**Objective**

Our goal is to use Major League Baseball play data from the first half of the Boston Red Sox 2018 season to classify each play in the second half of the 2018 season according to whether or not a model expects a resulting hit to occur. There are over 6,000 plays that occur in this given season and a wide range of attributes to leverage in our prediction effort providing an ample feature set and data size for training and testing a set of classification models. We will apply 4 different classification algorithms using 5 separate sets of features (4 of which are algorithmically generated) to identify a combination that is suitable to achieving our goal of classifying plays according to batter hit expectancy.

**Background Information**

The richest free source of play data for Major League Baseball games is found at [retrosheet.org](https://www.retrosheet.org/). The site hosts many types of data files which contain current and historic information on all ballparks, every player/manager/umpire who participated in a game, rosters, teams, and game events. Game event files contain every event for any team going back to 1921, where an event record is a set of attributes describing the conditions and outcome each time a batter participates in the game. Another term used interchangeably within the retrosheet documentation and throughout this report for event is **play**.

**The Data Set**

**Description**

Play records the entire series of pitches any time a batter faces an opposing team. The play records a total of 97 attributes, 44 of which were used in this project. The original attributes are described below. The index on the left of the arrow represents the index of that attribute in the original data set. The index on the right of the arrow represents the index of that attribute in the final data set. In the cases where the attribute is not represented in the final data set, there is only one index (the original) in parentheses. Furthermore, the attribute and description are italicized. The argument for dropping attributes is discussed in the preprocessing section below. The fields highlighted below, **hit value** and **hit flag**, represent the class attribute. The field hit value was converted to **hit flag**, where the values **1-4** were converted to **T**, and **0** converted to **F**. This is also further explained in the preprocessing section.

**Preprocessing: Converting event files to CSV**

As mentioned above, the original event files are taken from retrosheet.org. Event file records look like the following:

id,BOS201804050

version,2

info,visteam,TBA

info,hometeam,BOS

start,duffm002,"Matt Duffy",0,1,5

play,1,0,duffm002,32,BCSFBBX,43/G

Essentially, it is a report of a game identifier, a format version, meta information about the game, list of starting players, and a list of plays. The event file for the 2018 Boston Red Sox season was downloaded. Furthermore, a 2018 team roster file was downloaded containing all teams’ rosters for the 2018 season. The event file contains valuable information but is not in a convenient format for the purpose of this project. Fortunately, retrosheet.org provides a suite of tools for converting their data. One of these tools, BEVENT.EXE, takes the roster file for a given year and a team’s event file and converts it to a CSV generating up to 97 attributes (described in detail in Appendix A). The resulting file is perfect for the purpose of this project. One setback was that the BEVENT.EXE utility is only compatible with the Windows OS. Only having access to Linux and Mac OSes, we used AWS Workstations service to spin up Windows 2016 Server, copied the utility and data files onto the server and ran the converter (using the -f flag to produce all 97 attributes). The resulting CSV describing every play of the Boston Red Sox 2018 season was copied back onto our local systems and used in further preprocessing.

**Preprocessing: Reducing the overall data set**

As mentioned above, some of the 97 attributes describe the state of the game prior to the play’s occurrence, while other attributes describe the state of the play itself. Since we are trying to assess hit classification, the **hit value**, is one of those attributes describing the state of the play itself that we intend to retain. It was discovered that all of the other attributes describing the play itself contributed to the classifier to heavily. Furthermore, they are counter-productive to the objective of being able to predict the unknown (the result of a play) from the known (the state of the game prior to the play). So, we had to ask ourselves the following question. Which attributes are only those describing the state of the game prior to the play? Only those attributes, along with our class attribute, will compose our final data set. These are the 44 attributes noted in the **Description** section above. These 44 attributes are also listed in Appendix A below as those that are not in italics. The process of further reducing this data set via attribute selection algorithms is explored in the next section.

The Python code used to create the training, testing and all data sets is:

*from* drop\_cols *import* drop\_cols  
*from* headers *import* headers  
*import* pandas *as* pd  
*import* math  
  
file = '2018BOS.EVA.csv'  
extra\_drop\_cols = ['hit value', 'RBI on play', 'event type']  
df = pd.read\_csv(file, names=headers)  
df = df.drop(drop\_cols, axis=1)  
  
total\_count = len(df.index)  
train\_count = math.floor(total\_count/2)  
test\_count = total\_count - train\_count  
  
train\_df = df[:train\_count].copy()  
test\_df = df[train\_count:].copy()  
  
  
  
# hit - test  
hit\_test\_df = test\_df.copy()  
hit\_test\_df['hit flag'] = 'T'  
hit\_test\_df.loc[hit\_test\_df['hit value'] == 0, 'hit flag'] = 'F'  
hit\_test\_df = hit\_test\_df.drop(extra\_drop\_cols, axis=1)  
hit\_df\_cols = list(hit\_test\_df)  
hit\_test\_df.to\_csv('test.hit.csv', index=*False*)  
  
# hit - train  
hit\_train\_df = train\_df.copy()  
hit\_train\_df['hit flag'] = 'T'  
hit\_train\_df.loc[hit\_train\_df['hit value'] == 0, 'hit flag'] = 'F'  
hit\_train\_df = hit\_train\_df.drop(extra\_drop\_cols, axis=1)  
hit\_train\_df.to\_csv('train.hit.csv', index=*False*)  
  
# hit- all  
hit\_all\_df = df.copy()  
hit\_all\_df['hit flag'] = 'T'  
hit\_all\_df.loc[hit\_all\_df['hit value'] == 0, 'hit flag'] = 'F'  
hit\_all\_df = hit\_all\_df.drop(extra\_drop\_cols, axis=1)  
hit\_all\_df.to\_csv('all.hit.csv', index=*False*)

as mentioned, the **pandas** module was used for data manipulation and isolating and converting the hit value to a hit flag. The hit value can be **0 (no hit), 1 (single), 2 (double), 3(triple) or 4 (home run)**. The value **0** was converted to **F** indicating **no hit**, while all other values were converted to **T** indicating a **hit**. This series of values was stored in a new attribute called **hit flag**. At the beginning of the file, you can see that the first half of the plays in the season were used to compose the training set, and the second half of the plays in the season were used to compose the testing set. The **drop\_cols** variable externally stores all aforementioned columns to drop (also listed in italics in Appendix A). The Boston Red Sox 2018 play CSV used as input did not contain column headers. Retrosheet.org’s BEVENT.EXE utility does not produce these, so they were determined and stored in an external variable called **headers**. The files produced from the above code are **test.hit.csv** (containing the testing data), **train.hit.csv** (containing the training data), and **all.hit.csv** (containing the superset of both data sets).

Initially, **train.hit.csv** and **test.hit.csv** were found to be incompatible in Weka. The cause, it was determined, was that the set of all attributes in some of the columns was not the same. It seems that Weka needs to know the set of all nominal attributes in both the training and testing data sets. In order to rectify this, the training, testing and all CSV files were converted to ARFF files. All ARFF files, in their headers, list the sets of all nominal attributes. So, the total set of nominal attributes was taken from the all ARFF set and copied into the training and testing ARFF files. This solved the issue and allowed both the training and testing data sets to be successfully imported into Weka.

**Tools**

**Python**

Python is a very approachable programming language conveniently used for honestly most things these days. It is becoming increasing popular in the data engineering and data science communities these days particularly with the help of the modules **pandas,** for data manipulation and analysis, and **scikit-learn**, for machine learning. In this project the pandas module was used for data manipulation to read from CSV files, drop unneeded columns and manipulate a particular column to create our class attribute. This was performed 3 times in order to create data sets for training, testing and the superset of training and testing data.

**Appendix A**

**Attribute Definitions**

**game id (0->0)**:  Game ID following the format described in the "data.doc" file.

**visiting team (1->1)**: The team that has traveled out of its home area to play the home team (Boston Red Sox, in this case).

**inning (2->2)**:  Inning in which this play took place.

**batting team (3->3)**:  A one-character identification of the team at bat ("0" for the visiting team and "1" for the home team).

**outs (4->4)**:  Number of outs before this play.

**balls (5->5), strikes (6->6), *pitch sequence (7)***:  These three consecutive fields present the pitch information for this play.

**vis score (8->7)**:  Number of runs for the visiting team before this play.

**home score (9->8)**:  Number of runs for the home team before this play.

**batter (10->9)**:  Player ID code for the batter.

**batter hand (11->10)**:  One character which describes how the batter batted for this event (L or R).

**res batter (12->11) and res batter hand (13->12)**:  These fields are almost always the same as batter and batter hand.  They only differ if the batter is replaced during the time at bat and the final event is charged to the previous batter.  For example, if a pinch-hitter is inserted with two strikes and then takes strike three, the strikeout is charged to the first batter (the responsible batter)

**pitcher (14->13)**:  Player ID code for the pitcher.

**pitcher hand (15->14)**:  The hand with which the pitcher throws (L or R).

**res pitcher (16->15) and res pitcher hand (17->16)**:  Counterparts to res batter and res batter hand for those occasions when a pitcher is changed during an at-bat and the first pitcher is charged with the result.  For example, if a relief pitcher enters with a three-ball, no-strike count and throws ball four, then the walk is charged to the first pitcher.

**positions (18->17, 19->18, 20->19, 21->20, 22->21, 23->22, 24->23, 25->24)**:  The next eight fields contain the Player ID codes for the players at each of the eight fielding positions, in numerical sequence by position number.

**first runner (26->25), second runner (27->26), third runner (28->27)**:  These three consecutive fields contain the Player ID codes for the runner at each base.  If a base is not occupied, then the field has no width and there will be a pair of double quotes with no space between them.  For example, Bill Ripken on first as the only runner would look like this:

               "ripkb001","","",

With Joe Orsulak on first and Cal Ripken on third, these fields would look like:

              "orsuj001","","ripkc001"

***event text (29)****:  The complete description of the play using the format described for the event files.*

**leadoff flag (30->28):**  A one-character descriptor which is T for the first batter of each inning and F for all others.

**pinch hit flag (31->29)**:  Another one-character flag which is T for pinch-hitters and F for all others.

**defensive position (32->30)**:  The defensive position currently being played by this batter.  It is pinch-hitter (position 11) for pinch-hitters.

**lineup position (33->31)**:  Position in the batting order for this batter.

***event type (34)****:  There are 24 different numeric codes to describe the type of event.  They are:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *0* | *Unknown event* | *8* | *Pickoff* | *16* | *Hit by pitch* |
| *1* | *No event* | *9* | *Wild pitch* | *17* | *Interference* |
| *2* | *Generic out* | *10* | *Passed ball* | *18* | *Error* |
| *3* | *Strikeout* | *11* | *Balk* | *19* | *Fielder's choice* |
| *4* | *Stolen base* | *12* | *Other advance* | *20* | *Single* |
| *5* | *Defensive indifference* | *13* | *Foul error* | *21* | *Double* |
| *6* | *Caught stealing* | *14* | *Walk* | *22* | *Triple* |
| *7* | *Pickoff error* | *15* | *Intentional walk* | *23* | *Home run* |

***batter event flag (35)****:  One character indication of whether or not the event terminated the batter's appearance. T = yes, which is most common; F = no, meaning the same batter stayed at the plate, such as after a stolen base.*

***ab flag (36)****:  One character indication of whether batter was charged with at-bat (T = yes, F = no).*

***hit value (37)****:  One number indicating value of hit (0 = no hit; 1 = single; 2 = double; 3 = triple; 4 = home run).*

***SH flag (38)****:  One character indicating sacrifice hit (T = yes; F = no).*

***SF flag (39)****:  One character indicating sacrifice fly (T = yes; F = no).*

***outs on play (40)****:  Number of outs recorded on this play.*

***double play flag (41)****:  One character indicating DP or not.*

***triple play flag (42)****:  One character indicating TP or not.*

***RBI on play (43)****:  Number of RBI credited to batter on this play.*

***wild pitch flag (44), passed ball flag (45)****:  Two records with indication of whether there was a WP or PB on this play.*

***fielded by (46)****:  Identity of the fielder who played the ball. This is especially important for base hits when no formal fielding credit is given.*

***batted ball type (47)****:  Descriptor which is either F (fly ball), L (line drive), P (pop-up), or G (ground ball).*

***bunt flag (48)****:  Descriptor for whether or not play was a bunt.*

***foul flag (49)****:  Descriptor for whether or not ball was played in foul ground.*

***hit location (50)****:  The zone on the field where the ball was hit. Refer to the Scoring System attachments for a diagram of all locations.*

***num errors (51):*** *Number of errors on this play (a maximum of three is allowed).*

***error players and types (52, 53, 54, 55, 56, 57)****.  These are 6 consecutive fields which identify the player committing the 1st, 2nd or 3rd errors on the play and the type of error each was (throw or drop).*

***batter dest (58)****:  The base which the batter reached at the conclusion of the play.  If he was out, the base is 0.*

***runner dest (59, 60, 61)****:  The next three fields contain the base reached by each of the three runners at the conclusion of the play. If there was no advance, then the base shown will be the one where the runner started.  Note that these runner fields are not updated on plays which end an inning, even if the inning-ending play would have resulted in an advance of one or more runners had it occurred earlier in the inning.*

***plays (62, 63, 64, 65)****:  The next four fields indicate the play (if any) made on the batter and each of the runners (if any).*

***SB, CS, PO flags (66, 67, 68, 69, 70, 71, 72, 73, 74)****:  The next nine fields contain single character descriptors for each of the runners indicating whether he had a stolen base, was caught stealing or was picked off.*

***responsible pitcher for runner (75, 76, 77)****:  The next three fields indicate which pitcher was responsible for the runners on each base, if any.  This assignment reflects responsibility should the runner score.*

**new game and end game flags (78->32, 79->33)**:  The next two fields set a flag if this is the first record of a new game or the last record of the game.

**pinch-runners (80->34, 81->35, 82->36)**:  The next three fields indicate if a pinch-runner has entered the game and at which base.

**removed runners (83->37, 84->38, 85->39)**:  The next three fields contain the player ID of the runner who was just run for, one field for each base. If there is no pinch-runner at that base, the field contains the NULL string "".

**removed batter (86->40)**:  If there is a pinch-hitter, this field contains the player ID of the batter removed. If there is no pinch-hitter, this field contains the NULL string "".

**removed batter position (87->41)**:  If there is a pinch-hitter, this field contains the fielding position of the removed batter. If there is no pinch-hitter, this value is 0.

***fielder putouts (88, 89, 90)****:  The next three fields indicate the first, second, and third fielders credited with putouts on the play.*

***fielder assists (91, 92, 93, 94, 95)****:  The next five fields indicate which fielders got credited with assists on the play (maximum of five fielders).*

**event num (96->42)**:  All events are numbered consecutively throughout each game for easy reference.

**hit flag (43)**: One character field indicating if there was a hit resulting from the play. This field is one converted from the hit value field above.