:
The results from the models show that there has been no impact in removing the low correlation features that were used in the original models. Logistic Regression still seems to be the better model, as it produces slightly higher accuracy scores, and the learning curves are still indicating that there may be over-fitting happening within the model. That being said, there is still some reasonable argument that the if tested correctly or with different agents, the first player to have a group disband could an impact on the outcome of the game.

First Player Tank Disband Evaluation:
Based on the above models, one would conclude that losing the first or second unit in a game is not a strong indicator of who wins or loses. From the 5823 games of data the above models were built from, Player 1 won 33% of the games when they were the first player to have a tank disband and 17% of the time when Player 2 was the first player to have a tank disband and the same results for Player 2.

Analyzing the first player with a tank disband was ineffective. There were some expectations that the first player with a tank disband would have a strong correlation to win-loss ratio, but the models suggested otherwise. This may have been because the of the number of features used in the models or the agents used, as they were both Random Action agents. Developing smarter agents with could change the results and have an affect on the win outcomes.

4.3.3 Fortress Control: Node 4

Overall Hypothesis:
The player that controls the defensive fortress node last has a direct effect on the outcome of the game.

Overall Summary:
The goal of the following models is to find any correlation between the last player to control a fortress node and the effect it has on winning games.

Model #1

Feature Variables:
Player 1 Score
Player 2 Score
Win Type
Turn Count
Node Controller
Turn Control Occurred
Control Value
Player 1 Average Health
Player 2 Average Health
Player 1 Tank Unit Count
Player 1 Tanks Average Health
Player 2 Tank Unit Count
Player 2 Tanks Average Health
Player 1 Strikers Unit Count
Player 1 Strikers Average Health
Player 2 Strikers Unit Count
Player 2 Strikers Average Health
Player 1 Controller Unit Count
Player 1 Controller Average Health
Player 2 Controller Unit Count
Player 2 Controller Average Health

Target Variable:
Winner of the Game

Agents Used:
Player 1: Random Action Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest Trees
Support Vector Machines
Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on the each other as well as on the target.

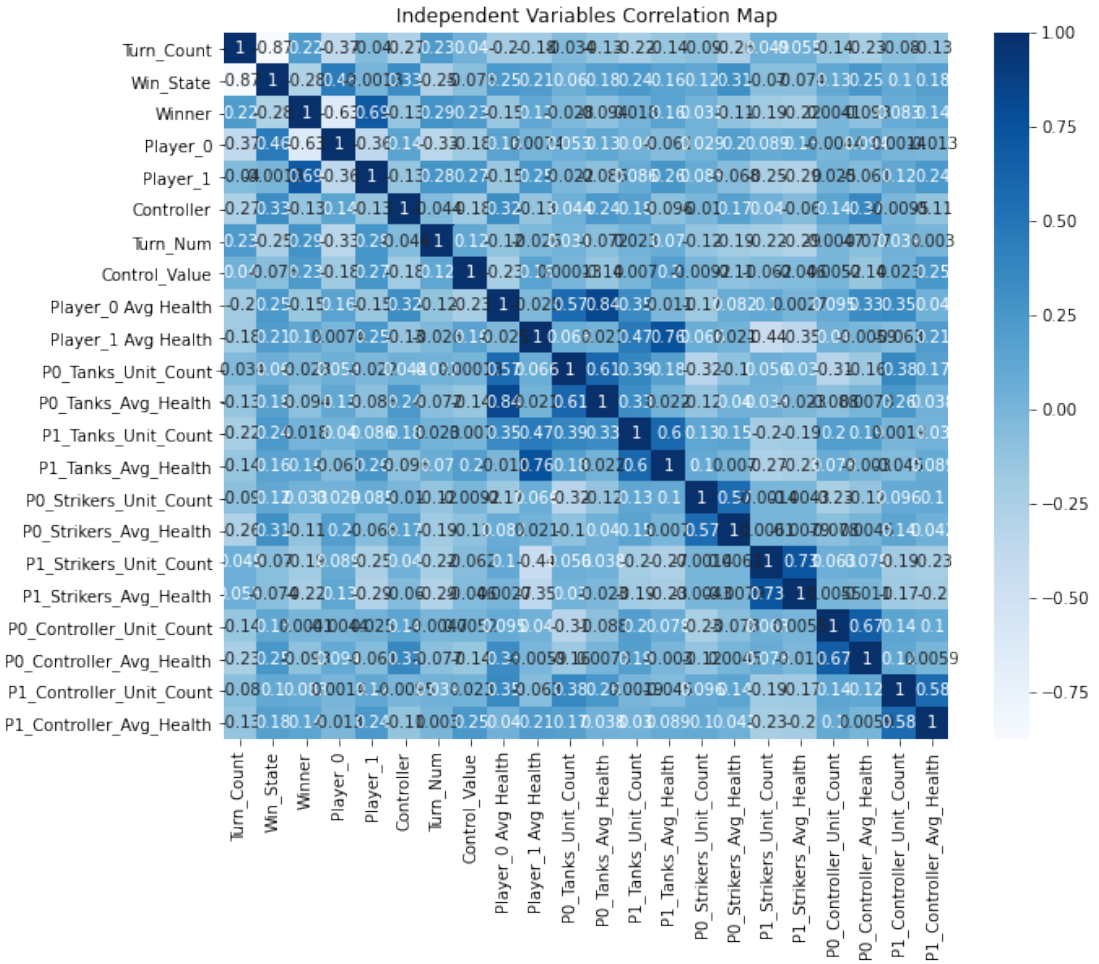


Figure 4.16: Independent Variable Correlation Map

Given these metrics, Player 1 Tank Unit Count, Player 1 Tanks Average Health,

Player 2 Tank Unit Count, Player 2 Tanks Average Health, Player 1 Controller Unit Count, Player 1 Controller Average Health, Player 2 Controller Unit Count, and Player 2 Controller Average Health has the lowest impact on the Winner and will likely be omitted in future iterations.

Base Model Evaluations:
A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C=1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

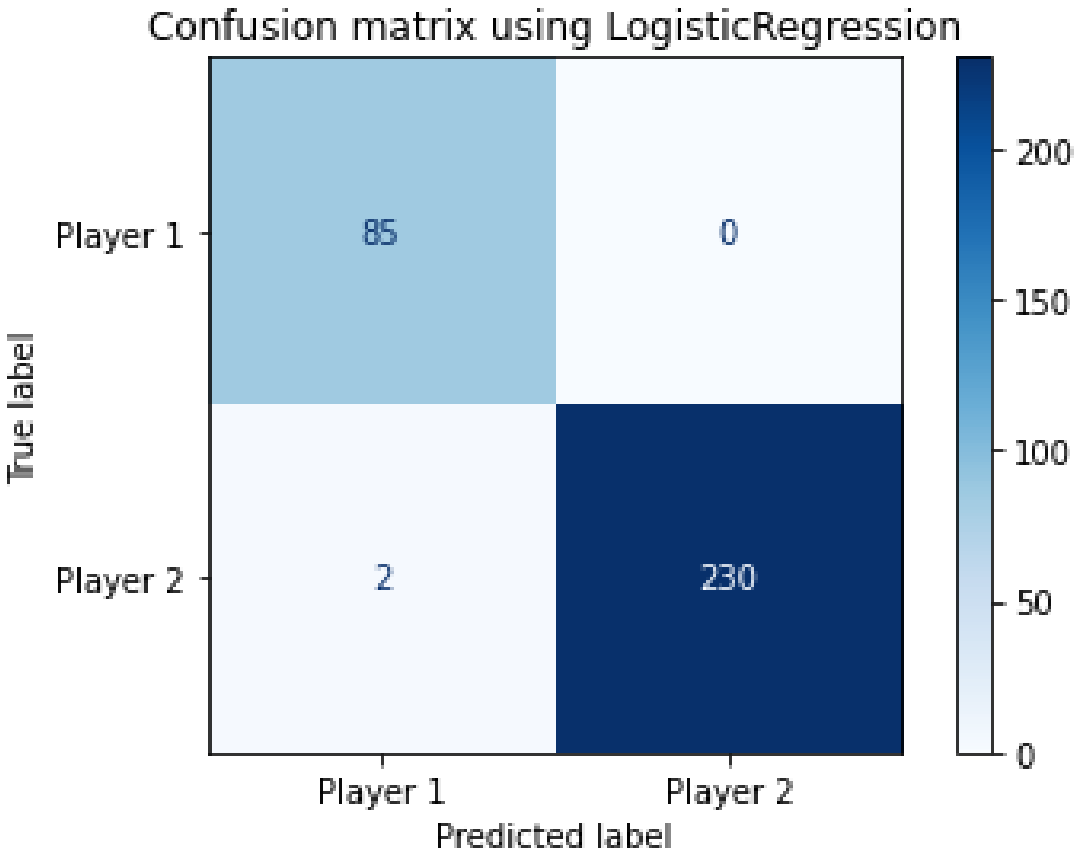
	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9992	0.9203	0.8849	0.9929
F1 Score	0.9985	0.8453	0.7256	0.9867

Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9937	0.9117	0.8864	0.9748
F1 Score	0.9884	0.8293	0.7313	0.9529

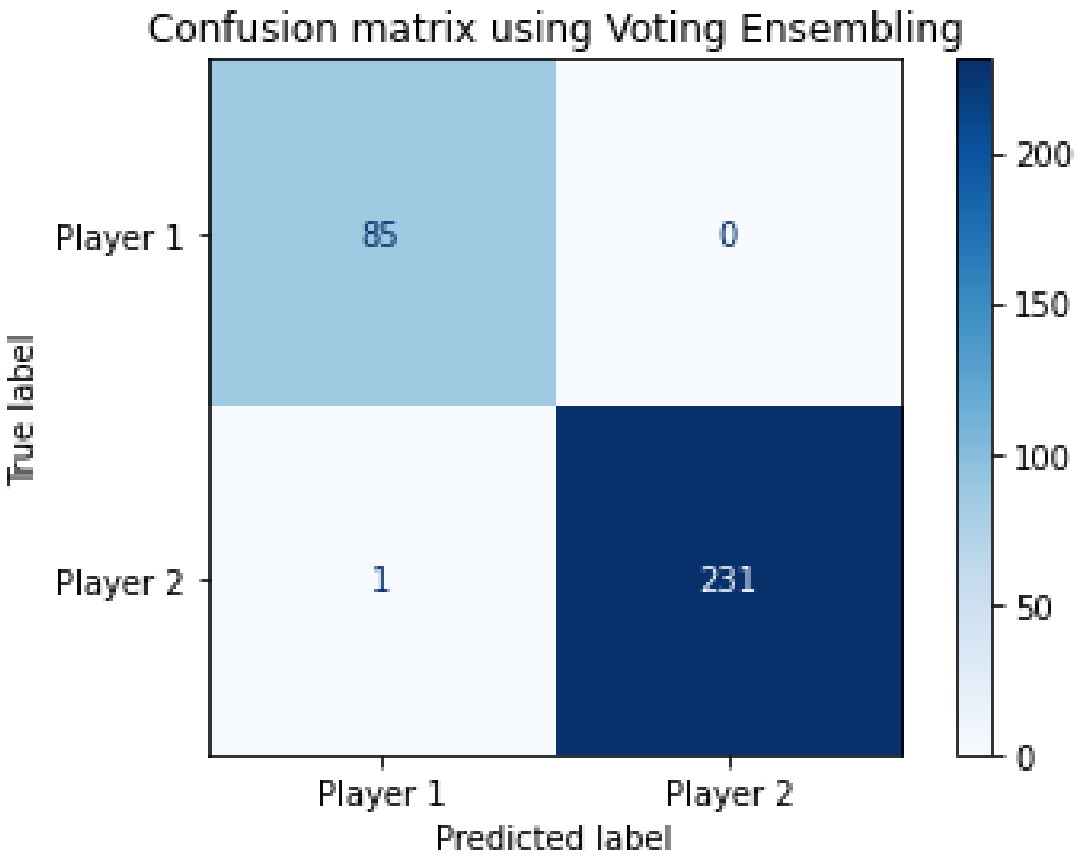
Based on the results, Logistic Regression had the highest F1-Score and Accuracy score for both the training set and test set. We will use Grid Search CV to tune the hyperparameters of the trained Logistic Regression model to increase the accuracy scores.

Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value, a C value of 25 (originally 1) produced the best result during Grid Search CV, resulting in the following statistics for the Logistic Regression model:



Logistic Regression Accuracy Scores:
Classification Accuracy: 0.9937
Classification Error: 0.0063
Precision: 0.9770
Recall: 1.0000
F1-Score: 0.9884

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 0.9968
Classification Error: 0.0032
Precision: 0.9884
Recall: 1.0000
F1-Score: 0.9942

Based on the two models, Voting Ensembles seems to be the better model, having an accuracy of 99.68% while Logistic Regression had an accuracy of 99.37%. The Recall, or how often the model is correct when Player 2 wins is 100% for Voting Ensemble and the Precision, or how often the model is correct when Player 1 wins, is 98.84%, yielding an F1 score of 99.42%.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both Logistic Regression and Voting Ensemble produces the following learning curves:

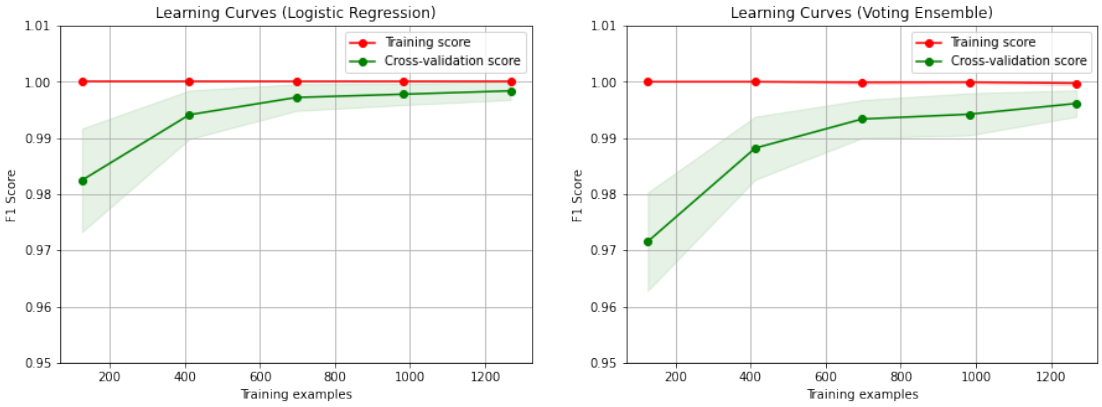


Figure 4.17: Learning Curves for Logistic Regression and Voting Ensemble

The curves for both models indicate that the models are overfitting. Also, neither curve would benefit from more training examples, as they have both converged.

Conclusion - Model #1:
The results from the models show that there is some correlation between the features and the target. Voting Ensemble seems to be the better model, as it produces slightly higher accuracy scores, although the learning curves indicate that there may be a bit of over-fitting happening within the model. That being said, there is still some reasonable argument that the if tested correctly or with different agents, the last player control a fortress node could have an impact on the outcome of the game. The next model will attempt to refine with above model by removing low correlation features to see if that has any impact on the results.

Model #2

Feature Variables:
Player 1 Score
Player 2 Score
Win Type
Turn Count
Node Controller
Turn Control Occurred
Control Value
Player 1 Average Health
Player 2 Average Health
Player 1 Strikers Unit Count
Player 1 Strikers Average Health

Player 2 Strikers Unit Count
Player 2 Strikers Average Health

Target Variable:
Winner of the Game

Agents Used:
Player 1: Random Action Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest Trees
Support Vector Machines
Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on each other as well as on the target.

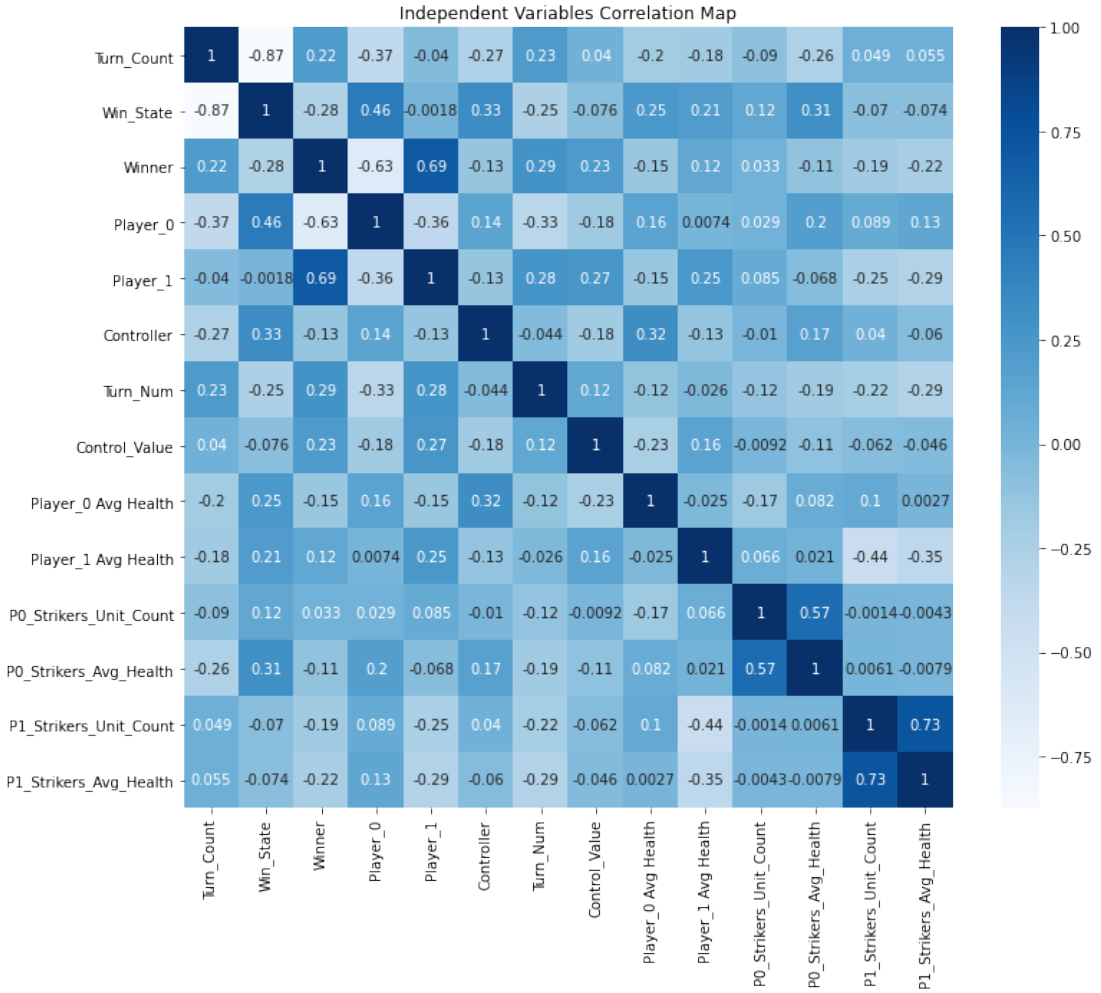


Figure 4.18: Independent Variable Correlation Map

Given these metrics, Player 1 Strikers Unit Count, Player 1 Strikers Average Health, Player 2 Strikers Unit Count, and Player 2 Strikers Average Health has the least impact on the Winner and will likely be omitted in future iterations.

Base Model Evaluations:

A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C=1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

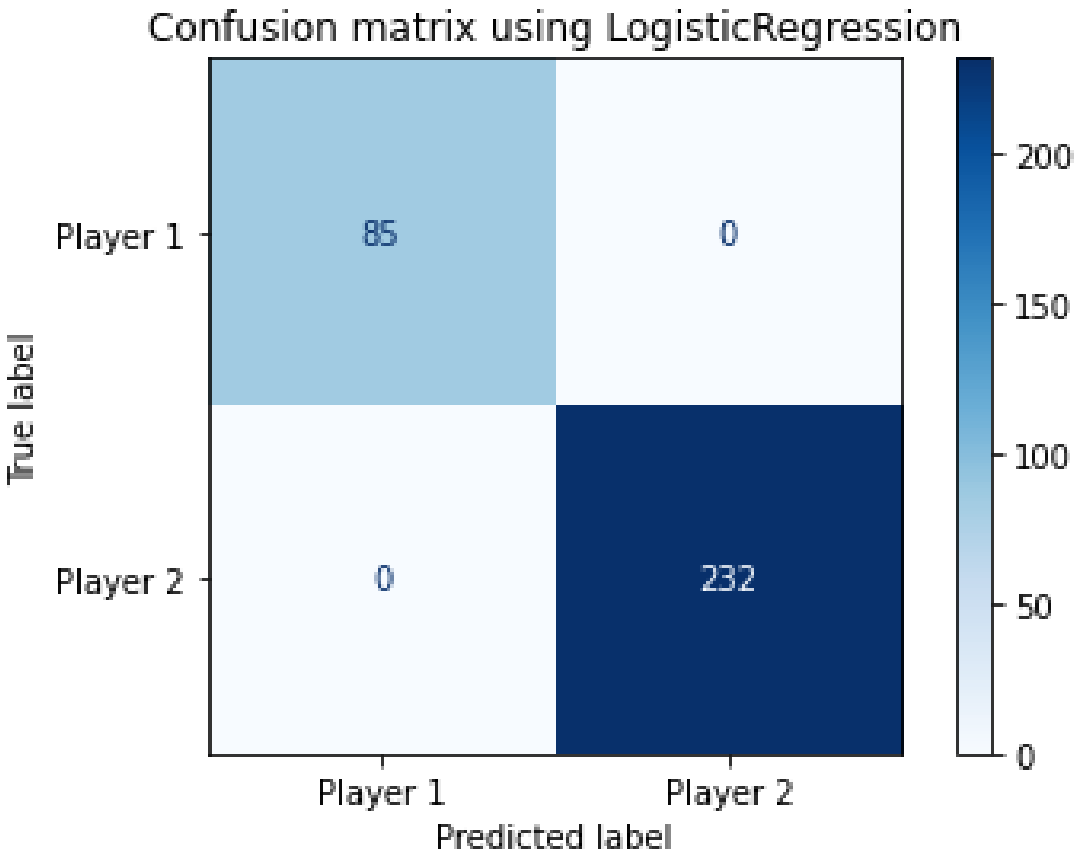
	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9992	0.9479	0.8880	0.9937
F1 Score	0.9985	0.9035	0.7361	0.9881

Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9937	0.9464	0.8927	0.9874
F1 Score	0.9884	0.9017	0.7500	0.9767

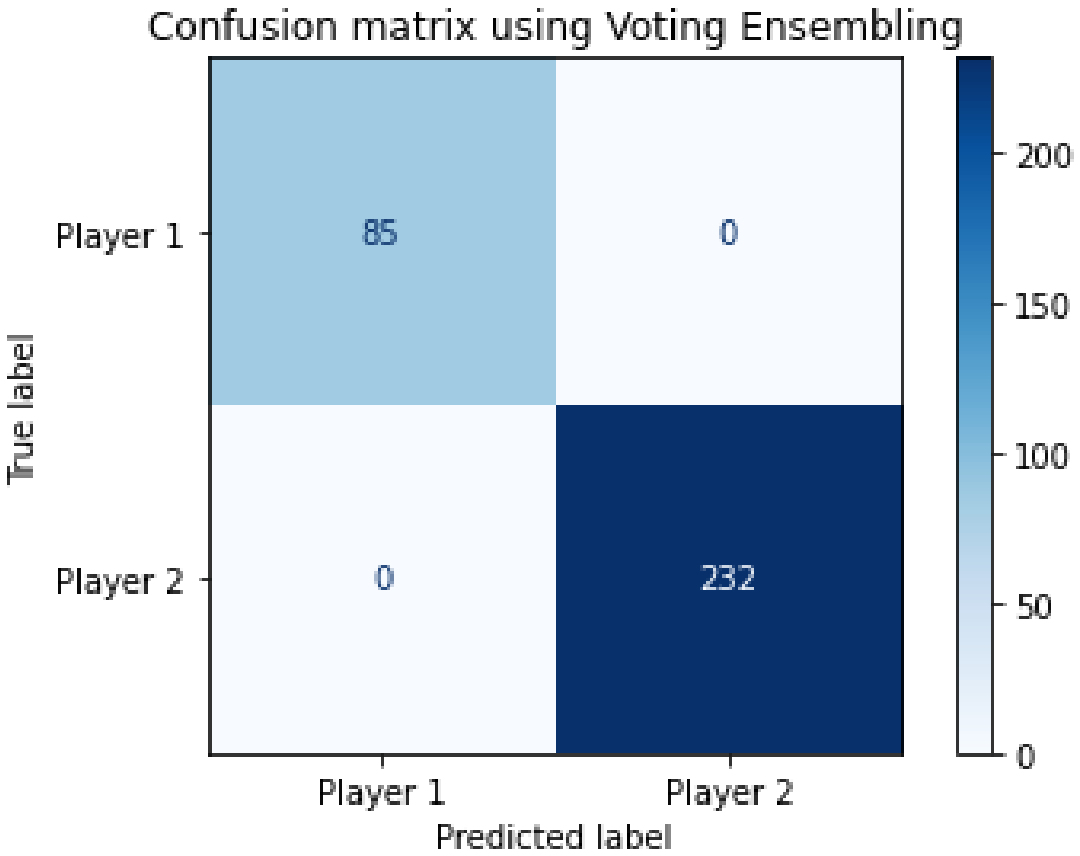
Based on the results, Logistic Regression had the highest F1-Score and Accuracy score for both the training set and test set. We will use Grid Search CV to tune the hyperparameters of the trained Logistic Regression model to increase the accuracy scores.

Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value, a C value of 0.09 (originally 1) produced the best result during Grid Search CV, resulting in the following statistics for the Logistic Regression model:



Logistic Regression Accuracy Scores:
Classification Accuracy: 1.0000
Classification Error: 0.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 1.0000
Classification Error: 0.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000

Based on the results, the two models are producing the same results, increasing their accuracy to 100%, with Logistic Regression having the biggest improve in results. The Recall, or how often the model is correct when Player 2 wins is 100% for both models and the Precision, or how often the model is correct when Player 1 wins, is 100% for both, yielding an F1 score of 100% for both models.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both Logistic Regression and Voting Ensemble produces the following learning curves:

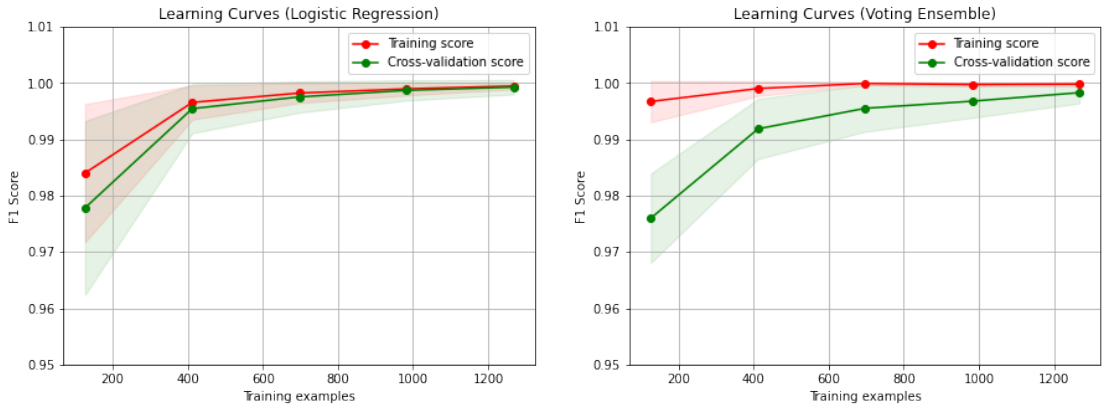


Figure 4.19: Learning Curves for Logistic Regression and Voting Ensemble

The Logistic Regression model has more noise around the training scores and having higher bias than the Voting Ensemble model, although the Voting Ensemble model seems to be over-fitting more than the Logistic Regression curve on the training set. Also, neither curve would benefit from more training examples, as they have both converged.

Conclusion - Model #2:
The results from the models show that removing the low correlation features did have an impact on the models, thus improving their accuracy scores. It's hard to say which model is better between Logistic Regression and Voting Ensemble, as they are both generating desirable F1-Scores, but Logistic Regression is showing high bias and Voting Ensemble is showing more over-fitting on the training scores. That being said, there is still some reasonable argument that, if tested correctly or with different agents, the last player control a fortress node could have an impact on the outcome of the game. The next model will attempt to refine with above model by removing more low correlation features to see if that has any impact on the results.

Model #3

Feature Variables:
Player 1 Score
Player 2 Score
Win Type
Turn Count
Node Controller
Turn Control Occurred
Control Value
Player 1 Average Health
Player 2 Average Health

Target Variable:
Winner of the Game

Agents Used:
Player 1: Random Action Agent
Player 2: Random Action Agent

Model Selection:
Logistic Regression
KNN
Random Forest Trees
Support Vector Machines
Voting Ensemble

Correlation Analysis:
A correlation map is used to determine each feature's effect on each other as well as on the target.

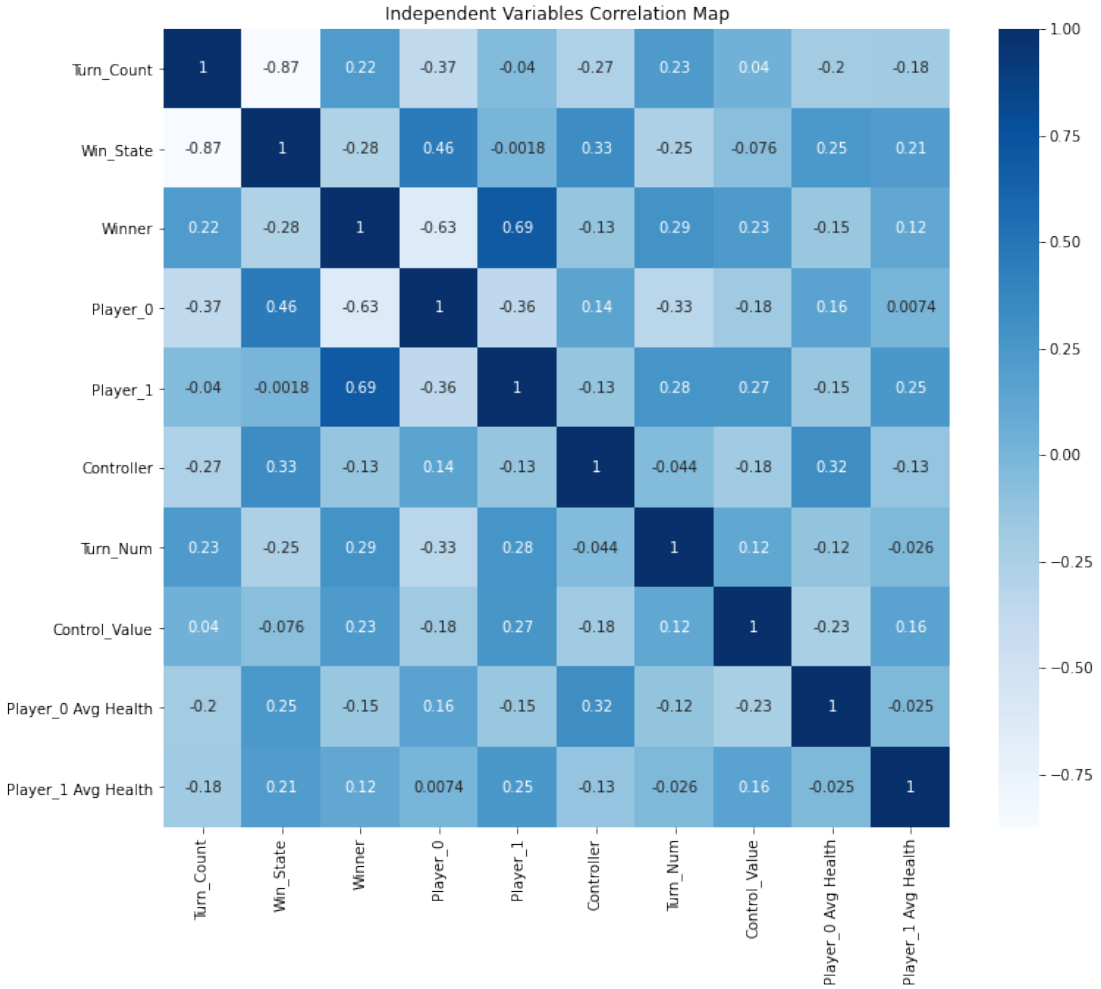


Figure 4.20: Independent Variable Correlation Map

Base Model Evaluations:

A base model case was used using Logistic Regression (C = 1), KNN (n-neighbors = 10), Random Forest Trees (max depth = 2), and SVM (C=1), yielding the following F1-Score and Accuracy scores on both the training and test set:

Training Set:

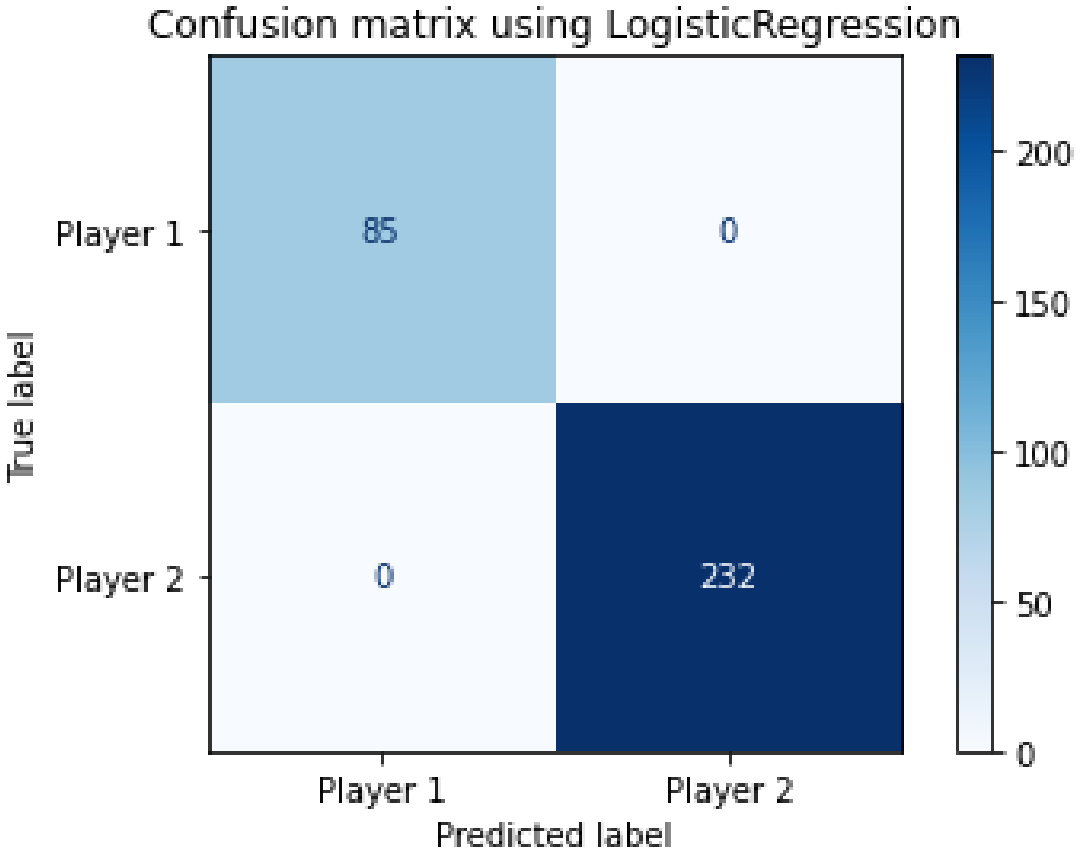
	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9976	0.9677	0.9259	0.9929
F1 Score	0.9956	0.9391	0.8401	0.9867

Test Set:

	Logistic Regression	KNN	Random Forest	SVM
Accuracy	0.9937	0.9716	0.9306	0.9874
F1 Score	0.9884	0.9467	0.8514	0.9767

Based on the results, Logistic Regression had the highest F1-Score and Accuracy score for both the training set and test set. We will use Grid Search CV to tune the hyperparameters of the trained Logistic Regression model to increase the accuracy scores.

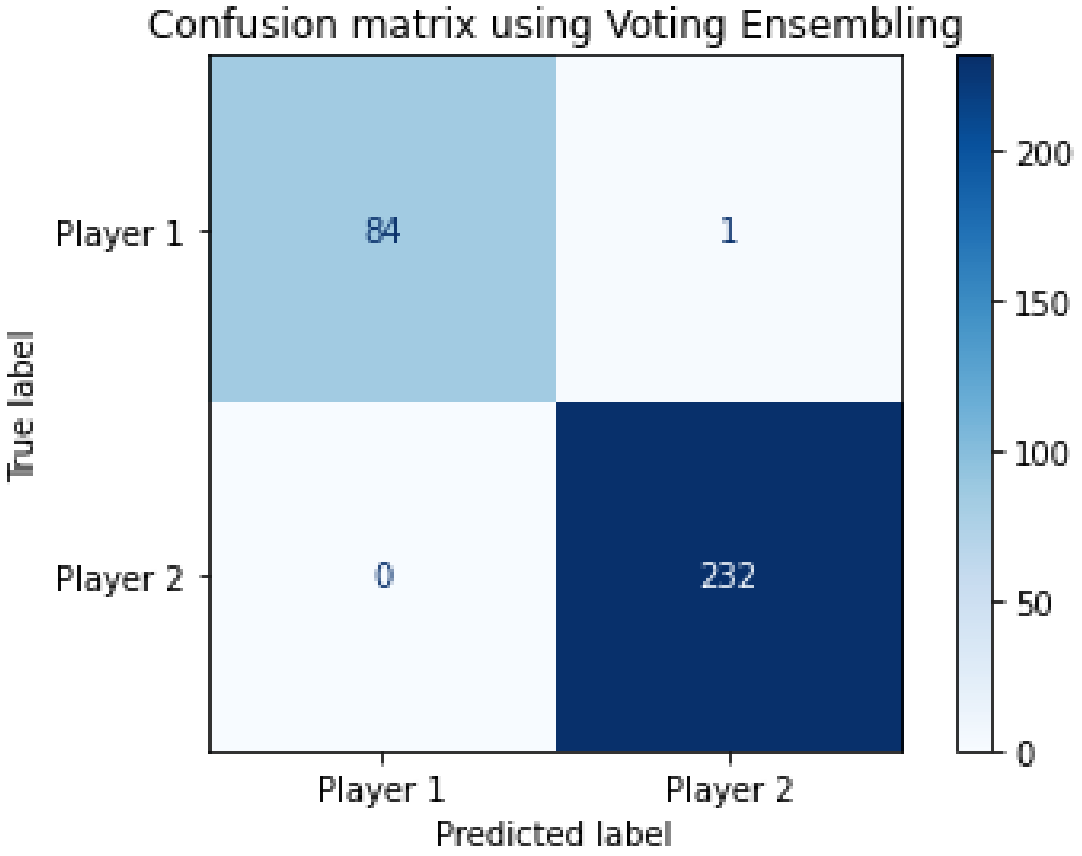
Grid Search Cross Validation Evaluation:
Using 10 for the K-Folds value, a C value of 0.09 (originally 1) produced the best result during Grid Search CV, resulting in the following statistics for the Logistic Regression model:



Logistic Regression Accuracy Scores:
Classification Accuracy: 1.0000
Classification Error: 0.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000

Using the hyperparameters from Grid Search CV for each model, we use Voting Ensemble to compare the results of a single model (Logistic Regression) to an

ensemble of the tuned Logistic Regression, K-Nearest Neighbors, and Support Vector Machine models. The confusion matrix for the Voting Ensemble model produced the following results:



Voting Ensemble Accuracy Scores:
Classification Accuracy: 0.9968
Classification Error: 0.0032
Precision: 1.0000
Recall: 0.9882
F1-Score: 0.9941

Based on the two models, Logistic Regression is producing the same results as the previous models but the Voting Ensemble model accuracy scores have decreased.

Learning Curve Evaluation:
Plotting the results of the F1-Score vs number of training examples for both Logistic Regression and Voting Ensemble produces the following learning curves:

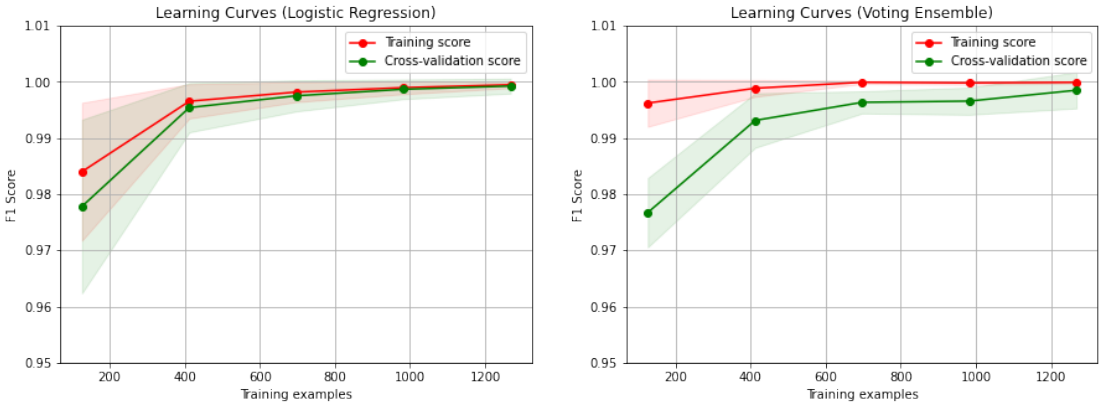


Figure 4.21: Learning Curves for Logistic Regression and Voting Ensemble

There has been no change in the learning curves from the previous iteration in Model #2.

Conclusion - Model #3:
The results from the models show that there was no significant impact in removing the Striker features. It's hard to say which model is better between Logistic Regression and Voting Ensemble, as they are both generating desirable F1-Scores, but Logistic Regression is showing high bias and Voting Ensemble is showing more over-fitting on the training scores. Logistic Regression also did not decrease the accuracy scores, which may indicate that it is a stronger model, but the results could be because of high bias in the model. That being said, there is still some reasonable argument that, if tested correctly or with different agents, the last player control a fortress node could have an impact on the outcome of the game. The next model will attempt to refine with above model by removing more low correlation features to see if that has any impact on the results.

Fortress Control: Node 4 Evaluation:
Based on the above models, one would conclude that losing the first or second unit in a game is a strong indicator of who wins or loses. From the 1585 games of data, the above models were built from, Player 2 won 70% of the games when Player 1 controlled Node 4 last. Defensive Node 4 defends the based node of Player 1, which shows that Player 2 is winning 70% of the time, even though Player 1 wins by having a higher score with a Time Up win state and not by base capture.
Analyzing the last player to was effective. There were some expectations that the last player to control node 4 would have a correlation to win-loss ratio. Although effect, these results could be skewed because of only checking one defensive node and not all 4 defensive nodes to determine out of all 4 nodes, which one was last controlled by which player and if that had any impact on the out of the game.

One primary potential issue with this data is the type of agents used in the models. The agents were the random action agents provided with the project. When placing the Random Action Age the Striker unit as the first unit lost did decrease by 6%, but with the Striker moving to a node with no support, it will likely be destroyed. Because of issues like these, smarter defensive agents that target defensive nodes could have an effect on the results.

4.4 Investigation Documentation Template

We believe that our findings will be more accessible to individuals not directly involved in our project if they are consistent and clear across all the metrics we investigate. For this reason, we thought it prudent to establish a formal template to complete as we take each metric and examine it using the methods explained in the Modeling and Testing sections. Formalizing our process will ideally lead to a more streamlined and refined investigation process as we iterate throughout our testing phase.

An important disclaimer is that this template is subject to change as we move forward with the project. We may find that some information listed in this outline of the documentation may be more or less relevant than we initial thought and choose to remove or emphasize certain aspects.

As our project is operating in conjunction with the other EVERGLADES teams, we believe formatting our findings into this standardized style will convey the information discovered during our investigation more accurately. This information will consist of the following:

4.4.1 Target Metric

The initial section of the investigation document will contain information regarding the Target Metric. This will include an explanation of how this metric is being calculated as well as a brief justification for why it is believed to be worth investigating. It will also include the origination of the target metric. If one of the AI development teams wants a specific metric to be investigated, we will list that as the motivation behind said investigation. If the metric is going to be represented as an acronym as it is referred to further in the documentation, that will

be mentioned as well.

4.4.2 Data Set

Moving into the Analytics portion of the form, there will be several important things to note. Due to the variance in Agent ability and training levels, it is important to note the data set used for modeling and analytics. The agents involved in the represented data should be noted and documented. The level of training that the agents have been exposed to, as well as the performance level of the agents outside of the examined data set, is an important factor in determining whether or not a metric is a valid predictor of success.

The method by which the metric is drawn from the game output files is also to be documented. For example, if we were looking at the ratio between combat on friendly controlled nodes compared to enemy controlled nodes, it would be beneficial to show which output files data was pulled from. How that data was queried from our database would also be useful for helping individuals unfamiliar with our design to understand our process.

4.4.3 Modeling and Classification

The final portion of the Analytics Section of the document will contain the visual representations of the data set and the hypothesis testing performed during model building. The content of this portion is more thoroughly outlined in Sections 3 and 4 of this document, in the Potential Models and Testing Process Overview sections, respectively.

4.4.4 AI Script Development

Following the Analytics section, would be a rundown of the AI Script Development process. This would include the relevant Agents and how they were modified, showing quantifiable information. This would include but is not limited to the different levels at which the critic introduced in the Actor-Critic model affected reward policy, the amount of time the Agent was given to train after the adjustments were made, and which agents it was trained against. Once again,

more in-depth explanations of the Reinforcement Learning process can be found in Sections 3 and 4 of this document, in the Reinforcement Learning and AI Script Development sections, more specifically.

4.4.5 Results

Wrapping up the investigation document will be a section containing the results of the investigation process. This would include analysis of the new data generated during the training of the Agents that have been modified. This analysis will allow us to determine whether a metric is a valid indication of win rate or not. We may also determine that our testing was insufficient and resubmit the same metric for further testing, using the knowledge gained to modify the Agents in a more impactful way.

Documenting our investigations in this organized, repeatable fashion will allow an outside individual to understand our methods from start to finish. This will greatly improve the accessibility of our findings and our ability to assist the AI Agent development teams.

The initial draft of our investigation template is provided in the following. It was developed as a PDF form with fillable fields.

Target Metric – [Name or Acronym]

Description: Why is this metric worth investigation? How is the metric calculated?

Origin: Was this metric generated from examining data sets? from an AI Development team?

Data Set – [Data Set Identifier]

Agent(s): List the Agents used to create the data set. Include descriptions of each Agent (average performance level, RL training method, relative age, etc.)

DB Query: How was this information pulled from the existing database?

Relevant Telemetry files: Which game output files contain information used to determine this metric?

Modeling and Classification

Model 1: Include all visual representations and relevant analysis.

Model 2: Include all visual representations and relevant analysis.

Model X: Include all visual representations and relevant analysis.

AI Script Development

Modified Agent(s): List the new Agents created by modifying the originals.

Adjustment Method: Explain how the Agents have been modified from their original state. How does this modification effectively test the target metric? Provide each degree of reward scaling used.

Training Time: How long was the Agent allowed to train before reexamination?

Opponent(s): Which Agent(s) opposed the modified Agents in training?

Result – Metric has been [Accepted/Rejected/Submitted for further testing]

Analysis of Data Set from Training:

Model 1: Include all visual representations and relevant analysis.

Model 2: Include all visual representations and relevant analysis.

Model X: Include all visual representations and relevant analysis.

Chapter 5

Administrative Content

5.1 Project Budget

There are no expected costs to be incurred in the course of completion of this project outside of the potential need for a server to test agents on. This server will be acquired through AWS or another provider and will cost less than $100.

The only other cost incurred by the group is the cost of printing and binding this document to be submitted at the culmination of Senior Design I. This cost will vary depending on whether the document will need to be printed in color or black and white. The cost will be less than $100.

5.2 Project Milestones - Senior Design I (Aug - Dec 2019)

5.2.1 Initial Research and Acclimation

Our team will be working with several different pieces of software for the first time and becoming familiar with them will take time. OpenAI Gym is being used to develop the Artificial Intelligence agents that participate in the game. SciKit Learn is a python based data analysis tool that allows for the examination of large data sets as well as provides visual representations of that data. Docker is

an open-source technology to run containers. These containers package code and dependencies so applications can run agnostic to their computing environment. Our team must develop competency operating with all of these tools.

5.2.2 Agile Sprints

Throughout this project, we will need to be able to adapt to changing AI agents as well as variance in the relevance of different metrics. To help manage this, our development process will be split into three different Agile sprints. Each sprint will consist of Hypothesis Generation and Data Preparation, AI agent development, testing of agents against our hypotheses, and final data review and adjustment. These iterations should allow us to adjust our approach and direction based on new findings as well as include a wider array of AI agents as the other EVERGLADES groups provide them.

Game / Historic Data Preparation

- Sprint 1: 10/18/2019
- Sprint 2: 11/15/2019
- Sprint 3: 01/24/2020

Before improvement or testing for AI agents can occur we must first generate a collection of testable hypotheses that allow us to compare certain statistics provided in game replays. This data collection and initial analysis will allow us to have a more focused examination of certain behaviors and actions AI agents employ.

SPRINT 1 AND 2 COMPLETED

AI Development

- Sprint 1: 10/23/2019
- Sprint 2: 11/22/2019
- Sprint 3: 01/31/2020

Once we have some testable hypotheses regarding relevant metrics, we can test those hypotheses by creating AI scripts of targeted behaviors. These AI agents could implement one or more scripts. Existing agents or those provided to us by other EVERGLADES teams could be retrofitted with these modifications to determine how the combination of factors impacts outcomes.

Agent Testing (Behavioral and Game State)

- Sprint 1: 10/28/2019
- Sprint 2: 12/03/2019
- Sprint 3: 02/07/2020

With AI agents that are ready to be released onto the EVERGLADES platform, we will be able to test our hypotheses and see how these agents interact with each other. Each game is played on a server that tracks the agents and environments, with each agent contained in a Docker container. The server, also in a Docker container, will return telemetry output data to log files. The rendering of the game is post processed.

SPRINT 1 AND 2 COMPLETED

Data Review and Revision

- Sprint 1: 10/31/2019
- Sprint 2: 12/08/2019
- Sprint 3: 02/14/2020

The amount of games that will be run to generate a comprehensive data set is large and will provide us with a healthy archive to either reject an invalid hypothesis or accept an accurate hypothesis and continue to investigate it further. Other teams can also suggest hypotheses during this review period for testing in a future sprint.

SPRINT 1 AND 2 COMPLETED

5.2.3 Final Design Document

12/05/2019
Generation of a final design document will require bringing together documentation from several different mediums including but not limited to Jupyter Notebooks, Trello, and Github. Creating a cohesive, well formatted, readable document from these resources will take some time.

COMPLETED

Task Name	Start	Finish
Research and Analyze Data	10/01/19	10/18/19
AI Script Development	10/14/19	10/23/19
Initial Prediction Algorithm	10/15/19	10/28/19
Design Review (LMCO)	10/30/19	10/30/19
Research and Analyze Data	11/04/19	11/15/19
AI Script Improvements	11/06/19	11/22/19
Prediction Algorithm Improvements	11/11/19	12/03/19
Final Design Document	11/21/19	12/05/19
Research and Analyze Data	01/06/20	01/24/20
AI Script Improvements	01/13/20	01/31/20
Prediction Algorithm Finalization	01/14/20	02/07/20
Real Time Analytics Engine	01/20/20	02/28/20
Code Review and Test	02/03/20	03/13/20
Documentation	03/02/20	03/27/20
Presentation	03/31/20	03/31/20

Figure 5.1: Senior Design 1 Gantt Chart.

5.3 Project Milestones - Senior Design II (Jan - Mar 2020)

5.3.1 AI Training Exploration

01/01/2020
With the highly variable nature of the EVERGLADES game platform, we are unsure as to the estimated amount of training that will be needed for us to see change in the behavior of Agents once we have made our modifications to them in the metric investigation process. Without knowing when we should expect to see change, we may improperly discard metrics that have actual relevance but were not given enough time to come to fruition.

For this reason, we plan on initializing the testing process for some basic metrics before the start of Senior Design II in January. The experience of the AI development teams, Team Black and Team Gold, will also provide us some insight into what we should expect during the training process.

5.3.2 Open Lines of Communication and Collaboration with other EVERGLADES Teams

01/15/2020
Several aspects of our objectives and aspects of our design lend themselves well to collaboration with the other EVERGLADES teams. We have communicated this possibility and the teams were receptive to the idea. Opening up a platform for exchange of ideas and communication, through a Discord Server for example, would make collaboration more efficient and viable.

We believe that the experience and knowledge the other EVERGLADES teams have gained through their development of the replay platform and training Reinforcement Learning Agents would make them valuable resources during our work.

5.3.3 Establish Connection between Testing Environments and our Database

01/15/2020
The more comprehensive our data sets are, the more relevant our analysis can become. For this reason, it is vital that we create a connection between the testing environments of the AI Agent development teams and our database. This would allow us to smoothly catalogue the telemetry output from the entirety of their Agent training.

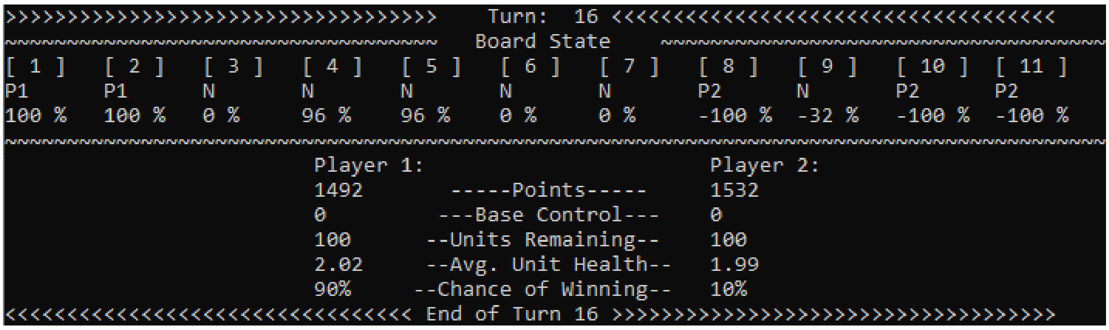
In addition to the valuable output produced during their testing, the agents developed by Team Black and Team Gold can be cloned and used to test the relevance of our metrics. Having access to this library of agents of varying proficiency and training exposure will allow us to make more comprehensive deductions when analyzing the telemetry data from our modified agents.

5.3.4 Real Time Analysis Engine

The Real Time Analytic Engine and Predictive Algorithm were designed so that in the future it could run with the EVERGLADES match playback system. The goals of the Real Time Analytic Engine and Predictive Algorithm were to run simultaneously with the client and provide information that the match playback system could utilize and display while running. The Real Time Analytic Engine provides insight into agent behavior and turn by turn prediction of which agent is likely to win. In order to achieve this task the client needed to be modified. The original form of the play back system utilized the telemetry data set that was generated by the game. The primary issue with this was that the telemetry data was generated after the game was over so it was not possible to run in real time in its' current state. The two options to remedy this issue were to alter the telemetry data and generate the CSV files turn by turn or to pipeline the telemetry data from the client to the engine turn by turn. We chose to pipeline the data from the client to the Real Time Analytic Engine turn by turn.

Game State and Agent Behavior

With the telemetry game data now pipe-lined it was possible for the game state to be tracked and updated turn by turn. This allowed access for the Real Time Analytic Engine to present information as the game is running. The data stream from the client to the Real Time Analytic Engine provides access of every data point the telemetry data tracks and generates to the CSVs. The information that was chosen to display is turn number, agent score, board states (which agent controls a node or is it neutral), base control values, units remaining and units average health.



```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>    Turn:  16 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   Board State    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[ 1 ]    [ 2 ]    [ 3 ]    [ 4 ]    [ 5 ]    [ 6 ]    [ 7 ]    [ 8 ]    [ 9 ]    [ 10 ]   [ 11 ]
P1       P1       N        N        N        N        N        P2       N        P2       P2
100 %    100 %    0 %      96 %     96 %     0 %      0 %      -100 %   -32 %    -100 %   -100 %
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                         Player 1:                          Player 2:
                         1492        -----Points-----       1532
                         0          ---Base Control---      0
                         100       --Units Remaining--      100
                         2.02      --Avg. Unit Health--     1.99
                         90%      --Chance of Winning--     10%
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< End of Turn 16 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

(a) Game State Summary

Prediction Algorithm Design

The prediction algorithm uses machine learning classification algorithms (Scikit learn) to make turn by turn predictions as to which agent is most likely to win. In addition to this, the classification algorithm provides the odds or the confidence of that prediction. The algorithms tested were K-Nearest Neighbors, Logistic Regression and Random Forest Classifier. Random Forest Classifier was providing the best results early in development so that classification algorithm was chosen to move forward with.

Features

With any classification problem feature selection is one of the most important aspects. In this case it needed to be determined which features of the game would be the best indicator for the response that is trying to be predicted, which agent is most likely to win the game. The features that could be used are quite large given the telemetry data that is provided but it was determined early to focus

on simple and intuitive aspects of the game. This would help keep the amount of features small and reduce complexity. The features chosen to focus on where turn number, agent points, base capture values, units remaining and average unit health.

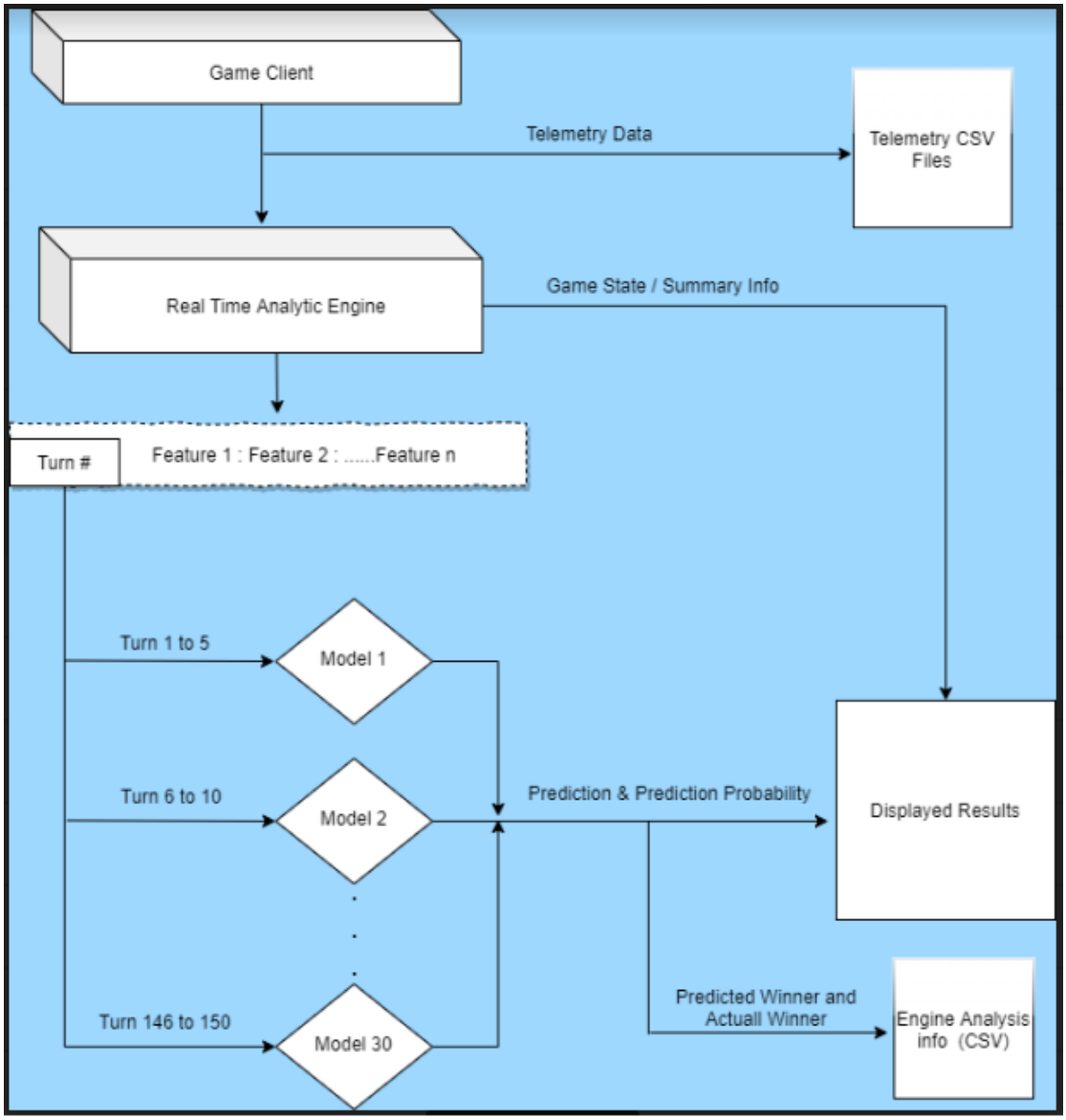
One important aspect of the features that was discussed early is the difficulty for the classification algorithms to place emphasis on turn number. Turn number was deemed to be important because every other feature means very little if turn number is not taken into consideration. For example, a prediction would be considerably different if an agent was winning by 500 points on turn 50 versus turn 140. Due to this a different approach would be used when training models and generating data sets.

Classification Models

Due to the importance of turn number it was determined a multi-model prediction system would be viable. Generally classification algorithms will use one data set to build a single model but for the reasons previously stated the multi-model approach was used.

The initial idea that was tested was to build models from data compiled from turn pairs. There would be 75 different models (150 potential game turns). Model one would be trained on data from turns one and two, model two would use turn data from three and four and this would continue to model seventy five which would be trained on data from turns one hundred and forty nine and one hundred and fifty. Now when predictions are made, instead of passing turn number in as a feature, turn number is used to determine which model to use and the features were then passed to the respective model. This would give a prediction using the chosen features directly related to that turn.

Overall this implementation was a success but there were issues, one in particular forced a redesign. Splitting the models and training them on pairs of turns lead to the models being too narrow. This implementation had issues capturing rare or uncommon occurrences; more specifically it was difficult to include data that could predict base capture scenarios. Games where agents capture the opponent's bases are sparse and generally require heavily scripted agents to perform this task. This made diversification of data with this tactic harder to collect. To remedy this issue the amount of turn data used per model was increased from two to five. This also reduced the amount of models used to thirty.



(a) Real-Time Engine Overview

Agents

The base agent used for data set generation was a random action agent. This agent simply did random actions and when matched against each other the two agents created a diverse pool of data that helped make predictions when agents would win on turn one hundred and fifty by points. This agent rarely captures bases so in order to generate this data, different agents were used.

To generate base capture data three different types of agents were used. Each of these agents were designed to capture the base early game, mid game and late game. The final model was comprised of the following data:

- 500 Games of Random Action Agent versus Random Action Agent
- 500 Games of Early Game Base Capture Agent versus Random Action Agent
- 1000 Games of Mid Game Base Capture Agent versus Random Action Agent
- 500 Games of Late Game Base Capture Agent versus Random Action Agent.

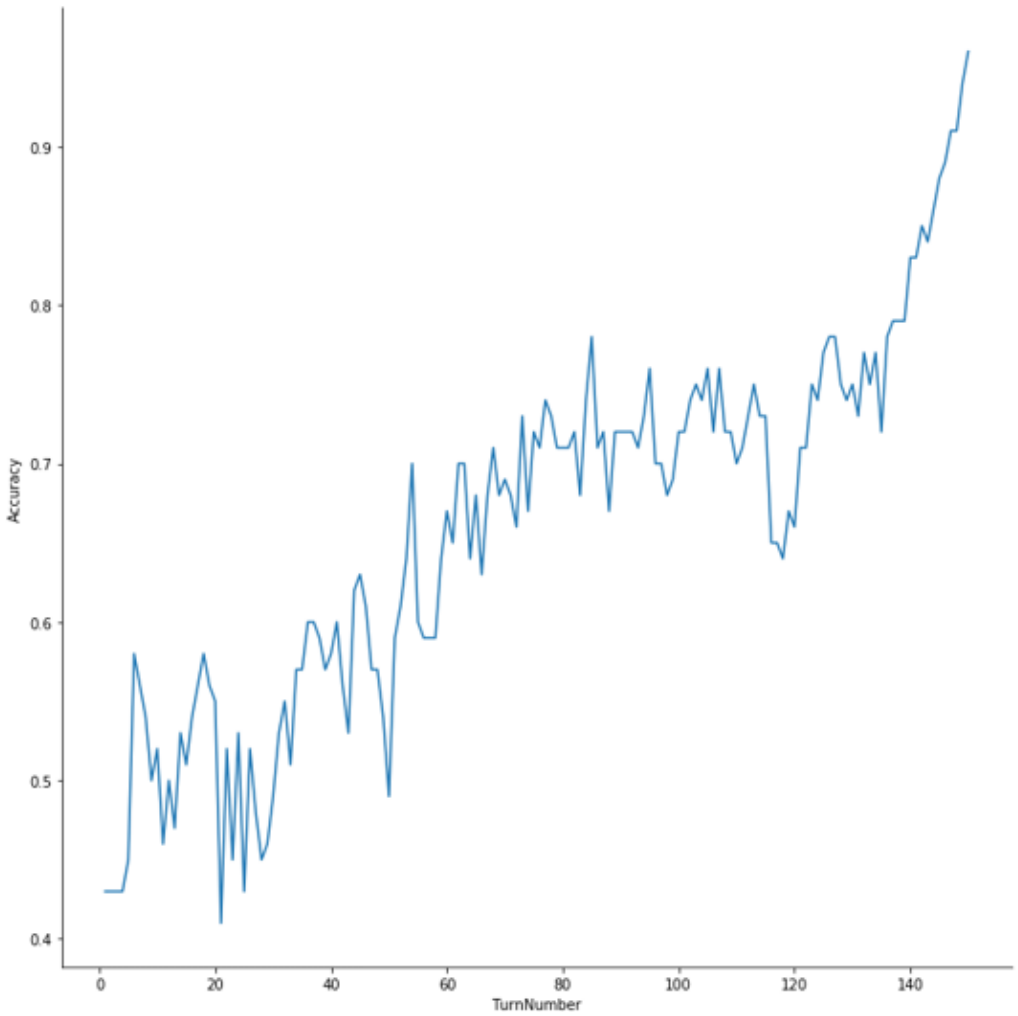
Tuning and Analysis

In the early stages of tuning an intuitive approach was taken. That is, does the prediction being made make sense with the current state of the game. For example, if player one is losing in all aspects of the game but is being favored to win this would be deemed a problem. Since the Real Time Analytic Engine is meant to run with the Playback system it made the most sense that from a viewing perspective that the predictions made sense.

In addition to this it was desired that the confidence of prediction algorithm improved over time. This does differ from a more common approach with how accurate a model is period. For instance, how accurate is a machine learning model at classifying a picture of a dog as dog. In the case of turn by turn predictions it would not be likely that a model could predict with high confidence which player was going to win the game on turn one or in this case, before turn one hundred.

In order to measure the effectiveness of the final models several aspects are taken into consideration, accuracy, specificity (recall), sensitivity, precision and the F Score. Accuracy is the overall performance while specificity measures how well the models predict player one as the winner and sensitivity measures how well the model predicts player two as the winner. Precision reflects how well the models are able to identify data points that are relevant and the F Score represents the trade off of precision versus specificity (recall). These three metrics were also analyzed over the course of the game and were expected to trend upwards as the game proceeded.

Below is an analysis of the Accuracy of the Real Time Analytic Engine that ran one-hundred games of Random Action vs Random Acton Agents.As seen here, as the game matures the models from the Real-Time Engine become more and more accurate. (To see all the results from all of the model, see the appendix.)



(a) Random Action Vs Random Action Accuracy

Real Time Engine Predictive Algorithm Conclusion and Issues

The models used in the final iteration of The Real Time Analytic Engine and Predictive Algorithm improved drastically from the original implementations but still had short comings. It was found that the models do well when predicting wins from points on turn one hundred and fifty but suffer more and more as data from base capture games was added to the data sets. It was found that as base capture data was added to data sets it skewed predictions in the mid-game and in some cases making illogical predictions. The final thirty models used did address this issue but was not completely remedied. In addition to this the agents used to generate base capture data were not diverse enough. The final version of the predictive algorithm would struggle against an agent that can reliably capture bases before turn thirty and between turn one hundred and one hundred and twenty.

5.3.5 Finding our SabreMetrics

02/01/2020
AI development and Supervised Learning models are facets of our project, but the main focus is on Data Analytics. Our top priority is to find some of these SabreMetrics, described in the Executive Summary. Finding these metrics, which go beyond the standard, combining several different elements into a single, more illuminating factor for success, is our base level requirement. With our establish framework, we will be able to examine an overwhelming number of varying factors, which will lead us to these SabreMetrics.

5.3.6 Critical Design Review and Design Showcase

03/31/2020
Our team will prepare a 25 minute presentation, showcasing our progress to date and providing an opportunity for feedback from our peers as well as our sponsors and professors.

We will also be presenting our findings over the course of the project to our sponsors at Lockheed Martin. Our team, along with the other EVERGLADES teams, have been invited back to their facilities in early March to showcase the result of our efforts.

Lastly, Our team will be participating in the Spring Senior Design Showcase. This is a platform for us to show off everything we have accomplished across the two semesters and have previous Computer Science graduates present to interact and engage with.

Chapter 6

Project Summary and Conclusions

6.1 Project Summary

In summary, this project is an exploration of the potential for improving the Reinforcement Learning process through Data Analytics. Our main objective is to find specific and in-depth metrics, similar to SabreMetrics in Baseball, to better illustrate the impact certain actions have on the performance of the EVERGLADES agents. Secondly, we want to test these metrics using the output data generated during the course of EVERGLADES matches. This data is catalogued in our database and used to create a win rate algorithm, trained using Supervised Learning methods. This algorithm can be used in conjunction with the replay visualization to show how events impact match outcomes in real time.

One of our main focuses in the development of this design was creating a system that could scale with the EVERGLADES platform as it grows and develops. We also wanted this process to be easily repeated, allowing for analysis to be conducted throughout the life cycle of the EVERGLADES platform.

We plan on accomplishing these goals by using the methodology outlined in Chapter 4. The Data Analytics portion will be the back bone of our development, with the Modeling, Regression, and Classification analysis providing direction in terms of what metrics we should be examining and testing.

Using the AC3 model, we will introduce a Critic into each EVERGLADES Agent that we are modifying to test the specific metric we are examining. The variation of the implemented Critic will provide us with a comprehensive set of telemetry data from the extensive training process the adjusted Agents are subjected to.

Using this generated data, we will be able to draw solid, concrete conclusions regarding which factors most highly impact success in the EVERGLADES platform. These conclusions will be backed by the body of work produced during the development and testing process.

We believe that the work accomplished during the course of our Senior Design Project, operating with our sponsors at Lockheed Martin on their EVERGLADES platform, will provide us with a wealth of real-world software development experience in addition to establishing a solid foundation for the EVERGLADES project moving forward.

It has been made clear on several occasions that this project is in its beginning stages and the potential for growth is tremendous. The mechanics of the game were designed with room to expand and allow for added complexity and being able to help shape the direction of the project is a valuable experience for our group.

6.2 Conclusions

At this point in the project, there are not many conclusions that can be drawn. Our goal is to use Data Analytics and Reinforcement Learning to create capable agents that can perform on the EVERGLADES system. The next few months of work will show how effective our established design is.

At the culmination of our project, we hope to be able to draw conclusions regarding the metrics we have investigated and how they can be used to more efficiently train the EVERGLADES AI Agents. We also believe that the system we will have established will allow for iteration and expansion as the EVERGLADES platform grows. We hope that other teams can use the progress we make to as a starting point in their own exploration of Reinforcement Learning and the EVERGLADES system.

The metrics we discover, akin to SabreMetrics in baseball, will ideally be used throughout future instances of the EVERGLADES platform to more accurately

represent the impact of each element of gameplay.

Bibliography

[1] The Method of Least Squares,
http://www.cs.cornell.edu/courses/cs1380/2018sp/textbook/chapters/13/3/method-of-least-squares.html

[2] Linear Regression,
https://condor.depaul.edu/sjost/it223/documents/regress.htm

[3] Machine learning: an introduction to mean squared error and regression lines,
https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/

[4] Linear Regression — Understanding the Theory,
https://towardsdatascience.com/linear-regression-understanding-the-theory-7e53ac2831b5

[5] A Simple Introduction to K-Nearest Neighbors Algorithm,
https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e

[6] Machine Learning Basics with the K-Nearest Neighbors Algorithm,
https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[7] Polynomial Regression,
https://towardsdatascience.com/polynomial-regression-bbe8b9d97491

[8] Logistic Regression,
https://christophm.github.io/interpretable-ml-book/logistic.html

[9] Logistic Regression – A Complete Tutorial With Examples in R,
https://www.machinelearningplus.com/machine-learning/logistic-regression-tutorial-examples-r/

[10] The Logistic Regression Algorithm,
https://machinelearning-blog.com/2018/04/23/logistic-regression-101/

[11] Using neural nets to recognize handwritten digits,
http://neuralnetworksanddeeplearning.com/chap1.html

[12] Classification Using Neural Networks,
https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f

[13] Ridge Regression for Better Usage,
https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db

[14] Lasso Regression: Simple Definition,
https://www.statisticshowto.datasciencecentral.com/lasso-regression/

[15] A comprehensive beginners guide for Linear, Ridge and Lasso Regression in Python and R,
https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

[16] Random Forests,
http://www.statsoft.com/Textbook/Random-Forest

[17] How Decision Tree Algorithm works,
https://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/

[18] How the random forest algorithm works in machine learning,
https://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/

[19] Sci-Kit Learn - K-Nearest Neighbors,
https://scikit-learn.org/stable/modules/neighbors.html

[20] Model Training in Sci-Kit Learn,
https://github.com/justmarkham/scikit-learn-videos/blob/master/04_model_training.ipynb

[21] Model Evaluation in Sci-Kit Learn,
https://github.com/justmarkham/scikit-learn-videos/blob/master/05_model_evaluation.ipynb

[22] Cross Validation in Sci-Kit Learn,
https://github.com/justmarkham/scikit-learn-videos/blob/master/07_cross_validation.ipynb

[23] Asynchronous Actor-Critic Agents (A3C),
https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

[24] Double Deep Q Networks,
https://towardsdatascience.com/double-deep-q-networks-905dd8325412

[25] Q and V in Reinforcement Learning,
https://medium.com/@zsalloum/q-vs-v-in-reinforcement-learning-the-easy-way-9350e1523031

[26] Reinforcement Learning: An Introduction,
http://incompleteideas.net/book/first/ebook/the-book.html

[27] Deep Reinforcement Learning Doesn't Work Yet,
https://www.alexirpan.com/2018/02/14/rl-hard.html

[28] Monte Carlo in Reinforcement Learning, the Easy Way,
https://medium.com/@zsalloum/monte-carlo-in-reinforcement-learning-the-easy-way-564c53010511

[29] Progressive Insurance Snapshot,
https://www.progressive.com

[30] Deep Q Learning Networks,
https://www.youtube.com/watch?v=OYhFoMySoVs

[31] Fitbit Picture,
https://www.fitbit.com/home

[32] UE4 Documentation,
https://docs.unrealengine.com/en-US/GettingStarted/index.html

[33] Docker Documentation,
https://docs.docker.com/

[34] Football analytics - a demo from SAP about Big Data,
https://www.youtube.com/watch?v=wUr2Useye2E

[35] Analyzing NFL Combine Data Analysis on Wide Receivers.,
https://public.tableau.com/views/DynamicClustering-NFLCombine/DynamicClustering-NFLCombine

[36] Health Economics, 1st edition by Jay Bhattacharya, Timothy Hyde, and Peter Tu ISBN-13: 978-1-137-02996-6 Publisher: Palgrave MacMillan; Copyright year: © 2014,

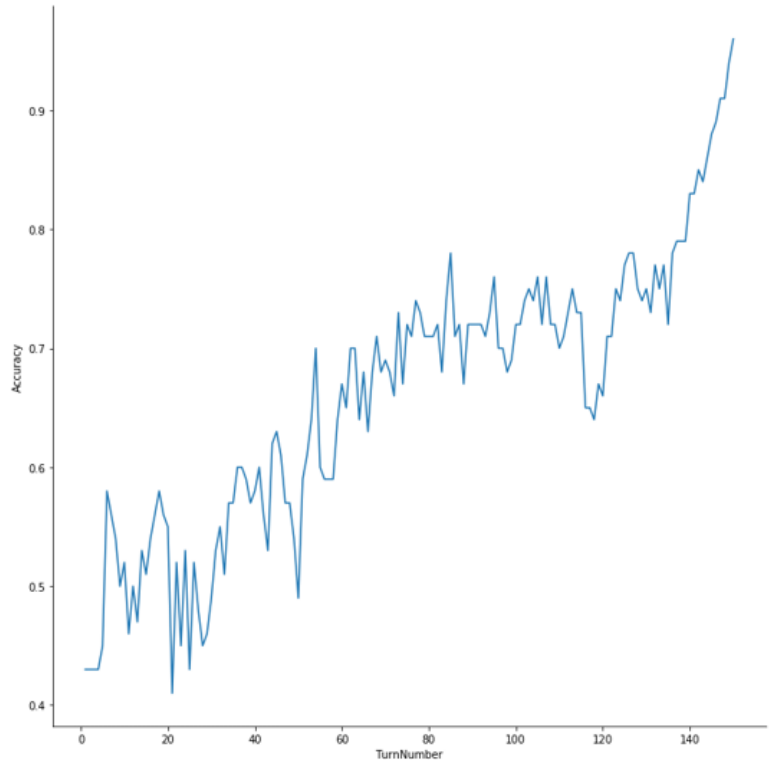
Appendix A

Appendix

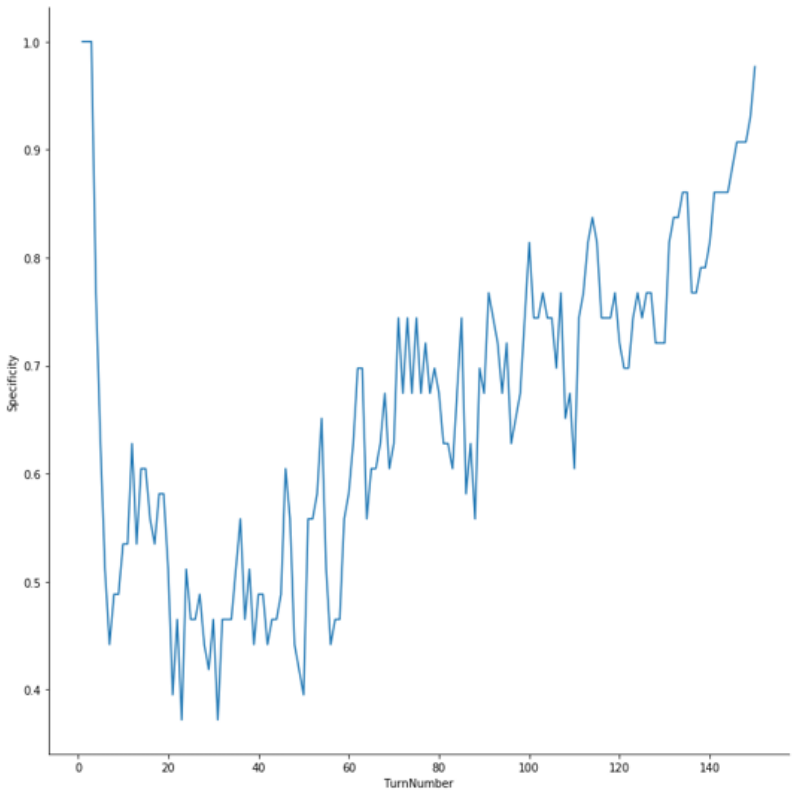
A.1 Real-Time Engine Analysis

A.1.1 Random Action vs Random Action

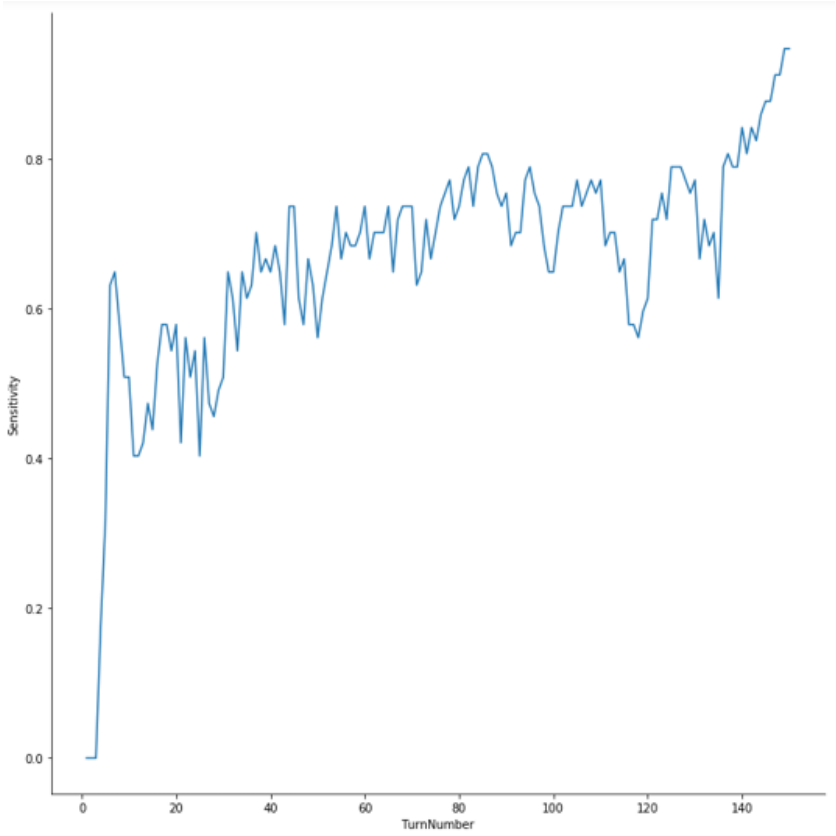
The following figures are from 100 games from Random Action Agents vs random action agents:



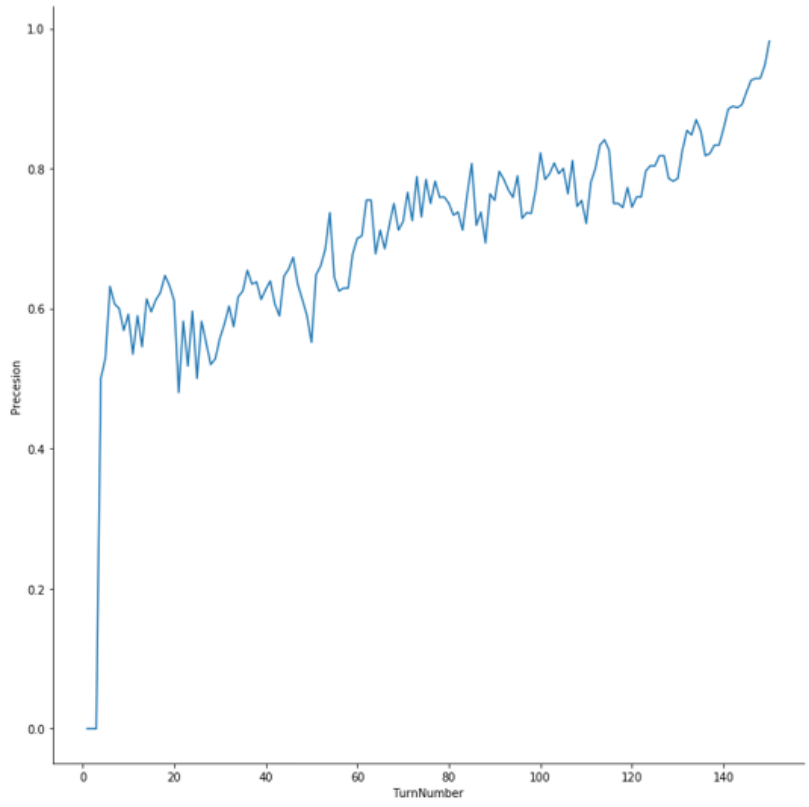
(a) Random Action Vs Random Action Accuracy



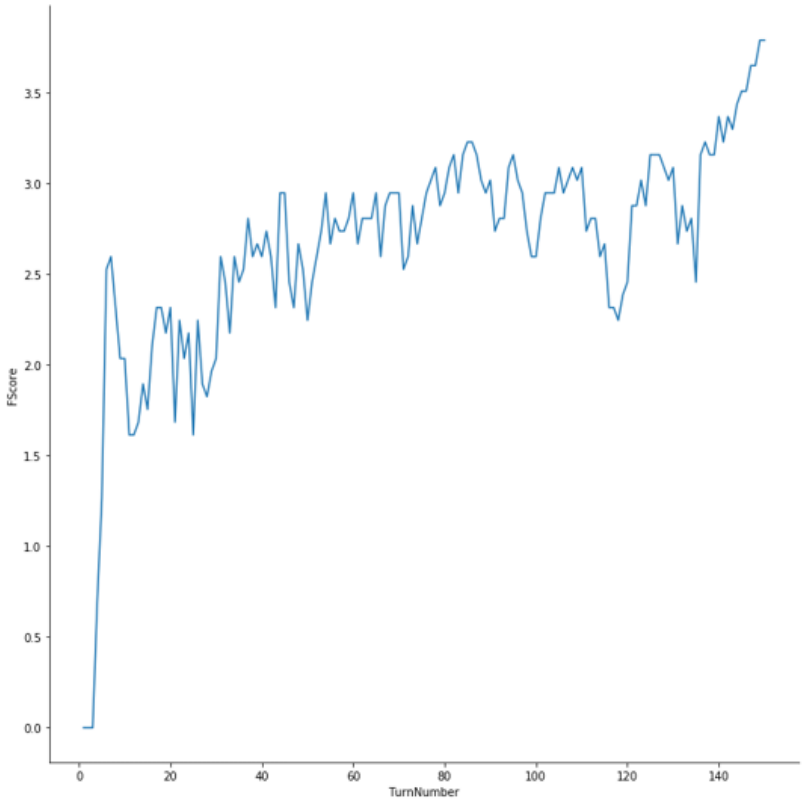
(a) Random Action Vs Random Action Specificity



(a) Random Action Vs Random Action Sensitivity



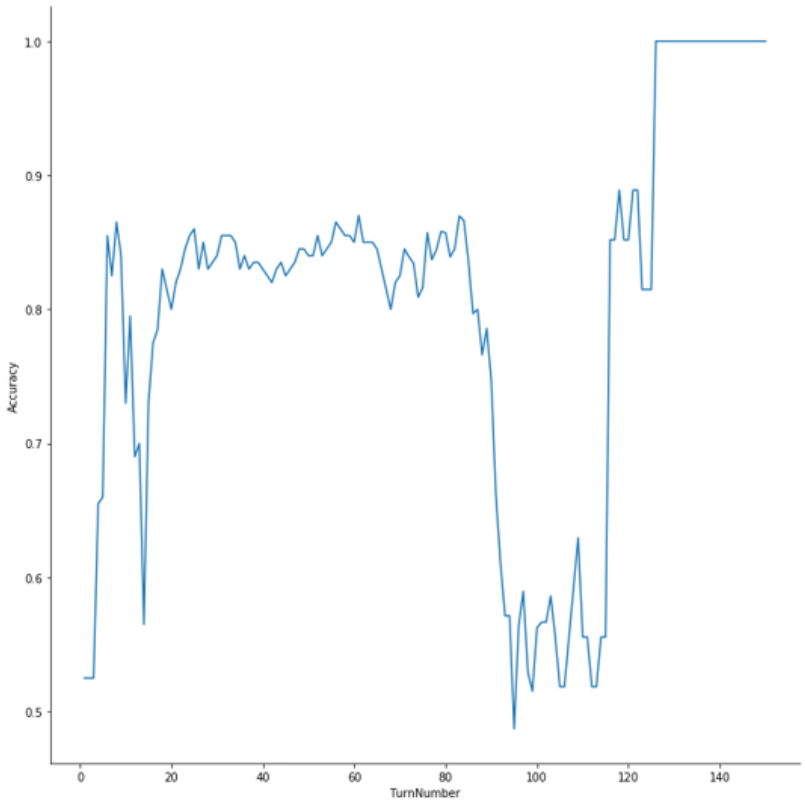
(a) Random Action Vs Random Action Precision



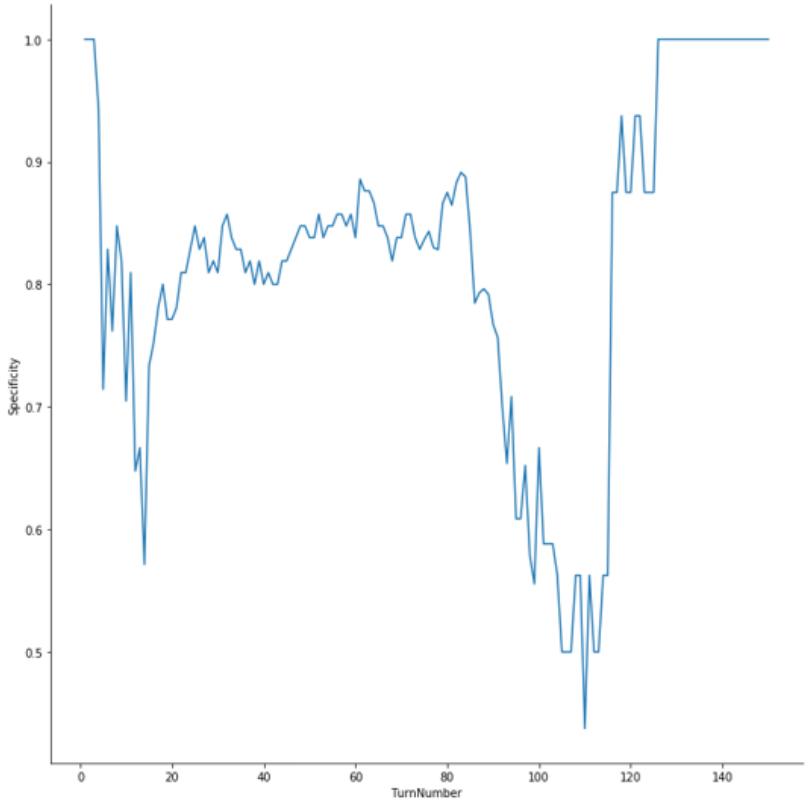
(a) Random Action Vs Random Action FScore

A.1.2 Mid-Game Base Rush vs Random Action

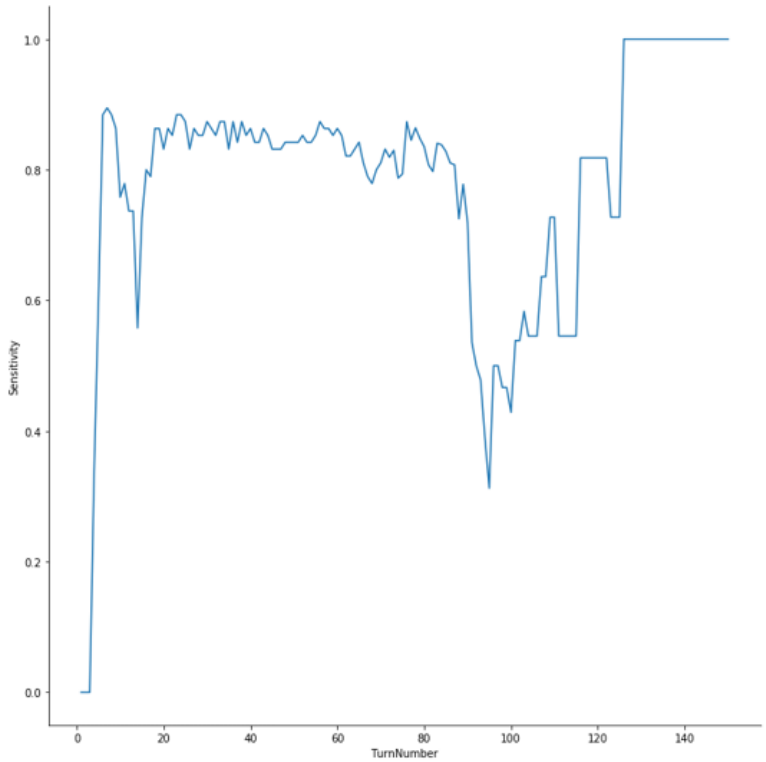
The following figures are from 200 games from Mid-Game Base Rush Agents vs Random Action Agents:



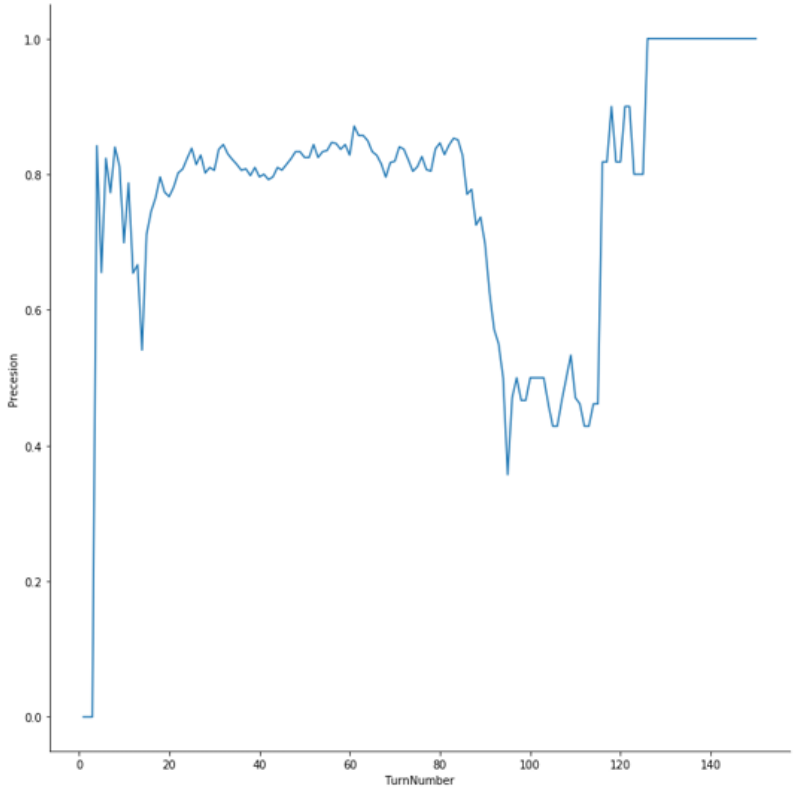
(a) Mid-Game Base Rush Vs Random Action Accuracy



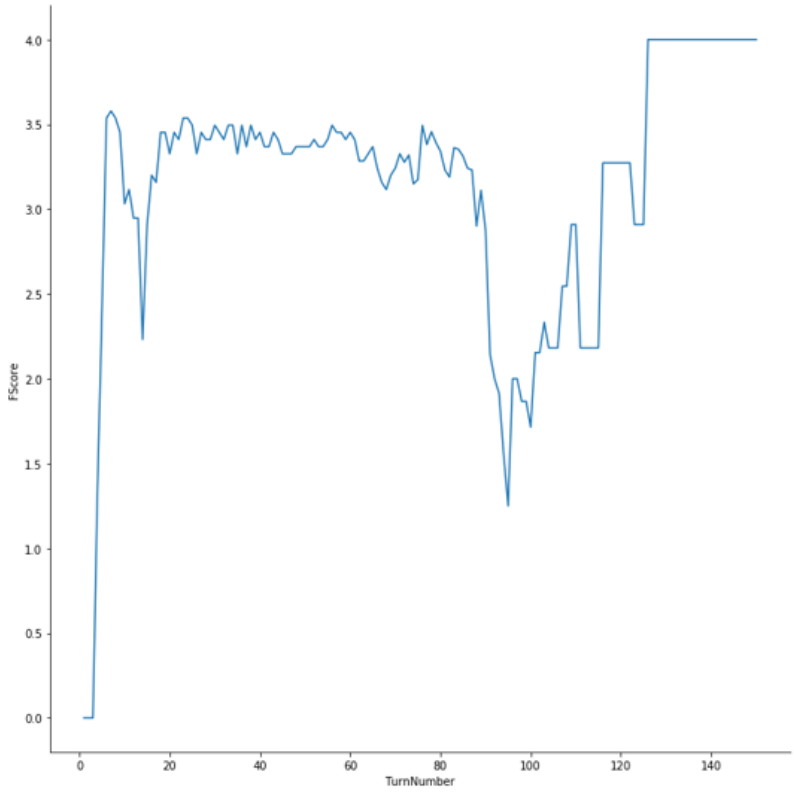
(a) Mid-Game Base Rush Vs Random Action Specificity



(a) Mid-Game Base Rush Vs Random Action sensitivity



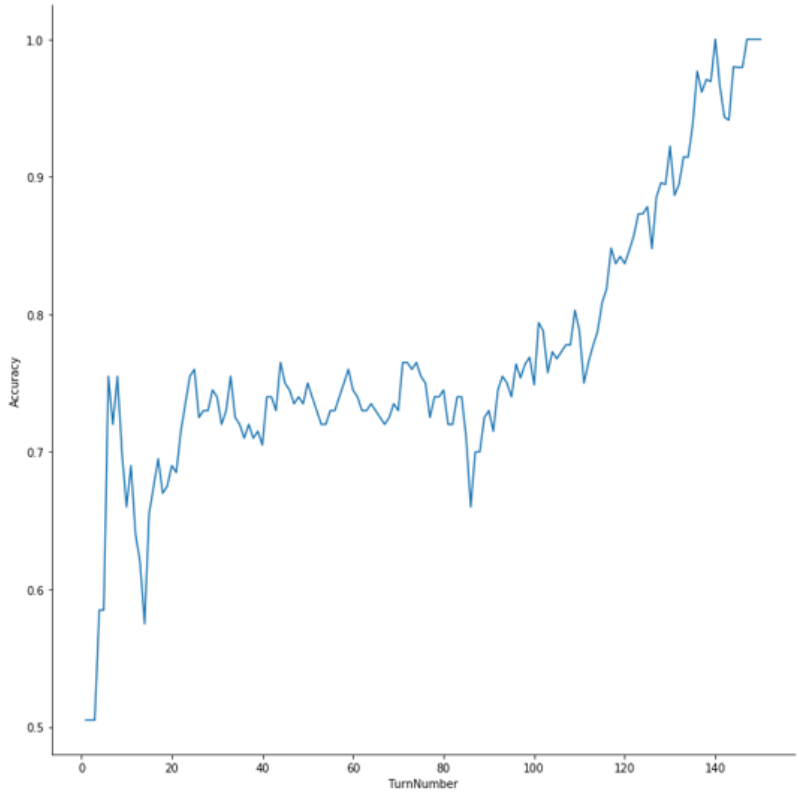
(a) Mid-Game Base Rush Vs Random Action Precision



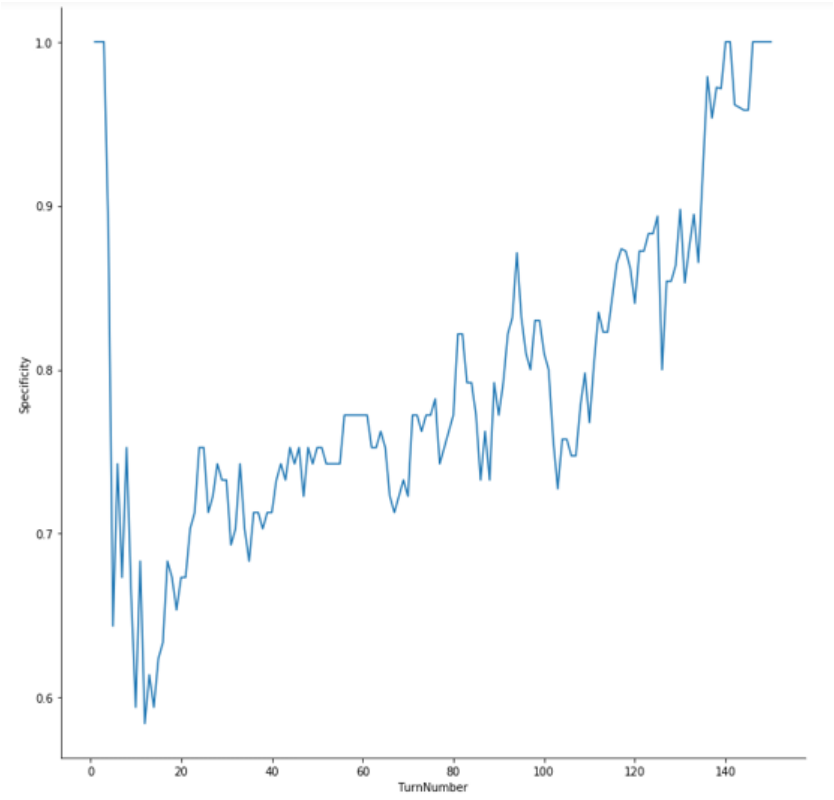
(a) Mid-Game Base Rush Vs Random Action FScore

A.1.3 Late-Game Base Rush vs Random Action

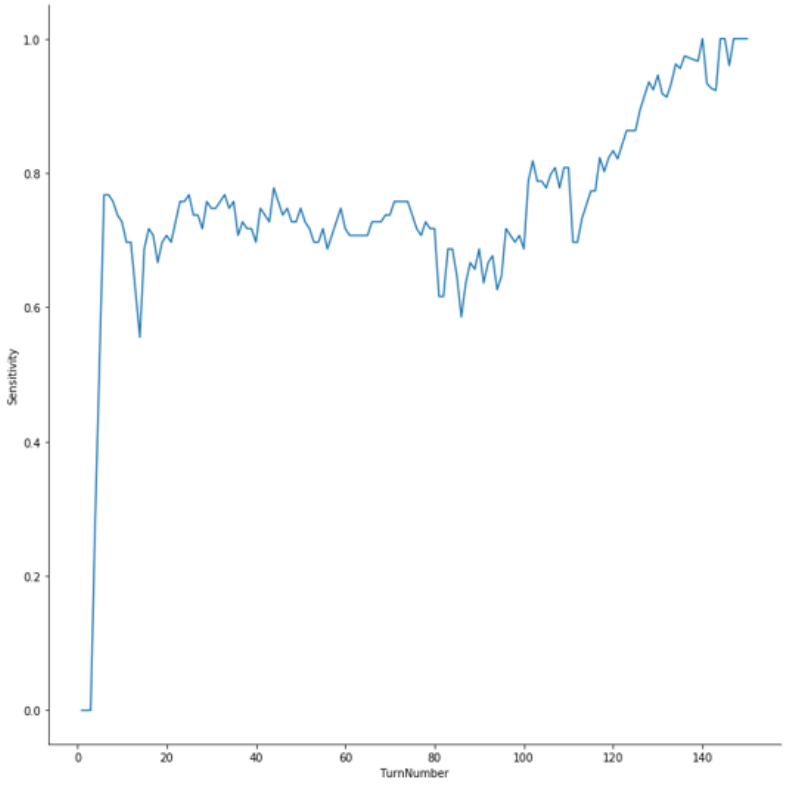
The following figures are from 200 games from Late-Game Base Rush Agents vs Random Action Agents:



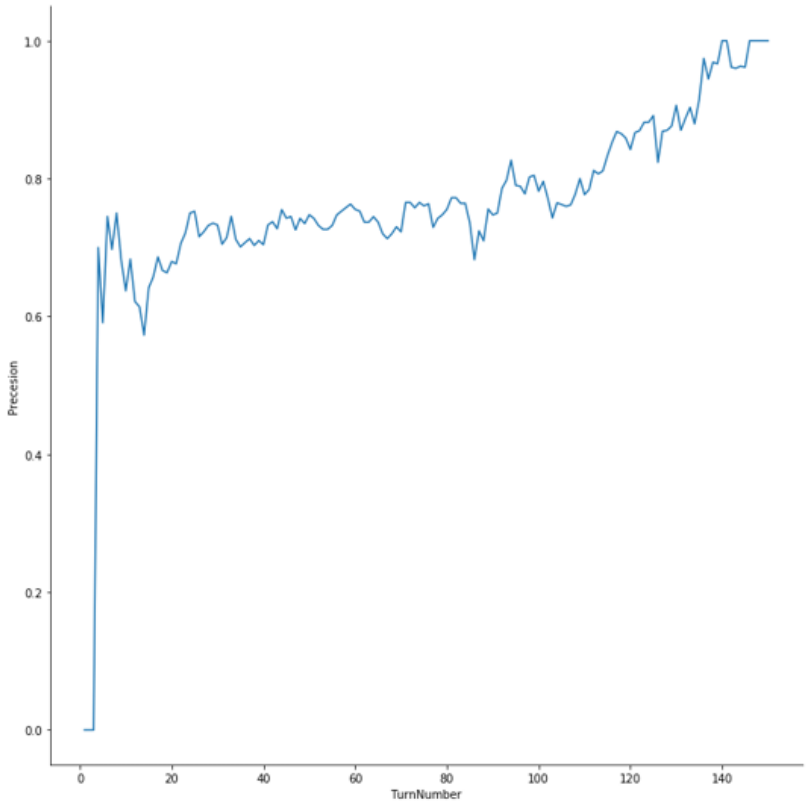
(a) Late-Game Base Rush Vs Random Action Accuracy



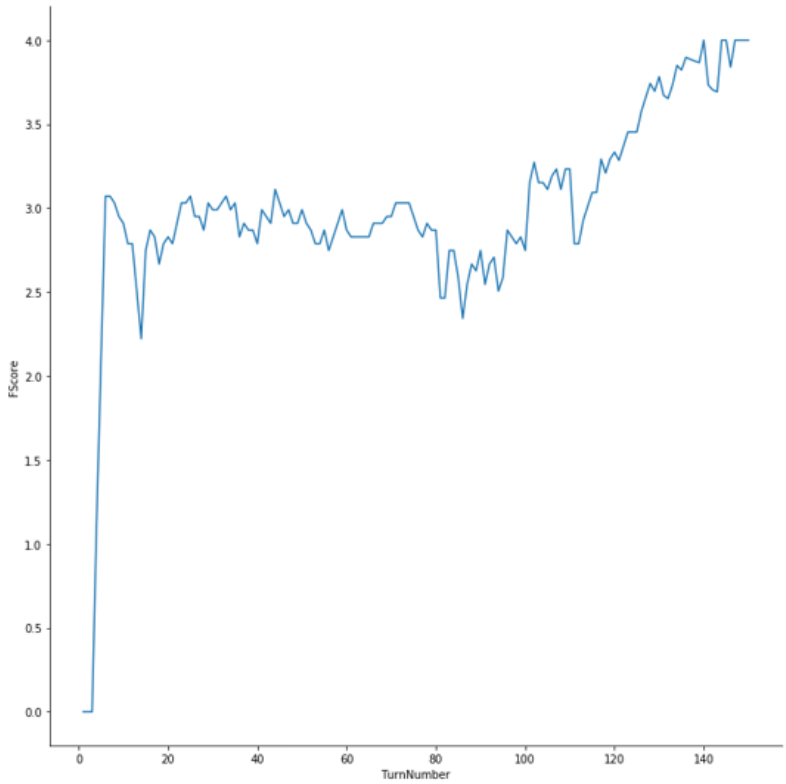
(a) Late-Game Base Rush Vs Random Action Specificity



(a) Late-Game Base Rush Vs Random Action Sensitivity



(a) Late-Game Base Rush Vs Random Action Precision



(a) Late-Game Base Rush Vs Random Action FScore