The Report committee for Javier Palomares, Jr. certifies that this is the approved version of the following report:

Suggesting Pitches in Major League Baseball

APPROVED BY

SUPERVISING COMMITTEE:

Constantine Caramanis, Supervisor

Alexandros Dimakis, Co-Supervisor

Suggesting Pitches in Major League Baseball

by

Javier Palomares, Jr.

Report

Presented to the Faculty of the Graduate School
of the University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in Engineering

The University of Texas at Austin
December 2019

I dedicate this to my parents who have always provided their unconditional support.

Acknowledgments

Many thanks to Professors Caramanis and Dimakis for their supervision and guidance, and to my parents, friends, and coworkers at Home Depot for their support. Thank you.

Suggesting Pitches in
Major League Baseball

by

Javier Palomares, Jr., MSE

The University of Texas at Austin, 2019

Supervisors: Constantine Caramanis and Alexandros Dimakis

Pitchers in Major League Baseball need to keep batters from anticipating the next pitch. They do this by selecting a good pitch type and zone to throw. Pitchers often make this selection haphazardly. In this paper, we present a machine learning model using the data from the PITCHf/x system installed in Major League stadiums to first predict good and bad pitches, and then to suggest the following pitch type to throw that will result in good outcomes.

Table of Contents

List of Figures

Chapter 1

Selecting the Pitch to Throw

Pitch selection is an important part of the game of baseball. The pitcher settles a pitch type and location to throw with the goal of throwing a strike and preventing the batter from making contact with the ball. A good pitch selection keeps batters guessing and unable to hit the ball. On the other hand, a bad pitch selection lets batter anticipate the pitch that is thrown and hit the ball into play.

During games, pitchers make their pitch selection in real time. When making this decision, pitchers can base their decision on many factors such as the pitcher's previous experiences, the current game score, the runners on base, the batter's skill level, the outcome of the batter's previous appearances, and the pitcher's skill level. However, in general pitchers make this decision based on instinct and gut feeling. There is no general framework on how to quantitatively take the factors mentioned above into effect and systematically find a best pitch to throw. This is despite the abundance of available data.Since 2007, Major League Baseball stadiums are equipped with high speed cameras that track over 40 metrics on thrown pitches. Ultimately, the data collected by Major League Baseball is being underutilized.

The proposal for this report is to use the data to develop a model that makes the pitch selection systematically. The model should evaluate the current game situation, the events that led to the current situation, the pitcher's and batter's previous appearances, and utilizing the available data, determine the pitch type that provides the pitcher with the best opportunity of meeting his goal.

Chapter 2

Relevant Work

2.1 Predicting whether the Next Pitch is a Fastball

In 2012, Gartheeban Ganeshapillai and John Guttag presented their paper Predicting the Next Pitch at MIT's Sports Analytics Conference. In the paper, they chose to model predicting if the next pitch is a fastball as a binary classification problem. Ganeshapillai & Guttag used a linear support vector machine to build a separate predictor for each pitcher and trained it using data from the 2008 season. After testing on data from the 2009 season, the model provided a mean improvement on predicting fastballs of 18% and a maximum improvement of 311% in comparison to a naive classifier that always predicts the pitch most commonly thrown pitch by that pitcher. Ganeshapillai & Guttag found the most useful features in predicting whether the next pitch is a fastball were the pitcher and batter from the previous pitch, the number of strikes and balls after the previous pitch, the previous pitch's type, result, velocity, and zone, and the score of the game. Ganeshapillai & Guttag did not find a significant improvement in the predictor's accuracy by including longer pitch sequences including more than the previous thrown pitch. Based on this observation, we decided to restrict our analysis for this report to only use data from the previous pitch. We used some of use the same features, in addition to others not available in the data set used by Ganeshapillai & Guttag, to first predict the outcome of a pitch, and then suggest the pitch to throw based on the previous pitch (Ganeshapillai and Guttag, 2012).

Chapter 3

PITCHf/x

Since 2007, MLB has tracked every single pitch thrown in games with a system of high speed cameras names PITCHf/x. Using this system, MLB collects over 40 data points such as velocity, acceleration, and spin rate in the x, y, and z directions. The PITCHf/x coordinate system is oriented to the catcher's perspective- the x direction is left-right across the plate, z is the vertical height, and y is the direction towards the pitching mound. Additionally, MLB measures players' skill levels by tracking batter's batting average (avg), runs batted in (rbi), and homeruns (hr), and pitcher's earned run average (era), $wins$, and $losses$. Player data is tracked for the season. The data isfreely available to the public through MLB's Gameday[1] portal. All of the data used in this report came from this portal.

3.1 Pitch Types

One of the data fields included in PITCHf/x is $pitch\_type$. This field is the most probable pitch type according to a neural net classification algorithm developed by Ross Paul of MLB Advanced Media. The algorithm classifies the pitch into the following classes by their abbreviation (Slowinski, STATCAST).

Pitch Type	Abbreviation
four-seam fastball	FA
fastball	FF
two-seam fastball	FT
cutter	FC
forkball	FO
splitter	FS
gyroball	GY
sinker	SI

Pitch Type	Abbreviation
split-fingered fastball	SF
slider	SL
screwball	SC
changeup	CH
curveball	CU
knuckle-curve	KC
knuckleball	KN
eephus	EP

Figure 3.1: The PITCHf/x packages classifies pitches using a neural net classification algorithm developed by Ross Paul of MLB Advanced Media.

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

3.2 Zone

Another field included in the data is $zone$ which classifies the pitch type according to the location of the pitch as it crossed home plate. The data also reports the x and z values of the pitch location, where the x value is measured from the center of the plate with distances to the right being positive and to the left being negative. The z value is measured from ground level. Values are given in feet.

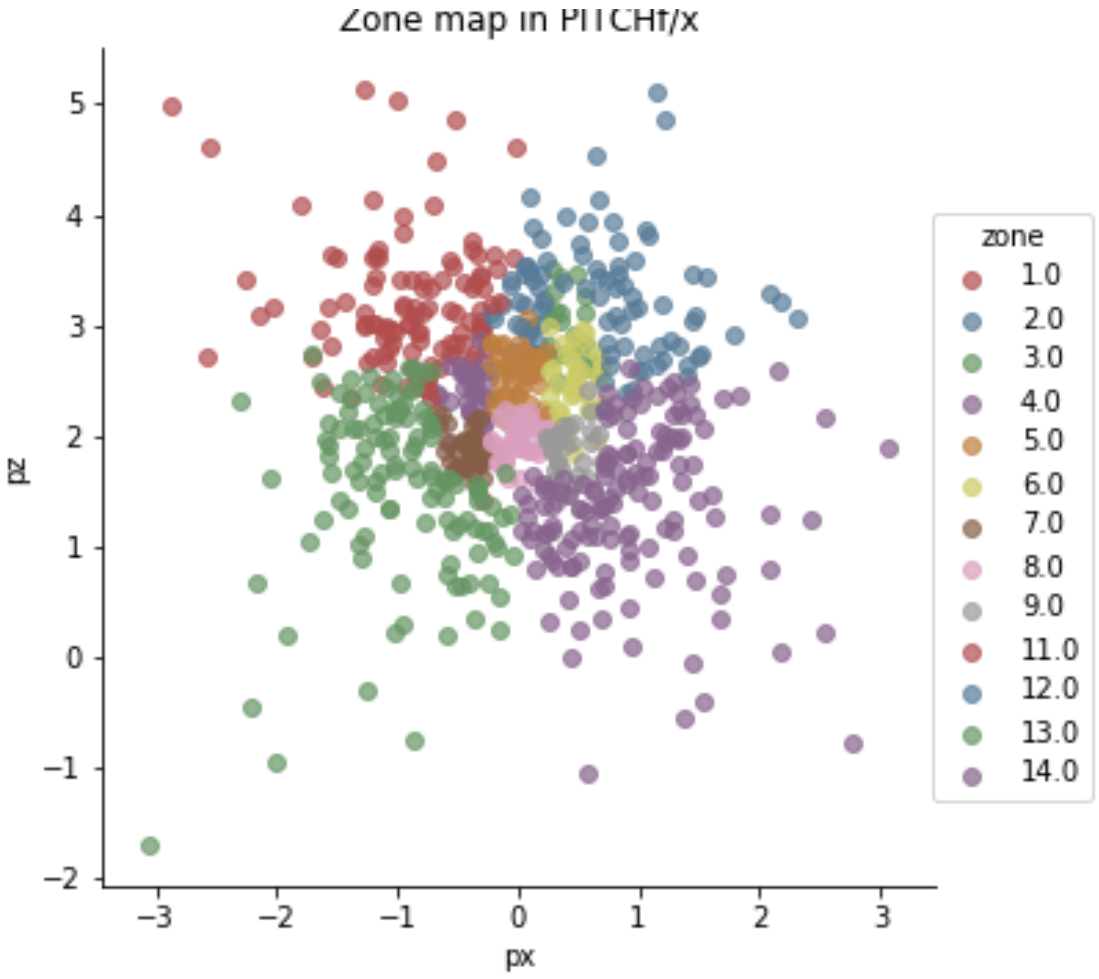

Figure 3.2: PITCHf/x classifies pitches into 1 of 14 values of $zone$.

3.3 Suggesting a Pitch

As previously stated, the proposal for this report is to suggest pitches. What this corresponds to is selecting the pitch type to throw as well as the zone to aim for out of the $16 \cdot 14$ possible combinations.

3.4 Other Data reported by PITCHf/x

PITCHf/x also reports the following data points (Fast):

- des: a brief text description of the result of the pitch: Ball; Ball In Dirt; Called Strike; Foul; Foul (Runner Going); Foul Tip; Hit by Pitch; In play, no out; In play, out(s); In play, run(s); Intent Ball; Pitchout; Swinging Strike; Swinging Strike (Blocked).

- id: a unique identification number per pitch within a game. The numbers increment by one for each pitch but are not consecutive between at bats.

- $type$: a one-letter abbreviation for the result of the pitch: B, ball; S, strike (including fouls); X, in play.

- $start\_speed$: the pitch speed, in miles per hour and in three dimensions, measured at the initial point, $y0$. Of the two speeds, this one is closer to the speed measured by a radar gun and what we are familiar with for a pitcher's "velocity".

- end_speed: the pitch speed measured as it crossed the front of home plate.

- sz_top: the distance in feet from the ground to the top of the current batter's rulebook strike zone as measured from the video by the PITCHf/x operator. The operator sets a line at the batter's belt as he settles into the hitting position, and the PITCHf/x software adds four inches up for the top of the zone.

- sz_bot: the distance in feet from the ground to the bottom of the current batter's rulebook strike zone. The PITCHf/x operator sets a line at the hollow of the knee for the bottom of the zone.

- pfx_x: the horizontal movement, in inches, of the pitch between the release point and home plate, as compared to a theoretical pitch thrown at the same speed with no spin-induced movement. This parameter is measured at $y = 40$ feet regardless of the $y0$ value.
- pfx_z: the vertical movement, in inches, of the pitch between the release point and home plate, as compared to a theoretical pitch thrown at the same speed with no spin-induced movement. This parameter is measured at $y = 40$ feet regardless of the $y0$ value.
- px: the left/right distance, in feet, of the pitch from the middle of the plate as it crossed home plate. The PITCHf/x coordinate system is oriented to the catcher's/umpire's perspective, with distances to the right being positive and to the left being negative.
- pz: the height of the pitch in feet as it crossed the front of home plate.
- $x0$: the left/right distance, in feet, of the pitch, measured at the initial point.
- $y0$: the distance in feet from home plate where the PITCHf/x system is set to measure the initial parameters.
- $z0$: the height, in feet, of the pitch, measured at the initial point.
- $vx0$, $vy0$, $vz0$: the velocity of the pitch, in feet per second, in three dimensions, measured at the initial point.
- ax, ay,az: the acceleration of the pitch, in feet per second per second, in three dimensions, measured at the initial point.
- $break\_y$: the distance in feet from home plate to the point in the pitch trajectory where the pitch achieved its greatest deviation from the straight line path between the release point and the front of home plate.
- $break\_angle$: the angle, in degrees, from vertical to the straight line path from the release point to where the pitch crossed the front of home plate, as seen from the catcher's/umpire's perspective.

- $break\_length$: the measurement of the greatest distance, in inches, between the trajectory of the pitch at any point between the release point and the front of home plate, and the straight line path from the release point and the front of home plate.

- sv_id: a date/time stamp of when the PITCHf/x tracking system first detected the pitch in the air, it is in the format $YYMMDD\_hhmmss$.

- $type\_confidence$: the value of the weight at the classification algorithmâĂŹs output node corresponding to the most probable pitch type, this value is multiplied by a factor of 1.5 if the pitch is known by MLB to be part of the pitcher's repertoire.

Chapter 4

Getting the Data

The data available at MLB's Gameday portal is formatted into xml files. There is 1 xml file per game,innings_all.xml, containing all of the innings, half innings, at bats, and pitches occurring during the game formatted as xml elements in sequential order. Within this file there is a *game* element composed of 9 *inning* child elements (or more if extra innings are needed to break a tie). Each *inning* element has a *top* and *bottom* child element for each half inning when teams swap batting and fielding. The *top* and *bottom* elements have *atbat* child elements for each batter appearance in the half inning. The child elements of the *atbat* element is the sequence of pitches in the at bat. Additionally, there is a players.xml file per game containing the data for the players in the game.

Listing 4.1: Section of XML file for 03/29/2018 game between the Chicago Cubs and Miami Marlins.

```
<game atBat="571506" deck="605119" hole="643265" ind="F">
<inning num="1" away_team="chn" home_team="mia" next="Y">
<top>
<atbat num="1" b="0" s="0" o="0" start_tfs="164311" ... >
<pitch id="3" type="X" tfs_zulu="2018-03-29T16:43:11Z" x="
   107.75" y="170.38".../>
<pitch id="4" type="X" tfs_zulu="2013-06-02T01:36:25Z" x="
   80.69" y="137.29".../>
...
</atbat>
...
</top>
<bottom>...</bottom>
</inning>
<inning num="2" away_team="chn" home_team="mia" next="Y">...
   </inning>
```

```
<inning num="3" away_team="chn" home_team="mia" next="Y">...
   </inning>
...
</game>
```

4.1 pypitchfx

As of the start of the work undergone to write this report, there are various tools available for getting the data available through Gameday by parsing the xml files, such as Carson Sievert's pitchRx package written for the R programming language [1]. However, we found that they did not preserve the parent-child relationships between pitches, at bats, half innings, innings, and game elements found in the XML files, which made sequential analysis of pitches very difficult. These tools make it simple to get all of the pitches thrown in a game, but without being able to relate pitches to at bats, it is not simple to arrange pitches into sequences as was necessary for our analysis. Because of this, we chose to implement a Python library pypitchfx from scratch which not only parsed the data in the XML files, but also preserved the relationship between the elements. The package parses the data into Python classes in addition to a relational database as detailed in chapter 5. This effort took considerable effort and time. The package is now available for free and open use at as well as through the Python package manager pip [2].

4.2 Data for the 2018 season

Our analysis used the data from the 2018 season. The available data includes 2468 games, 22800 innings, 186647 at bats, and 728669 pitches. Using the pypitchfx library, all of the data was written to a relational database using the code sample that follows.

[1] Available at https://pitchrx.cpsievert.me/
[2] Can be installed using pip install pypitchfx

Listing 4.2: Using the $pypitchfx$ library, the data for the 2018 season was loaded to a PostgreSQL database running on Google Cloud Platform. The engine is a $sqlalchemy$ engine used to connect to the database. The library automatically creates the tables described in chapter 5 and populates the tables with data.

```
from sqlalchemy import create_engine
from pypitchfx.scrape import scrape_games_players

engine = create_engine('postgresql+psycopg2://postgres:
    username@password:port/')

scrape_games_players(start='2018-03-29',end='2018-04-30',
    engine=engine)
scrape_games_players(start='2018-05-01',end='2018-05-31',
    engine=engine)
scrape_games_players(start='2018-06-01',end='2018-08-31',
    engine=engine)
scrape_games_players(start='2018-09-01',end='2018-10-01',
    engine=engine)
```

Chapter 5

Relational Model

After parsing the xml files for data, the pypitchfx library writes the data to a relational database using a sqlalchemy engine connection to the database [1]. For this project, the data was written to a PostgreSQL database hosted on Cloud-SQL instance on the Google Cloud Platform. The database model preserves the parent-child relationship of elements in the Gameday xml file. Foreign key enforce 1 parent per child, and allow parents to have 0 to many children.

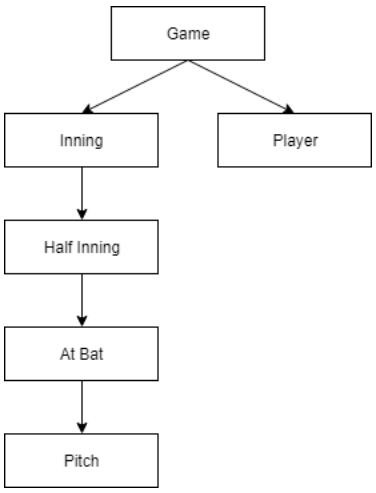

Figure 5.1: Hierarchy of elements in the data. Foreign key constraints enforce exactly one parent per child.

Additionally, unique identifiers are generated at parse time to guarantee uniqueness and to use as the tables' primary keys. The complete database model is show on the following figure.

[1] Documentatation available at https://www.sqlalchemy.org/

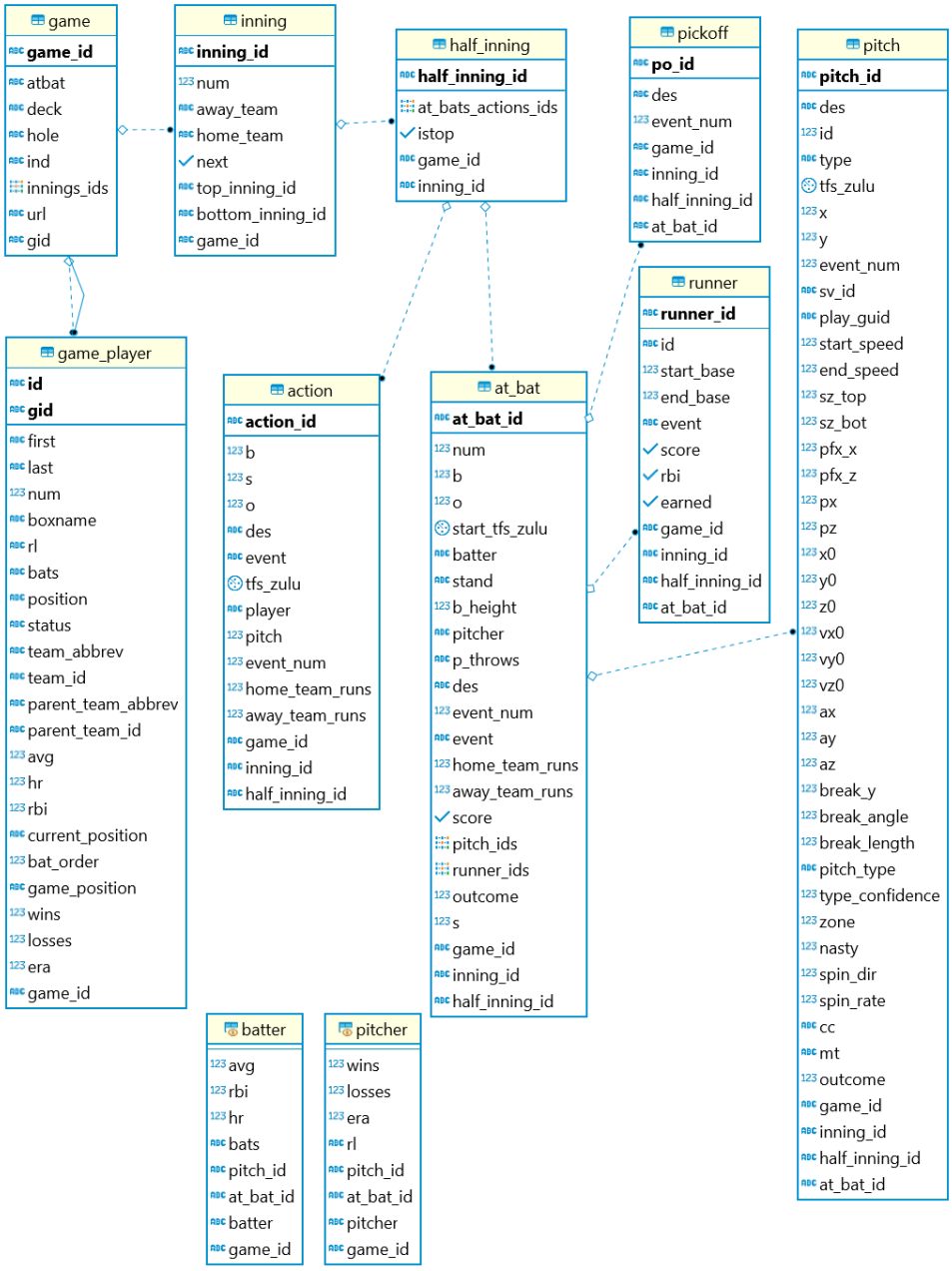

Figure 5.2: Entity-Relationship diagram of the database model the $pypitchfx$ tool writes the Gameday data to. All of the data fields available in the xml fields are mapped to columns of the same name. Generated UUIDs are the tables' primary keys. Foreign key constraints enforce parent child relationships. Entities maintain list of the parent to allow for fast lookups of the parent entity. The DDL statements used to insert and create the tables are available in the pypitch repository in Queries.py Batter and Pitcher are materialized views to used to distinguish pitchers and batters in the Game_Player table.

Chapter 6

Predicting the Outcome of a Pitch

The objective proposed in the introduction of this report is to train a model to suggest the pitch to throw following a sequence of pitches. However, rather than beginning by developing this model, the first action taken after populating the database was to model a simpler problem in support of the original objective. Instead of beginning to develop a model for pitch suggestions, the first action taken was to train a classifier for "good" and "bad" pitches. This was done to get familiar with the data as well as to determine if we can make inferences from it.

6.1 Binary Classification

We modeled predicting good and bad pitches as a binary classification problem. The input to the model is a feature vector consisting of the data points describing the physical trajectory of the pitch such as $vx0$, $vy0$, $vy0$, ax, ay, az, pfx_x, and pfx_z. Any meta data such as des, id, and sv_id is not included in the vector. Note that des contains a description of the result of the pitch. This and any fields containing any textual details of the outcome were not used in the model. Categorical data points such as $zone$ and $pitch\_type$ were one-hot encoded. Figure 6.1 shows the input vector in more detail.

The model's value to predict is a binary classifier - 1 for good pitcher, 0 for bad pitches. The pypitchfx library derives an $outcome$ value for every pitch. The values is set to 1.0 if the pitch resulted in a strike, including foul balls, or an out. Any other outcome is given a value of 0.0. We used this value as the output value to predict in the training data.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 117868 entries, 0 to 118110
Data columns (total 28 columns):
x                 117868 non-null float64
y                 117868 non-null float64
start_speed       117868 non-null float64
end_speed         117868 non-null float64
sz_top            117868 non-null float64
sz_bot            117868 non-null float64
pfx_x             117868 non-null float64
pfx_z             117868 non-null float64
px                117868 non-null float64
pz                117868 non-null float64
x0                117868 non-null float64
y0                117868 non-null float64
z0                117868 non-null float64
vx0               117868 non-null float64
vy0               117868 non-null float64
vz0               117868 non-null float64
ax                117868 non-null float64
ay                117868 non-null float64
az                117868 non-null float64
break_y           117868 non-null float64
break_angle       117868 non-null float64
pitch_type        117868 non-null object
type_confidence   117868 non-null float64
zone              117868 non-null object
nasty             117868 non-null int64
spin_dir          117868 non-null float64
spin_rate         117868 non-null float64
outcome           117868 non-null int64
dtypes: float64(24), int64(2), object(2)
memory usage: 26.1+ MB
```

Figure 6.1: Data fields contained in the feature vector used for the classification model. $zone$ and $pitch\_type$ are categorical values that were one hot encoded.All other fields are numerical values. $outcome$ is used as the binary variable to predict in training data.

Listing 6.1: The pypitchfx library classifies a pitch from its value in des. The data field contains a textual description of the outcome of the pitch.

```
def get_pitch_outcome(pitch):
    des = pitch.des.lower()
    # good pitch if we get foul or out or strike
    if 'foul' in des or 'out' in des or 'strike' in des:
        return 1.0
    return 0
```

6.2 Training the Model

We trained a gradient boosting model using the XGBoost library on a data set containing 200000 pitches thrown in the 2018 season. The data was split into test (20%) and training (80%) data sets. The data sets were used to train and validate an XGBClassifier using the parameter values given in Figure 7.1. A contains a code sample used to train and validate the classifier.

Parameter	Value
learning_rate	.1
n_estimators	1000
max_depth	5
gamma	0
subsample	0.8
colsample_bytree	0.8
scale_pos_weight	1
objective	binary:logistic
metric	auc
seed	1301

The data set contains 200000 pitches.

Figure 6.2: Parameter values used in XGBClassifier training.

6.3 Results

The XGBClassifier trained on the data as given above measure an accuracy of 0.885 and AUC score of 0.934 on the test data. These results suggest there is structure in the pitch data to use in order to make inferences on best pitches to throw.

Listing 6.2: This model has an accuracy of 0.885 and AUC score of 0.934 on test data.

```
Model Report
Accuracy (Train) : 0.9036770685098597
AUC Score (Train): 0.9562583322095853
Accuracy (Test) : 0.8855458426509453
AUC Score (Test): 0.9340879039026049
```

Given the results, we are confident that given a pitch, we can predict if it results in a good or bad outcome with good accuracy. Given the good results, we moved onto the more complicated problem of suggesting pitches.

Chapter 7

Suggesting The Next Pitch To Throw

As detailed in chapter 6, a binary classifier is able to predict the outcome of a pitch and achieve good results. On the other hand, predicting the outcome of a pitch isn't the problem we originally proposed to solve. By itself, this model provides the pitcher the answer to "If I throw this pitch, will it result in a good or bad outcome?", when what we are looking for is "Given the game situation, which pitch should I throw that will get the best outcome?" Thus, we formulated a different model to suggest the pitch type to throw following a pitch.

7.1 The Model

We approached the problem of predicting the next pitch type to throw as a multi class classification. Given the input, the model needs to classify to 1 of the 16 classes. The class represents the pitch type that has the best chance of producing a good outcome given the pitch in the input. The input to the model includes the features listed in section 6.1 as well as:

- $wins$: The number of wins the pitcher has for the season.
- $losses$: The number of losses the pitcher has for the season.
- era: The pitcher's earned run average for the season.
- rl: The pitcher's handedness.
- avg: The batter's batting average for the season.
- rbi: The batter's runs batted in for the season.
- hr: The batter's number of home runs for the season.
- $bats$: The batter's handedness.

7.2 Forming the Training Data

The first step in forming the training data was to get pitch data in sequence using the following SQL statement. The statement also joing the pitcher and batter data.

Listing 7.1: Getting the pitch, batter, and pitcher data in sequence

```
SELECT pitch.*, batter.avg, batter.rbi, batter.hr, batter.bats,
    pitcher.wins, pitcher.losses, pitcher.era, pitcher.rl,
    pitcher.pitcher as pitcher_id
FROM pitch
JOIN batter on pitch.pitch_id = batter.pitch_id
JOIN pitcher on pitch.pitch_id = pitcher.pitch_id
order by pitch.at_bat_id, pitch.id
```

The SQL query produced all the pitches in the 2018 season ordered in sequence within an at bat. Given this, it's simple to get the next pitch's pitch type that the model needs to predict. However, the model should only suggest pitch types that resulted in a good outcome, so the training data was modified depending on the outcome of the of the next pitch. If the next pitch resulted in a good outcome, then no modification was made. Otherwise, the next pitch type resulted in a bad outcome and the model should not suggest that pitch type. The model should predict another pitch type that is likely to produce a good outcome. The model makes the assumption that any pitch type is equally likely to result in a good outcome and for next pitches that had a bad outcome, the pitch type was replaced by any of the other 15 pitch types with equal probability $\frac{1}{15}$.

7.3 Training the Model

The data was also split into test (20%) and training (80%) data sets. Once again, we used the XGBoost library's XGBClassifier to model using the following parameters.

Parameter	Value
learning_rate	.1
n_estimators	1000
max_depth	5
gamma	0
subsample	0.8
colsample_bytree	0.8
scale_pos_weight	1
objective	multi:softprob
metric	auc
seed	1301

Figure 7.1: Parameter values used for multi-class XGBClassifier training.

7.4 Evaluating the Model

After training the classifier, we are left with a model that given a pitch, outputs the soft probability of each pitch type resulting in a good outcome. It is hard to evaluate this accuracy given the dataset since the model is predicting what could happen, not what did. We found that a suitable approach for evaluating the model is to take the test set and have the model predict the next pitch type with the highest probability of a good outcome. Based on the prediction, we'll form a dummy pitch with average feature values given the pitch type and pitcher. If the pitcher does not throw the predicted pitch then we'll take the average feature values across all pitchers for the given pitch type. Finally, we'll input this pitch into the classifier from chapter 6 predict if the pitch type will result in a good or bad pitch.

7.5 Results

The classifier from chapter 6 predicted a mean good outcome probability of .80 on the test data. More explicitly, the probability of the suggested pitches for the test data had the following distribution:

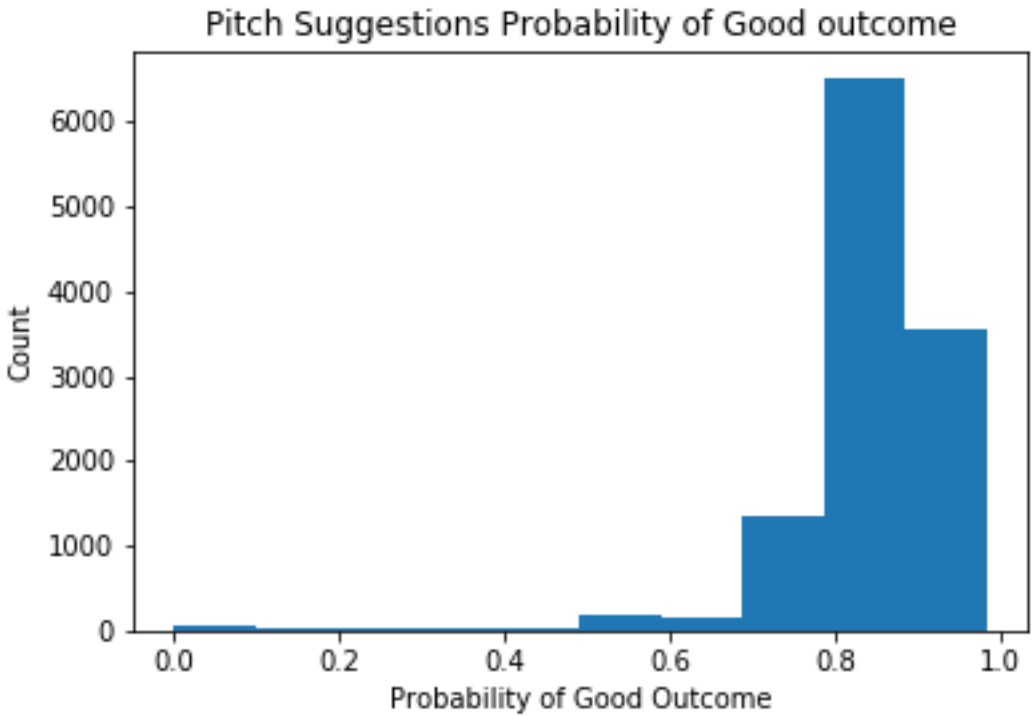

Figure 7.2: The probability of the suggested pitch resulting in a good outcome according to the binary classifer detailed in chapter 6. The model predicts that most pitches have a significant probability of resulting in a good outcome, so we can conclude that our model for suggesting pitches produces the desired result.

Chapter 8

Conclusion

The proposal for this report was to develop a model that made pitch selection systematically by evaluating the current game situation, the events that led to the current situation, the pitcher's and batter's previous appearances, and using the available data, the model should determine the pitch type that provides the pitcher with the best likelihood of a good outcome. In this paper, we see how this is possible with 2 neural networks, one to suggest pitches, and the other to predict a good or bad outcome of the suggested pitch. In chapter 6 we trained a classifier to classify pitches into good and bad outcomes that achieved good results. In chapter 7 we developed a model to suggest the next pitch's pitch type to result in a good outcome. Using the 2 models, we were able to suggest the next pitch that across the test data had significant probability of resulting in a good outcome. At this point, we are satisfied with the model and imagine as a future possibility how a more refined model can be used in live games.

8.1 Next Steps

Prior to using the model in live games, we would make the following improvements:

First, we would modify the outcome vector for bad pitches that we used in our model. At the moment, we assume that any other pitch type would do equally as good. Determining which pitch types can do better instead of the bad pitch will improve the performance of the model.

Additionally, we can improve by introducing the pitch $zone$ to the model so that the model also suggests where to throw the pitch. This will introduce a cross product to the set of possible pitches by giving the choice of 1 of 16 pitch types and 1 of 12 zones.

Another enhancement to the model is to refine the good/bad classification of pitches. We gave each pitch a binary good or bad label for simplicity- pitches that result in a strike, foul, or out are good, pitches that result in a ball, or hit are bad. What is more practical is to give a scalar value measuring how good or bad

the pitch since, for example, outs are better than strikes or fouls, and balls are less bad than hits. Similarly, not all hits are equally bad. A homerun is worse than a single with no runners on base. By giving a scalar value to the outcome, we would train the model to avoid the worst possible outcome.

Appendices

Appendix A

Code to Train a Binary Classifier using XGBoost Library

Python code used to train the XGBClassifier. The objective function is logistic regression, and the output is the probability that a pitch is classified as good. The evaluation metric is the area under the curve

Listing A.1: Training a classification using XGBClassifier

```
from sklearn import metrics
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

def modelfit(alg, x_train, y_train,x_test,y_test,
    useTrainCV=True, cv_folds=5, early_stopping_rounds=50):

    if useTrainCV:
        xgb_param = alg.get_xgb_params()
        xgtrain = xgb.DMatrix(x_train.values, label=y_train)
        cvresult = xgb.cv(xgb_param, xgtrain,
                          num_boost_round=
                          alg.get_params()['n_estimators'],
                          nfold=cv_folds,
                          metrics='auc',
                          early_stopping_rounds=
                          early_stopping_rounds)
        alg.set_params(n_estimators=cvresult.shape[0])

    #Fit the algorithm on the data
    alg.fit(x_train, y_train,eval_metric='auc')
```

```
    #Predict training set:
    dtrain_predictions = alg.predict(x_train)
    dtrain_predprob = alg.predict_proba(x_train)[:,1]

    # Predict testing set:
    dtrain_predictions_test = alg.predict(x_test)
    dtrain_predprob_test = alg.predict_proba(x_test)[:,1]

    #Print model report:
    print("\nModel Report")
    print("Accuracy (Train) : {}"
          .format(metrics.accuracy_score(y_train,
            dtrain_predictions)))
    print("AUC Score (Train): {}"
          .format(metrics.roc_auc_score(y_train,
            dtrain_predprob)))
    print("Accuracy (Test) : {}"
          .format(metrics.accuracy_score(y_test,
                                         dtrain_predictions_test
                                           )))
    print("AUC Score (Test): {}"
          .format(metrics.roc_auc_score(y_test,
            dtrain_predprob_test)))

learning_rate=.1
n_estimators=1000
max_depth=5
min_child_weight =1
gamma = 0
subsample = 0.8
colsample_bytree = 0.8
scale_pos_weight = 1
```

```
objective = 'binary:logistic'
metric = 'auc'
seed = 1301
xgb1 = XGBClassifier(learning_rate=learning_rate,
                     n_estimators=n_estimators,
                     max_depth=max_depth,
                     min_child_weight=min_child_weight,
                     gamma = gamma,
                     subsample = subsample,
                     colsample_bytree = colsample_bytree,
                     objective = objective,
                     eval_metric = metric,
                     nthread=4,
                     scale_pos_weight = scale_pos_weight)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size
                                                       =0.2,
                                                    random_state
                                                       =42)
modelfit(xgb1,X_train,y_train,X_test,y_test)
```

Appendix B

Code Repositories

All code written for this report is available on github. The pypitchfx library is open source and found at https://github.com/JavierPalomares90/pypitchfx. The code used to train is available at https://github.com/JavierPalomares90/masters_report. The code in this repository differs from what was used only by the login information from the PostgreSQL database which was removed for security.

Bibliography

Mike Fast. Glossary of the gameday pitch fields. URL https://fastballs.wordpress.com/2007/08/02/glossary-of-the-gameday-pitch-fields/.

Gartheeban Ganeshapillai and John Guttag. Predicting the Next Pitch. 2012. URL http://www.sloansportsconference.com/wp-content/uploads/2012/02/98-Predicting-the-Next-Pitch_updated.pdf.

Steve Slowinski. Pitch type abbrevations & classifications. URL https://library.fangraphs.com/pitch-type-abbreviations-classifications/.

STATCAST. Statcast interface product specifications. URL https://www.daktronics.com/Web%20Documents/HSPR-Documents/DD3322256.pdf.

Vita

Javier Palomares, Jr. was born in Chicago, Illinois to Mexican immigrants who quickly relocated to El Paso, Texas to avoid the cold. After completing high school and graduating from Parkland High School, Javier went off to attend college at Stanford University. After graduation, he worked for several years as a software developer before entering Graduate School at the University of Texas at Austin. He is currently employed at The Home Depot working in the online department.

Address: javierp@utexas.edu

This report was typed by the author.