StarCraft 2: An Analysis of Player Ranking

James Bladen and Steven Chang

May 9, 2014

Part I

Introduction

Source code and data may be found at github.com/stchang22/Stat222

StarCraft 2 is a one-on-one real-time strategy game where players gather resources to build up armies and go defeat their opponents' armies. When playing online against other people, players are classified into one of six possible leagues based on their wins and losses: Bronze (the lowest), Silver, Gold, Platinum, Diamond, and Master (the highest).

This project aims to focus on this classification aspect of StarCraft 2 (referred to as SC2 from here on out). There are two goals for the project: to determine which skills help set players apart from one another, and to predict a league for a new player. Casual players often want to know how skilled they are without having to spend the time to play enough games for an accurate representation of their skill level. Hopefully, this project will give these players a better idea of their skill without the drawbacks of the ranking system that is currently in place.

The current ranking is based off Arpod Elo's rating system that was originally created as a way to improve chess rating system. This method has been adopted by many sports and games. The basic idea of the rating system is that you are assigned a numeric value for your rating, and when playing against other players your rating goes up or down based on whether you win or lose. When playing against a player with a higher rating, you are able to win more points than when you play against an opponent with a less rating. The general idea is that if you are playing against a 'better' opponent you should be able to win more, and should not be punished as hard when you lose. In SC2 they assign leagues based off of your current ELO rating, essentially binning them into 6 categories for each of the 6 leagues.

There are some drawbacks that come along with this particular rating system, which this project is going to try to find solutions for. There are three main problems that occur in the implementation for SC2. First, it takes many games in order for your rating points to accurately match your skill level. In the short term, there is a lot of variability in your rating so it is hard to use it as a reliable measure of your skill. Secondly, if you happen to have a rating that is higher than your actual skill level, the system encourages you to stop playing in order to protect your rating. Because the ranking is based purely off wins and losses, if you stop playing your ranking will stay the same. The last problem with the ranking is that due to inflation of points, it is difficult to use it to compare your skill to somebody from a previous time period. The ranking is what is called a zero sum game. When someone gains points, someone else loses that many points. So with more people playing the game, there are more points available and thus point inflation happens. Deflation can happen due to players joining the leagues as new players and leaving once they have already gotten many points. Very good players who stop playing will take away many points from the overall pool of players, never to be seen again. Thus, when comparing your rating with someone from a different time period, you have no idea what your point value truly means compared to a point value from a different time period.

Our goal in this project is to come up with a way of determining player skill that circumvents these issues. Ideally, we want to find some variables that account for actual player skill better than relying on wins and losses. That way we can group players based on skill level, which is the whole point of a rating system. We would also want to be able to do this without requiring many games to be played. We could then even use this skill level measure to see trends over time of the overall playerbase. You could actually look at your in-game statistics and compare them to someone else's, regardless of when that person played compared to you.

Part II

Data

Our analysis is based off of a dataset found on the UCI repository. Players were asked through a forum to donate a replay of one of their games. Replays were then sent through a processing software to convert them into in-game statistics. The dataset originally consisted of 3,395 observations (in this instance, a unique replay is an observation). We had to drop about 100 replays because either there was missing information or some of the information was unreasonable. For example, players were asked how many hours in total they played the game and one player reported 1,000,000 hours. This converts to over 114 years which is impossible to be true. All in all, there are 19 variables that could be useful: the response variable is the league a player is placed in, and the remaining 18 variables are a mix of personal information (age, hours played per week, and total hours played) and statistics gathered from the replay.

Part III

Distribution of Observed Leagues

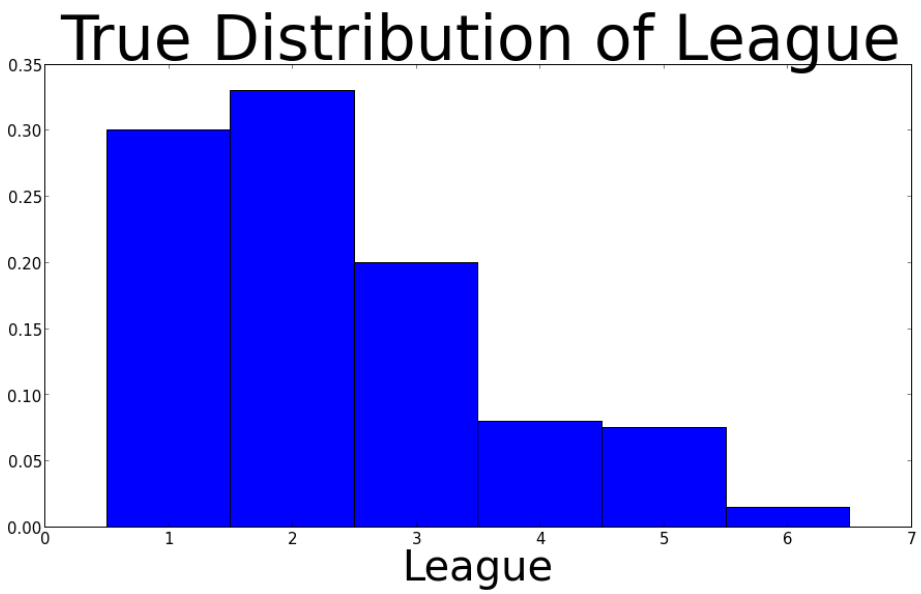

Fig. 1) Note that the true distribution of players is skewed to the right. This means that there should be more lower-level players. This data was collected from sc2ranks.com.

Our replays came from players who voluntarily submitted their games to the forum. This means that our data is not a simple random sample. The types of players who frequent a forum and would also be willing to donate a replay are probably not representative of the player population. This is seen quite clearly in these histograms of the distribution of our data versus the distribution of the population. Our data is biased towards having many more players in the higher leagues because those players are the types that would be more willing to send in their games. This means that we have to deal with a biased sample and make sure to use models that will not be heavily influenced by this. Any model that requires the probabilities of being in a certain class would probably fail due to our data being so heavily skewed.

Fig. 2) Notice that we have the opposite skew for the data that we actually collected. This means that our data will be slightly biased towards better players, thus making our predictions slightly skewed towards larger values.

Part IV

Variables Used

The software that was used to process the game replays did so by discretizing them into screenshots which were then used to count game actions. Most variables in the dataset are just averages of those actions over all screenshots. These screenshots are referred to as Perception Action Cycles (PAC). These PAC's refer to when a player is currently viewing some part of the map, so a single PAC is the player's current view. If the player starts to view somewhere else in the game that would be considered a new PAC. That gives rise to some more variables like how often do they change their PAC, and how long does it take for players to make their first action once they are in a new PAC.

If you look at the game screenshot below, take note of the bottom left of the image. That image is called a minimap. In terms of sports, it would be considered as the entire field of play. The point of dividing things into different screenshots is that in this game you cannot actually view the entire field at once. You can only view small sections of the field, so the idea is that we want to know what kinds of things a player is doing while viewing a certain part of the playing field.

Out [33]:

Another important definition to understand is something called a hotkey. Hotkeys are used by assigning a certain keyboard key to perform an in-game action. The in-game actions can range from selecting a groups of units, having those units move to a location, having them attack something, having them perform a special attack, and so on. The idea is that it is much faster to automatically select a group and then tell them to do something with two key presses rather than having to find them on the map, select them all with your mouse, then clicking the button that refers to that action. We believe that a player's use of hotkeys should do well at determining skill and thus a few of the variables used are directly related to how a player handles hotkey usage. In terms of the image above, note the bottom right portion of the screen. That shows you all the actions that the selected unit can take. When actually playing the game, the time it would take for you to find your unit on the map and then click the button for the correct action is precious time that can be saved using hotkeys. In a game where speed counts, every little advantage you can get on your opponent should lead to more wins.

Here is a list of all the variables created for this data set:

1. GameID: Unique ID number for each game (integer)
2. LeagueIndex: Bronze, Silver, Gold, Platinum, Diamond, Master, GrandMaster, and Professional leagues coded 1-8 (Ordinal)
   This is our response variable. It is an ordered variable that attempts to predict how skilled you are right now.
3. Age: Age of each player (integer)
   We are curious if certain age groups are better players than other groups. It is widely believed that older players tend to perform worse than younger players.
4. HoursPerWeek: Reported hours spent playing per week (integer)
   A possible measure of how dedicated the player is to improving.
5. TotalHours: Reported total hours spent playing (integer)
   A measure of how much experience a player has playing StarCraft 2.
6. APM: Action per minute (continuous)
   Average number of actions you make throughout the game. Many players believe this is the only or best measure of skill.
7. SelectByHotkeys: Number of unit or building selections made using hotkeys per timestamp (continuous)
8. AssignToHotkeys: Number of units or buildings assigned to hotkeys per timestamp (continuous)

9. UniqueHotkeys: Number of unique hotkeys used per timestamp (continuous)
   We believe the more you use hotkeys the more skilled you are as a player.
10. MinimapAttacks: Number of attack actions on minimap per timestamp (continuous)
11. MinimapRightClicks: number of right-clicks on minimap per timestamp (continuous)
    The use of the minimap is an alternative to moving your entire view. It requires some amount of foresight to use properly.
12. NumberOfPACs: Number of PACs per timestamp (continuous)
13. GapBetweenPACs: Mean duration in milliseconds between PACs (continuous)
    These two are related to map awareness. The more the player scouts the more information the player has about his opponent.
14. ActionLatency: Mean latency from the onset of a PACs to their first action in milliseconds (continuous)
    A measure of reflex time. It may also signify how much the player has planned ahead of time.
15. ActionsInPAC: Mean number of actions within each PAC (continuous)
    The higher this variable is, the more likely the player is just looking at one particular area of the map.
16. TotalMapExplored: The number of 24x24 game coordinate grids viewed by the player per timestamp (continuous)
    A measure of map awareness. It is generally believed that scouting provides useful information about the opponent.
17. WorkersMade: Number of SCVs, drones, and probes trained per timestamp (continuous)
    Vital for establishing a strong economy. Strong economies tend to allow players to build stronger armies.
18. UniqueUnitsMade: Unique units made per timestamp (continuous)
    A measure of how diverse a player's army is. A more diverse army is harder to defeat but often requires more maintenence.
19. ComplexUnitsMade: Number of ghosts, infestors, and high templars trained per timestamp (continuous)
20. ComplexAbilitiesUsed: Abilities requiring specific targeting instructions used per timestamp (continuous)
    When used properly, complex units and spells may provide additional advantages over the opponent.

Part V

Trees

We attempted to address variable selection by utilizing decision trees. Through some preliminary exploratory analysis we discovered that skills which differentiate players of two particular leagues may not be the same skills that differentiate players of two other leagues. Intuitively, this happens because players of the lower leagues are at a different stage of learning the game versus players of the higher leagues. Because of this, we made the decision to focus on every combination of two leagues rather than looking at everybody at once. For each combination (referred to as a 'matchup' from here on out), we created a classification tree. The idea behind this choice is that by looking at each matchup individually, perhaps we will find some trends in the important variables that are used in separating the players in each league. Then we will take some aggregate of those variables and use them in our classification method.

We used 10-fold Cross Validation to find two optimal parameters: the minimum number of samples required to split an internal node and the minimum number of samples required to be at a leaf node. We used a zero one loss function to evaluate each pair of parameter values.

We will now take a look at the trees of four particular matchups to get a sense of which variables the trees are utilizing. This first tree represents the comparison of the two lowest leagues.

Out [23]:

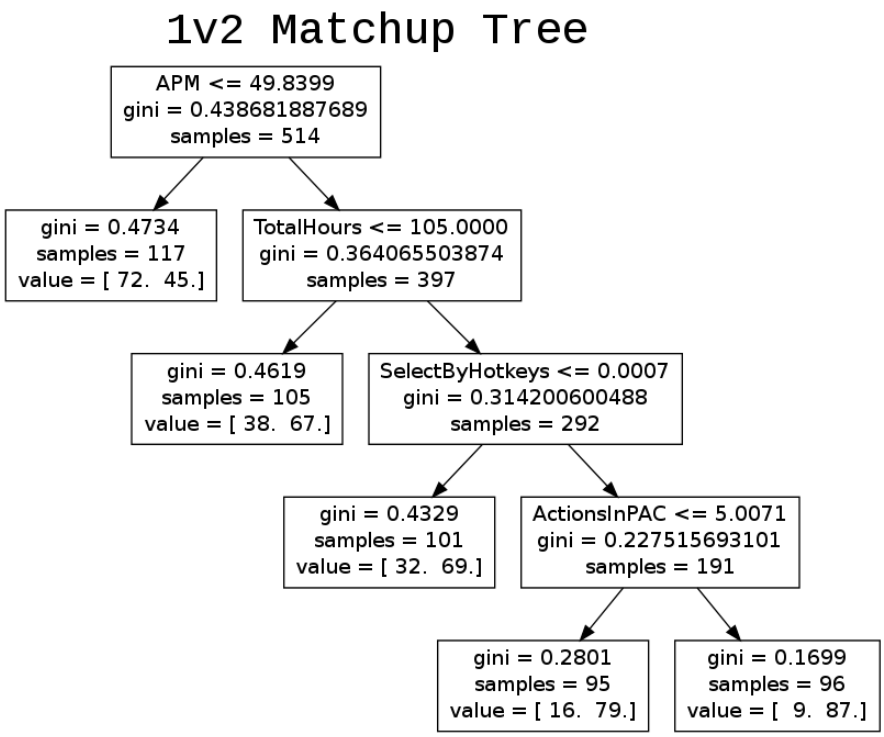

Fig. 3 This tree features a matchup of the two lowest leagues. Note that APM is the first variable the tree splits on.

The first thing to notice is that APM is the first variable that the tree splits on. APM, or Actions Per Minute, is the average number of actions a player makes per minute. In other words, APM measures how quickly you move and make actions throughout the game. Further down the tree, notice the value for SelectByHotkeys is really small. This suggests that if you select something using a hotkey even once you are more likely to be in the higher league, thus matching our intuition that using hotkeys is related to player skill.

This next tree represents the comparison of a low league and a middle league.

Out [22]:

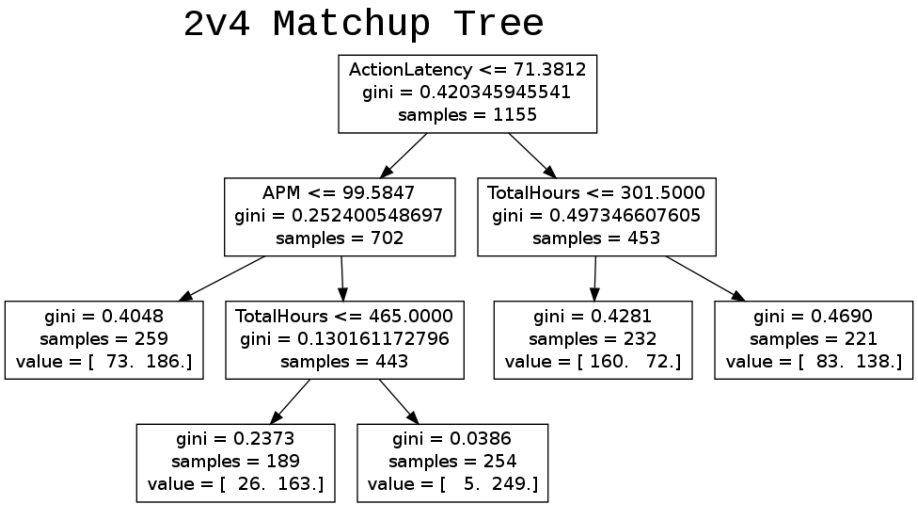

Fig. 4) This tree features a matchup of a low and middle league. We see that APM is still present, but ActionLatency has become the first variable the tree splits on.

We see that APM is still present in the tree, but it is no longer the first variable the tree splits on. Rather, the tree splits on ActionLatency first. ActionLatency refers to the time it takes from starting a new PAC to making an action. Furthermore, the tree consists of only three variables and two of them are measures of game mechanics. ActionLatency and APM combined with TotalHours suggest that the difference between these two leagues is sheer amount of practice.

This next tree represents the comparison of a middle league and a high league.

Out [21]:

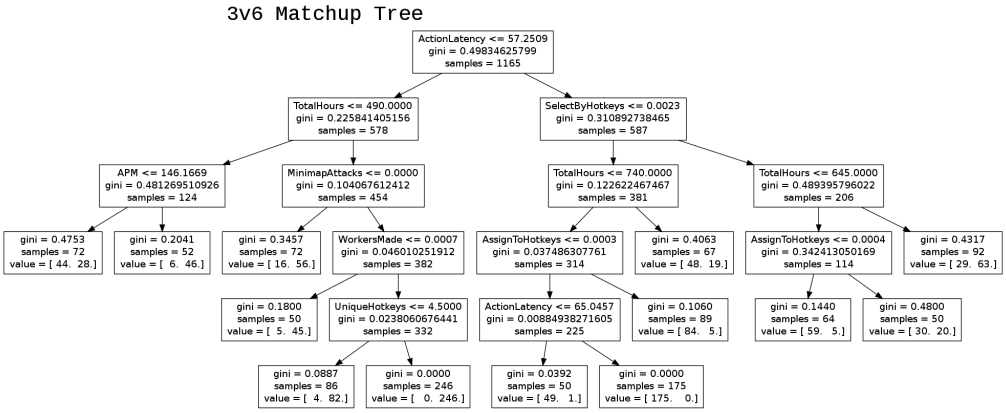

Fig. 5) This tree features a matchup of a middle and high league. Notice how it is more complicated than the previous two trees. New variables involve the use of hotkeys.

ActionLatency is the first variable the tree splits on, like the previous tree. The most notable thing about this particular tree is the amount of splits required to get a decent separation. The complexity of the tree represents the difficulty our variables have in quantifying the difference between medium players and good players. From personal experience, medium level players tend to have similar if not the same mechanical skills that high players have but they lack experience in making decisions. This type of difference is not something that our variables do a good job of taking into account, hence why there are so many required to separate the leagues. Notice that there are multiple variables that concern the use of hotkeys that were not present in the previous two trees. It suggests that higher level players are more comfortable at using hotkeys and thus are more likely to queue more complex sequences of actions.

For the final tree, we compare the two highest leagues.

Out [20]:

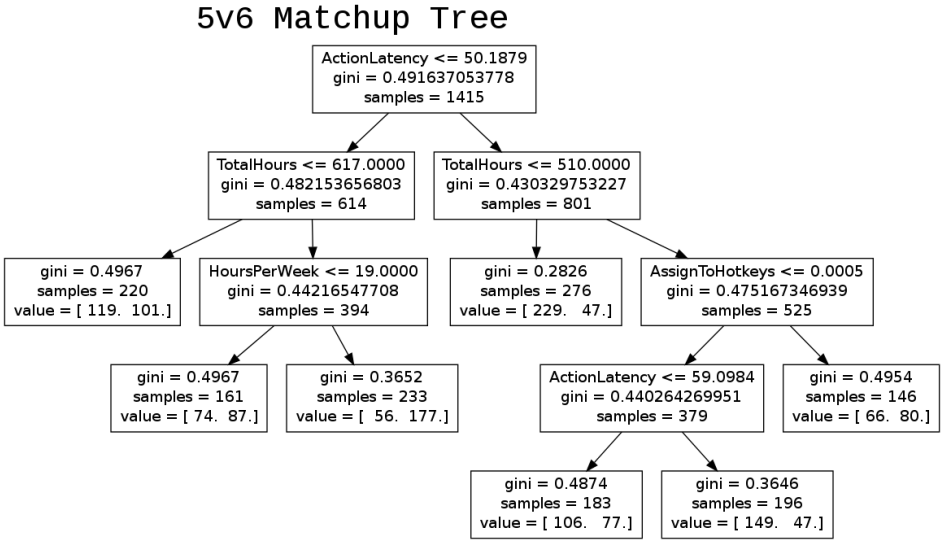

Fig. 6) This tree features a matchup of two high leagues. It is not as complicated as the previous tree. ActionLatency is still the first variable the tree splits on.

The tree is not as complicated as the previous one. ActionLatency remains at the top, but note that TotalHours and HoursPerWeek are present once again. What this suggests is that the biggest difference between the two leagues is just experience and the amount of time players put into the game. This makes sense because their skill levels are quite

similar mechanically, thus the differences come from smaller details that are gained through playing the game more often.

To give the reader a summary of all the matchups, we created a table that lists the three most important variable used by each of the trees.

Important Variables for Each Matchup

1v2	APM	SelectByHotkeys	TotalHours
1v3	APM	NumberOfPACs	TotalHours
1v4	APM	ActionLatency	TotalHours
1v5	APM	ActionLatency	TotalHours
1v6	APM	TotalHours	ActionLatency
2v3	NumberOfPACs	TotalHours	APM
2v4	ActionLatency	TotalHours	APM
2v5	APM	ActionLatency	TotalHours
2v6	APM	TotalHours	ActionLatency
3v4	ActionLatency	TotalHours	MinimapAttacks
3v5	ActionLatency	SelectByHotkeys	TotalHours
3v6	ActionLatency	TotalHours	SelectByHotkeys
4v5	ActionLatency	APM	TotalHours
4v6	ActionLatency	TotalHours	SelectByHotkeys
5v6	ActionLatency	TotalHours	AssignToHotkeys

Table 1. Variables are arranged so that importance decreases from left to right. We see that in the lower leagues, APM is deemed most important. As we go up to higher leagues, ActionLatency becomes more important. This makes intuitive sense because lower players are still learning the mechanics of the game while at the higher levels, players have more or less the same level of mechanics; it is more a question of reflex times and map awareness that make an impact.

The table is structured so that importance decreases from left to right. Notice that for all the matchups concerning the lowest league APM is the most important variable. As we go up in leagues, ActionLatency becomes more important. There is a clear shift in the trend of variables which confirms our hypothesis that different matchups require different variables to describe them.

Part VI

K Nearest Neighbors

Now that we have an understanding of the variables that are important for each of the leagues, we want to classify players into leagues using these variables. We are looking to find a method that circumvents the problems that the current ranking system has. For reference, the issues that we have with the current ranking systems are threefold. First, it requires too many games to give a reliable result. Second, players can stop playing in order to protect their current ranking. And third, it is hard to use ranking to compare your skill to players from previous times due to inflation of points. Ideally, we would want a model that addresses all 3 of these issues.

Since our dataset involves only a single game per person, if we can find a method to classify that is at least fairly reliable then we have addressed the first concern. The second and third concerns are due to the way they use point values to keep track. So rather than grouping players by point values, we should try to group them by skill level. We want players of similar skill statistics to be classed into similar leagues, thus the intuitive model choice here is K nearest neighbors (KNN). This method is especially useful because it requires no distributional assumptions and the effect of our biased sample is not going to be so bad. Remember that our sample came from people who volunteered their games, and we had a heavy bias towards higher leagues. KNN does a fairly good job of not giving too biased of predictions even with that kind of sample.

Now we want to get a sense of the way that KNN will behave for our dataset. Since it is impossible to plot out all the variables used, we decided to plot on ActionLatency and GapInPAC. GapInPAC is essentially looking at how often

a player moves the screen around to look at new things while ActionLatency is how long it takes the player to do something after moving the view to a new part of the map. Together they should be at least somewhat representative of a player's multitasking ability. Intuitively, better players will have smaller values for GapInPAC and ActionLatency, meaning they are moving around a lot and acting fast. We should expect then for this result to show up in the following KNN decision plots.

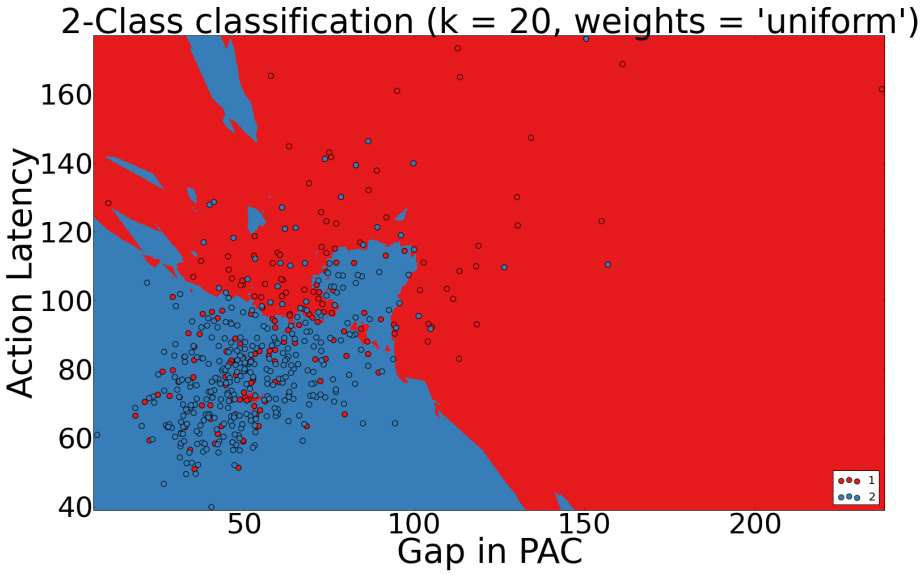

Fig. 7) The point of this graph is to show you that when you are looking at two classes that are right next to each other, then you see a lot of overlap in the data. Here, the red color is class 1 (bronze) and the blue color denotes class 2 (silver). It would seem that KNN will not really do a good job of accurately being able to distinguish between any of the classes that are one apart. This should be pretty obvious, since players from classes right next to one another probably have similar skill levels, and perhaps have not converged to their true league yet.

This plot uses a subset of the data containing only players from class 1 (the worst) and class 2 (second worst). As far as the boundaries go, it follows our intuition that the better league is the one closer to the origin (smaller values for both variables). However, when it comes to leagues that are right next to each other it seems like there is a very large overlap of the players. This means that players who are in leagues one off from another (i.e. leagues 1 and 2) tend to have similar skill statistics. In terms of ranking, this could be due to the fact that they are not yet in their 'true' league because they haven't played enough games to get there. So our belief then is that the KNN model is grouping people on their current skill which does a better job of predicting their true skill than their wins and losses. As long as our variables are representative of a player's skill, then this model is doing very well.

This next plot is made to contrast the one from before.

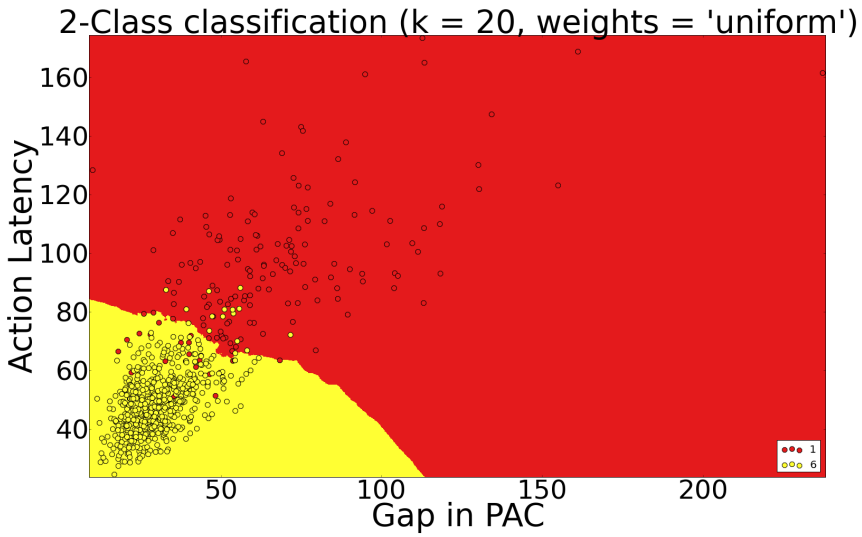

Fig. 8) This graph is supposed to be parallel to the one previously, but now we are looking at two classes furthest apart from each other. You see that this time the boundary does a much better job of identifying the two. Here, the red color is class 1 (bronze) and the yellow color is class 6 (diamond). Even with the classes being very far from each other, there are still some data points that are categorized wrong. What this suggests is that perhaps these two variables are not enough to distinguish between players, and are not a great representative of player skill. Our actual model uses more than just these two variables, so it is okay for this not to be perfect at classifying here.

This one contains only players from class 1 and class 6, which are on the opposite sides of the skill spectrum. Notice here that the separation is much more clearly defined and there is a lot less misclassification. This is to be expected because even if the current ranking system is not so good, it is not extremely off. Saying that someone currently in league 1 is actually playing at a league 6 level is a pretty huge statement. Luckily this does not seem to happen too often so our model is not completely contradicting the current ranking, which is good. Theoretically, we should expect to see that the full KNN model should have many similar predictions and many slight differences, but not too many large contradictions with the current rankings.

Part VII

Building the Model

Now we fit a KNN model by cross validating over K, using the zero-one loss function. We used the variable selection done in the first section and selected the following 5 variables:

1. TotalHours: Reported total hours spent playing (integer)
2. APM: Action per minute (continuous)
3. AssignToHotkeys: Number of units or buildings assigned to hotkeys per timestamp (continuous)
4. MinimapAttacks: Number of attack actions on minimap per timestamp (continuous)
5. ActionLatency: Mean latency from the onset of a PACs to their first action in milliseconds (continuous)

This is the confusion matrix that compares our predicted league versus the player's current league.

KNN Confusion Matrix

	Pred=1	Pred=2	Pred=3	Pred=4	Pred=5	Pred=6
League=1	1.2	1.6	1.2	1.1	0.0	0.1
League=2	0.6	3.1	2.9	3.2	0.7	0.1
League=3	0.1	1.7	5.5	5.9	2.7	0.7
League=4	0.1	1.4	3.1	11.0	6.8	2.2
League=5	0.0	0.2	1.2	6.6	10.8	5.6
League=6	0.0	0.0	0.2	2.2	6.4	9.9

Table 2. This is a confusion matrix to compare the values of the predicted league compared to the current league league of the player. The values inside the matrix represent the percentage of the data that is in that category. So this means that any data in the diagonal of the matrix would be where our method is the same as the current league of the player and sums up to about 40% of the data. Also keep in mind that since the response variable is ordered, the level of misclassification is different. For example, if we predict someone plays in the same way that players of league 6 do but in reality their current league is 1, then we are believing that the current ranking is extremely wrong. Since we can assume the current ranking israrely extremely off, we would want very few data points to be in the bottom left and top right corners of the matrix. This is the case, although we do tend to class people as being better (to the right of the diagonal) than their true league compared to worse (to the left).

The values inside the matrix are the percent of players in that particular cell. The variable on the top is our predicted league using KNN, and the variable on the left is the current league. That means that any data that falls along the diagonal of the confusion matrix is when our model is in agreement with the current rank. This sums up to 40% of our data that we consider to be in the correct league at the moment.

Since our data is ordered, the level of difference between predicted value and current rank is different. If someone is currently rank 6 and we think they are playing at the same level as rank 1 players, then we are stating that the current ranking system is immensely inadequate at predicting skill. So in reference to the matrix, this means any data in the upper right and lower left corners signifies a very large difference in what we predict versus current ranking. We said previously that we expect for this to happen very rarely, which is indeed the case if you check the values: 80% of the data lies either on the diagonal or directly off the diagonal.

Part VIII

Conclusion

We found several interesting absences of variables from the matchups. For starters, Age was never one of the top three variables for any tree to split on. Typically, the belief is that older players have slower reflexes which would put them at a disadvantage but this did not show up prominently. None of the in-game variables (i.e. WorkersMade, UniqueUnitsMade, etc.) showed up as well, signifying that there is no optimal strategy that leads to becoming a better player. Flipping it around, we were surprised to see TotalHours show up in many of the matchups. This is actually a bad thing because time is not a measure of the quality of play: a player may spend many hours playing horribly but our model would assume the opposite, that the player spent many hours playing very well.

If we were to improve the study, we would want to try and find a more random sample. In particular we would like a sample that reflects the current league distribution, so we would want more lower league players and less higher league players. In addition, we would want to try more advanced methods such as random forests and support vector

machines although those methods would lack interpretability.

In conclusion, defining skills are heavily based on context; there is no one variable that can be used well in every matchup. For matchups involving lower league players, APM is very important. As the leagues go up, ActionLatency becomes more important. Our KNN model does a good job at predicting the true league of a player while maintaining a simple and intuitive interpretation. Players may now parse their own replays and send the output through our KNN model to see which league matches their current skills. We have successfully created a method of predicting player skill that does not have the problems that the current ranking system has.

Part IX

References

Mark Blair, Joe Thompson, Andrew Henrey, Bill Chen. 'SkillCraft1 Master Table Dataset Data Set'. http://archive.ics.uci.edu/ml/datasets/SkillCraft1+Master+Table+Dataset#.

'Scikit-learn: Machine Learning in Python', Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.