Large-Scale Adversarial Sports Play Retrieval with Learning to Rank

MINGYANG DI and DIEGO KLABJAN, Northwestern University
LONG SHA, Queensland University of Technology
PATRICK LUCEY, STATS LLC

As teams of professional leagues are becoming more and more analytically driven, the interest in effective data management and access of sports plays has dramatically increased. In this article, we present a retrieval system that can quickly find the most relevant plays from historical games given an input query. To search through a large number of games at an interactive speed, our system is built upon a distributed framework so that each query-result pair is evaluated in parallel. We also propose a pairwise learning to rank approach to improve search ranking based on users' clickthrough behavior. The similarity metric in training the rank function is based on automatically learnt features from a convolutional autoencoder. Finally, we showcase the efficacy of our learning to rank approach by demonstrating rank quality in a user study.

CCS Concepts: • **Information systems** → **Information retrieval**; **Retrieval models and ranking**; **Learning to rank**;

Additional Key Words and Phrases: Similarity measures, learning to rank, convolutional autoencoder

ACM Reference format:
Mingyang Di, Diego Klabjan, Long Sha, and Patrick Lucey. 2018. Large-Scale Adversarial Sports Play Retrieval with Learning to Rank. *ACM Trans. Knowl. Discov. Data* 12, 6, Article 69 (August 2018), 18 pages.
https://doi.org/10.1145/3230667

1 INTRODUCTION

As sports games are more and more analytically driven, the collection of sports tracking data has quickly become the norm. For instance, STATS' SportVU basketball tracking system generates location coordinates for every player, ball, and referee 25 times per second along with detailed logs of events such as shots, passes, fouls, and so on. However, given the overwhelming amount of data being collected, a large-scale information retrieval system that can effectively access and rank sports plays has not yet surfaced. To close this gap, we present a retrieval system that can quickly find and prioritize similar basketball plays from a large number of games (e.g., one season or multiple seasons). With our system, finding a play of interest with certain characteristics no longer relies on experts watching video footage and judging by domain knowledge. Once an input query is issued, the system is able to return a ranked list of similar plays ordered by the relevance

Authors' addresses: M. Di and D. Klabjan, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60202; emails: mingyangdi2012@u.northwestern.edu, d-klabjan@northwestern.edu; L. Sha, 2 George St, Brisbane City QLD 4000, Australia; email: sharon81818@gmail.com; P. Lucey, 203 North LaSalle St., Suite 2200, Chicago, IL 60601; email: plucey@stats.com.

https://doi.org/10.1145/3230667

to the query at interactive speeds. It is by nature a search engine in the field of sports, and although we focus on basketball, it is straightforward to extend the context to other sports like soccer or American football.

An immediate question that arises from the development of such a search engine is how to design a query format that is both intuitive to use and rich enough to capture the spatiotemporal nature of basketball plays. Unlike traditional search engines (e.g., Google and Yahoo) that use keyword queries, it is almost impossible to specify basketball plays with keywords. On one hand, if we use simple keywords like "three-point shot," the number of retrieved plays can easily reach several thousand or more. Users still have to browse through a large collection of results before finding the specific targets. On the other hand, if we try to describe a play in more detail – for instance, how each player and the basketball are moving and where and when the three-point shot is made – using keywords would be very tedious. Therefore, in this article, we employ a user interface (UI) that allows users to issue an input query either by selecting a play through browsing previous games or by drawing a play of interest on an interactive mobile device similar to how coaches draw plays on white board. Similar UI was initially proposed in [30]. However, they only provide users with the 2D visualizations of player and basketball trajectories (e.g., Figures 8–10), while on our interface users can watch a real game.

Another challenge is the vast amount of data the system has to store and process in real time. For example, SportVU generates over 100GB of tracking data for a season of games, and building a database of multi-agent trajectories with each trajectory consisting of hundreds of tracking records can easily reach several terabytes. In this article, a trajectory is defined as a spatial trait of a player or ball over a short period of time, and a play is simply a collection of trajectories of one or more players and the ball. Our system is designed to handle multiple seasons through a carefully designed database model on top of key-value databases. To get around the scalability challenge, we propose an encoded representation of basketball plays based on k-means clustering and player's role alignment. In particular, we perform k-means clustering separately on ball, offensive player and defensive player trajectories and replace each trajectory in a play with the index value of the closest centroid. This representation dramatically reduces the amount of data needed to be stored and also enables fast identification of candidate plays. A *key-value* database using the ball trajectory index as a primary search key is constructed to store the encoded plays, and a streaming-based distributed framework is developed to process candidate plays at scale to achieve real time performance. Given an input query, we first compare its ball trajectory against the centroids of ball trajectories and acquire all the plays that resemble the query at least from the ball trajectory perspective. We then compare each candidate play to the query in parallel and decide the top candidates to be returned to the user. Such parallelism (scalability) is one of the unique aspects that distinguishes our system from any previous work.

Finally, a good search engine would personalize search ranking by mining each user's search log and clickthrough behavior. Similar functionality is achieved on our system through a pairwise ranking approach – rankSVM, and an advanced feature representation of basketball plays learnt via a convolutional neural network (CNN)-based autoencoder is used in training the ranking model. As to be elaborated, the user-specific ranking is another novelty of our system.

In the numerical section, we show that the response time of our system is around 5–7 seconds once an input query is issued. Considering the granularity of our data and the (extremely large) number of plays being processed, it is a remarkable speed according to domain experts' opinions. We demonstrate the effectiveness of our learning to rank approach by evaluating the predicted rankings using real user feedback. Experiments on how to learn the encoded representation are also presented.

The contributions of this work are as follows:

(1) It is the debut of the first information retrieval system (search engine) in the domain of sports that can process industry-sized data. The concept and algorithms can be easily extended to a variety of other sports such as soccer, ice hockey, and American football.
(2) We present a novel quantization method of multi-agent sports plays by using clustering-based indices. The associated key-value database leads to a promising direction of accessing and managing overwhelmingly large sports tracking data.
(3) This is the first effort to apply learning to rank to sports play retrieval. It makes our system more adaptive to user's specific interest and significantly improves rank quality.
(4) Finally, we use convolutional autoencoder to extract features from sports tracking data. The proposed CNN model unveils the dynamic, hidden structure of basketball plays and could potentially be applied to analyze, predict and simulate sports games.

The rest of this article is organized as follows. In Section 2, we discuss the related literature. Section 3 illustrates the major components of the system, and Section 4 gives implementation details and the findings from our computational experiments. Finally, we conclude in Section 5.

2 RELATED LITERATURE

Previous work on information access in the sports domain largely focuses on improving categorization of plays [4, 10, 29]. Sha et al. [30] develop the first sports play retrieval system that allows users to issue an input query by drawing a play of interest. Through alignment, templating, and hashing techniques tailored for multi-agent trajectories, the system is able to search the most relevant plays from a few dozen games. Although our system employs a similar query paradigm that allows users to either select or draw a play, the core parts are completely different. With a key-value database that stores the encoded plays and a distributed architecture, we are able to retrieve from multiple seasons at interactive speeds. Sha et al. [30] use only ball trajectory clustering, while we also perform clustering on player trajectories to cope with scalability [20, 23]. Furthermore, we apply deep learning techniques to obtain an advanced similarity metric, and use it in training a user-specific ranking model. This is definitely not addressed in [30].

There are plenty of previous works studying how to measure distance/similarity between trajectories [9, 12, 33, 39, 40], but the majority of them focus on comparing single trajectories rather than multi-agent ones. In our learning to rank approach, we compare 2 plays, each consisting of 11 trajectories (10 for players and 1 for ball), based on features learnt through a CNN-based autoencoder. While in literature neural networks have been widely used in sports prediction [15, 25, 27, 36] or classification [3, 34], we have not yet seen it used for measuring similarity of sports plays. The features revealed by the CNN preserve more information than the Euclidean distance based on role alignment [5, 26, 35], and thus reveal more of the structural essence of basketball plays.

Modern search engines all utilize clickthrough data to improve rank quality. Recent works by Joachims et al. [16, 17, 18] and others [1, 2] have shown the value of incorporating implicit user feedback into the ranking process. Learning to rank is widely employed in a variety of applications in information retrieval, natural language processing, and data mining [21, 24]. Solution methodologies include pointwise approaches [11, 22, 31], pairwise approaches [6, 7, 13, 16], and listwise approaches [8, 32, 37, 38]. In this article, we follow a pairwise approach, namely rankSVM, to train a linear classifier using both the features from our CNN-based autoencoder and self-crafted features.

3 A SEARCH ENGINE OF BASKETBALL PLAYS

In this article, we use over 1,100 games of tracking data from a professional basketball league. The data was collected by the SportVU tracking system, and contains location coordinates for every

player, ball, and referee along with detailed log information, such as team and player identifications (ID), time, and events. To develop a search engine that can effectively retrieve from this large number of games, we first extract all the plays from the tracking data. A play is defined as trajectories of one or more players and the ball that are organized to achieve a certain outcome (e.g., a shot made or free throw attempt) within a short period of time. Similar to [30], in this article, we only consider plays with time-windows of 1, 2, 3, 4, and 5 seconds. We then encode each play with 11 indices based on k-means clustering and player's role alignment. A key-value database is constructed to group the encoded plays based on ball trajectory similarity. That being said, once an input query is issued, we can quickly identify all the candidate plays that possess similar ball trajectories to the query. With a distributed framework, each candidate is compared to the query in parallel and once the top results are determined, we apply a user-specific rank function to them before displaying to users. The user-specific rank function is learnt via a pairwise learning to rank approach and an advanced similarity metric leveraging deep learning techniques.

In this section, we first propose the encoded representation and a key-value database to store the encoded plays. A streaming-based retrieval process built upon a distributed framework is then elaborated. Finally, we discuss how to learn a similarity metric via a CNN-based autoencoder and how to train a user-specific rank function based on it.

3.1 An Encoded Representation of Basketball Plays

We first discuss the methodology to create an encoding of basketball plays. The algorithm is summarized in Algorithm 1 where (x_{ie}^{j}, y_{ie}^{j}) represents the x and y coordinates of agent e (in basketball it corresponds to each one of the 10 players and the ball) in game j at frame i. We assume that each game has T frames (for simplicity of writing the algorithm we assume in the algorithm that each frame corresponds to 1 second).

We first extract all the plays with time-windows of 1, 2, 3, 4, and 5 seconds from the tracking data (step 5 in the algorithm). To do this, we use each second as the starting point and get the multi-agent trajectories for various time-windows. This gives us over 15 million plays (over 3 million for each time-window) for a season of games, and an unnecessarily large database to store them as consecutive location coordinates.[1] Therefore, the following encoded representation is proposed to dramatically reduce the amount of data needed to be saved and facilitate fast access to the potential candidates. The first step is to match the player trajectories of each play to 10 unified roles (i.e., offensive PG, SG, SF, PF, and C;[2] defensive PG, SG, SF, PF, and C) using the role alignment algorithm proposed in [5, 35]. This is captured by step 7 in the algorithm where function *Roles* takes a play as input and it returns a mapping between the players and the roles (we assume that the function always returns ball as ball). This role-based representation reveals the intrinsic nature of basketball plays regardless of players' identities and positions, and thus leads to more accurate retrieval. Figure 1 illustrates the intuition behind the role-based representation. In 1(A), Kyrie Irving (PG) and Kevin Love (PF) are running classic pick-and-roll, and in 1(B), LeBron James (SF) and Tristan Thompson (C) are running an almost identical play. If we align the trajectories based on player's identity or position, they would be deemed as very different plays. However, if we assign both Kyrie and LeBron to the offensive point guard role and Love and Thompson to the offensive power forward role, the two plays would be considered similar.

[1]Each tracking record is saved multiple times. For example, two consecutive 5-second plays have 80% overlap in terms of (x, y) coordinates, and a 3-second play completely overlaps with the 5-second one starting at the same time frame.
[2]Basketball terminology: PG – Point Guard; SG – Shooting Guard; SF – Small Forward; PF – Power Forward; and C – Center.

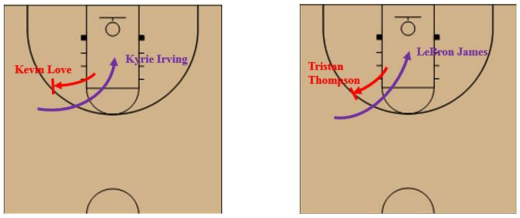

Fig. 1. (A) Kyrie Irving and Kevin Love are running pick-and-roll; (B) LeBron James and Tristan Thompson are executing an almost identical play.

ALGORITHM 1: Construction of plays

Result: t-second plays D^t
/* for all games */
1 **for** *each game* j **do**
2 $D^t = \emptyset$
  /* for all starting frames in game */
3 **for** $i = 1$ **to** T **do**
    /* for all time windows */
4 **for** $t = 1$ **to** 5 **do**
      /* S is a vector corresponding to all of the coordinates of all players and the ball in a play */
5 $S = (x^j_{i+u,e}, y^j_{i+u,e})^t_{u=0,e}$
6 Normalize the coordinates in S to get new coordinates $(\bar{x}, \bar{y})$
7 $r = \text{Roles}(S)$
8 $D^t = D^t \cup \{(\bar{x}^j_{i+u,r(e)}, \bar{y}^j_{i+u,r(e)})^t_{u=0,r(e)}\}$
9 **end**
10 **end**
11 **end**
12 Return D^t for $t = 1, \ldots 5$

Once aligning the trajectories in the order of offensive PG, SG, . . . , defensive PG, SG... and the ball, we normalize a play so that the offensive team always attacks the same basket (step 6 in the algorithm). This ensures that both plays happening on the same side of the court and plays on opposite sides are compared consistently. Finally, we perform k-means clustering based on frame-by-frame L_2 distance separately on ball, offensive player and defensive player trajectories for various time-windows and encode each play based on the clustering results. Let $D^t_h = \{(x, y)_r \in D^t | r \in h\}$ where $(x, y)_r$ denotes the coordinates in (x, y) pertaining to role r and h is a set of roles or ball. Let C^t_b, C^t_o, and C^t_d be the respective clustering results for time-window t based on data D^t_o, D^t_d, D^t_b, respectively, where o is the set of all offense roles, d the set of all defense roles, and b corresponds to the ball. To encode a t-second play, we first compare its ball trajectory to the centroids in C^t_b and replace it with the index value of the closest centroid. We then do the same to player trajectories by comparing the offensive player trajectories against the centroid of every cluster in C^t_o and comparing the defensive player trajectories against those in C^t_d, and obtaining the best match. This results in 11 indices for each play, see Algorithm 2. In the algorithm $f(\cdot)$ represents

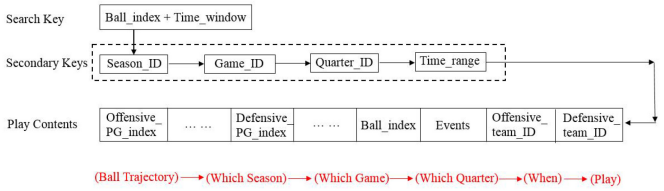

Fig. 2. Structural flow of the playbook.

the mapping from roles and the ball to D_o^t, D_d^t, D_b^t. As we can see, this encoded representation does not provide direct restoration to the original trajectories as it is based on centroids, but in the retrieval process, we do find them from the database exhibited next.

ALGORITHM 2: Encoding of plays

Result: Encoded plays U
$U = \emptyset$
for $t = 1$ **to** 5 **do**
  for *each* $(x, y) \in D^t$ **do**
    for *each role* r **do**
      Find the closest centroid index u_r in $C^t_{f(r)}$ for vector (x, y)
    end
    $U = U \cup \{(u_r)_r\}$
  end
end
Return U

3.2 Database Model

We propose a key-value database model that consists of two tables. In such databases, a table is a list of nested key-value pairs, where each data record is indexed through a primary search key and a number of secondary keys. In the first table, denoted as the *playbook*, we store plays of various time-windows (1–5 seconds) based on the encoded representation. In particular, we group all the plays that have the same ball trajectory index and time-window using the corresponding (compound) primary search key. With this setup, given an input query of t seconds, we can quickly find all the plays that resemble the query at least from the ball trajectory perspective, and thus reduce the number of candidates to a manageable level for further comparison. This also makes intuitive sense as in basketball all players' movements are centered around the ball. If ball trajectories of two plays significantly differ (e.g., a three-point shot and a fast-break layup), the players' trajectories can hardly be similar.

For proper indexing, secondary keys are chosen to be season, game, and quarter IDs, and starting and ending time of the play. These allow us to locate any exact play from historical games. Each play is stored in its encoded form using the 11 indices. To allow richer queries featuring a specific event (e.g., offensive rebound and assist) or between particular teams, a list of major events associated with the play and the offensive and defensive team IDs are also included. The structure of the table is depicted in Figure 2. We observe that this table can be readily constructed from the output of Algorithm 2. When trajectories are constructed in Algorithm 1 we also attach with each one of them the game, season and quarter IDs, and the frame timestamp together with any other relevant information. This enables the construction of the entire playbook.

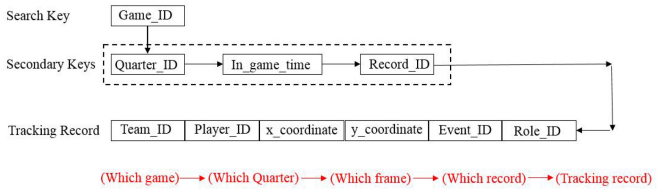

Fig. 3. Structural flow of the trackbook.

We have another table that stores tracking data in its original format. We call it the *trackbook*. As shown in Figure 3, the search key is simply the game ID and the secondary keys are quarter ID, in-game time, and record ID. In-game time allows us to fetch tracking data at the frame level, and record ID is used to differentiate each player and the ball at any given time. Each tracking record consists of the location coordinate along with other information such as the player ID, role and event. This table jointly works with the playbook to restore any encoded play back to the original trajectories, which plays a crucial role during the retrieval.

3.3 Distributed Query-Candidate Processing

As shown in Figure 5, the overall retrieval process can be summarized as follows. Given an input query of t seconds, we first compare its ball trajectory against the centroids of every cluster in C_b^t and find the closest match. This gives us the primary search key in playbook. After constructing all secondary keys (if any are present in the query), we next search the playbook and fetch a list of candidate plays that have ball trajectories similar to the query's. Notice that at this point, the candidate plays are encoded with 11 indices. To determine which candidates are most relevant to the input query, the encoded representation is converted back to trajectories using centroids. For instance, if a candidate play has ball trajectory index n, $n = 1, 2, \ldots$, we use the nth centroid of C_b^t to approximate its ball trajectory. By doing the same to the other player indices, we re-establish (approximate) each candidate play with 11 centroid trajectories. Since our approach aligns trajectories based on player's role, we can compare these re-established candidate plays to the query play by calculating L_2 distance between trajectories that have the same role. The 11 distances can then be summed up as a single similarity measure. We choose L_2 instead of other more sophisticated metrics such as dynamic time wrapping or longest common subsequence because (1) it is easy to use; and (2) according to [30], using other metrics does not lead to any significant improvement in this step.

Each candidate play is re-established and compared to the query in parallel. Once the top candidates and the ranking are decided, we use information contained in the secondary keys to obtain original tracking data of each top result from the trackbook and generate output accordingly. This is how we restore to the original trajectories as mentioned in Section 3.1. As noted in Figure 5, the comparison and ranking are based on approximate plays while only the real plays are shown to users. The question mark represents a user-specific rank function, which is elaborated in Section 3.4.

The described process can be adapted to handle queries with only a partial number of player trajectories specified (note that the ball trajectory must always be specified). The extraction from the playbook is performed in the same way and it results into 11 cluster indices. If a subset of player trajectories is specified in the query, we use only the relevant cluster indices corresponding to the positions that are part of the query trajectories from the returned results (we retain all of the resulting records but from each one we use only a subset of cluster indices). The distance between the query trajectory and these centroids is then calculated only with respect to all of the

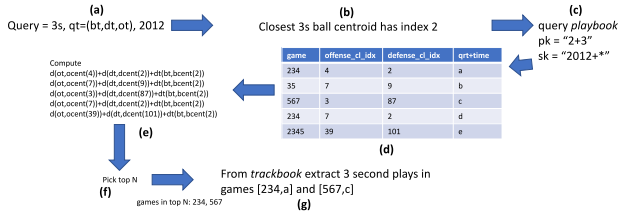

Fig. 4. Example of query processing.

matching trajectories. The rest of the process is identical to the process when all of the trajectories are specified.

We next provide a small example. In Figure 4, we assume that the query is for all plays in season 2012 of duration 3 seconds (3s) – part (a) in the figure. For simplicity we do not show 5 offensive positions and 5 defensive positions, instead we assume only a single trajectory of each, for example, the trajectory part qt of the query consists of ball trajectory bt, offense trajectory ot and defense trajectory dt. The algorithm first finds cluster index 2 corresponding to the closest ball trajectory centroid for the query ball trajectory bt (part (b) in the figure). Next the query into the playbook is formed, which in our case is based on the compound primary key specifying cluster 2 and play duration of 3 seconds. The only component of the secondary key is the season 2012. This is depicted in part (c) in the figure. In the example, the query returns the table in part (d) in the figure. Each returned record corresponds to an original play that is encoded by cluster indices. It includes a game id (some identifier in this case), the cluster index of offense, the cluster index of defense (note that in reality these are 10 cluster indices), and remaining part of the secondary key (in this case the quarter and the time in the quarter). Based on the table, there is game 234 with offense cluster 4, defense cluster 2, and the quarter and the time represented as "a." Likewise, there is game 567 with cluster indices 3 and 87. Note that the ball cluster index 2 is common to all of the returned records. The next step is to compute the distance between the query trajectories and the centroids of the clusters exhibited by the playbook. This is depicted in part (e) in the figure where $ocent$, $dcent$, and $bcent$ return the actual centroid of the cluster given index with respect to C_o^t, C_d^t, and C_b^t, respectively. We denote by "d" the distance function in part (e) where top N plays are then identified in part (f). The corresponding game identification numbers, together with the corresponding parts of the secondary key obtained in part (d), are finally used to extract the actual plays of the top N candidates from the trackbook. Note that the secondary key information associated with each candidate includes the game identification, quarter, and the time range within the quarter of the play.

3.4 Learning to Rank for Sports Play Retrieval

The ranking obtained so far is based on agent-to-agent L_2 distance. However, given a user's search and clickthrough history, an adequate search engine would be able to capture user-specific interest and adjust the search ranking accordingly. To this end, a pairwise learning to rank approach based on rankSVM is taken for learning a user-specific rank function based on the clickthrough data and the similarity metric learnt via a convolutional autoencoder. Let Q be the set of all queries as part of the training dataset. For each query $q \in Q$, we have ordering $(v_1, v_2, v_3, \ldots)$ of the returned results from the retrieval system and sequence C^q containing the ranks of the clicked-on results in the order of clicking. The goal is to re-order $(v_1, v_2, v_3, \ldots)$ so that it reflects the relevance judgments conveyed by C^q.

To obtain an embedding capturing more structural differences between basketball plays, we apply CNN to the 2D visualizations of trajectories. Based on [15], we produce visualization images by

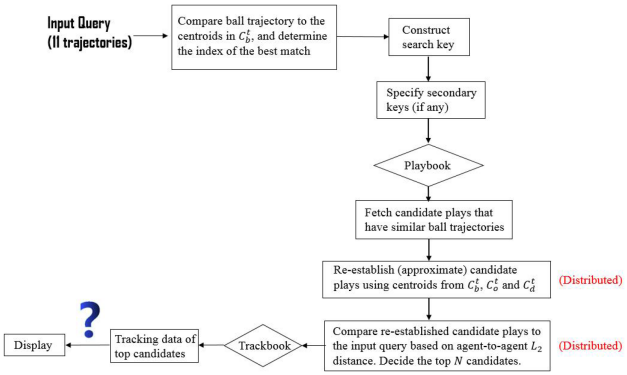

Fig. 5. Summary of the streaming-based retrieval process.

breaking the 94 by 50 feet court down to 1 square foot regions corresponding to individual pixels. In order to identify offense and defense, we use 3 channels to separate offensive players, defensive players, and the ball. For each player and ball trajectory, and at each time frame t, we place a 1 at the pixel location of the corresponding channel that contains (x_t, y_t). The CNN-based autoencoder takes these $3 \times 94 \times 50$ visualization images from each play as input. The tailored network configuration is specified in Section 4.1. The output of the last encoding layer is a compressed feature representation. These features are then used to describe the match between query q and result v as follows:

$$\Phi(q, v) = [f_v, f_v \circ f_q, d_{qv}^1, \ldots, d_{qv}^t],$$

where f_q and f_v are feature vectors from the autoencoder, and d_{qv}^i, $1 \le i \le t$, is a list of role-to-role L_2 distances during the ith second. The time-window t can be inferred from either q or v, and $f_v \circ f_q$ is an element-wise product (Hadamard product). We include this product term to preserve information of q. We use $f_v \circ f_q$ instead of just f_q because rankSVM is trained on pairwise subtractions of Φ's and hence using f_q would result in all zeros and complete loss of information regarding q.

Next, we extract users' preference feedback from the clickthrough data. Since based on [16, 17], the clickthrough data does not convey absolute relevance judgments, but partial relative relevance judgments for the clicked results; in this article, we extract only pairwise preferences. Given C^q for query q, we extract these pairs from a display page based on (1) clicked results are more relevant than unclicked ones, and (2) prior clicked results are more relevant than later clicked ones. Further details are provided in Section 4.1. A linear rankSVM classifier $\vec{w}$ is then trained based on $v_i <_{r^*} v_j \Leftrightarrow \vec{w}\Phi(q, v_i) < \vec{w}\Phi(q, v_j)$, where $<_{r^*}$ denotes the user preferred ranking, using the stochastic gradient descent (SGD) method. User-specific ranking based on a test query can, hence, be achieved by applying the resulting $\vec{w}$ to the pairwise combinations of the top results.

4 IMPLEMENTATION

4.1 Implementation of the Retrieval System

The retrieval system is executed on Amazon Web Service Elastic Compute Cloud. The cluster consists of 8 m3. × large instances, each having 4 cores and 15GBytes of memory. Apache Spark version 1.6.0 is the backbone engine for distributed data processing. The streaming-based retrieval is achieved via Spark Streaming, a Spark generic API for batch processing. We chose Cassandra 2.1.5 as our key-value database because (1) it is open source; (2) it scales up easily for future seasons; and (3) it excels with Spark.

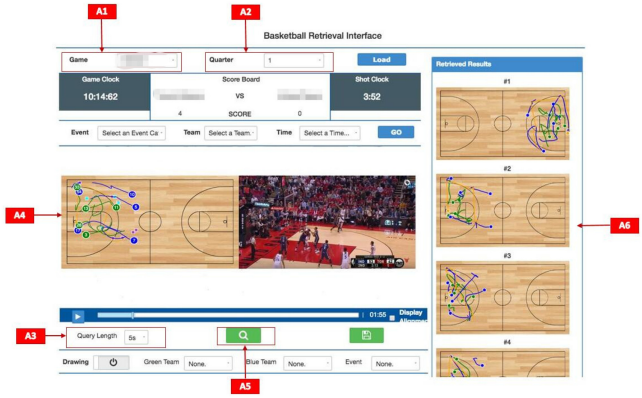

Fig. 6. On the browsing interface, users can (A1) select a game, (A2) select a quarter, (A3) specify the time-window of the play, and (A4) start watching the game (right) and the corresponding 2D visualization (left). Once a play of interest is found, users can (A5) click the retrieval button and (A6) the system returns all the similar plays in the display panel. The ball trajectory is in yellow, and the offensive and defensive player trajectories are in blue and green, respectively.

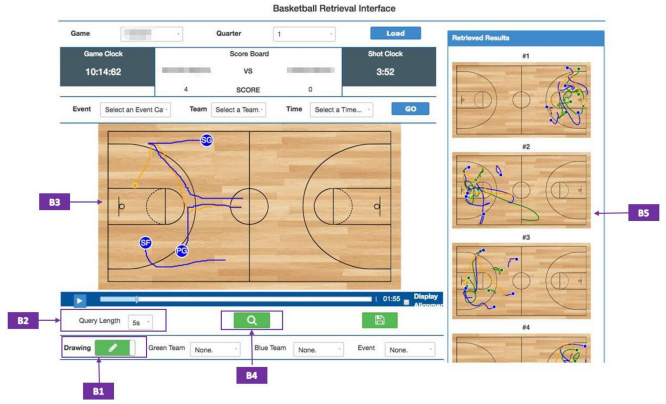

Fig. 7. On the drawing interface, users can (B1) select the drawing tool, (B2) specify the time-window of the play, and (B3) draw a play of interest. Users can then (B4) click the retrieval button and (B5) the system returns all the similar plays in the display panel.

The data was generated by STATS' SportVU tracking system. It utilizes 6 cameras to track all player locations (including referees and the ball) 25 times per second along with detailed log information, such as pass, shot, dribble, and turnover. This dataset is unique since before SportVU, there was very little data available to track player movements and most of the sports analysis was performed based on elementary statistics, such as points and rebounds. We use over 1,100 games of tracking data in this work.

We develop the following UI that allows users to issue an input query either by selecting a play from previous games or by drawing a play of interest similar to how coaches draw plays on boards. Figure 6 shows how users can select a specific play while streaming a game, and Figure 7 gives an example of how to draw a play using the drawing tools. It is similar to [30], but instead of browsing the visualizations of game trajectories, users can watch real games via our interface.

We perform k-means clustering separately on ball, offensive player and defensive player trajectories for various time-windows (1–5 seconds). Due to the volume of the data, the clustering is done in Spark. We choose the number of clusters k so that each cluster in C_b^t contains less than 1,000 trajectories, and each cluster in C_o^t and C_d^t contains less than 5,000 trajectories. To ensure this, we keep dividing clusters that have more than 1,000 (or 5,000) trajectories into subclusters by applying another round of k-means until the criterion is met.

Once the clusters are computed, it is not computationally intensive to construct the playbook and trackbook for one season on a distributed database system. The most intensive task is clustering, however as we show later (see Figures 12 and 13) the centroids are very stable across months and thus they only need to be recomputed periodically. If multiple seasons are to be addressed, the most straightforward strategy is to simply use season in the playbook as part of the secondary key as suggested earlier. The scalability of such a strategy has not yet been tested since we have the data only for a single season. An alternative strategy that definitely scales is to compute centroids based on a single season (or even over a month in a season) and use them for all of the seasons under consideration. With respect to the two tables described in Section 3.2, we should have separate playbooks and trackbooks, one for each season. When a query is issued requiring plays over several seasons, the query can be executed in a distributed fashion over the seasons considered. The results would then be merged and only top N presented. Such a strategy scales linearly with respect to the number of seasons.

The retrieval system comprises of a frontend interface and a Spark streaming-based backend engine. Once an input query is issued via the interface, either by selecting from a previous game or by drawing, a comma separated values file that contains the query information is generated and pushed to a directory in S3.[3] The backend engine Spark streaming monitors the corresponding directory and processes any files created there. With this setup, we avoid going through the Spark-submit initialization steps every time a new query is issued, and thus significantly reduce the response time. This is definitely not the case if each query is submitted to the backend engine as an individual application. Finally, the retrieval output is also saved to S3 based on the final ranking, and the frontend simply plots and displays them sequentially.

Through extensive numerical experiments, we finalize the configuration of our CNN-based autoencoder as 7 convolutional layers and 2 max-pooling layers in the encoding phase, and 7 deconvolutional layers and 2 up-sampling layers in the decoder. We use the typical KL-divergence as the loss function, and ReLU (rectified linear unit) as the activation function. Adaptive subgradient method is utilized to minimize the loss function during training. The whole network is implemented in Keras with Theano backend, and the parameters are calibrated based on validation loss and the quality of the reconstructed output. The inputs to the autoencoder are the $3 \times 94 \times 50$ images described in Section 3.4, and the embedding has dimensionality $8 \times 10 \times 5$.

Lastly, based on the eyetracking experiments by Joachims et al. [17] and the unique design of our interface,[4] we propose the following ways to interpret users' clickthrough behavior, which generalize the single-page logic described in Section 3.4.

(1) For each pair of clicked results (v_i, v_j) on the same page, we have $v_j <_{r^*} v_i$ if v_j is ranked lower but clicked before v_i.
(2) For each pair of results (v_i, v_j) on the same page, we have $v_j <_{r^*} v_i$ if v_j is clicked but v_i is not.

[3]S3 stands for Amazon Simple Storage service.
[4]We display only 5 results on each page. Users are supposed to make eye contact with all 5 results before making any clicks.

(3) For each pair of results (v_i, v_j) with v_i on a higher ranked page and v_j on a lower ranked page, we have $v_j <_{r^*} v_i$ if v_j is clicked but v_i is not.

Consider the following example. A user issues an input query, and then clicks results 5, 1, 3, 4 on the first page, and result 8 on the second page. Since the fifth result is clicked before results 1, 3, and 4, according to (1), we have $v_5 <_{r^*} v_1$, $v_5 <_{r^*} v_3$, and $v_5 <_{r^*} v_4$. Analogously, (2) implies that $v_1 <_{r^*} v_2$, $v_3 <_{r^*} v_2$, $v_4 <_{r^*} v_2$, $v_5 <_{r^*} v_2$ on the first page, and $v_8 <_{r^*} v_6$, $v_8 <_{r^*} v_7$, $v_8 <_{r^*} v_9$, $v_8 <_{r^*} v_{10}$ on the second page. Finally, (3) gives us $v_8 <_{r^*} v_2$. The rankSVM model is trained using simulated data (described next) and solved by the SGD algorithm implemented in Python scikit-learn.

4.2 Simulation of the Clickthrough Data

As mentioned, the rankSVM model is trained on simulated data because:

(1) Our sports play retrieval system is the first of its kind and hence there is no historical data to exploit.
(2) The target user base is relatively narrow and the frequency of usage is also much lower than in typical search engines. Therefore, it is almost impossible to collect extensive user feedback in a short period of time.
(3) The model based on simulated data leads to rankings that well capture user-specific interests expressed in the test dataset (explained later).

We simulated a variety of user preferences including users focusing on close shots (15%), users focusing on mid-range shots (5%), users focusing on three-point shots (25%), users focusing on shot attempts from the corner (5%), users focusing on shot attempts from the wing (5%), users focusing on shot attempts from the top (5%),[5] and normal users focusing on ball trajectory similarity (40%). The percentages are designed to reflect the league's recent trend in three pointers and close shots (dribble layups, dunks, and free throws). All the input queries are randomly generated from previous shot attempts and the clickthrough behavior is simulated based on the following:

(1) We start from the *first page*: for users that have a specific interest, if there is any result (out of 5) matching the preference, click all the qualified results based on their original ranking.
(2) For all users, if the ball trajectory in any result is very close to the query's (i.e., L_2 distance per frame is less than 3.5 feet), click all the qualified results based on their original ranking.
(3) All the remaining results are clicked according to a base probability and how close the ball trajectory is to the one in the query. In particular, the probability for each result to be clicked is base $\times(L_b^*/L_b)$, where L_b denotes the L_2 distance between the query and result's ball trajectories and L_b^* is the smallest L_b among all the returned results. The base probabilities are set as 0.9, 0.9, 0.85, 0.8, and 0.75 for the 5 results on the first page.
(4) We then proceed to the *second page*. It has 0.6 probability to be browsed, and if so, repeat steps 1–3 with base probability 0.5 for all 5 results.
(5) Repeat step 4 for the *third page*, if any, with 0.15 probability to be browsed and base value 0.25.
(6) Repeat step 4 for the *fourth page*, if any, with 0.05 probability to be browsed and base value 0.2.

[5]Corner shots: shot attempts with shot angle $0 - 30^\circ$ on both sides of the court; wing shots: shot attempts with shot angle $30 - 60^\circ$ on both sides of the court; and top shots: shot attempts on top of the three-point line.

Table 1. Satisfaction Rate of the Retrieval Quality

	Participant 1	Participant 2	Participant 3
Yes	35	37	27
No	5%	3%	3%

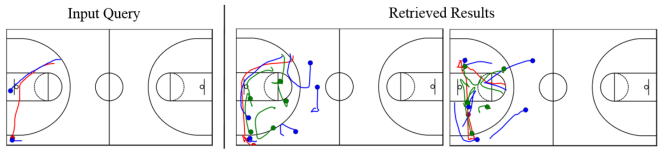

Fig. 8. Selected retrieval results for a penetration and pass (red depicts ball, blue offensive players, and green defensive players – dots represent end of trajectories).

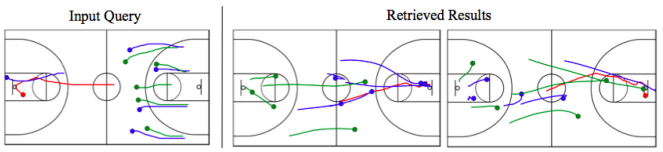

Fig. 9. Selected retrieval results for a fast-break layup (red depicts ball, blue offensive players, and green defensive players – dots represent end of trajectories).

All the probabilities are specified based on the eyetracking experiments in [17]. We assume no more than 20 results are returned for each query and simulate 22,000 search sessions with time-window 5 seconds. The total running time of the simulation is 7 days (one week), which dictates the number of search queries. We split 5% as the validation set, and leave the rest as the training set.

5 NUMERICAL RESULTS

We validate our system via a user study. All participants are basketball domain experts who have previous experience with sports data analytics and naturally understand the concept of our search engine. They were asked to issue queries either by selecting from a previous game or by drawing a play of particular interest, and scan the results top-down and click retrieved plays that they think are most relevant to the input query. The average response time of our system to display the first result is around 5–7 seconds, and the overall processing time to display all 20 (default value) results is less than 15 seconds. All participants prefer to use our system for basketball play retrieval over the traditional approaches (e.g., watching video footage) and are satisfied with the response time. They were also asked if they think the system has done a "good" job in finding the most similar plays for each query, and the number of "yes" and "no" answers are listed in Table 1. As the numbers suggest, around 90% of the time, the participants are satisfied with the retrieval quality.

By examining the queries issued by the participants, we also verify that our system performs equally well on common queries (e.g., dribble and pass) and on relatively rare queries (e.g., fast break). As shown in Figures 8–10, the system not only retrieves well for a normal penetration play (Figure 8), but also returns high quality results for plays like fast break (Figure 9) and offensive rebound (Figure 10).

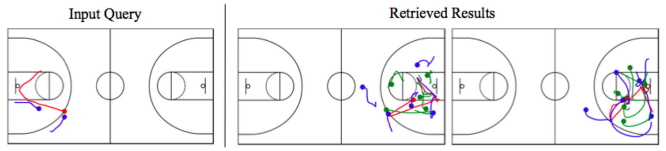

Fig. 10. Selected retrieval results for an offensive rebound and pass out (red depicts ball, blue offensive players, and green defensive players – dots represent end of trajectories).

Table 2. MAP and Kendall's τ for All Strategies

	MAP	Kendall's τ
Rank on each page	**0.817**	0.158
Rank on every 2 pages	0.739	0.382
Rank on all the results	0.506	**0.403**

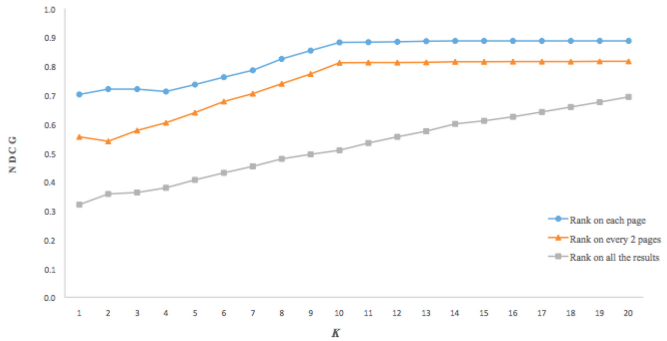

Fig. 11. nDCG at K for all strategies.

Next, we show the effectiveness of learning to rank by evaluating the predicted ranking using participants' clickthrough data. The evaluation is based on metrics that are widely used in information retrieval: Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (nDCG), and Kendall's τ. We omit the details of these metrics since they have been well documented in a variety of literature [14, 16, 19, 28] and are out of the scope of this article. We tune the parameters of the rankSVM model (e.g., number of epochs and L_2 regularization multiplier) by a grid search and assessing MAP on the validation data. Due to the specific design of our display interface (i.e., displaying 5 results on each page), we have the following three ways of applying the rank function: (1) apply to all the top results; (2) sequentially apply to the results on each page and always rank those on previous pages higher; and (3) sequentially apply to the results on each two consecutive pages.[6] Therefore, we repeat tuning three times to get the best model for each measurement setting.

We apply the three models to the 110 test queries from our user study. The predicted rankings are then evaluated based on MAP, Kendall's τ, and nDCG at K. The results are reported in Table 2 and Figure 11, and all the values are averaged across the test queries. As we can see, the MAP values are all on the high-end especially when we apply the model sequentially on each page or

[6]This makes intuitive sense as we only display five results on each page, and thus users are likely to proceed to the second page to examine more results after browsing the first one.

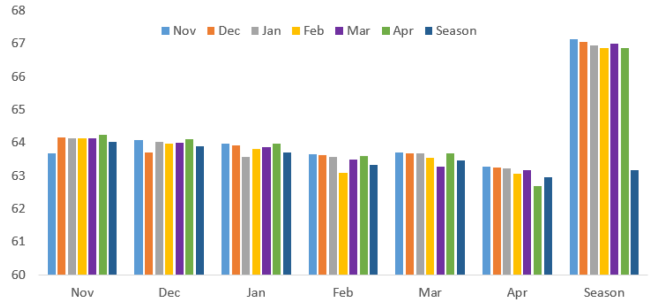

Fig. 12. Average reconstruction errors for 5-second ball trajectories (in feet).

on every 2 pages. We can draw similar insights from nDCG at K. As shown in Figure 11, the gain is higher at each K when we rank the results on each page separately. It increases from 0.702 to 0.887, suggesting really high quality rankings. Similar to MAP's, the nDCG values from the every-2-page approach indicate good rankings as well. The only exception happens to be the Kendall's τ as we achieve the highest score when applying the rank function to all the results. This is because Kendall's τ captures relationships that pagewise strategies cannot. A simple example is as follows. If result 12 on the third page is clicked while result 3 on the first page is not, then based on the criteria illustrated in Section 4.1, we should at least have $v_{12} <_{r^*} v_3$. However, this would be missed by either of the pagewise strategies. Fortunately, the Kendall's τ from the every-2-page approach is very close to its maximum value since users are unlikely to browse many plays from the third page onward. Therefore, based on all three metrics, applying user-specific rank function sequentially on every two pages always leads to good rankings and is thus recommended. Due to the fact that all the depicted numbers are higher than or at least comparable to those in the literature [1, 7], we successfully demonstrate the effectiveness of our learning rank approach with CNN-based embeddings.

We also explore if it is possible to learn the encoded representation with less data. As discussed in Section 3.1, the centroids are learnt from over 1,000 games and more than 3 million trajectories. To see if we really need that much data to learn representative movement patterns, we conduct k-means clustering on monthly data of the 2012–2013 season and examine the reconstruction error from assigning a month's data to other months' clusters. In particular, the reconstruction error is computed as the sum of L_2 distances from any trajectory to its assigned center. We still choose k so that each cluster contains less than 1,000 ball trajectories or 5,000 player trajectories. As we can see from Figure 12,[7] for 5-second ball trajectories, the smallest errors are always achieved when assigning the data to its original cluster; however, the numbers are all close, implying that the cluster centers remain stable from each month to the full season. Larger errors are observed when assigning the whole season's data to clusters of each month, but the differences are not significant (approximately 4%). This is expected since a season has many more plays than an individual month and the centroids in the first 6 cases in the "Season" batch are used based on monthly clustering (which also dictates a lower number of clusters based on how k is determined). It is less likely that monthly centroids are able to capture all the representative patterns of a larger dataset over the entire season and produce the smallest reconstruction error. Note also that clustering all plays in a season with respect to clusters obtained on the same data yields a low error. Finally, the quality of the clustering is also demonstrated as the reconstruction errors suggest around 5 feet distance at each frame.

[7]Playoff months, May and June, are not included as there are much fewer games in these months.

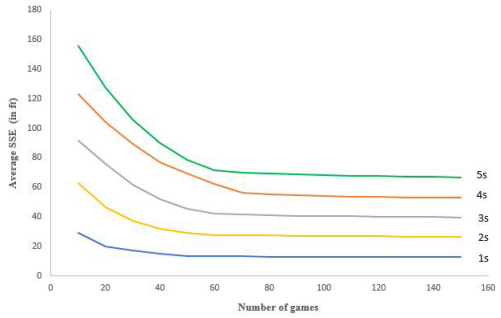

Fig. 13. Average reconstruction errors against the number of games for ball trajectories.

We reach similar conclusions for ball and player trajectories of various time-windows (1–5 seconds) and thus demonstrate that the encoded representation can be learnt from partial data.

To further determine the minimum amount of data required to obtain stable clusters, we perform the same k-means clustering on increasing numbers of games and compute the reconstruction errors of the full data. As shown in Figure 13, for ball trajectories from 1 to 5 seconds, the average values stabilize once the number of games exceeds 70. Similar patterns are observed on player trajectories.

6 CONCLUSION

In this article, we have developed a search engine for basketball plays that can achieve real-time high quality retrieval from a large number of games. With an advanced similarity metric learnt via a CNN-based autoencoder and a pairwise learning to rank approach, our system is able to adapt to user's specific interest and offer user-specific rankings. To examine its effectiveness, we compared the system's output (predicted) ranking to real user feedback. It would be of interest of follow-up work to extend the system's setting to other sports contexts or even other domains of spatiotemporal tracking data, such as animal behavior. The clustering-based encoded representation can be further improved by splitting each cluster with respect to events (e.g., shots, pass, and turnover) or by employing a tree-based hierarchical approach.

REFERENCES

[1] Eugene Agichtein, Eric Brill, and Susan Dumais. 2006. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 19–26.

[2] Eugene Agichtein and Zijian Zheng. 2006. Identifying best bet web search results by mining past user behavior. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 902–908.

[3] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. 2010. Action classification in soccer videos with long short-term memory recurrent neural networks. In *Proceedings of the International Conference on Artificial Neural Networks*. Springer, 154–159.

[4] Alina Bialkowski, Patrick Lucey, Peter Carr, Yisong Yue, and Iain Matthews. 2014. Win at home and draw away: Automatic formation analysis highlighting the differences in home and away team behaviors. In *Proceedings of the 8th Annual MIT Sloan Sports Analytics Conference*. Citeseer.

[5] Alina Bialkowski, Patrick Lucey, Peter Carr, Yisong Yue, Sridha Sridharan, and Iain Matthews. 2014. Large-scale analysis of soccer matches using spatiotemporal tracking data. In *Proceedings of the 2014 IEEE International Conference on Data Mining*. IEEE, 725–730.

[6] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*. ACM, 89–96.

[7] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. 2006. Adapting ranking SVM to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 186–193.

[8] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*. ACM, 129–136.

[9] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. ACM, 491–502.

[10] Sheng Chen, Zhongyuan Feng, Qingkai Lu, Behrooz Mahasseni, Trevor Fiez, Alan Fern, and Sinisa Todorovic. 2014. Play type recognition in real-world football video. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*. IEEE, 652–659.

[11] David Cossock and Tong Zhang. 2006. Subset ranking using regression. In *Proceedings of the International Conference on Computational Learning Theory*. Springer, 605–619.

[12] Ke Deng, Kexin Xie, Kevin Zheng, and Xiaofang Zhou. 2011. Trajectory indexing and retrieval. In *Computing with Spatial Trajectories*. Springer, 35–60.

[13] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research* 4 (2003), 933–969.

[14] Li Hang. 2011. A short introduction to learning to rank. *IEICE Transactions on Information and Systems* 94, 10 (2011), 1854–1862.

[15] Mark Harmon, Patrick Lucey, and Diego Klabjan. 2016. Predicting shot making in basketball using convolutional neural networks learnt from adversarial multiagent trajectories. *arXiv:1609.04849* (2016).

[16] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 133–142.

[17] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2005. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 154–161.

[18] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. 2007. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems* 25, 2 (2007), Article 7.

[19] Maurice George Kendall. 1948. *Rank Correlation Methods*. Griffin.

[20] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. 2009. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data* 3, 1 (2009), Article 1.

[21] Hang Li. 2014. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies* 7, 3 (2014), 1–121.

[22] Ping Li, Qiang Wu, and Christopher J. Burges. 2007. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems*. 897–904.

[23] Siyuan Liu, Qiang Qu, and Shuhui Wang. 2015. Rationality analytics from trajectories. *ACM Transactions on Knowledge Discovery from Data* 10, 1 (2015), Article 10.

[24] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (2009), 225–331.

[25] Bernard Loeffelholz, Earl Bednar, and Kenneth W. Bauer. 2009. Predicting NBA games using neural networks. *Journal of Quantitative Analysis in Sports* 5, 1 (2009), 1–15.

[26] Patrick Lucey, Alina Bialkowski, Peter Carr, Yisong Yue, and Iain Matthews. 2014. How to get an open shot: Analyzing team movement in basketball using tracking data. In *Proceedings of the 8th Annual MIT Sloan Sports Analytics Conference Conference*. 1–10.

[27] Alan McCabe and Jarrod Trevathan. 2008. Artificial intelligence in sports prediction. In *Proceedings of the 5th International Conference on Information Technology: New Generations, 2008*. IEEE, 1194–1197.

[28] Brian McFee and Gert R. Lanckriet. 2010. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning*. 775–782.

[29] Armand McQueen, Jenna Wiens, and John Guttag. 2014. Automatically recognizing on-ball screens. In *2014 MIT Sloan Sports Analytics Conference*.

[30] Long Sha, Patrick Lucey, Yisong Yue, Peter Carr, Charlie Rohlf, and Iain Matthews. 2016. Chalkboarding: A new spatiotemporal query paradigm for sports play retrieval. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*. ACM, 336–347.

[31] Amnon Shashua and Anat Levin. 2002. Ranking with large margin principle: Two approaches. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)*. 937–944.

[32] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. Softrank: Optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM, 77–86.
[33] Kevin Toohey and Matt Duckham. 2015. Trajectory similarity measures. *SIGSPATIAL Special* 7, 1 (2015), 43–50.
[34] Kuan-Chieh Wang and Richard Zemel. 2016. Classifying NBA offensive plays using neural networks. In *Proceedings of MIT Sloan Sports Analytics Conference*.
[35] Xinyu Wei, Long Sha, Patrick Lucey, Stuart Morgan, and Sridha Sridharan. 2013. Large-scale analysis of formations in soccer. In *Proceedings of the 2013 International Conference on Digital Image Computing: Techniques and Applications*. IEEE, 1–8.
[36] Kasun Wickramaratna, Min Chen, Shu-Ching Chen, and Mei-Ling Shyu. 2005. Neural network based framework for goal event detection in soccer videos. In *Proceedings of the 7th IEEE International Symposium on Multimedia*. IEEE, 8 pages.
[37] Ou Wu, Qiang You, Fen Xia, Lei Ma, and Weiming Hu. 2016. Listwise learning to rank from crowds. *ACM Transactions on Knowledge Discovery from Data* 11, 1 (2016), Article 4.
[38] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*. ACM, 1192–1199.
[39] Zhang Zhang, Kaiqi Huang, and Tieniu Tan. 2006. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In *18th International Conference on Pattern Recognition*. Vol. 3, IEEE, 1135–1138.
[40] Yu Zheng. 2015. Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology* 6, 3 (2015), Article 29.

Received October 2017; revised May 2018; accepted May 2018