3.1 Learning Pair Models

We adopt a discriminative approach that learns the characteristic correspondence patterns that distinguishes a pair from all other pairs. Recently in computer vision, Exemplar Support Vector Machines (ESVM) has shown great success in learning what is unique about an image that can distinguish it from all other images [Malisiewicz et al., 2011; Shrivastava et al., 2011]. The main idea is very simple, yet surprisingly effective. To learn what is unique about each example, one can fit an SVM with only one positive instance and large number of negative instances. The main intuition is that an example can be defined as what it is not, rather than what it is like. Despite being susceptible to overfitting, the proposed hard negative mining method gets away from this issue.

This framework suits our problem setting very well because we learn for all pairs of sentences and events in each bucket and do not need training labels for which pairs are correct. We extend this approach to learn models of pairs (called PairModel). A PairModel demonstrates how to weigh features of a pair against each other in a discriminative manner. If a learned model for a pair produces a positive score when applied to another pair, then two pairs share analogous patterns of correspondences.

**Feature Vector**: The features for each pair $p_{ij} = (S_i, e^j_i)$ of a sentence $S_i$ and event $e^j_i$ are $\vec{\phi}_{ij} = (\vec{\Phi}_{S_i}, \vec{e}^j_i, \vec{\Phi}_{st_i})$. $\vec{\Phi}_{S_i}$ is a binary vector representing the sentence $S_i$, where each element in the vector shows the presence of a word in the vocabulary. The vocabulary consists of frequent words in the domain except the words that can occur in the string arguments. For instance, the vocabulary does not include the player names. $\vec{\phi}_{ij}$ is a binary vector representation of the event $e^j_i$ that includes the event type together with its arguments. Each element in the vector represents the presence of the corresponding event type or the argument value in $e^j_i$. $\vec{\Phi}_{st_i}$ is a binary vector with one element for every string type in the event arguments. Every element in the vector $st_i$ denotes if the string argument is matched with a word in the sentence. For instance, the feature vector has an argument to demonstrate whether or not the player name (argument of the event $e^j_i$) has occurred in the sentence $S_i$. The output of the PairModel $M_{ij}$ learned for the pair $p_{ij} = (S_i, e^j_i)$ is a weight vector $\vec{\Theta}_{ij}$. The confidence of applying $M_{ij}$ over a new pair $p_{kl} = (S_k, e^j_k)$ is computed as $\text{Conf}(M_{ij}, p_{kl}) = \vec{\Theta}_{ij} \cdot \vec{\Phi}_{kl}$. This confidence compares the patterns of correspondence between $S_i$ and $e^j_i$ and patterns of correspondence between $S_k$ and $e^j_k$. The PairModel tries to weight the features that are most important to discriminate the corresponding pair from the rest. High confidence means that those important features are “on” in an example; therefore, it is following the same pattern. Figure 2 demonstrates three PairModels, the top-weight words for each pair, and the nearest sentence under the learned patterns of correspondences. Non-discriminative measures of similarity like Cosine or Euclidean are not desirable because they treat all the dimensions in the same way. For comparisons and experimental evaluations please see Section 4.

A PairModel $M_{ij}$ likes the pair $p_{kl}$ by $\text{Conf}(M_{ij}, p_{kl})$. We score the similarity between two pairs by looking at their mutual likeness meaning that they both share analogous patterns of correspondence. We can now start reasoning about the popularity of PairModels by aggregating all mutual likeness scores. To propagate the mutual likeness information we adopt a strategy similar to Google PageRank.

3.2 Ranking Pairs

The input to this module are all of the pairs and their learned models, and the output is the popularity scores of the pairs relative to each other. A pair $p_{ij} = (S_i, e^j_i)$ is likely to be a correct match if the pattern of correspondence extracted by $M_{ij}$ occurs frequently relative to other pairs; this means that $p_{ij}$ is popular. A pair is frequent if many popular pairs like that pair with high confidence. This resembles similar problems in ranking Web pages.

We adopt an approach similar to the PageRank algorithm [Brin and Page, 1998] utilized by Google to order the importance of webpages. PageRank examines the graph of webpages (called Webgraph) and assigns a high score to a web page if many important pages link to the page.

Here, we are interested in computing relative scores of the pairs. We build a graph of pairs by assigning a node to each pair. Unlike the Webgraph, the edges are undirected and weighted. We assign a weighted edge between a pair $p_{ij}$ and a pair $p_{kl}$, if they mutually like each other (i.e., $\text{Conf}(M_{ij}, p_{kl}) > 0$ and $\text{Conf}(M_{kl}, p_{ij}) > 0$). We call this edge the popularity link. The graph also has self loops; i.e., a node can also be connected to itself if $\text{Conf}(M_{ij}, p_{ij}) > 0$. This self loop encodes how com-