Project 5 - Capstone Project - Predict the Next Pitch

Define the problem

What is the problem you're trying to solve, why is it important to solve it?

Baseball has become a very analytical sport and now it seems as if everything can be analyzed with data. In this new world, teams have started to implement numerous analytical techniques including defensive shifts, individual statistical importance, batter tendencies based on pitch locations, etc. This data based strategy had become popular from Billy Bean and his Sabermetrics. Here, I will try to apply another statistical approach on the basis on machine-learning by trying to predict the next pitch a pitcher will throw by incorporating information that is available to a batter such as the count, the current game state, pitcher's tendency to throw a particular type of pitch, etc. This becomes a multi-classification problem by predicting the next pitch type.

How does it affect people? How does it affect you?

Competitive sport teams will do anything to win, deflating footballs or scuffling baseballs for instance. This new technique of pitch prediction can help teams gain a competitive advantage against their competitor. This strategy can help the offensive team by telling them the next pitch thrown by the pitcher for that current situation. This technique can also inform the current team's pitcher of how predictable he can be and that he needs to mix up his pitches better. Even though some players will always be stubborn and refuse this information, since they will always believe human intuition is the only way the play baseball, this technique can greatly increase the pitch guess expectations. This problem affects me and is close to my heart as a former college baseball player, as it would have been useful to know some of this information when I played.

Describe a solution

How accurate do your results need to be? How quickly does your algorithm need to produce an answer?

To make this an suitable prediction problem, the prediction accuracy needs to above the random guess line (25% for a pitcher with 4 pitches distributed evenly) and probably closer to10 -15% accuracy above the random guess line to have any real effect on teams. The suitable metric in this situation would be classification accuracy with the only the correct pitch type being acceptable (number correct / total pitches). With a multi classification problem we could not use any binary metrics like ROC curves and as this is not a binary (yes / no) problem. Unlike a binary problem, the accuracy in this instance will not be as high, since there are many different types of pitches that could be thrown. We could predict fastball / off speed to make it a binary problem and would expect the classification accuracy to be higher, however it is more intuitive to use multiclass prediction.

The runtime of the algorithm would not matter a great and not need to be super fast, as a team could run the algorithm well before they needed it. However, the prediction process run time becomes much more important in one case. If the algorithm is used during a game, it would need to be run on a pitch by pitch basis and therefore need to produce an answer quickly. The other tactic to use the algorithm is that it could be used before the game to gain some insight on

pitcher tendencies and would not need to be as quick. The run time of the algorithm also will depend on the current dataset being read into the pandas dataframe. Therefore a pitcher with more tenure in the league, meaning that they will have more games played, will have thrown more pitches, and thus more rows in the data frame.

Analyze the problem

What is the size of your dataset?
The current pitchers dataset has 30660 rows and 72 columns

How many features are present?
There are 70 number of features present

Which features seem most promising?
The features that seem to be the most promising in this dataset are strikes, balls, inning, prev_pitch_type, and prev_pdes. With some substantive expertise in this area, the situation of the at-bat, the stance of the player, and history of the batter is probably the most important. With only a subset of the desired features, strikes, balls and previous pitch type seem be the most important here.

Are there any categorical variables that may need to be converted?
With basically the entire dataset being categorical variables, I will need to convert many features. Therefore, mostly all the created and transformed predictors will need to be converted to dummy variables and/or bined, with exception to a couple which we already suitable and close enough to being normalized (-1 / 1) and don't needed to be scaled. These variables including stand, prev_pitch_type, prev_pitch_outcome, prev_pitch_location, prev_pdes, prev_des, prev_pitch_speed_bins, and inning. The issue here is that this will create a ton of new columns in the dataframe and make the runtime of the classifier much more complex and slower, so I will probably subset these columns before I pass the `X` predictors into the algorithm.

Which machine learning algorithms would you expect to perform well?
1) Support Vector Machine
 - Using a soft-margin as their might be certain data points that might need to overlap the margin and they could be ignored easily and SVM can easily extend to a nonlinear boundary with the use of kernels. I will need to use either OneVsRest or OneVsOne to make an SVM work with multi-class.
2) Naive Bayes Classification
- I can incorporate class priors to use a bayesian approach to denote my prior beliefs in each pitch being thrown. Therefore, I can try and influence the data with the mean of each pitch being thrown. For multi-class classification, I can use MultinomialNB, however if I do introduce negative values into the classification, Naive Bayes will have trouble and cannot be used.
3) Decision Tree
 - Can work with a tree like structure to simulate a the decision process. I like this technique in this type of problem as it sort of replicates the human brain and stimulates questions being

contemplated before each pitch. It becomes like a person thinking process and will my background in this area this is how a pitcher thinks although it might not be as apparent, but subconsciously a pitcher will go through many of these steps.
4) Ensemble Methods - Random Forest and Gradient Boosted Machines
- Incorporate the decision tree structure with the improved accuracy. GBM can also learn based on the previous mistakes and classify more appropriately on what it gets incorrect.

How easy is it to convert the available data into a suitable form?
With being data scraped from a website, this data took some wrangling to get it into a desired format to even create a problem to be solved. I had to alter and create many different columns and shift rows around to be able to perform the suitable task. This was definitely not a simple read in and chung type of problem, as I had to perform multiple steps to just get it into the correct format. But as they say 80% of the time doing Machine Learning is data acquisition and data wrangling, and no different here. Another thing I had to account for is that I also had to make this compatible with every different pitcher read into the data frame. I had to make sure that the columns and rows will line up with different datasets, since different pitchers will throw a different number of pitches. This format will build a separate algorithm for each pitcher's data. Therefore, this way it will scale well.

Implement a solution

What pre-processing operations do you have to carry out on the features?
There are a couple of different preprocessing steps needed for this dataset. Scaling and normalization are not really needed in this analysis as almost all of the data has been transformed into dummy variables / bins and the ones that are not are already low enough values and suitable for the algorithms. The two steps that were really needed are transformation and selection of features. Transformation becomes present as I had to create multiple new columns to be able to work with this data and even have a question to answer. Also feature selection is going to be necessary as there will probably be a lot of features with all of the dummy variables created and I will likely want to shorten the amount of features for a more precise solution.

Are there any incomplete data points or outliers that you have to work around?
In this dataset there aren't really any outliers as all of the points are reasonable for a pitcher. For example, there are not any pitches that were thrown 120 mph or any erroneous amounts. The only data points that aren't suitable for analysis were a couple type pitches, including intentional walks and pitch outs. The good news here is that I can simply just delete these points as intentional walks don’t need to be predicted and pitch out are so rare and aren't determined by previous actions, but the player on base.

Using the metric(s) you defined earlier, measure your current performance. Is it close to what you expected?
Using the current metric and untuned parameters, the testing error for most models is between 53% - 65%. This result is not exactly what I was hoping for as I was hoping for a better accuracy model, but this error rate does make sense if one thinks about it. This problem should be a hard

problem to solve as the data is based on human intuition and is not like a simple text classification problem when there is true inputs, but the data is based on impulse decisions. Therefore, it makes sense that the classification accuracy would not be as high, in a problem like this. In these current models, the accuracy is about a 10% increase over just predicting the pitcher's main pitch, fastball, for each pitch. It is obvious here that the decision tree and the random forest is overfitting with the base parameters. Hopefully, with some tweaks and parameter tuning, I can improve the model and it can reveal even more true insight into this data.

Are there any better metrics you can come up with?
When evaluating a multi class classification problem, it is typically difficult to use many other evaluation metrics as they do not scale well when there are more than two classes present. I would have liked to use an ROC curve to evaluate the predictions, but with a multi-class problem one really can't use this metric. (It is possible but evaluating this metric could possibly be even more confusing). I could decide to use something like a 'Balanced Error Rate' or 'Combined Error Rate' but straight accuracy is still the most suitable in this situation for a multi-class problem.

Refine your solution

For each version of your solution, track what changes you make and how they affect performance. Does it ever become worse? If so, note down and figure out why.
I chose to explore two of the models further, one ensemble method and the Support Vector Machine. For each I experimented with the number of features in the model and used an loop to choose the best the best K for SelectKBest. Then I used grid search to discover the the most relevant parameters for each classifier. Also, intertwined in these iterations, I have made numerous changes in all works of the analysis process. There was a lot of going back and forth, running simple models, creating new features and removing features.
Throughout this process, some newly created features worked while others did not, it consisted of a lot of trial and error to get the final models for each classifier. With some tweaks the models did sometimes became worse when I did add some certain few features. For instance, my initial intuition was to add batter_id, however when I did the model became a worse and much slower to run because it became much more complex and overwrote some other better parameter as it added like 400 more dummy columns to the feature vector. Also, when I added pitch count it did the same thing, probably because of the size of this vector contrast with the size of most others in the feature vector.

Report how your project evolved, and what changes you made to your specifications (if any).
This project was a lot from beginning to end. Before I had even began, I knew that I needed to do a lot of work on just to obtain this data, make a response variable, and needed to think about what would be the best predictors for this type of problem. By starting out this problem with a solid plan and knowing that I would have to create a lot of variables and do many transformations before hand, I had a good grasp of the situation and knew what i was getting myself into.

As I worked through this problem, with a project like this where most of the predictors are created vectors, it became a lot of trial and error. The trial and error approach can be very tedious and might not be the most efficient. The hardest part was trying to understand data and how it work with the algorithms. It seemed as if after every iteration of a model, I discovered something new or bad about the data and either added it or subtracted it from the classification problem. Which meant, it took a lot of iterations on the problem to get a good grasp on everything involved in the analysis.
After running these final models and continuing to tweak and create features, I finally realize that if I wanted to increase the performance anymore, I would need to obtain more relevant information on the teams and more on the current game situation. I would need more on the 'Batter Profile', including his Slugging Percentage and runs for each pitch class. Other features like defensing formation or something like weather could also be helpful in determining the environment the pitcher is throwing in.

What was your experience like working on this project? Do you feel more confident taking on open-ended projects like these in the future?

My experience with this project is that it was more fun and engaging than other projects as it was based on something I have passion for as I was able to use some of my expertise in this field. On the other hand, with a project like this, I have realized that it is a lot of hard work, a lot of time commitment, and a lot of going back and forth between data modeling, parameter tuning, data manipulation and brain power to achieve an ideal solution. And even after that the most important thing when it comes to machine learning is the data. I have discovered that the data is king and influences everything in the problem. Therefore, a good analysis cannot change the what the data says. I have heard this phrase a lot, but never really experienced it for myself, but in this analysis I have truly understood what this really means.
After doing this analysis, I do feel very confident in taking on open-ended projects. The good thing in a problem like this is that there is no true end point and one can work and long as desired trying to improve the chosen metric.