STATISTICAL LEARNING FOR ESPORTS MATCH PREDICTION

A Thesis

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Mathematics

By

Kevin Bailey

2020

SIGNATURE PAGE

THESIS:	STATISTICAL LEARNING FOR ESPORTS MATCH PREDICTION
AUTHOR:	Kevin Bailey
DATE SUBMITTED:	Spring 2020
	Department of Mathematics and Statistics

Dr. Jimmy Risk
Thesis Committee Chair
Mathematics & Statistics

Dr. Alan Krinik
Mathematics & Statistics

Dr. Adam King
Mathematics & Statistics

ACKNOWLEDGMENTS

This thesis is dedicated to my family, friends, and professors who have supported me throughout my education. I would not have gotten through graduate school had it not been for many of you. A special thanks to that of my girlfriend, Rita, and her family for allowing me an affordable place to stay during this stressful time. And another special thanks to Dr. Jimmy Risk for dealing with me in his office hours many times each week every semester throughout my graduate school career, as well as agreeing to be my thesis advisor and dealing with my pestering throughout it.

ABSTRACT

Traditional sports have been a hot topic for many statistical and machine learning models. Not only for prediction, but these models can help to make informed decisions on fantasy drafts as well. When only caring about the result of a match, many popular classifications such as logistic regression and support vector machines (SVMs) are used. However, one statistical learning concept that has been gaining traction, but is still not widely used, is that of Gaussian processes (GPs). This paper aims to analyze of many different models that are common in practice and give a quick rundown on the theory of them, as well as going into Gaussian processes to see how they will compare. Our application is to apply these models on a League of Legends data set of professional matches, and comparing them to see which one works best for this classification of winning or losing. Theoretically, Gaussian processes should be a great fit for this problem since local observations help the predictions more than distant ones, and the domain of some very important factors (gold and gold-difference) tend to be clustered around the average; in fact, it is nearly impossible to stray extremely far from that average.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Esports, short for electronic sports, is a form of competitive sports using video games. Commonly, this takes on the form of organized, multiplayer video game competitions, either individually or as a team. In the last decade, esports became a significant factor in the video game industry, leading to many developers designing games with esports in mind. The common video game genres associated with esports are first-person shooters (FPS), fighting, real-time strategy (RTS), and multiplayer online battle arenas (MOBA). With the ease of access to streaming platforms such as *YouTube*, *Twitch*, *Douyu*, and more, allowing viewers from all over the world to tune in and watch as well as stream themselves, esports has begun to rise in viewership to the point of being noticed by traditional sports networks such as ESPN. During the 2019 League of Legends (LoL) World Championship, records were broken when viewership reached a peak of 1.743 million on *Twitch* alone, beating out the previous record of 1.618 million which was held by Fortnite's "The End" event [1]. Reports say that overall, the LoL 2019 World Championship finals peaked at 44 million concurrent viewers between all platforms, being broadcast in

16 different languages [11]. Furthermore, the average salary for a player in the League of Legends Championship Series (LCS) (see Appendix A.1) was $105,000 in January 2017, and has recently tripled to $300,000 in April 2019 [5]. From a monetary standpoint, it is crucial to identify what factors can influence game outcomes. Maximizing spectator enjoyment is an additional factor, which can be fortified by providing real time match predictions based on current game statistics, which has been done in Defense of the Ancients 2 (DOTA2), a game that is extremely similar to that of LoL, in [16].

The match prediction problem, however, is not an easy one. A single game of LoL contains a multitude of real time variables, all of which that can decide the outcome of a game. Aside from just player/team performance, there are many in-game features that have a drastic impact and can turn the game into the other team's favor or amplify the current lead a team has. We provide a quick glance into some of the basic ideas and complexities involved in a possible game of LoL:

- The game consists of two teams, each team consisting of five players.
- Each of the ten players individually choose from a pool of 148 champions, all of which have individual strengths and weaknesses, and some perform better than other against certain champions. In addition, champions typically have different power thresholds at various points in the match. The champion choice is fixed throughout a match.
- Players gain *gold* and *experience* throughout the game which also grants power to their champion.
- Major events happen at varying points throughout the game whose outcome can impact a team's total gold and experience.

- All of this data is available continuously to a live spectator of the game.

Further details about LoL's game mechanics are provided in Appendix A. One could simply view this as a large dimensional data problem that needs data reduction, but any expert player or analyst will be able to tell you obvious implications that some of these variables have on how likely a team is to win at say, the ten or fifteen minute mark. Thus, in this high dimensional problem, interpretability and/or prior beliefs are a crucial factors.

There is a clear similarity to sports, which we first look to borrow ideas from. In particular, predicting events/outcomes from past game results in sports has been a popular area of research for a long time, with many different methods and approaches to the area. In football, Min [3] uses Bayesian inference and rule-based reasoning along with in-game time-series to provide live win percentages. Many data mining techniques have also been used for sports prediction, as displayed in a study by Cao [7] for basketball outcome prediction; in this study, a model was constructed using a variety of machine learning algorithms such as a simple logistic classifier, support vector machines (SVM), naive Bayes and artifical neural networks (ANN). Interestingly enough, the Simple Logistic Classifier resulted in the highest accuracy. DOTA2, mentioned above, is similar to LoL in almost every component, and is its direct competitor, leading to many parallels being drawn between the two games. In the DOTA2 literature, logistic regression was trained using just champion selection as well as a different model using champion, player, and win-rate. Neural networks have been done on the same set and random forests were trained on champion selection along with their win-rate, all in [16]. The metric used in [16] was cross-validation accuracy, which we will be using here as well.

Another popular machine learning algorithm involves Gaussian processes (GPs), which have not been seen much in the literature for esports prediction thus far. Typically used in regression and originating in geostatistics [2], GPs as a regression tool have performed quite well. They can also be used in classification by utilizing them as the prior in the probability for a categorical response.

This thesis aims to build upon the previous work done in DOTA2 and general sports. Specifically, we will explore LoL match prediction using different types of models and in-game information. The main goal is to produce an accurate predictive model, but it is also important to recognize what predictors have a large impact on the outcome of a match. In particular, we seek to produce match predictions based on data available at the fifteen minute mark. This is done in DOTA2 as discussed in [16], but not yet done in LoL. We believe this is valuable to a LoL game broadcast and can easily and accurately be done with the available data. The rest of the thesis outlines the task at hand. In Chapter 2 we discuss the general framework of probabilistic classification, and provide assessment metrics for general classification models. Next, Chapter 3 discusses the theoretical details behind the models we use in this analysis. Next, Chapter 4 introduces the data set to be analyzed and investigates basic relationships between the variables. Finally, Chapter 5 discusses the model fitting, diagnostics, and prediction results. We conclude in Chapter 6.

Chapter 2

Classification Problem and Assessment Metrics

2.1 Classification Problem

Let $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$ be a *design*, where $x_i \in \mathbb{R}^p$ and $y_i \in \{0, 1\}$ is a binary response for a given x_i. The purpose of classification is to produce a binary prediction y based on a location x, based on the observed design $\mathcal{D}$. The means to do this is usually to model directly a probability

$$p(x) \equiv p(y = 1|\mathcal{D}), \tag{2.1}$$

i.e. fit a model that produces the probability of obtaining $y = 1$ given a data set $\mathcal{D}$. The resulting classification model is called a *classifier*. To perform the actual classification prediction $\hat{y}$ of y, the methods typically consist of fitting the model $p(x)$ in Equation (2.1) and making class assignments for y (i.e. is $y = 1$ or $y = 0$) based off whether $p(x)$ surpasses a certain threshold, η, to produce the *class estimate* $\hat{y}$. For example, if $\eta = 0.5$, $p(x) > 0.5$, corresponds to the prediction

$\hat{y} = 1$ since it is saying it is *more likely* to obtain $\hat{y} = 1$ than $\hat{y} = 0$, and conversely, $p(x) < 0.5$ produces $\hat{y} = 0$. Using $\eta = 0.5$ makes sense for most binary response cases because there are only two classes, so whichever one has a probability above 0.5 has a higher chance of occurring than the opposite class.

2.2 Classification Metrics

Suppose that we want to assess a classifier $p(x)$ on a general data set $\mathcal{D}' = ((x'_1, y'_1), \ldots, (x'_m, y'_m))$, which may or may not be equal to $\mathcal{D}$. Here, m is the number of points for $p(x)$ to be evaluated on. In the case where $\mathcal{D}' = \mathcal{D}$, we are assessing the *in sample* classification ability of $p(x)$. Whenever $x' \notin \mathcal{D}$, i.e. x' is not in the design, $p(x')$ is an *out of sample* prediction, so y'_i is assumed unknown. We discuss common metrics used to quantify the performance of a classifier.

2.2.1 Basic Terms

A *false positive* (FP) is when we predict a positive result, but the result is actually a negative,

$$\text{FP}_i = \mathbb{1}_{\{y'_i \neq \hat{y}'_i,\, y'_i = 0\}}$$

where $\mathbb{1}_{\{y'_i = \hat{y}'_i\}}$ is the indicator function of the event $\{y'_i = \hat{y}'_i\}$:

$$\mathbb{1}_{\{y'_i = \hat{y}_i\}} = \begin{cases} 0 & \text{if} \quad y'_i \neq \hat{y}'_i \\ 1 & \text{if} \quad y'_i = \hat{y}'_i \end{cases} \tag{2.2}$$

A *false negative* (FN) is when we predict a negative result, $\hat{y} = 0$, but the result is actually a positive,

$$\text{FN}_i = \mathbb{1}_{\{y'_i \neq \hat{y}'_i,\, y'_i = 1\}}.$$

Both of these are incorrect predictions, which a good classifier should seek to minimize. Similarly, a good classifier should aim to maximize the number of *true negatives* (TN), i.e. when $\hat{y}_i = y_i = 0$, and *true positives* (TP), $\hat{y}_i = y_i = 1$, both of which would be correct predictions.

To produce a metric for a classifiers performance over an entire dataset, we define the following:

- The *False Positive Rate* (FPR) is the ratio of false positives to the total number of negatives, given by

$$\text{FPR} = \frac{\sum_{i=1}^{m} \text{FP}_i}{\sum_{i=1}^{m} [\text{FP}_i + \text{TN}_i]} = \frac{\sum_{i=1}^{m} \mathbb{1}_{\{\hat{y}'_i=1, y'_i=0\}}}{\sum_{i=1}^{m} \left[\mathbb{1}_{\{\hat{y}'_i=1, y'_i=0\}} + \mathbb{1}_{\{\hat{y}'_i=y'_i=0\}}\right]}. \tag{2.3}$$

- The *specificity* is $1 - \text{FPR}$, which measures the proportion of actual negatives that are correctly identified as such.

- The True Positive Rate (TPR), also called the *sensitivity*, is the ratio of true positives to the total number of positives, given by

$$\text{TPR} = \frac{\sum_{i=1}^{m} \text{TP}_i}{\sum_{i=1}^{m} [\text{TP}_i + \text{FN}_i]} = \frac{\sum_{i=1}^{m} \mathbb{1}_{\{\hat{y}'_i=y'_i=1\}}}{\sum_{i=1}^{m} \left[\mathbb{1}_{\{\hat{y}'_i=y'_i=1\}} + \mathbb{1}_{\{\hat{y}'_i=0, y'_i=1\}}\right]}. \tag{2.4}$$

In regards to these metrics, a classifier should aim to *minimize* the FPR (equivalent to maximizing the specificity) and *maximize* the TPR.

2.2.2 Accuracy

Accuracy is the average number of correct predictions,

$$\text{Acc} = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{y_i=\hat{y}_i\}}}{n}. \tag{2.5}$$

This results in the percentage of correct predictions from a model, or the number of true positives and true negatives out of the true and false positive/negatives; we want this number to be *high*. Equivalently, one could also report the misclassification error rate, defined as $1-\text{Acc}$; this number we would want to be *small*.

2.2.3 AUC and ROC Curve

The Receiver Operator Characteristic (ROC) curve is constructed by plotting the FPR on the horizontal axis and the TPR on the vertical axis. A desirable classifier aims to maximize the TPR while simultaneously minimizing the FPR. One can show that the TPR increases as the FPR also increases (specifically by tweaking $0 \leq \eta \leq 1$), so the hope is that the TPR increases *as rapidly as possible* as the FPR increases. Hence, the *area under the ROC curve* (AUC) should be maximized.

Ideally, we want our ROC curve to hug the top left of the graph, where the TPR is 1 and the FPR is 0, giving an AUC of 1, meaning there are all true positives and no false positives. Because of this, when evaluating a model's performance on the basis of AUC, the closer it is the 1, the better it is predicting.

An example of two ROC curves can be seen in Figure 2.1, showing that the better performing model in red has its TPR increasing very rapidly from the start, trying to hug the top left of the plot. The poor performing model in blue has its TPR go up almost at a one-to-one rate with the FPR, performing no better than a random chance classifier (the diagonal line) which assigns a score sampled from the uniform distribution between 0 and 1 to each instance, then classifying it based on η.

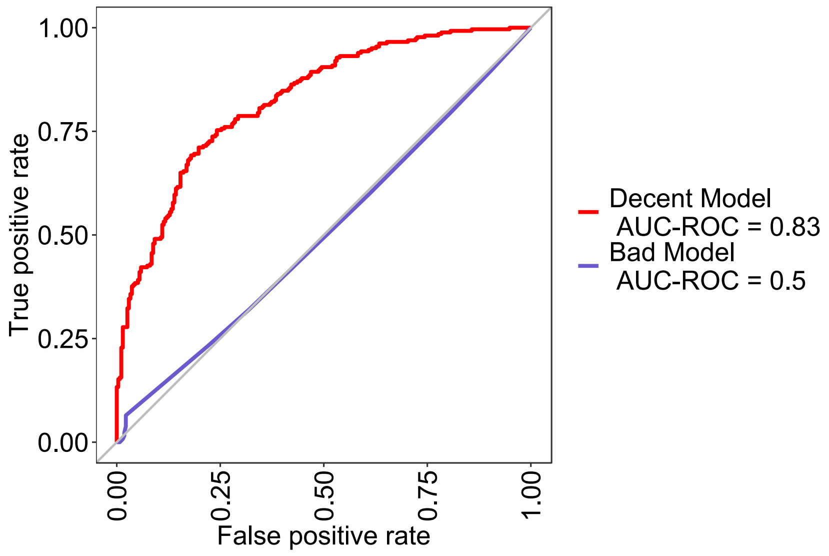


Figure 2.1: Example of two ROC curves, one model that performs fairly well drawn in red and one poorly performing model shown in blue. The poorly performing model has an AUC similar to that of a random chance classifier.

2.3 Out of Sample Prediction

In general, a classifier that performs well in sample may not perform well out of sample. In practice, since out of sample data is not readily available, one needs to use the a full data set to both fit the model and also assess out of sample performance.

We reiterate the notation defined above, with $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$ being the *design* or *training set*, and $\mathcal{D}' = ((x_1', y_1'), \ldots, (x_m', y_m'))$ be the set on which a classifier $p(x)$ is evaluated. It is possible that $\mathcal{D}$ and $\mathcal{D}'$ overlap, or are identical. However, as the aim of this paper is to produce a model that is optimal in predicting out of sample, from henceforth we denote $\mathcal{D}'$ as the *test set*, such that $\mathcal{D} \cap \mathcal{D}' = \emptyset$, that is, there is no overlap. In this regard, the names of these sets are accurate:

the model is fitted (trained) on $\mathcal{D}$, and its out of sample performance is tested on $\mathcal{D}'$. We do note that the error metrics above can still be assessed on $\mathcal{D}$, however, this can result in *overfitting* since the predictions $\hat{y}_i$ are themselves correlated with the training observations y_i. In fact, if y_i' is a hypothetical out of sample response associated with x_i, then one can generally show that

$$\mathbb{E}\left[\sum_{i=1}^{n}(y_i' - \hat{y}_i)^2\right] = \mathbb{E}\left[\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right] + \frac{2}{n}\sum_{i=1}^{n}\text{Cov}(\hat{y}_i, y_i), \tag{2.6}$$

from T. Hastie [14].

2.3.1 Validation Set

Methods to evaluate a classifier out of sample, or at least estimate its out of sample prediction error, typically are done by artificially introducing a test set from a given (full) data set. Denote the full data set by $\tilde{\mathcal{D}} = ((\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_N, \tilde{y}_N))$; this would be the full data set obtained from a source online or from an experiment. The *validation set method* produces $\mathcal{D}$ and $\mathcal{D}'$ by splitting $\tilde{\mathcal{D}}$ into two non-overlapping parts at random. The obvious benefit to this procedure is that we have artifically created "out of sample data", i.e. $\mathcal{D}'$, the test set. However, if the original data set had $|\tilde{\mathcal{D}}| = N$ observations, and produced $|\mathcal{D}| = n$, $|\mathcal{D}'| = m$, i.e. $N = n + m$, then there is a tradeoff in the split: higher n improves model fit, while higher m improves reliability of out of sample performance, but we are faced with the constraint that $n + m = N$ and N is the amount of data available to us. There is no hard rule for how this splitting should be done, but most commonly used are $n = m = 0.5N$ (50% / 50% split), $n = \frac{2}{3}N$ and $m = \frac{1}{3}N$ ($\frac{2}{3}$ / $\frac{1}{3}$ split) , or $n = \frac{3}{4}N$ and $m = \frac{1}{4}N$ (75% / 25% split).

2.3.2 K-Fold Cross-Validation

Cross-validation (CV) is a common resampling procedure that is used to estimate prediction error. The procedure is designed to repeatedly emulate the validation set method and average the results. Often, this procedure is done after splitting $\tilde{\mathcal{D}}$ into $\mathcal{D}$ and $\mathcal{D}'$, which we assume below, but in general there could be no initial data splitting and it could be done on $\mathcal{D} = \tilde{\mathcal{D}}$.

To provide the specifics, K-fold cross-validation randomly splits $\mathcal{D}$ into K non-overlapping parts $\mathcal{D}_k, k = 1, \ldots, K$, using all but one of those parts as a new "training set", and evaluating $\mathcal{D}_k$ as the "test set", for whatever performance metric(s) desired. This procedure is repeated K times, until all of $\tilde{\mathcal{D}}$ is used as the test set. Finally, then the average performance of those K runs is taken to produce the K-fold cross-validation (K-fold CV) estimate. The process for K-fold cross-validation is as follows:

1. Let $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be the data set (without loss of generality, assume that the indices are randomized)

2. Divide the data into K folds:

$$\begin{aligned}
\mathcal{D}_1 &= \{(x_1, y_1), \ldots, (x_{n_1}, y_{n_1})\}, \\
\mathcal{D}_2 &= \{(x_{n_1+1}, y_{n_1+1}, \ldots, (x_{n_2}, y_{n_2})\} \\
&\vdots \\
\mathcal{D}_K &= \{(x_{n_{K-1}}, y_{n_{K-1}}), \ldots, (x_{n_{K-1}+1}, y_{n_{K-1}+1}), \ldots, (x_{n_K}, y_{n_K})\}
\end{aligned}$$

   where n_k, $k \in \{1, \ldots, K\}$ represents the index of the last observation in that kth fold.

3. For each $k = 1, \ldots, K$,

(a) Fit a classifier using $\mathcal{D}\backslash\mathcal{D}_k$ as the training set, i.e. to the full data set with the kth fold removed.

(b) Use $\mathcal{D}_k$ as the test set using the classifier that was fit from part (a), that is, compute accuracy with $\mathcal{D}_k$ as the test set:

$$\text{Acc}_k = \frac{1}{n_k - n_{k-1}} \sum_{i=n_{k-1}+1}^{n_k} \mathbb{1}_{\{y_i=\hat{y}_i\}} \tag{2.7}$$

where $\hat{y}_i$ was produced with the classifier fit on $\mathcal{D}\backslash\mathcal{D}_k$.

4. Average all of the accuracies obtained in the prior step to get the K-fold cross-validation accuracy:

$$\text{CV}_K = \frac{\sum_{k=1}^{K} \text{Acc}_k}{K} \tag{2.8}$$

Clearly, the larger K is, the longer the computations will take, but the more accurate they will be. Because of this, K is typically taken to be 5 or 10 as these still give similar results to that of leave-one-out cross-validation (LOOCV, which is K-fold CV with $K = n$), but the computations to fit the models are quick relative to that of LOOCV. LOOCV has the advantage of always giving the same result, since each fold will be the exact same on each run, whereas the folds for K-fold CV are randomly assigned observations, leading to CV results that have a component of randomness within them.

To put the computation time of K-fold CV into perspective, suppose model fitting takes 5 minutes and prediction takes 1 minute, then one fit and prediction takes a total of 6 minutes. With $10-$fold CV, we must fit the model 10 times and predict 10 times, giving us a total of 60 minutes to run the entire $10-$fold CV process. Now, imagine this with LOOCV and a data set of 500 rows, then you are fitting and predicting 500 times, yielding a runtime of $30,000$ minutes, which is almost 21 days of straight runtime. This is obviously not desirable, so K-fold CV helps cut this time down substantially for slightly less accurate results than LOOCV.

2.4 Subset Selection

Picking a subset of predictors that we believe to be related to the response is considered *subset selection*. The classifier is then fit using these predictors, and assessed as such.

2.4.1 Best Subset Selection

Best Subset Selection (BSS) is the process where we build a separate classifier for every possible combination of the p predictors. Once this process is finished, we look at all of the models to see which one fits the goal of performing the best. The process for BSS is as follows:

1. Fit a model with no predictors.
2. For $k = 1, 2, \ldots, p$:
    (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
    (b) Pick the best among these $\binom{p}{k}$ models, based on the metrics in mind (e.g. Acc, AUC, CV_K).
3. Step 1 and 2 together yield $p + 1$ models, specifically the best model with k predictors, so we pick the best model out of those.

This task is not trivial, and with a larger p, is sometimes impossible to compute, as we would be fitting 2^p different models and selecting from some subset among those. When possible, we would want to perform BSS and see if there is a clear winner through the process. There are alternatives to BSS with lower computation times, as we describe below.

2.4.2 Forward Stepwise Selection

Forward Stepwise Selection (FSS), explores a far more restricted set of models than BSS, but has a remarkably smaller run-time. FSS begins with a model containing no predictors, and adds one predictor at a time to the model until all predictors are in the model. At each step, the variable that gives the largest improvement to the fit is added to the model. This process can be detailed as:

1. Fit a model with no predictors, call it M_0.
2. For $k = 0, \dots, p-1$:
   (a) M_k uses k predictors; look at all possible $p-k$ models that utilize the predictors in M_k and add a single predictor (not contained in M_k
   (b) Compare metrics (Acc, AUC, CV_K) among M_k and all $p-k$ considered models in this step; the best performing model is the new M_{k+1}.
   (c) If $M_k = M_{k+1}$, there was no improvement, so the procedure ends and we pick M_k. Otherwise, continue.

While BSS fit a total of 2^p models, FSS fits $1 + p(p+1)/2$ models. For example, if $p = 28$, BSS would have to fit $268,435,456$ models, but for FSS we only have to fit 407 models.

Although FSS is a good alternative, it is not guaranteed to find the best possible model out of all 2^p models containing subsets of the p predictors. As a simple example, consider a case where the best one-predictor model involves x_1, but the best two predictor model involves x_2 and x_3, then FSS will miss this best two-predictor model because it will not be able to include both of x_2 and x_3 from it's one-predictor model selection.

Chapter 3

Classification Models

The classifier $p(x)$ in Equation (2.1) is typically built with the goal to minimize some loss criteria utilizing the design $\mathcal{D}$. The metrics in the last section offer various ways to assess and quantify this loss, but do not directly produce a classifier themselves. We explain commonly used classifiers and how they are built.

3.1 Logistic Regression

Logistic regression defines $p(x)$ implicitly in terms of a generalized log-odds or *logit* function,

$$\log\left(\frac{p(x)}{1-p(x)}\right)\Bigg(= \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p, = \beta_0 + \beta^T x, \tag{3.1}$$

where $x = (x_1, \ldots, x_p)$ and $\beta = (\beta_1, \ldots, \beta_p)$. We can then re-write (3.1) as

$$p(x) \equiv p(x;\beta) = \frac{e^{\beta_0 + \beta^T x}}{1 + e^{\beta_0 + \beta^T x}}. \tag{3.2}$$

Furthermore, logistic regression uses maximum likelihood to estimate the coefficients $\beta_0, \beta_1, \ldots, \beta_p$ by maximizing a likelihood function using the fact that each y_i

is Bernoulli distributed with probability of success $p(x_i)$:

$$p_{y_i|\mathcal{D}}(y_i) = p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}.$$

Hence, the log-likelihood can be written

$$\begin{aligned}\ell(\beta) &= \sum_{i=1}^{n} \big\{ y_i \log p(x_i;\beta) + (1 - y_i)\log(1 - p(x_i;\beta)) \\ &= \sum_{i=1}^{n} \big\{ y_i(\beta_0 + \beta^T x_i) - \log(1 + e^{\beta_0 + \beta^T x_i}) \ . \end{aligned} \tag{3.3}$$

Note that there is no closed form maximum for this, and the optimal β must be solved for numerically.

Now, increasing x_i by one unit with everything else fixed changes the log-odds by β_i, or equivalently it multiplies the odds by e^{β_i}. Since the relationship between $p(x)$ and x is not a straight line or hyperplane, β_i does not correspond to the change in $p(x)$ associated with a one-unit change in x_i. However, if β_i is positive then increasing x_i is associated with increasing $p(x)$, and similarly if β_i is negative then increasing x_i will be associated with decreasing $p(x)$.

3.2 LASSO for Classification

Least absolute shrinkage and selection operator (LASSO) for logistic regression uses the same formulation as logistic regression with the ℓ_1 penalty on the coefficients $\beta_1, \ldots, \beta_p$, which forces the coefficients $\beta_j, j = 1, \ldots, p$, to shrink toward 0.

To induce this ℓ_1 penalty on logistic regression, we maximize a penalized version of Equation (3.3):

$$\max_{\beta_0,\beta} \left\{ \sum_{i=1}^{n} \big[y_i(\beta_0 + \beta^T x_i) - \log(1 + e^{\beta_0 + \beta^T y_i}) \big] - \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{3.4}$$

where λ is a penalty term that is part of the ℓ_1 regularization that can lead to certain coefficients vanishing.

LASSO regularization works by adding a penalty term to the log likelihood function, so before the regularization it is equivalent to that of logistic regression as can be seen with the similarities between Equations (3.3) and (3.4). LASSO shrinks the coefficient estimates towards zero, however, it uses the ℓ_1 penalty on β which has the effect of forcing some of the coefficient estimates to be exactly equal to 0 when the tuning parameter λ is sufficiently large, which results in a form of model and feature selection. The tuning parameter, λ, is chosen using cross-validation and is critical to having a proper model [9].

3.3 Linear and Quadratic Discriminant Analysis

3.3.1 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is an alternative approach to classifying. Unlike logistic regression, the LDA classifier initially proposes a conditional distribution for the *predictors* $x_i|y_i, i = 1, \ldots, n$, then it places a prior on y_i, and finally uses Bayes theorem to obtain $p(x)$.

First, we introduce some notation: let $\pi_k, k = 0, 1$ be the prior distribution for each y_i, e.g. $\pi_1 = p(y_i = 1)$, and let $q_k(x) = p(x|y = k)$ be the conditional distribution of x whose corresponding y value is k. Then, Bayes' theorem states that our quantity of interest $p(x)$ can be obtained as such:

$$p(x) = p(y = 1|x) = \frac{\pi_1 q_1(x)}{\pi_0 q_0(x) + \pi_1 q_1(x)}. \tag{3.5}$$

Technically, our notation $p(x)$ also indicates a dependency on $\mathcal{D}$, so with slight abuse of notation, it is assuming that the parameters appearing in the right-hand side have all been fitted using $\mathcal{D}$, described below.

LDA assumes that $x = (x_1, x_2, \ldots, x_p)$ is drawn from a multivariate normal (MVN) distribution with mean vector, $\mu = (\mu_1, \ldots, \mu_p)$ where $\mu_i = \mathbb{E}[x_i]$, and a common covariance matrix, Σ, with i, j entry $\operatorname{cov}(x_i, x_j)$. Formally, the multivariate normal density is defined as

$$q(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)\Bigg(\tag{3.6}$$

In the case of $p > 1$ predictors, the LDA classifier assumes that the observations in the kth class are drawn from a MVN distribution $x|y = k \sim N(\mu_k, \Sigma)$, where μ_k is a class-specific mean vector, and Σ is a covariance matrix that is common to all classes. Plugging the density function for the kth class, $p_k(x)$, into Equation (3.5) reveals that the LDA classifier assigns an observation x to the class k for which

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log(\pi_k) \qquad k = 0, 1 \tag{3.7}$$

is largest [14]. In practice, we do not know the parameters of the MVN distributions, so we estimate them using $\mathcal{D}$:

$\hat{\pi}_k = n_k/n,$ where n_k is the number of class-k observations;

$\hat{\mu}_k = \sum_{y_i=k} x_i/n_k,$ where $\sum_{y_i=k}\Bigg($represents the sum over all indices where $y_i = k$;

$$\hat{\Sigma} = \left[\sum_{y_i=0}(x_i - \hat{\mu}_0)(x_i - \hat{\mu}_0)^T + \sum_{y_i=1}\Bigg((x_i - \hat{\mu}_1)(x_i - \hat{\mu}_1)^T\right]/(n-2).$$

Equation (3.7) is linear in x, which indicates that LDA tries to draw the best *linear* decision boundary to classify the observations. When the distributions of the predictors are assumed to be normal, the model is very similar in form to logistic regression, but has some benefits [9]:

- When the classes are well-separated, the parameter estimates for the logistic regression are surprisingly unstable. LDA does not suffer from this.
- If n is small and the distribution of the predictors is approximately normal in each class, LDA is again more stable than logistic regression.
- LDA is very popular when there are more than two response classes.

The LDA classifier, which produces $y = 1$ if $p(x) > 0.5$ and 0 otherwise, has the lowest possible error rate out of all decision rules for this set of posterior probability classifiers [9].

3.3.2 Quadratic Discriminant Analysis

Like LDA, the QDA classifier results from assuming that $x|y = k$ is MVN. However, QDA assumes that each class has its own covariance matrix, unlike LDA, meaning that the distribution of an observation from the kth class is of the form $x|y = k \sim N(\mu_k, \Sigma_k)$, where Σ_k is a covariance matrix for the kth class, $k = 0, 1$. By plugging the resulting density (see Equation (3.6)) into the Bayesian derivation for $p(x)$ (3.5), QDA assigns $y = k$ depending on which of

$$\delta_k(x) = -\frac{1}{2}x^T\Sigma_k^{-1}x + x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\log|\Sigma_k| + \log(\pi_k) \tag{3.8}$$

is largest, $k = 0, 1$. Unlike LDA, x is quadratic in the decision boundary $\delta_k(x)$, resulting in a quadratic decision boundary.

3.4 Support Vector Machine

3.4.1 Hyperplane Classifiers

Support vector machines (SVM) are recently popular classifiers, whose goal is to determine a *separating hyperplane* from the training observations. A p-dimensional hyperplane, for $x \in \mathbb{R}^p$, is of the form

$$\{x : f(x) = x^T\beta + \beta_0 = 0\}, \tag{3.9}$$

where $\|\beta\| = 1$. For x not in the hyperplane, we either have $f(x) > 0$ or $f(x) < 0$, which indicate which side of the hyperplane the observation x lies on. Thus, a classification rule induced by $f(x)$ is

$$G(x) = \text{sign}[x^T\beta + \beta_0]. \tag{3.10}$$

Note here that $G(x)$ is not probabilistic, and directly provides the value of $\hat{y}$ for a given x, i.e. $\hat{y} = 1$ if $G(x) > 0$, $\hat{y} = -1$ if $G(x) < 0$. The positive/negative formulation of y is used for convenience in the hyperplane formulation; this coding is equivalent to $y \in \{0, 1\}$ as we have used previously.

3.4.2 Support Vector Classifiers

The *support vector classifier* chooses the $\beta_j, j = 0, \ldots, p$ to minimize the distances of each of $(x_1, \ldots, x_n)$ to the hyperplane, according to the sign of values y_i. Since it is generally impossible to construct a classifier $\hat{G}(x)$ based on a hyperplane $\hat{f}(x)$ so that each observation exactly produces $\hat{G}(x) = y_i$, we introduce a *cost* C which represents how much we are willing overall to deviate from the hyperplane, and *slack variables* $0 \leq \xi_i \leq 1$ for each $i = 1, \ldots, n$, which can vary depending on how difficult a particular variable is to place on a certain side of the resulting hyperplane. The cost is specified ahead of time, while the ξ_i are part of the minimization procedure. It can be shown that the corresponding minimization problem is [14]

$$\min_{\beta,\beta_0} \frac{1}{2} \sum_{j=1}^{p} \beta_j^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to } \xi_i \geq 0, y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i \ \forall i. \tag{3.11}$$

The solution can be found by finding the Lagrange function, and minimizing accordingly. Alternatively, one can apply the Kuhn-Tucker theorem [4] to obtain a specific solution:

$$\hat{\beta} = \sum_{i=1}^{n} \hat{\alpha}_i y_i x_i. \tag{3.12}$$

The $\hat{\alpha}_i$ depend only on $x_i, y_i, i = 1, \ldots, n$, and are found through the minimization process. The remarkable result here is that this results in a fitted hyperplane of

$$\hat{f}(x) = x^T \hat{\beta} + \hat{\beta}_0 = \sum_{i=1}^{n} \hat{\alpha}_i y_i \langle x, x_i \rangle + \hat{\beta}_0, \tag{3.13}$$

i.e. a closed form involving the inner products of x and the training observation points x_i.

3.4.3 Support Vector Machines

Specifically, *support vector machines* (SVM) are extensions of the support vector classifier that use *kernel* functions that associate "closeness" between x and x_i rather than the traditional dot product $\langle x, x_i \rangle$ used above. These are motivated by the fact that certain shapes in observed data are not necessarily linear in the arguments. Consider for example $x \in \mathbb{R}^2$ where $y = 1$ corresponds to points on a donut and $y = -1$ corresponds to points in the donut hole. In this case, any linear separator $\beta_0 + \beta_1 x_1 + \beta_2 x_2$ is useless, whereas one that took radii into account could work. The major mathematical difference in formulation of SVM and support vector classifier is how their solutions arise. Specifically, the minimization problem above corresponded to minimizing the Kuhn-Tucker dual Lagrangian

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle, \tag{3.14}$$

whereas when we use kernels, we have

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j). \tag{3.15}$$

The result is a classifier

$$\hat{G}(x) = \text{sign}[\hat{f}(x)], \tag{3.16}$$

where

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + \hat{\beta}_0; \tag{3.17}$$

There are three types of kernels that were of interest for this thesis, and those are the *linear*, *polynomial* and *radial/gaussian* kernels. The linear kernel is simply

$$K(x, x') = \langle x, x' \rangle \tag{3.18}$$

which produces the support vector classifier above (see Equation (3.13)). The polynomial kernel of degree d is,

$$K_p(x, x') = (\langle x, x' \rangle + c)^d \tag{3.19}$$

with d being a positive integer greater than 1. This kernel leads to a much more flexible decision boundary than that of the linear kernel since it has a decision boundary of the form of whatever degree the polynomial is [9]. Many libraries slightly change this kernel to include a scale parameter γ and reparameterize c as

$r = \gamma c$, so that

$$\begin{aligned} K_p(x, x'; \gamma) :&= (\gamma\langle x, x'\rangle + r)^d \\ &= (\gamma\langle x, x'\rangle + \gamma c)^d \\ &= \gamma^d(\langle x, x'\rangle + c)^d \\ &= \gamma^d K_p(x, x'). \end{aligned}$$

The other kernel of interest, the radial kernel, takes the form

$$K_r(x, x') = \exp\left(-\gamma \sum_{j=1}^{p} (x_j - x_j')^2\right) \tag{3.20}$$

where γ is a positive constant. The radial kernel has very local behavior, where only nearby training observations (in the sense of Euclidean distance) have an effect on the classification of a test observation, and far off points have essentially no role in the prediction [9].

3.5 Gaussian Processes

A Gaussian process (GP) is a defined as a collection of random variables $(f(x)),\ x \in \mathbb{R}^p$, such that every *finite* collection $(f(x_1), \ldots, f(x_n))^T$ of random variables has a multivariate normal distribution. With a GP comes its *mean* and *covariance* functions, defined as $\mu(x) = \mathbb{E}[f(x)]$ and $C(x, x') = \text{cov}(f(x), f(x'))$ respectively. From a modeling standpoint, the goal is to treat f as an *unknown function* with input x, and use its MVN properties to obtain a posterior prediction given the data. [13].

In modelling, one typically wants to build a GP from their own specified mean and covariance function, and one can show that under certain regularity conditions for C (specifically, it must be symmetric and positive definite), there exists a unique

GP with μ and C as its covariance function; see the Kolmogorov Extension theorem [12].

The GP classification procedure goes as follows. First, specify a prior mean and covariance function; we let $\mu(x) = 0$, this decision is discussed in Section 3.5.1, and any positive definite symmetric function $C : \mathbb{R}^p \times \mathbb{R}^p \mapsto \mathbb{R}$. Next, assume $x_1, \ldots, x_n$ are fixed and $y_1, \ldots, y_n$ are binary. For notational simplicity, when a general x is given, let y_x be its corresponding binary response and $f_x \equiv f(x)$ be the GP evaluated at x (so, in this case, $f(x)$ is univariate normal). The probabilities of y_x are then defined conditionally:

$$p(y_x = 1 | f_x) = \frac{1}{1 + e^{-f_x}} \tag{3.21}$$

Our goal is to produce a probability for prediction

$$p(x) = p(y = 1 | \mathcal{D}) = p(y = 1 | y_1, \ldots, y_n), \tag{3.22}$$

but the probabilities of y and y_i, $i = 1, \ldots, n$ are only defined conditionally through Equation (3.21). One can try to compute $p(x)$ by utilizing tower property type conditioning techniques on the right hand side of Equation (3.22), but no closed form integral exists. To illustrate the difficulty in evaluating $p(x)$, we illustrate first how difficult even the $\mathcal{D} = ((x_1, y_1))$ (a single training observation) is to handle. An outline of the method goes as follows:

- Find the marginal distribution of $p(y_1 = 1)$ by integrating out $f(x_1)$.
- Introduce conditioning on $f(x)$ in Equation (3.22) via the tower property.
- The last step requires the conditional distribution of $f(x)|y_1$, which again can be understood through the tower property by conditioning on $f(x_1)$.

- Bayes' theorem must be used in the previous step to switch the conditioning $f_1|y_1$ to $y_1|f_1$.

The details are provided below. First, through the tower property,

$$\begin{aligned} p(y_x = 1) &= \mathbb{E}_{f_x}[p(y_x = 1|f_x)] \\ &= \mathbb{E}_{f_x}\left[\frac{1}{1+e^{-f_x}}\right] \\ &= \int_{-\infty}^{\infty} \left(\frac{1}{1+e^{-t}}\right) p_{f_x}(t)dt \\ &= \int_{-\infty}^{\infty} \left(\frac{1}{1+e^{-t}}\right) e^{-\frac{t}{2C(x,x)}} dt, \end{aligned} \tag{3.23}$$

which is analytically intractable except in a few specific cases for C.

Now, suppose we observe data (x_1, y_1) and want to produce $p(x)$. we again need to use the tower property, since we only know the distribution of $y'|f'$:

$$\begin{aligned} p(x) = (y_x = 1|y_1) &= \mathbb{E}_{f_x|y_x}[p(y_x = 1|y_1, f_x)] \\ &= \mathbb{E}_{f_x|y_1}\left[\frac{1}{1+e^{-f_x}}\right] \\ &= \int_{-\infty}^{\infty} \left(\frac{1}{1+e^{-s}}\right) p_{f_x|y_1}(s|y_1)ds, \end{aligned} \tag{3.24}$$

where we used the fact that y_1 yields no information on y_x if f_x is known: $p(y_x = 1|y_1, f_x) = p(y_x = 1|f_x)$. The conditional distribution $p_{f_x|y_1}(\cdot|y_1)$ can be understood again through the tower property:

$$\begin{aligned} p_{f_x|y_1}(s|y_1) &= \mathbb{E}_f[p_{f_x|y_1,f_1}(s|y_1, f_1)] \\ &= \int_{-\infty}^{\infty} p_{f_x|y_1,f_1}(s|y_1, t)p_{f_1|y_1}(t|y_1)dt. \end{aligned} \tag{3.25}$$

Now, $p_{f_x|y_1,f_1} = p_{f_x|f_1}$ since y_1 yields no additional information about f_x beyond what f_1 can, and the distribution of $f_x|f_1$ is known since the GP property assures that $(f_1, f_x)^T$ is multivariate normal. For the next term, Bayes' theorem yields

$$p_{f_1|y_1}(t|y_1) = \frac{p_{y_1|f_1}(y_1|f = t)p_{f_1}(t)}{p_{y_1}(y_1)}, \tag{3.26}$$

and $p_{y_1|f_1}(y_1|f_1 = t) = \frac{1}{1+e^{-t}}$, $p_f(t)$ is known (since f is a GP), and we found $p_{y_1}(y_1)$ in Equation (3.23) above. To obtain the final posterior probability $p(x)$, we would plug (3.23) into (3.26), (3.26) into (3.25), and then (3.25) into (3.24). We see that the final result contains several intractable integrals. Thus, we need numerical methods, either MCMC to approximate the non-conjugate prior in (3.26), or other numerical approximations to the intractable integrals. What we presented is the simple case of one training and test point, but in practice we have multiple of both, making the problem even more computationally challenging though the underlying issue of intractable integrals and solution of solving them remains the same.

Luckily, the response of this thesis is binary, so we can look to the Laplace approximation which has similar results to that of an MCMC approximation, but with much less computation time [6]. To put this into perspective, MCMC approximation has a cubic runtime complexity, so n^3, but the Laplace approximation has a worst-case runtime of $n^3/6$, a sixth of the time needed for MCMC approximations. In fact, only one part of the algorithm for Laplace approximation has that time complexity of $n^3/6$, all the other pieces are *quadratic* in runtime, so n^2. This difference in computation time is drastic, and MCMC approximations take a *very* long time to run, so this decrease in computational time is a huge boon for the Laplace approximation.

However, the Laplace Approximation may give a poor approximation to the true shape of the posterior, as it assumes elliptical contours but the peak could be much more broad or narrow [6]. In practice, this has shown to not be too big of a concern as the results are still so close and the time saved is enormous, but this is where the errors come from and is the difference between the Laplace Approximation and MCMC. For this thesis, we will be using the Laplace Approximation for our GP classifications.

3.5.1 Mean and Covariance Priors

Recall that the covariance function is

$$C(x, x') = \text{cov}(f(x), f(x')). \tag{3.27}$$

Since $x \in \mathbb{R}^p$, the covariance function acts on $\mathbb{R}^p \times \mathbb{R}^p$ and maps into $\mathbb{R}$. We focus our analysis on one-dimensional *base kernels* $g_i(x_i, x_i'), i = 1, \ldots, d, \ldots$ which measures similarity in the ith coordinate. These are completely analogous to the kernels described for support vector machines, simply in one dimension. Furthermore, the kernels are assumed stationary, so they only depend on the distance $h_i \equiv |x_i - x_i'|$. The resulting covariance function is

$$C(x, x') \equiv C(h) = \sigma^2 \prod_{i=1}^{p} g_i(x_i, x_i'), \tag{3.28}$$

where σ^2 is referred to as the *process variance*. Each $g_i, i = 1, \ldots, d$ could be the same for each i, but they all depend on *length-scale* parameters θ_i which determine how large the distance h_i can get before covariance diminishes in the ith coordinate.

In this paper specifically, we will be looking at the Gaussian kernel in Equation (3.29) as our covariance function, which is equivalent to the radial basis function for SVM.

$$g_{\text{gau}}(h; \theta) = \exp\left(-\frac{h^2}{2\theta^2}\right) \quad h \geq 0 \tag{3.29}$$

If θ is overestimated, the exponential will behave almost linearly and the higher-dimensional projection will start to lose its non-linear power. On the opposite end, if underestimated, then the function will lack regularization and the decision boundary will be highly sensitive to noise of the training set [6].

With regards to the mean function, when talked about in regard to regression, it has little effect in sample and is mostly used for out of sample prediction [13]. Since our work is not extrapolating, we simply use a prior mean function of $\mu(x) = 0$ which greatly simplifies the fitting process.

Chapter 4

League of Legends Dataset

4.1 Dataset

The dataset was obtained from a website called Oracle's Elixir, which can be found at (https://oracleselixir.com/match-data/), and analyzes all of the 2019 season which is divided into a spring and summer split as well as the international Worlds Championship event. We tried to get access to Riot Games' API to pull data from high-level matches, not just professional matches, but were unable to do so. Nonetheless, this data set was thorough and large enough to continue using on its own, just with the intent of prediction in the LCS instead of high-level games in general.

The raw data was presented with twelve rows per game, corresponding to player and team data. A snapshot can be seen in Table 4.1. We condensed this data into a usable format by changing each section of twelve rows into a single row of data; a snapshot can be seen in Table 4.2. This is the data set that we used for our analysis, which had $n + m = 671$ rows and 29 columns (one of which being a response, so

Table 4.1: First 12 rows and 5 columns of raw data.

Side	**Position**	**Player**	**Team**	**Gold at 10 Minutes**
Blue	Top	Licorice	Cloud9	3668
Blue	Jungle	Svenskeren	Cloud9	3341
Blue	Mid	Nisqy	Cloud9	3719
Blue	ADC	Sneaky	Cloud9	3400
Blue	Support	Zeyzal	Cloud9	1990
Red	Top	V1per	FlyQuest	4205
Red	Jungle	Santorin	FlyQuest	3080
Red	Mid	Pobelter	FlyQuest	3193
Red	ADC	WildTurtle	FlyQuest	2924
Red	Support	JayJ	FlyQuest	2034
Blue	Team	Team	Cloud9	16118
Red	Team	Team	FlyQuest	15436

Table 4.2: First 4 rows and 5 columns of summarized data. All data is with respect to the blue side.

`resultB`	`topB_gdat10`	`topB_gdat15`	`adcB_csdat10`	`adcB_csdat15`
1	10	-481	12	0
1	244	647	-5	-11
1	185	350	19	16
0	720	1047	0	-3

$p = 28$).

Notationally, our binary response is $y \in \{0, 1\}$, which indicates whether or not the blue side won or lost via a 1 or 0 respectively, and each match's specific outcome is an individual response, y_i. We also have our $p = 28$ predictors as $x \in \mathbb{R}^p$, with x_{ij} denoting the observation in the ith row and the jth column.

4.2 Features

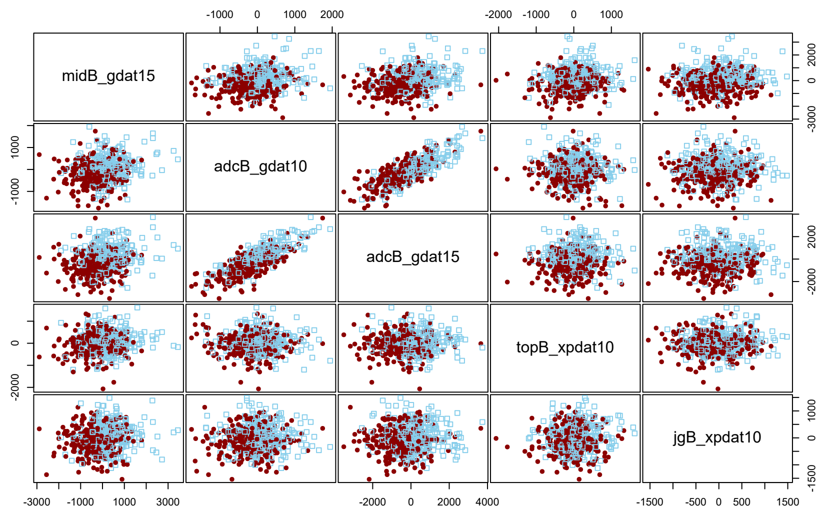


Figure 4.1: Scatterplot matrix of a subset of predictors that are colored based on result, where blue means blue side won, while red means red side has won.

The features, or the columns used for prediction in this new data frame, consisted of many variables which were relative to the blue side team. For example, the outcome y_i, represented as `resultB` was a 1 if the blue team won, and a 0 if the blue team lost. All of these variables can be easily obtained for the red side as well, as they are either binary factors or a numerical variable in which the red side just has the negative quantity of the blue, with the exception of the players and first kill of the match (first blood) and first dragon obtained of the match (first dragon). So, if the result of the blue team (`resultB`) is 1, that means the team on the blue side won and the red side's team lost, and if the blue side top laner's gold difference at 10 minutes (`topB_gdat10`) is 10, we know the red side's is -10.

In Figure 4.1, you can find a scatterplot matrix showing a small subset of predictors. This subset of predictors is colored based on the result of the match, so blue means the blue team won, and red means the red team won (blue team lost). This figure also includes what will be found later on to be the two most important predictors. What we are looking for in scatterplot matrices like these are for clear, easy divides in the coloring of our result to help with our modeling process and to help pick out our predictors. A clear divide, such as a straight line that can divide the red and blue easily, means that this predictor may be an important one to use in classifying the result.

Chapter 5

Results

In this chapter, we will be fitting our models and assessing them, followed by determining which ones were the best to move forward with and evaluating on the test set.

5.1 Fitting and Diagnostics

This section focuses on creating models using the training set and comparing them based on their metrics. In the literature for classification of sports, some people make their own metrics such as the points metric found in [3], and others may report just accuracy [7].

For each of the models below, we first fit them with certain subsets of predictors, or using all predictors, based on which technique is being used. We then performed 10-fold cross-validation on the Acc metric to assess performance of these models, as well as plotted their ROC curves on the training set to get the AUC and compare how different models may perform on different thresholds of η. Both of these metrics are popular in the classification literature, and in our area specifically of esports prediction, [16] uses 10-fold CV on the Acc metric. With a binary response, ROC curves are commonly sought after to help visualize the flexibility of a model at the different probability thresholds of classification, so those are included as well to aid in model selection.

However, while AUC is a flexible metric since it allows us to adjust η and get the TPR and FPR accordingly, it does have its downsides [10]. In prediction of esports results, we tend to have the decision threshold fixed at $\eta = 0.5$, meaning the flexibility of the AUC is not of *great* importance to us, although it still helps with model selection. Due to this, we give a bit more weight towards K-Fold Acc, and that is seen in the literature for esports prediction as well.

5.1.1 Logistic Regression

Going through the model selection process for logistic regression, we did forward stepwise selection (FSS) on the basis of Acc (m1) and AUC (m2). After getting models using these methods, we checked how they all performed on the basis of 10-fold cross-validation on the accuracy. Along with 10-fold CV, we also looked at the area under the ROC curve (AUC) as a factor on which model performed best as well.

Table 5.1: Logistic regression models for consideration, predictors were chosen using FSS on the basis of Acc (m1) and AUC (m2). The highlighted areas represent the winner in that category.

Model	10-Fold CV Acc	AUC
(m1)	0.7799	0.81
(m2)	0.7647	0.83

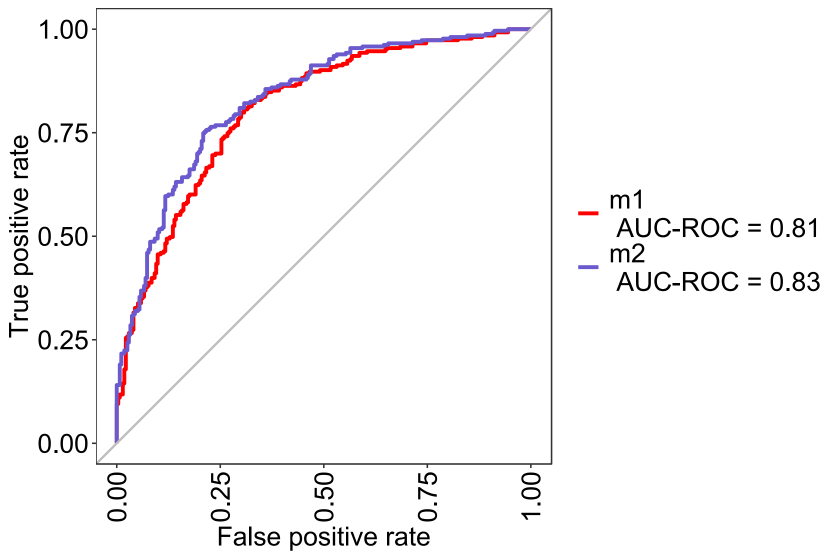


Figure 5.1: AUC of the two logistic regression models from FSS. (m1) shown in red was selected on basis of Acc, while (m2) in blue was selected on the basis of AUC.

First, FSS on the basis of Acc came up with the model (m1) using 11 predictors, detailed in Table 5.2. This model, as expected, had the best 10-fold CV Acc of 0.7799. After this, we performed FFS on the basis of AUC rather than Acc, resulting in the model we call (m2), which had the best AUC between the two models at 0.83. Table 5.1 shows these results as well as the 10-fold CV Acc of (m2), and AUC of (m1), in a more concise manner, and highlights the best performing model on each metric.

While both models are similar in performance, we see that (m1) did have the better accuracy by about 1.5%, while (m2) had the better AUC by about 2%. We lean towards Acc mattering a bit more here as our η is fixed to 0.5, but we look at Figure 5.1 to see that the AUCs are very close with (m2) having a slight edge throughout.

To interpret these coefficients from Table 5.2, we have to remember that these

Table 5.2: (m1)'s coefficients and p-values. The variables are defined in Appendix B.

	Estimate	**P-value**	**Variable Name**
β_0	-0.2965	0.0508	Intercept
β_1	0.0013	1.32e-04	jgB_gdat15
β_2	0.0010	4.05e-09	adcB_gdat15
β_3	-0.0046	0.0043	midB_csdat15
β_4	0.0152	0.0062	topB_csdat15
β_5	-0.0162	0.0019	adcB_csdat15
β_6	0.0012	1.82e-05	midB_gdat15
β_7	0.7175	0.0019	fdB1
β_8	0.0029	0.0031	midB_csdat10
β_9	-0.0011	0.0092	jgB_gdat10
β_{10}	-0.0002	0.0014	adcB_xpdat10
β_{11}	-0.0004	0.0093	midB_gdat10

are for the change in the log-odds of the outcome. Meaning that for a one-unit increase in the gold difference at 15 minutes for the mid laner with everything else held constant, the log-odds of winning will increase by 0.0012, or the odds will increase by a factor of $e^{0.001} \approx 1.0012$. Keep in mind that while these numbers are small, the gold differences at 15 minutes for just the mid laner tend to be in the hundreds or low thousands, with some values even reaching almost 4000. Also note that the blue side getting the first dragon increases the log-odds of winning by 0.7175, or the odds by a factor of $e^{0.7175} \approx 2.0493$, this is the only categorical predictor that was included.

We took the two most significant predictors from (m1), which were the mid

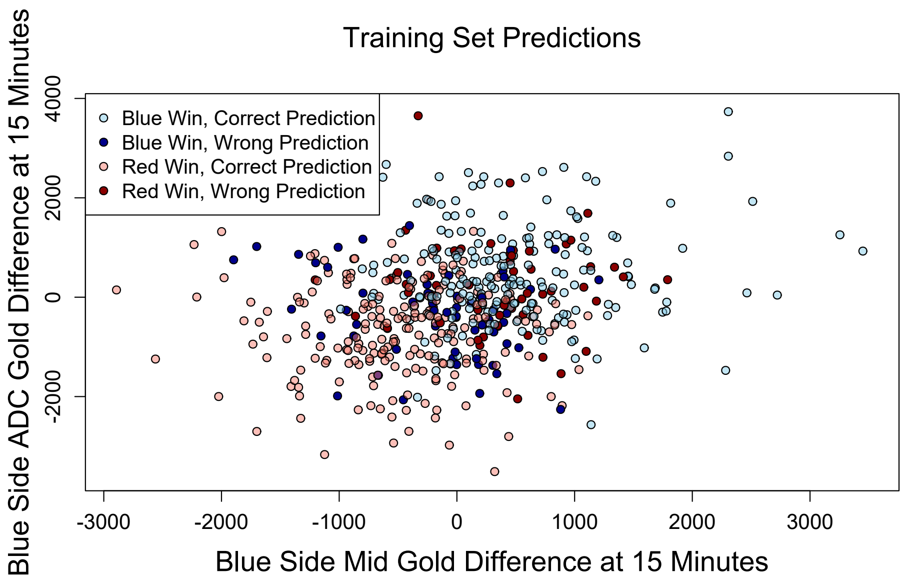
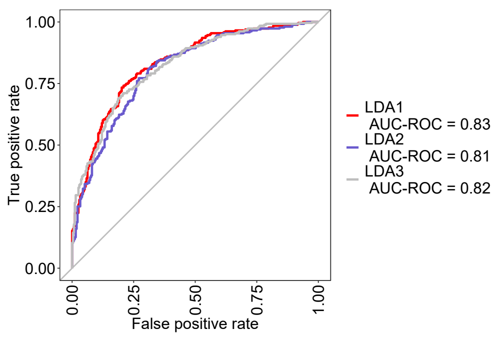


Figure 5.2: Two most important predictors from (m1), the gold difference at 15 minutes for mid laner and ADC, plotted against each other. The blue points represent a blue win, so $y_i = 1$, and kthe red points represent a blue loss, so $y_i = 0$. The darker points are wrong predictions, so a dark blue point is $\hat{y}_i = 0, y_i = 1$, and a dark red point is $\hat{y}_i = 1, y_i = 0$.

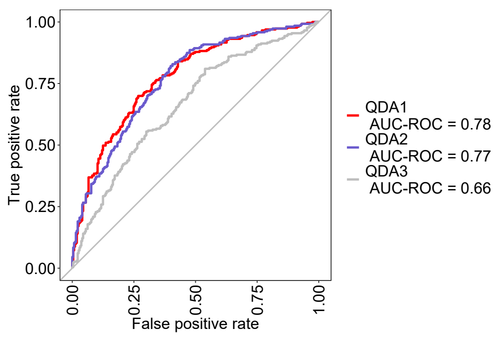
laner's and ADC's gold difference at 15 minutes, and plotted them along the horizontal and vertical axis, while coloring them based on result of the match with darker shades indicating incorrect predictions. This plot can be seen in Figure 5.2. And visually, as expected, the higher the gold difference is for the two roles, the more likely it seems that they will win. There are still some losses in the upper right quadrant, as well as some wins in the lower left quadrant, but the majority tend to be blue and red respectively.

5.1.2 Linear and Quadratic Discriminant Analysis

We created three LDA models: (LDA1) and (LDA2) used the subset of predictors from (m1) and (m2) respectively in the logistic regression section, while (LDA3) was fit using all predictors. When performing QDA, we did the same thing and labelled them as (QDA1), (QDA2), and (QDA3) with the same set of predictors as the corresponding LDA models.



(a) ROC curves for all LDA models.



(b) ROC curves for all QDA models.

Figure 5.3: AUC for the ROC curves of all LDA models on the left, with all QDA models on the right, all on the training set.

Referencing Table 5.3, for the LDA models, (LDA1) has the best performance on both metrics of 10-fold CV Acc and AUC, at values of 0.7609 and 0.83 respectively, giving indication of it being the best LDA model. The table highlights the best performer in each category, while Figure 5.3 shows the ROC curves for all of these models to supplement.

When looking at the same table for the QDA models, we see that (QDA1) is performing the best on both 10-fold CV Acc and AUC, but is noticeably worse than (LDA1). Using all predictors for (LDA3) and (QDA3), we see that LDA does quite well but QDA does very poorly on both metrics. In general with our problem, the

Table 5.3: Three models of each LDA and QDA along with their metrics to help with selection. Highlighted areas are the best performances for each metric.

Model	10-Fold CV Acc	AUC
(LDA1)	0.7609	0.83
(LDA2)	0.7347	0.81
(LDA3)	0.7423	0.82
(QDA1)	0.7144	0.78
(QDA2)	0.6937	0.77
(QDA3)	0.6192	0.66

QDA models seemed to struggle much more than their LDA counterparts.

5.1.3 LASSO

We will be looking at two LASSO models using all predictors and whose λ values were chosen through cross-validation on the two different metrics: (LASSO1) will use Acc, and (LASSO2) will use AUC.

Table 5.4: Table of LASSO model information on the training set. (LASSO1) had λ chosen on the basis of Acc, while (LASSO2)'s was chosen on the basis of AUC. The highlighted area represents the best performance on that metric. Both AUCs are equivalent, so neither one was highlighted.

Model	10-Fold CV Accuracy	AUC	λ
(LASSO1)	0.7535	0.83	0.0040
(LASSO2)	0.7501	0.83	0.0040

(LASSO1) was using $\lambda = 0.0040$ and included *all* predictors except for `jgB_xpdat10`. The other LASSO model, (LASSO2), uses $\lambda = 0.0040$ as well, but cut out many more predictors than (LASSO1), it used all predictors **except**: `ftB, supp_fbB, adcB_csdat15, midB_csdat15, topB_csdat10, adcB_xpdat10, midB_xpdat10, jgB_xpdat10`. leaving 21 predictors. Reference Appendix B again for what these predictors represent.

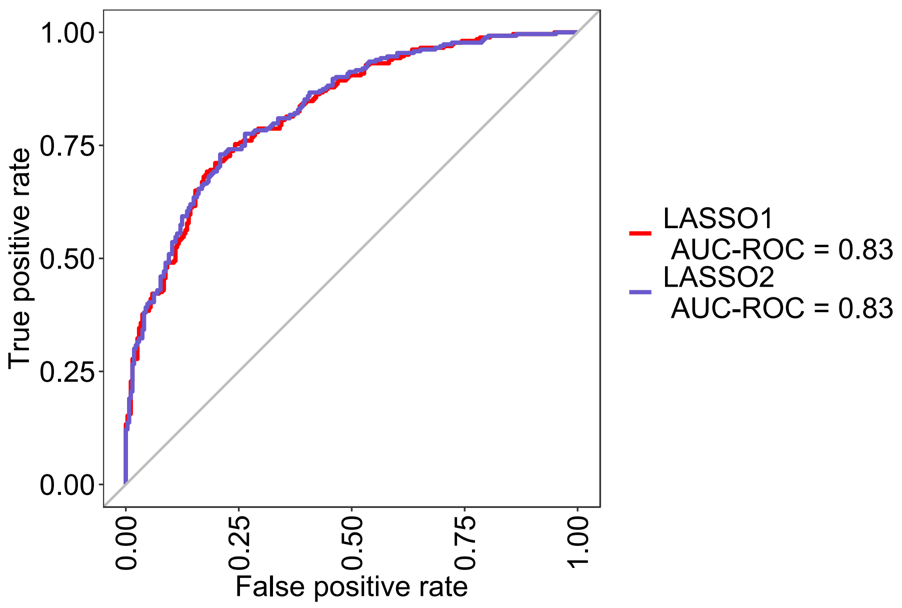


Figure 5.4: ROC curves for (LASSO1) and (LASSO2) on the training set.

When assessing the metrics for each LASSO model, (LASSO1) performed best on the basis of 10-fold CV Acc with a value of 0.7535, though (LASSO2)'s was just barely worse. The two models are very similar in performance, even having equivalent AUCs of 0.83. The ROC curves for these models can be seen in Figure 5.4 and Table 5.4 will provide the metrics.

5.1.4 Support Vector Machine

When creating different SVMs, we tried the linear kernel, the polynomial kernel with degrees $d = 2, 3$, and the radial kernel. The motivation for these kernels, aside from them being regularly used, is that the data looked as if it could be divided by a linear decision boundary or polynomial the best when looking at the 2-dimensional plots in Figure 4.1, maybe a large circle could also cover the the classifications well.

All of these SVMs had their hyperparameters tuned from a list of $c = \{0.1, 0.25, 0.5, 1, 1.5, 2, 4, 8, 10, 16, 25, 32, 64, 100\}$ and for the radial kernel, we also had $\gamma = \{0.5, 1, 2, 3, 4\}$, as these values help cover both common and extreme cases while there are not large differences outside of these ranges [9]. The hyperparameters were chosen through 10-fold cross validation. These models also used all predictors as SVMs are supposed to be fairly resistant to overfitting, which can be adjusted based on the cost parameter [14].

Table 5.5: Training set evaluation of the SVMs using different kernels with all predictors. The best performing metrics are highlighted.

Kernel	10-Fold CV Acc	AUC
Linear	0.7591	0.83
Polynomial with $d = 2$	0.7552	0.82
Polynomial with $d = 3$	0.7515	0.81
Radial	0.7050	0.74

When assessing the performance of the different SVM kernels, we found that the linear kernel with $c = 0.1$ performed the best on both metrics with a 10-fold CV Acc of 0.7591 and an AUC of 0.83. The polynomial kernels with degree $d = 2$ and $c = 100$, and the kernel with degree $d = 3$ and $c = 8$, both were near the performance of the linear kernel, but were getting worse as the degree increased up to the cubic. Finally, the radial kernel performed the worst on both metrics by a significant margin, being nowhere near that of the linear or polynomial kernels. These results can be seen in Table 5.5, with the ROC curves for each of these kernels to supplement in Figure 5.5.

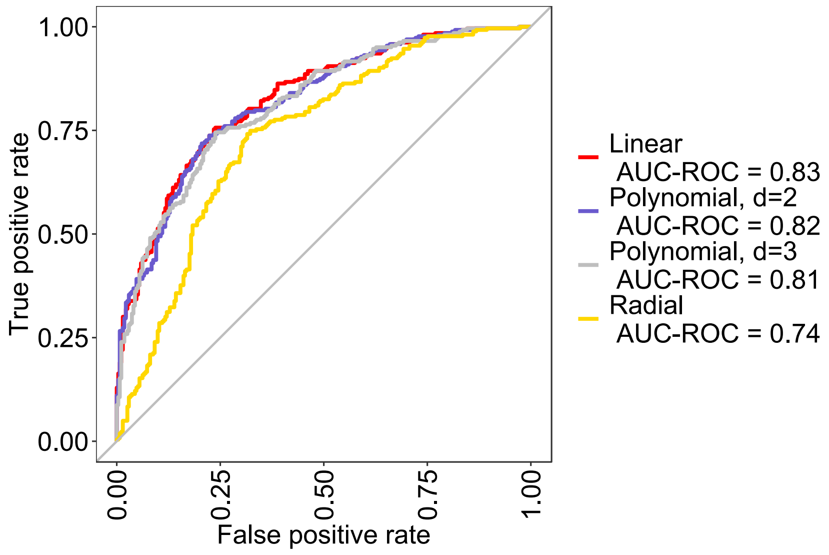


Figure 5.5: ROC curves on the training set for each of the kernels used for SVMs.

5.1.5 Gaussian Processes

For Gaussian process classification, we fit three GPs all on the Gaussian (radial) kernel: (GP1) uses the same set of predictors as that of (m1) while (GP2) uses the same set of predictors as that of (m2), both from the logistic regression section, and (GP3) uses all predictors. All of the GPs used the zero function as their mean function. To assess how well these models do, we look at both the 10-fold CV Acc and the AUC, as we have done with other methods thus far. The Laplace

Approximation for the binary GP classifier mentioned before is used as well.

Table 5.6: Training set evaluation of the GPs using the Gaussian kernel. The best performing metrics are highlighted.

Model	10-Fold CV Acc	AUC
(GP1)	0.7330	0.81
(GP2)	0.7216	0.78
(GP3)	0.7320	0.80

Looking at Table 5.6, we see that (GP1) performed the best on both metrics with a 10-fold CV Acc of 0.7330 and an AUC of 0.81. Notice that with (GP3), the one using all predictors, the performance was not far off of (GP1), only a slight bit worse on 10-fold CV Acc (less than 1%), and only 1% worse on the AUC. The ROC curves for the three GPs can be seen in Figure 5.6.

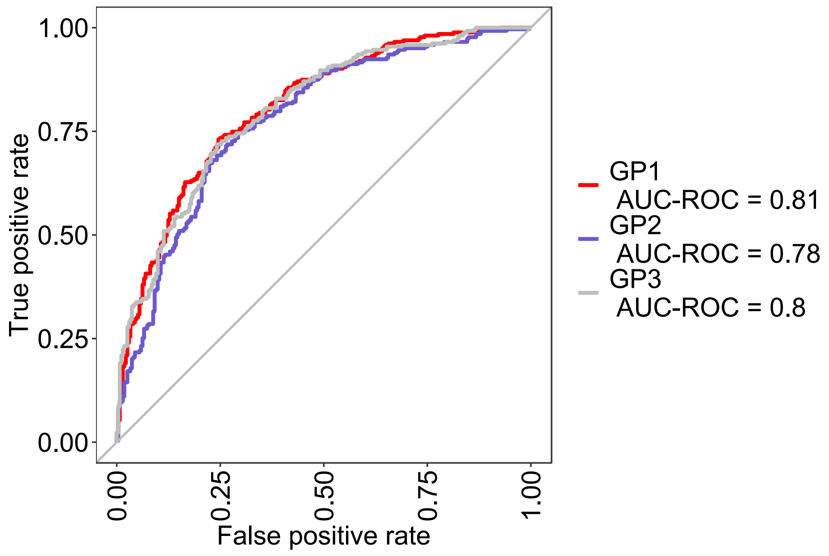


Figure 5.6: ROC curves for each of the GPs using the Gaussian kernel. (GP1) and (GP2) used the predictors from (m1) and (m2) respectively, while (GP3) used all predictors.

5.1.6 Training Set Results

Table 5.7 gives the previous results of the best performing models in a condensed manner. We can see that with regards to 10-Fold CV Acc, (m1) from logistic regression did the best, with (LDA1) and (SVM-Lin) being very close, and (LASSO1) a bit behind those. The best performing GP with the Gaussian kernel, (GP1), did noticeably worse than models mentioned above, but (QDA1) did much worse than *every* model.

On the basis of AUC, however, (LDA1) performed equally as well as (LASSO1) and (SVM-Lin), while (GP1) and (m1) were not far behind. Again, (QDA1) did substantially worse than every other method, so it did poorly with regard to both metrics.

Table 5.7: Information of the metrics for the best performing models on the training set using each method.

Method	**Model**	**10-Fold CV Acc**	**AUC**
Logistic Regression	`(m1)`	0.7647	0.81
LDA	`(LDA1)`	0.7609	0.83
QDA	`(QDA1)`	0.7144	0.78
LASSO	`(LASSO1)`	0.7535	0.83
SVM	`(SVM-Lin)`	0.7591	0.83
Gaussian Process	`(GP1)`	0.7330	0.81

Looking at all metrics, there was no clear model that was ahead of the rest. A case could be made for many of the models outside of (QDA1), even GPs with how they behave when local observations are taken into account and considering the domain of the problem. (LDA1) had the second highest 10-Fold CV Acc and was tied for the best AUC, and while (m1) had the best 10-Fold CV Acc by a small margin (less than 0.5%), its AUC was 2% lower than that of (LASSO1), (SVM-Lin), and (LDA1), all of which performed well still on the 10-Fold CV Acc metric.

5.2 Test Set Evaluation

For the test set evaluation, we took each of the models from Table 5.7 and evaluated their Acc as well as their AUC on the test set. We then look at the models to see if any of them clearly performed the best on both metrics. If there were not any, then we look to see which methods may have an edge on certain metrics, such as Acc or AUC.

Table 5.8: Performance metrics of the best performing models evaluated on the test set.

Method	**Model**	**Acc**	**AUC**
Logistic Regression	`(m1)`	0.7556	0.80
LDA	`(LDA1)`	0.7556	0.84
QDA	`(QDA1)`	0.7704	0.84
LASSO	`(LASSO1)`	0.7481	0.82
SVM	`(SVM-Lin)`	0.7333	0.82
Gaussian Process	`(GP1)`	0.7852	0.83

To view the performance of each model on the test set, we look to Table 5.8. In terms of Acc on the test set, we see that (GP1) performs the best, while it actually had a fairly poor performance on the 10-Fold CV Acc above. While this was fairly surprising, also note how well (QDA1) performed on the test set, about 1.5% behind (GP1), while it was the *worst* performing on 10-Fold CV Acc. A bit behind (QDA1) is both (m1) as well as (LDA1), who have equivalent Acc on the test set and both were the two best performing on the basis of 10-Fold CV Acc. Surprisingly, (SVM-Lin) performed the worst here, when it was not far behind (m1) and (LDA1) when evaluating on the training set.

Now, looking at the AUC on the test set, we notice that both (LDA1) and (QDA1) are tied for performing the best on this metric. Again, what is most surprising is how well (QDA1) has done on this metric when referencing its training set counterpart in Table 5.7, where it performed the worst by a significant margin.

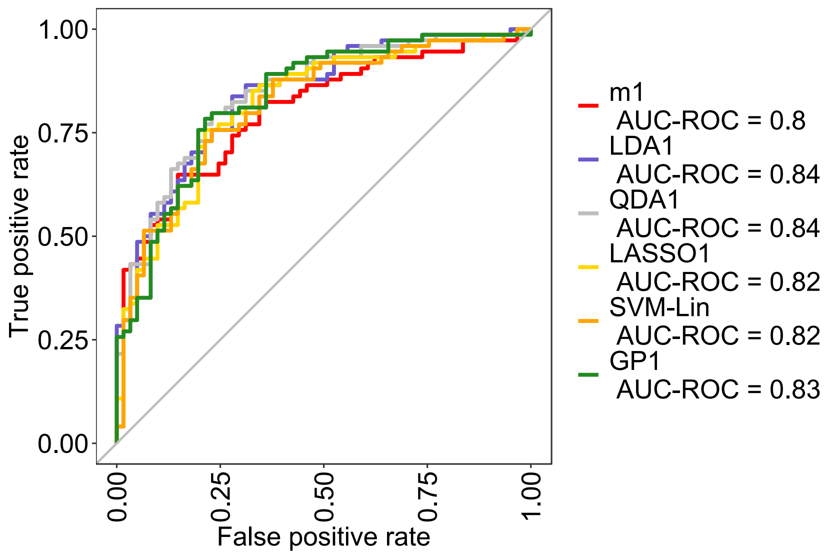


Figure 5.7: ROC curves for all of the best performing models evaluated on the test set.

(LDA1) performed very well on both the training and test set, so this was not much of a surprise to see. Next, just barely behind those two models, was (GP1), which performed well on this basis as well on the training set. Behind those was (LASSO1) and (SVM-Lin), which had equivalent AUCs, and last was (m1) who was noticeably worse than the others. While (m1) was not the best performing on the training set's AUC, performing so poorly on the test set was a bit of a shock. The ROC curves for each of these models on the test set can be found in Figure 5.7, notice how they are mostly indistinguishable from each other, though (m1) does stick out a bit below the others.

When looking at the overall performance of models on the test set, (QDA1) seemed to have the biggest improvement, but (GP1) saw large improvement as well. (QDA1)'s Acc and AUC improving as much as it did was not something that we expected intuitively, nor was the magnitude at which (GP1) improved. While we did expect (GP1) to at least stay consistent due to the nature of how Gaussian processes deal with data in a domain such as this where many observations are local to each other, we did not imagine it to jump so high.

We note that K-fold CV Acc is intended to estimate test set accuracy, so we investigate possible reasons as to why there is a discrepancy in relative model performance on these metrics. First, the size of our training set ($n = 536$), though the test set has a much smaller size ($m = 135$), so the test set may have simply not been large enough to produce a reliable large sample test set error. It is also possible that 10 folds may have not been enough for K-fold CV to be representative of test set prediction error. One possibility as to why GPs did so well on the test set compared to the training set is that their parameters are fixed across the K-folds, meaning they may have been somehow correlated with the training set values. Refitting hyperparameters on each fold is a possibility for future studies, however, this is computationally intensive.

Chapter 6

Conclusion

6.1 Discussion

As with traditional sports, making models to predict the result of an esports game is also a difficult task. With so many factors being considered (some of which we did not touch on, but could lead down a very complex rabbit hole of work), it is nearly impossible to pinpoint what may be considered a “best” model. We went through many popular methods, such as logistic regression, LDA, QDA, SVMs, and the lesser-practiced Gaussian processes, to try and come up with an ideal fitting model. Using tools such as plots, model diagnostics, and powerful software libraries, we can run through this selection process to compare methods and see what works best to make an educated decision on what to use for this problem.

Thankfully, all of these methods seemed to be around a similar ballpark in terms of 10-Fold CV Acc, with the exception of QDA, displayed in Table 5.7. Even when assessing their performances on the test set in Table 5.8 we can see how similar most methods are, and how some of the methods that did poorly on the training set come out to do well on the test set.

In the end, (m1) and (LDA1) did extremely well on the training set, while (GP1) seemed to perform the best overall on the test set with Acc mattering a bit more than AUC for this area of classification. The theory behind GPs also seems fit for this type of problem, as local behavior is usually something to be more considered in this space, as we will rarely (if ever) have a gold difference for any one specific role in a LoL match hitting large magnitudes from the average, such as a gold difference of 6000 at 15 minutes. Gaussian processes work very well when points are observed and desired predictions are nearby, making this something that should be wanted for usage in the prediction space of League of Legends.

Overall, when taking into account both the training and test set results, (LDA1) was consistently doing well across the board on all metrics. Intuitively this makes sense as it seems that a line can be drawn through the center of many two-predictor plots to separate blue and red wins (see Figures 4.1 and 5.2), which can extend as a hyperplane through many-predictor results. We thought QDA would do very well for this also, as a quadratic separator also seems viable visually, with its flexibility to catch prediction errors in the center with its vertex (see Figure 5.2).

6.2 Future Work

As mentioned in the prior section, there are a *lot* of variables to use with esports prediction, and many other factors to consider than what we used. Had we been given access to different data, such as information at *every* minute up to 15 minutes, and including all the statistics at 15 minutes and even 10 minutes, we would have been able to do much more. This would allow for sort of "live" predictions and seeing which events turn out to be most monumental or truly game-changing in a match. We also would like to be able to pull this data from high-ranked matches, not just professional matches. With access to something like that, our data would increase in size dramatically and be relevant not just in the esports/professional scene, but in high-ranked solo queue games as well, without the professional organization of a team.

Additional data to incorporate could be champions picked per role, player names, and possibly interactions between those factors. Generally, players tend to be champion "specialists", in the sense that they excel when using certain champions. Furthermore, certain champions often perform better or worse against others, clearly influencing win-rate. All of this data is available and fixed at the beginning of the game. It would, however, require a much larger data set, since certain players only appear in a handful of matches in our data set.

Another issue that we would have liked to go over would be hand-picking different sets of predictors for the models outside of logistic regression. Trying to find the best subsets for methods such as GPs, LDA and QDA where all predictors did not do the best (but did well in some cases) could come up with a different set of predictors to use than that of our forward stepwise selection on logistic regression.

We would also like to look into other methods that were not used here but are more popular in the machine learning area. Methods such as random forests, gradient-boosted trees, and neural networks are all very popular in classification problems, and are more popular in the computer science field. Since there is much overlap between statistical and machine learning areas, we think these could all be worth looking into.

6.3 Final Words

Currently, professional League of Legends broadcasts have hundreds of thousands of viewers in a single weekend, and the game has a large monetary involvement (for example, the average salary of top players is more than $300,000). Despite this, broadcasts currently lack match prediction statistics, and from our findings, there is little to nothing in the literature about current LoL match prediction.

This paper has illustrated that even with a fraction of possible predictors in a LoL data set, our models can accurately predict which team wins approximately 78% of the time, and our discussion in Section 6.2 provides a pathway to many possible improvements. We believe that the work in this paper can serve as a guide to other researchers with similar goals, to esports organizations to identify how to improve their team's performance, and even to companies like Riot games to help supplement their esports engagements.

Bibliography

[1] A. Starkey. League of legends worlds 2019 final beats fortnite to break twitch record with 1.7 million viewers, 2019. URL https://metro.co.uk/2019/11/11/league-legends-worlds-2019-final-beats-fortnite-break-twitch-record-1-7-million-viewers-11077467/.

[2] Bruce Ankenman, Barry L Nelson, and Jeremy Staum. Stochastic kriging for simulation metamodeling. In *2008 Winter Simulation Conference*, pages 362–370. IEEE, 2008.

[3] C. Choe H. Eom B. McKay B. Min, J. Kim. *A compound framework for sports results prediction: A football case study*. 2008.

[4] Vladimir Boltyanski, Horst Martini, and Valeriu Soltan. *Geometric methods and optimization problems*, volume 4. Springer Science & Business Media, 2013.

[5] C. Patterson. Average lcs player salary revealed and it's surprisingly high, 2019. URL https://www.dexerto.com/league-of-legends/average-lcs-player-salary-revealed-surprisingly-high-575486.

[6] C. Williams C. Rasmussen. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[7] C. Cao. *Sports Data Mining Technology Used in Basketball Outcome Prediction*. 2012.

[8] Esports Contributors. Lcs spring 2019 viewership, 2019. URL https://escharts.com/tournaments/lol/lcs-spring-2019.

[9] T. Hastie R. Tibshirani G. James, D. Witten. *An Introduction to Statistical Learning*. Springer, 2013.

[10] R. Real J. Lobo, A. Valverde. *AUC: a misleading measure of the performance of predictive distribution models*. 2007.

[11] K. Webb. More than 100 million people watched the league of legends world championship, cementing its place as the most popular esport, 2019. URL https://www.businessinsider.com/league-of-legends-world-championship-100-million-viewers-2019-12.

[12] Daniel Revuz and Marc Yor. *Continuous martingales and Brownian motion*, volume 293. Springer Science & Business Media, 2013.

[13] J. Risk. *Three Applications of Gaussian Process Modeling in Evaluation of Longevity Risk Management*. 2017.

[14] J. Friedman T. Hastie, R. Tibshirani. *The Elements of Statistical Learning*. Springer, 2001.

[15] T. Spangler. Ten franchise teams for league of legends north american esports league unveiled, 2017. URL https://variety.com/2017/digital/news/league-of-legends-north-america-league-teams-esports-1202619230/.

[16] Y. Lei Y. Yang, T. Qin. *Real-time eSports Match Result Prediction*. 2016.

Appendix A

League of Legends Game Concepts

League of Legends (LoL) falls into the video game genre of a Multiplayer Online Battle Arena (MOBA) and is developed as well as published by Riot Games. The map that players compete against each other on is always the same, and is a symmetric path with three lanes: top, middle, and bottom. Between these lanes are areas that contain neutral objectives that are not controlled by any players, and these areas are called the jungle. Each of the five players on a team is designated a role that typically specifies the lane they will be in for most of the game, which can be referred to below.

Role	**Lane**
Top	Top
Jungle	Jungle
Mid	Middle
Attack Damage Carry (ADC)	Bottom
Support	Bottom

In LoL, the bases are located in the lower-left and upper-right locations on the map, and the goal of the game is to destroy your enemy's central unit, called the nexus, to claim victory. In order to reach the nexus, however, you must destroy enough towers (turrets) and at least one inhibitor. Each lane consists of an outer, inner, and nexus turret, with an inhibitor in between the inner and nexus turrets that spawns stronger creeps/minions for the team that destroyed that inhibitor in that specific lane. Since the map is symmetric, this is true for both sides, so 3 ally turrets in one lane and an ally inhibitor, while there are three enemy turrets and an enemy inhibitor directly mirroring in the same lane. The quickest way to the reach the nexus is to simply go down one lane, but this is usually not possible, especially in a professional setting.

Destroying turrets, killing an enemy champion (they respawn after a certain time, which gets longer as the match extends), and hitting smaller minions (creeps) the last time before they die (referred to as last-hitting) all generate gold, as well as experience to level up a character that is nearby, giving them more skill points to assign to their skills and an increase to their stats. Gold is earned per champion and there are bonus ways to get gold, such as getting the first kill in a match, destroying the first turret and turret plates before the turret plates fall, shutting down an enemy that is on a kill streak, etc.

There are currently 148 unique champions in league, each being categorized by a certain role: assassin, fighter, mage, marksman, support, and tank. While these roles tend to align with the lane the champion goes to (such as mage usually being a mid lane role), there is nothing to enforce this categorization. The role also tends to imply certain stats or stat growth on the champion, such as tanks having higher health pools and higher defense/magic resistance. Each of these champions also has a set of four unique abilities which they can enhance one of each time they gain a level. Due to all of these unique characterizations on each champion, it is natural that matchups are considered in the competitive realm, such as the fact that some champions who are ranged will have an advantage in lane against a champion who is melee focused.

All of these terms mentioned are things that influence the outcome of a game, although some more than others. Since we are looking at the 10- and 15- minute predictions, it makes sense for us to look at certain factors such as the difference in creep score between competitors of the same lane, gold differences, which team got the first kill or first turret, among many others. Items in League of Legends are what make certain characters spike in terms of power, as well as certain level thresholds, and whether or not they are scaling into the late-game or if they are powerful early on. While some of these are hard to keep track of and require much time and care into modeling with them, we anticipate that certain factors such as the in-game currency and creep scores to have large influence on the outcome of a match. As such, they are a subset of the predictors that we will be looking into.

Clearly, the game is very complex, and it is very difficult to cover everything about the background of the game and its nuances when that is not this paper's purpose. If interested, more reading cane be found on the League of Legends website, https://na.leagueoflegends.com/en-us/how-to-play/.

A.1 League of Legends Championship Series

In 2013, League of Legends publisher and developer Riot Games started the League of Legends Championship Series (LCS), where 10 professional teams consisting of five main players each and sometimes about three substitute players compete each season. The LCS has a championship series in multiple regions but using different abbreviations. North America's is just called the LCS, Europe's series is now called the LEC, South Korea's is called the League of Legends Champions Korea (LCK), China's is now Tencent League of Legends Pro League (LPL), Taiwan's is the League of Legends Master Series (LMS), and there are many other minor regions who have a Riot Games-sanctioned series as well. These games are streamed weekly on a website called Twitch.tv, where NA alone brings in more than 100,000 viewers each weekend [8]. There are also two major international events that happen once a year each: Mid-season Invitational and Worlds Championship Series, where the best teams in the world gather to compete.

The growth for the LCS alone has been huge, with franchising coming in to secure a team's spot in the series as well as players'. Franchising brought in many well-known backers, including traditional sports teams such as the Golden State Warriors and former NBA professionals such as Rick Fox [15]. The average salary in the LCS was $105,000 in January of 2017, and as of April 2019 the average salary has roughly tripled to be a little more than $300,000 with the minimum base salary being $75,000 [5].

With all of this backing from well-known and established companies as well as venture capitalists investing in teams, the tremendous growth in viewership, as well as salaries, it is clear that people believe esports is worth looking at. The LCS has a huge broadcast each weekend with great quality and casters just like traditional sports have. Along with these relations to traditional sports also comes the discussion of statistics in between games and sometimes at the start of a game.

Appendix B

Summarized Data Descriptions

Table B.1: Column names and description of the binary predictors used.

`topB_fbB`, `jgB_fbB`, `midB_fbB`, `adcB_fbB`, `suppB_fbB`	Binary variable indicating whether or not a member in their respective role on the blue side got first blood (first kill of the game). 1 represents yes, 0 represents no.
`fdB`	Binary variable indicating whether or not any member on the blue side got the first dragon. 1 represents yes, 0 represents no.
`ftB`	Binary variable indicating whether or not any member on the blue side destroyed the first turret. 1 represents yes, 0 represents no.

Table B.2: Column names and description of the differences in creep score, gold, and experience for each role, as well as the response of the data..

resultB	The response to be predicted. Blue side's result. 1 represents a win, 0 represents a loss.
topB_gdat10, jgB_gdat10, midB_gdat10, adcB_gdat10, suppB_gdat10	Gold difference at 10 minutes with respect to the blue side in each role.
topB_gdat15, jgB_gdat15, midB_gdat15, adcB_gdat15, suppB_gdat15	Gold difference at 15 minutes with respect to the blue side in each role.
topB_xpat10, jgB_xpat10, midB_xpat10, adcB_xpat10, suppB_xpat10	Experience difference at 10 minutes with respect to the blue side in each role.
topB_csdat10, jgB_csdat10, midB_csdat10, adcB_csdat10, suppB_csdat10	Creep Score difference at 10 minutes with respect to the blue side in each role.
topB_csdat15, jgB_csdat15, midB_csdat15, adcB_csdat15, suppB_csdat15	Creep Score difference at 15 minutes with respect to the blue side in each role.