MFB Play-by-Play Stat Extraction Module

User Documentation

1. Introduction

This module was developed as a text mining process to extract stats from the play-by-play text for Men's Football.

In general, the process has the following steps:

- Scanned, typewritten PDFs of play-by-play records for a game provided by sports team.
- Optical Character Recognition (OCR) of PDFs by AWS into text files.
- Cleaning and simple standardizing on the text file by Jaisys.
- Ruled by various data models, contents in the cleaned files are analyzed line by line:
    - Text cleaning and standardizing;
    - Sentence categorization;
    - Player name locating and matching;
    - Stat extraction on sentence/play level;
    - Algorithm based stat validation and correction.
- Play-by-play stats are aggregated on player and team level.
- XML, if already manually inputted using Statscrew, are parsed to provide player/team stats validation.
- The output was organized to a template designed to let Business Analysts quickly manually modify the extracted stats.
- The finalized stats are re-organized into XML format and stored inside Athlyte's database.

The module consists of 2 steps, each with 1 object to do game-level functions and 1 object to do line-level functions. That is, say for a game with 200 lines, the algorithm will create 1 game-level object and 200 line-level objects for each step. The game-level object provides some vital information for all 200 line-level objects, and also stores them after they each are analyzed. A brief description of the objects is displayed below.

Step	Object Name	Description
Step 1	OneGame	• Read in text file, • Obtain game roster from MongoDB, • Create OneLine objects and analyze each of them • Output step one results to be used in step 2
	OneLine	• Preprocess the raw sentence • Recognize, separate, and parse the context of the play, if the context exists • Look for patterns of actions[1] and results[2] • Categorize a sentence into a specific “sentence type[3]” • Find patterns in the sentence and map the sentence to a “ParentName[4]”
Step 2	TwoGame	• Read in the output from Step One • Obtain game roster from MongoDB • Create TwoLine objects and analyze each of them, • Aggregate all TwoLine objects by players and teams • Obtain validation for aggregated stats from existing XML for the same game • Organize all extraction results according to the review template
	TwoLine	• Parse the context • Locate player names from the text • Assign a role to each player names according to the sentence’s “ParentName” • Extract stats for each player given the respective role • Validate and correct the extracted stats by considering surrounding sentences • Organize the stats to prepare for aggregation and output • Generate alerts for Business Analysts’ review

[1] actions: A standardized expression of an action performed by players. For example, “dashed”, “sprinted”, “run” are matched as the standard action “rush”, and “handed”, “throw”, “find” are matched as “pass”.

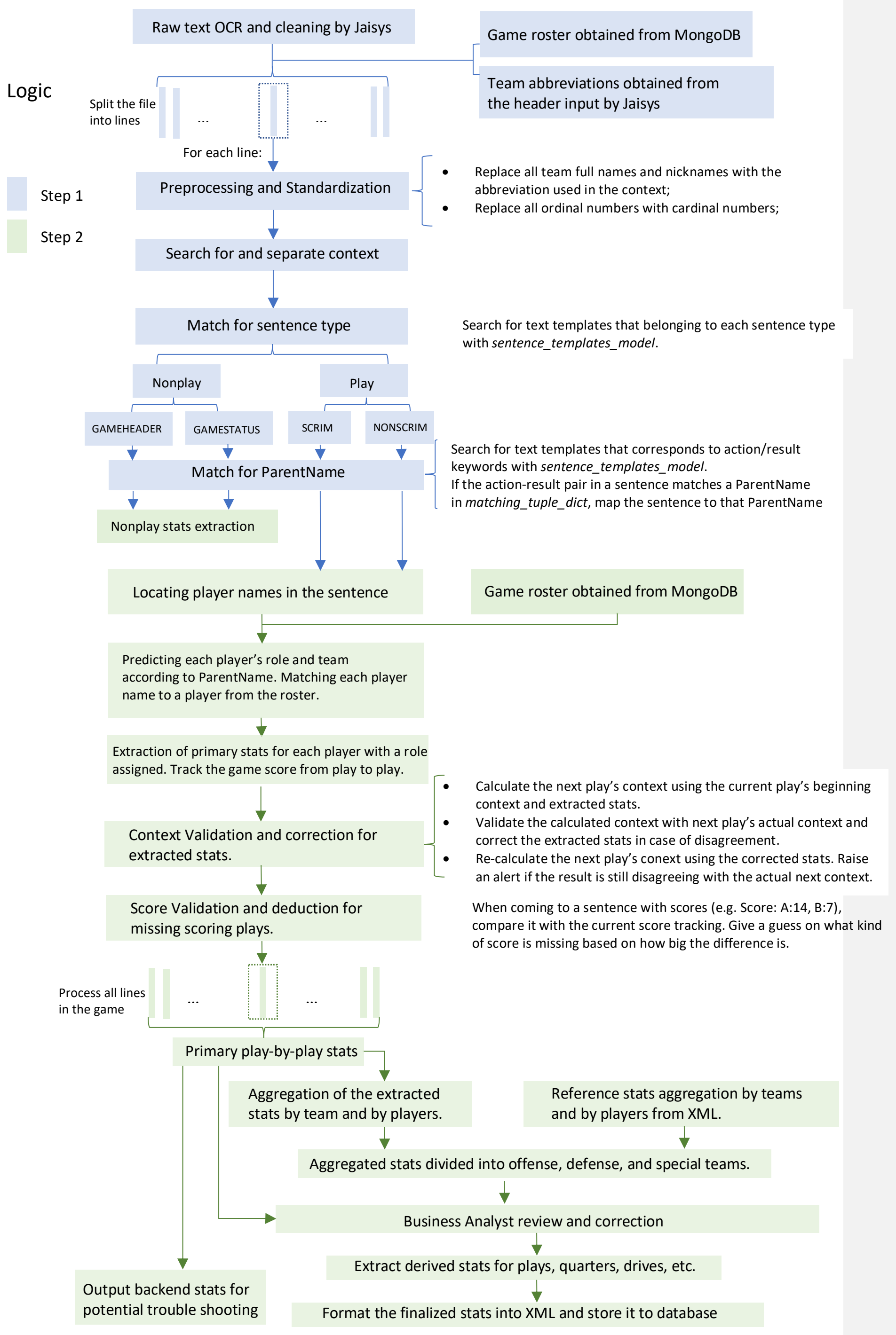
[2] results: A standardized expression of the result of an action. For example, “good”, “successful” are matched as the standard result “completed”, and “td”, “touchdown” are matched to the result “touchdown”.

[3] sentence type: Play / Nonplay. Play is further divided into Scrimmage and Non-scrimmage. Nonplay is further divided into GameStatus and GameHeader.

[4] ParentName: Subordinating sentence categories under each sentence type to reflect the actions and results existing in the sentence. E.g. Under Scrimmage, there are RushSimple, RushDowned, RushFumble etc for rush plays.

** For a more detailed list of sentence types and ParentNames, please see Appendix I.

2. Logic



Raw text OCR and cleaning by Jaisys
Game roster obtained from MongoDB
Team abbreviations obtained from the header input by Jaisys
Split the file into lines
…
…
For each line:
Step 1
Step 2
Preprocessing and Standardization
• Replace all team full names and nicknames with the abbreviation used in the context;
• Replace all ordinal numbers with cardinal numbers;
Search for and separate context
Match for sentence type
Search for text templates that belonging to each sentence type with *sentence_templates_model*.
Nonplay
Play
GAMEHEADER
GAMESTATUS
SCRIM
NONSCRIM
Match for ParentName
Search for text templates that corresponds to action/result keywords with *sentence_templates_model*.
If the action-result pair in a sentence matches a ParentName in *matching_tuple_dict*, map the sentence to that ParentName
Nonplay stats extraction
Locating player names in the sentence
Game roster obtained from MongoDB
Predicting each player's role and team according to ParentName. Matching each player name to a player from the roster.
Extraction of primary stats for each player with a role assigned. Track the game score from play to play.
Context Validation and correction for extracted stats.
• Calculate the next play's context using the current play's beginning context and extracted stats.
• Validate the calculated context with next play's actual context and correct the extracted stats in case of disagreement.
• Re-calculate the next play's conext using the corrected stats. Raise an alert if the result is still disagreeing with the actual next context.
Score Validation and deduction for missing scoring plays.
When coming to a sentence with scores (e.g. Score: A:14, B:7), compare it with the current score tracking. Give a guess on what kind of score is missing based on how big the difference is.
Process all lines in the game
…
…
Primary play-by-play stats
Aggregation of the extracted stats by team and by players.
Reference stats aggregation by teams and by players from XML.
Aggregated stats divided into offense, defense, and special teams.
Business Analyst review and correction
Extract derived stats for plays, quarters, drives, etc.
Format the finalized stats into XML and store it to database
Output backend stats for potential trouble shooting

3. Data Preparation

3.1 Data cleaning requirements for Jaisys

3.2 File naming routines:

A game is uniquely identified using Home team + Visit Team + game year, i.e. the primary key for a game. The routine is necessary for identifying the correct files to read in and for ensuring the information flow from Step 1 to Step 2.

File	Name template	Example
Game text file	Home Team Vs Visit Team YYYY.txt	*Purdue Vs Ohio State 1996.txt*
Step 1 output file	Home Team Vs Visit Team YYYY.xlsx	*Purdue Vs Ohio State 1996.xlsx*
Step 2 backend stats	loop2_Home Team Vs Visit Team YYYY.xlsx	*loop2_Purdue Vs Ohio State 1996.xlsx*
Step 2 BA review file	BA_Review Home Team Vs Visit Team YYYY.xlsx	*BA_Review Purdue Vs Ohio State 1996.xlsx*

4. Data Models

Data Models contain game rules, relationships between variables, sentence patterns, etc. A brief exhibit of the data models used in the module is below:

Data Model Name	Structure	Usage
sentence_pattern_model	Action/Result{ sentence type{ standard keywords {text patterns}}}}	Identifying what actions/results a sentence contains by searching for respective text patterns. Use the result to predict the sentence type and ParentName.
matching_tuple_dict	ParentName {action-result pair}	If a specific action and a specific result are both found in the same sentence, map the sentence to the respective ParentName.
ParentName_role_model	ParentName {role type {role} }	Tells the algorithm for each ParentName what defense/offense roles should be searched for. Assign the roles to player names in the sentence and predict the player's team.
role_stats_model	Role {stats}	Tells the algorithm for each role what stats to expect.
stat_templates_model	Stats{ text templates}	Tells the algorithm for each stat what text template can possibly exist.

ParentName_context_model	ParentName{ possession change, ending down, ending yards to goal, ending ball spot}	Used for context validation. For each ParentName tells the algorithem the rules to calculate the context of the next play using the extracted stats, which is then compared to the real context of the next play to validate the stats.

The data models are created by python scripts and stored in JSON files. When analyzing the file, the JSON files of data models were loaded into dictionaries to be used. Each time an update is made to any data model, the respective python script should be re-run to make sure the JSON file is also refreshed. Otherwise, the change won't be reflected in the algorithm.

5. OneGame and OneLine objects
   - 5.1 OneGame Object
   - 5.2 OneLine Object
6. TwoGame and TwoLine objects
   - 6.1 TwoGame Object
   - 6.2 TwoLine Object
7. Business Analyst Feedback Process
8. Appendix
   - 8.1 Detailed list of sentence types and ParentNames
   - 8.2 Detailed list of Parent, Roles, and Stats expected

Commented [LC1]: Detailed documentation for all attributes and all methods for the objects.

Commented [LC2]: Describe the structure, design, and usage of the template.