MLND Capstone Report

Predicting Victory in Online Team-Based Zero-Sum Games

Aaron Dharna

July 9th - August 13th

Beginnings and Definitions

As the popularity of E-Sports has grown dramatically, and approaches that of conventional sports, the prize pools have approached larger and larger sums. In 2016 *League of Legends* World Championship, the total prize pool was 6.7 million dollars. Likewise, at the 2016 *Defense of the Ancients* International tournament, the total prize pool was 20 million dollars. In both of these tournaments all of this money hinged upon how well the teams that were competing for it managed to play together, how much did the play-styles of the different players synergize.

The initial question that this paper was going to look at was "can we build a team of players whose play-styles complement each other when given clusters of play-styles"? Can you pull players from across multiple clusters to form a single well-balanced team? In short, how can you use clustering methods to enhance matchmaking? The central question was: "How can one build a good team in the matchmaking process using player data?", and the lynchpin in that question is "what is the definition of 'good'?"

Building out a cluster-based matchmaking system has the ability empower and create teams of players. If these players synergize well together, it would not only improve the play experience for many players, but could also build teams who rival professionals. This paper uses 2013's *Destiny* as a vehicle to examine this claim.

According to Jon Weisnewski, a senior gameplay designer at Bungie, the twin pillars of *Destiny* -- a first-person shared-world shooter video game -- are "synchronicity" and "Player Fantasy." The former meaning that one's guardian (player character) should feel the same no matter whether you play in single or multiplayer modes. The latter meaning that when and how you play should evoke or fulfill certain feelings. These pillars are reflected in everything from the weapon design (i.e. The Last Word -- a revolver that evokes the feeling of being a gunslinger from the Old West) to the Raids -- 6-person multi-hour team-activities -- design (i.e. The Vault of Glass). *Destiny* is designed to impart feelings of empowerment onto the player. Something almost as fundamental to *Destiny's* design is the idea of the team. Teamwork as a design philosophy is reflected in everything from the Player versus Environment (PvE) game design to the Player versus Player (PvP) game design. Most strikingly, in order to *even participate* in the end-game[1] content of *Destiny*, a pre-formed team is required on the part of the players. This in turn locks the majority of players away from the best gear and guns in the game if they do not possess a team.

The goal for this project was to use player data and machine learning to promote individual learning and lasting group creation through the use of clustering algorithms to enhance matchmaking and promote inherent design principles within the game. An application of such a system would be to open up complex

[1] End Game content is content that is specifically developed to be played once players have maxed out their ability progression. An example of PvE end-game content are the Raids as these activities can only be *attempted* once the player has reached a specific level. An example of a PvP end-game activity is The Trials of Osiris (ToO). ToO is a competitive tournament-esque game mode. Both of these end-game activities reward the players who play them with the best gear and guns in the game.

game modes -- ones that require multiple skill sets to deftly complete -- to the average (and often alone) player. As such, the problem to be solved has multiple parts:

1) Feature creation: Using player match-history data, create behavioral/play-style clusters[2] (Drachen et al. 2016).
2) Using a Recurrent Neural Network (of which an LSTM is a specific type) to analyze team-structure and build up teams of players.
3) Use an MultiLayer Perceptron (MLP) network to predict which team will win.

To optimally understand this paper, knowledge of neural networks is required. The topic will be briefly touched upon, but only as a refresher.

Due to time and financial concerns, I determined it would be best for this capstone project to instead focus on only one portion of this initial scope. Further enhancements will be worked on post-graduation. Therefore, from here on out the focus will be on what I am calling *Victory Net* -- a neural network which will take in two teams of players and determine how likely team Alpha is to win the competition based on player and match statistics.

The metric that makes most sense for this project is accuracy. Given the framework where you have the number of predictions by the model classified into:

$$Tp = TruePositive$$
$$Fp = FalsePositive$$
$$Tn = TrueNegative$$
$$Fn = FalseNegative$$

Accuracy is defined as:

$$Tp + Tn/(AllPositive + AllNegative)$$

Due to the nature of the game being zero-sum, this makes the classification problem (Did team A win or lose) generate balanced classes. For each victory that the algorithm sees, it will see a corresponding defeat. Furthermore, the number of examples of each class in the data set is balanced. As shown in the confusion matrix at the results section of the paper the support for each class is roughly 12,000 examples.

Every time a person plays a PvP match, information is recorded about that player as well as any other players in the match. The data collected is typically descriptive of each player's performance, as well as some engagement information such as the duration of the match. Although there are many different types of PvP matches in *Destiny*, the data tracked in each match is similar. As such, we can treat each of the fundamentally different match-types in the same manner and extract similar statistics no matter the match-type.

[2] https://goo.gl/qClnPU This paper here by Dr. Anders Drachen provided the inspiration for this project.

Before the proposal section, I reached out to Dr Anders Drachen since he has done similar research into the applications of machine learning to video-games. When talking with him about my project he agreed to let me use the dataset that he has used in his research. However, now that the scope of my project for Udacity has changed, I will instead be using a smaller, but modelled-after-Dr-Drachen's, dataset. My smaller data set was built up by successive calls to Bungie's public-facing API. That code has been included in my submission/Github.

Preliminary Analysis

To rebuild this data set requires two -- for lack of a better word since it will be used in just a second -- keys. First, an API key to allow access to Bungie's servers, and secondly a list of match-IDs. With both of these pieces of information and the request library, you can ask Bungie for the PostGameCarnageReport[3] (PGCR) of a specified match. This server call will return a JSON object completely describing the player (and match) level statistics for the specified match.

The features present in the returned PGCR JSON are both broad (team-wide) and granular (individualized) descriptors; however, many of the individual level statistics are hyper-focused. As such, a drastic dimensional reduction can take place off the bat by combining the too granular statistics. As an example, there exists in the PGCR not only how many kills did each participating character achieve in the match, but the kill count is further broken down into "how many players using class A/B/C did this character kill?" Likewise the death statistic can, and is, again broken into "how many players using class A/B/C killed this person?" Therefore, the PGCR can be reduced by a large degree by using the more-general-individual-features and by combining the highly-specific-individual-features into slightly more general ones. For example, rather than recording each of the different kills a player achieved with each possible different primary weapon -- "weaponKillsAutoRifle", "weaponKillsScoutRifle", etc. -- instead combine all four primary weapon kills into weaponKillsPrimary for each person for each match. While this method does result in some information loss, the information can be regained by looking, in conjunction, with other statistics. If a person has a high number of "weaponKillsSecondary" and a low "averageKillDistance" then that person is likely using a short-range based playstyle reliant on shotguns and revolvers.

The attached link here describes the PGCR by breaking it up into three parts.

1) The complete tree structure stats-breakdown of a PGCR.
2) A complete list of all reachable endpoints in the match for a single character (might vary based on weapons used or by which game mode your specific PGCR is).
3) The cleaned data set of features that I extracted for each person for each match.

With that out of the way, the features that were extracted for each player for each match are:

[3] https://github.com/DestinyDevs/BungieNetPlatform/wiki/GetPostGameCarnageReport

kills	… Number of characters you killed during the match
averageKillDistance	… Average distance of your kills in this match
deaths	… Number of times you died in the match
score	… A numeric representation of your achievements this match
averageScorePerKill	… Average number of points per kill
averageScorePerLife	… Average number of points per life
weaponKillsPrimary	… Built by adding up all primary (eg automatic rifle) gun kills
weaponKillsSpecial	… Built by adding up all special (eg shotgun) gun kills
weaponKillsHeavy	… Built by adding up all heavy (eg rocket launcher) gun kills
weaponKillsOther	… Built by adding up all ability (eg grenade) kills
assists	… Number of assists achieved in the match
completed	… Did you finish the match? 1 == yes
killsDeathsRatio	… Kills / death
killsDeathsAssists	… ((Kills + assists) / death)
activityDurationSeconds	… How long were you in the match?
standing	… 0 == win, 1 == loss
combatRating	… Bungie's internal skill metric. Used heavily in matchmaking
CharacterId (Unnamed: #)	… Character Identification Number
matchId	… Match Identification Number
Team	… Which team were you on; 16 == Alpha, 17 == Bravo

These twenty statistics are what was initially pulled from Bungie's servers. In further preprocessing, the 20 individual-scope features were concatenated together for all players who played together in the same match. After having combined all players who partook in one match into a single vector, matchId, CharacterId, and standing were removed. MatchId was removed because it served its purpose of linking together players who played on the same teams. CharacterId was removed because its primary purpose was to be used in the clustering aspect of the original scope of the project. Furthermore, CharacterId was used when pulling the individualized statistics from the PGCR as a way of indexing whose stats were being pulled at that moment. Lastly, standing -- the target variable -- got cut and replaced with a new statistic that only appears once per match "alphaVictory." If team Alpha had a standing variable of 0, then alphaVictory is 0, or else alphaVictory was 1.

The goal of this pared down project is to take PvP matches played in *Destiny* and determine which team is more likely to win said match -- that is equivalent to whether or not team Alpha won. Since the game is zero-sum, only Alpha or Bravo team can win. As such, the probability that Alpha won the match equals one minus the probability that Alpha lost the match.

$$1 = P(AWon) + P(ALost)$$
$$P(AWon) = P(BLost)$$
$$P(ALost) = P(BWon)$$
$$P(BWon) + P(BLost) = 1$$

Given now that there is a target variable, the problem has been phrased as a supervised learning problem. Therefore, by making alphaVictory the target value, and by creating a neural network which performs a binary classification, we can take a softmax over the resulting classification and have the probability that team Alpha won the match that the neural network just saw. The closer to 0, the more likely that Alpha won that match.

Aside/Lemma:

Before we proceed, here is a quick discussion of how Neural Networks work.

The reasoning behind why this algorithm is called a Neural Network is because the design of the algorithm takes influence from how neurons are arranged in brains. Each neuron will take in an input, perform an action, and then send out the result of that action as part of the inputs for the next neuron.

Therefore, what do we need for a single neuron? We need an input vector, X. A corresponding target vector, Y. Weights, W -- whose dimensionality matches the input vector -- for the algorithm to tune. A threshold, T. A continuous non-linear activation function, F(). And a Loss function, L(), to derive the backpropagation from. The Loss function takes the predicted outputs of the network and compares them against the true outputs for each example. Up first comes the scalar valued function of vectors X and W, whose result, Alpha, lives in **R1**.

1) $$XdotW = Alpha$$

2) $$Alpha + T = Beta$$

3) $$F(Beta) = Gamma$$

4) $$L(Gamma, Y) = Loss$$

Now, once this framework is scaled up to an entire network rather than just one neuron, the Gamma's come together to make up the input vector of the next neuron in a chain of neurons -- repeating the process until you reach an output layer. Using the predicted class as well as the true class of the input, you can compete the Loss of the network. By taking the derivative of the Loss, you create the Gradient of that pass. You can then backtrace through the network the effect that each individual neuron had on the overall gradient (since your activation function is continuous) using a clever implementation of the multivariate chain rule. This allows the neurons in the network to individually tune their weights based on their exact contribution to the end result.

For *Victory Net*, the input space, X, is one 1x205 vector. This vector represents a single 12 person match of *Destiny* where the format of the vector is:

$$X =:< person0TeamA'sstats, ..., person5TeamA'sstats,$$
$$person6TeamB'sstats, , person11TeamB'sstats, alphaVictory >$$

where each person contributed all of the stats talked about above except for characterId, matchId, and standing. Removing these three features for each person in a 12 person match results in 204 statistics. The 205th statistic, alphaVictory, is the target class and represents whether team Alpha won said match.

AlphaVictory is cut from the vector and used as the target variable, which in turn makes the input dimension one 1x204 vector.

Now, this entire description was with respect to a 12-person match; however, I also pulled data for 6 person matches (two teams of 3 each). In this case the above holds just scaled down to one 1x103 vector. All else remains the same and after cutting out alphaVictory as the target, the vector's size is 1x102.

Given that *Victory Net* is attempting to classify a match as either won by team Alpha or lost by team Alpha, the classifier/benchmark to beat is the ZeroR classifier. The ZeroR (or Zero Rules) classifier takes the task of simply guessing the same output variable for every single instance/match it sees. This in turn, will result, on average, with a 50% success rate. Therefore, if *Victory Net* can beat 50%, then we know it is doing better than just guessing that "team Alpha won this match".

Methodology

Prerequisites for Creating the dataset

Given that this research is novel, there did not exist a dataset that I could freely use. As such, I had to create my dataset. Above I talked about what my dataset looks like and the statistics that it pulls for each player for each PVP match in *Destiny*, but I did not talk about how I created, cleaned, and used the dataset. That conversation starts here.

As I mentioned above, there are two key components in the creation of my data set. One, the Bungie API Key and two, the list of matchId numbers used to request the match details from Bungie's servers. With those two items and the requests library in Python, you can re-build the data I used in this project. As such, I will make the matchId keys public.

Creation and Cleaning

The preprocessing that was performed to create the dataset starts from parsing each PGCR that comes in and saving the relevant statistics with respect for each player in each match (in this case to a CSV). Since the original work of the project was to perform a clustering using these statistics in at attempt to tease out play-styles, the dataset created directly from the PGCRs was playerId indexed such that each row was one player's stats for one (specified in the columns) match.

Using the PlayerCSV file, with the columns as listed above, missing data was filled in with a -1 and outliers were removed from the data set. Given that this game is skill-dependent and that some of the statistics are correlated, what constituted an outlier was not very strict (in fact, strengthening the definition of outliers will change the results and will be investigated at a later date). Instead of declaring that all values which fell outside of the interquartile range of each statistic were outliers, the window was widened the range from 25%-75% to 10%-90% taking the interdecile range. Furthermore, to ensure that only extreme outliers were removed, the characterId of each outlier was tallied up across all 20 statistics. If a character showed up as an outlier in less than 3 areas/features, they were allowed to remain in the data set. Furthermore, given how several of the collected statistics for each person are correlated, as one statistic goes up so do others (e.g. as kills go up, so does score, KDR, KADR, weaponKills[insert weapon type], etc). Therefore, if someone was truly an outlier, then they would shows up as an outlier across many of the features. Being lenient on what constitutes an outlier was done to ensure that: people who happened to play well that match, or people who happened to play against extremely poor opponents, or people who played in match-types where scoring is scaled were not removed.

In order to remove an outlier from the data set, the entire match which that person had participated in had to be removed. If only that instance of that individual person was removed from the dataset, this would have resulted in an imbalance in the teams -- whereas before a match was 6v6, now it was 6v5. Removing the entire match was done because if only that person was removed that specific match, it would have ruined that match entirely from a learning perspective as the match is no longer fair or balanced. Furthermore, it would have de-standardized the input vectors for *Victory Net.*

To begin with, the data set contained 250,000 matches. Removing the outliers dropped the size of the data set to around 220,000 matches. And splitting the data set into subsets of 12-person matches and 6-person matches lost another 10 percent due to the fact that for some of the characters the matches were underpopulated leading to the problem discussed in the preceding paragraph. For matches where people entered the arena, left, and then the vacant spots were refilled (resulting in more than 12 people per match), players who abandoned the match part-way through were removed. This allowed for match size to be standardized to 6 or 12 people.

Once the far-reaching outliers were removed, the data was normalized using the min-max method. Following the normalization, the data was then restructured such that instead of being characterId indexed, it was now matchId indexed. This resulted in many 12x20 matrices which were then flattened into rows of 1x240. Removing certain identification statistics -- characterId, matchId, standing -- results in the vector shrinking to 1x204. Finally, the statistic of alphaVictory, which represented the result of the match, was included. AlphaVictory was previously represented at an individual level showing up n-times per match (once for each character who participated in said match), and was rescoped it to a team-wise statistic.

One of the most important statistics that was pulled was combatRating. This number represents Bungie's own internal metric used to evaluate players with respect to skill. CombatRating can be thought of almost like an inner-product since it is a scalar-valued function mapping a multidimensional input to an R1 output. Furthermore, combatRating is a Bayesian implementation and expansion based on the ELO system from chess, but known as TrueSkill (Herbrich 2007). CombatRating -- or any ELO style system -- functions as the backbone of a matchmaking system to pair together and against players of similar skill levels.

Given that the final form of my data set was simply a flattened version of when the data was indexed by characterIds, data visualizations was performed on the characterId-indexed data set. Due to how the data set was created, while being characterId indexed, all players who participated in a given match together were also grouped together. This allowed easy slicing of the specific matches, as each match occupied a contiguous section of the CSV. However, even if that feature were not present and the corresponding people were spread out all across the data set, the match could be recovered due to the matchID key.

Visualizing Matches

Histograms and distribution curves of each of the variables can be seen here.

To fully understanding the parallel plots, we should first discuss some of the descriptive statistics about the dataset along with the histograms linked above.

Due to the correlative nature of some these features, it should not be surprising that they will share distribution types. Many of the distribution curves for the variables present in the data set follow, if not exactly a Poisson distribution, with lambda between 4 and 10 than something close. The features with lower lambdas range from: averageKillDistance, averageScorePerLife, KDR, to KADR. While other features such as Kills, Deaths, and Assists, seem more normal/to have higher lambdas. As such, it seems safe to say that the statistics are a bit skewed. Due to either missing pieces of data -- which were filled in with negative ones -- or simply many zeros in the data, applying a function such as the natural log of the data to help with the skew just did not work in this case. Furthermore, the skewed nature of the data was in part due to how outliers were -- or were not -- removed. Being lenient on what constitutes an outlier lengthens the distribution curves for each variable. Ultimately, I decided that having slightly skewed data was alright in this case due to how it reflected the skill-based nature of the game.

In addition to the histograms and distribution curves above, rectangular-coordinate-framed radar-charts were used to plot out the statistics of each player with respect to each match. Given that the aim of *Victory Net* is to predict which team will win in a competition, a radar chart seemed like a natural extension. This allowed for direct comparisons of the statistics for each player (and for each team) both individually and as a group. Since there are over 200,000 matches to visualize, a random subset of the dataset was generated and used for the parallel-plots. Examples of themes found in the visualizations have been picked out, specifically three examples which showcase the different interactions.

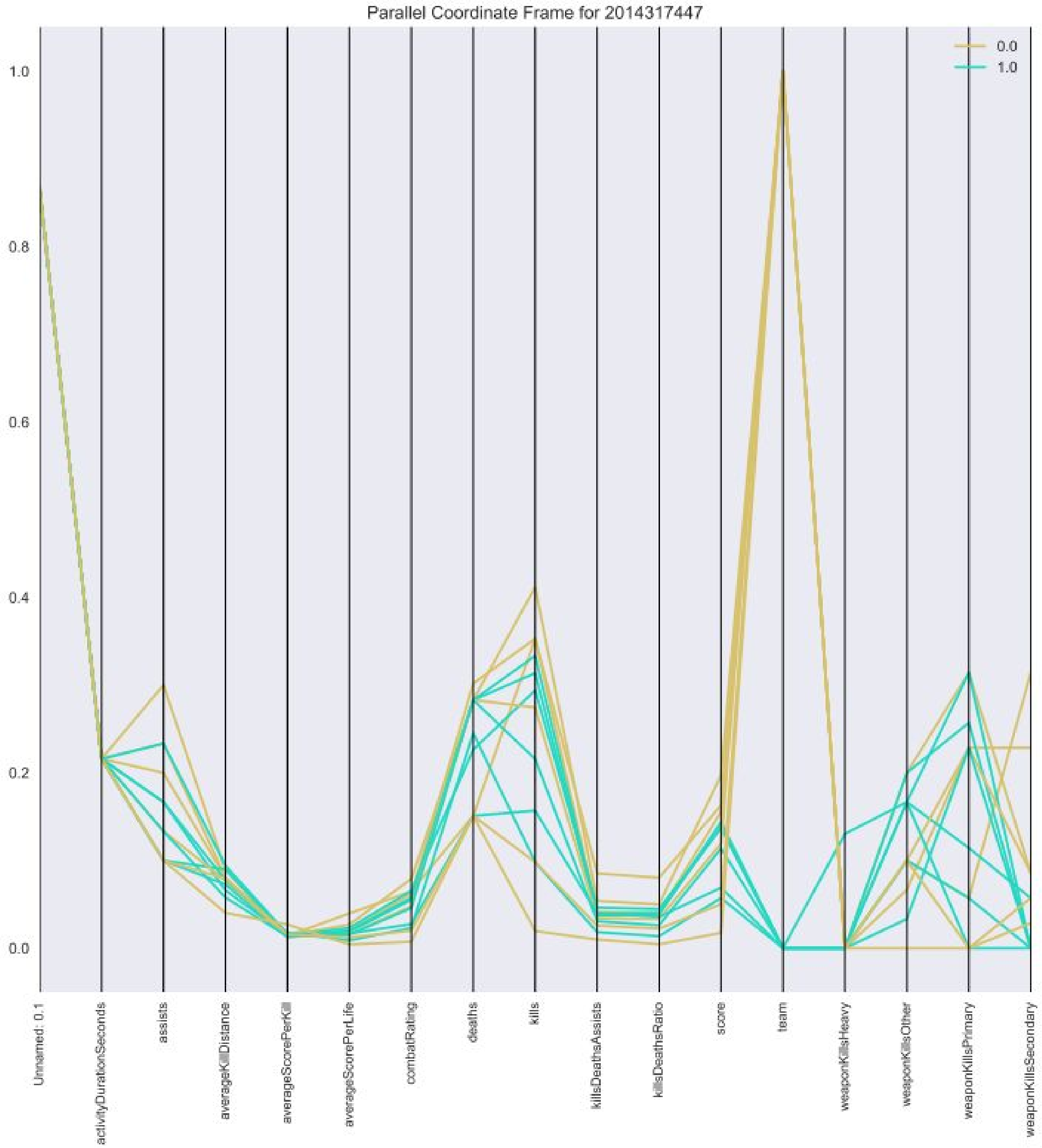

Figure 1:
Parallel Plot for match 2014317447
Team Alpha -- Green (1 -- Lost)
Team Bravo -- Yellow (0 -- Won)

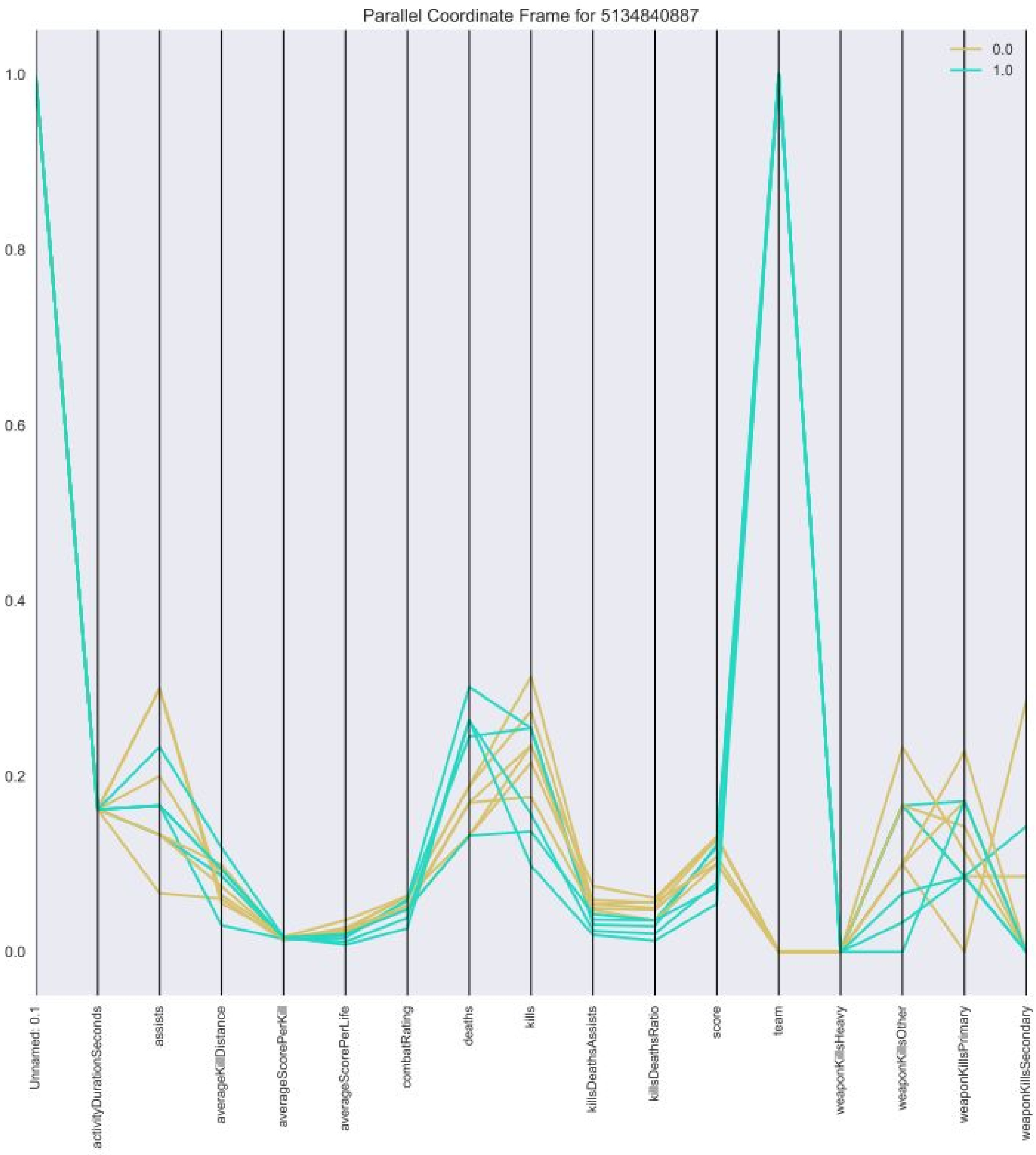

Figure 2:
Parallel Plot for match 5134840887
Team Alpha -- yellow (0 -- Won)
Team Bravo -- Green (1 -- Lost)

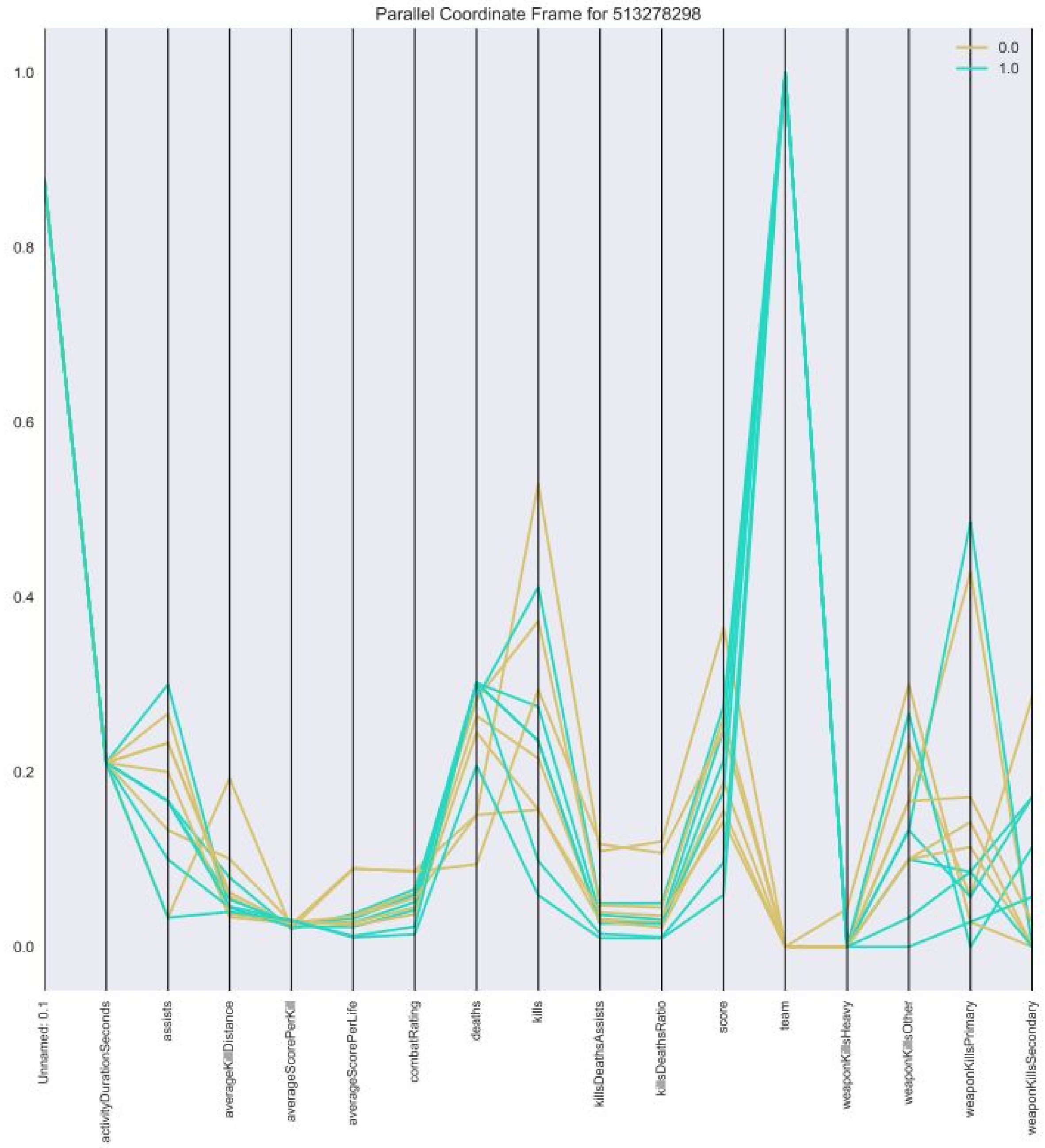

Figure 3:
Parallel Plot for match 513278298
Team Alpha -- yellow (0 -- Won)
Team Bravo -- Green (1 -- Lost)

Figures 1, 2, and 3 all have an additional visualization at the end of the results section which take each of the lines in the parallel plots and maps them into a Fourier series as an attempt to visualize high-dimensional clustering in a smoother manner.

In each of figures 1, 2, and 3, a general trend from the data can be witnessed. The two teams can be identified by the "team" variable. Team Alpha has a value of 0 in the team feature while team Bravo has a value of 1 in the team feature. Using that and the coloring of the two teams -- yellow → victory; green → defeat -- there are some patterns which can be teased out. The advantage of using a Parallel Plot, is the direct comparison of each team member's statistics. On average, it appears that the yellow team has higher values in the columns of averageScorePerKill, Kills, KDR, KADR, and Score. This would seems to indicate that those features have more descriptive power than others.

In the case of figure 1, yellow almost entirely has higher values on each independent axis than green across all statistics. For figure 3, it is very much a mixture. There are one or two people on yellow who are above everyone else, however, for the most part yellow and green seem intermixed. This could indicate that the match which took place was between players of similar skill levels. That in turn would make the match that figure 3 represents one that is difficult for *Victory Net* to reach a confident answer upon. Figure 2, on the other hand, seems similar to figure 1, however, as will be seen at the end of the paper, these have drastically different Fourier embeddings in **R2**.

The fact that there exists features where the winning team has higher values than the losing team implies that creating *Victory Net* should be possible. That there exists -- in some n-dimensional space -- a way to separate the two teams according to their performance. To view more matches visualized in this manner, go here.

Learning Matches, Predicting Victory, and showing Results

An important note going forward is that the size of the data set to be used was reduced to just the 12-person matches. This was done so that the algorithm implementations could be standardized. Furthermore, the 12-person matches data set was chosen over the 6-person matches with the thinking that if an answer to this more complex question could be found, then scaling down the models should be a simple addition for a later date.

The algorithm that was used when designing and making *Victory Net* was a multi-layered, fully-connected neural network -- a Multilayer Perceptron (MLP). The structure for *Victory Net* from taken Huang et. al. (2003) equations[4] for the optimal two-layer structure.

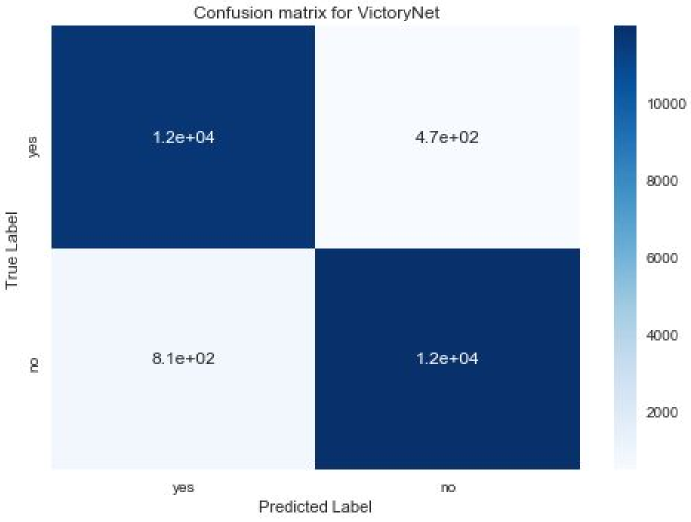

figure 4

[4] equations 1, 2, and 3: http://dstath.users.uth.gr/papers/IJRS2009_Stathakis.pdf

The first attempt to build *Victory Net* used Tensorflow[5]; however, when the model remained stuck at 0% loss and 100% accuracy no matter what subset of the data set was used, that method was shelved and instead Keras[6] with a Tensorflow back-end was used.

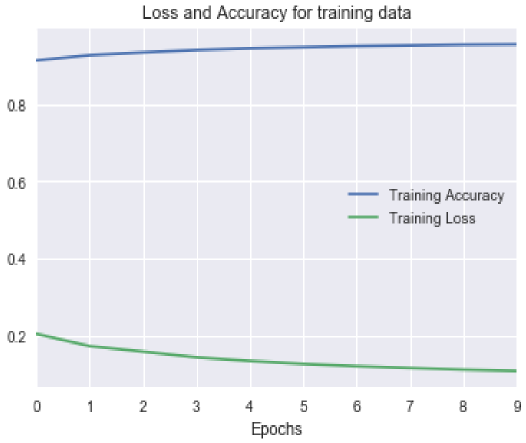

figure 5

And, just as the the Parallel Plots seem to tell us, *Victory Net* manages to correctly predict whether or not team Alpha wins the match. Once given the summary statistics of the match for each participant, *Victory Net* manages to achieve a final accuracy of 94.8% and loss of 12.2% on a testing set. This is a resounding success.

The confusion matrix for *Victory Net* can be see in figure 4. From that confusion matrix, it can be seen that the precision, recall, and F1 scores of *Victory Net*, are all 95%.

Furthermore, the learning curves -- with respect to both training and validating -- for *Victory Net* (figures 5 and 6) show how the accuracy and loss of *Victory Net* changed over its 10 epochs of training time. Figure 5 shows the training curves while figure 6 shows the validation curves.

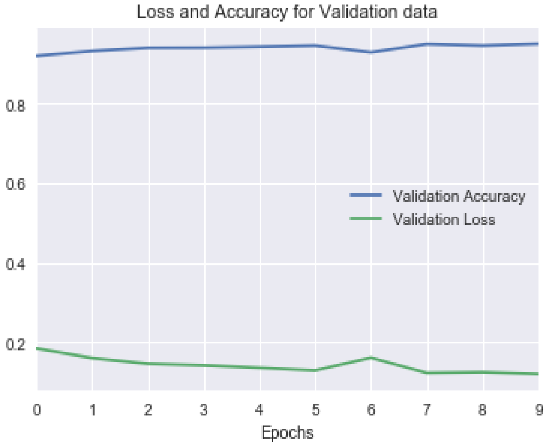

figure 6

A fair question to address here is why a neural network over other methods. Part of that reason is this was envisioned as an experiment and learning process with a neural network -- to really dive deep in their uses. However, due to initial trouble with Tensorflow, in order to check if the problem was systemic to the data, the data was also fed into a Support Vector Machine (SVM). This out-of-the-box SVM, without much hyperparameter tinkering, managed to achieve an accuracy of 92.4% on testing data. Furthermore, as shown in Figure 7, the learning curves for the SVM almost mirror the accuracy curves generated by *Victory Net*.

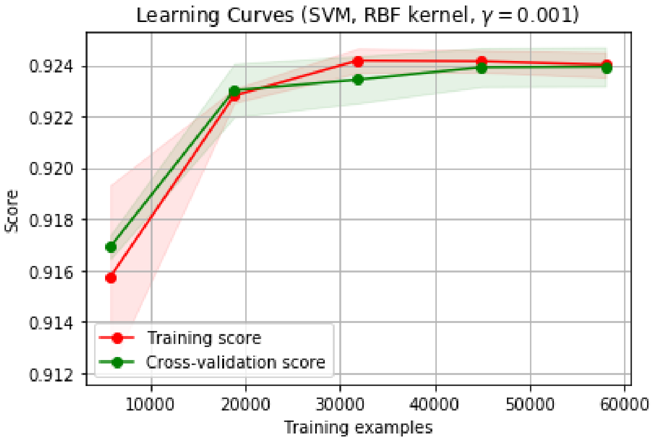

figure 7

The hyper parameters that are up for review in neural networks -- specifically MLPs -- are:

1) The number of Epochs to train for
2) The number of layers in your network

[5] https://www.tensorflow.org/
[6] https://keras.io/

3) The number of neurons in each layer
4) The learning rate of your network and the type of update rule for your weights

As can be seen in the Validation data graphs (figure 6), at epoch 7 is where learning seems to stagnate. This tells us that continuing past epoch 7 adds diminishing amounts of new information. With respect to the number of hidden layers in *Victory Net* and the number of neurons in each layer, Huang et al's equations linked above were used. These say that the optimal number of nodes in a two-layer neural network is dependent on the number of examples that the network sees and the number of different possible classifications of each training example -- positing the three equations in footnote 5 as an optimal heuristic to go by for a two layer MLP (86,000 inputs and two classes → hidden layer 1 (511 neurons) → hidden layer 2 (170 neurons) → output layer (1 neuron)).

For *Victory Net's* learning rate update rule, adagrad[7] (adaptive gradient) (Duchi 2011) was used. This update rule has two paramount parts. First, a cache of the sum of previous gradients squared is kept in memory, followed by the weights updating according to the rule: weights += -learning_rate * dx / sqrt(cache) + smoothing term. The cache is used in order to not allow the weights to make drastic immediate changes. As the amount of epochs grows, so too does the cache which in turn -- if you want to think about it in terms of a momentum update -- "preserves the momentum" of the weight changes. Since the gradient is the derivative of the Loss of the network, the update rule for each weight, in order to minimize your position on the learning_shape, is the negative of the gradient. In this case, due to how there were only two classes for *Victory Net* to classify training examples into, Loss was computed using binary cross-entropy.

Not only was the initial threshold beaten, it was soundly surpassed. So much so, in fact, that the obvious question these results beg is: "can *Victory Net* be trusted?" *Victory Net's* success could be for several reasons. It is possible that the metrics that Bungie records do a good job of articulating the different aspects of a person's skill in their game. It is possible that the variables that were recorded for each person for each match, were highly expressive. It is possible that due to the fact that one player's kill is another player's death each player impacts the statistics of each other player and what is recorded is the total record of each person's game play. Or it could be that since some of Bungie's features -- specifically combatRating -- are themselves integral to Destiny's matchmaking and evaluating every player's skill; therefore, these statistics can be formatted together to most of the time predict an outcome of a game.

In all likelihood, the truth for why *Victory Net* performed so well is probably a combination of each of the reasons mentioned above as well as good preprocessing of the data with respect to how the variables were cleaned -- at an individual level -- and structured -- at a match level -- for the network.

While *Victory Net* is clearly robust enough to solve the problem, the fact that these results likely depend on the implementation of Bungie's work on skill-evaluating-features, it seems fair to say that *Victory Net's* results cannot be inherently trusted without further testing.

[7] First proposed by Duchi et al. http://jmlr.org/papers/v12/duchi11a.html

As stated in the Pandas documentation for Andrews Curves:

> “Andrews curves allow one to plot multivariate data as a large number of curves that are created using the attributes of samples as coefficients for Fourier series. By coloring these curves differently for each class it is possible to visualize data clustering. Curves belonging to samples of the same class will usually be closer together and form larger structures”[8]

Therefore, the question then becomes, what structural difference become apparent upon graphing the matches with the Andrews Curves? What clustering effects are brought to light? And what do these clusters tell us about the data?

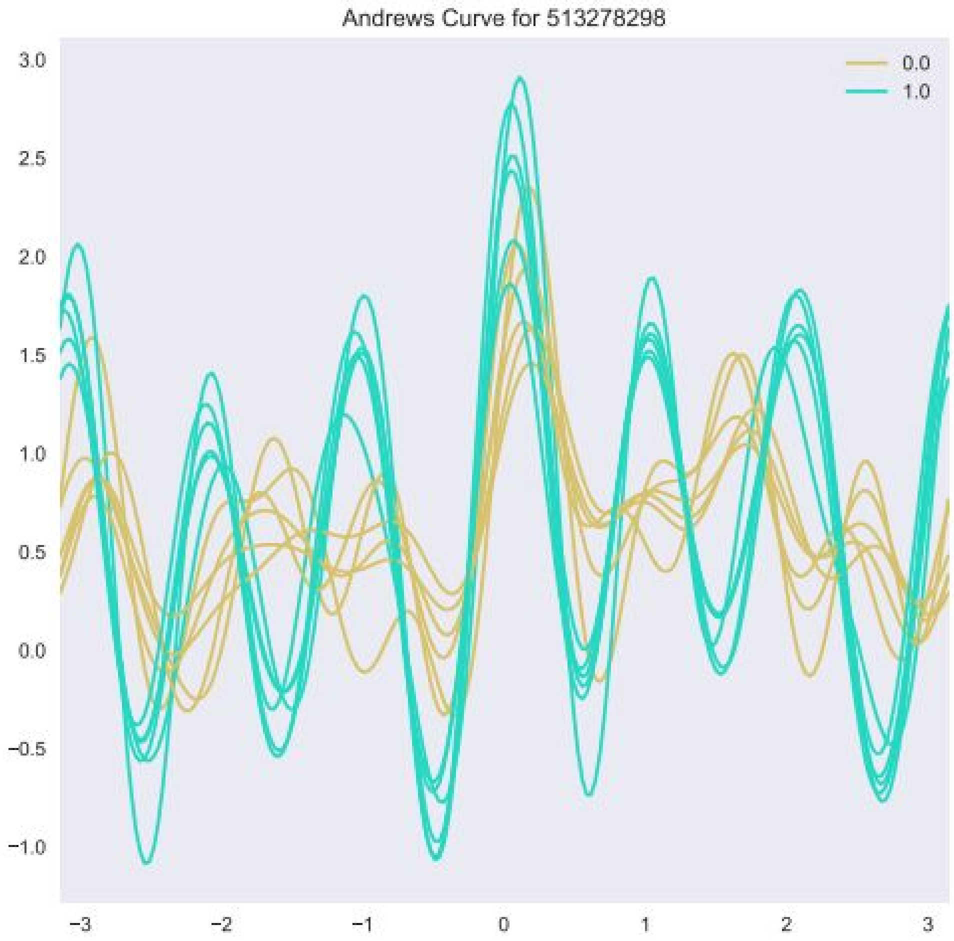

Figure 8:
Paired Andrews Curve for figure 3.

[8] https://pandas.pydata.org/pandas-docs/stable/visualization.html#andrews-curves

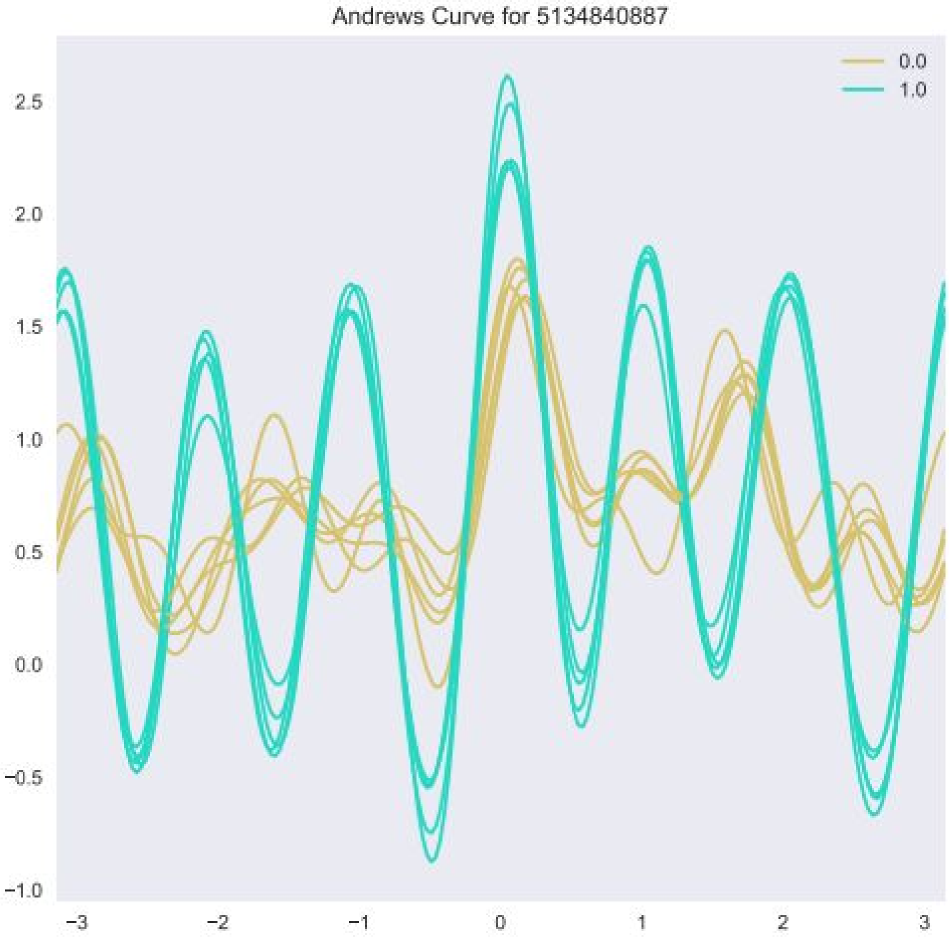

Figure 9:
Paired Andrews Curve for Figure 2

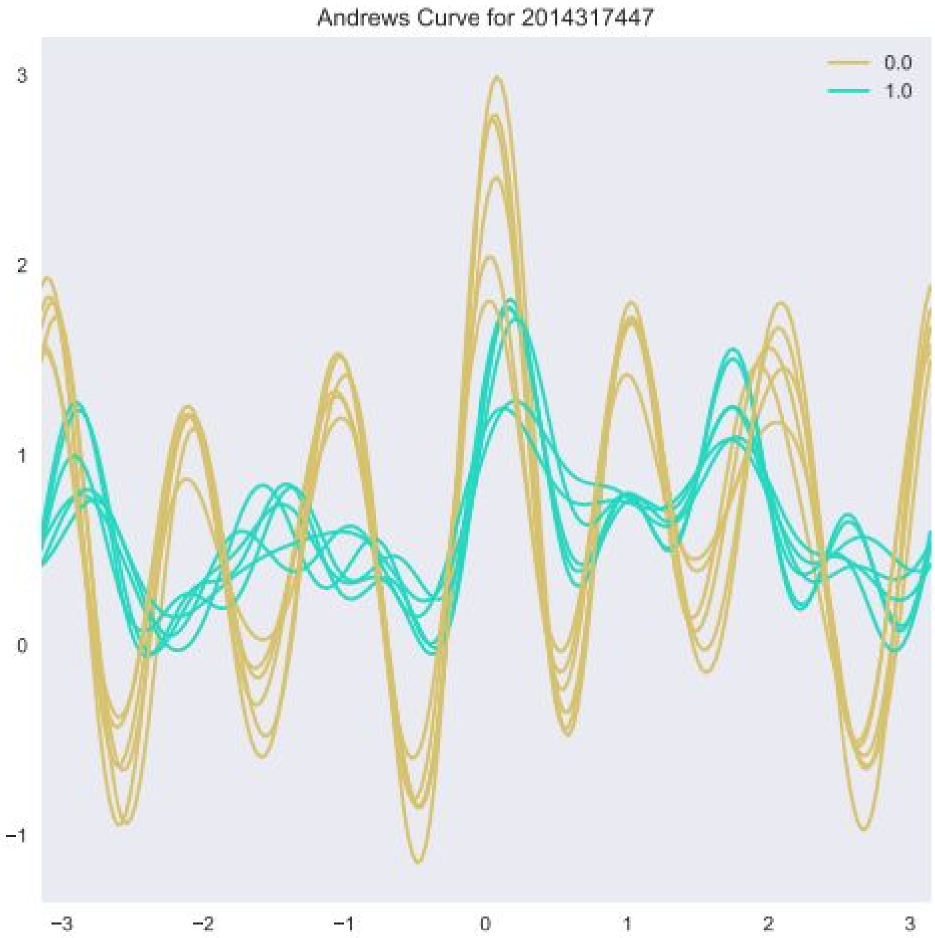

Figure 10:
Paired Andrews Curve for Figure 1

As can be seen in the associated Andrews Curves for figures 1, 2, and 3, the Fourier compression of the match statistics can vary quite a bit. For Figure 1, the two teams seem almost separable by the Andrews Curves. Team A, in yellow, has a much higher amplitude in its curves separating team Alpha from Bravo in a very clear way. Meanwhile, for Figure 2, the reverse is true despite Team Alpha still being the team which won that match. Finally, in Figure 3, we see that although Team Alpha's curves are smaller, the two sets are very close -- almost indistinguishable even if the team-coloring were removed. While the majority of the visualized data, falls into the category of Figure 1 -- curves being separable and Team Alpha winning -- there is a non-insignificant amount which are like Figure 2 -- separable but with the losing team having the higher-amplitude curves. Finally there exists a small amount of matches which are more like Figure 3 -- maybe separable, but mostly very similar curves.

In any case, it is clear that there existed an embedding of the match-data such that the two teams were separable. The Andrews Curves show one such embedding using the Fourier series, while *Victory Net* -- probably -- found another such embedding.

Conclusion

In this paper, an objective method of evaluating teams of players was discussed. This has applications in other video-games and both e-sports and traditional sports as a predictive method for teams competing against each other. In order to generalize though to other team-based activities, the video-game-shooter-specific metrics would need to be replaced with more general statistics for each person and team. Furthermore, although 12-person matches were the bedrock of this paper, there is no reason why this method cannot scale up or down to team/match sizes of other integers. One blockade however, is that this is not built for matches between more than two teams or for games that are not zero-sum. One such place where *Victory Net* might be used would be in evaluating and predicting outcomes of *League of Legends* matches. While the individualized statistics are different, the broader framework is the same -- a team skill-based zero-sum game. In fact, the comparison is apt since with the upcoming *Destiny 2* release, Bungie seems to be wanting to push their PvP community into the competitive e-sports scene.

Further improvements to the project would be to perform tests to pinpoint exactly which features contributed most to the final solution for each example. This could be done by creating "parallel" sets where specific types of data have been removed. As an example one such parallel set could have all cluster-relevant data removed using purely skill based metrics. Or further tests could be performed comparing the impact of features suspected to be powerful due to their very nature and the problem.

These results should not be used as a new benchmark when studying a team skill-based zero-sum game. This is because some of Bungie's data was used without the knowledge of how Bungie arrived at the scores they did. Although we know the general update rules for combatRating, we do not know specifics of how performance inside of a match can impact the metric. Nor we do we know exactly how that metric is used in Bungie's internal matchmaking. If that feature is the core component, then naturally, it will map very closely to the answer.

Finally, the future work from this research is quite straightforward. *Victory Net* is the third piece of the original data pipeline proposed at the start of this paper. Much of the data that was collected was to be used in building playstyle clusters (i.e. averageKillDistance, weaponKills[weaponType]). Using these playstyle clusters and the fact that all teams share identifications, these teams can be seen as sequences of people. This makes modeling the selection of players via play-style-classes possible with a recurrent neural network, theoretically. If team A is made up of play-style clusters A, B, A and they won their match, then this looks like a good cluster of play styles which complement each other. Therefore, using an LSTM to build teams can be seen as a precursor to evaluating how well the two teams might perform against each other -- which is exactly what *Victory Net* predicts.

Special thanks to Dr Anders Drachen, Dr Rafet Sifa, and Destiny Trials Report for their help in making this project possible.

Citations

G. J. G. C. H. E. L. P. S. R. . K. D. Drachen, A., “Guns and guardians: Comparative cluster analysis and behavioral profiling in destiny,” in Proceedings of IEEE Computational Intelligence in Games, 2016

Herbrich Ralf, Minka Tom, Graepe Thore, 2007, TrueSkill: A Bayesian Skill Ranking System. In Proceedings: Advances in Neural Information Processing Systems 20. 569-576, MIT Press.

HUANG, G.-B., 2003, Learning capability and storage capacity of two-hidden-layer feedforward networks. IEEE Transactions on Neural Networks, 14, pp. 274–281.

John Duchi, Elad Hazan, Yoram Singer, 2011, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization; Journal of Machine Learning Research 12(Jul):2121−2159.