University of Delaware

Contemporary Applications of Mathematics

Probabilistic Modeling of Situational At Bat Interactions

Authors:
Ryan McKenna
Briana Lamet
Michela Gao
Benjamin King

December 9, 2015

Abstract

In this report, we introduce a set of mathematical models that are designed to accurately predict the distribution of outcomes in a particular at bat interaction. Our approach differs from more commonly used approximations in two ways. First, our models are built using more factors than are usually incorporated in standard models. Second, our models are player-specific, meaning that we utilize behavioral information about the players involved in the at bat, so that the models can identify situations in which the strengths of the pitcher correspond with the weaknesses of the batter, and vice versa, ultimately leading to higher quality predictions.

1 Introduction

Baseball is not only a great American pastime, but also a great game of statistics. Sabermetrics is a term that comes from "SABR" which is the "Society for American Baseball Research" to describe the empirical analysis of baseball. The movie "Moneyball" (2011) is just the tip of the iceberg when it comes to how valuable such statistical research can be to the actual game play. As a result, there is a large data pool of baseball statistics that is readily available for empirical analysis.

Data is becoming increasingly important in a wide variety of fields today, including sports, and baseball in particular. Many MLB teams are starting to realize the importance of understanding the wealth of data readily available in the baseball community. Sabermetrics was invented to give teams a competitive advantage in all aspects of the game - from making in-game decisions to figuring out how to optimize a lineup, or to make trade proposals.

1.1 The Problem

We seek to build a mathematical model from that can make predictions about the outcome of any particular at bat. To that end, we would like to be able to predict the likelihood of each possible outcome given a situation where the batter, pitcher, and game invariants are known before the at bat starts. Currently, formulas exist to approximate these types of interactions, such as the log5 formula and simple linear regression formulas. These formulas use common statistics tracked and reported on baseball websites, but typically ignore other possibly relevant important for sake of simplicity. We seek to improve upon these methods by taking into account more information. In particular, we want to utilize data tracked on pitches, as this type of information often provides a unique insight into the behavior of the players involved in the pitch.

Once we have a model, we must be able to evaluate the quality of it's predictions. To do this, we are going to use two well known objective functions: log-loss and Brier score because these are often used to rank probabilistic classifiers. Both scoring metrics have the property that the expected score is minimized when the assigned probabilities match the true probabilities. Our formal objective is to construct a mathematical model that is attains the lowest possible log-loss score. The Brier score will be used to understand where the model thrives and where it needs work and will be discussed in more detail in the results section.

1.2 Impact

The impact of this work is potentially huge. Having an accurate handle on batter-pitcher interactions would allow for more accurate baseball simulations. Major League Baseball teams could use

this information to make a decision about possible substitutions for particular at-bats. For example, the fielding team can select a reliever to maximize the probability of getting an out against a specific batter in key game situation. The batting team can do something similar when considering making a pinch hitter. Batting teams can use this information to set up an optimal lineup order. In essence, teams would be able to create at-bat situations with their ideal probabilities using our models.

2 The Setup

2.1 The Data

Every MLB game ever played since 2008 has been electronically recorded through MLB.com's GameDay service, which is a program that allows fans to track live baseball games through the internet. The GameDay service records a wealth of data about events that happen during baseball games and it makes this data openly available[1] in XML format to baseball fans and statistics enthusiasts alike. We scraped this website for every game from the 2010-2014 seasons, which contained over one million at bats and almost four million pitches. The data was parsed into three files: atbats.csv, games.csv, and pitches.csv, and samples from these files are shown in tables **??**, **??**, and **??** respectively.

ab_num	game_path	pitcher_id	batter_id	event	b_stand	p_throws	inning_half
1	*	407853	461416	Strikeout	L	L	top
2	*	407853	465784	Strikeout	R	L	top
3	*	407853	408305	Groundout	R	L	top
4	*	431162	400085	Groundout	L	R	bottom
5	*	431162	408210	Groundout	L	R	bottom

Table 1: This table shows a sample of the data in the atbats.csv file. Note that this table can be merged with the games table by doing an inner join on the game_path columns.

game_path	date	time	away_team	home_team
*	11-2-27	7:05 PM	sdn	sea
*	11-2-27	7:05 PM	kca	tex
*	11-2-27	7:05 PM	bos	min
*	11-2-27	7:05 PM	oak	chn
*	11-2-28	7:05 PM	oak	ana

Table 2: This table shows a sample of the data in the games.csv file. Note that game_path acts as a unique id for the row, and the value has been ommitted to save space.

2.2 Understanding the Data

In order to understand our models, you must first understand the data. We are ultimately interested in predicting the outcome of an at bat, and the possible outcomes are shown in table **??**. However,

[1] http://gd2.mlb.com/components/game/mlb/

ab_num	game_path	type	x	y	des	balls	strikes
1	*	FF	114.16	149.38	Called Strike	0	0
1	*	CU	120.17	145.06	Called Strike	0	1
1	*	FF	133.91	146.79	Ball	0	2
1	*	CU	121.03	151.97	Foul	1	2
1	*	FC	79.83	142.47	Swinging Strike	1	2

Table 3: This table shows a sample of the data in the pitches.csv file. Note this table can be merged with the atbats table by doing an inner join on the ab_num and game_path columns.

the majority of the items in the table can be categorized as a hit, and out, or a walk. To simplify the prediction task, we will only try to predict these three high level categories, and leave finding more fine grained predictions as future work. Outcomes that can not be categorized in this way (like interference) are rare and generally unpredictable, so we will ignore those outcomes as well. In a similar manner, there are a number of possible outcomes for a pitch, but each of these outcomes can be categorized as a ball, a strike, a foul, a hit, an out, or a walk (hit by pitch), so do that as well.

It should be fairly intuitive what each of the columns in the data tables mean, and the sample data should help enforce that intuition. game_path serves as a unique id for games, and ab_num serves as a unique id for at bats within a specific game. pitcher_id and batter_id are unique ids assigned to the pitcher and batter involved in the current at bat. event is the outcome of the at bat, which we are ultimately trying to predict. b_stand and p_throws are the dominant hands of the batter and the pitcher respectively, and inning_half is either top or bottom – if this value is top, then the pitcher is home, otherwise the batter is home. type is an abbreviation for the type of pitch the pitcher threw, and is classified based on the trajectory of the pitch using a neural net classification algorithm (**?**). x and y are the locations that the ball crossed the plate, but the units of measurement and the choice of origin relative to the plate are not clear. Finally, balls is the number of balls in the at bat when the pitch took place, and strikes is the number of strikes in the at bat when the pitch took place.

Outcome	Occurrences	Outcome	Occurrences	Outcome	Occurrences
Strikeout	205658	Grounded Into DP	21313	Fielders Choice Out	1956
Groundout	204381	Field Error	10312	Bunt Pop Out	990
Single	165254	Hit By Pitch	9190	Strikeout - DP	827
Flyout	142364	Sac Bunt	8094	Fielders Choice	607
Walk	80147	Sac Fly	7341	Fan Interference	239
Pop Out	52493	Intent Walk	5668	Batter Interference	209
Double	48755	Triple	5279	Catcher Interference	156
Lineout	44215	Double Play	2918	Sac Fly DP	115
Home Run	26521	Runner Out	2629	Bunt Lineout	59
Forceout	22913	Bunt Groundout	2397	Triple Play	19

Table 4: This table shows what the possible outcomes of an at bat could be, and the number of occurrences of each in the data set.

2.3 Pre-processing the Data

Before we can analyze the data, it is necessary to clean it up and transform it into a usable format. This required multiple steps:

- Merge (inner join) the games (which contain information about the given situation) with the at bats.
- Merge (inner join) the at bats with the pitches
- Group the at bat outcomes into three equivalence classes: hit, out, and walk
- Group the pitch outcomes into 5 equivalence classes: ball, strike, foul, hit, out, hit by pitch (walk)
- Convert text columns to numeric columns (machine learning framework only works with numeric data)
- Derive day/night column from time column
- Remove noisy data: this includes at bats which could not be categorized as hit,out, or walk, games that didn't occur in the regular season[2], and pitches that could not be put into one of the above categories

For a final pre-processing step, we remove most of the data, keeping only at bats from 2014. We do this in order to keep old data from biasing the model.

2.4 Methodology

Figure 1: We setup a general framework for this problem - a program that can easily be extended to incorporate new and better models. This diagram outlines how different components of the problem were separated out to allow for this easy extendability, and it shows how the evaluation of our models is incorporated into the program.

The purpose of this section is to communicate three things to the reader. First, it demonstrates how the models are being created and scored. Second, it shows how easily the framework can be extended to incorporate new prediction models. Third, it shows some of the technical details of the work so it can be easily reproduced. Figure **??** shows what the execution flow looks like for the program we created to test our different models. The steps in the diagram are explained in more detail below:

1. Load the csv files into Data Frames.
2. Merge the data as described in section **??**. Randomly split the data into a training set and a test set. The training set is used to build the model, while the test set is used to evaluate the predictions of the model.

[2]preseason and postseason games are tracked as well, but we are mainly interested in understanding the common case of regular season games

3. Take the training set and partition it by batter_id and pitcher_id, and discard groups with less than 250 rows[3].

4. Build a model for each player (group in the partition of the training set). The actual models that we build in this step are described in section **??**.

5. Use the prediction functions to make predictions about the at bats in the test set (ignoring at bats for which the batter/pitcher doesn't have a model).

6. Use the results from the previous step to score each prediction function.

In addition to the diagram above, it is also important to understand what the prediction functions do. A prediction function consumes an at bat, where an at bat is a heterogeneous array that contains information about the at bat that is known before the outcome is known. A prediction function uses the batter_id and pitcher_id field to fetch statistics relevant to the players involved in the at bat, and uses the other features in the at bat to skew the statistics based on the situation at hand. The prediction function produces a 3-element vector $[P(Hit), P(Out), P(Walk)]$ that is the estimated distribution of outcomes for that particular at bat.

3 The Mathematical Models

We explored this problem from many different angles, and we ultimately combine ideas from each of these approaches into one big mathematical model. In order to not overwhelm the reader, we separate these approaches into different sections, adding a layer of complexity to each successive section.

3.1 Establishing A Baseline

In order to understand and evaluate our mathematical models, we first have to develop some baseline models to compare against. There are three natural choices for baseline models which are outlined below. Each of these models are constructed only using the at bats and their outcomes – they ignore situational statistics such as game location, time, and handedness of the players.

3.1.1 Baseline 1

This model approximates the outcome probabilities from the observed data for the batter only. The outcome probabilities are approximated as follows:

$$\begin{aligned} P(Hit) &= \frac{Hits}{AtBats} \\ P(Out) &= \frac{Outs}{AtBats} \\ P(Walk) &= \frac{Walks}{AtBats} \end{aligned} \tag{1}$$

While $P(Hit), P(Out)$, and $P(Walk)$ are actually unknown, the law of large numbers tells us our rational approximations converge to the true probabilities as the number of at bats becomes large. An everyday player typically has 500 - 600 at bats over the course of a season, so we can be reasonably confident in this approximation for most batters.

[3]We do this because our models cannot make reliable predictions without enough data

3.1.2 Baseline 2

Similar to the first baseline model, this model approximates the outcome probabilities of a particular at bat by looking at all the at bats associated with the pitcher. Since this model ignores the batter, it will work best when the batter is average.

3.1.3 Baseline 3

The first baseline model ignores the pitcher while the second baseline model ignores the batter. The next natural step to take is to use a formula that can combine the two predictions so that we can get a match-up formula. Unfortunately, there is no magic formula that solves this type of problem, but approximations do exist that have some nice properties and are easy to use.

One such approximation is called the log5 formula (**?**), which is a formula invented by Bill James to address the question of approximating the probability that team A would beat team B based on their respective winning percentages. This formula can be extended to approximate the probability that a batter B will get a hit against a pitcher P. In it's extended form, the log5 formula states:

$$\begin{aligned} P_{BvP}(Hit) &\propto \frac{P_B(Hit)P_P(Hit)}{P_A(Hit)} \\ P_{BvP}(Out) &\propto \frac{P_B(Out)P_P(Out)}{P_A(Out)} \\ P_{BvP}(Walk) &\propto \frac{P_B(Walk)P_P(Walk)}{P_A(Walk)} \end{aligned} \tag{2}$$

where $P_{BvP}(Hit)$ denotes the probability that a batter B will get a hit against a pitcher P, $P_B(Hit)$ is the probabilities that the batter B gets a hit (against a random pitcher), $P_P(Hit)$ is the probability that the pitcher P gets a hit (against a random batter), and $P_A(Hit)$ is the probability that an *average* batter A gets a hit against an average pitcher. These probabilities are approximated using formula **??**.

In general, the approximated probabilities need to be normalized as they are not guaranteed to sum to 1. In addition to their simplicity, these formulas have a number of intuitive properties that are easy to verify:

1. $P_B(X) = P_A(X) \rightarrow P_{BvP}(X) = P_P(X)$
2. $P_B(X) > P_A(X) \rightarrow P_{BvP}(X) > P_P(X)$
3. $P_B(X) < P_A(X) \rightarrow P_{BvP}(X) < P_P(X)$
4. The same properties hold for $P_{BvP}(X)$ in relation to $P_B(X)$.

While this formula has some nice properties, and it has been shown to work remarkably well in practice (**?**), it relies on a few assumptions. Namely that the pitchers that the batter has faced have been about average (overall, not individually), and that he batters that the pitcher has faced have been about average as well. If there was a batter who mainly faced elite pitchers, then his statistics aren't likely to be reflect how well he would have done against an average pitcher, so the log5 formula would not provide a good approximation. When there is sufficient amount of data, this is an unlikely occurrence so it's a reasonable assumption to make.

3.2 Machine Learning on Situational Statistics

Now that we have established some baseline models, we can construct models that take into account more information about the situation at hand. Every at bat in our data set contains the date, time, and location of the game associated with the at bat, as well as the batter, pitcher, and their dominant hands. Each of these features can significantly skew the distribution of the at bat outcomes.

For example, some ballparks are considered hitter friendly while other ballparks are considered pitcher friendly based on the shape and dimensions of the ballpark, so the location of the game is an important factor to take into account. Additionally, it is generally well known within that lefty-lefty and righty-righty match-ups favor the pitcher while lefty-righty and righty-lefty match-ups favor the batter.

We used machine learning to automatically find the correlations between these features and at bat outcomes. We built each of the machine learning models for every player, and they were constructed using 4 input features: game location (home or away), game time (day or night), batter stance (left or right), and pitcher stance (left or right). Since our goal is to predict outcome probabilities (as opposed to concrete classifications), we decided to explore two machine learning models: Random Forests and Naive Bayes as these algorithms can naturally produce probabilistic estimates.

To predict the outcome of the at bat, we queried the machine learning models for both the batter and the pitcher then combined the predictions using formula **??**.

3.2.1 Random Forests

A Decision Tree classifier is one of the most well known machine learning algorithms. It works by recursively partitioning the training data into subsets based on the information gain heuristic [4]. A Random Forest classifier is a collection of decision tree classifiers, where each decision tree is assembled from a different subset of the training data. The individual trees in the forest then use a voting mechanism to combine the individual predictions. While random forests and decision trees are usually used as concrete classifiers, they can also be used as a probabilistic classifier. Random forests are generally considered superior models to decision trees although that is not always the case.

3.2.2 Naive Bayes

A Naive Bayes classifier is a probabilistic classifier based on Bayes' Theorem. The Naive Bayes algorithm relies on the assumption that the input features are conditionally independent given the output feature (hit, out, or walk). While it's difficult to know if this assumption holds in general, intuitively it makes sense that our input features would be independent or nearly independent. Further, Naive Bayes is generally considered a decent model even if the conditional independence assumption is not satisfied.

$$\begin{aligned} P(Hit|A_1, \dots A_n) &\propto P(A_1|Hit) \cdots P(A_n|Hit)P(Hit) \\ P(Out|A_1, \dots A_n) &\propto P(A_1|Out) \cdots P(A_n|Out)P(Out) \\ P(Walk|A_1, \dots A_n) &\propto P(A_1|Walk) \cdots P(A_n|Walk)P(Walk) \end{aligned} \tag{3}$$

[4] information gain is a heuristic that attempts to quantify the quality of a split by looking at the amount of randomness present before and after the split

where A_i represents one of the input features (location, time, batter stance, pitcher stance), and the conditional probabilities are approximated from the training data.

3.2.3 Weighted Decision Trees

In the scikit-learn implementation of decision trees, the predicted class probability is given as the fraction of samples of the same class in a leaf (**?**). This has huge problems when the number of samples that reach a particular leaf is small. For example, as shown in Figure **??**, there are only 21 at bats for Ian Kinsler against left handed pitchers during a home day game. In those at bats, he had 7 hits, 14 outs, and 0 walks. Thus, scikit-learn would report $P(Hit) = 0.333$. However, it's unlikely that Ian Kinsler would be able to keep up this excellent performance because in other similar situations the proportion of hits was much lower. For other players, the amount of data in this bin might be even smaller than 21, so we need to come up with a generic solution that will work for all players.

Smoothing is the natural solution to this problem. Smoothing is a heuristic that trades bias for precision – we incorporate less relevant information in order to have some sense of consistency in the probability estimates across different samples. Given a probability vector p constructed from n samples and less relevant probability vector q (the parent node in the decision tree) constructed from a larger set of samples, the smoothed probability vector is calculated with the formula $p' = \frac{n \cdot p + k \cdot q}{n+k}$ where k is the smoothing constant, which we chose to be $k = 50$. This formula is recursively applied up the decision tree. Smoothing has been shown to substantially increase the accuracy of probabilistic predictions (**?**). Tables **??** demonstrates how the situational probability estimates would change after smoothing has been applied.

			Unsmoothed				Smoothed		
Hand	Loc	Time	Hit	Out	Walk	Total AtBats	Hit	Out	Walk
left	away	day	0.25	0.63	0.13	32	0.26	0.67	0.07
		night	0.26	0.74	0.00	39	0.24	0.72	0.04
	home	day	0.33	0.67	0.00	21	0.28	0.69	0.03
		night	0.22	0.73	0.05	41	0.26	0.59	0.05
right	away	day	0.26	0.69	0.05	85	0.26	0.69	0.05
		night	0.20	0.73	0.07	110	0.22	0.71	0.06
	home	day	0.26	0.73	0.02	62	0.27	0.70	0.03
		night	0.31	0.64	0.05	150	0.30	0.65	0.05

Table 5: Tabular representations of an unsmoothed/smoothed decision tree for Ian Kinsler. Note how the rows with less at bats are skewed more than the rows with more at bats after smoothing has been applied.

Note that this smoothing heuristic simultaneously solves two problems. First, it ensures that none of the probabilities are so far away from the global average that they are obviously incorrect. Second, it naturally fills in missing data for situations that are possible but no sample data exists.

3.3 Markov Models

All of the models we've implemented up to this point have been completely based on the at bat data, and there is a lot of data on pitches that are currently not being utilized. By looking at pitch data, we can get a better sense of the type of player a specific batter/pitcher is. For example, by

looking at the pitch data, we can get a sense of what pitches the pitcher likes the pitcher likes to throw the most, where the pitcher likes to throw those pitches, and how the batter has done against similar pitches in the past.

Now that we've established some motivation for incorporating pitch data, we will show that we can reduce the task of predicting at bat outcomes to predicting pitch outcomes by thinking of an at bat as a Markov process. Markov processes can be used to model systems where there are a set of states, and a set of events that cause transitions from one state to another. We can think of an at bat as having three absorbing states and 12 transient states [5], where the absorbing states represent the three possible outcomes of the at bat: hit, out, and walk, and the transient states capture the information that can change over the course of an at bat, which is just the current count [6]: $(0,0),(0,1),\ldots,(3,2)$. The result of every pitch can be categorized as one of six distinct events: ball, strike, foul, hit, out, or walk (hit by pitch). Figure **??** and Figure **??** demonstrate how these events cause transitions in the Markov chain using color coding in both a flow chart and a table of probabilities.

Figure 2: Each gray circle is a transient state. Each coordinate in the grid represents the current count of balls and strikes. The colored terminal states represent the outcome of the at bat (Hit, Walk, or Out). You can transition between transient states by moving down or to the right, representing a ball or a strike/foul respectively. You can also transition from any transient state to any terminal state, but these connections are not explicitly drawn.

Figure 3: This figure is a tabular representation of figure **??**. Each coordinate in the table represents the current count, and the letters in each cell of the table correspond to the events that would result a transition from one state to another, where S=Strike, F=Foul, B=Ball, X=Out, H=Hit, and P=Hit By Pitch.

With accurate estimates for the probabilities of these pitch events and a little bit of linear algebra, we can accurately get the absorbing state probabilities (outcomes of the at bat), which is ultimately what we are interested in predicting (**?**). In general, these probabilities will depend on batter, the pitcher, and other game factors, and they will have to be approximated using the sample data. The next few sections describe how we can fully utilize the data available to calibrate probabilities specific to the situation at hand. The methods below build from each other, with each new model increasing in complexity by utilizing additional information about the pitches being thrown.

3.3.1 Simple Markov Model

Now that the Markov model framework has been set up, we can focus on accurately estimating the transition probabilities. To do this, we have to look into the pitch data. Every pitch has access to a handful of stats, including the batter, the pitcher, situational game information, the current count when the pitch was thrown, the type of pitch thrown, the location of the pitch (the x and y location where the pitch crossed the place perpendicular to the pitcher and the front of home plate), and the outcome of the pitch (ball, strike, foul, hit, out, walk). We are ultimately interested

[5] Once a absorbing state is reached, it cannot be left. A transient state is any state that is not a absorbing state

[6] The count is (b,s) where b is the number of balls and s is the number of strikes

in incorporating all this information into our model, but to keep things relatively simple and to avoid over-fitting, we will start off with a few simplifying assumptions by only incorporating the current count.

For a particular player, we want to look into the sample observations to approximate the distribution of outcomes for a particular count. However, if we partition our pitches into twelve distinct classes based on the current count, we run into the same problem that we had earlier – some bins don't have enough data. To fix this, we refer back to the idea of smoothing introduced in section **??**. When there is enough data in a particular bin, the probabilities that we report will be very close to the observed proportion, but when there is not enough data, we will rely on less relevant data.

		Proportions							Smoothed Proportions					
Balls	Strikes	Ball	Foul	Hit	Out	Strike	Walk	Total Pitches	Ball	Foul	Hit	Out	Strike	Walk
0	0	0.35	0.09	0.04	0.09	0.42	0.00	562	0.35	0.10	0.05	0.09	0.41	0.00
	1	0.44	0.15	0.07	0.18	0.15	0.00	298	0.42	0.16	0.08	0.18	0.16	0.00
	2	0.41	0.23	0.08	0.17	0.10	0.00	124	0.37	0.24	0.08	0.19	0.12	0.00
1	0	0.31	0.20	0.10	0.16	0.24	0.00	196	0.32	0.18	0.09	0.15	0.27	0.00
	1	0.31	0.25	0.09	0.18	0.17	0.00	205	0.32	0.24	0.09	0.18	0.17	0.00
	2	0.31	0.22	0.12	0.23	0.12	0.01	173	0.30	0.23	0.11	0.23	0.13	0.01
2	0	0.35	0.07	0.06	0.15	0.37	0.00	54	0.35	0.10	0.06	0.13	0.37	0.00
	1	0.25	0.24	0.13	0.16	0.22	0.00	87	0.29	0.22	0.11	0.17	0.20	0.00
	2	0.18	0.34	0.09	0.27	0.12	0.01	141	0.21	0.31	0.09	0.25	0.13	0.01
3	0	0.37	0.00	0.00	0.00	0.53	0.00	19	0.35	0.09	0.04	0.08	0.44	0.00
	1	0.19	0.25	0.06	0.25	0.25	0.00	32	0.29	0.22	0.08	0.20	0.21	0.00
	2	0.13	0.37	0.05	0.31	0.15	0.00	62	0.21	0.32	0.07	0.26	0.14	0.00

Table 6: This table demonstrates how the pitch outcome probabilities are approximated for each count using the smoothing procedure introduced in section **??**.

At this point we have a table of probabilities for the batter and a table of probabilities for the pitcher for any count. Now what do we do with them? Recall that we want to incorporate them into the Markov chain as the transition probabilities, but we need some way to incorporate both tables. Only considering the batters table doesn't makes sense because if he were facing a strong pitcher, the probabilities should definitely be different than if he were facing an average pitcher. Similarly, it doesn't make sense to use the pitchers table alone. To combine them, we will refer back to section **??** and use the log5 formula to approximate the match-up probabilities. Note that we must also construct one of these tables for an average player[7] in order to use the formula.

3.3.2 Markov Model 2

Now that we have a decent framework for approximating the transition probabilities in the Markov chain, we can extend it to take into account information specific to the batter and the pitcher. Until now, we haven't considered player behavior. Utilizing this type of information will allow us to identify match-ups in which the batter and pitcher are "compatible". For example, if pitcher P likes to throw a lot of fastballs, and batter B is exceptionally good at hitting fastballs, then you would expect B to do well against P in general. However, if B was below average at hitting fastballs, then clearly P would have a distinct advantage.

This idea can be formalized as follows: Let X be a pitch type, and let $P(X)$ denote the probabil-

[7]the average table is calculated using all data in the data set

ity that the pitcher throws a pitch of type X in the given situation[8]. Now let $P(ball|X), P(foul|X)$, $P(hit|X)$, $P(out|X), P(strike|X), P(walk|X)$ denote the conditional probabilities of the outcomes given the pitch type. Then the following follows from the law of total probability:

$$P(ball) = \sum_{X \in pitch\_types} P(X)P(ball|X) \tag{4}$$

where $pitch\_types = \{FF, SL, FT, SI, CH, CU, FC, KC, FS, KN\}$. We can represent these calculations for all pitch outcomes compactly as a matrix-vector product as shown in equation **??**.

$$\begin{bmatrix} P(ball|FF) & P(ball|SL) & \dots & P(ball|KN) \\ P(strike|FF) & P(strike|SL) & \dots & P(strike|KN) \\ P(foul|FF) & P(foul|SL) & \dots & P(foul|KN) \\ P(hit|FF) & P(hit|SL) & \dots & P(hit|KN) \\ P(out|FF) & P(out|SL) & \dots & P(out|KN) \\ P(walk|FF) & P(walk|SL) & \dots & P(walk|KN) \end{bmatrix} \begin{bmatrix} P(FF) \\ P(SL) \\ P(FT) \\ P(SI) \\ P(CH) \\ P(CU) \\ P(FC) \\ P(KC) \\ P(FS) \\ P(KN) \end{bmatrix} = \begin{bmatrix} P(ball) \\ P(strike) \\ P(foul) \\ P(hit) \\ P(out) \\ P(walk) \end{bmatrix} \tag{5}$$

We can get a handle on these probabilities by using the sample data for the players involved. In general, these probabilities will depend on the current state. For instance, Rick Porcello is much more likely to throw a fastball when the count is $(0, 0)$ than when the count is $(0, 2)$. To approximate the pitchers pitch distribution, we first partition the data by current count, then compute the proportions of each pitch type. In our current implementation, these probabilities are not smoothed, but it is something that is worth exploring.

We can approximate $P(outcome|X)$ for a batter in a similar fashion – filter the pitches that are relevant to the current situation (current count and pitch type), then compute the proportions of each outcome in the filtered data. These probabilities are smoothed for the same reason described in section **??**.

The astute reader may have noticed a flaw in this method – and indeed they'd be right but it is easy to address. The problem is that different pitchers often throw pitches with different levels of nastiness [9]. Theoretically there could be two pitchers, A and B that throw the exact same distribution of pitch types in every situation, where A's pitches are much nastier than B's. The current model doesn't incorporate that type of information. Again, the log5 formula comes in handy here. If we compute $P(outcome|X)$ for the batter, the pitcher, and the average player, we can apply equation **??** to get new conditional probability estimates that are based on the match-up. These formulas naturally account for the pitchers nastiness because the pitchers conditional probability density function will be different between two pitchers who have different levels of nastiness.

It should be noted that this model assumes that the pitcher chooses his pitch at random according to the distribution specified and the batter doesn't know *a priori* what the pitcher is going to throw. For future work, we would like to look into the game theory of this interaction, as in reality the pitcher might choose the pitch based on the batter he is facing, and the batter might prepare for that pitch.

[8] note that in general this could depend on many things, including the count, the handedness of the opposing batter, etc.

[9] a pitch is said to be nasty if it was executed exceptionally well, making it difficult for the batter to hit

3.3.3 Markov Model 3

There is one more piece of player-specific information from the pitch data that we can bring into account – namely pitch location. In baseball there is this concept of a heat map, which is typically thought of as a color coded square grid that describes (1) where the pitcher likes to throw the ball and (2) where the batter likes the ball to be thrown. These vague but hopefully intuitive ideas can be formalized in a similar manner as pitch type, but there is an added layer of complexity in dealing with two dimensional continuous variables.

We can start by understanding where the pitcher likes to throw the ball. To do that we should search for a function $f(x, y)$ that describes the probability density of the pitcher throwing the ball at location (x, y) over the plate[10]. The most common technique for approximating this type of function is to build a histogram of the x and y values in the data set. However, histograms have a number of issues that become especially problematic in the multivariate case (**?**). For example, when there is not enough data or the dimensionality is too high, the resulting histogram is highly dependent on the domain discretization. This is a highly undesirable property, but luckily much research has gone into this problem.

One well studied solution is called Kernel Density Estimation (KDE) (**?**). KDE is a non-parametric way to estimate the shape of an unknown probability density function. We can use KDE to understand where the pitcher likes to throw the ball, and build a smooth approximation of the function $f(x, y)$. Figure **??** shows an example of what these functions look like for Rick Porcello for different pitches in his arsenal (without considering the current count).

Figure 4: These figures show the pitch location distribution for four different pitch types for Rick Porcello. It's clear from the figures that pitch type and pitch location are not independent, as the shape of the density function is so different for each pitch type.

Now that we have a strategy to obtain functions that describe where the pitcher has thrown the ball historically, we need to come up with a model that will tell us the probability of each pitch outcome given a pitch location for the particular batter[11]. We seek to find functions $P(ball|x, y)$, $P(strike|x, y)$, $P(foul|x, y)$, $P(hit|x, y)$, $P(out|x, y)$, and $P(walk|x, y)$. While it's not immediately obvious how to model these functions [12], it turns out that they can be decomposed into components that we know how to find using KDE:

$$P(outcome|x, y) = \frac{f(x, y|outcome)P(outcome)}{f(x, y)} \tag{6}$$

where $f(x, y|outcome)$ is the distribution of the (x, y) location for pitches with the given outcome (approximated with KDE), $P(outcome)$ is the probability of the given outcome, and is approximated using the proportion of of pitches that resulted in that outcome, and $f(x, y)$ is the distribution of the (x, y) location of all pitches (regardless of outcome), and is calculated as $f(x, y) = \sum_{o \in outcomes} f(x, y|o)P(o)$ where $oucomes = \{ball, strike, foul, hit, out, walk\}$.

We are ultimately interested in using these functions to approximate $P_{BvP}(outcome)$, and we can accomplish this by using a continuous variant of formula **??** as shown below.

[10] note that this function will depend on the current count, the pitch type, and possibly other factors as well

[11] technically we actually need to figure this out for the batter, the pitcher, and the average player so we can combine the functions together with equation **??**

[12] our original idea was to discretize the domain and count occurrences of each outcome in every bin

Figure 5: These figures show the conditional PDF's $P(outcome|x, y)$ for each outcome for Ian Kinsler generated with KDE. The shapes of the distributions are pretty intuitive. For example, the probability of a ball increases as the ball moves further away from the center of the strike zone. The probability of a hit peaks in the middle of the strike zone, but it also has local maxima on the edges of the domain, which show where some of Ian Kinsler's *hot spots* are.

$$P_{BvP}(outcome) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P_{BvP}(outcome|x, y) f_P(x, y)\, dy\, dx \tag{7}$$

While the formula above uses integration bounds of $(-\infty, \infty)$, in practice the empirical data has finite bounds. As shown by the plots in figure **??**, the bounds for are approximately $20 < x < 180$ and $80 < y < 200$. This computation can be fairly time consuming, especially if these integrations must be computed for every at bat in the test set, even for efficient integration techniques like adaptive quadrature. The bottleneck of this integration is the function call to the KDE model – to address this bottleneck, we evaluate the KDE over a mesh in the model building phase, so that it only has to be done once per player, then use the trapezoid rule to approximate the value of the definite integral in the prediction phase. We chose to use a mesh size of 100 by 100 to balance the memory usage with the granularity of the mesh and the accuracy of the approximation.

Equations **??** and **??** can be combined into one equation that utilizes both pitch type and pitch location as shown below:

$$P_{BvP}(outcome) = \sum_{type \in pitch\_types} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P_{BvP}(outcome|x, y, type) f_P(x, y|type) P_P(type)\, dy\, dx \tag{8}$$

All of these terms can be easily estimated from the sample data. $P_P(type)$ is estimated by using the empirical distribution of pitch types in the sample data, $f_P(x, y|type)$ is estimated using KDE as demonstrated in figure **??**, and P_{BvP} is estimated using conditional KDE as demonstrated in figure **??**, and applying equation **??** to account for both the batter's stats and the pitcher's stats.

3.3.4 Bringing it All Together

Now that we've established ways of utilizing the pitch data information in section **??** and created algorithms to take into account other situational factors in section **??**, we would like to combine these ideas into one all-encompassing model. To achieve this, we must modify equation **??**, so that $P_{BvP}(outcome|x, y, type)$, $f_P(x, y|type)$, and $P_P(type)$ depend on the situation, which is defined by the location and time of the game, and the dominant hand of the two players involved in the at bat. This is most easily achieved by extending the work introduced in section **??**. This will give us slightly different function approximations for each situation, and the smoothing will mitigate the possibility of over-fitting.

4 Analysis

Now that we've introduced the models that we investigated, we must be able to compare them based on how well they generalize to the test set. We use two well known scoring metrics: log-loss

and brier score. The log-loss formula is used to evaluate the quality of predictions in a probabilistic classifier.

$$logloss = \frac{1}{N}\sum_{i=1}^{n} h(y_i) \tag{9}$$

where there are N rows in the test set, y_i is the actual outcome of at bat i (Hit, Out, or Walk), and $h(Hit) = -\log(P(Hit))$, $h(Out) = -\log(P(Out))$, and $h(Walk) = -\log(P(Walk))$.

The log-loss formula has two properties that are worth noting. First, the expected value of the log-loss score is minimized when the predicted probability distribution matches the actual probability distribution. Second, while the theoretical best log-loss score is indeed 0, that lower bound is unattainable unless every outcome is predicted perfectly with probability 1. Any model that does this is clearly over-fitting since the outcome of an at bat is a random variable. Thus, it is hard to derive meaning from the log-loss score directly, but since we have tested many different models, we can use the log-loss score indirectly as a ranking mechanism.

The brier score is another scoring metric that is often used to evaluate probabilistic classifiers.

$$BS = \frac{1}{N}\sum_{i=1}^{N}\sum_{o\in\{Hit,Out,Walk\}} (y_{i,o} - p_{i,o})^2 \tag{10}$$

where $y_{i,o} = 1$ if at bat i had outcome o, and $p_{i,o}$ is the estimated probability of that event from the model. The brier score can be decomposed into three components, each with a nice interpretation that provides insight into where the model does well and where it needs work. The decomposition is $BS = REL - RES + UNC$, where REL denotes the *reliability* (how close the forecast probabilities are to the true probabilities), RES denotes the *resolution* (how well the model is able to make different predictions for different at bats), and UNC denotes the *uncertainty* (how much randomness is in the test set) (**?**). We are particularly interested in the resolution component, as that tells us how well our model is able to detect at bats with probability distributions away from the mean. A good model would have a low brier score, so we want to minimize $REL - RES$ (UNC does not depend on the model, only the data in the test set).

Table **??** shows how each model compared in terms of the log-loss score and the brier score. While we did not get the results we expected, namely that Markov 1 and Markov 2 actually performed worse than the other models according the log-loss and brier-score, we still believe the framework we setup and the models we introduced are valuable contributions to the problem. The markov models were merely a way to reduce at bat predictions to pitch predictions, so they can only be as good as the underlying model that predicts the pitch outcomes. This pitch data provides player-specific insights, and we believe if is used in the right way, the markov models can achieve a significantly higher score than any model that only uses the at bat information. Another thing thing that stands out in the table of results is that the first baseline model had the best brier score, and that model completely ignores the pitcher's statistics. This indicates that the result we are seeing are either being caused by the high degree of noise in the data, or that our other models are over-fitting the data in the training set.

5 Conclusion and Future Work

While we were only able to improve upon the log5 formula by a small amount using the Naive Bayes probabilistic classifier, we believe our other ideas have some merit, even though the were

Algorithm Test Result					
Test Set	Log Loss	Brier Score	Reliability	Resolution	Uncertainty
Naive Bayes	0.80825	0.47772	0.01461	0.01488	0.47799
Baseline 3	0.80827	0.47786	0.01297	0.01310	0.47799
Baseline 1	0.80860	0.47737	0.00377	0.00435	0.47799
Weighted DTree	0.81017	0.47882	0.01740	0.01658	0.47799
Baseline 2	0.81276	0.47903	0.00417	0.00312	0.47799
Markov 1	0.81660	0.48023	0.01553	0.01327	0.47799
Markov 2	0.81662	0.48039	0.01475	0.01237	0.47799
Random Forest	∞	0.48348	0.02930	0.02383	0.47799

Table 7: This table of results shows the scores for each model. While it is difficult to derive much meaning from the raw scores, these results can be interpreted by looking at the relative scores in each category. Note in particular that the Naive Bayes algorithm performed the best in terms of the log-loss metric, while baseline 1 performed the best in terms of the brier score. Also note that the Weighted DTree attained the best resolution out of all the models with reliability < 0.02

outperformed by the baseline models. One key takeaway is that there is a high degree of uncertainty in the data, so constructing a model that can make accurately calibrated predictions is really challenging.

While we were ultimately unable to achieve the goals that we set out to achieve, our framework can be used as a platform to build better models. There are many avenues for future work that can be explored to improve our models. One such path is to explore alternatives to the weighted decision tree algorithm. All of the markov models rely heavily on that being a good heuristic, but it turned out to hurt the quality of the predictions more than it helped, so replacing smoothed probability estimates with some other heuristic could improve the quality of the predictions. Additionally, optimizing the smoothing parameter is worth exploring as well. Another avenue for future work is to come up with an alternative to the log5 formula for combining batter and pitcher predictions. One possible approach to explore is called logistic regression, which can be used to correlate continuous variables with probability estimates of classifications (**?**). In our case, the independent variables are the probability estimates for each outcome of the at bat for both the batter and the pitcher, and the dependent variable is the probability estimate of the at bat interaction between the batter and the pitcher. If this method improves the quality of the predictions from the log5 method, then all of the models can be improved as a result by replacing the call to the log5 formula with a call to the logit formula. Finally, we want to explore better testing methods such as 10-fold cross validation to give us a higher degree of confidence in the reproducibility of our work. Computational limitations prevented us from doing that in this work, but using a cross validation procedure would give us more confidence in the results, and reduce the variations due to noise in the test set.

6 Acknowledgements

We would like to thank all the people who helped make this work possible, including Dr. Louis Rossi for continuously providing us with feedback on our work, Travis Johnston and the GCLab for contributing ideas to make this work high quality, and Jacek Cencek for brainstorming heuristics and verifying theoretical soundness of our methods. Additionally, we would like to acknowledge the pandas framework, for greatly simplifying our data manipulation and analysis, and scikit-learn for

providing an interface for machine learning methods, allowing us to abstract those predictions away from our program.