Zachary Mulder
22453936

Parsing eSports Matches Into Meaningful Data

Introduction
The world of Electronic Sports is growing rapidly every single day. With a wide variety of games being played throughout the world, major tournaments with multimillion dollar prize pools and viewership in the tens of millions, eSports has transformed from a childhood fantasy into a profitable career path. Especially with the advent of video game streaming websites such as twitch.tv, professional gamers and streamers alike can make six figure salaries all from the comfort of their own home. As popularity continues to grow, eSports will only attract more talent and media attention increasing the value of an already profitable industry.

However, unlike other sports, very little statistical analysis is conducted on professional eSports leagues. This is very surprising as the game is carried out on a machine that is capable of recording virtually any desired statistic as opposed to other professional athletics that require manual collection of data by observers. Thus, I undertook this project as a proof of concept to show that data analysis and machine learning can be applied to the realm of eSports. In this project, I focused on the game where I have the most experience, Counter Strike: Global Offensive.

Background
Counter Strike: Global Offensive is a first person shooter where two teams of five members battle to respectively protect and destroy two objectives. Each round, both teams are spawned in a designated area where the defenders are generally closer to the objectives so they have time to prepare a defense (a popular map layout is shown in figure 1). Either team can win a round by eliminating all members of the other team, or by fulfilling their objective. The defending team wins if the clock (one minute and 45 seconds in professional leagues) expires before the attacking team can activate the bomb at either objective site. The attackers win the round if they can activate the bomb and defend it from being disarmed for 35 seconds. Each match is a best of 30 series and team switch sides after playing 15 rounds. Matches can continue into overtime if the score is tied 15 to 15 after 30 rounds where each league has their own rules for overtime play.

While the game is simple on the surface, one of the key difficulties that players face is effective team management of their in game economy. The main driver behind winning a round comes from the equipment that each player purchases at the beginning of the round using funds they've accumulated throughout the half. Equipment includes armor and a helmet, to protect against damage, various weapons, auxiliary grenades, and a defuse kit for players on the defending side that decreases the time required to defuse a planted bomb by half. Money is earned mostly by winning rounds and eliminating enemy players, though the losing team does receive a much smaller amount at the end of a round that increases for each consecutive round lost. This leads to rapidly evolving meta strategies as teams will purchase equipment based on the anticipated equipment of their opponent in the following round. This is one of the key points I wanted to examine in this project as some teams have adopted a strategy called "force buying." That is, when a team has won a few rounds in a row, if they lose a round and are facing a questionable economy, they will try to buy as much equipment as they can afford at the. The hope is that if they win the next round, their opponents, who just had their consecutive losing round bonus reset, will now have very little money and may have to wait a few rounds before making a full

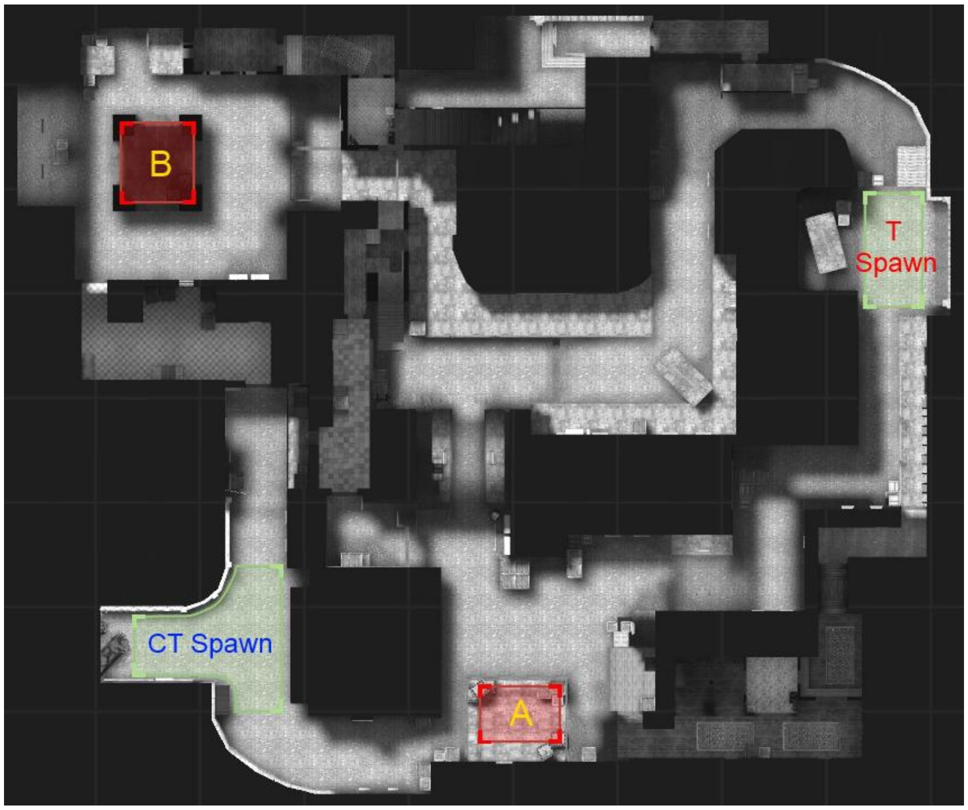


Figure 1:
The layout of a typical map in Counter Strike. Notice that the defending team, the counter terrorists, spawn closer to both objective sites than the attackers. Also notice how there are three or four lanes from spawn to spawn. A typical defense for this map, de_mirage, is two defenders on the A site, one defender watching the middle lane, and two defenders on the B site.

buy. This strategy has been successful for some teams, while not very successful for others, which led me to address this strategy in my analysis.

Another key factor is the coordinated use of auxiliary grenades as a team. There are four main types of auxiliary grenades in the game: flash bangs which temporarily blind and deafen a player, smoke grenades which obscure a medium sized area through smoke, incendiary grenades which cover a medium sized area of the ground with damage dealing fire, and last HE grenades which purely do damage to any player in the blast radius. Using flash bangs as a team can be the difference between easily taking an objective site or instantly losing the round as a well-placed flash bang blinds your entire team leaving your opponents with easy targets. Many teams have multiple set plays, similar to a play in football, where each member will throw flash bangs and smoke grenades into specified areas during a coordinated push to block off vision and blind the enemy players. Thus in this project I wanted to quantify how effective uses of flash bangs and other grenades influenced winning a round. Smoke grenades and incendiary grenades are often used to take control over a certain area, which is more difficult to quantify. So instead, I focused on measuring how effectively a player uses flash bangs by measure the average amount of penalty time a player is blinded per flash bang and how much damage is dealt by a HE grenade on average.

Data

One of the main challenges faced in this project was converting recordings of each match into useable data for analysis. These records, known as demo files, are essentially logs of how the server updated the state space of the game based on the inputs it received from each client every 1/16 of a second. The developer of the game, Valve, published source code a few months ago for a tool to convert demo files into text specifically to encourage statistical analysis; however, the converted demo files are structured in a way that is designed for a server to analyze the inputs and change state variables, not in a way that is simple to extract meaningful data for analysis. Thus, what started as a process I thought would be fairly straight forward quickly spiraled into a much more complex problem.

Since the converted demo files mirrored the changes of the state variables in the server, I created a parser in Java that not only read in the commands to the server, but continually updated a simulation of the match that was structured in a much more data oriented way. This was an incredibly difficult task as very little documentation was provided on the structure of the server information, and most insights came from manually combing through the literal millions of lines of text that were output from each demo file. Essentially, I came to rely on reading in key events as they occurred, storing them in a number of different data structures, and then equating those events with a player as the state space variables related to each player updated in the next server tick. This system worked well for most events as the fine granularity of server updates meant that most server ticks saw only one or fewer events. However, difficulties arose when multiple events happened at the same time. This occurred more often than I expected as sometimes multiple HE grenades would detonate, multiple players would fire their weapons, multiple flash bangs would detonate, or any other combination of the above with a number of other events would occur all in the same server tick. This required more of a heuristic approach where events were peeled off one at a time based on the best guess of what happened by the state variable changes; however, the system is not perfect. The data I collected is not a perfect recreation of the match, but is a fairly accurate representation of how the match progressed. I've included the 3000 lines of Java code needed to parse the converted demo file and simulate the match with the project report. Overall, this portion of the project dominated the majority of the time I spent on the project. While I'm happy with the end result, I underestimated the scope of this project and should have chosen something more reasonable.

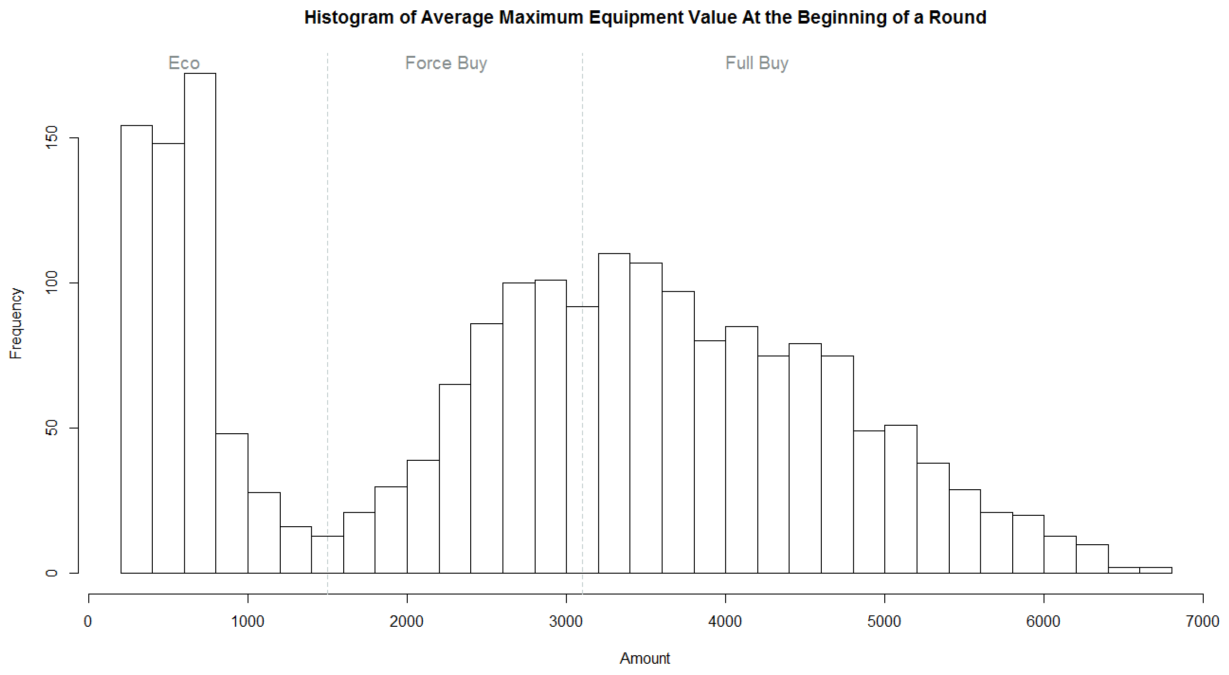
Another issue I faced in the data collecting process is the scarcity and inconsistency of demo files across various leagues. Currently there are a number of independent leagues that most professional teams compete in simultaneously. However, each league uses its own methods for hosting matches, meaning that much of the structure of a demo file that is needed to identify players and information about the match is not consistent from one league to another. Thus, from the onset of the project I chose to tailor the demo parser and analysis to one of the more established leagues, ESEA, as ESEA provides free access to many demo files from the past three months of league play. Additionally, most leagues are divided by region, meaning I had to choose between analyzing the past ESEA season in North America or Europe. I chose the European league as the North American league is much more volatile than their European counterparts. Upsets frequently occur in North America and European teams often completely dominate North American teams during international competition. Thus, I took all matches from the Season 18 European Invite ESEA League that had a demo available, removed any matches with forfeits, and parsed the remaining 42 matches with 9 different teams playing a total to 2056 rounds. These matches combined were approximately 12 GBs of textual data, and took approximately an hour to parse using the provided code. A sample server tick of textual data is provided in appendix A.

Variables

The primary question I sought to answer during my analysis is what factors influence the outcome of a round. I broke up the factors into essentially two categories: the equipment that the team has purchased at the beginning of the round, and indicators of how the team has been performing recently. Even though the European league is much more stable than the North American counterpart, many top tier teams when competing against one another have streaks of both poor and exceptional performance. This is also related to the constantly changing meta game strategies as teams often devise new strategies that take time for other teams to adapt to. Thus I created a model that takes into account both the current economic situation and their recent performance.

First, to represent the in game economy, I chose to categorize the amount of equipment that each team purchased into a discrete set representing the conventional notion of various purchasing strategies. The four categories are first round buy (both teams have very little resources, automatically assigned to the first round of each half), “eco buy” (purchasing very little equipment to save resources for the next round), “force buy” (purchasing as much as possible with the hopes of resetting the other teams economy), and a “full buy” (purchasing essentially all equipment available). Each round, a team receives a buy categorization that comes from the maximum equipment value of each player at the beginning of the round averaged together. I transformed the continuous variable into a discrete set because a change in value is not uniformly significant over the different economy values. For example, in a full buy scenario, having an additional $1,000 in equipment is fairly unimportant, as that is most likely the difference between one or two players upgrading their weapons. However, at an eco buy level, $1,000 could easily be the difference between and eco buy and a force buy. To judge where to define the cutoffs, I plotted a histogram of the average maximum equipment value over the 42 matches and 2,056 rounds. The results can be seen in figure 2.

Figure 2:
Histogram of the average maximum equipment value across all recorded rounds.



The second choice I made is what variables to include as indicators of the team's recent performance. Originally, I established a large number of variables that I thought would quantify the margin of victory. For example, I sought to include detailed information such as the amount of time remaining on the bomb timer when a defender defused the bomb, or how quickly the bomb was planted in a given round. However, many of these variables had very high correlation with one another, which would often produce singular matrixes that were not invertible. Thus, I whittled down the long list of variables into a much smaller subset that captured as much of the information in a round as possible:

First Elimination – A binary variable indicating if the team eliminated a player on the opposing team before any of their own players were eliminated. This is also known as an entry frag. This variable captures how effectively a team is able to attack or hold an objective site, as often the first elimination dictates the success of a push on an objective site.

Number of Opponents Surviving at end of Round – This variable is important to capture as often the winning team will not eliminate all opponents as an opponent may try to hide and survive the round instead of attempting to win the round if they have substantial equipment. This is also an indicator of the strength of the opponent's economy. If a team is consistently eliminating most of their opponents, the opponent's economy will be very low as they have to continually rebuy equipment every round, even if they're winning the rounds. Last, this variable represents how close a loss was. If a team loses the round and a low number of opponents remain at the end of the round, then that would indicate they were close to winning a round as opposed to a complete loss. Note that this variable is normalized to be a ratio between zero and one, where one represents all opponents are still alive.

Number of Teammates Surviving at end of Round – This variable helps quantify how close a round was, and also is an indicator of the team's overall economy. If the team consistently wins rounds with many players alive, then they will have a strong economy. This variable was also scaled to be between zero and one.

Average Accuracy – The game is a first person shooter and ultimately many encounters come down to players simply hitting their shots or not, like many other professional sports. This is a strong indicator for success as often top European teams are unstoppable when their aim is on.

Average Duration of Blinded Players per Flash bang - As I mentioned in the introduction, auxiliary grenades are incredibly important. This variable demonstrates how effective a team is at beginning a push or defending against one. Additionally, effective use of flash bangs often accompanies a strong defense or retake after the bomb has been planted. Scaled such that if during a round the team flashes an opponent for the maximum amount of time possible (about five seconds) with each flash bang they throw, then they will have a value of one.

Average Damage per HE Grenade – Again, this variable represents how effectively a team is using auxiliary grenades. Grenades are often important in the beginning of the round when a team is looking for a first elimination or after the bomb has been planted to eliminate wounded players at the end of the round.

Model:
In many sports, the best indicator of future success is the most recent result. Teams often go through slumps or winning streaks, and a team's more recent performance may be more indicative than

anything else. Thus, we're looking for a way to relate the variables defined above to guess if a team will win the next round. This leads to the following formulation:

$$y = \beta_0\left[x_0\right] + \sum_{j=1}^{p} \beta_j\left[x_0\right] \cdot \left(x^j - x_0^j\right) + \varepsilon$$

Where *x* is a set of measurements that are sufficiently close to the current state of the system x_0, that is $||x_i - x_0||<h$ for some $h>0$). Many different interpretations of distance can be used, but this project focuses on the difference in time between measurements. Thus, based on the input x_0 and the tuning parameter h (known as bandwidth), we need to calculate the optimal β, and β_0 that minimizes the sum of squared error. This problem can be reformulated as:

$$\begin{bmatrix} \hat{\beta}_0[x_0] \\ \hat{\beta}[x_0] \end{bmatrix} = \underset{\beta_0,\beta}{\arg\min} \sum_{i=1}^{n} K(\|x_i - x_0\| / h) \cdot (y_i - \beta_0 - (x_i - x_0)'\beta)^2$$

Where x_0 contains the indicator variables representing the amount purchased at the beginning of the current round and the other variables specified above based on the previous round. Y_i is an indicator representing if round *i* was won, and $||x_i - x_0||$ is a function representing the difference in time between the measurement x_i and x_0, and K is a kernel function. Multiple kernels such as the Epanechnikov kernel, quartic kernel, and triweight kernels were all tested and did not have a significant effect on the output of the model, so assume an Epanechnikov kernel is used throughout the rest of this project. This model can then be rewritten as:

$$\begin{bmatrix} \hat{\beta}_0[x_0] \\ \hat{\beta}[x_0] \end{bmatrix} = \underset{\beta_0,\beta}{\arg\min} \left\| W_h^{1/2}(Y - \left[1_n X_0\right] \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix} \right\|_2^2$$

Which is solved by:

$$\begin{bmatrix} \hat{\beta}_0[x_0] \\ \hat{\beta}[x_0] \end{bmatrix} = \left(\left[1_n X_0\right]' W_h \left[1_n X_0\right]\right)^{-1} \left(\left[1_n X_0\right]' W_h Y\right)^{-1}$$

Where $X_0=X-x'_0 1_n$.

This leads to two design decisions in the model: the bandwidth *h*, and a function to determine the distance in time between two rounds. I tested two different functions in this project, one that is a linear scaling based on the difference in the number of rounds, and another that attempts to take into account the number of days between matches. The first function is fairly straight forward:

$$\|x_i - x_0\| = f_1\left(x_i, x_0\right) = \left(1 - \frac{i}{r(x_0)}\right)$$

Where $r(X_0)$ is the row of x_0 assuming the rows are ordered in chronological order. This function can also be seen as a merely a ratio of how long ago the i-th round was relative to x_0, where rounds that were a long time in the past have a higher value of then those that are more recent (which means the kernel function weights values that are closer in time). The drawback with this function is that regardless if the last match was two days or two weeks ago, the distance between a round in the current match and the last match is always the same. The other function used is a little more complex, and takes into account the difference in days between matches:

$$\|x_i - x_0\| = f_2(x_i, x_0) = \frac{x_0^d - x_i^d}{x_0^d - x_1^d} + \left(1 - \frac{x_i^r}{x_i^p}\right) \qquad \text{if } x_0^d - x_i^d > 0$$

$$\|x_i - x_0\| = f_2(x_i, x_0) = \left[\min_i \left\{ \frac{x_0^d - x_i^d}{x_0^d - x_1^d} + \left(1 - \frac{x_i^r}{x_i^p}\right) \middle| \left(x_0^d - x_i^d\right) > 0 \right\}\right] \cdot \left(1 - \frac{x_i^r}{x_0^p}\right) \qquad \text{otherwise}$$

Where x^d is the day of the i-th match, x^r is the round number of the i-th match played on that day, and x^p is the total number of rounds played the day of the i-th match. The second function is essentially a composite of two different functions, one for when matches are not on the same day, and a second for when multiple matches are held in the same day. When a previous match is on a different day, the day portion is weighted by the differences in days between matches over the total timespan of the season for the team modified by the ratio of the cumulative number of rounds played by the i-th match over the total number of rounds played that day. If the previous match occurs on the same day, then the rounds that day are essentially weighted in the same way as function one, but they are scaled by minimum distance of the previous day. That way the function is generally decreasing in distance as matches are read forward chronologically.

The other design parameter in this model is the bandwidth, *h*. In order to determine the optimal bandwidth to minimize mean squared error, I used a time series variant of cross validation where I would begin with some initial value, say m, and use data points (x_i, y_i) for i=1,…,m-1 to determine the value of y_m based on the input x_m. I would then compute the mean squared error, e_m, and continue the process with m+1, continuing until I reached n-1. I would then compute the mean squared error as the average of all the individual errors. Typically, a starting place of m=4n/5 was used, or approximately 80% of the data was used to fit the model, and the remaining 20% was used for validation. Each team was treated independently of the others, and the opponent was not taken into account. The results from the experiments can be seen in figures 3-5.

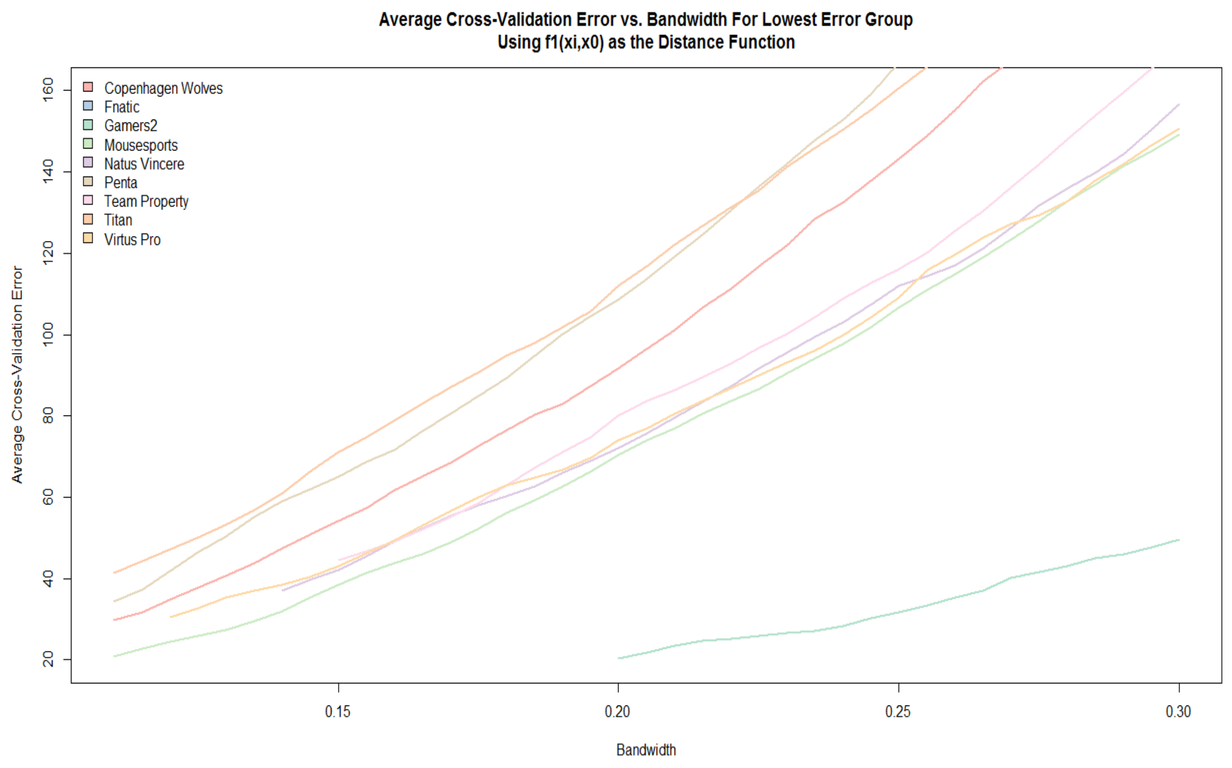


Figure 3:
Average cross-validation error when using the first time distance function.

Results

For experiments involving the first time difference function, the average cross validation error decreases in bandwidth. However, each team reaches a cutoff where the inverted term in the solution becomes singular and can no longer be calculated, which could potentially be remedied by recording more data. Each team also has a fairly similar shaped error curve, except for Gamers2, who have approximately 50 fewer rounds of data than the other teams. This could contribute to the higher bandwidth since the number of rounds is lower and requires less scaling than the other teams. This result shows that a linear scaling of the differences in rounds produces a very consistent bandwidth selection, and most likely will lead to a more uniformly applicable results as further data is collected.

On the other hand, the results for the second time difference function are a little bit less promising, as shown in figures 4-5. Again, Gamers2 is the team with the lowest average cross validation error; however, this time they have a much lower bandwidth than the next set of teams. This would indicate that more information is being excluded and there is a greater focus on recent matches than with other teams, which seems to contradict the previous result with the other time difference function. However, the differences in bandwidth at each stage are most likely due to a scaling compensation since both time difference functions rely on the overall length of the time horizon and the number of rounds played. Additionally, the shape of the curve seems to indicate that instead of rounds being spaced apart linearly in distance like the other time difference function, the difference in time between rounds is more clustered. This is indicated by the sharp decreases in error as a large number of rounds are dropped in rapid succession as bandwidth decreases.

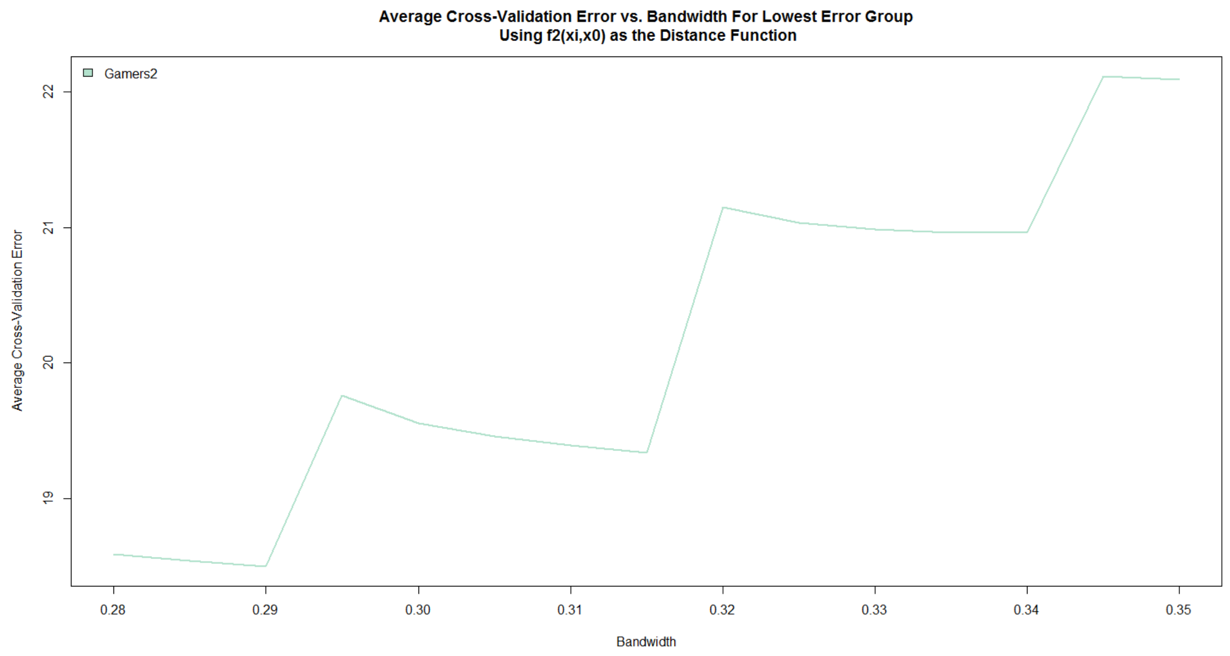


Figure 4:
Average cross-validation error when using the second time distance function. Note that Gamers2 was plotted alone because their average error was much lower than any other teams error term.

Alternatively, every other team had a much higher validation error when the second time difference function was used, as seen in figure 5. Two teams, Titan and Virtus Pro, had such a large change that they no longer could be represented on a graph with the other teams. This result indicates that the second time difference function is often outperformed by the simpler alternative of linear scaling. Even in certain circumstances when the second time difference function outperforms the linear method, there is not a significant difference in value.

Additionally, the results with the two time distance functions seems to indicate that large gaps in schedules do not necessarily have a large impact on a team's performance. This could also be attributed to the many other matches that most professional teams play across leagues on a daily basis. Very rarely do teams take a break from practices or matches, which would also support the result that a linear scaling time difference is a reasonable and accurate model.

Last, the average cross-validation error is very high for predicting either a value of one or zero. I noticed this was due to a relatively large number of values being used in the aggregate guess for y. Increasing the amount of data collected would allow the bandwidth to decrease without creating a singular matrix, and thus would lead to a decrease in the average cross-validation error as less values are used for the aggregate guess of y.

Extensions

As a proof of concept, I've shown that sufficiently useable data can be extracted from the recorded demos of professional eSports matches and analyzed to better understand the trends in the game. While the work I performed was very valuable both in terms of personal programming development and statistical analysis, there are many extensions that would add more depth to this project in the future:

Increased Data – As I mentioned before, professional teams often play multiple matches a day across a number of different leagues. Expanding the demo parser I created to accommodate a number of different leagues would lead to higher accuracy models and a more comprehensive image of the professional eSports scene.

Accounting for Map – One of the biggest factors in a professional match is what map is played. Teams often have a pool of maps that they consistently practice and choose for matches, meaning that they have better strategies and more comfortable on some maps as opposed to others. Accounting for the map being played would add a very critical piece to the model.

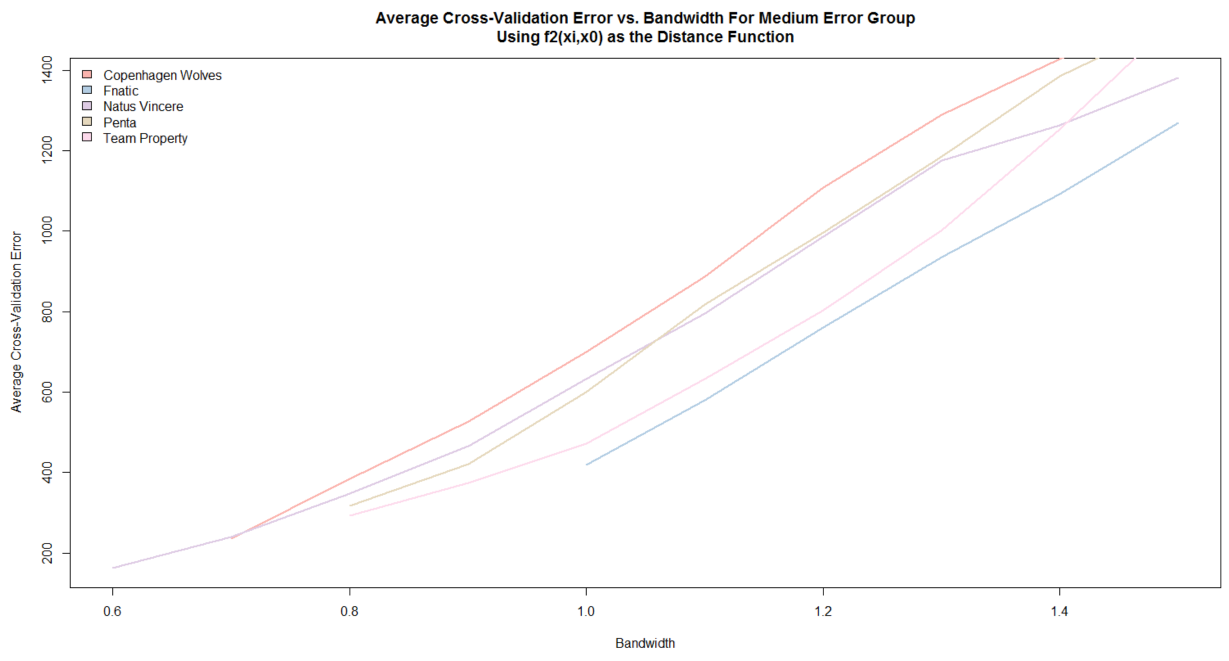
Accounting for Opponent – The most prominent feature in any matchup is who you're playing. As with many other sports, each team has their own unique play styles. Some teams have elaborate coordinated set plays to push their way onto an objective, whereas other teams are incredible aimers and rely on their raw skills to win matches. Your own team's play style coupled with the playstyle of your opponent can have a drastic effect on a match and would be a useful addition to the model.

Alternative Modeling Techniques – An interesting approach to characterizing the system would be as a control feedback loop. During a round, a player can only control their own moves and has to base their play off the state of their teammates and opponents. Once a move is made, the other players in the game make their own moves, the system is updated, and another move can be played. Alternative formulations could lead to interesting insights on some of the mechanics I established in this project.

Conclusion

The world of eSports is growing at an immense rate every day. As competition continues to intensify, teams and players will start looking towards new ways of analyzing and seeing the game. This creates a large opening in the field to produce groundbreaking and meaningful results as the industry begins to see the significance of data driven analysis. I've shown in this project as a proof of concept that such analysis is feasible, and I will actually continue to improve and refine the tools I've created for this project to show how meaningful analysis of this kind can be to the eSports industry.

Figure 5:
Average cross-validation error when using the second time distance function. The error terms are much higher using the second time distance function as opposed to the first.



Appendix A – Sample Server Tick (16 per second)

```
---- CNETMsg_Tick (12 bytes) -----------------
tick: 136051
host_computationtime: 4914
host_computationtime_std_deviation: 1169
host_framestarttime_std_deviation: 85
Entity Delta update: id:1, class:34, serial:365
Table: DT_CSPlayer
Field: 1, m_nTickBase = 136051
Entity Delta update: id:2, class:34, serial:781
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 51
Field: 1, m_nTickBase = 136052
Field: 2, m_vecOrigin = 1096.165894, -1489.290039
Field: 4, m_vecVelocity[0] = -120.744820
Field: 5, m_vecVelocity[1] = -119.586380
Field: 7, m_vecOrigin = 1096.165894, -1489.290039
Field: 18, m_angEyeAngles[0] = 0.703125
Field: 19, m_angEyeAngles[1] = 224.296875
Field: 22, m_flGroundAccelLinearFracLastTime = 1062.898438
Field: 23, m_cycleLatch = 6
Entity Delta update: id:3, class:34, serial:839
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 55
Field: 1, m_nTickBase = 136056
Field: 2, m_vecOrigin = 589.897888, 466.319092
Field: 3, m_vecOrigin[2] = -520.509338
Field: 4, m_vecVelocity[0] = 33.920345
Field: 5, m_vecVelocity[1] = 50.227203
Field: 7, m_vecOrigin = 589.897888, 466.319092
Field: 8, m_vecOrigin[2] = -520.509338
Field: 23, m_cycleLatch = 0
Entity Delta update: id:4, class:34, serial:773
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 58
Field: 1, m_nTickBase = 136059
Field: 2, m_vecOrigin = 336.840912, -1088.759644
Field: 3, m_vecOrigin[2] = -254.136307
Field: 4, m_vecVelocity[0] = 13.066470
Field: 5, m_vecVelocity[1] = -197.714493
Field: 7, m_vecOrigin = 336.840912, -1088.759644
Field: 8, m_vecOrigin[2] = -254.136307
Field: 18, m_angEyeAngles[0] = 33.046875
Field: 19, m_angEyeAngles[1] = 278.437500
Field: 22, m_flGroundAccelLinearFracLastTime = 1062.953125
Entity Delta update: id:5, class:34, serial:152
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 51
Field: 1, m_nTickBase = 136052
Field: 2, m_vecOrigin = 88.557991, -1197.625366
Field: 4, m_vecVelocity[0] = -33.087921
Field: 5, m_vecVelocity[1] = -208.589767
Field: 7, m_vecOrigin = 88.557991, -1197.625366
Field: 22, m_flGroundAccelLinearFracLastTime = 1062.898438
Entity Delta update: id:6, class:34, serial:470
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 51
Field: 1, m_nTickBase = 136052
Field: 2, m_vecOrigin = 738.151489, -576.096680
Field: 4, m_vecVelocity[0] = 8.613561
Field: 5, m_vecVelocity[1] = -9.472559
Field: 7, m_vecOrigin = 738.151489, -576.096680
Field: 18, m_angEyeAngles[0] = 355.781250
Field: 19, m_angEyeAngles[1] = 219.375000
Field: 22, m_flGroundAccelLinearFracLastTime = 1062.898438
Field: 23, m_cycleLatch = 9
Entity Delta update: id:7, class:34, serial:341
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 55
Field: 1, m_nTickBase = 136056
Field: 2, m_vecOrigin = 210.876282, -1284.886597
Field: 4, m_vecVelocity[0] = 136.680557
Field: 5, m_vecVelocity[1] = 20.196390
Field: 7, m_vecOrigin = 210.876282, -1284.886597
Field: 18, m_angEyeAngles[0] = 3.867188
Field: 19, m_angEyeAngles[1] = 354.023438
Field: 22, m_flGroundAccelLinearFracLastTime = 1062.929688
Field: 23, m_cycleLatch = 5
Entity Delta update: id:8, class:34, serial:721
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 54
Field: 1, m_nTickBase = 136055
Field: 2, m_vecOrigin = -392.867004, -938.766052
Field: 4, m_vecVelocity[0] = 0.000000
Field: 5, m_vecVelocity[1] = 0.000000
Field: 7, m_vecOrigin = -392.867004, -938.766052
Entity Delta update: id:9, class:34, serial:280
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 53
Field: 1, m_nTickBase = 136054
Field: 2, m_vecOrigin = 520.843201, -1210.014038
Field: 4, m_vecVelocity[0] = 182.619568
Field: 5, m_vecVelocity[1] = 78.343468
Field: 7, m_vecOrigin = 520.843201, -1210.014038
Field: 18, m_angEyeAngles[0] = 346.289063
Field: 19, m_angEyeAngles[1] = 89.648438
Field: 22, m_flGroundAccelLinearFracLastTime = 1062.914063
Entity Delta update: id:10, class:34, serial:920
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 55
Field: 1, m_nTickBase = 136056
Field: 2, m_vecOrigin = 1836.430298, -1707.787354
Field: 4, m_vecVelocity[0] = -194.846207
Field: 5, m_vecVelocity[1] = -40.037720
Field: 7, m_vecOrigin = 1836.430298, -1707.787354
Field: 22, m_flGroundAccelLinearFracLastTime = 1062.929688
Entity Delta update: id:11, class:34, serial:12
Table: DT_CSPlayer
Field: 0, m_flSimulationTime = 55
Field: 1, m_nTickBase = 136056
Field: 2, m_vecOrigin = 90.845116, -1118.738525
Field: 4, m_vecVelocity[0] = 103.531563
Field: 5, m_vecVelocity[1] = -187.610428
Field: 7, m_vecOrigin = 90.845116, -1118.738525
Field: 17, m_flFOVTime = 1062.898438
Field: 18, m_angEyeAngles[0] = 6.679688
Field: 19, m_angEyeAngles[1] = 174.375000
Field: 22, m_flGroundAccelLinearFracLastTime = 1062.929688
Field: 311, m_hLastWeapon = 1728659
Field: 584, m_hActiveWeapon = 1270247
Field: 623, m_iAddonBits = 204
Field: 624, m_iPrimaryAddon = 13
Entity Delta update: id:85, class:112, serial:184
Table: DT_PredictedViewModel
Field: 8, m_nAnimationParity = 5
Field: 10, m_nNewSequenceParity = 6
Field: 11, m_nResetEventsParity = 6
```

```
Entity Delta update: id:87, class:112, serial:432
Table: DT_PredictedViewModel
Field: 8, m_nAnimationParity = 6
Field: 10, m_nNewSequenceParity = 7
Field: 11, m_nResetEventsParity = 7
Entity Delta update: id:95, class:112, serial:595
Table: DT_PredictedViewModel
Field: 8, m_nAnimationParity = 7
Field: 10, m_nNewSequenceParity = 2
Field: 11, m_nResetEventsParity = 2
Entity Delta update: id:98, class:112, serial:700
Table: DT_PredictedViewModel
Field: 8, m_nAnimationParity = 2
Field: 10, m_nNewSequenceParity = 1
Field: 11, m_nResetEventsParity = 1
Entity Delta update: id:103, class:112, serial:954
Table: DT_PredictedViewModel
Field: 8, m_nAnimationParity = 2
Field: 10, m_nNewSequenceParity = 2
Field: 11, m_nResetEventsParity = 2
Entity Delta update: id:106, class:112, serial:483
Table: DT_PredictedViewModel
Field: 0, m_nModelIndex = 350
Field: 1, m_hWeapon = 1270247
Field: 4, m_nSequence = 0
Field: 8, m_nAnimationParity = 4
Field: 10, m_nNewSequenceParity = 7
Field: 11, m_nResetEventsParity = 7
Entity Delta update: id:147, class:205, serial:844
Table: DT_WeaponGalilAR
Field: 402, m_fEffects = 4257
Field: 457, m_iState = 1
Entity Delta update: id:153, class:205, serial:378
Table: DT_WeaponGalilAR
Entity Delta update: id:161, class:9, serial:435
Table: DT_BaseCSGrenadeProjectile
Field: 0, m_flSimulationTime = 51
Field: 2, m_cellX = 541
Field: 3, m_cellY = 460
Field: 6, m_vecOrigin = 2.500000, 9.687500, 8.000000
Field: 7, m_angRotation = 258.442383, 0.000000, 178.110352
Field: 98, m_vecVelocity = -639.546631, -637.108521, 73.682373
Entity Delta update: id:328, class:1, serial:237
Table: DT_WeaponAK47
Entity Delta update: id:332, class:219, serial:494
Table: DT_WeaponP250
Field: 8, m_fAccuracyPenalty = 0.009100
Entity Delta update: id:336, class:202, serial:509
Table: DT_WeaponFiveSeven
Field: 8, m_fAccuracyPenalty = 0.009100
Entity Delta update: id:338, class:9, serial:968
Table: DT_BaseCSGrenadeProjectile
Field: 0, m_flSimulationTime = 51
Field: 6, m_vecOrigin = 5.437500, 12.562500, 2.500000
Field: 7, m_angRotation = 22.412109, 0.000000, 240.908203
Field: 98, m_vecVelocity = -158.985474, -78.751892, -69.146149
Entity Delta update: id:346, class:38, serial:301
Table: DT_WeaponDEagle
Field: 8, m_fAccuracyPenalty = 0.015457
Entity Delta update: id:348, class:9, serial:922
Table: DT_BaseCSGrenadeProjectile
Field: 0, m_flSimulationTime = 51
Field: 2, m_cellX = 533
Field: 6, m_vecOrigin = 14.437500, 7.312500, 11.500000
Field: 7, m_angRotation = 277.646484, 0.000000, 90.922859
Field: 98, m_vecVelocity = 353.329681, -167.682281, 84.151299
Entity Delta update: id:352, class:76, serial:964
Table: DT_FuncRotating
Field: 7, m_angRotation[1] = 168.793945
Field: 9, m_flSimulationTime = 51
Entity Delta update: id:481, class:28, serial:546
Table: DT_WeaponC4
Entity Delta update: id:485, class:227, serial:339
Table: DT_WeaponSSG08
Entity Delta update: id:487, class:88, serial:620
Table: DT_WeaponKnife
Field: 402, m_fEffects = 4225
Field: 457, m_iState = 2
---- CSVCMsg_TempEntities (11 bytes) -----------------
num_entries: 1
entity_data: "\302\236\377\177\001\014\264"
player_blind eventid:164
 userid: 5
player_blind eventid:164
 userid: 6
player_blind eventid:164
 userid: 13
flashbang_detonate eventid:151
 userid: 14
 entityid: 338
 x: 673.486267
 y: -1525.389771
 z: -412.927612
```