Machine Learning Nanodegree Capstone Project: League of Legends Player Skill Classification

David Garwin

October 14, 2019

Contents

1 Definition

1.1 Project Overview

League of Legends (LoL) is a popular, multiplayer online battle arena (MOBA) game. There are multiple game modes available, but the one studied here is known as "Ranked Solo Queue 5v5", (referred to as "Ranked" here). In Ranked, people are on one of two teams, five players each. Each player controls a single character (champion). The goal of each team is to kill each others characters and ultimately win by destroying the enemy's home base (nexus). Ranked games are set up with LoL's ranking algorithm, attempting to match people of similar skill. Skill is measured using a type of categorical elo system, where players are put into different "tiers" as they gain and lose elo (the rough equivalent to elo, "League Points"), proportional to the relative skill level of players om each team.

While there is nothing wrong with using an elo system to measure a player's skill, it would be illuminating to measure player's skill using other, likely multidimensional, measures. In this project, we assume that a person's rank is a is a measure of skill that can be measured ignorant to the relative skill of other players. With a grasp of what characteristics make a good LoL player, one can then then use this data to solve the questions almost every competitive player (myself included) tries to answer: how can I get better?

1.2 Problem Statement

Given a LoL player with a ranked tier, we want to be able to predict what that ranked tier is without knowing what it is, and without knowing the tiers of the player's opponents. This problem has a great many challenges.

1.2.1 Getting Data

Data is obtained from the LoL public web API (the API)[1]. The API consists of many endpoints, each with filter and extended data options. This is detrimental for two reasons. The first is the API is not designed for massive data studies (restrictive rate limits, no bulk data fetch). The second is there is just so much data, with so many options, that exploring just the different types of data is a project well beyond the scope of a Nanodegree project.

1.2.2 Preprocessing Data

Assuming there exists a black box that will tell exactly what data to fetch from the API, there is still the issue of parsing and preprocessing the data. The data consists of both categorical and numeric data that is always time dependent and different values are often incredibly correlated. This data is far too raw for any machine learning algorithm to understand without significant and preprocessing, making this an involved and critical step.

1.2.3 Create a Classifier

Once the data is preprocessed, a classifier must be trained and tuned. Models with different types and parameters will be explored and vetted to produce the model that best predicts a player's ranked tier. This cycle is repeated as many times as it takes to get satisfactory classification performance.

1.3 Metrics

Accuracy will be used to judge a classifier's performance, defined to be:

$$Accuracy = \frac{\#\text{ of players assigned their proper tier}}{\text{Total }\#\text{ of players}} \tag{1}$$

Such a metric is poor for a few reasons. One reason is accuracy does not tell us how poorly we perform on a poor tier level (is one tier predicted disproportionately high?). Another reason is it simply doesn't tell us how well we're doing on a per class level (how good are we at predicting Diamond players?). Finally, it does not capture the severity of the misclassification (Bronze misclassified as Diamond is much worse than Bronze misclassified as Silver). To handle the first to

issues, we run per-tier calculations on the top performing model and try to understand more about how it performs per tier. We calculate Precision_T and Recall_T for each tier T, defined as:

$$\text{Precision}_T = \frac{\#\text{ of players correctly classified as in tier}T}{\text{Total }\#\text{ of players classified (correctly or incorrectly) as in tier}T} \tag{2}$$

$$\text{Recall}_T = \frac{\#\text{ of players correctly classified as in tier}T}{\text{Total }\#\text{ of players actually in tier}T} \tag{3}$$

These metrics help us understand how well we're classifying members of each class (precision) and how often we predict that class (recall).

2 Analysis

2.1 Data Exploration

2.1.1 Data Selection

As implied above, there is more data available than can be reasonably expected to be analyzed and used. To handle all of this data, a number of restrictions were made to it before analysis even began. These restrictions are detailed below.

Sample Distribution As stated, we attempt to collect data whose tier distribution matches the general population[2]. This is used to incorporate prior probabilities into the data set.

Tiers Used There are seven ranked tiers; in ascending order: Bronze, Silver, Gold, Platinum, Diamond, Master, Challenger. The vast majority of players (>99.9) of players are in the first five tiers. Because there are so few of these players, and that generating enough data to be able to classify them, with the constraint above, data collection would take a tremendous amount of time. Therefore, only the first five (Bronze to Diamond) tiers were considered.

Player Selection Due to the limited nature of the API, it is difficult to get a wide sample of players. Players were obtained by starting at a root player, finding other player from past games, and recursively finding more players in the same way. At the end of this selection, the tier distribution of players matched the general population.

API Endpoints Used Due to restrictions on the API, the only API endpoints used for feature selection was the "Recent Games" endpoint, which returns get basic statistics about a single player and his or her team for the player's most recent (up to 10) games. Other endpoints that return more comprehensive game data, including other players and timeline game data, were not used. Severe rate limiting of the API made it practically impossible to collect a large enough sample of data.

Regions Considered Players can only be matched to play with others in the same region (North America, Europe, Asia, etc). A side effect of this is the skill of a player in one tier, might not correlate with the skill of another player in the same tier, but in a different region. To reduce this potential for error, only the North America region was used.

2.1.2 Data Structure

Over the course of approximately 10 hours, 15,000 players had their recent match histories pulled; how this was done is mentioned above under "Player Selection". The API documentation for the method used[1] can be found at the LoL website[1]. Each player has at most 10 games from this endpoint. Players with fewer than 6 ranked games were not included in this 15,000 players. Only ranked games were used in analysis.

For each game, the API returns a **stats** object, containing 55 values related to the the player and the player's teams's performance during the game. These values include features such as: number of player deaths, number of player kills, and whether the player's team won. The data can be broken down into a few main categories:

[1] /api/lol/region/v1.3/game/by-summoner/{summonerId}/recent

Kill, Death, Assist Counts and Summaries This includes both simple counts of kills, deaths, and assists, for both players and objectives[2], and more some more detailed counts. Detailed counts include number of killing sprees[3] and largest multi-kill[4].

Damage Summaries These variables include data such as how much damage was received, dealt, and healed. These features missing indicate a value of zero. For example, if **magicDamageDealt** is not returned from the API, the player did no magic[5] damage for the entire game.

Items Purchased There are seven features that enumerate the final items[6] players may have at the end of a game. If the game ends and the player has fewer than seven items, some features will not be present.

Gold Summaries This includes how much gold (in-game currency) was earned and spent. As with other fields, these being missing from the API indicate a value of zero.

Player Role and Player Position In LoL, like with most sports, players have different roles[7]. Furthermore, they can fulfill these roles in different positions on the map[8]. If a player does not, at the beginning of the game, indicate the role and position he or she would like, these values will be empty.

In the the appendixA you can find summary statistics for data points over all players. Upon inspection of the data, some interesting features are found. The only feature of the data that was explicitly addressed in preprocessing is the wide range of order of magnitude of variables. Some variables are on the order of one (barracksKilled), while others are on the order of 10,000 or 100,000 (magicDamageDealtPlayer). SVMs[3] and neural networks[4] perform better using normalized feature sets, which we obviously do not have. Another feature of the data is that there are multiple distribution functions for features. Some have hard upper and lower limits (team), many of which have bounds of simply zero and one (playerRole, win). Other data appears normally distributed (totalMagicDamageDealt). And some data has what could be considered outliers (pentaKills). This is not necessarily a bad thing, merely an observation. Further research would be needed to understand the implication of including or excluding features with certain distribution characteristics.

2.2 Exploratory Visualization

2.2.1 Data Distribution

One of the most important things to understand about this data set is how the classes are distributed. As seen in 1, the distributions of the bottom three tiers (Bronze, Silver, Gold) have, at the very least, the same order of magnitude frequencies in the population. However, the Platinum and Diamond divisions contain far fewer total players; there are over twenty times as many Silver players as there are Diamond players.

2.2.2 Feature Correlations

It is suspected that many of the features are closely correlated, as many features are sums of other features. For example: one can determine the largest multi-kill by looking at the number of double kills, triple kills, etc. Therefore in 2, we analyze correlations between games. To get this figure the data was first preprocessed (see3.1). The preprocessed players had Pearson correlation coefficients calculated for each pair of variables. As it can be seen, many variables are highly correlated, as expected.

[2]Objectives are structures that teams destroy to help win the game. The nexus, towers, and inhibitors are all objectives.

[3]A killing spree is a series of kills without any deaths in between kills.

[4]A multi-kill is a collection of kills in quick succession.

[5]Damage can be either physical or magical in nature.

[6]An item is something that can be purchased using currency earned as the match progresses. A player can have up to seven items, but often have fewer.

[7]Like in baseball there are pitchers, catchers, basemen, etc.

[8]Think first basemen, second basemen, etc.

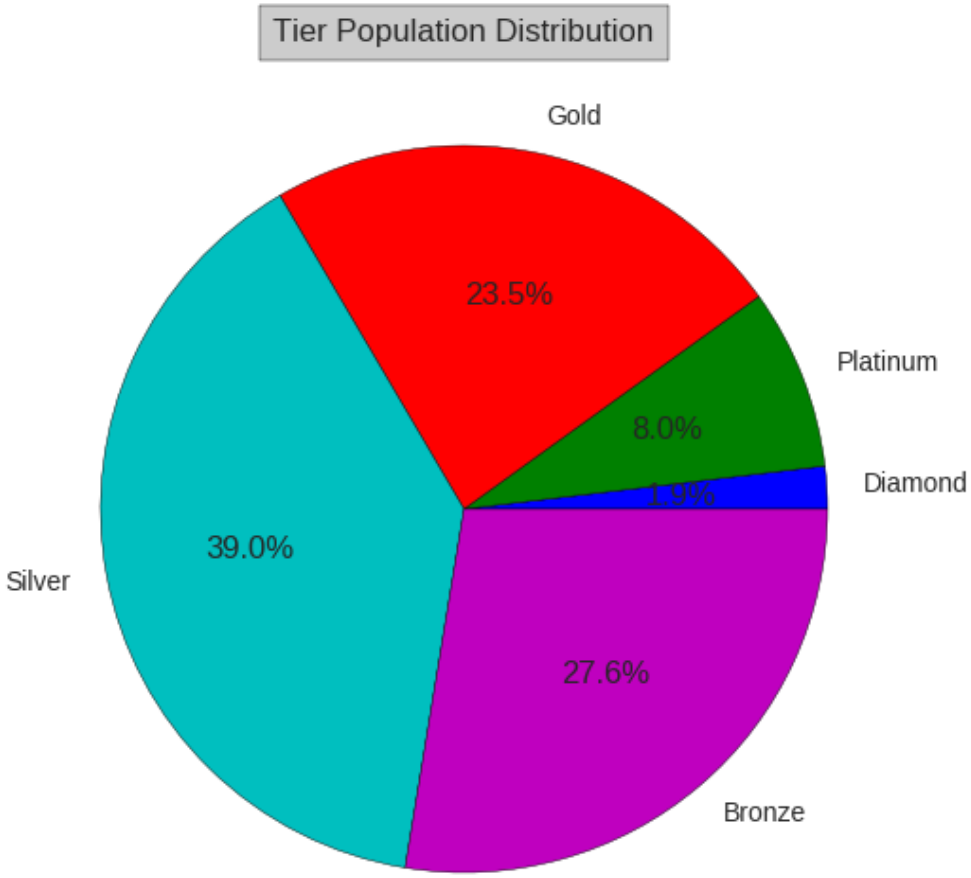

Figure 1: Tier population distribution.

In 1, it can be seen that there are some very high correlations, which often haven logical explanations. For example, the first correlation, between neutralMinionsKilled[9] and neutralMinionsKilledYourJungle[10]. Such a high correlation makes sense because neutral minions tend to be killed in your jungle as it's often too dangerous to kill minions in the enemy's jungle. Another easy to explain correlation is physicalDamageTaken versus totalDamageTaken, as often much of the damage taken is physical.

Feature A	Feature B	Correlation Coefficient
neutralMinionsKilled (mean)	neutralMinionsKilledYourJungle (mean)	0.99
goldEarned (mean)	goldSpent (mean)	0.97
tripleKills (mean)	tripleKills (stdev)	0.93
physicalDamageTaken (mean)	totalDamageTaken (mean)	0.92
physicalDamageTaken (stdev)	totalDamageTaken (stdev)	0.85

Table 1: Select sample of features with high correlations.

[9]Minions are non-player characters that either fight for either one of the teams, or for nobody. When they fight for nobody, they are "neutral".

[10]The "jungle" is a region of the map where neutral minions reside. "Your" jungle is the half of the jungle that resides on the same side as your nexus.

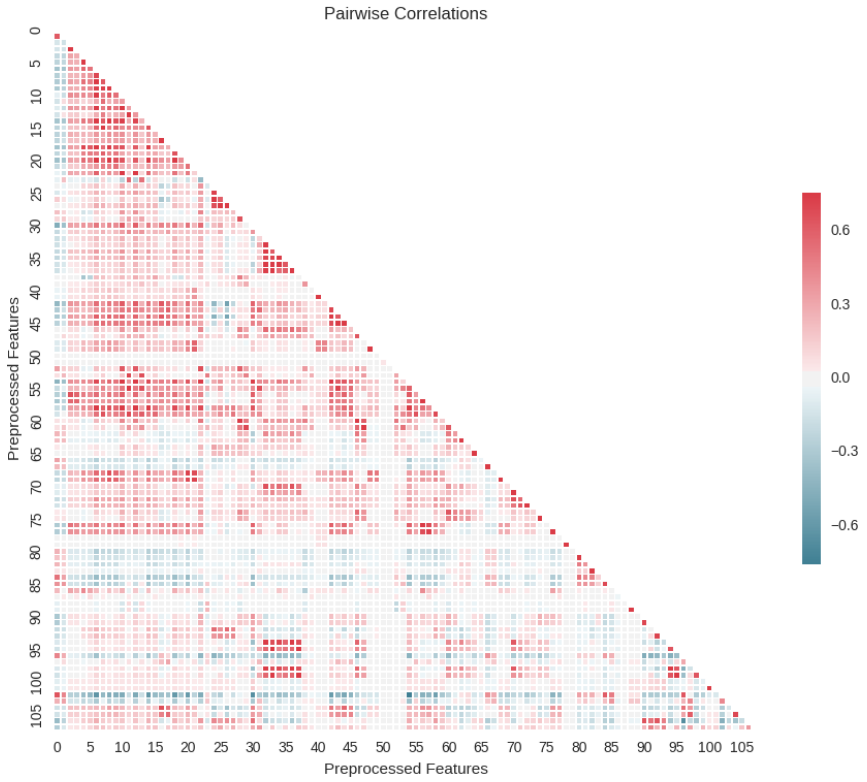

Figure 2: Correlation map.

2.3 Algorithms and Techniques

The only significant algorithms used were classification-based algorithms. No advanced preprocessing like PCA, ICA, KMeans, etc were used. The four classification algorithms experimented with are:

Support Vector Machines Support Vector Machines (SVM)[5] are strong binary classifiers. Using kernel transforms of the data, they can embed data in a much more complex and potentially higher dimensional space to make separating classes easier. However, finding a good kernel function is not easy, leaving researchers with a few common kernel functions to use. This is contrary to classifiers such as neural networks, which learn their transformation functions.

Random Forest Random Forests (RF)[6] are ensemble classifiers that use collections of decision trees together to make a final classifier. This is another strong, standard benchmark for comparison, like SVMs. However, one way it is different that has potential to affect performance is that it handles multiple classes out of the box due to the flexibility of decision trees. This flexibility is also a downside, as it can easily overfit training data, especially if many trees are used or they're allowed to get very deep.

Gradient Boosted Trees Gradient boosted trees (GBT)[7] are another, tree-based ensemble method. The key difference between GBTs and RFs is how they are trained, with RFs training trees in isolation while GBTs are trained sequentially, improving the performance of the model based on existing trees. Because of the similar underlying structure, GBTs share the flexibility and strength of RFs. Sadly, they also suffer the same weakness as RFs, as they too tend to overfit. However, GBTs have more and different ways to regularize, making it possible to tune it to perform better than RFs.

Multilayer Perceptron Neural networks have taken the world by storm due to their phenomenal performance in a number of fields, including computer vision, speech recognition, and text recognition. One of simplest architectures is a Multilayer Perceptron (MLP)[8]. The strength of MLPs and neural networks overall is that they mix feature extraction with classification. For this project, that is especially good as relevant features (or feature combinations) are not well understood at all, making an algorithm to find the optimal derived feature set invaluable. One major problem with neural networks is that they often require more data than other algorithms to reach their full potential. It is hoped that a sufficient amount of data is collected to take full advantage of the strength of the MLP.

2.4 Benchmark

The benchmark accuracy is 39%, the highest single-class frequency. Such a simple benchmark is used because the primary goal of this research is not necessarily to get perfect accuracy, but to prove that *anything* can be learned from the data provided.

3 Methodology

3.1 Data Preprocessing

To re-iterate, the data obtained from the API for each player consists of the player's personal and own team summary statistics for the player's last 5-10 games, where each game consists of 55 features that are either numerical or categorical.

The first thing that was done is eliminate features that would significantly raise the dimensionality of the feature set. The features excluded were championId and item0, item1...item6. These were all excluded because they are categorical variables with over 100 categories each, leading to sparse, high dimensional data. Other categorical data (playerRole and playerPosition) were not removed, as they had under half a dozen categories each, not significantly increasing the dimensionality.

In future work, ignored categorical features should be taken into consideration, perhaps with preprocessing to reduce dimensionality. For example, champions can be accurately represented by a few variables (ranged/melee type, magic/physical type, etc), keeping the dimensionality low. Items can also likely be grouped together by whether they are consumable, by if they have an effect, and by their price. Doing such preprocessing is certainly possible, but requires work beyond the scope of this project.

With the reduced feature set, it was necessary to address the different numbers of games per player, as most classifiers can't handle variable sized feature sets out of the box. The mean and standard deviation for each players set of games was taken to generate a single feature set. Just before this, categorical variables were one-hot encoded.

Finally, the data was split into a training and test set, with 20% of the data in the test set. Both sets share the same tier distribution. Because some models perform better with normalized features, both training and tests sets were scaled to have zero mean and unit variance based on the mean and variance of the training set.

No attempt was made to detect or eliminate outliers. Outliers in this data set would be games in which the player was physically unable to play for extended periods of time due to server error. This affects only a small percentage of games, and because we are taking an average, it has a further muted impact on the ultimate data point. Future work should investigate further data quality issues, but will likely only have a negligible effect.

3.2 Implementation

Each of the four (SVM, Random Forest, Gradient Boosted Trees, Multilayer Perceptron) classification algorithms were fed training data, utilizing grid search with 3-fold cross validation to find the optimal model. Parameters were selected to minimize validation error. Parameters that were not grid-searched stayed as library defaults.

3.2.1 SVM

For multiclass classification, SVMs were trained using a one-vs-one training scheme. To attempt to aid in dealing with the significantly imbalanced classes (tiers), we experimented with weighting the

classes proportional to the class distributions. C, γ, and different kernels were also grid searched. The values tested were:

C: 0.1, 1, 10

γ: $1/n_{features}$(1/108), 0,1, 0,001

Kernel: Radial Basis Function (RBF), Sigmoid

Class Weight: Uniform, Proportional to distribution

3.2.2 Gradient Boosted Trees

Gradient Boosted Trees also were tried with weighted classes, and with the following parameters grid-searched:

Maximum Tree Depth: 3, 6, 9

Number of Estimators: 50, 100, 200, 400

Class Weight: Uniform, Proportional to distribution

3.2.3 Random Forest

The features chosen to grid search for Random Forests were deliberately chosen to be similar to the gradient boosted trees parameters. They are:

Maximum Tree Depth: No maximum, 2, 4, 8

Number of Estimators: 10, 20, 40, 80, 160, 320[11]

Maximum Number of Features Used : None, $\sqrt{n_{features}}$

3.2.4 Multilayer Perceptron

The most complicated model to design and train was the MLP[8], as the architecture had to be designed from scratch. The model consisted of linear layers, followed by an activation layer, followed by a dropout layer. Hidden layers each contain half the number of outputs as the previous layer. The models were allowed to train until there were two consecutive epochs of decreasing performance. A batch size of 256 was used for training, RMSProp was used for optimization, and 20% of the test data was used for validation each epoch. The grid searched parameters are:

n_{hidden}: 1, 2, 3, 4

Number of Outputs for First Hidden Layer: 256. 512, 1024

Dropout Probability: 0, 25, 50, 75

3.3 Refinement

3.3.1 Model Refinement

As stated above, all models underwent basic hyperparameter search. Once this first pass of training and parameter search was done, only the best of the remaining models was further refined, exploring a superset of the parameters tested. This two-stage process was done simply to save time. It is wholly possible the true best model was not selected due to this selection procedure.

The best performing model increased in accuracy from 55.7% to 56.1%. Please see 4 for more details.

[11]The default number of estimators for RF vs XGB are very different (10 vs 100), which is why the same set of numbers was not used for both.

3.3.2 Feature Set Refinement

Because the features are so redundant, an attempt was made to reduce the number of features used and explore how it affects the best model. Top features based on the ANOVA F-Test [9] [10] were used to reduce the feature set. In the best case, reducing features would enable the model to perform better, but it most likely would marginally reduce performance.

The best performing model, using 25% of the original features, had its accuracy decreased from 55.7% to 53.9%. This model still out-performs all other models while only using a fraction of the original features. See 3 for a visualization of the performance change.

4 Results

4.1 Model Evaluation and Validation

4.1.1 Optimal Model Selection

After training and tuning the four types of models, ultimately the SVM classifier had the best performance, with a training test score of 55.7% of the samples classified accurately. The other models are not far behind, all performing in the low-mid 50% range, all beating the basic benchmark of 30.0%. As mentioned earlier, to further attempt to refine the winning model, hyperparameters

Model	Training Accuracy(%)	Testing Accuracy(%)
SVM	60.9(64.9)	55.7(56.1)
Random Forest	100.00	52.2
Gradient Boosted Trees	68.4	52.2
MLP	63.0	52.8

Table 2: Model performance results. Numbers in parenthesis are results using further tuned parameters.

were further fine tuned. Below is the final set of parameters grid searched. The final performance of the SVM has a testing accuracy of **56.1%**.

C: 0.1, 1, 10, 50, **100**, 150, 200

γ**:** 0.0001, **0.0005**, 0.001, $1/n_{features}$(1/108), 0.1

Kernel: RBF, Sigmoid

Class Weight: Uniform, Proportional to distribution

4.1.2 Model Analysis and Comparison

Each of the models were roughly the same in terms of training performance, except for the Random Forest. This is due to lack of regularization and the ability of a decision tree to, given sufficient depth, overfit to any training set. Similarly, the Gradient Boosted Trees is the second most prone to overfitting due to the same underlying tree structure.

In terms of training data, the runner up is the MLP. Due to the vast number of ways to architect a neural network solution, it is very possible that an MLP could outperfom the SVM, especially with a larger data. One interesting avenue of research would be to use a convolution or recurrent network to utilize all games individually, without first taking the mean and standard deviation per player.

4.2 Justification

To further understand the SVM, we show a table of the precision and recall for each tier3. Unsurprisingly, the model is the best at predicting the two most common tiers: Bronze and Silver-accurately predicting approximately two thirds of the players in these classes. On the other hand, it is quite bad at predicting the two least common tiers: Platinum and Diamond-correctly classifying less than 10% of the examples in these classes. With the simple benchmark accuracy of 39.0%, the best performing model easily demonstrates that something is learned, although there is obviously much room for improvement.

	Precision	Recall	Count
Bronze	0.73	0.66	1011
Silver	0.54	0.69	1461
Gold	0.44	0.44	882
Platinum	0.38	0.07	302
Diamond	0.25	0.01	69

Table 3: Precision and Recall data for optimal SVM model.

5 Conclusion

5.1 Free-Form Visualization

One piece of information that would be helpful to know is what features characterize a good or bad player. What do you need to know about a player's match history to know how "good" he or she is? A simple way to explore this is to see how many features are needed before getting a model's optimal performance.

As mentioned earlier, we reduce the number of features using ANOVA F-Scores to order the usefulness of features. As can be see in3, surprisingly few features are needed, as most of the them are fairly redundant. Using only 11 features (10%), we bet the benchmark by over 7%. Using just 25 features (25%), we come within nearly 2% accuracy of the optimal classifier, using all 108 features.

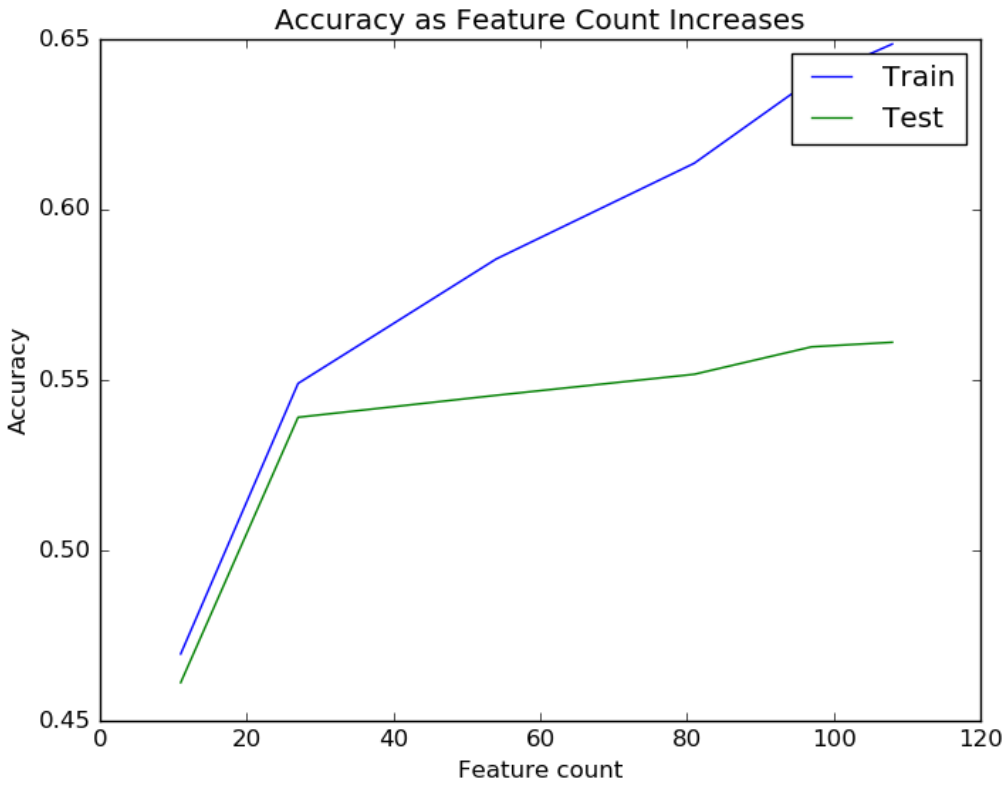

Figure 3: Number of features used versus training performance using best model (SVM) architecture.

It is interesting to observe the train-test divergence as the number of features is increased. While the testing performance gets marginally better, the training performance shoots past it quickly, showing the model's capacity to learn more, but the data lacking much more usable information as features are added.

5.2 Reflection and Improvement

The most difficult part of this research was not the model selection or model tuning, but on data collection. Because a pre-digested collection of player data was not freely available online, it was necessary to first collect it.

The process of reading documentation, determining what data would be both easy and quick to obtain, yet useful for the problem at hand, and finally pulling a sufficient quantity of data for analysis took up approximately 60% of the time for this research. The initial plan was to pull match statistics not for a player, but for the entire team. It was also planned to pull the same number of ranked games per player. However, doing both of these things would explode the time it would take to collect data.

Once the data was selected, filtering out irrelevant features and preprocesing required further analysis based on knowledge of both machine learning and the problem domain. In hindsight, it might have been useful to attempt to not exclude any features automatically, and let a feature selection algorithm exclude them programatically.

Finally, training, tuning, and choosing the best model was a rather rote task of plugging neat, clean data into packaged algorithms. It was rather surprising that the simplest model performed the best, beating kaggle favorites Gradient Boosted Trees and Neural Networks. As it is rare that, given enough data, a Neural Network cannot outperform an SVM, it would be interesting to see how the two would compare if more data-both data points and features-were collected.

While this research shows that it is possible to learn something about players from their recent games, the model is not nearly performant enough to release for public use. If the recall sores of the three highest tiers could be raised to the level of the lowest two, I would argue the model has some use. Until then, more research must be done.

References

[1] *Get recent games by summoner ID.* [Online]. Available: https://developer.riotgames.com/api/methods#!/1078/3718

[2] *Rank distribution.* [Online]. Available: http://www.leagueofgraphs.com/rankings/rank-distribution (Accessed 2016-09-15).

[3] A. B. Graf, A. J. Smola, and S. Borer, "Classification in a normalized feature space using support vector machines," *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 597–605, 2003.

[4] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[5] *sklearn.svm.svc.* [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[6] *sklearn.ensemble.RandomForestClassifier.* [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[7] *Scalable and Flexible Gradient Boosting.* [Online]. Available: https://xgboost.readthedocs.io/en/latest/

[8] *Keras: Deep Learning library for Theano and TensorFlow.* [Online]. Available: https://keras.io/

[9] *sklearn.feature_selection.f_classif.* [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classif

[10] *sklearn.feature_selection.SelectPercentile.* [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html

A Features

A.1 Feature list

Below are the raw features used for classification. For each feature, for each player, the man and standard deviation were calculated. There are 54 raw features, making for a total of 108 features after mean and standard deviation are calculated. Items that contain an underscore (_) character are the one-hot encoding of categorical features.

1. assists
2. barracksKilled
3. bountyLevel
4. championsKilled
5. doubleKills
6. goldEarned
7. goldSpent
8. killingSprees
9. largestCriticalStrike
10. largestKillingSpree
11. largestMultiKill
12. level
13. magicDamageDealtPlayer
14. magicDamageDealtToChampions
15. magicDamageTaken
16. minionsKilled
17. neutralMinionsKilled
18. neutralMinionsKilledEnemyJungle
19. neutralMinionsKilledYourJungle
20. numDeaths
21. pentaKills
22. physicalDamageDealtPlayer
23. physicalDamageDealtToChampions
24. physicalDamageTaken
25. quadraKills
26. team
27. timePlayed
28. totalDamageDealt
29. totalDamageDealtToBuildings
30. totalDamageDealtToChampions
31. totalDamageTaken
32. totalHeal
33. totalTimeCrowdControlDealt
34. totalUnitsHealed
35. tripleKills
36. trueDamageDealtPlayer
37. trueDamageDealtToChampions
38. trueDamageTaken
39. turretsKilled
40. unrealKills
41. visionWardsBought
42. wardKilled
43. wardPlaced
44. win
45. playerPosition_0.0
46. playerPosition_1.0
47. playerPosition_2.0
48. playerPosition_3.0
49. playerPosition_4.0
50. playerRole_0.0
51. playerRole_1.0
52. playerRole_2.0
53. playerRole_3.0
54. playerRole_4.0

A.2 Statistics

	mean	std	min	50%	max

assists	mean	9.30	3.06	1.17	8.80	26.80
	std	5.26	1.92	0.55	5.00	20.11
barracksKilled	mean	0.20	0.19	0.00	0.17	2.00
	std	0.39	0.29	0.00	0.41	2.19
bountyLevel	mean	0.84	0.66	0.00	0.71	4.60
	std	1.32	0.73	0.00	1.46	2.74
championsKilled	mean	6.59	2.77	0.12	6.50	27.40
	std	4.06	1.65	0.35	3.93	13.85
doubleKills	mean	0.65	0.50	0.00	0.57	4.75
	std	0.83	0.49	0.00	0.78	3.29
goldEarned	mean	12,440.17	1,711.31	6,287.17	12,414.10	20,392.50
	std	3,521.63	1,058.13	456.39	3,446.42	9,370.25
goldSpent	mean	11,345.84	1,662.89	5,341.67	11,322.95	23,967.60
	std	3,385.73	1,120.98	531.39	3,270.67	14,145.20
killingSprees	mean	1.53	0.71	0.00	1.50	6.00
	std	1.16	0.44	0.00	1.14	3.56
largestCriticalStrike	mean	233.31	242.69	0.00	163.37	1,774.22
	std	264.64	209.28	0.00	264.71	1,234.25
largestKillingSpree	mean	2.88	1.32	0.00	2.86	11.17
	std	2.16	0.91	0.00	2.01	8.74
largestMultiKill	mean	1.43	0.37	0.10	1.40	3.50
	std	0.66	0.25	0.00	0.64	2.07
level	mean	14.93	1.04	9.70	15.00	18.00
	std	2.22	0.63	0.00	2.18	8.28
magicDamageDealtPlayer	mean	45,411.14	31,528.11	40.25	38,251.80	227,405.60
	std	39,624.50	25,126.64	113.84	36,983.84	200,083.07
magicDamageDealtToChampions	mean	9,042.18	5,905.03	7.62	7,703.74	43,869.60
	std	7,625.56	4,853.29	21.57	6,939.49	36,366.87
magicDamageTaken	mean	9,007.55	2,324.64	1,779.67	8,791.70	24,156.33
	std	5,018.66	2,054.70	599.05	4,662.09	22,959.12
minionsKilled	mean	124.22	53.29	5.33	127.63	317.10
	std	58.08	24.17	1.21	57.57	187.54
neutralMinionsKilled	mean	19.37	19.63	0.00	12.30	135.00
	std	17.63	14.10	0.00	14.01	76.56
neutralMinionsKilledEnemyJungle	mean	3.74	3.50	0.00	2.78	31.67
	std	4.14	3.15	0.00	3.45	27.76
neutralMinionsKilledYourJungle	mean	15.64	16.84	0.00	9.30	108.80
	std	15.01	12.43	0.00	12.04	60.95
numDeaths	mean	6.60	1.77	1.14	6.50	15.33
	std	2.99	0.88	0.00	2.92	9.94
pentaKills	mean	0.00	0.02	0.00	0.00	0.25
	std	0.01	0.05	0.00	0.00	0.63
physicalDamageDealtPlayer	mean	74,489.90	45,463.04	2,542.20	72,412.40	356,980.30
	std	53,275.71	30,498.31	654.21	53,770.33	249,984.86
physicalDamageDealtToChampions	mean	9,825.85	6,426.17	199.40	9,125.75	42,429.80
	std	7,714.77	4,692.52	76.02	7,587.41	35,271.86
physicalDamageTaken	mean	15,493.33	4,432.99	4,344.10	14,992.27	38,405.17
	std	6,900.27	2,871.61	797.96	6,481.85	24,196.92
quadraKills	mean	0.01	0.04	0.00	0.00	0.67
	std	0.03	0.11	0.00	0.00	1.06
team	mean	149.64	17.48	100.00	150.00	200.00
	std	49.54	7.08	0.00	51.64	54.77
timePlayed	mean	2,027.06	177.71	1,363.80	2,020.80	2,889.71
	std	444.27	130.37	51.61	434.87	1,351.76
totalDamageDealt	mean	125,110.98	44,730.91	8,914.40	128,482.31	383,820.40
	std	59,701.88	26,479.10	1,794.10	57,128.22	254,230.53
totalDamageDealtToBuildings	mean	4,400.93	2,440.30	0.00	4,012.35	24,621.00
	std	4,055.17	2,087.35	0.00	3,806.94	18,656.43

totalDamageDealtToChampions	mean	19,816.20	6,351.48	2,799.30	19,644.83	51,447.30
	std	10,111.58	4,374.16	635.86	9,584.88	33,934.58
totalDamageTaken	mean	25,471.20	6,193.53	9,089.67	24,924.85	60,534.50
	std	10,554.48	4,321.31	978.43	9,831.89	36,517.96
totalHeal	mean	5,191.37	3,182.63	287.90	4,384.27	43,382.50
	std	4,005.00	3,105.97	209.34	3,157.67	34,921.24
totalTimeCrowdControlDealt	mean	514.50	418.14	5.80	418.53	8,761.20
	std	510.36	563.03	8.49	331.74	8,174.91
totalUnitsHealed	mean	2.38	2.55	0.60	1.80	68.00
	std	1.66	3.00	0.00	1.08	35.05
tripleKills	mean	0.09	0.14	0.00	0.00	1.50
	std	0.20	0.25	0.00	0.00	1.75
trueDamageDealtPlayer	mean	5,209.19	4,343.77	0.00	4,120.45	46,193.70
	std	5,223.02	4,024.56	0.00	4,603.65	38,979.58
trueDamageDealtToChampions	mean	947.56	900.41	0.00	725.40	11,073.20
	std	1,079.78	1,021.46	0.00	748.90	12,685.88
trueDamageTaken	mean	969.64	495.65	63.00	868.87	6,521.50
	std	932.41	613.28	50.52	783.95	6,679.90
turretsKilled	mean	0.99	0.60	0.00	0.90	4.30
	std	1.08	0.48	0.00	1.03	3.58
unrealKills	mean	0.00	0.00	0.00	0.00	0.14
	std	0.00	0.00	0.00	0.00	0.38
visionWardsBought	mean	0.88	1.02	0.00	0.60	11.60
	std	0.68	0.60	0.00	0.63	7.02
wardKilled	mean	2.08	1.56	0.00	1.67	16.43
	std	2.25	2.26	0.00	1.64	27.51
wardPlaced	mean	12.14	5.08	0.20	11.30	43.80
	std	4.96	2.78	0.00	4.15	44.13
win	mean	0.50	0.18	0.00	0.50	1.00
	std	0.49	0.07	0.00	0.52	0.55
playerPosition_0.0	mean	0.01	0.03	0.00	0.00	0.33
	std	0.02	0.08	0.00	0.00	0.52
playerPosition_1.0	mean	0.19	0.25	0.00	0.10	1.00
	std	0.24	0.23	0.00	0.32	0.55
playerPosition_2.0	mean	0.20	0.25	0.00	0.11	1.00
	std	0.25	0.22	0.00	0.32	0.55
playerPosition_3.0	mean	0.21	0.27	0.00	0.10	1.00
	std	0.24	0.23	0.00	0.32	0.55
playerPosition_4.0	mean	0.39	0.35	0.00	0.30	1.00
	std	0.29	0.22	0.00	0.38	0.55
playerRole_0.0	mean	0.21	0.27	0.00	0.10	1.00
	std	0.24	0.23	0.00	0.32	0.55
playerRole_1.0	mean	0.02	0.06	0.00	0.00	0.71
	std	0.06	0.14	0.00	0.00	0.55
playerRole_2.0	mean	0.20	0.30	0.00	0.00	1.00
	std	0.17	0.21	0.00	0.00	0.55
playerRole_3.0	mean	0.20	0.27	0.00	0.10	1.00
	std	0.22	0.23	0.00	0.00	0.55
playerRole_4.0	mean	0.38	0.32	0.00	0.30	1.00
	std	0.33	0.21	0.00	0.42	0.55