MLND

Capstone Project

Zach Freedman
2/6/2018

1. Definition

Project Overview

Overwatch is a popular first person shooter game available for PC, Xbox, and PS4. In competitive Overwatch, a matchmaker pits 2 teams of 6 similarly skilled players against one another in objective based games.

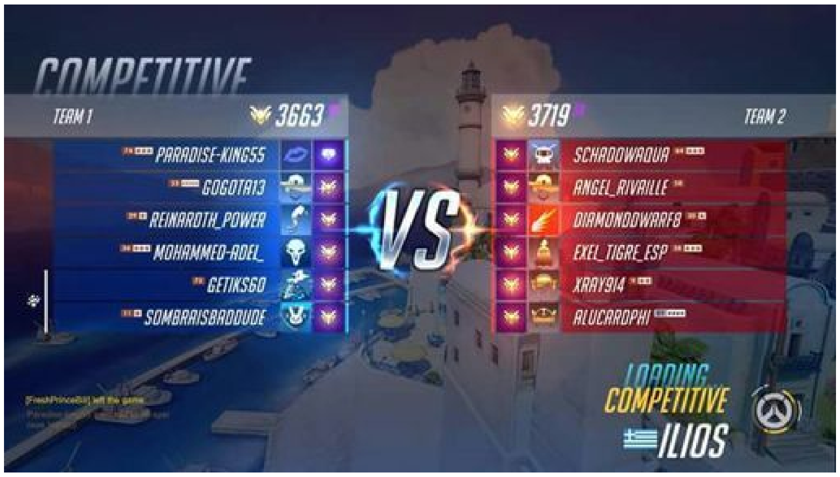


Figure 1: Overwatch Matchmaking

The matchmaker always tries to find a *fair* match, where fairness is determined by the difference in SR (skill rating) average of the 2 teams. In the above image, the SR averages of the blue and red teams are 3663 and 3719, respectively. The SR ladder is bounded below and above on the interval [1, 5000], where low rated SR players are deemed less adept at the game in comparison to high rated SR players. The following list displays how Overwatch breaks down SR classes based on the [1, 5000] range:

- Bronze [1, 1499]
- Silver [1500, 1999]
- Gold [2000, 2499]
- Platinum [2500, 2999]
- Diamond [3000, 3499]
- Master [3500, 3999]
- Grandmaster [4000, 5000]

Once a match is found, players from each team pick from a pool of 26 heroes (at the time of this report, Season 8 competitive Overwatch has not released hero 27) without replacement. This means that any given hero, say Hanzo, can only be played by 1 red and 1 blue player at a time. However, once a player picks a character, they can freely swap off to a different character throughout the game.

During a game, players eliminate enemies, heal their teammates, block incoming damage, and more, in an attempt to complete more objectives than the opposition. The game is finished when a maximum number of objectives or a time cap has been reached. If a blue team player *b* plays 2 heroes in the match, say Pharah and Lucio, *b*'s ingame performance for each hero will be updated in Overwatch's database for entries b_{Pharah} and b_{Lucio}. Given that the blue team wins the game, *b* is guaranteed an SR increase, whereas if the blue team loses, *b*'s SR decreases. In the event of a draw, the SR of each player on both teams does not change.

Problem Statement

Seeing as how high rank players win games in order to rank up, the hypothesis is that high ranking players must perform well (on average) during games to contribute to their 6 man team's success. Using hero performance data as a feature set and the above 7 SR classes as targets, the project goal is to figure out which (if any) hero performance features for a given hero separate players of different SR classes. For example, between Bronze and Silver players using Mercy, how different on average are their hero performances, and does there exist a Mercy performance feature subset that can definitively separate Bronze and Silver Mercy players?

Through the use of supervised learning, this project investigates said question.

Metrics

In order to assess how well a supervised learner *s* classifies player-hero entries with respect to their SR class, accuracy can be used to measure, in total, how many instances does *s* classify correctly. For a concrete example, take any Overwatch Season 8 character, like Junkrat. The true positives for an SR class, say Platinum, is the count of the total number of Platinum Junkrat entries which *s* correctly identifies to be Platinum. For this 7 class classification problem, the accuracy is measured by the sum of the true positives over the total number of Junkrat entries. Given D_{Junkrat} is the dataset of all junkrat players and TP_c is a true positive Junkrat count for some target SR class *c*, the multiclass Junkrat accuracy is

$$acc_{\text{Junkrat}} = (TP_{\text{Bronze}} + TP_{\text{Silver}} + TP_{\text{Gold}} + TP_{\text{Platinum}} + TP_{\text{Diamond}} + TP_{\text{Master}} + TP_{\text{Grandmaster}}) / |D_{\text{Junkrat}}|,$$

where the notation $|D_{\text{Junkrat}}|$ is the cardinality (total number of entries) int the Junkrat dataset.

However accuracy alone will not give enough perspective on how well some *s* actually classifies the data. It's very important that the model performs well with respect to precision (out of all predicted Gold Junkrats, how many are actually Gold) and recall (out of the actual Silver Junkrats, how many were predicted Silver). Using a balanced F1 score, precision and recall can be taken into account equally when evaluating *s*. The balanced F1 is given by

$$F = 2PR / (P + R),$$

where *P* and *R* are precision and recall, respectively, and the metric is bound on [0,1].

2. Analysis

Data Exploration

SR Distribution

Although supervised learning is performed for each of the 26 heroes, for simplicity purposes this section will focus on 1 hero. All topics discussed in this section can be generalized to the other 25 heroes, who have small differences with respect to hero-specific features.

First, it's important to understand the target distribution for Overwatch heroes. Without loss of generality, let's examine Tracer, a commonly played DPS (attack) hero. The below graph shows the SR distribution in the domain of [1, 5000] along the horizontal axis. The vertical axis counts the total number of players for a specified SR bin.

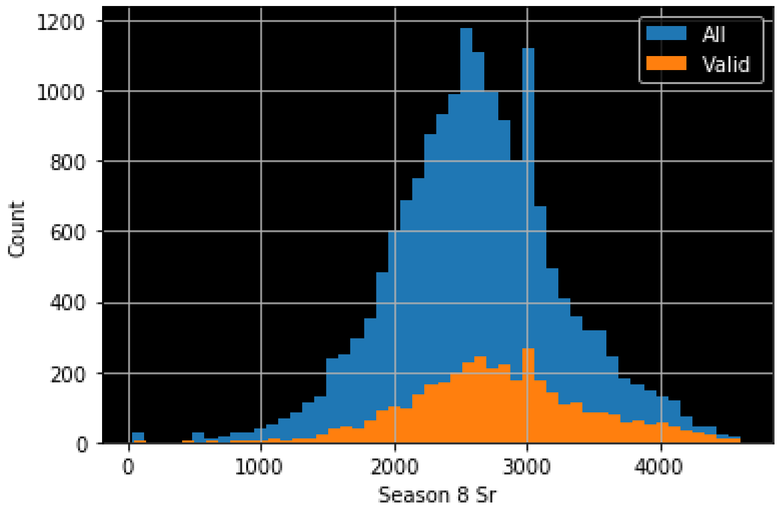


Figure 2: Tracers with 3600+ seconds

This graph depicts the SR for 16,934 Tracer players (blue), of which 3,727 played Tracer for an hour or more (gold) at the time the data was scraped (about half way through the 2 month season). The distribution is normal with the exception of the 3000 SR bin. This uptick in player count exactly at the 3000 SR mark is most likely due to SR decay. When a player is ranked diamond or higher (3000+), account inactivity results in SR decay with a maximum decay to SR = 3000. So, although there are many players with an SR of 3000, a sizable fraction of these are players previously ranked at a higher SR who haven't played in a while.

Valid Entries

The gold histogram depicts what's referred to as *valid* entries throughout the codebase. *Valid* is used to describe players who've played a specific hero for *enough* time in order to deem their entry worth observing, where *enough* is open to interpretation. To understand why validity is important, it's crucial to note that for any player *p*, *p*'s Season *n-1* final SR heavily influences where they start Season *n* at. In the opinion of most Overwatch players, based on their experience, Season *n-1*'s final SR has greater impact on Season *n*'s starting SR than how well/poorly someone plays during placement matches (the first 10 matches of a competitive Season).

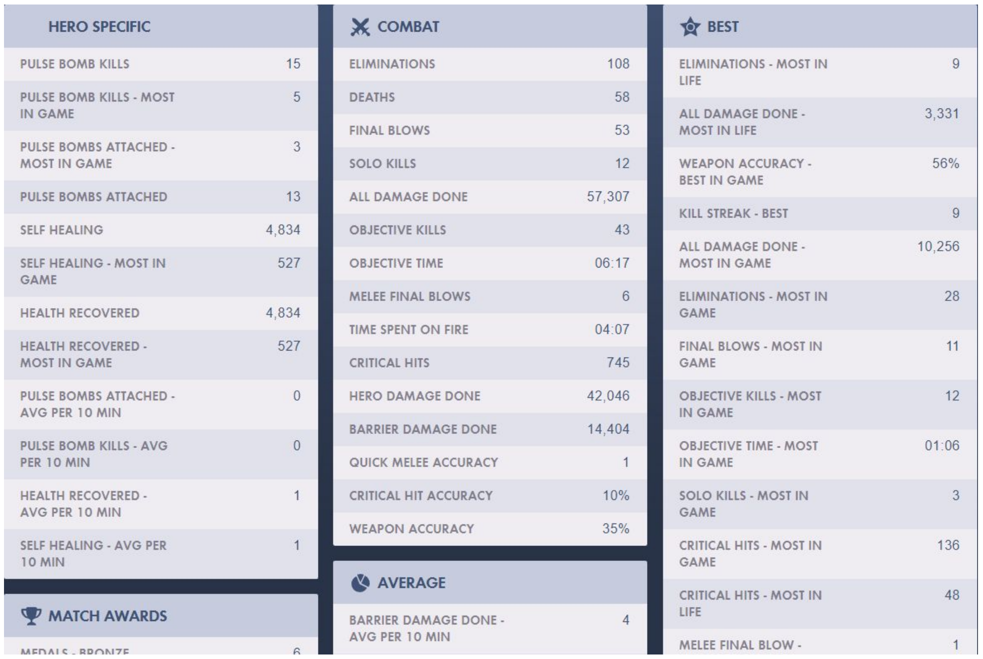
What kind of players would be considered invalid? It's hard to tell with the project's dataset, so a real life example is better suited. Take player *M* who mained (played mostly) Mercy in Season 7. Mercy is a support hero that does not require much mechanical skill (ability with mouse and keyboard) in comparison to a hero like Widowmaker, a DPS hero requiring arguably some of the highest mechanical skill with respect to the Season 8 competitive hero pool. Suppose *M* climbed to the rank of Grandmaster with an SR of 4000 while playing 20 hours of Mercy and 1 hour of Widowmaker during Season 7.

During Season 8 placements, if *M* plays only Widowmaker and finishes their placement matches, it's likely for *M* to place in the [3500, 4500] SR neighborhood, with a larger bias towards the lower range due to *M*'s inexperience with Widowmaker in the previous competitive season. The assumption is that *M* won't perform terrifically on Widowmaker at their current SR because they ranked up playing mostly Mercy, whose skill set does not transfer well to Widowmaker. After placements, *M* would have collected roughly 3-4 hours of Widowmaker-only gameplay for Season 8. But, it is possible that *M* is placed close to 4000 (their Season 7 final SR) due to the heavy influence Season 7 SR has on Season 8's placement (start of season) SR. Assuming *M* can't play Widowmaker at a 4000 SR quality, they *should* lose games until they derank to a point where they can compete equally while playing Widowmaker (assuming *M* continues to spam pick Widowmaker).

Suppose it's known through some prophecy that *M* will derank to SR = 2500. If *M*'s profile is scraped at SR = 3400, using *M*'s Widowmaker entry would provide noise for the Widowmaker Diamond [3000, 3499] subset. In order to maximize data integrity, such *invalid* players should be removed.

Raw and Processed Datasets

During data collection, 2 datasets are created: *raw* and *processed*. In these datasets, each feature is numeric, for every hero, with the exception of battletags (player name + unique ID). The *raw* dataset corresponds to data in its original form at the time of scraping from PlayOverwatch, where hero performance is maintained as running sums.



HERO SPECIFIC	
PULSE BOMB KILLS	15
PULSE BOMB KILLS - MOST IN GAME	5
PULSE BOMBS ATTACHED - MOST IN GAME	3
PULSE BOMBS ATTACHED	13
SELF HEALING	4,834
SELF HEALING - MOST IN GAME	527
HEALTH RECOVERED	4,834
HEALTH RECOVERED - MOST IN GAME	527
PULSE BOMBS ATTACHED - AVG PER 10 MIN	0
PULSE BOMB KILLS - AVG PER 10 MIN	0
HEALTH RECOVERED - AVG PER 10 MIN	1
SELF HEALING - AVG PER 10 MIN	1

MATCH AWARDS	
MEDALS - BRONZE	6

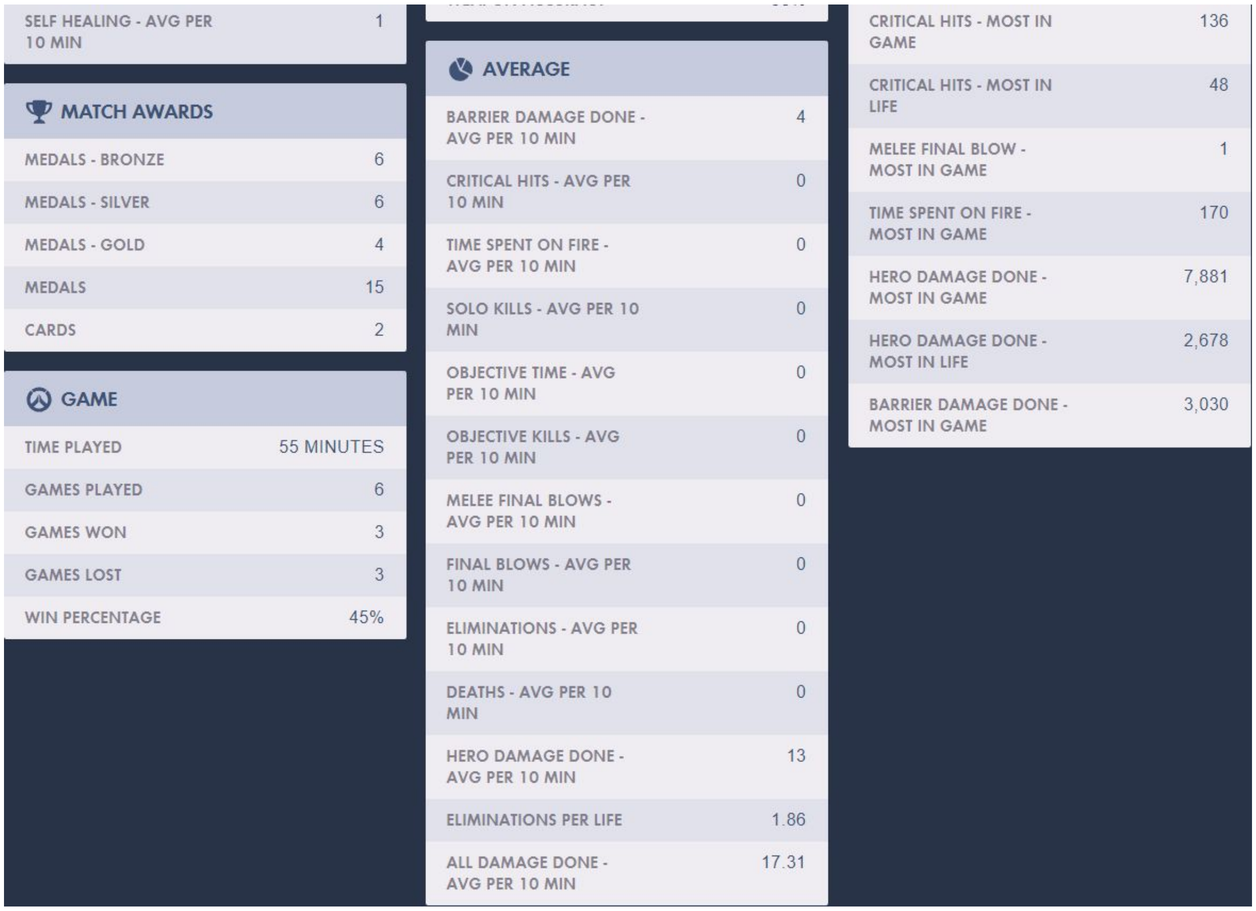
COMBAT	
ELIMINATIONS	108
DEATHS	58
FINAL BLOWS	53
SOLO KILLS	12
ALL DAMAGE DONE	57,307
OBJECTIVE KILLS	43
OBJECTIVE TIME	06:17
MELEE FINAL BLOWS	6
TIME SPENT ON FIRE	04:07
CRITICAL HITS	745
HERO DAMAGE DONE	42,046
BARRIER DAMAGE DONE	14,404
QUICK MELEE ACCURACY	1
CRITICAL HIT ACCURACY	10%
WEAPON ACCURACY	35%

AVERAGE	
BARRIER DAMAGE DONE - AVG PER 10 MIN	4

BEST	
ELIMINATIONS - MOST IN LIFE	9
ALL DAMAGE DONE - MOST IN LIFE	3,331
WEAPON ACCURACY - BEST IN GAME	56%
KILL STREAK - BEST	9
ALL DAMAGE DONE - MOST IN GAME	10,256
ELIMINATIONS - MOST IN GAME	28
FINAL BLOWS - MOST IN GAME	11
OBJECTIVE KILLS - MOST IN GAME	12
OBJECTIVE TIME - MOST IN GAME	01:06
SOLO KILLS - MOST IN GAME	3
CRITICAL HITS - MOST IN GAME	136
CRITICAL HITS - MOST IN LIFE	48
MELEE FINAL BLOW -	1

Figure 3: Sample Tracer PlayOverwatch profile

However, for whatever reason, the averages PlayOverwatch maintains are always incorrect.



SELF HEALING - AVG PER 10 MIN	1

MATCH AWARDS	
MEDALS - BRONZE	6
MEDALS - SILVER	6
MEDALS - GOLD	4
MEDALS	15
CARDS	2

GAME	
TIME PLAYED	55 MINUTES
GAMES PLAYED	6
GAMES WON	3
GAMES LOST	3
WIN PERCENTAGE	45%

AVERAGE	
BARRIER DAMAGE DONE - AVG PER 10 MIN	4
CRITICAL HITS - AVG PER 10 MIN	0
TIME SPENT ON FIRE - AVG PER 10 MIN	0
SOLO KILLS - AVG PER 10 MIN	0
OBJECTIVE TIME - AVG PER 10 MIN	0
OBJECTIVE KILLS - AVG PER 10 MIN	0
MELEE FINAL BLOWS - AVG PER 10 MIN	0
FINAL BLOWS - AVG PER 10 MIN	0
ELIMINATIONS - AVG PER 10 MIN	0
DEATHS - AVG PER 10 MIN	0
HERO DAMAGE DONE - AVG PER 10 MIN	13
ELIMINATIONS PER LIFE	1.86
ALL DAMAGE DONE - AVG PER 10 MIN	17.31

CRITICAL HITS - MOST IN GAME	136
CRITICAL HITS - MOST IN LIFE	48
MELEE FINAL BLOW - MOST IN GAME	1
TIME SPENT ON FIRE - MOST IN GAME	170
HERO DAMAGE DONE - MOST IN GAME	7,881
HERO DAMAGE DONE - MOST IN LIFE	2,678
BARRIER DAMAGE DONE - MOST IN GAME	3,030

Figure 4: Sample Tracer PlayOverwatch averages

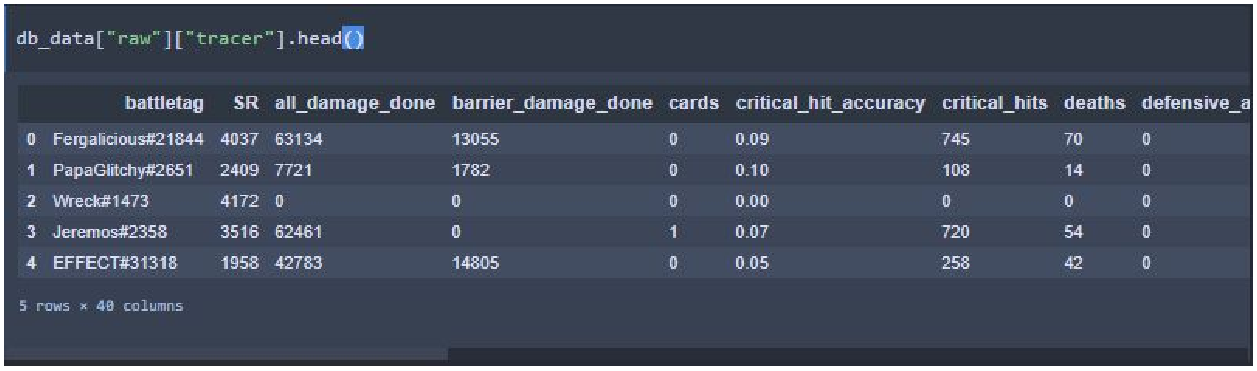
Because of this, averages are removed before writing scraped data to *raw* files.

The *processed* dataset includes redefined averages with respect to 2 different features: eliminations and time played. This means that any feature worth averaging has 2 distinct averages. The reasoning behind this was to represent features with respect to time and life efficiencies. A Tracer with a high damage/second value is efficient at putting out damage quickly, whereas Tracers with high damage/life value are efficient at damaging while maintaining importance on their own life. It could be an important distinction to make, seeing as how quick eliminations can be indicative of skilled players. However, it's well known that 5v6 team fights usually favor the team with 6 heroes, so high damage/life values could signal players' team centric mindsets.

Additionally, new features are manufactured and added. For example, Tracer, who has features for *Pulse Bomb Kills* and *Pulse Bombs Attached* (see Figure 3 Hero Specific table), has a manufactured feature for the ratio between attached pulse bombs and pulse bomb kills. The idea behind feature manufacturing is to seed the dataset with relationships having the potential to increase learning.

Sample Entries

Raw and *processed* data are written to separate directories, each containing a CSV file for each of the 26 heroes. Features named like *Feature Here* are converted into *feature_here*. Below is a sample of 5 Tracer entries from the *raw* dataset as the data exists in Pandas.



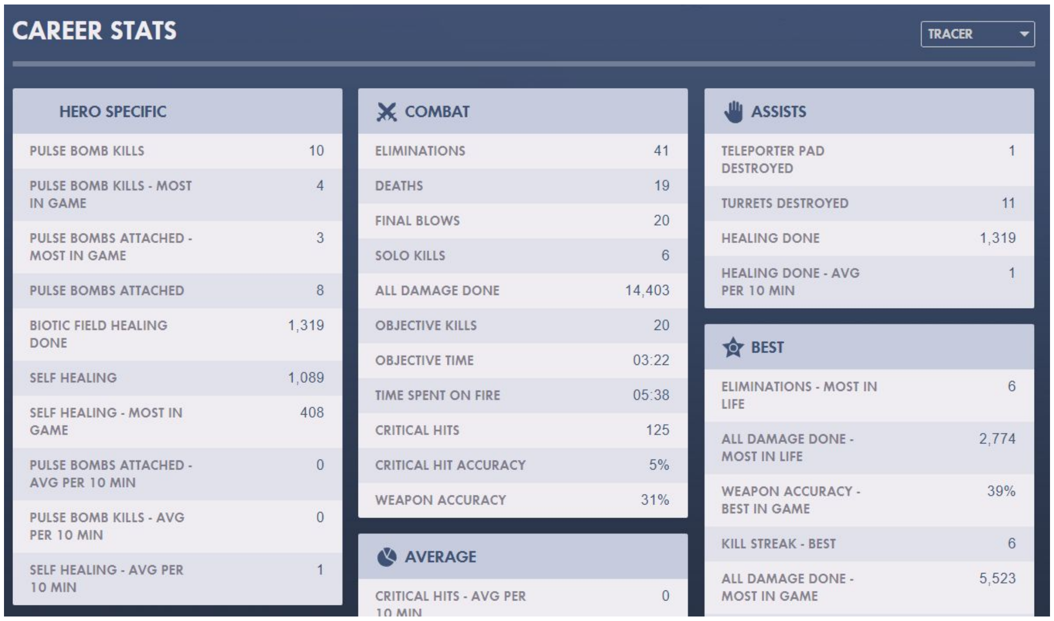
```
db_data["raw"]["tracer"].head()

     battletag        SR    all_damage_done  barrier_damage_done  cards  critical_hit_accuracy  critical_hits  deaths  defensive_a
0    Fergalicious#21844  4037  63134    13055    0    0.09    745    70    0
1    PapaGlitchy#2651    2409  7721     1782     0    0.10    108    14    0
2    Wreck#1473          4172  0        0        0    0.00    0      0     0
3    Jeremos#2358        3516  62461    0        1    0.07    720    54    0
4    EFFECT#31318        1958  42783    14805    0    0.05    258    42    0

5 rows × 40 columns
```

Figure 5: First 5 *raw* Tracer entries

Noisy Entries

At the time of collection, some data points have problems. For example, for players who have only played a few minutes on Tracer, PlayOverwatch has a competitive entry for them, but the displayed data on the website is minimal. This is because PlayOverwatch hides most 0 valued fields (HTML isn't generated). Therefore, if a player swaps to Tracer for a moment and the game ends immediately, they could have 0 data on PlayOverwatch with the exception of a Time Played value in seconds and the result of the game.

Furthermore, some entries actually have features for abilities their hero doesn't have. For example, this Tracer profile has a sizable Biotic Field Healing Done entry (Hero Specific table, left), which corresponds to the Biotic Field ability *only* available on Soldier: 76.



CAREER STATS — TRACER

HERO SPECIFIC	
PULSE BOMB KILLS	10
PULSE BOMB KILLS - MOST IN GAME	4
PULSE BOMBS ATTACHED - MOST IN GAME	3
PULSE BOMBS ATTACHED	8
BIOTIC FIELD HEALING DONE	1,319
SELF HEALING	1,089
SELF HEALING - MOST IN GAME	408
PULSE BOMBS ATTACHED - AVG PER 10 MIN	0
PULSE BOMB KILLS - AVG PER 10 MIN	0
SELF HEALING - AVG PER 10 MIN	1

COMBAT	
ELIMINATIONS	41
DEATHS	19
FINAL BLOWS	20
SOLO KILLS	6
ALL DAMAGE DONE	14,403
OBJECTIVE KILLS	20
OBJECTIVE TIME	03:22
TIME SPENT ON FIRE	05:38
CRITICAL HITS	125
CRITICAL HIT ACCURACY	5%
WEAPON ACCURACY	31%

AVERAGE	
CRITICAL HITS - AVG PER 10 MIN	0

ASSISTS	
TELEPORTER PAD DESTROYED	1
TURRETS DESTROYED	11
HEALING DONE	1,319
HEALING DONE - AVG PER 10 MIN	1

BEST	
ELIMINATIONS - MOST IN LIFE	6
ALL DAMAGE DONE - MOST IN LIFE	2,774
WEAPON ACCURACY - BEST IN GAME	39%
KILL STREAK - BEST	6
ALL DAMAGE DONE - MOST IN GAME	5,523

Figure 6: Tracer with Biotic Field Healing Done feature

Feature Transformation

It's important to note that each hero's *raw* and *processed* databases undergo several transformations while testing learners. These transformations include *valid* filtering, outlier detection, logistic scaling, minmax scaling, and principal component analysis, which are discussed later in the Methodology section.

Exploratory Visualization

The goal of the supervised learning task is to find feature relationships that can provide class separation. For example, if Bronze players perform poorly for a specified feature, then in an ideal scenario, Grandmaster players, on the opposite side of the SR spectrum, would perform drastically better. To see the type of sought after separation, observe the SR distribution separated by class for *valid* = 3600 seconds (1 hour).

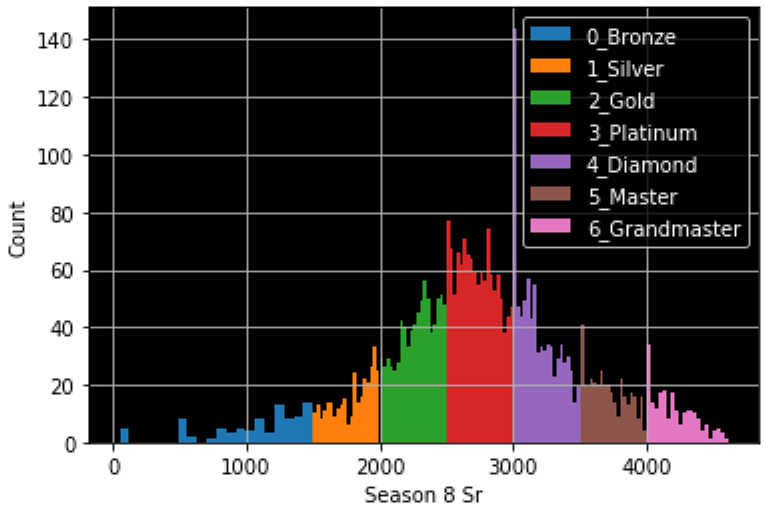


Figure 7: *Valid* Tracer SR distribution split up by class (histogram)

Of course this separation exists. The SR ranges for each SR class have already been established, so it makes sense Bronze players represented by the blue histogram only appear for SR below 1500. Note here that the height of the histogram at any point in time is not so relevant. What's important when analyzing this feature histogram distribution across the 7 SR classes is the separation between each class with respect to feature measurement.

The same data can also be drawn as a scatter plot.

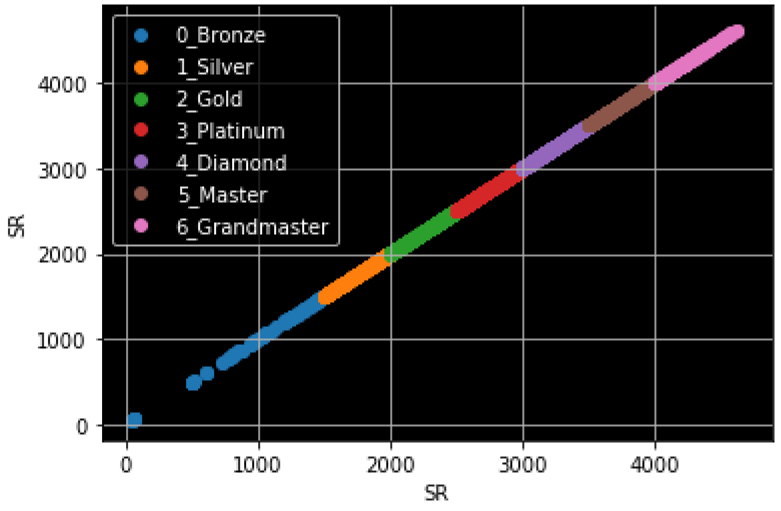


Figure 8: *Valid* Tracer SR distribution split up by class (scatter plot)

Now observe a histogram representation of Tracer players' Critical Hits per Life.

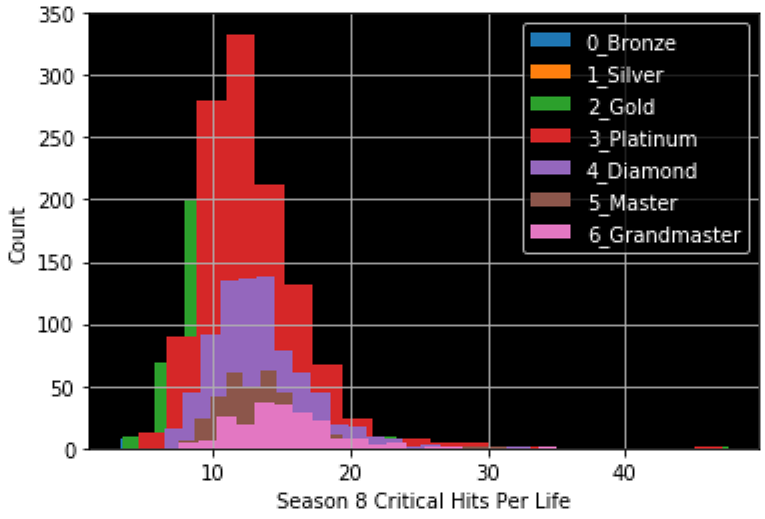


Figure 9: *Valid* Tracer Critical Hits per Life (histogram)

Looking at the histogram, it would appear as though the Critical Hits per Life distributions for the SR classes overlap a great amount. Additionally, it's impossible to see the distribution of Bronze and Silver ranking *valid* Tracers due to draw order (Grandmaster drawn on top of Master, for example). Critical Hits per Life is visualized by scatter plot in the next figure.

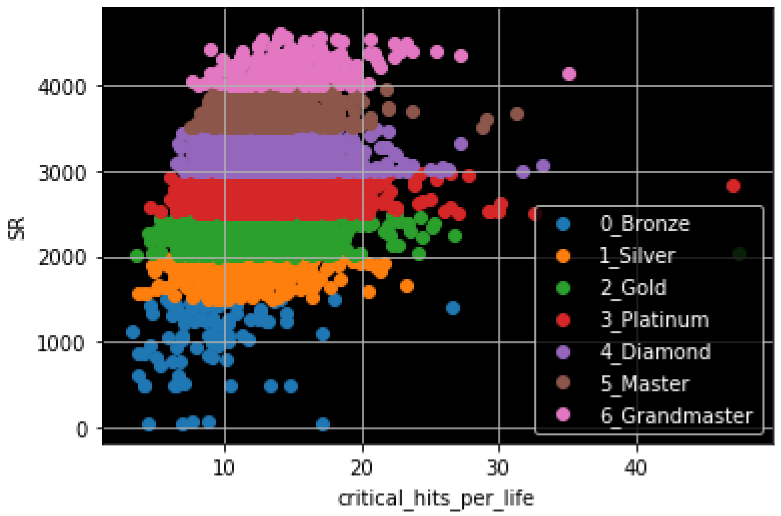


Figure 10: *Valid* Tracer Critical Hits per Life (scatter plot)

This new visualization shows the same data in a different light. Although there's a great deal of overlap, look at the average minimum Critical Hits per Life for Bronze players versus Grandmaster players. There's a measurable difference. Additionally, looking at the left side of the scatter plot as a whole, it's possible to bound the minimum feature values from Bronze to Grandmaster definitively in an increasing manner as SR increases. Therefore, it should be possible for some supervised learner *s* to make decisions like disregarding a *valid* Tracer entry as Grandmaster if their Critical Hits per Life falls well below 8.0.

Furthermore, to show confidence in the decision to treat *valid* players separately from all players, look at the same scatter plot for all Tracers.

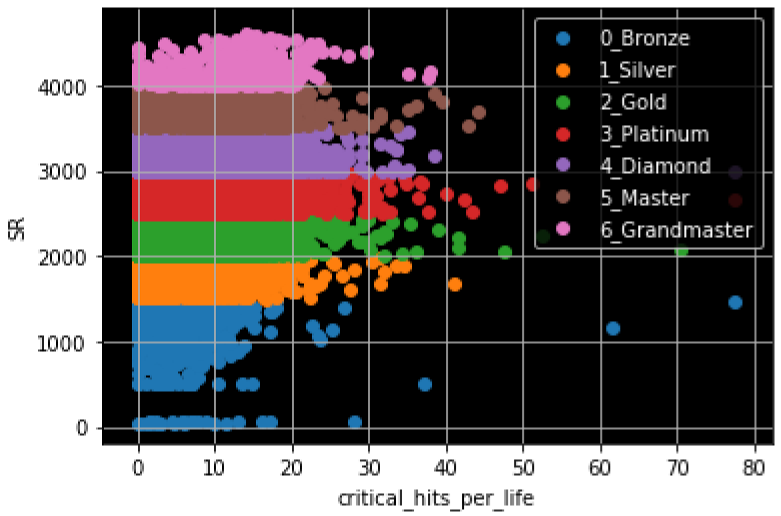


Figure 11: All Tracer Critical Hits per Life (scatter plot)

When plotting all Tracer entries, the lower bound information just observed has been lost. Findings like this instil confidence in the positive effects segregating *valid* entries can have on learning. Furthermore, when raising the time played requirement for *validity* to 7200 seconds (2 hours), data points are lost, but the lower boundaries for Critical Hits per Life become more well defined.

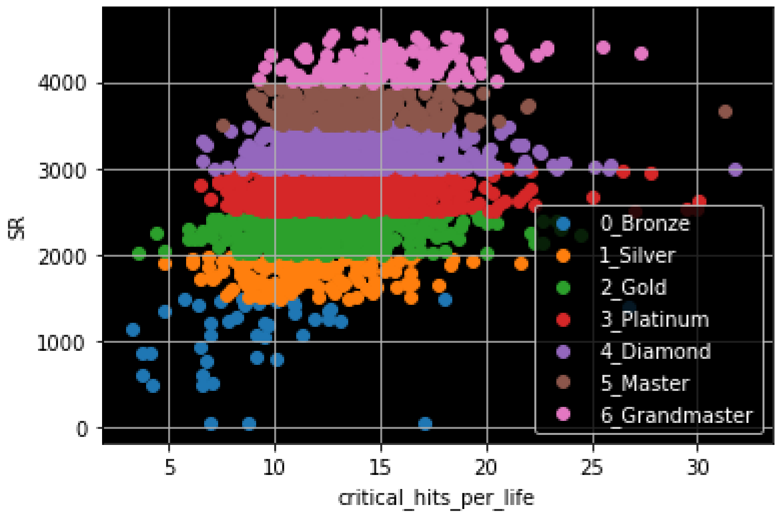


Figure 12: *Valid* Tracer Critical Hits per Life (scatter plot, *valid=7200*)

Algorithms and Techniques

Supervised learners are implemented in the attempt to learn the feature patterns that separate players of varying SR rating. Specifically, this problem makes use of decision trees, random forests, and neural network classifiers.

Decision Trees

A decision tree is a supervised learner which splits data into separate query branches based on the value of an individual feature. For a given branch, a decision tree continues to split data until a minimum impurity is reached. These classification nodes at branch ends are called leaf nodes.

Random Forests

Random forests are ensemble variants of decision trees. These learners use many decision trees to classify data based on majority vote, where each tree can be trained with a subset of the features describing the data.

Random forests are a solid choice because they scale well for high dimensional data and easily implement the dropout regularization technique.

Neural Networks

Basic neural networks consist of 3 layers. The input layer is the first layer of the network, where a node exists for each feature from the feature set. The input layer feeds forward directly into a set of hidden layers. Each node *n* in a hidden layer *h* connects to each of the previous nodes in the previous layer. Finally, the output layer is connected to the last hidden layer, where the number of output nodes corresponds to the number of possible classes. The network is trained by initializing weights of each node with some distribution. After feeding data forward through each layer, the network learns and adjusts by incrementally shifting the weights of each node to minimize the error during the next epoch.

Neural networks are chosen because they have been shown to perform well with high dimensional and complex data, often outperforming classic machine learning algorithms for difficult problems.

Benchmark

The benchmark for this learning task is a simple decision tree. This learner is trained and tested after scraping a total of roughly 90,000 out of 164,000 entries. After filtering, the decision tree is fit to the *valid processed* Tracer set, involving no outlier removal, logistic/minmax scaling, or PCA. The *validity* for this dataset is 14400 seconds (4 hours), which yields 376 Tracers with the following SR distribution.

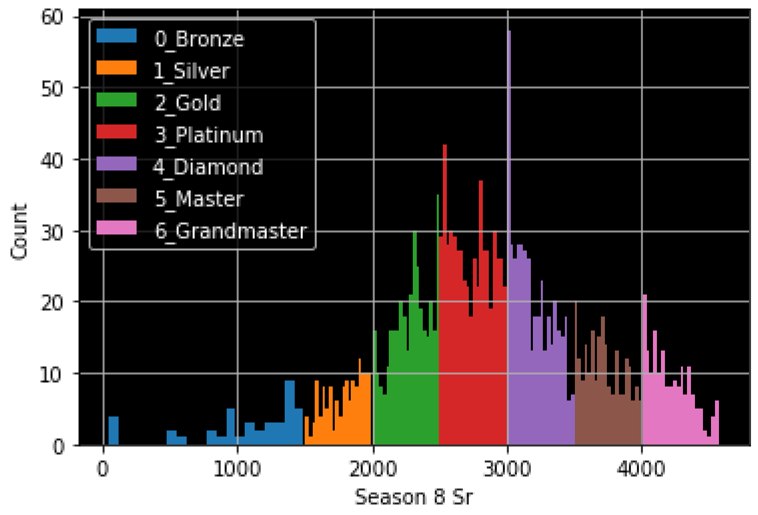
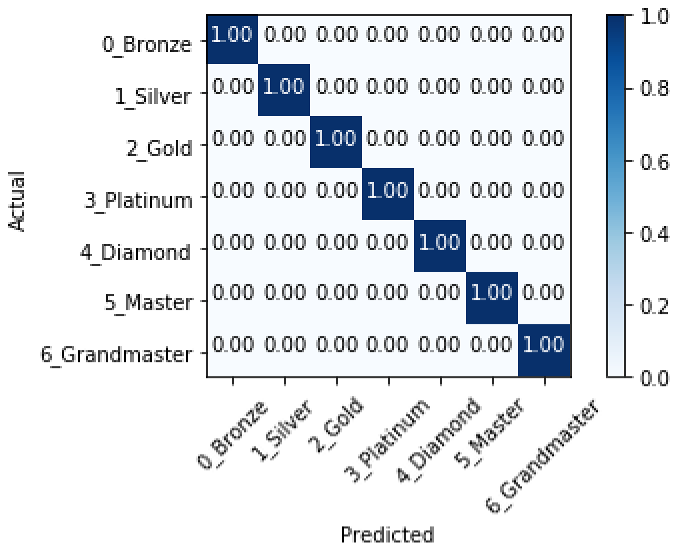


Figure 13: *Valid* Tracer SR distribution for benchmark (*valid=14400*)

The benchmark classifies with a 26.60% accuracy and a balanced F1 of .2866. The confusion matrices for the training and testing set are below. Note that the decision tree classifier shows clear signs of overfitting seeing as how it perfectly predicts the training set, visualized by a perfect diagonal in the first illustration.



	0_Bronze	1_Silver	2_Gold	3_Platinum	4_Diamond	5_Master	6_Grandmaster
0_Bronze	1.00	0.00	0.00	0.00	0.00	0.00	0.00
1_Silver	0.00	1.00	0.00	0.00	0.00	0.00	0.00
2_Gold	0.00	0.00	1.00	0.00	0.00	0.00	0.00
3_Platinum	0.00	0.00	0.00	1.00	0.00	0.00	0.00
4_Diamond	0.00	0.00	0.00	0.00	1.00	0.00	0.00
5_Master	0.00	0.00	0.00	0.00	0.00	1.00	0.00
6_Grandmaster	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Figure 14: Benchmark train confusion matrix (*valid=14400*)

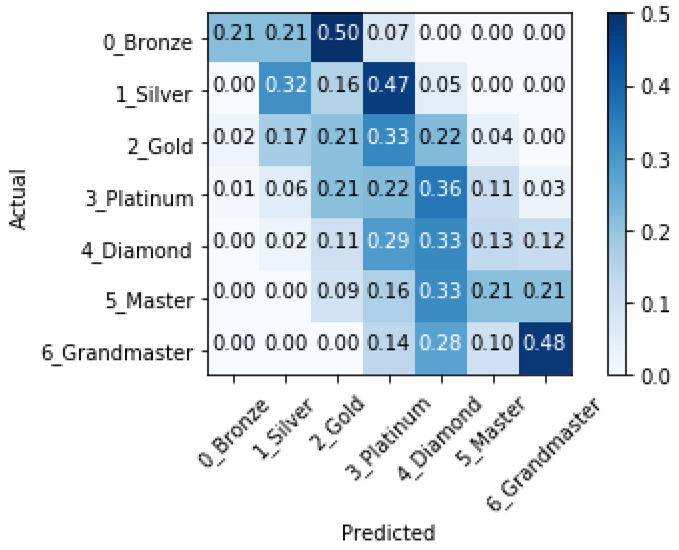


Figure 15: Benchmark test confusion matrix (*valid=14400*)

3. Methodology

Data Preprocessing

Missing Features

As previously described, some entries are absent features during data acquisition. Because of this, battletags with missing hero data have their hero-specific entries sparsely filled with 0s wherever missing features occur. Additionally, all available average data from PlayOverwatch is incorrect, so these features are recalculated for every acquired battletag.

Mismatch Features

Some battletags actually have features for a hero like Pharah that could only be available for Symmetra. In these extremely rare cases, the battletag is purged from the database. This means that all hero data related to a player with a singular feature mismatch is lost.

Manufactured Features

Some hero-specific features are manufactured in an attempt to provide nonlinear relationships for learning that aren't readily available through PlayOverwatch. Disregarding the recalculated averages, hero-specific features are added to the feature set for each hero, where the number of manufactured features ranges between 1-4 total features.

Valid Entries

From Data Exploration, it's known that valid entries are attained by filtering battletag-hero entries by Time Played. The minimum valid time used throughout the learning notebook is 14400 seconds, with some functions using default values of 3600. While increasing the validity threshold cleanses hero datasets of infrequent hero users, this cleansing measurably decreases the size of the dataset, which can be detrimental to learners requiring many data points.

Outlier Removal

Outlier entries are prevalent for each hero. Such outlier entries are detected by examining each feature for each hero. If a battletag for a specific hero has a singular feature value greater than 3 standard deviations from the feature mean, the battletag entry for the hero is removed.

Feature Scaling

Highly skewed features with a skew measurement exceeding 1.5 in magnitude are logistically scaled. Furthermore, all numeric features (everything minus battletags) are minmax scaled to the range [0, 1].

Feature Removal

Running sum features used for average calculations are removed from the dataset. These features include Time Played and Deaths. Furthermore, Games Lost, Games Played, Games Tied, and Games Won are better measured using the Win Percentage feature, so these 4 features are also purged from every hero feature set.

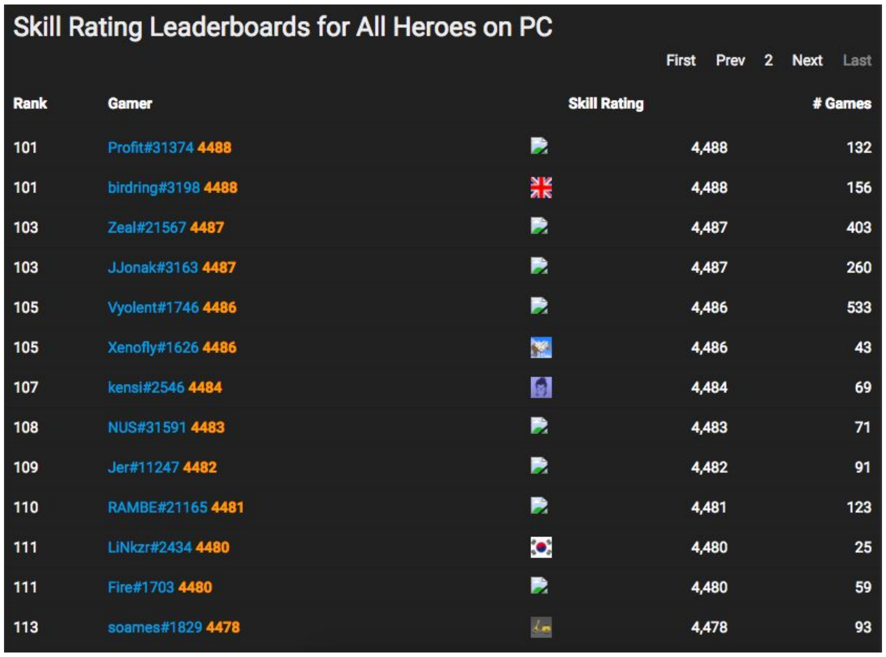
PCA

Lastly, principal component analysis is used for each hero to attain a new set of features explaining 99% of the data's variance. This greatly reduces the dimensionality for every hero, but it does not always improve supervised learning performance.

Implementation

Data Collection: OverwatchTracker

To start data collection, battletags need to be attained. OverwatchTracker provides a listing of current season Overwatch competitive players by platform (PC, Xbox, PS4) in a leaderboard sorted by global ranking.



Skill Rating Leaderboards for All Heroes on PC

First Prev 2 Next Last

Rank	Gamer		Skill Rating	# Games
101	Profit#31374 4488		4,488	132
101	birdring#3198 4488		4,488	156
103	Zeal#21567 4487		4,487	403
103	JJonak#3163 4487		4,487	260
105	Vyolent#1746 4486		4,486	533
105	Xenofly#1626 4486		4,486	43
107	kensi#2546 4484		4,484	69
108	NUS#31591 4483		4,483	71
109	Jer#11247 4482		4,482	91
110	RAMBE#21165 4481		4,481	123
111	LiNkzr#2434 4480		4,480	25
111	Fire#1703 4480		4,480	59
113	soames#1829 4478		4,478	93

Figure 16: Sample OverwatchTracker leaderboard

From this leaderboard, although SR, games played, and global rank are available, the only features of interest at this point are battletags. In the above image, the battletag for the rank 107 player is kensi#2546, where 2546 is the unique ID. The SR is not of importance at this point because the SR may change between the time of scraping the battletag from OverwatchTracker and the time when the hero-specific performance data is scraped from PlayOverwatch.

This data is acquired using a Python data scraping package, BeautifulSoup4, in conjunction with urllib2, which provides the functionality to make a web request and receive a response. The HTML of a specific page is scraped, collecting the battletags into a list. Afterwards, the OverwatchTracker scraping module crawls to the next page, repeating the process. Periodically, the module writes current battletags to a CSV file and updates the last scraped leaderboard page, maintained in a text file, so that scraping can be resumed without issue when interrupted.

Data Collection: PlayOverwatch

PlayOverwatch is the website managed by Blizzard Entertainment which keeps track of battletag profiles. Each battletag profile maintains Overwatch performance stats broken down by hero for quickplay (non-competitive play). If the battletag user plays competitive, there is an additional competitive profile separate from the quickplay profile which maintains similar hero-specific statistics.

For each battletag in the OverwatchTracker CSV file, a second module reaches out to the corresponding PlayOverwatch page. The url format for US PC battletags is

https://playoverwatch.com/en-us/career/pc/[player_name]-[player_ID].

If a player plays *n* heroes during the current competitive season, PlayOverwatch displays *n* competitive profiles, which may be varying levels of sparse/complete. This data is scraped with BeautifulSoup4/urllib2, but undergoes a lengthy preprocessing algorithm before being written to file to account for noisy entries. After being cleaned, battletags are saved in a dictionary maintaining all the recently scraped data, where hero names like Doomfist are used as keys and battletag player-profiles for Doomfist are values. Once enough battletags have been temporarily stored, the recent battletag profiles are written to CSVs for each hero. Again, it’s important to note that 2 CSVs exist for each hero, corresponding to the *raw* and *processed* databases described previously.

Lastly, during this writing to file period, the index of the most recently scraped battletag is incremented and rewritten to a text file so that scraping can be resumed when interrupted.

Data Cleanup

Some battletags appear several times in the OverwatchTracker battletag database. This is because player SR can be updated during scraping. If a player wins a match, they can force players they just surpassed to climb down in global ranking. Furthermore, if a player loses, they themselves will fall in ranking. Seeing as how battletags are scraped from highest rank to lowest rank, duplicate battletags are a common occurrence.

Additionally, some battletags managed to get through preprocessing with an incorrect feature count. These entries are not as common, but still need to be expunged.

In order to clean the scraped hero profile data, hero CSVs are reread into Python. Battletags are unique, and so for any specified hero, if a battletag occurs more than once in the CSV, it's a guaranteed duplicate. Once the entire dataset is checked, the duplicate lines are removed. A similar process is done for entries containing improper feature counts.

The remaining entries are written to post scrape cleanup directories.

Reading in Data

At the start of the learning notebook, the CSV files for each hero in each of the *raw* and *preprocessed* databases are read into Pandas dataframes. The dataframes are stored in a dictionary of dictionaries, where the level 1 key corresponds to *raw* or *processed*, and the level 1 value corresponds to a dictionary mapping hero names (level 2 keys) to their Pandas dataframe (level 2 values).

Preprocessing: Valid Entries

The first preprocessing step is to filter out *invalid* entries, which is done by removing any entries not meeting a minimum Time Played. During this step, filtered entries are written to a new multilevel dictionary, where the lowest level values are still Pandas dataframes. However, it's at this point where the data previously read in is split into features and targets. The features dataframe drops the battletag (non-numeric/unique) and SR features, and a target dataframe is created containing only the SR.

Preprocessing: Outlier Removal, Feature Scaling, Feature Removal, PCA

For each of these 4 preprocessing steps, the methodology has already been described. However, it's important to note, like during *valid* entries preprocessing, the data is written to a new cleaned set of dataframes after each preprocessing step. This is done to test supervised learners after each data preprocessing to see if the preprocessing positively or negatively impacted learning.

Supervised Learning

Supervised learners are refit to the data after each preprocessing step. This refitting is done by splitting the data with an 80/20 train/test split. The supervised learners of choice for this dataset are decision trees, random forests, and neural networks. The accuracy and balanced F1 for each set of learners is recorded after fitting and testing. Furthermore, confusion matrices are plotted for each set of learners after testing to visualize where misclassifications occur for each SR target.

Refinement

Decision Tree

The decision tree classifier shows early signs of overfitting when being fit on the *valid* dataset. For decision trees, overfitting can be avoided with pre/post pruning. Additionally, overfitting can be heavily influenced by dominating classes in multiclass classification. This dataset is heavily dominated by Gold, Platinum, and Diamond SR classes due to the normal distribution of SR.

The benchmark was not intended to outperform either the random forest or neural network classifier, but rather to quickly prove that a supervised learner could greatly increase upon random guessing, which would yield a 1/7 = 14.29% accuracy score. With no hyperparameters set except for the random state = 0 (note all learners use this hyperparameter value), the benchmark attained the previously mentioned 26.60% accuracy and .2866 balanced F1 on the *valid* Tracer set with a minimum Time Played threshold of 14400 seconds.

However, when setting max depth to 5, the classifier’s accuracy and balanced F1 reach a maximum of 36.44% and .3217. This maximum is obtained with the underprocessed *valid* set, outperforming further processed dataset variations.

Random Forest

Similar to the decision tree classifier, the random forest shows signs of overfitting prior to implementing further preprocessing. This is clearly visible by firmly diagonalized confusion matrices with the training sets for each hero. In an attempt to alleviate this, different hyperparameters are altered. The best set of hyperparameters included the classifier count and minimum leaf samples. The minimum leaf samples parameter dictates a pruning criteria for how many samples need to end up in a leaf node for it to remain connected to the tree. The classifier count dictates how many randomized decision trees exist in the random forest for majority voting.

The hyperparameters classifier count and minimum leaf samples are assigned values of 500 and 5, respectively. On the *valid* Tracer dataset, the random forest classifier achieved metric scores of 37.77% and .3340. However, although outperforming the single decision tree classifier by a small amount, the random forest classifier training confusion matrix overfit a great deal more.

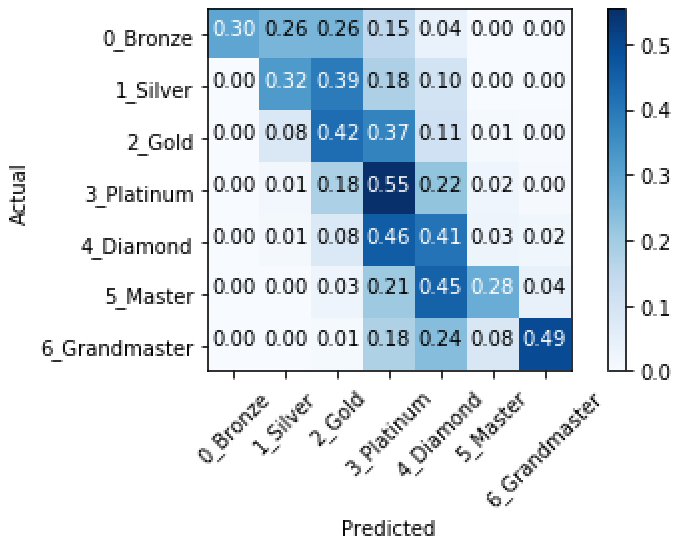


Figure 17: Decision tree training confusion matrix (*valid*)

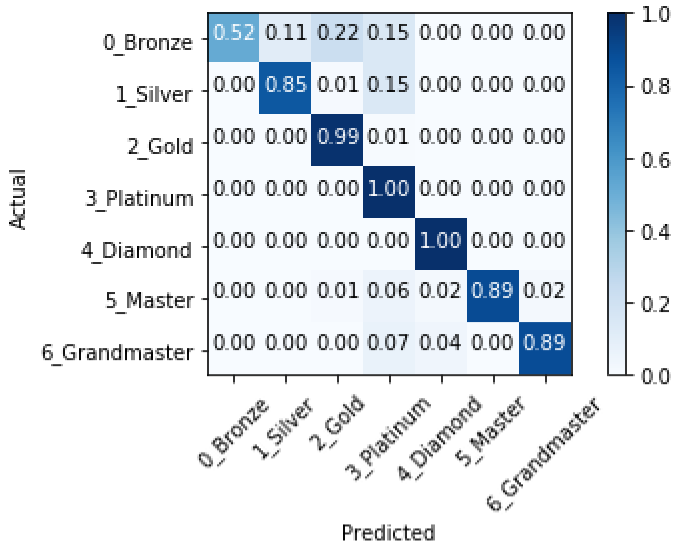


Figure 18: Random forest training confusion matrix (*valid*)

However, after outlier removal (following *valid* filtering), the random forest classifier *still* displays an overfitting confusion matrix for the training set, but outperforms the benchmark with 43.14% accuracy and .4452 balanced F1.

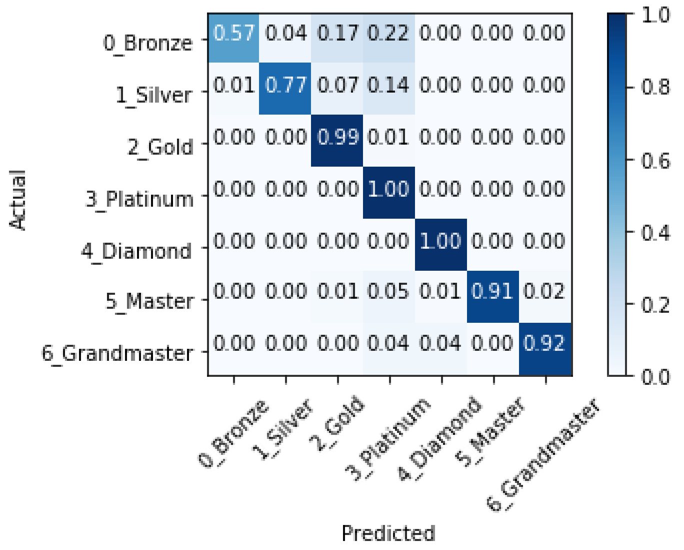


Figure 19: Random forest training confusion matrix (*valid*, outlier)

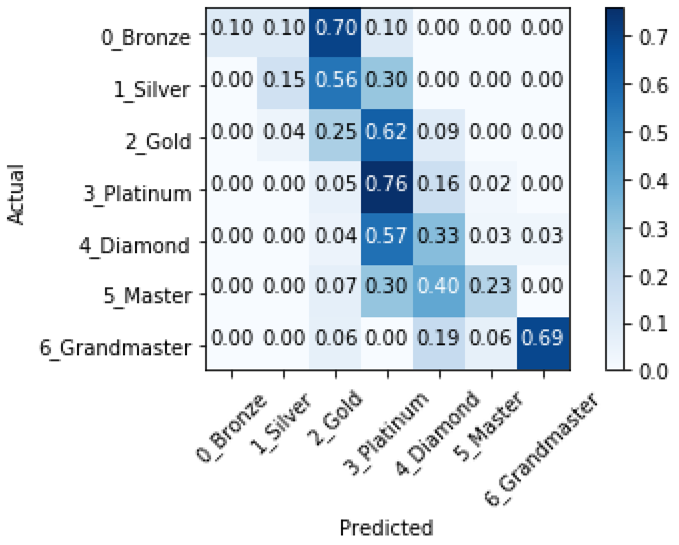


Figure 20: Random forest testing confusion matrix (*valid*, outlier)

It's important to note that throughout analysis of the supervised learners presented, performing well in Platinum classification can greatly increase the performance metrics in a total sum manner. However, the accuracy and F1 scores are normalized, so this is not the case with these findings.

Neural Network

The neural network classifier shows the greatest signs of overfitting during initial training. For the *valid* Tracer set with Time Played no less than 14400 seconds, the neural network initially predicts only Platinum.

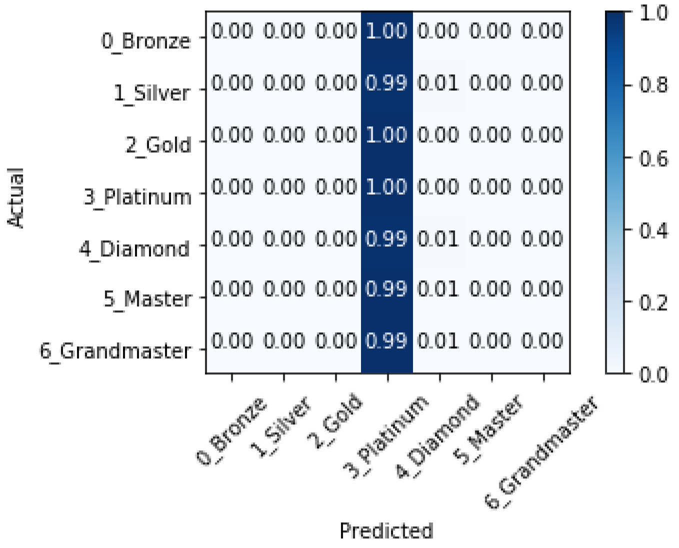


Figure 21: Neural network training confusion matrix (*valid*)

This learner produces metric scores 27.93% and .0468. Here are the hyperparameters:

- (32, 32, 32) node structured hidden layer
- Hyperbolic tangent activation function
- Stochastic gradient-based adam optimizer
- .0001 learning rate
- 5000 epochs
- .001 momentum

Despite clear initial signs of poor performance, the neural network outperforms the other learners after further preprocessing. After *valid* filtering, outlier removal, feature scaling, feature removal, and PCA, this classifier produces the following confusion matrices with a 45.82% accuracy and .4140 balanced F1.

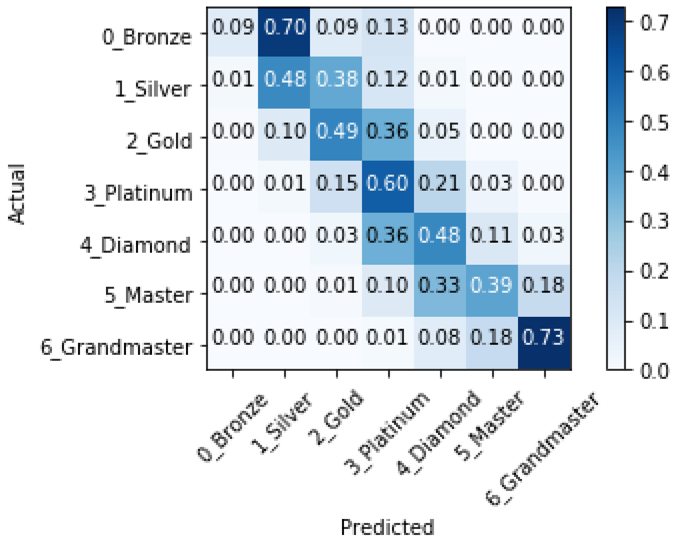


Figure 22: Neural network training confusion matrix (*valid*, outlier, scaling, removal, PCA)

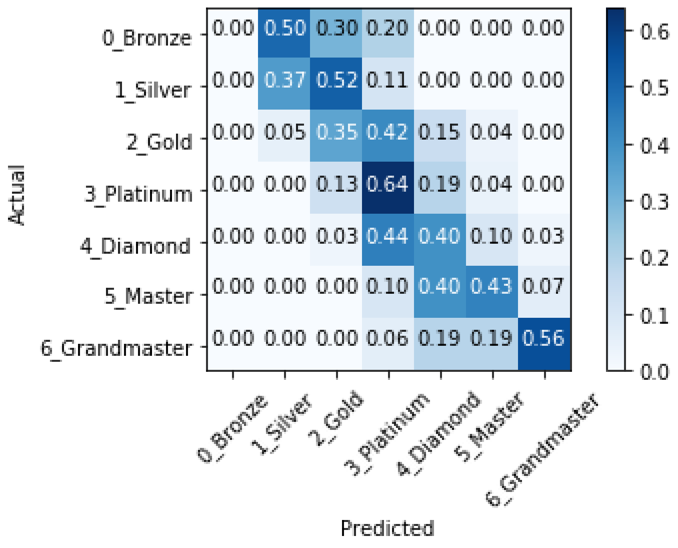


Figure 23: Neural network testing confusion matrix (*valid*, outlier, scaling, removal, PCA)

3. Results

Model Evaluation and Validation

Final Model

The best performing model from the supervised learners chosen is the neural network classifier. The entry count for the 14400 *valid*, outlier removed Tracer dataset (entries aren't removed in the remaining processing steps) is 1495, yielding an 80/20 train/test split of 1196/299. Although the random forest classifier showed very similar best performing metric values, the neural network's confusion matrix diagonalizes much better with respect to Silver and Master ranked players. However, it's important to note that few of the supervised learners generalize well to Bronze classification. This is most likely due to the dataset having the following distribution:

- Bronze: 33
- Silver: 111
- Gold: 274
- Platinum: 436
- Diamond: 342
- Master: 177
- Grandmaster: 122

In fact, each hero database has a very small number of Bronze players. With few data points to learn from, it's understandable that learners do not generalize well for this class specifically. Additionally, such a high *valid* criteria reduces the number of players meeting the Time Played threshold. Through experimentation, decreasing the Time Played threshold on the interval [0,4] hours, while increasing the dataset's cardinality, on average decreases metric scores. *Valid* filtering eliminates noisy entries that corresponds to players who do not have familiarity with a character. However, players of any SR level are free to play any character they choose, regardless of how often they play them. With this contextual knowledge, it can be assumed with confidence that reducing the dataset in this manner should not decrease the sought after generalization.

Generalizing Beyond Tracer

The objective is to classify SR based on hero-performance throughout Overwatch's 26 hero roster. To see how the neural network performs on different heroes, first look at Soldier: 76, who is the most picked DPS/overall hero in the dataset. His test metrics are 45.82% and .4140.

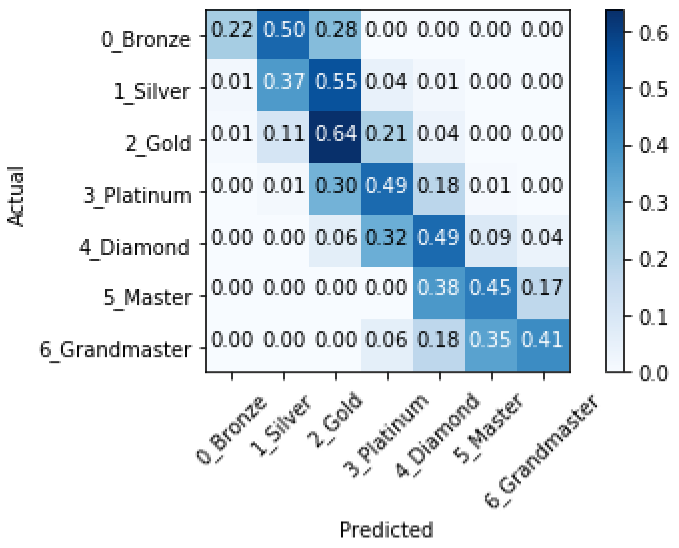


Figure 24: Soldier: 76 neural network testing confusion matrix (*valid*, outlier, scaling, removal, PCA)

The second most picked hero, D.Va, is a popular tank. The classifier learns to achieve 51.27% accuracy and .4927 balanced F1 in predicting her SR based on performance features.

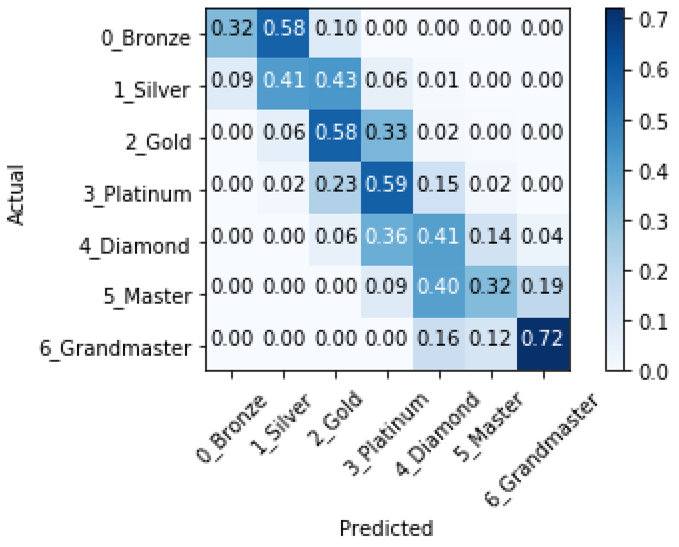
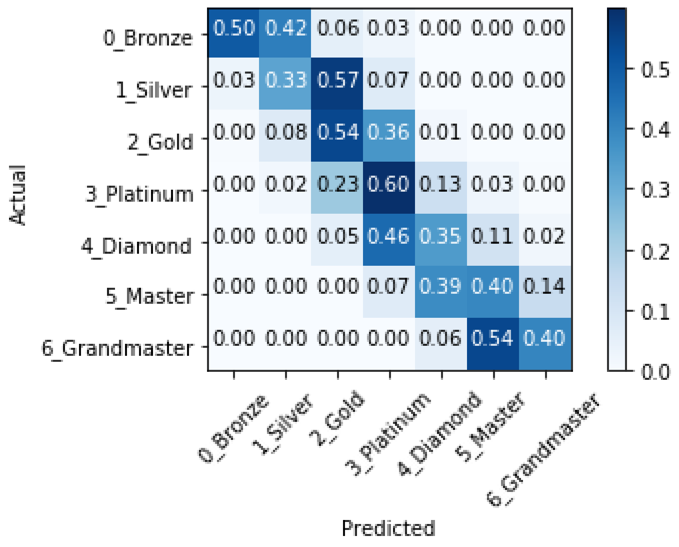


Figure 25: D.Va neural network testing confusion matrix (*valid*, outlier, scaling, removal, PCA)

Mercy, a commonly picked support hero, recently underwent big changes to her hero abilities. Because of this, possibly more than 20,000 of the Mercy entries in the database before processing correspond to new Mercy, whereas the remaining entries correspond to a Mercy profile obtained only with the original kit.

However, because her change is so recent, it's likely that player profiles do not yet reflect the change's impact, since most profiles are established by the larger portion of the season prior to her rework.

Regardless, the Mercy neural network, which undergoes *valid* filtering, outlier removal, feature scaling, feature removal, and PCA, achieves 48.32% accuracy and .4883 balanced F1. Also, notice how well she's classified for Bronze with respect to other heroes previously shown.



Actual \ Predicted	0_Bronze	1_Silver	2_Gold	3_Platinum	4_Diamond	5_Master	6_Grandmaster
0_Bronze	0.50	0.42	0.06	0.03	0.00	0.00	0.00
1_Silver	0.03	0.33	0.57	0.07	0.00	0.00	0.00
2_Gold	0.00	0.08	0.54	0.36	0.01	0.00	0.00
3_Platinum	0.00	0.02	0.23	0.60	0.13	0.03	0.00
4_Diamond	0.00	0.00	0.05	0.46	0.35	0.11	0.02
5_Master	0.00	0.00	0.00	0.07	0.39	0.40	0.14
6_Grandmaster	0.00	0.00	0.00	0.00	0.06	0.54	0.40

Figure 26: Mercy neural network testing confusion matrix (*valid*, outlier, scaling, removal, PCA)

Justification

Comparing the neural network's performance (45.82%, .4140) to the benchmark (26.60%, .2866) for Tracer, the neural network increases Tracer classification accuracy by 72.26% and balanced F1 by 44.45%. Although the benchmark undergoes little preprocessing with respect to the final solution, the benchmark decision tree actually performs the same if not worse after identical preprocessing, yielding 28.43% accuracy and .2088 balanced F1. Thus, it can be said with confidence that the final neural network model outperforms the decision tree benchmark.

However, the problem posed has not been solved. Although achieving close to 50% accuracy and .5 balanced F1 on *some* heroes is a significant increase in comparison to random guessing, for confident 7 class classification, greater performance is desired. But, given the distribution of the feature space and context of the problem, it's not surprising that higher metric values aren't achieved. This will be discussed more in the following Conclusion section.

5. Conclusion

Free Form Visualization

One of the greatest difficulties with this dataset is the feature distribution with respect to varying SR classes. Because of Overwatch's matchmaker, players are, for the most part, paired against teams of equal skill level. If matches were randomized so that Bronze players could play with Grandmasters, the dominant ability of high ranking SR players would be more prominent, whereas the lack of skill in low ranking players would also be more pronounced.

For example, suppose 2 Grandmaster McCree mains go into a 1v1 game mode variant, both of 4000+ SR level. Over 10 games there's a decent chance of 5 wins going to each McCree. However, if one of the Grandmaster McCree mains goes into a 1v1 match against a Gold tier Mcree main, whose SR is bounded above by 2500, it's very likely that the Grandmaster will take 8, 9, or even 10 wins. However, if the same 1v1 scenario is repeated between 2 Gold tier McCree mains, it's likely the matches will be split 5-5 as well. So, with a balanced matchmaking system, Gold and Grandmaster McCree mains can show similar win percentages.

(Note: An important detail here is that for extremely high SR players (~4500+), in order to gain the same amount of SR lost from losing 1 game, oftentimes 2-3 games need to be won. This does not apply to lower SR players.)

To visualize this issue, observe 6v6 win percentages for *valid* McCree players.

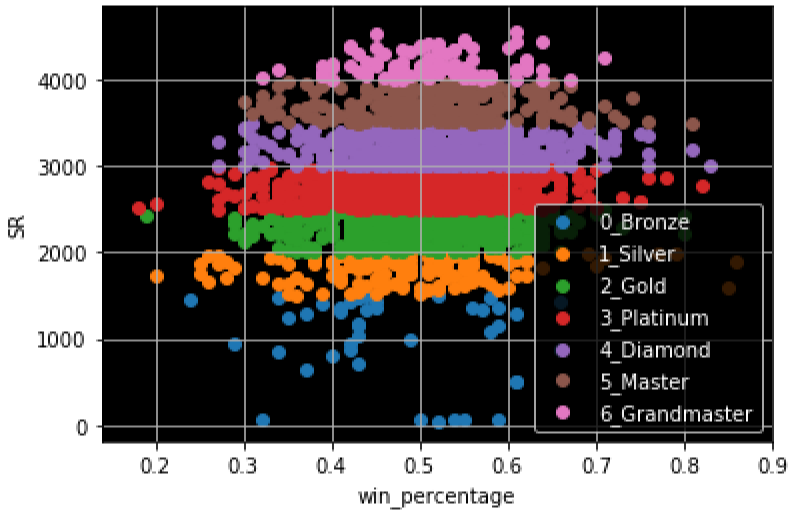


Figure 27: *Valid* McCree Win Percentage (scatter plot)

Regardless of the SR level, the win percentage for *valid* McCree players is roughly distributed evenly about [.25, .75], with a small decrease in lower and upper bounds for Grandmaster players. Because of this lack of feature value distinction between SR classes, learning how to discriminate between SR classes is difficult.

Reflection

Work Summary

The above work demonstrates the end to end workflow for an application of supervised learning in Overwatch. The investigation starts with data collection, where the first stage consists of collecting battletags from OverwatchTracker's competitive leaderboard. These battletags are then used to collect hero-specific information from PlayOverwatch's public battletag profiles. After cleaning scraped hero data, the data goes through a second cleaning for duplicate/noisy entry removal prior to supervised learning. Lastly, twice cleaned hero information is read into dataframes, where decision trees, random forests, and neural networks are fit for each hero. Refittings occur after each preprocessing step in order to judge the positive/negative influences each post-collection filter has on learning.

Points of Interest

The greatest difficulty in this work is decisively choosing models and tweaking hyperparameters. Although data collection takes the most amount of time to write/execute, the feature overlap exhibited throughout every feature and every hero makes this task difficult. Furthermore, the shared feature values across the SR range cause a great deal of overfitting, so identifying the correct hyperparameters to include and adjust is challenging. Using grid search helps alleviate some of this work, but because multiple learners are trained for multiple differently processed iterations of the same data, coding and evaluation are still time consuming.

Improvement

Testing multiple learners against the data is important, and because of this, preprocessing is currently written in a way which forces different learners to observe data using the same preprocessing steps. To improve, the code should be refactored to customize preprocessing for each supervised learner individually, in order to maximize metric scores per classifier.

For example, because each hero dataset is normally distributed with a mean SR of Platinum, battletags corresponding to Gold, Platinum, and Diamond profiles dominate the dataset in terms of entry count. Decision trees are known to grow in a biased manner when class frequency is imbalanced, and because of this, a decision tree-specific preprocessing step could prune the dataset to contain an even distribution of each SR class. In doing this, the hope is to see more diagonalization along the decision tree confusion matrices, especially for edge SR classes Bronze and Grandmaster.