Using Machine Learning to Describe How Players Impact the Game in the MLB
Connor Heaton & Prasenjit Mitra
(czh5372|pum10)@psu.edu
Baseball
N94F9VL0W

Abstract
Although common across all major sports in today's day and age, Major League Baseball (MLB) has the most storied history of using statistical analysis to better understand the game, with an entire discipline known as sabermetrics dedicated to the craft. At the end of the day, all of the methods of analysis across sports try to describe some aspect of how an individual player or team impacts the game. In general, an aspect of the game humans deem “important” is discretized, and players or teams are evaluated and described based on the number of times they induce said discrete events, or in some cases keep it from happening. In doing so, the meaning of each discrete event is set in stone - a single is a single regardless of how it happens - and players are described based on the rate at which these events occur while they are in the game. We contend it would be more advantageous to work in the reverse order - learn the meaning of in-game events based on the impact that they have on the game and context in which they occur, then describe how players impact the game with this new understanding in mind. To this end, we draw on recent work in Natural Language Processing (NLP) and Computer Vision (CV) and propose a novel method of describing player form in the MLB - how players impact the game over a short period of time. Concretely, a player’s form is described by a 64-element vector. We show how these vectors can be used to describe players over the short- and long-term, and contain signal useful for predicting the outcome of games.

1. Introduction
As the sport of baseball grew in popularity, fans, players, and managers desired a more pointed way of discussing, and arguing over, the game. The more mathematically inclined fans realized that they could use statistics to describe how players have historically performed, and how those performances have translated in to advantages for their team - and thus, sabermetrics was born1.

While sabermetrics has undoubtedly changed how players, fans, and front offices alike interact with the game - introducing new statistics, such as Wins Above Replacement (WAR)2 and Batting Average on Balls In Play (BABIP)3 among others, and inspiring new strategies such as the infield shift - such statistics are fundamentally limited by the fact that they are largely derived from simple counting statistics. BABIP, for example, simply counts the number of balls put in play while a pitcher was on the mound and the number of times the batter reached base in said situations, then divides the latter by the former. WAR relates the rate at which players induce different events (walk, single, home-run, etc.) to a “replacement level” player, adjusting for position, but the principal inputs are simple counting statistics.

1 A somewhat simplified history
2 https://www.mlb.com/glossary/advanced-stats/wins-above-replacement
3 https://www.mlb.com/glossary/advanced-stats/babip
To demonstrate why traditional counting statistics are insufficient to describe a player’s impact on the game, consider the scenario in which player X recorded a single in his last plate appearance. Player X could have hit a dribbler down the third base line, advanced a runner from first to second and beat the throw to first or hit a ball to deep left-field and reached first base comfortably, but didn’t have the speed to push for a double. Describing both situations as resulting in “a single” is accurate, but does not tell the whole story.

In 2015, Statcast systems were added to all 30 MLB stadiums [1]. These systems record highly detailed information about many aspects of the game including player positioning in the field, base occupancy, thrown pitch type, pitch velocity, pitch rotation, hit distance, batted ball exit velocity, and launch angle, among others, for every pitch thrown in every game. Analysis of this Statcast data has already influenced how isolated events are understood, leading to a better understanding of which launch angles are more conducive to home runs, for example, but we believe new insights can be found by analyzing Statcast data as a sequence of records in context instead.

To this end, we draw upon recent work in Natural Language Processing (NLP) and Computer Vision (CV) and present a novel method for describing player form in the MLB - that is, their influence on the game over the short term. We demonstrate how these form embeddings, which are a type of representation capturing a player’s in-game influence, can effectively be used to describe a player in the short term, such as the span of 15 plate appearances (PAs), as well as over longer time periods, such as the player’s career. Furthermore, we demonstrate how the form embeddings can be combined with traditional sabermetrics to predict the winner of a game with over 59% accuracy. We make the code and data used in this project publicly available on Github4.

2. Related Work

For an extended description of the many sabermetrics, we direct the interested reader toward Understanding Sabermetrics by Costa, Huber, and Saccoman [2]. Below, we describe similar work towards describing players across sports and related work in machine learning (ML).

2.1. Player Embeddings

Put simply, an embedding is a set of numbers that describe something. Embeddings are a popular way of describing the semantics of characters, words, sentences, and documents when working with language, for example. We desire embeddings that capture players’ form, so we describe methods to construct different types of player embeddings in sports analytics below.

2.1.1. (batter|pitcher)2vec

The (batter|pitcher)2vec model was proposed in 2018, motivated by recent advances in natural language processing (NLP) [3]. Player embeddings were learned by modeling at-bats - Given IDs for the batter and pitcher taking part in an at-bat, the model was asked to learn embeddings that describe each player and can be used to predict the result of said at-bat. The model was trained using MLB at-bats from 2013 through 2016.

Once an embedding was learned for each player, the authors analyzed the embeddings through visualization and prediction probing – that is, using the learned embeddings to predict the result of the at-bat. To visualize the embeddings, they performed principal component analysis (PCA) to

4 https://github.com/c-heat16/learning_player_form
project the embeddings to a 2D plane, and players were colored according to handedness, single rate, and homerun rate. (Figure 1). The embeddings appear to capture meaningful signal about players, predominantly motivated by player handedness. Furthermore, the authors compared the ability to predict the outcome of an at-bat using traditional sabermetrics and their embeddings, finding that their embeddings lead to less error. While useful, we question the assumption that the same player can or should be described the same way in all situations.

Figure 1: Visualization of embeddings learned in (batter|pitcher)2vec. In the left-most figure describing handedness, red denotes left-handed players, green right-handed, and blue switch hitters. The middle figure describes rate of singles per PA and the right figure homeruns per PA. In both cases, a darker color denotes a higher rate. We observe handedness seems to be the primary factor differentiating players and associations form for single- and homerun-rate.

2.1.2. Learning Agent Representations (for Ice Hockey)
Liu et al use a Markov Game Model to model games in the National Hockey League [4]. The game is converted into discrete time steps and each time step is described by a tuple \((\text{plt}, s_t, a_t, r_t)\), where \(\text{plt}\) is the “on-puck” player, \(s_t\) is the gamestate, \(a_t\) is the action taken, and \(r_t\) is the reward earned at time step \(t\). The reward is a binary value indicating if a goal was scored at that time step. An RNN is used to process the sequence of tuples (without \(\text{plt}\)), emitting an embedding at each time step that is then used to predict the “on-puck” player.

Once trained, five random games in their test set were processed and the resulting embeddings are presented in figure 2. We see that position appears to be a significant way in which players are differentiated (leftmost plot in Figure 2), and while there is certainly some overlap, clusters of individual players appear. Again, we question the assumption that the same player can or should be described the same way in different contexts at different points in time.

2.2. Player & Action Valuation
Recent works have aimed to evaluate in-game actions by players in a variety of sports. One approach involves training a model to predict the probability that a team scores given the occurrence of some discrete event, and evaluating how events change the likelihood of scoring [5]. If some action a player took increased their team’s chance of winning, the action has a positive value; if it decreased the likelihood they score, it has a negative value. Individual players can be evaluated based on the total change in probability of their team scoring resulting from actions they have taken. The authors used this methodology to identify top-tier players in a variety of European soccer leagues.

Another way to approach player and action valuation is through the lens of reinforcement learning (RL) and the learning of a Q-function. In general, the Q-function estimates the likelihood that some outcome will happen given a game state and action to be taken [6]. Liu et al demonstrate how this approach can be applied to games in the National Hockey League and European soccer leagues, introducing a new metric known as Goal Impact Metric (GIM) [6, 7]. In both cases, an LSTM is used to process a sequence of events, emitting an embedding at each time step, and an MLP is used to predict the likelihood of each team scoring in the next time steps. The authors demonstrate how individual players can be evaluated based on the summation of the impact they had on their team scoring, which they term GIM.

While these valuation methods can be used to describe the value of a player’s actions, they do so with a very narrow aim. That is, they only describe the value with respect to the scoring of a goal. While that is a significant way in which players can impact the game, it is not the only way. For example, GIM does not capture the defensive contributions of a player, or his ability to complete passes successfully.

2.3. Next Gen Stats
In the mid-2010’s the NFL started their “Next Gen Stats” initiative, teaming up with Zebra Technologies, Wilson Sporting Goods, and Amazon to provide advanced performance statistics to clubs via tracking systems installed in every NFL stadium [8]. The tracking systems include up to 30 in-stadium sensors and 2-3 RFID tags in players’ shoulder pads, providing high-resolution position data. This line of work has demonstrated how data from in-game sensor suites can be leveraged to describe player performance [9]. Quarterback aggressiveness (AGG%), for example, describes the proportion of time a quarterback throws to a receiver with a defender within 1 yard. Runner efficiency (EFF) relates the number of yards earned to the real distance traveled.

This analysis is certainly useful, but it is largely the result of new sensor technology, not of the method of analysis. Data from these sensors serve as new features for downstream analysis methods, but the methods of analysis themselves still remain the same in principle. AGG%, for example, still simply divides one quantity by another, using information collected from sensors to subset the studied population of events.

2.4. Transformers, BERT, & iGPT
The transformer architecture rose in popularity thanks in large part to its use in the BERT language model [10, 11]. The motivating principle behind BERT is the notion that the meaning of words can be inferred by analyzing the context in which they naturally appear, and the transformer architecture, along with a special training regimen, enable BERT to do just that.

To learn the language, BERT browses the internet and performs two tasks when it comes across a piece of text: 1) Masked Language Modeling (MLM) and 2) Next Sentence Prediction (NSP). MLM is
essentially BERT creating fill-in-the-blank questions for itself. For example, if BERT comes across the text “I love you,” it may create a fill-in-the-blank question in the form of “I love ____.” By analyzing the context surrounding the blank, the model is likely to fill the blank with “you.” Instead, if the fill-in-the-blank question were “I go to the gym every day. I love ____,” the context may induce the model to fill the blank with “exercise.” By learning to fill in the blank correctly, BERT is learning to infer from the context the meaning associated with different words.

The NSP task helps BERT learn the emergent meaning associated with various sequences of characters. Given two sentences, the model is asked to make a binary prediction as to whether or not these sentences appeared next to each other “in the wild.” For example, if the example above was separated and given to the model as two sentences in “I go to the gym every day” and “I love exercise,” the model would be expected to respond affirmatively. By repeatedly performing this test, the model will begin to understand the semantics emerging from the sequence of words and characters - if you do something every day, you likely “love” it, and the “gym” is associated with “exercise,” for example.

Upon seeing the success BERT and similar models had working with natural language, Chen et al. noted that their MLM training objective closely resembled that of Denoising Autoencoders, which were originally designed to work with images, and explored the extent to which transformers could be used to learn image representations using a similar training scheme [12]. Instead of learning to “fill in the blank” as BERT would, this model, dubbed Image-GPT, would learn to impute the missing pixels in a corrupted image. In much the same fashion that BERT understands language as a collection of characters and words and learns their meaning by analyzing the context in which they appear, Image-GPT perceives images as a collection of pixels with varying intensities of red, green, and blue, and learns the role they play in the emergent semantics of the image by analyzing the context in which they appear.

BERT and Image-GPT demonstrate that when paired with an appropriate training objective, transformers can be effective learners of atomic-element representations by leveraging the context in which these atomic-elements appear. Here, we use atomic-element to refer to the lowest unit of information the model is capable of expressing or understanding - for BERT, groups of English characters are the atomic-elements, while pixel values are the atomic-elements for Image-GPT. Furthermore, they demonstrate how the same model that learned these atomic-element representations also learns how to discern an emergent meaning when these representations are viewed in conjunction with one another.

2.5. Self-Supervised Learning
Self-supervised learning describes a family of methods for learning from unlabeled data. A particular flavor, contrastive learning, is very popular in Computer Vision (CV) and encourages the model to minimize a contrastive loss objective among a batch of sample records [13]. The motivating theory behind contrastive learning is that similar inputs should result in similar outputs from a representation-learning model. When learning via self-supervised contrastive loss, for example, the model is given two different views of a single image, and encouraged to produce similar outputs [14]. Furthermore, the output produced by two views of the same image are expected to be dissimilar to outputs resulting from views of different source images. The different views are often obtained by a combination of randomly cropping, rotating, resizing, inverting, or otherwise distorting the source image.
The self-supervised contrastive loss objective is given in equation 1, where \( I \equiv \{0, 1, \ldots, 2N - 1\} \) is the set of indices in the batch, \( j(i) \) is the positive sample(s) for record \( i \), \( \cdot \) is the inner product, \( \tau \in \mathbb{R}^+ \) is a scaling temperature value, and \( a \in A(i) \) is the set including the positive and negative samples for record \( i \) [15]. Record \( j \) is a considered positive sample for record \( i \) if it is derived from the same source image, otherwise it is considered a negative sample.

3. Our Method

The principal motivation behind our work towards describing player form is very much the same as that of contrastive learning - players who impact the game in similar ways should be described using similar form vectors. We do not have ground truth player form labels (vectors), but we do know a series of plate appearances for the same batter at two very close points in time should impact the game in a similar fashion, and thus, be described similarly by our model. In the sections that follow, we describe how data was collected, our player form model was trained, and the embeddings produced by our model can be used for different purposes, both descriptive and predictive.

3.1. Data Collection & Organization

While data was originally collected by the Statcast system, we use the Python package pybaseball\(^5\) to collect data used for our study and populate a local sqlite\(^6\) database. We collected two types of data using this package: 1) pitch-by-pitch data and 2) season-by-season statistics. Pitch-by-pitch data was collected for the 2015 through 2019 seasons, and contains information such as pitch type, batted ball exit velocity, and launch angle among others. Season-by-season statistics were collected for the 1995 season through 2019, and contain position-agnostic information such as WAR and age in addition to position-specific information such as WHIP for pitchers and OPS for batters.

\[
L_{\text{self}} = \sum_{i \in I} L_{i}^{\text{self}} = -\sum_{i \in I} \log \left( \frac{\exp \left( z_i \cdot z_{j(i)} / \tau \right)}{\sum_{a \in A(i)} \exp \left( z_i \cdot z_a / \tau \right)} \right)
\]

Equation 1: Contrastive learning loss equation

Each record in our pitch-by-pitch table is accompanied by three key values: 1) \textit{game.pk}, 2) \textit{AB.number}, and 3) \textit{pitch.number}. The \textit{game.pk} is a unique value associated with each game played in the MLB. Within each game, each at-bat has a corresponding \textit{AB.number}, and each pitch thrown in an at-bat an associated \textit{pitch.number}. By using these three pieces of information, we can completely reconstruct the sequence of events, which constitute an MLB game. A summary of our collected dataset is given in table 1.

| & # Games & 12,147 & # PA & 925k & # Pitches & 3.6M & # Batters & 1,893 & # Pitchers & 1,525 |
|---|---|---|---|---|---|---|---|---|---|

Table 1: Pitch-by-pitch dataset summary.

3.2. Describing In-game Events

Typically, one would use terms like \textit{single}, \textit{home run}, or \textit{strikeout} to describe the outcome of an at bat. Using this terminology to describe the outcome of at-bats to our model would be insufficient, however, as it tells an incomplete story. For example, did other runners advance on the play? For this reason, we describe the outcome of a pitch in terms of the change in the \textit{gamestate}, where the \textit{gamestate} refers to 1) ball-strike count, 2) base occupancy, 3) number of outs, and 4) score. These

\(^5\) https://github.com/jldbc/pybaseball

\(^6\) https://docs.python.org/3/library/sqlite3.html
changes in *gamestate* will constitute the vocabulary that our model will learn to understand. In total, we identify 325 possible changes in *gamestate* and the result of any thrown pitch can be described by one of these changes in *gamestate*. We will refer to these *gamestate* changes as *gamestate deltas*. From our pitch-by-pitch table, we also have information describing the thrown pitch and batted ball that induced this change in the game state, such as pitch type, pitch rotation, location over the plate, and batted ball distance and launch angle among others.

In aggregate, the *gamestate deltas* describe what happened and Statcast data illuminates how, but they do not describe who was involved. We describe the pitcher, batter, and historical matchup between the two using traditional sabermetrics and provide this as additional input to our model. We use statistics derived from four different temporal scales when describing the pitcher, batter, and matchup between the two. A summary of these supplemental features is given in table 2. When presenting this information to the model, the 1,541 supplemental features are projected to a lower dimension such that roughly half the data at each input index describes the *gamestate delta* and the other half describes the players involved in the at-bat, thrown/batted ball, and stadium.

### 3.3. Player Form Learning

We seek to describe player form - how a player has impacted the game in their recent appearances - so we must identify a *window* of activity (consecutive plate appearances) we wish to describe for each player. Once this *window* is identified, we can then create two *views* (subsets of consecutive plate appearances) of the player’s influence on the game in this *window* of activity. These two *views* describe the same player over a relatively small period of time, so they should induce similar outputs from our player *form* model. Furthermore, *views* from the same *window* of activity for the same player should be dissimilar to *views* derived from *windows* of activity of other players, and even other *windows* for the same player.

For batters, we define a *window* of activity as a sequence of 20 consecutive PAs for that batter. For each *window* of activity, we derive the first *view* as the first 15 PAs in the *window*. Then, a random offset/overlap $i$ between 1 and 5 is selected, and the second *view* is selected as PAs $i$ to $i+15$ in the window. For pitchers, we define a *window* of activity as a sequence of 75 consecutive PAs they pitched, and a *view* as 60 PAs. The first 60 PAs in the *window* serve as the first *view*, then a random offset/overlap $i$ between 1 and 15 is chosen and PAs $i$ to $i+60$ serve as the second *view*. Batters have an average 4.2 PAs per game\(^7\), and starting pitchers face an average of 23.3 batters per start\(^8\), so *view* sizes were selected such that each view covered *roughly* three to four games per player.

Present in each input sequence will also be a special `[CLS]` token, which will be used in a similar fashion as BERT’s `[CLS]` token. That is, our model will learn to process the input data such that the processed `[CLS]` embedding will sufficiently describe the entirety of the input.

---

\(^7\) [https://fivethirtyeight.com/features/relievers-have-broken-baseball-we-have-a-plan-to-fix-it](https://fivethirtyeight.com/features/relievers-have-broken-baseball-we-have-a-plan-to-fix-it)

\(^8\) [https://blogs.fangraphs.com/starting-pitcher-workloads-have-been-significantly-reduced-in-2020](https://blogs.fangraphs.com/starting-pitcher-workloads-have-been-significantly-reduced-in-2020)
Our model describes players over a short period of time, i.e., 15 at-bats, while (batter|pitcher)2vec describes players over a much larger time scale of four seasons. However, we would still be able to describe players over a much larger time-scale by viewing consecutive sequences of 15 at-bats in the aggregate, making our model much more versatile - the same model can be used to derive a description of a player over the course of 15 at bats, or four seasons.

### 3.3.1. Model Architecture

We use a multi-layer, bidirectional transformer encoder, based on the original implementation used in BERT [10, 11]. Our model consists of 8 transformer layers, 8 attention heads in each layer, and a model dimension of 512. Our model learns embeddings to describe many aspects of the input data, including gamestate deltas, stadiums, player positions, and pitch types. The remaining information at each input index is derived from a two-layer projection of the supplemental player inputs, described in section 3.2, and real-valued attributes of the thrown pitch and batted ball. Additionally, the model learns embeddings, which help position the inputs with respect to one another, such as the at-bat number within the window and the pitch number within said at-bat. Separate models are trained to describe pitcher and batter forms with no shared weights.

### 3.3.2. Training

We use two tasks to train our model: 1) Masked Gamestate Modeling (MGM) and 2) Self-supervised Contrastive Learning. The MGM task is akin to MLM, with roughly 15% of gamestate delta tokens in the input sequence masked and the model asked to impute the missing values. In addition to learning the relation between gamestate delta tokens - e.g., three consecutive balls are often followed by a fourth - the model also learns the relation between different types of batters and pitchers participating in the at-bat. For example, if the supplemental inputs describe a shutdown pitcher, poor batter, and pitcher-friendly stadium, the corresponding gamestate delta is likely to be to the pitcher’s benefit.

![Figure 3: Example of how our model processes a batch of data while learning to describe batters. Each record in the batch consists of 15 plate appearances and a special [CLS] token, and each at-bat consists of one or more pitches. The model is asked to predict the masked gamestate delta tokens using the context in which the masked token appears. Once processed by the model, the embedding for the [CLS] assumed to describe the 15 corresponding plate appearances. These [CLS] embeddings are projected to a 64-dimensional space before being used to compute self-supervised contrastive loss.](image)

The self-supervised contrastive learning task is used to train our model to induce representations that are similar for views from the same window and dissimilar for views of from different windows. Concretely, our model produces a 64-dimensional representation for each view, which is used in computing the self-supervised contrastive loss. An example of how the model processes a batch of
inputs is presented in figure 3. We use an Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and learning rate of $5\times 10^{-4}$. When learning to describe batters, our model is trained using a batch size of 256 for 75,000 iterations, with 2,000 of the iterations being warm-up, on a single GPU. A pitchers’ form is described by a larger number of at-bats, so we train our pitcher model using a batch size of 48 for 90,000 iterations, 4,000 iterations of warm-up, on two GPUs.

3.3.3. Discretizing Player Forms
We compute a form representation for all players in the starting lineup at game-start for all regular season games from 2015-2019. Then, to identify players who have impacted the game in a similar capacity at various points in time, we perform agglomerative clustering with Ward linkage on the form representations to obtain discrete form ID’s [16]. For a point of comparison, we follow a similar clustering process using traditional sabermetrics to describe players. That is, the players’ corresponding supplemental inputs, mentioned in section 3.2, without the in-game split. We perform PCA on the statistics used to describe each type of player prior to clustering.

4. Results
Our model produces embeddings that can directly be used “as-is” to describe a player over the short-term, but the embeddings can also be averaged together to produce descriptions of a player over longer time-scales, such as over the course of a few seasons. We will demonstrate both use cases in this section, as well as examples of how the produced embeddings can be used for predictive purposes.

4.1. Game-Start Player Embeddings
A natural application of our model would be to generate game-start-time player form embeddings. That is, before the start of each game, use our model to describe the N most recent plate appearances for players in the starting lineup for each team, where N=15 for batters and N=60 for pitchers. Comparing clusters of players derived from traditional sabermetrics and from form representations can give an intuition as to the information contained in the representations. We present a case study below, but more characteristics can be found in the appendix.

Figure 4: Discrete form clusters for select batters at game-start time from 2015-2019. With the exception of Mike Trout, batters seem to sporadically jump between form clusters from game to game in the stat-based plot (left). This is “smoothed out” in the form-based plots (right) while still allowing for fluctuation in the way the player is described.
Figure 4 presents a comparison of cluster membership over time for four batters: Bryce Harper, Mike Trout, Giancarlo Stanton, and Neil Walker. Harper and Trout are somewhat similar, high impact outfielders, while Stanton can tend to be more of a streaky power hitter, and Walker is perhaps more of a utility infielder. In analyzing the plot describing stat-clusters in figure 4, we see that, with the exception of Mike Trout, players sporadically bounce between clusters as their career progresses, with Neil Walker being perhaps the most sporadic. Conversely, we see a more consistent mapping in the form-cluster plot, but players can still present differently for short periods of time.

Figure 5 presents a comparison of cluster membership over time for four starting pitchers: Gerrit Cole, Alex Wood, Trevor Bauer, and Justin Verlander. Cole had rather poor performances throughout 2016 and some of 2017 before snapping back into All-Star form with Houston in 2018 and 2019. In looking at Cole’s stat-clusters in figure 5, we see that he is consistently mapped to cluster 12 from 2016 onward. This would seem to be an undesirable way to describe how he impacted the game, then, as he clearly impacted the game in very different ways in 2016 & 2017 versus 2018. Fortunately, we do not see the same behavior in looking at Cole’s form-clusters.

4.2. Long-Term Player Embeddings
A different use-case for our model could be to describe the general way in which a player impacts the game over a longer period of time, such as a few seasons. For example, we aggregate the game-start time embeddings constructed in the previous section by player, resulting in a single vector describing the player’s impact on the game from 2015 through 2019. These player-embeddings are presented in figure 6.

In general, both methods of describing players - stat-based and form-based - do reasonably well in differentiating between “good” and “bad” players. However, it seems that our embeddings do a better job in differentiating the ways in which “good” players impact the game. Consider WAR and HR per PA for batters, for example. In the stat-based visualization, the portion of the plot describing players with high WAR values is completely subsumed by the portion of the plot denoting players with seemingly high HR rates. If this is to be interpreted, it would push us towards thinking that a player must have a high HR rate to have a high WAR. In the form-based visualization, we see a
relatively small overlap among the portions of the plot denoting players with high WAR and HR rates. This leads to the interpretation that a high HR rate is one way in which a player can achieve a high WAR, but not the only way.

Figure 6: Aggregated game-start embeddings for players in the starting lineup from 2015-2019. We notice that both methods are able to differentiate “good” players from “bad” players, but the form-based embeddings provide context for the ways in which “good” players impact the game. In the sabermetric-based pitcher plots, we see no association based on breaking ball usage but clear delineation based on handedness. The form-based pitcher plots appear to take breaking ball usage into account and although handedness appears to be considered, it acts more as a supplemental feature. In the sabermetric-based batter plots, we see the portion of the plot describing batters with high WAR is subsumed by the portion describing homerun hitters, and it is difficult to notice any strong associations with batter handedness. In the form-based batter plots, however, there is only partial overlap of players with high WAR and homerun rate, and there is not significant overlap of players who bat from different sides of the plate.

4.3. Probing Predictive Power
To probe the predictive ability of our player form embeddings, we explore their influence in the ability to predict the winner of a regular season game in the MLB from 2015 through 2019. We identify three machine learning models and three types of data we will use in this study. We create train/test splits stratified by margin of victory for this section of the work.

For models, we will use random forest, logistic regression, and support vector machines (SVM). The three types of data we identified are: 1) metadata describing the two teams (win percent, runs scored, runs allowed, etc.), 2) statistical data describing the players in each team’s starting lineup.
(batting avg, WHIP, etc.), and 3) **form embeddings** emitted from our model\(^9\). We drew upon the previously published state-of-the-art work in predicting the winner of an MLB game in selecting which statistics to use.

<table>
<thead>
<tr>
<th></th>
<th>Stat+Meta</th>
<th>Form+Meta</th>
<th>Stat+Form+Meta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Forest</td>
<td>58.23% / 0.65</td>
<td>57.25% / 0.65</td>
<td><strong>59.39% / 0.66</strong></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>58.23% / 0.64</td>
<td>55.35% / <strong>0.66</strong></td>
<td>58.15% / 0.64</td>
</tr>
<tr>
<td>SVM</td>
<td>57.08% / 0.63</td>
<td>54.01% / 0.62</td>
<td>56.43% / 0.62</td>
</tr>
</tbody>
</table>

**Table 3**: Ability of various models to predict the winner of an MLB game using varying input data. Scores presented are accuracy/F1.

In general, we see that random forest models tend to perform best in our application, and the best scores are achieved when all three types of input data are utilized. Interestingly, random forest was the only architecture that saw performance improvements from the *form* features, so we explore the role they play in the model by analyzing the feature importance. It turns out that of the 20 most important features, 9 of them are *form* features. Furthermore, of the 216 total features used to fit the models, the lowest importance rank of a *form* feature is 83rd. This perhaps suggests that the *form* features serve as higher level descriptions of team-team matchup type that cannot be fully leveraged by the more rudimentary models.

### 5. Conclusions and Future Work

We believe this work serves as a strong starting point in a line of work towards a new way of describing how MLB players - and athletes in other sports - impact the course of play. Our method produces player *form* embeddings that can be viewed in isolation, as a “short-term” player description, or in the aggregate, as a “long-term” player description, and can be used to improve the efficacy in predicting the winner of a game in the MLB. Plots comparing our form-embeddings and sabermetric-embeddings demonstrate how both methods can be used to differentiate “good” players from the “bad” ones, but our embeddings provide more detail as to how the “good” players impact the game – the propensity of a pitcher to throw a breaking ball or the handedness of a batter, for example.

Moving forward, we would like to explore different ways in which contextual sports analytics can be leveraged from both a sport and computational perspective. From the sporting perspective, we want to explore which different types of phenomena can be analyzed - how can this methodology be used to model how events *within game* relate to one another? How can we adapt the approach to analyze/describe team-personnel such as managers and offensive/defensive coordinators? From a computational side, we would like to explore the efficacy of the “pre-train - fine-tune” paradigm in our application. We have shown how our model can be used to describe *players* in both short- and long-term scales. However, we are curious if we could pre-train a general baseball model that could be fine-tuned for a variety of tasks such as player description, performance prediction, and manager evaluation, among others.

---

\(^9\) The five first principal components of form embeddings for each player were used in place of the full 64-element vector.
References


Appendix
A. Form Characteristics

Here we present figures describing the impact that players tend to have – the events they induce and the manner in which they induce them – while they are in a certain form.

a. Batters
b. Pitchers
B. Time in form

Here we present figures describing the number of consecutive games for which players were described the same way (with the same form ID). This can help give intuition as to which forms are longer lasting, and which are more short term. In general, we see that the average number of consecutive games in each form is relatively low and not constant across forms. This is encouraging as we do not assume that players have a constant level of performance over time. Decreasing the window and view size will likely lead to shorter time in form, and increasing the size a longer time.

a. Batters

![Average Number of Games in Each Form for Batters](image)

b. Pitchers

![Average Number of Games in Each Form for Pitchers](image)

C. Number of Unique players in each form

Here we present figures describing the number of unique players to ever be described by each of the discrete form IDs. This can help give intuition as to which forms are rare, and which are rather common and can be used to describe many players. When 75 discrete
batter forms are used, for example, we see that form ID 5, the form ID that describes Mike Trout for almost the entire period analyzed, is only ever used to describe 51 of the 1,893 batters (2.7%) included in this study.

a. Batters

![Bar chart showing the number of unique players ever in each form for batters.]

b. Pitchers

![Bar chart showing the number of unique players ever in each form for pitchers.]

D. Transitions Between Forms

Here we present figures describing the transitions between discrete form IDs at game-start time as presented in section 4.1. Transition matrices are presented for pitchers and batters at different time deltas. This can help provide context for the meaning associated with each cluster, helping to answer the question “If player X is in form $y$ before the start of this game, what form will they be in $n$ games from now?” We present figures describing transitions for varying time periods. For each figure, the “source form” is along the y-axis (form before game $i$) and the “target form” along the x-axis (form before game $i+\text{offset}$). In general, we see that the further in to the future we look, the less likely the player is to be in the same form.
a. Batters

Here, we present figures describing form transitions 1, 10, and 25 games ahead.

i. 1 game ahead
ii. 10 games ahead
iii. 25 games ahead
b. Pitchers

For pitchers, we present figures describing how players transition between forms 1, 5, and 15 games (starts) ahead.

i. 1 game ahead

![Diagram showing form transitions 1 game ahead]

ii. 5 games ahead

![Diagram showing form transitions 5 game ahead]
iii. 15 games ahead