Learning to Describe Player Form in the MLB

Connor Heaton and Prasenjit Mitra

The Pennsylvania State University, State College, PA 16802, USA
{czh5372,pmitra}@psu.edu

Abstract. Major League Baseball (MLB) has a storied history of using statistics to better understand and discuss the game of baseball, with an entire discipline of statistics dedicated to the craft, known as *sabermetrics*. At their core, all *sabermetrics* seek to quantify some aspect of the game, often a *specific* aspect of a player's skill set - such as a batter's ability to drive in runs (RBI) or a pitcher's ability to keep batters from reaching base (WHIP). While useful, such statistics are fundamentally limited by the fact that they are derived from an account of *what* happened on the field, not *how* it happened. As a first step towards alleviating this shortcoming, we present a novel, contrastive learning-based framework for describing player *form* in the MLB. We use *form* to refer to the way in which a player has impacted the course of play in their recent appearances. Concretely, a player's *form* is described by a 72-dimensional vector. By comparing clusters of players resulting from our *form representations* and those resulting from traditional *sabermetrics*, we demonstrate that our *form representations* contain information about how players impact the course of play, not present in traditional, publicly available statistics. We believe these embeddings could be utilized to predict both in-game and game-level events, such as the result of an at-bat or the winner of a game.

Keywords: Machine Learning · Player Valuation

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 born[1].

While *sabermetrics* have undoubtedly changed how players, fans, and front offices alike interact with the game - introducing new statistics, such as WAR, OPS+, SIERRA, and BABIP among others, and inspiring new strategies, such as the defensive infield shift, offensive launch angles, etc. - such statistics are fundamentally limited in their descriptive power by the fact that they are derived from an account of *what* happened on the field, not *how* it happened.

[1] A somewhat simplified history

To see why a description of *what* happened is less desirable than a description of *how* it happened, consider two at-bats: one between *pitcher A* & *batter B*, and another between *pitcher X* & *batter Y*. In the former, *pitcher A* got ahead in the count 0-2, but *batter B* battled back to a full-count, eventually hitting a ball to deep right-field and reaching first base comfortably. For the latter, *pitcher X* fell behind 2-0 in the count, before *batter Y* hit a dribbler down the third base line and beat the throw to first. The most simple way of describing the two at bats would be to say a single was recorded in both cases, which would do nothing to differentiate between the two at bats. Information could be included about how many pitches were thrown in each at-bat, but that still doesn't tell the whole story. For example, it wouldn't convey that *batter B* was able to battle back from an 0-2 count and reach base or the way in which either pitcher sequenced their pitches. Furthermore, it would not convey that *batter B* had power-hit a fly-ball to deep-right nor that *batter Y* had the speed to beat out the throw to first.

Typically, *sabermetrics* have been used to describe some aspect of a player's game over a relatively large time-scale. Intuitively, in a statistical sense, this makes sense - they are statistics derived from a sample population, and the larger the sample population, the more *accurate* the computed sample statistic will be with respect to describing the population at large. The interpretations of these *sabermetrics* often "break down" when working with a small number of samples. For example, it would not make much sense to use batting average or WHIP to describe the players participating in the at-bats mentioned above. *Pitcher A* and *pitcher B* will have a WHIP of ∞ while *batter X* and *batter Y* will have a batting average of 1.000. They aren't incorrect, however - they accurately describe *what* happened, but do little to reveal *how* it happened. For this reason, we believe it is sub-optimal to derive a description of a player's short-term performance using traditional *sabermetrics*.

In 2015, Statcast systems were added to all 30 MLB stadiums [2]. These systems record highly detailed information about many aspects of the game including player positioning in the field, thrown pitch types, 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 the game is understood, drawing more attention to batters' launch angle, for example. We believe new insights can be found by analyzing Statcast data as a sequence of records instead of records in isolation.

2 Related Work

For an extended description of the many *sabermetrics*, we direct the interested reader toward *Understanding sabermetrics* by Costa, Huber, and Saccoman. Below, we describe similar work towards obtaining player representations and related work in machine learning (ML).

2.1 (batter|pitcher)2vec

The (batter|pitcher)2vec model was proposed in 2018, motivated by recent advances in natural language processing (NLP) [1]. Player embeddings were learned by modeling at-bats - Given an ID 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 author demonstrated how they can be used to make predictions as to the result of an unseen at-bat with more accuracy than previous methods.

2.2 Transformers, BERT, & Image-GPT

The *transformer* architecture rose in popularity thanks in large part to its use in the BERT language model [9][5]. The motivating principal 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 [3]. 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.

2.3 Contrastive Learning

Contrastive Learning is a training scheme often used in Computer Vision (CV) applications where a model's training objective is to minimize a contrastive loss objective among a batch of sample records [4]. The motivating theory behind contrastive learning is that similar inputs should result in similar outputs from a representation-learning model - in our application, similar sequences of at-bats should be described with similar *form* vectors. 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 [7]. 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 $I \equiv \{0, 1, ..., 2N-1\}$ 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 [6]. 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.

$$L^{self} = \sum_{i \in I} L_i^{self} = -\sum_{i \in I} log \frac{exp(z_i \cdot z_{j(i)}/\tau)}{\sum_{a \in A(i)} exp(z_i \cdot z_a/\tau)} \tag{1}$$

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 the same batter at two very close points in time should be described similarly. In the sections that follow, we describe how data was collected, our player *form* model was trained, and discrete player forms were obtained.

3.1 Data Collection and Organization

While data was originally collected by the Statcast system, we use the Python package `pybaseball`[2] to collect data used for our study and populate a local `sqlite3` 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 2018 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 2018, 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.

Each record in our pitch-by-pitch table is accompanied by three key values- 1) *game_pk*, 2) *AB_number*, and 3) *pitch_number*. The *game_pk* is a unique value associated with each game played in the MLB. Within each game, each at-bat has a corresponding *AB_number*, and each pitch thrown in an at-bat an associated *pitch_number*. By using these three pieces of information, we can completely reconstruct the sequence of events which constitute an MLB game. In total, our collected dataset contains 2.9M pitches thrown in 9,860 games, involving 1,333 unique pitchers and 1,690 unique batters.

3.2 Describing In-game Events

Typically, one would use terms like *single*, *home run*, or *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 *gamestate*, where the *gamestate* refers to 1) ball-strike count, 2) base occupancy, 3) number of outs, and 4) score. These 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 colloquially 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 which 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 *how*, but do not describe *who* was involved. We describe the pitcher, batter, and historical matchup between the two using traditional sabermetrics.

We use statistics derived from four different temporal scales when describing the pitcher, batter, and matchup between the two: 1) career, 2) season, 3) last 15 days, and 4) this game. We use 552 values to describe the pitcher, 578 the batter, and 411 the historical matchups between the two, for a total of 1,541 supplemental features. When presenting this information to the model, the 1,541

[2] https://github.com/jldbc/pybaseball

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 appearances) we wish to describe for each player. Once this *window* is identified, we can then create two *views* (sets of consecutive 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 at-bats for that batter. Then, for each *window* of activity, we derive the first *view* as the first 15 at-bats in the *window*, and the second *view* as the final 15 at-bats in the window. For pitchers, we define a *window* of activity as a sequence of 100 consecutive at-bats for which they pitched, and a *view* as 90 at-bats. That is, the first 90 at-bats in the *window* serve as the first *view* while the final 90 at-bats serve as the second *view*. Batters have an average 4.2 plate appearances per game[3], and starting pitchers face an average of 23.3 batters per start from 2015-2018[4], so *view* sizes were selected such that each view covered *roughly* 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.

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, 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.

Model Architecture We use a multi-layer, bidirectional transformer encoder, based on the original implementation used in BERT [9][5]. 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, pitch types, and

[3] https://fivethirtyeight.com/features/relievers-have-broken-baseball-we-have-a-plan-to-fix-it/

[4] https://blogs.fangraphs.com/starting-pitcher-workloads-have-been-significantly-reduced-in-2020/

pitch locations over the plate. 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.

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 shortly 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.

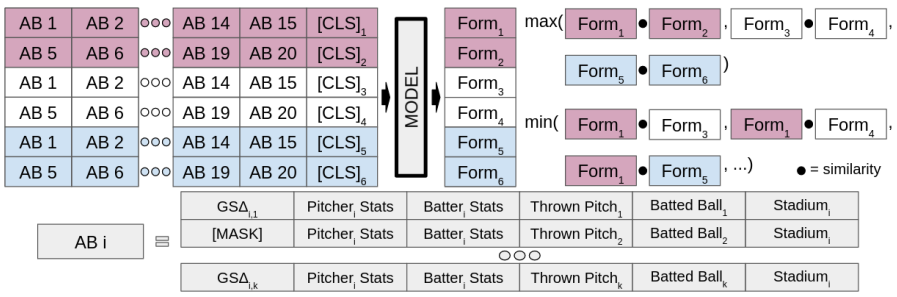

Fig. 1. Example of how our model processes a batch of data while learning to describe batters. Each record in the batch consists of 15 at-bats and a special *[CLS]* token, and each at-bat consists of one or more pitches. For each pitch, the model is given statistics describing the pitcher and batter involved, metrics describing thrown pitch type and batted ball (when applicable), and embeddings describing the stadium and the resulting *gamestate delta*. 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 at-bats. These embeddings are projected to a 72-dimension space before being used to compute the self-supervised contrastive loss.

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 72-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 1.

We use an Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and learning rate of $5e^{-4}$. When learning to describe batters, our model is trained using a batch size of 78 for 90,000 iterations, with 7,500 of the iterations being warm-up. A pitchers' *form* is described by a larger number of at-bats, so we train our pitcher model using a batch size of 36 for 35,000 iterations with 2,500 iterations of warm-up.

3.4 Discretizing Player Forms

We compute a *form* representation for all players in the starting lineup at game-start for all regular season games from 2015-2018. 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 [8]. 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 3.2, without the *in-game* split. We perform Principal Component Analysis on the statistics used to describe each type of player prior to clustering.

4 Results

While a more thorough analysis is required to better understand the representations produced by our model, comparing clusters of players derived from traditional sabermetrics and from *form representations* can give an intuition as to the information contained in the representations.

Figure 2 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 2, we see minimal overlap between Harper and Trout. This is an undesirable representation of form, as in our estimation, Harper and Trout tend to impact the game in a similar way. Conversely, we seem to notice a strong association between Harper and Trout in the plot describing their *form-clusters* in figure 2.

Figure 3 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 in to All-Star form with Houston in 2018 and 2019. In looking at Cole's *stat-clusters* in figure 3, we see that he is consistently mapped to cluster 8 from 2016 onward. This would seem to be an undesirable to describe how he impacted the game, then, as he clearly impacted the game in very different ways in 2016 & 2017 versus 2018. While at the moment we cannot say for certain what signal is contained in our *form representations* we see they describe 2016 Cole differently than 2018 Cole.

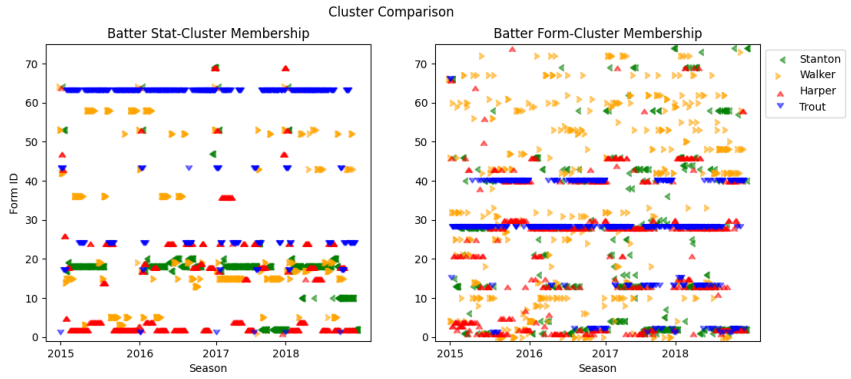

Fig. 2. Discrete batter forms at game-start for games in the 2015 through 2018.

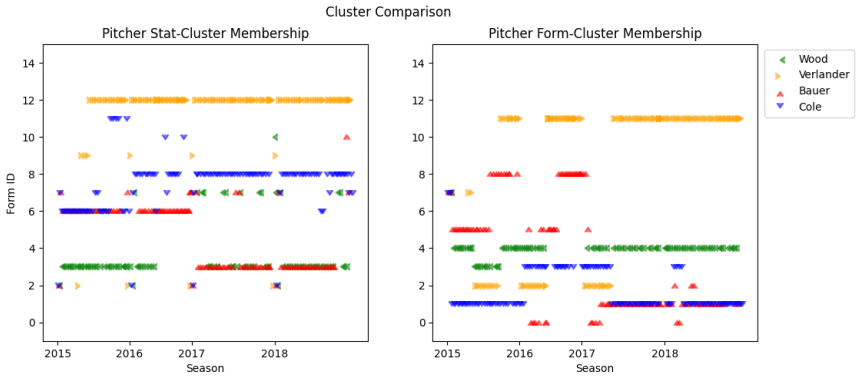

Fig. 3. Discrete pitcher forms at game-start for games in the 2015 through 2018.

5 Conclusion & 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, in a manner not contained in existing, publicly available statistics. Moving forward, we would like to gain a stronger understanding of the contents of the produced *form representations* and how they can be leveraged towards specific ends, such as predicting the result of an at-bat, the winner of an MLB game, or the occurrence of an injury. Furthermore, it would be interesting to take a closer look at the embeddings learned by our model. It would be interesting to explore the relation between different stadiums, for example, from the perspective of both the batter and pitcher.

References

1. Alcorn, M.A.: 2vec: statistic-free talent modeling with neural player embeddings. MIT Sloan Sports Analytics Conference (2016)
2. Casella, P.: Statcast primer: Baseball will never be the same. https://www.mlb.com/news/statcast-primer-baseball-will-never-be-the-same/c-119234412 (2015), [Online; accessed 20-June-2021]
3. Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., Sutskever, I.: Generative pretraining from pixels. In: International Conference on Machine Learning. pp. 1691–1703. PMLR (2020)
4. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: International conference on machine learning. pp. 1597–1607. PMLR (2020)
5. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
6. Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., Krishnan, D.: Supervised contrastive learning. arXiv preprint arXiv:2004.11362 (2020)
7. Misra, I., Maaten, L.v.d.: Self-supervised learning of pretext-invariant representations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6707–6717 (2020)
8. Rokach, L., Maimon, O.: Clustering methods. In: Data mining and knowledge discovery handbook, pp. 321–352. Springer (2005)
9. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. arXiv preprint arXiv:1706.03762 (2017)