The Oracle Baseball Analytics Project

Abstract

The Oracle baseball analytics project at Johns Hopkins University has as its goal the creation of a software system for the facilitation of complex queries, analysis and visualization regarding Major League Baseball information compiled from the Pitch F/X database. While Pitch F/X provides extensive data for each pitch in a major-league season, the process of extracting meaningful and actionable information typically involves a programming process, making all but the most basic data accessible only to specialists in the field; even for specialists, the process of extracting the information is more laborious and repetitive than it need be. For instance, programming basic baseball concepts such as "inning" and determining if a pitch was likely to be a called strike had the batter not swung are non-trivial tasks at present. Finally, there is likely to be useful information hidden in the raw data which is impossible to discern without a more systematic and automated approach.

The Oracle project aims to resolve these issues by creating a software system that presents the user with a natural-language interface so that the user can pose analytics questions in an intuitive, rapid way and receive answers instantly, presented with appropriate visualizations such as graphs, tables, and so forth. The goal is to leapfrog existing websites that are helpful but offer relatively limited options for obtaining data from Pitch F/X.

The software architecture of Oracle consists of a core and a potentially unlimited number of modules that contain the specialized processing capabilities for generating advanced analytics. The key idea is to open the module development to the broader analytics community to contribute extensive analytics knowledge to Oracle that can be augmented into the foreseeable future.

1. Introduction

The use of data in driving decisions is as clear today as ever before. It permeates organizations in larger volume and variety, empowering those with access to this information the ability to extract answers to pressing questions in real-time. Effective use of this information (for a wider audience) is contingent on the presence of tools that facilitate data discovery in ways that are truly intuitive: approaches rooted in a user's need to implement programming solutions from scratch distance those with the necessary domain knowledge from asking the right questions. Similarly, dashboards for data visualization can equivocate if not supplemented with results that address the user's questions directly. The Oracle is a system that facilitates analysts/fans of any background to generate actionable insight from their data by focusing their intuition on asking the right questions rather than prepping their data and laboring over implementation. Specifically, an opportunity exists for natural language processing to augment the data discovery pipeline. Our research focuses on the development of the Oracle, a system to query data sources in natural language and to return relevant answers via text and data visualizations (**Figures 1,2**). After initializing the system with a corpus of domain knowledge (provided intuitively through a web interface and custom API), the

Oracle prompts a user to ask a question about his or her data, as if a search query, and be guided to an answer.

Whether siloed across heterogeneous sources or incorporated into a unified data view, the architecture is aimed at automating the systematic components of the data discovery pipeline. It does so by leveraging techniques in Natural Language Processing and Information Retrieval. Specifically, the Oracle leverages the fact that user intent, when parsed to its logical form, can be reduced to a sequence of filters and user-specified actions. These core filters compose a finite set that we believe to be relevant to the vast majority of user queries. Ultimately, these reductions can be applied iteratively to produce answers relevant to the user's intent. Much of our research has been focused on the development of the mapping scheme that associates components of a user's query to relevant features/values in the dataset.

As an example, an MLB analyst in the Baltimore Orioles' analytics department could use the Oracle to ask: "What is Zach Britton's strike rate on changeups and four-seam fastballs against lefties on pitches where the start velocity of the pitch is over 90mph?" The user query is parsed into its logical form via finite state transduction. Each component of the query is tagged by a part of speech tagger to isolate subject entities (denoted actors) from filters and verb phrases (denoted actions). These tokens are then run through an algorithm to match a sequence of tokens to the most likely <feature, value> pair in the dataset by comparing this sequence to terms described in the system's domain knowledge (before sequentially applying each filter). The tool's output will be a dashboard containing a textual answer where relevant (such as Britton's actual strike rate over the filtering criteria) in addition to a supplementary dashboard of automated visualizations.

2. Architectural Design

2.1. Backend

The backend for the Oracle is constructed in a piecewise fashion. To facilitate a conversation with a user, it is necessary that the Oracle be endowed with domain knowledge for the application at hand. This domain knowledge is incorporated into the system in the initialization phase. Before deployment in an organization, an analyst is provided an option to input domain knowledge into Oracle. This knowledge defines terms and actions within the context of the data source. For instance, using MLBAM's pitchFx as the underlying database, a user might specify descriptors such as "fastball" or "outs" to match on the "pitch_type" and "o" columns of the dataset. The user could also define domain specific actions such as computing a pitcher's strike rate or a batter's weighted-on-base-average (WOBA). An analyst should be walked through this step in the initialization phase, being asked to map features to domain knowledge and define any relevant functions of the kind described above.

2.2. Module Development

Much of the utility of the Oracle is the manner in which teams can upload domain knowledge to the system. Definitions of functions of interest, such as computing a player's WOBA or filtering out "meaningless game situations", are defined once and published to the Oracle. The publishing mechanics are handled by our custom API, which enables a user to follow a prescribed interface to upload code that defines the operation of interest, and publish their code with an associated lexicon that can be queried against in natural language. As a concrete example, an analyst could define a

specific variant of the computation of a player's on-base-percentage and associate it with the lexicon {OBP, on-base-percentage}. Once published, analysts on their team can all use 'OBP' in a query and the Oracle will carry out the associated computation in a manner consistent with the rest of the query.

2.3. User Interface

In automating a natural language pipeline for querying a collection of data sources, the Oracle's design is divided into several interrelated components. The Oracle will be hosted publicly. As such, our UI is aimed at facilitating a user's ability to ask questions in an intuitive way. At the top of the view is a search box through which the user can query their data. Output, in the form of textual answers that address the user's intent directly, is provided first. Automated visualizations are output in a chat-like interface below these responses. The output can be thought of as a log in which users search on queries made in the past and favorite insights generated by the Oracle to their own dashboard. This log is cumulative and so an analyst can scroll up and down reviewing their thought processes as they generated questions relevant to the problem at hand. The "conversation" between the analyst and Oracle can be shared among others in his or her analytics team.

The associated visualizations generated by the Oracle are populated according to a ranked dashboard. The code for ranking visualizations via a RankSVM is currently functional. Derived features associated with each visualization (including variation along a primary dimension such as length, angle, or area and a plot interpretability score) are passed to this ranking framework to order each visual by its predicted utility to the user and relevance to the query. The dashboard is incorporated inline. Arrows to the right and left of an inline plot allow the user to toggle through each visualization in the ranked dashboard.

3. Natural Language Processing

The majority of our work on the Oracle has went into automating the Oracle's natural language pipeline. The current approach employs finite state transduction in NLP to reduce the user query to a sequence of actions, filters, and entities. Taken in reverse, an entity is defined to be the subject-like components of the user query, filters defined to be a finite set of keywords (and their synsets) intended to "slice and dice" the dataset, and actions defined to be any non-filter verb phrases. Common filters include words such as "by", "on", "in," etc.

In order to parse the query to this logical form, a part of speech tagger tags each component of the query to isolate these subject entities from filters and actions. As mentioned above, these tokens are run through an algorithm to match a sequence of tokens to the most likely <feature, value> pair in the dataset by comparing this sequence to terms described in the system's domain knowledge. Evidence is accumulated for each feature based on its similarity token-by-token to a given part of the user query. The current algorithm tokenizes each component of the query (in its logical form) before comparing it against tokenized elements in the system's domain knowledge. A notion of string similarity is defined to compare each token (in each component) of the query against these tokens in the system's domain knowledge. The current implementation uses the jaro distance to make these comparisons, computing a score between 0 and 1 depending on the extent to which one string is similar to another. The feature with the maximum similarity (after evidence accumulation) is selected before filtering the dataset on this criterion.

4. Machine Learning and Relevance Feedback for Automated Query Suggestion

A sub-problem within Oracle exists in the ability to churn out potentially relevant analyses for a user. From an information retrieval (IR) perspective, the user is prompted with relevant insights that could guide the kinds of analyses they are conducting. These suggestions are incorporated into the user interface as chat-like messages that link to more detailed views for the analyst to peruse if interested. These insights are generated in a fashion similar to relevance feedback in the IR setting. As the user's intent transforms over time, Oracle has access to the terms being queried on and the relationship between entities that are often compared against one another across the organization. As such, the system builds a running log of potential queries of utility to the user by considering analyses relevant to ones the user has logged up until that point. This relevance feedback is currently implemented by the Rochio algorithm. In essence, relevance feedback for Oracle provides a way to make the process of generating queries more two-directional, prodding a user with divergent ways to think of their problem while simultaneously answering their questions.

5. Conclusions and Future Work

The Oracle baseball analytics project had demonstrated that complex queries, analysis and visualization can be readily accomplished in the context of Major League Baseball information compiled from the Pitch F/X database. The goal is to create a community of users that contribute innovative analytics modules to enrich the scope of Oracle.

Following refinement of the core and development of a set of basic, or pilot, modules, Oracle will be made available to the public to encourage usage and module development to add to the Oracle library. In addition, it is anticipated that Oracle will be open-source so that organizations have the ability to bring Oracle in-house for proprietary queries and module development.

Future work includes creating a "crunch mode" in which Oracle continually analyzes billions of combinations of conditions to identify facts that it has been taught are interesting to report back. For instance, a near-deterministic event such as a hitter performing extremely well (or poorly) under an unintuitive combination of circumstances could be detected and reported by Oracle.

In the future, a causal analysis module will be included to determine if there is a probably cause-effect relationship for such an event.

Finally, the team will investigate applications of the Oracle approach and software to other areas of interest beyond sports.

Snapshots of the Oracle Data Discovery Pipeline

```
Query your data: what is the on-base percentage of lefty batters on fastballs or curveballs against oriole pitchers o
n pitches where the start velocity of the pitch is over 90, the horizontal movement of the pitch is under 12, height
is over 2
[['(on-base percentage)', 'NN'], ['of', 'IN'], ['(lefty batters)', 'NNS'], ['on', 'IN'], ['(fastballs)', 'NNS'], ['or
', 'CC'], ['(curveballs)', 'NNS'], ['against', 'IN'], ['(oriole pitchers)', 'NNS'], ['on', 'IN'], ['(pitches)', 'NNS'
], ['where', 'WRB'], ['(start velocity)', 'NN'], ['of', 'IN'], ['(pitch)', 'NN'], ['over', 'IN'], ['(90)', 'CD'], ['(
horizontal movement)', 'NN'], ['of', 'IN'], ['(pitch)', 'NN'], ['under', 'IN'], ['(12)', 'CD'], ['(height)', 'NN'], [
'over', 'IN'], ['(2)', 'CD']]
(lefty batters)->(on-base percentage)=>*(%filter%)((fastballs)OR(curveballs))=>*(%filter%)((oriole pitchers))=>*(%fil
ter%)((pitches))=>*(%filter%)((start velocity))=>*(%over%)((90))=>*(%filter%)((pitch))=>*(%filter%)((horizontal movem
ent))=>*(%under%)((12))=>*(%filter%)((pitch))=>*(%filter%)((height))=>*(%over%)((2))
filtered:       ['=>*(%filter%)((fastballs)OR(curveballs))', '=>*(%filter%)((oriole pitchers))', '=>*(%filter%)((pitc
hes))', '=>*(%filter%)((start velocity))', '=>*(%over%)((90))', '=>*(%filter%)((pitch))', '=>*(%filter%)((horizontal
movement))', '=>*(%under%)((12))', '=>*(%filter%)((pitch))', '=>*(%filter%)((height))', '=>*(%over%)((2))']
entity:         ['(lefty batters)', '->', '(on-base percentage)']

feature association: most relevant feature is pitch_type
(pitch_type, fastballs): match_arg_to_feature_value
generate_criteria: ['FA', 'FF', 'FT', 'FC', 'FS']
(pitch_type, curveballs): match_arg_to_feature_value
generate_criteria: ['CB', 'CU']

feature association: most relevant feature is team_id_pitcher
(team_id_pitcher, oriole pitchers): match_arg_to_feature_value
generate_criteria: ['balmlb']

feature association: most relevant feature is start_speed

feature association: most relevant feature is pfx_x

feature association: most relevant feature is pz
(pitch_type, ['FA+FF+FT+FC+FS+CB+CU']): filter_

config.filtered = config.filtered[(config.filtered['pitch_type'] == 'FA') | (config.filtered['pitch_type'] == 'FF') |
(config.filtered['pitch_type'] == 'FT') | (config.filtered['pitch_type'] == 'FC') | (config.filtered['pitch_type'] ==
'FS') | (config.filtered['pitch_type'] == 'CB') | (config.filtered['pitch_type'] == 'CU')]
(team_id_pitcher, ['balmlb']): filter_

config.filtered = config.filtered[(config.filtered['team_id_pitcher'] == 'balmlb')]

config.filtered = config.filtered[(config.filtered['start_speed'] > 90)]

config.filtered = config.filtered[(config.filtered['pfx_x'] < 12)]

config.filtered = config.filtered[(config.filtered['pz'] > 2)]
(stand, lefty): match_arg_to_feature_value
(stand, batters): match_arg_to_feature_value
(stand, ['L']): filter_

config.filtered = config.filtered[(config.filtered['stand'] == 'L')]
(stand, ['L+R']): filter_

config.filtered = config.filtered[(config.filtered['stand'] == 'L') | (config.filtered['stand'] == 'R')]

Oracle's response: 0.3385918784691791

(Relevance Feedback) Investigate Features More Like This: ['pz', 'pfx_x', 'start_speed', 'team_id_pitcher', 'pitch_type']
```

Figure 1.

The snapshot above is sample output of our system on the query: “What is the on-base-percentage of lefty batters on fastballs or curveballs against oriole pitchers on pitches where the start velocity of the pitch is over 90, the horizontal movement of the pitch is under 12, and the height of the pitch is over 2?” The query is reduced via finite state transduction such that each token can be mapped to the most likely feature/filter/operation it was associated to. For instance, domain knowledge associated the values FA,FF,FT,FC,FS in the pitch_type field to the descriptor “fastball” as well as the values CB,CU in the pitch_type field to the descriptor “curveball”. The appropriate query conditions are evaluated by the Oracle and transformed into a filtering operation over the underlying table. The output 0.338591878 is returned as the answer to the user’s query, alongside a list of potential features to explore based on the user’s past query history.

```
Query your data: what is the strike rate of nationals pitchers by pitch type
[['(strike rate)', 'NN'], ['of', 'IN'], ['(nationals pitchers)', 'NNS'], ['by', 'IN'], ['(pitch type)', 'NN']]
(nationals pitchers)->(strike rate)=>*(%by%)((pitch type))
filtered:       ['=>*(%by%)((pitch type))']
entity:         ['(nationals pitchers)', '->', '(strike rate)']

feature association: most relevant feature is pitch_type
(team_id_pitcher, nationals): match_arg_to_feature_value
(team_id_pitcher, pitchers): match_arg_to_feature_value
(team_id_pitcher, ['wasmlb']): filter_
config.filtered = config.filtered.apply(lambda g: (g[g['team_id_pitcher'].isin(['wasmlb'])]))
(team_id_pitcher, ['tormlb+texmlb+detmlb+anamlb+oakmlb+wasmlb+nynmlb+kcamlb+nyamlb+arimlb+balmlb+sdnmlb+pitmlb+milmlb
+seamlb+minmlb+slnmlb+lanmlb+chamlb+tbamlb+colmlb+stlmlb+sfnmlb+phimlb+chnmlb+cinmlb+miamlb']): filter_
config.filtered = config.filtered.apply(lambda g: (g[g['team_id_pitcher'].isin(['tormlb', 'texmlb', 'detmlb', 'anamlb
', 'oakmlb', 'wasmlb', 'nynmlb', 'kcamlb', 'nyamlb', 'arimlb', 'balmlb', 'sdnmlb', 'pitmlb', 'milmlb', 'seamlb', 'min
mlb', 'slnmlb', 'lanmlb', 'chamlb', 'tbamlb', 'colmlb', 'stlmlb', 'sfnmlb', 'phimlb', 'chnmlb', 'cinmlb', 'miamlb'])]
))
```

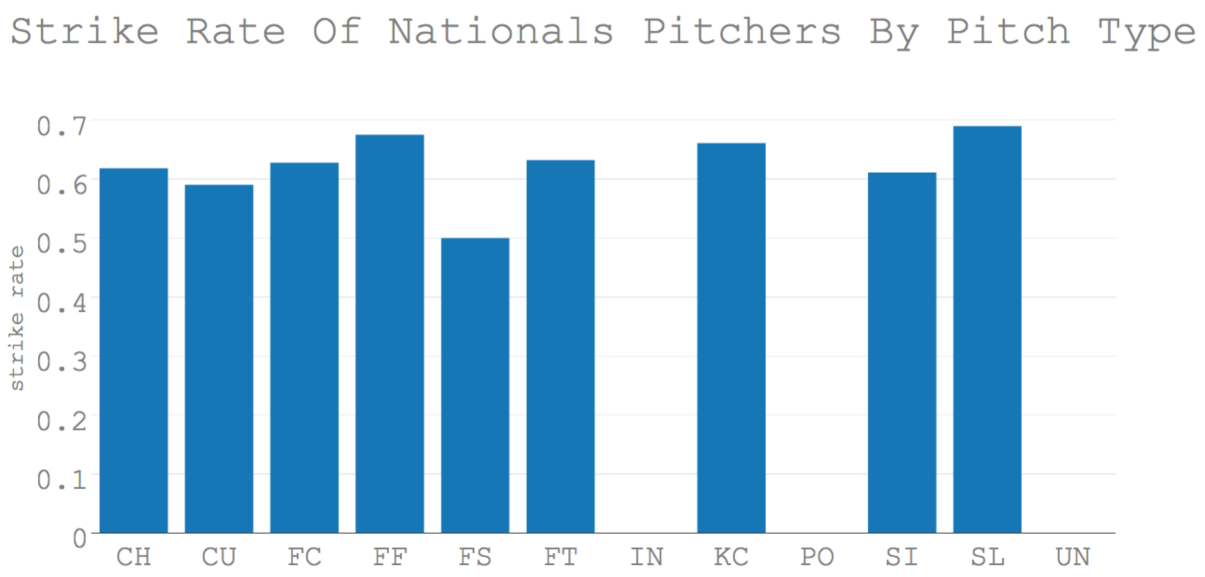

Figure 2. The Oracle generated an automated visualization associated with the query for the user's query on the distribution of National's pitchers pitch types.