Algorithm 1. PairModel(S,E)
- Input: pair (S,E)
1. $\Phi \leftarrow$ feature vector for $(S, e_j)$ $\forall e_j \in E$
2. $\text{Pos} \leftarrow \bigcup_j (S, e_j)$ $\forall e_j \in E$
   // Generate Negative Examples
3. $\forall$ pairs: sort $D_c \leftarrow \text{Dist}(e, e_c)$, sort $D_S \leftarrow \text{Dist}(S, S_k)$
4. $\text{Neg}_1 \leftarrow (S_k, e_1) : S_k \in D_{\text{S}}}^{\text{neg}} : e_1 \in D_{\text{e}}}^{\text{neg}}$
5. $\text{Neg}_2 \leftarrow (S_k, e_1) : S_k \in D_{\text{S}}}^{\text{neg}} : e_1 \in D_{\text{e}}}^{\text{neg}}$
6. $\text{Neg} \leftarrow \text{Neg}_1 \cup \text{Neg}_2$
7. return $\text{SVM}(\text{Pos}, \text{Neg})$ with the weight vector $\Theta$

Algorithm 2. ComputeConf(pij, pik)
- Input: pairs $p_{ij} = (S_i, E_j)$ and $p_{ik} = (S_k, E_l)$ with feature vector $\Phi_{kl}$
1. $M_{ij} \leftarrow \text{PairModel}(p_{ij})$ (Alg. 1)
2. $\Theta_{ij} \leftarrow$ weight vector of $M_{ij}$
3. $\text{Conf}(M_{ij}, p_{ik}) \leftarrow \max_{e_j \in E_i} \Theta_{ij, e_j}$, $\Phi_{kl}$

Algorithm 3. PairRank(pairs)
- Output: ranks of all the pairs
   //Build adjacency matrix
1. for pairs $p_{ij}, p_{ik}$:
   (a) $\text{ComputeConf}(p_{ij}, p_{ik})$ (Alg. 2)
   (b) $P_i \leftarrow$ scores of applying $M_{ij}$ on all pairs
   (c) $P_k \leftarrow$ scores of applying $M_{kl}$ on all pairs
   (d) $\text{edge}(p_{ij}, p_{ik}) \leftarrow$ $(\text{rank}(p_{ij})$ in $P_i)$ and $(\text{rank}(p_{ik})$ in $P_k)$
   //Iteration $t$
2. while $|\tilde{p}_1 - \tilde{p}_{t-1}| \geq \epsilon$
   (a) for every pair $p_{ij} = (S_i, E_j)$
      i. $\rho_t(p_{ij}) \leftarrow$ Equation 2
      ii. $\rho_t(p_{ij}) \leftarrow \rho_t(p_{ij}) \frac{\text{frequency}(E_j)}{\text{sum}(\rho_t)}$
      iii. $\rho_t(p_{ij}) \leftarrow \rho_t(p_{ij})$
3. return $\tilde{\rho}$

Algorithm 4. ReturnMacroEvent(pairs, k)
- Output: best macro event for each bucket
1. Iteration $l = 0$
   (a) $E \leftarrow \emptyset$
   (b) $\tilde{\rho} \leftarrow \text{PairRank}(\text{pairs})$
   (c) $A^* \leftarrow$ highest ranked pair in each bucket according to $\tilde{\rho}$
2. Iteration $l$
   (a) $E \leftarrow E \oplus A^*$
   (b) $\tilde{E} \leftarrow \text{PairRank}(\text{pairs} \cup A^*)$
   (c) for $e_i$ in every bucket:
      i. $\tilde{E}_i \leftarrow \text{PairRank}(\text{pairs} \cup E \oplus e_i)$
      ii. $\Delta_{e_i} \leftarrow \rho_t(S_i, E \oplus e_i) - \rho_1(S, E)$
      (d) $A^* \leftarrow \text{arg max}_{e_i} \Delta$
3. repeat until $l \leq k$
4. return $E$ for each bucket

4 Experiments

We evaluate our method on how accurately it aligns sentences in professional soccer commentaries with events in the actual games. We use our professional soccer dataset as the main testbed and compare our method with state-of-the-art methods and several different baselines. For comparison, we also test our model on a benchmark dataset of RoboCup soccer [Chen and Mooney, 2008].

4.1 Professional Soccer Commentaries

Our dataset consists of time-stamped commentaries and event logs of 8 games in 2010-2011 season of English Premier League. For evaluation purposes, we label ground-truth annotations for the correspondences between sentences and events throughout the dataset. There are 935 sentences, 14,845 events, 2,147 words, and 306 players in total. Each sentence on average has 16.62 words. For each sentence in the commentaries, we assign a bucket by selecting events that occur in an interval of 150 seconds around the time the sentence has been generated. We then pair each sentence with all of the events in the corresponding bucket. This results in 38,332 pairs. On average, there are 42 pairs in each bucket. Of course, not all of these pairs are correct correspondences. There are in total 1,404 pairs labeled as correct matches in the ground-truth labels. On average there are 2 correct pairs in each bucket.

Each event is represented with an event type followed by a list of its arguments. There are 55 event types and several arguments defined in the event logs. Examples of the arguments are the time that the event occurred, the player name that is the agent of the event, the team name, the outcome of the event, and the body part. Most of the arguments are categorical. The only string field is the player name that can provide useful information for finding initial guesses for correct correspondences. However, identical player names occur in multiple events in each bucket.

We apply our method (Algorithm 4) on this dataset. We start by pairing sentences with events in the corresponding buckets. This step is followed by learning PairModels for every pair $p_{ij} = (S_i, E_j)$ that has at least one matching player name between the sentence $S_i$ and the event $E_j$. The features for the pair $p_{ij}$ are $\Phi_{ij} = (\Phi_{S_i}, \Phi_{E_j}, \Phi_{St})$ where $\Phi_{S_i}$ and $\Phi_{E_j}$ are binary vectors representing the sentence and the event, and $\Phi_{St}$ is an integer value corresponding to the number of players matched between the sentence and 3 consecutive events. At the first step, each PairModel $M_{ij}$ is trained using one positive example $(S_i, E_j)$ and 100 negative examples generated automatically as follows. We sort all sentences according to their Euclidean distance to the sentence $S_i$ and all events based on their Euclidean distance to the event $E_j$. We generate negative examples from the sentences and events that are not identical to $S_i$ and $E_j$. Half of the negative pairs are generated from the sentences with lowest distance to $S_i$ and events with highest distance to $E_j$. The other half are generated by pairing sentences with highest distance to $S_i$ and events with lowest distance to $E_j$. We make sure that no pairs with matching player names between sentences and events exist in the negative set. This way we try to make sure that negative examples do not contain the same patterns of correspondences as the positive examples.