Jacob Levy (jal477), Wyatt Marshall (wgm46), Jerry Wu (jw798)

Professor Bart Selman

CS 4701 Final Paper

December 15, 2021

GitHub Link: https://github.coecis.cornell.edu/jal477/4701

Video Link:

https://drive.google.com/file/d/1NLcV3mhl_JhHyTvldxC7EMGImGazEQLH/view?usp=sharing

Using Machine Learning Techniques to Predict the Next Pitch Type

Introduction and Overall Goals

The use of artificial intelligence in sports is rapidly increasing as organizations realize its potential for improving strategies, maximizing player efficiency, and ability to analyze large quantities of data [1]. In baseball, artificial intelligence is being used to predict a player's future performance, improving player mechanics and training, and to create advantages on both sides of the pitcher-batter matchup [2]. This matchup is arguably the most important part of the game of baseball, as each play of the game begins with a pitch. Both the hitter and the pitcher are constantly looking to exploit advantages over their adversary. A significant attribute of the pitcher-hitter relationship is that the pitcher has the advantage of knowing what pitch they will throw, whereas the hitter must react to the pitch.

In 2017, the Houston Astros won the World Series. Following the season, reports emerged that the Astros were using cameras to steal the opposing catcher's signs, and then signaling to the hitter via a buzzer what type of pitch was coming next: either a fastball or an offspeed pitch. In perhaps the defining moment of this saga, Houston's Jose Altuve hit a walk off home run against the New York Yankees in game 6 of the ALCS to send the Astros to the World Series. After the game, he refused to take his jersey off, leading some to believe that he was wearing a buzzer and knew what type of pitch was coming [3]. While there is some debate as to whether or not this cheating is the main reason the Astros won the 2017 World Series, the reaction of the MLB showed a clear condemnation of the practice [4].

The main problem with the Astros' sign stealing campaign is that they were using cameras to view the catcher's signs to the pitcher, and then communicating the pitch type to the hitter via a buzzer or hand signs from the dugout. It was specifically this usage of a camera and buzzer that was against the rules of the MLB, and is what led to the outrage from the baseball community against Houston as well as

the suspension (and eventual firing) of their manager and general manager. However, there are no rules against using AI to predict what type of pitch will be thrown next. In this project, we decided to employ machine learning techniques in order to give the hitter a slight increase in predicting what pitch type the next pitch thrown will be, thereby slightly improving the hitter's chances of either getting a hit or avoiding swinging at a bad pitch.

The overall goal of this project is to improve a hitter's chance of correctly guessing the type of the next pitch that will be thrown. Hitters generally approach an at-bat with the goal of hitting a fastball, then reacting to an offspeed pitch if needed. However, it can be difficult to discern whether a pitch is a fastball or offspeed until it is too late, so sometimes hitters will guess on a pitch [5]. If they guess right, they're more likely to make good contact, but if they guess wrong it is nearly impossible to recover. We want to give hitters a better chance at guessing right. Broadly speaking, pitches come in three flavors: fastball, fastball with movement, or an offspeed pitch. Therefore, we have three categories of pitch type: regular_fastball, moving_fastball, and offspeed. It's not possible to guess every pitch correctly, but if we can slightly improve a hitter's chance of guessing a pitch correctly, then it is likely that the hitter's overall batting average and other hitting statistics will also improve.

We took a few different approaches in our attempt to achieve this goal. We firstly decided that we needed to have some naive predictions against which we could measure our AI generated predictions in order to see if there was an improvement. We implemented two different naive prediction models: a truly naive model, and a model that used the heuristic of the count. Then, we used three different machine learning techniques in order to beat these naive models: an ngram model, a neural network, and an SVM. In the following paper we will explain our data selection and processing, our choice of two naive models for a baseline, the design and implementation of the three machine learning models we developed, and our analysis of how the machine learning models compared to the naive models.

Software

A link to our code is found here: https://github.coecis.cornell.edu/jal477/4701. In our code, which is written entirely in python, we used multiple packages, primarily pandas, numpy, and sklearn, to help in data cleaning, feature engineering, model fitting and evaluation. We also utilized jupyter notebooks for fast and easy analysis.

IMPORTANT: our main data file is too big to be hosted on GitHub. These same instructions are in the README of the attached repository, but it bears repeating, since it is so critical: go to https://www.kaggle.com/josephvm/mlb-game-data?select=pitches.csv, and download the 'pitches.csv' file, and then locally drag that file into the inner of the two '4701' folders of this repo, once you have downloaded it onto your machine. The file is greater than the file size limit that GitHub allows, but is

crucial for this project to run. Without it, you will certainly experience errors, and be unable to run anything in the package.

Data and Approach

For this project, we used data we found open-sourced on Kaggle [6]. This data encompasses each pitch thrown in a Major League Baseball game from the beginning of the 2016 season, all the way until the end of the 2021 season. It also includes other data and metadata, some of which we used in feature engineering. Through feature engineering, we ended up with a large number of pitches, and for each pitch, features such as the current balls and strikes, the previous pitch thrown, if this a home or away game, etc.

With this data, we decided to consider the pitches of 10 different, all-star caliber pitchers. There are a few reasons why we segmented this by pitcher, as opposed to making a generalized model for all pitchers. Each pitcher has a different repertoire of pitches, a different pitching style, and serves a different role (starter or reliever). The 10 pitchers we chose were: Yu Darvish, Jacob deGrom, Dallas Keuchel, Rick Porcello, Max Scherzer, Jon Lester, Justin Verlander, Craig Kimbrel, Kenley Jansen, Nathan Eovaldi. We chose these pitchers because they have all made the MLB All Star Game. This is a good proxy for being successful pitchers, which means a few things of interest to us. It means that they are possibly very good at being unpredictable, as they have been successful in fooling batters. It also means they will pitch often (they are unlikely to be benched or cut), and just generally are more interesting to talk about and analyze than a random pitcher. Lastly, among these 10 pitchers, 2 (Kimbrel, Jansen) are relievers, and 8 are starters. Relievers traditionally possess a smaller arsenal of pitches, relying less on mind games and more on overpowering speed and movement. Our results illustrate this well, and give insight into the potential benefits and pitfalls of our approach.

Prediction and Model Evaluation

We discussed heavily what exactly the model should predict. There is a clear tradeoff between grouping the pitches categorically, and not grouping at all. If you don’t group in categories, then a prediction is more specific and useful to the hitter; however, it is also harder to predict correctly. If you group pitches into categories, the smaller the category, the easier it is to predict correctly, but the information you are presenting to the hypothetical batter is less useful. In the end, we resolved on

grouping all pitches into three categories: (1) fastball without movement (four-seam fastballs), (2) fastballs with movement (two-seam fastballs, sinkers, cutters), and (3) offspeed (changeups, curveballs, sliders, etc.). This grouping is small enough to be able to predict correctly, but wide enough that the information is valuable. Each category of pitches has a similar speed (MPH) range, and similar movement, which is useful in helping hitters with their timing, and anticipating how the ball will move in the .4 seconds they have to react. It is also similar to how the Astros reportedly would relay pitches to their hitters in their cheating scandal, which further indicates there is some value in segmenting this way.

For different pitchers, how many pitches are in each category varies quite a bit. Because of this, we had to decide between evaluating our models on either pure accuracy, or on some sort of weighted scoring mechanism. Though weighted scoring is often the right choice for imbalanced datasets (spam email prediction, for example), we resolved that in our case, accuracy alone would be the most useful way to evaluate how well a model is doing. For spam email, it is very bad if you mispredict an important, non-spam email as spam, but not too bad to mispredict an annoying spam email as normal email; thus weighted scoring makes sense in that case. There is no such analogue in our case; a batter is equally happy at any correct prediction, as all he wants is to time/react to the pitch better, regardless of the pitch. With that, we resolved on accuracy to be the metric we optimize for, and did not employ techniques such as upsampling, that are often useful when trying to optimize for weighted scoring with unbalanced datasets.

Naive Model

Our naive model follows a heuristic: it asks, what is the most typical pitch that the pitcher throws in this specific count? The count is the number of balls and strikes at the moment; there can only ever be 0, 1, 2, or 3 balls, and 0, 1, or 2 strikes. This yields 12 total possible counts. The model looks retrospectively and finds, at this point in time, what the most commonly-thrown pitch is for this pitcher in this count. One could argue that this is not such a “naive” method; while it might use a bit more of a nuanced heuristic, it nonetheless uses a heuristic. Furthermore, implementing such a strategy would already be very easy for an MLB team: simply print out a 3x4 chart, marking the most likely pitch for that pitcher in each of the possible counts. This is important because it represents what a (somewhat smart) human could do without any AI, and we are striving to build an AI that can do better than humans. Therefore, a well thought out benchmark is crucial for evaluating performance. When we compare the results of our other models to our “naive” model, this is the model we are referring to.

Ngram

Early on, we noticed a connection between predicting the next pitch type and the natural language processing task of predicting the next word in a sentence. One approach that NLP takes for this situation is to use an N-gram model. N-gram models have been shown to be very effective in generating language models, word/sentence prediction, and text classification [7]. N-gram models leverage conditional probabilities, using a Markov assumption that a word's use is influenced by the word(s) immediately preceding it. We decided to take an n-gram approach to pitch type prediction, making the analogous assumption that the probability of the next pitch type is influenced by the type of the pitch(es) preceding it. N-gram is certainly limited compared to the other machine learning methods we explored, but has the tradeoff of being much lighter weight and computable. Ultimately, we created a unigram, bigram, and trigram model. The bigram and trigram models beat the purely naive unigram model, but failed to perform better than the other machine learning approaches.

The first n-gram model we developed was the unigram model. This is the most naive model that is possible: it predicts the same pitch every time. The unigram model is created by computing **Count(pitch_type) / Count(total_pitches)** for the three pitch types (regular_fastball, moving_fastball, offspeed). Then, the unigram model chooses the max of these three probabilities every time. For example, pitcher Yu Darvish's pitch type probability breakdown is as follows: **{'regular_fastball': 0.3519388486344454, 'moving_fastball': 0.3352177721156143, 'offspeed': 0.31284337924994027}**. This means that over the course of his career, Darvish has thrown roughly 35% regular fastballs, 34% moving fastballs, and 31% offspeed pitches. Therefore, the unigram model for Darvish always predicts **regular_fastball**, as he throws this pitch type (slightly) more often than the other two pitch types. One thing to add here is that for some pitchers, the unigram model performs surprisingly well. To put this another way, some pitchers throw one type of pitch disproportionately more often than other types. For example, 80% of the pitches Kenley Jansen throws are moving fastballs. The fact that the unigram model performs well for Jansen is not a credit to the model, but rather is due to Jansen's pitch type distribution. The unigram model is by far the worst model. It is completely naive, and it underperforms our naive model that we described above. However, it is incredibly simple. All in all, it shouldn't be used, but it is interesting to note that some pitchers (like Jansen) throw significantly more of one pitch type than any other.

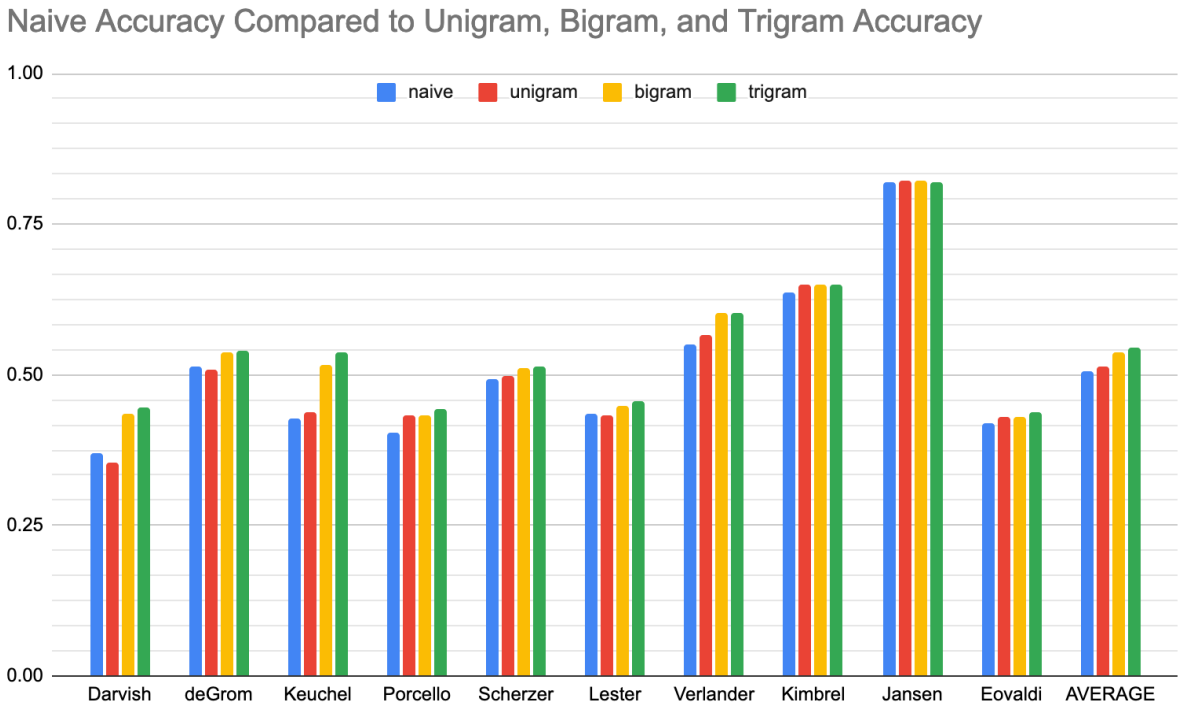
We implemented the bigram model using the calculation **Count(previous pitch, next pitch) / Count(previous pitch).** This is an estimate of the conditional probability of the next pitch type given the pitch type of the previous pitch. For example, the probability that Craig Kimbrel throws an offspeed given

he just threw a regular_fastball is about 0.33. The input to the bigram model is the previous pitch thrown, and the output is the pitch type that is most likely to come next based on the probabilities calculated for each bigram. Therefore, the bigram model stores three key, value pairs. It can be used easily because all a coach needs to do is enter the previous pitch type thrown, and the output will be the most likely next pitch type. This makes the bigram model highly usable in a real life situation. Since all calculations are done beforehand, the model can be used in real time during a game as all mappings are previously defined. Overall, the bigram model was a slight improvement over the naive approach. For some pitchers, like Yu Darvish and Dallas Keuchel, there was a roughly 8% increase in accuracy. From there, it went down to around a 3-4% increase for Jacob deGrom and Justin Verlander, and for the rest of the pitchers there was between a 1% increase or no difference at all.

The trigram model was created in the same way as the bigram model, with the difference being the equation was **Count(previous previous, previous pitch, next pitch) / Count(previous pitch, next pitch).** The conditional probability exploited here is given a pitcher's threw pitch type **a** then pitch type **b,** what is the probability of their next pitch being pitch type **c**. For example, if Dallas Keuchel's previous two pitches were offspeed type, then he has about a 34% chance of the next pitch also being offspeed. In the final trigram model, for each combination of two pitches, we stored a mapping to the most likely next pitch. This, like with the bigram, makes the trigram model usable in a real game, because all calculations are completed beforehand and the coach only needs to enter the previous two pitch types to receive the most likely next pitch type. The trigram model's performance was hardly an improvement over the bigram model. Dallas Keuchel saw the largest increase in accuracy, with the trigram model outperforming the bigram model by 2%. The rest of the pitchers saw improvements between 0-2%, with most being closer to 0.

Overall, the ngram strategy was a small success. The unigram model was clearly the worst, as it always returns the same prediction. The bigram model had the potential to be a significant improvement, but it was more likely to be no better than the unigram. And the trigram model usually performed about as well as the bigram. In comparing the bigram and trigram models to the "smart" naive model (as defined in the previous section), the results are similar. The bigram and trigram models outperform the smart naive model with some significance in a few cases, but generally they tend to have effectively the same performance. The advantages of all three models is that they are all computed upfront, so can be used either by a coach on the sideline who signals to the hitter, or even memorized completely by the hitter. Also, as some pitchers did see a large increase in bigram accuracy over unigram accuracy, there are clear situations where using this model will give the hitter a slight improvement. However, for most pitchers the bigram and trigram models offer no help at all. The fact that these models don't perform very well makes a lot of sense, as the main drawback of the models is that they take into account no information

outside of previous pitch(es). There are many other circumstances surrounding each pitch, such as inning in the game, runners on base, the count, etc. These things all affect the pitch type that will be thrown. All in all, the n-gram approach was interesting to explore, but it only results in a small improvement over the naive approaches for some specific pitchers.



Neural Network

Another approach we wanted to explore was the Neural Network. We believed a Neural Network would be powerful in this case because we originally wanted to predict individual pitch types in a multilabel classification. For example, we wanted the network to output 0 for Fastball, 1 for Cutter, 2 for Curve, etc. and eventually be able to predict the exact next pitch type.

In order to create the Neural Network we needed to first create a test, validate, and training set for each of the 10 target pitchers we were going after. In order to implement this, we created a dictionary with each pitcher and ran through our preprocessing to parse and extract each pitcher with the designated last name. Once we extract each pitcher's data from the pitcher.csv, we then input each individual pitcher's data through our data splitting mechanism.

The data splitting mechanism first splits all our pitch data on each individual pitcher into a training and testing set at an 80/20 split. Following this, we then split the training set into a training and validation set at the same 80/20 split. Thus, in the context of the overall data input, 20% of the data was

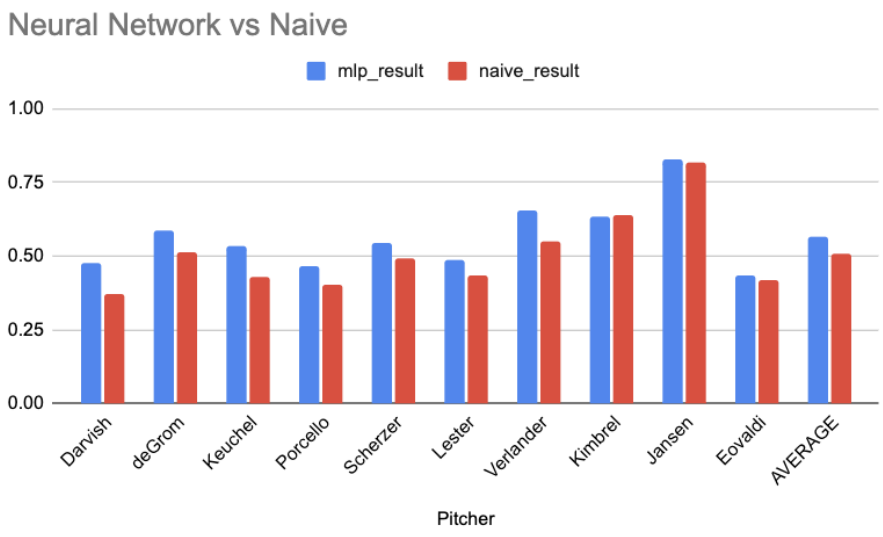
allocated to testing, 16% was allocated to validation, and 64% was allocated to training. We then used the training data to train the individual Neural Network for each pitcher's data set.

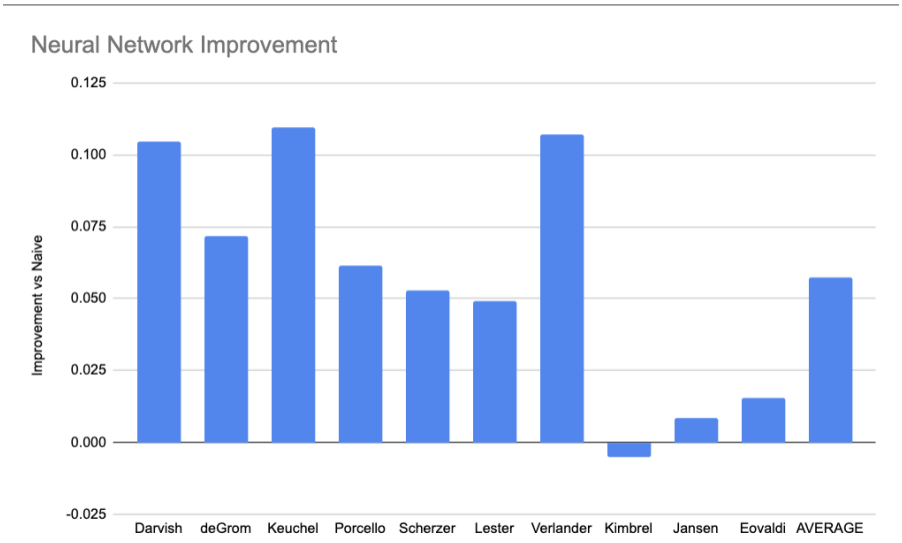
The process of training the actual Neural Network involved fine-tuning 3 parameters: the activation function, solver, and number of hidden layers (nxm). We passed the training data through each parameter combination and evaluated their score on the validation set. We then used the best performing model from the validation set to predict the test data.

As mentioned above, we originally wanted the model to predict a multilabel classification, where the input data could be mapped to one of six specific pitch types (labeled by integers 0 through 5). However, this method produced an accuracy that capped between 30-40%. Further investigation showed why; oftentimes the model would simply predict one type of pitch for 90-95% of the output labels. Therefore, the accuracy just happened to correlate with each pitcher's most common pitch types thrown; basically identical to our naive model results.

We then decided to simplify the label classification to a tertiary label of [0 = fastball without movement, 1 = fastball with movement, 2 = offspeed]. We felt this decision would still allow for practical applications in the field while also allowing the model to beat a naive approach. Upon running with this new classification, we saw mixed results compared to the baseline naive method (Figure 2).

Figure 2





On average, we did see an improvement above naive by about 5-5.5%. However, Jansen and Eovaldi saw near-zero improvement and Kimbrel saw a negative improvement compared to the naive baseline.

Further research into these specific pitchers shows why. Kimbrel and Jansen are relievers while the other eight pitchers are starters. As mentioned earlier, relievers usually throw a smaller range of pitch types. Consequently, it makes sense that the naive and neural network models performed similarly as there is less variance in the types of pitches thrown. For Eovaldi, he is the youngest pitcher of the entire group, which correlates with less pitch data. This smaller training data size could be a reason for why the neural network also had a slightly lower improvement margin over the naive baseline.

After these findings, we tried various additional fine-tuning methods in attempts to further increase our prediction accuracy. We ran through multiple methods such as:

(1) Increasing the number of hidden layers and number of nodes per layer

(2) Testing different learning rates

(3) Testing different L2 regularization parameters

However, all these methods produced no significant improvements to the model performance and only served to exponentially increase the model training times. Therefore, we settled on just fine-tuning the activation, solver, and smaller hidden layer set as this approach provided the best tradeoff between performance and training time.

In sum, the neural network was a small success. Similar to n-gram, an upside to the Neural Network is that it can be pre-trained before the game to allow for quick decision making when the batter is actually at the plate. However, an average improvement of 5-5.5% may not be a large enough margin to prove effective usage for a manager on the sideline during an actual game. Moving forward, there are a

few ways we can investigate to try and further improve the Neural Network's performance. Primarily, information on the specific batter at the plate would provide significant insight as pitchers often tailor their throwing strategies to their specific opponent. By cross-referencing both the pitcher's previous data and the batter's previous data, we hypothesize that the neural network will see significant boosts in performance.

SVM

The final model we attempted to use was SVM. We knew there would be several benefits with an SVM approach [8]. SVMs achieve success in multilabel classifications because they are not bound to linear differentiation. If each output label is distinct enough, then the SVM is able to make threshold decisions based upon data that is not necessarily linearly separable. Moreover, SVMs tend to be more memory efficient than neural networks, providing benefits if implemented as a sideline tool for baseball analytics.

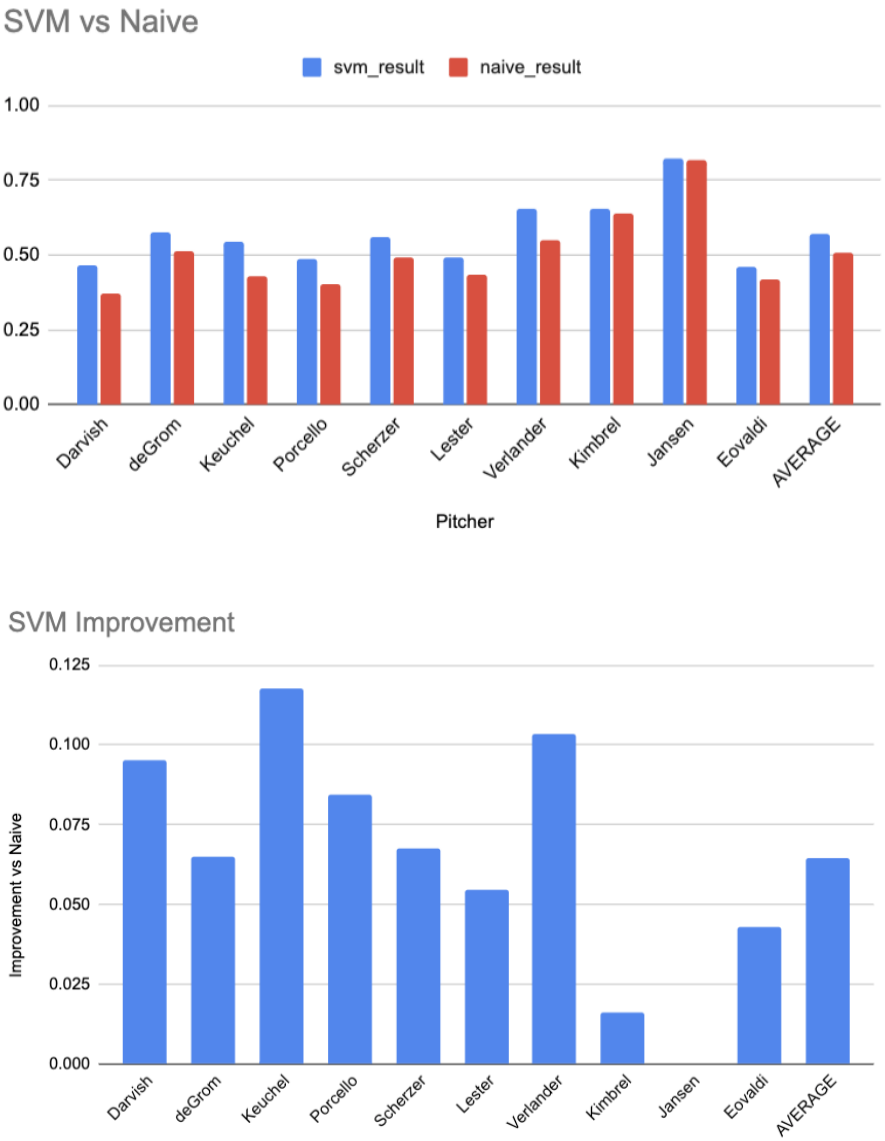
We were also aware of the potential disadvantages with an SVM. We ran the risk of low accuracy if the output labels were not individually distinct. This is because SVMs have trouble differentiating between classes that "overlap." This is another reason why we shifted our output labeling method from six pitch types to three pitch categories; the SVM would struggle to distinguish between similar pitches with different output labels. The three pitch category labeling method allowed the SVM to operate more accurately with well-defined target class differences. Indeed, this was reflected in the results. When we ran the SVM with the six pitch type labeling, we reached a threshold of 35-40% accuracy. However, our three pitch category labeling method saw significant increases in performance, reaching 55% average accuracy and in some cases significantly outperforming the naive model [Figure 3].

In order to implement the SVM, we used an identical train, validation, test splitting method as the Neural Network. The data on each individual pitcher is split 80/20 between a test set and a training set. The training set was then split 80/20 for a training and validation set to fine-tune our SVM. This data was then fed into the model, with each iteration fine-tuning with a kernel type. We then selected the kernel approach that scored the best on the validation set and used that to score on our test sample.

Interestingly, almost all the pitchers' models performed best with rbf; the only exception was deGrom whose model used a polynomial kernel. We considered removing the fine-tuning and using rbf as the "best fit' kernel approach to decrease training times since fine-tuning incurs a high overhead cost. However, we decided against this because our sample size is only 10 pitchers. deGrom's model using a polynomial kernel indicates that fine-tuning is effective and different pitchers may have predictive models that perform better with other kernels. Therefore, if this SVM training is leveraged for a larger set of all

MLB pitchers, we want the fine-tuning to offer the model flexibility to adjust to all different possible pitchers.

Figure 3



As shown above, we saw an average improvement margin above the baseline naive model by about 6-6.5%, out performing all our previous models. Some specific pitchers saw massive improvements; for example, Keuchel saw a 12% improvement above the naive baseline, and Verlander saw a 10.3% improvement above the naive baseline. These are significant margins and imply there is a direction for real-world applicability of this model in a gametime scenario.

Interestingly, the underperforming models were identical to the results seen with the neural network. Our relief pitcher models saw the worst improvement above baseline, with Kimbrel improving around 1.75% and Jansen not improving at all. Additionally, Eovaldi's model had the worst improvement among all starters at around 4%, albeit much higher than the margin seen with the neural network. This result supports our hypothesis about relief pitchers. Since they have a lower variance of pitch types and are more predictable from a pitching perspective, the model would perform similarly to the naive in prediction accuracy. Our hypothesis on Eovaldi's lower performance correlates with his status as the youngest pitcher of the set, resulting in less data that can be used to create an accurate model. In sum, it is encouraging to see that the neural network and SVM underperformed with the same pitchers, as it supports our above hypotheses.

Overall, the SVM was a success. It performed the best out of all our models, with the average margin of improvement above the baseline being 6-6.5%, higher than the 5-5.5% seen with the neural network. However, despite the SVM being the best in-house model we created, a 6-6.5% improvement is still marginal in the context of real world application. Similar questions arise as to whether or not such a marginal improvement justifies replacing current methods of baseball preparation, coaching, and intuition. Moving forward, we can try to improve the performance of our SVM by incorporating batter data and fine-tuning more parameters in the model training phase.

Summary and Analysis of Results

This section will be a summary of the comparisons of each model to naive, and comparing models to each other in terms of their predictive power, limitations, actual usability in a game scenario, and ultimately which model is better than naive, if any.

The details of the results of each approach we took are discussed in their own sections, but it's important to take a moment to summarize the overall performance of our models. To begin, our naive approach is actually a pretty strong, realistic baseline. Most hitters go up to the plate knowing which pitch a certain pitcher likes to throw, and even likely knows which pitch will be thrown in specific counts. Therefore, using this as our baseline ensures that by beating it with our machine learning strategies, we really are (slightly) improving the hitter's chances of predicting what type of pitch will be coming next.

The n-gram approach performed better than we initially thought it would. Of course, the unigram model is poor, but this is to be expected. However, seeing the bigram and trigram models outperform the unigram and naive models by a significant amount (for a few specific pitchers) was a bit of a surprise. There isn't a lot of information that the n-gram models take into account, so the fact that they were able to beat the naive approach at all is impressive. Also, the n-gram technique is extremely reasonable to use in a real game, as the most likely next pitch can be calculated in seconds just by knowing the previous

pitch(es). This means that a coach could easily have time to compute the most likely next pitch and signal this to the hitter before the pitcher is ready to throw.

The neural network approach outperformed the n-gram approach. This is likely because it took in much more information than the n-gram models did, allowing it to make more accurate predictions. For some pitchers (Darvish, Keuchel, and Verlander), the neural network made over 10% more accurate predictions than the naive model. This is a significant improvement, and would absolutely improve a hitter's chance of getting a hit. However, for the other pitchers, the increase in accuracy was much smaller, and it's notable that for Craig Kimbrel, the neural network actually performed worse than the naive approach. Furthermore, the neural network takes a fairly long time to train. However, once it's been trained, it can be used to make predictions quickly, so if the model were just trained before the game, it could make predictions in-game quickly, making it totally usable in a real-world scenario.

Finally, our best approach was the SVM. We saw improvements over the naive approach across the board, with Keuchel's pitches being predicted about 12% better than the naive model. This model had the highest average improvement over naive, and it also had the highest improvement over naive for one specific pitcher (Keuchel). The main issue with the SVM model is that it takes a long time to train, and it takes a long time to input the conditions in order to make a prediction. Like the neural network, this is also solvable by training it before the game, and then providing a simple user interface for a coach to input the correct data needed to construct a feature vector that the model will be able to predict quickly.

Conclusion

In all, we are pleased to have created a model that is able to predict the category of the next pitch of 10 phenomenal pitchers at a rate that is significantly better than a human could (represented by our naive model). Even though the performance boost is only about 6.5 pitches more out of a hundred, consider that about 28% of MLB games are decided by one run, and a team will face an average of 150 pitches per game. If one of those approximately 10 extra correctly-pitched pitches per game can be hit for a homerun, that has the potential to change the outcomes of many games. Baseball is a game of small advantages; analytics departments across the MLB will spend copious amounts of money for an improvement in expected runs per game. We believe our project is a good step in doing this, and that with more time and resources, we could provide even more accurate and robust predictions. If we could have access to the hitter's batting statistics at the time, for example, we believe that stronger results could have been obtained. Getting such data would have cost us some real money, so we decided against it, but it's not unrealistic to assume that with the level of data that an MLB team could give us, our model could significantly improve.

Baseball analytics has taken off over the past 10 years, and has changed everything about the game significantly. Teams now rarely longer bunt or steal bases, plays that were commonplace not 15 years ago. Players are encouraged to swing at fewer pitches, and swing for the fences when they do. Though we do not claim our results as they stand are going to change baseball from where it currently is, we believe that it shows that this can be done. Many teams will likely be afraid to try to tackle this problem, because of the technical difficulty of it, as well as the stain of the Astros scandal. We believe whatever team tackles this problem (potentially employing a solution similar to ours) will reap great benefits for doing so.

References

[1] Nadikattu, Rahul Reddy and Nadikattu, Rahul Reddy, Implementation of New Ways of Artificial Intelligence in Sports (May 14, 2020). Journal of Xidian University, Volume 14, Issue 5, 2020, Page No: 5983 - 5997, Available at SSRN: https://ssrn.com/abstract=3620017 or http://dx.doi.org/10.2139/ssrn.3620017

[2] Hall, Brian. " Artificial Intelligence, Machine Learning, and the Bright Future of Baseball ." *Society for American Baseball Research*, Sabr /Wp-Content/Uploads/2020/02/sabr_logo.Png, 21 July 2021, https://sabr.org/journal/article/artificial-intelligence-machine-learning-and-the-bright-future-of-baseball/.

[3] Zucker, Joseph. "Aroldis Chapman: Jose Altuve's Actions after Walk-off vs. Yankees 'Suspicious'." *Bleacher Report*, Bleacher Report, 13 Feb. 2020, https://bleacherreport.com/articles/2876118-aroldis-chapman-jose-altuves-actions-after-walk-off-vs-yankees-suspicious.

[4] Verducci, Tom. "Why MLB Issued Historic Punishment to Astros for Sign Stealing." *Sports Illustrated*, Sports Illustrated, 13 Jan. 2020, https://www.si.com/mlb/2020/01/13/houston-astros-cheating-punishment.

[5] Cañal-Bruland R, Filius MA, Oudejans RR. Sitting on a fastball. J Mot Behav. 2015;47(4):267-70. doi: 10.1080/00222895.2014.976167. Epub 2014 Nov 25. PMID: 25425271.

[6] Joseph Mohn. (11/26/2021). MLB Game Date, Version 14. Retrieved November from https://www.kaggle.com/josephvm/mlb-game-data.

[7] Jurafsky, Dan, and James H Martin. "Speech and Language Processing (3rd Ed. Draft) Dan Jurafsky and James H. Martin." *Speech and Language Processing*, Stanford University, 21 Sept. 2021, https://web.stanford.edu/~jurafsky/slp3/.

[8] S. Karamizadeh, S. M. Abdullah, M. Halimi, J. Shayan and M. j. Rajabi, "Advantage and drawback of support vector machine functionality," *2014 International Conference on Computer, Communications, and Control Technology (I4CT)*, 2014, pp. 63-65, doi: 10.1109/I4CT.2014.6914146.