Learning to Rank from Distant Supervision: Exploiting Noisy Redundancy for Relational Entity Search*

Mianwei Zhou†, Hongning Wang†, Kevin Chen-Chuan Chang†,*

† Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
* Advanced Digital Sciences Center, Illinois at Singapore, Singapore

{zhou18, wang296, kcchang}@illinois.edu

Abstract—In this paper, we study the task of relational entity search which aims at automatically learning an entity ranking function for a desired relation. To rank entities, we exploit the redundancy abound in their snippets; however, such redundancy is noisy as not all the snippets represent information relevant to the desired relation. To explore useful information from such noisy redundancy, we abstract the task as a distantly supervised ranking problem – based on coarse entity-level annotations, deriving a relation-specific ranking function for the purpose of online searching. As the key challenge, without detailed snippet-level annotations, we have to learn an entity ranking function that can effectively filter noise; furthermore, the ranking function should also be online executable. We develop Pattern-based Filter Network (PFNet), a novel probabilistic graphical model, as our solution. To balance the accuracy and efficiency requirements, PFNet selects a limited size of indicative patterns to filter noisy snippets, and inverted indexes are utilized to retrieve required features. Experiments on the large scale CuleWeb09 data set for six different relations confirm the effectiveness of the proposed PFNet model, which outperforms five state-of-the-art relational entity ranking methods.

I. INTRODUCTION

Traditional entity search methods [7], [22], [3], which return entities that match the relation described by user-given keywords, would fail if the relation is represented by different keywords in Web pages. For example, given a query “microsoft founded by #person,” which looks for a person-typed entity who is the founder of “microsoft,” the entity search engine should return “bill gates” and “paul allen” as the answers. However, if the relation is represented by different keywords in the Web documents (e.g., “started” instead of “founded by”), such a search scheme would not work well.

To address the issue, we propose to study the problem of relational entity search in this paper. In relational entity search, with respect to a desired relation (e.g., FounderOf()), given a query (e.g., “microsoft”), the system will rank entities by an automatically learned relation-specific ranking function such that a higher-ranked entity (e.g., “bill gates” and “paul allen” as #person) is expected to better match the desired relation for the query. Such a search scheme has two advantages: first, by relying on a relation-specific ranking function, it relieves users from the burden of specifying possible keywords for the target relation; second, we can expect that ranking functions carefully designed for different relations (e.g., FounderOf() and PublisherOf()) would achieve better performance than a general one.

In order to rank entities for a given query with respect to a desired relation, the ranking function will draw upon the aggregation of their snippets – the textual fragments recording how the query and the entity co-occur. Figure 1 shows an example of relational entity search, which aims at finding the founders of “microsoft.” Formally, given “microsoft” as a query q, we are looking for a person-typed entity e, i.e., e ∈ #person, such that the relation FounderOf(“microsoft”, e) is true (i.e., “bill gates” and “paul allen” underlined in Figure 1). To fulfill such a goal, the relational entity search engine retrieves a list of candidate person entities that ever co-occur with the query “microsoft,” and each entity is represented as a bag of snippets describing how entity e and query q are related. Based on such abundance of snippets, the relational entity search engine will draw upon a ranking function to rank the entities.

Hence, to realize effective relational entity search, we must essentially exploit the redundancy of the Web – an insight that many Web applications (e.g., Web-based QA [8], [4], Web IE [9]) have exploited. In large corpora like the Web, true facts are often repeated many times, and we can leverage such redundancy of Web data to achieve a more reliable entity ranking. Taking Figure 1 as an example, different snippets (s_{11}, s_{12}) mention the fact that “bill gates” founded “microsoft,”

<table>
<thead>
<tr>
<th>Entity</th>
<th>Snippets with “microsoft”</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1: “bill gates”</td>
<td>s_{11}: Microsoft was founded by Bill Gates and …</td>
</tr>
<tr>
<td>s_{12}: Bill Gates met Microsoft CEO at his home …</td>
<td></td>
</tr>
<tr>
<td>s_{13}: Bill Gates dropped out of college and started Microsoft.</td>
<td></td>
</tr>
<tr>
<td>e2: “steve ballmer”</td>
<td>s_{21}: Steven A. Ballmer is CEO of Microsoft.</td>
</tr>
<tr>
<td>s_{22}: Steven Ballmer has headed several Microsoft divisions.</td>
<td></td>
</tr>
<tr>
<td>e3: “paul allen”</td>
<td>s_{31}: Paul Allen is best known to be the co-founder of Microsoft.</td>
</tr>
<tr>
<td>s_{32}: Microsoft is founded by … Paul Allen in 1970’s …</td>
<td></td>
</tr>
<tr>
<td>e4: “jerry yang”</td>
<td>s_{41}: Microsoft CEO met Yahoo co-founder Jerry Yang …</td>
</tr>
<tr>
<td>s_{42}: If Jerry Yang is upset that Microsoft bid to buy the company …</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1. Example entities and instances (snippets).
based on which we can confidently conclude that “bill gates” is a correct answer for FounderOf(“microsoft”, e).

However, while redundancy of the Web is an opportunity to exploit, it comes with inherent noise which hampers effective ranking in our setting. Such noisy redundancy exists as not all the snippets represent information relevant to the target relation (e.g., $s_{12}$ to the relation FounderOf($q$)). Failing to filter out such noisy snippets will tempt the ranking model to recognize a negative sense of evidence for supporting the desired relations, e.g., treating $s_{12}$ as an evidence for FounderOf(). Existing redundancy-based Web applications [7, 22, 3] rely on manually-crafted or user-given keywords (e.g., “founded by”) to match relevant snippets, whereas, in relational entity search, without such keywords as given, the problem boils down to learning a relation-specific ranking function that can effectively filter noisy snippets.

Learning such a noise-robust relational entity ranking function is non-trivial. In order to filter out noisy snippets, one straightforward solution is to first annotate each $(q, e, s)$ triplet (e.g., by annotating $s_{11}$ and $s_{13}$ as positive, $s_{12}$ as negative), and train a classifier for each relation. Unfortunately, such a solution is impractical as even a single entity is associated with a large number of snippets; although one can sample a small set of snippets for annotation, it is difficult to guarantee that the sampled snippets would cover different representations of the relation to achieve good generalization capability.

In this paper, rather than requiring detailed snippet-level annotations, we propose to utilize entity-level annotations as distant supervision [17] for learning the ranking model. Such entity-level annotations are usually easy to obtain from various public knowledge bases such as Freebase and Wikipedia. For example, in Figure 1, according to Freebase, we can label “bill gates” and “paul allen” as positive, without indicating which snippets actually match the desired relation ($s_{11}$ and $s_{13}$ do, but not $s_{12}$), and we are mandated to learn the ranking function for FounderOf() based on such coarse-grained annotations.

Therefore, we abstract our task as the distantly supervised ranking problem which aims at learning a relation-specific ranking function for entity search based on distant supervision. In particular, in the offline training phase, with respect to a target relation, given a set of training queries with labeled entities and unlabeled snippets, one needs to derive a ranking function, such that, in the online searching phase, given a new query, the ranking function could correctly identify its positive entities that satisfy the target relation based on their associated snippets. Towards addressing the distantly supervised ranking problem, we are facing two challenges:

First, in terms of accuracy, due to the lack of detailed snippet labels, we have to filter noise before leveraging the redundancy for ranking. Taking Figure 1 as example, the ranking function should recognize $s_{12}$ as a noisy snippet, and only rely on the snippets of $s_{11}$ and $s_{13}$ to rank entity $e_1$.

Second, to facilitate an online search engine, the ranking function must be efficient. In particular, we should resort to existing inverted indexes to fetch features for snippet filtering, and therefore we can only choose features that are “indexable.” Moreover, since such index traversing is also time consuming, the number of “indexable” features should also be limited by a small budget.

In order to exploit noisy redundancy accurately and efficiently, we believe the key lies in leveraging redundancy itself. As our insight, a positive entity will have more relevant snippets as support, which we identify as the redundancy ranking principle. Moreover, there always exist some common keywords/phrases that are indicative of the target relation in a relevant snippet (e.g., “founded by” for FounderOf()). We name such keywords/phrases as indicative patterns and leverage such pattern redundancy to address the two challenges. First, in terms of accuracy, we can identify such indicative patterns by their redundancy for noise filtering. Taking Figure 1 as an example, since the pattern “founded by” co-occurs most of time with positive entities “bill gates” and “paul allen,” we can confidently conclude that “founded by,” rather than “met ... CEO,” is a good indicative pattern for FounderOf(), and thus $s_{11}$ and $s_{13}$ should be chosen as evidences and $s_{12}$ should be filtered as noise. Second, in terms of efficiency, due to pattern redundancy, a small number of indicative patterns can already cover a large range of relevant snippets and retrieve almost all positive entities (by covering at least one of their relevant snippets) for ranking, and thus we can limit the number of patterns to be small to fulfill the efficiency requirement.

Based on such an insight, we develop the Pattern-based Filter Network, named PFNet, in the framework of Markov network, as our solution. Supported by the power of probabilistic models, we encode our redundancy ranking principle in a principled way: to tackle with noise and improve efficiency, PFNet selects a restricted set of indicative patterns to fetch a small size of snippets as evidences for the candidate entities by inverted indexes, and filter other snippets as noise; the candidate entities are ranked based on the aggregation of such chosen evidences. A step-wise greedy algorithm is proposed to estimate PFNet efficiently.

To validate the effectiveness of the proposed PFNet model, we performed large-scale evaluation over 100 million web pages from the ClueWeb09 dataset (on a cluster of 30 nodes), for a variety of six relational entity search tasks (e.g., FounderOf() and PlaceOfBirth()). In order to demonstrate the necessity of automatically learning relational ranking functions and the importance of leveraging redundancy and filtering noise, we compared our algorithm with different baseline methods including EntityRank which is an unsupervised ranking model proposed in [7], multi-instance learning [19] which models noise but does not leverage redundancy, and SVMRank [13] which directly leverages redundancy without noise filtering. We observed consistent improvement (NDCG@5 +15% in average over all relations) compared with those baselines.

II. RELATED WORK

There are several related efforts of relational entity search in the literature. Cheng et al. propose EntityRank, a domain-independent probabilistic entity ranking model [7], where local recognition and global access information are integrated in an
unsupervised manner. In Zhou et al.’s DoCQS system [22], the searching capability has been distilled by asking users to manually craft the ranking rules for their inquired relations. Banerjee et al. [3] study to answer quantity consensus queries, where each answer is a tight quantity interval extracted from evidences in thousands of snippets. In addition, our work is also closely related to Web-based QA. Traditional Web-based QA work [15], [4], [1] heavily relies on standard search engines to retrieve relevant documents based on the input question and then extract answers according to some predefined templates, which would fail if the answer is represented by different keywords in the Web page. In contrast, our work leverages the abundance of relational data as training data to automatically learn more robust and accurate retrieval models for specific relation types. Thus, our work could be viewed as an important vehicle to support traditional Web-based QA.

The idea of leveraging Web redundancy is first proposed by Clarke et al. [8], and it is further validated and widely adopted in various Web applications such as Web-based QA [4], [16], Web IE [9] and entity search [7]. Most of such works rely on manually-crafted patterns or user-given queries to retrieve relevant snippets, and they model redundancy by different approaches, e.g., [16], [9] measure the redundancy of the query-answer co-occurrences by Pointwise Mutual Information (PMI); while in [7], Cheng et al. use a global access layer to aggregate the redundancy of snippet scores. In contrast, without patterns that are indicative of the target relation as given, starting with snippets where the query and the entity simply co-occur, we filter noisy snippets by automatically learned patterns, and rank entities based on the filtered snippets according to the redundancy ranking principle.

Besides, some recent work has explored the idea of distant supervision. Mintz et al. utilize Freebase data as distant supervision to learn relational classifiers [17]. In their work, various types of features ranging from lexical (N-gram and POS tagging patterns) and syntactic (dependency parsing) features are introduced to enhance the classification performance. However, they do not consider the potential noise in the training data, and use all the snippets to train their classifier. Riedel et al. realize the existence of noise and propose to utilize the distant supervision in a more sophisticated manner [19]: they assume that positive entities should have at least one evidence snippet and solve the problem by Multi-Instance Learning [2]. Hoffmann et al. extend it by presenting a conditional extraction model to learn relations from heuristically labeled training data and combat with noise [11]. Although previous studies have noticed the adverse impact introduced by the noisy unlabeled data, their solutions are not directly applicable to our relational entity search problem. First, in terms of efficiency, they heavily depend on the expensive IE features, e.g., POS tagging and dependency parsing, to distinguish noisy snippets, which are infeasible for an online system; second, in terms of accuracy, the “redundancy” is not explicitly modeled in their methods, that would make the model vulnerable to erroneous positive snippets in negative entities. In our work, we restrict our model to the “indexable” features for online processing purpose and exploit the redundancy within the associated snippets to solve the relational entity search problem.

III. EXPLORING NOISY REDUNDANCY FOR RELATIONAL ENTITY SEARCH

In this section, we describe our general framework for relational entity search. As the major component of the framework, the distantly supervised ranking problem is formally defined in the end of the section.

A. Relational Entity Search Framework

In relational entity search, given a target relation $r(q, \#E)$, in which we are interested in the entity $e$ of type $\#E$ that forms relation $r$ with $q$. For example, for query $FounderOf(\"microsoft\", \#person)$, we need to find $e \in \#person$ (e.g., “bill gates” and “paul allen”) such that $FounderOf(\"microsoft\", e)$ is true. The desired output is a list of entities $e \in \#E$, ranked by the confidence that $r$ holds between $q$ and $e$.

To retrieve the candidate entities $e$, we appeal to an entity-aware searcher (e.g., [5], [7]), which, essentially, is a document
retrieval system with an additional index of entities to support entity finding. Figure 2 shows an example index for entity type \#person, where each posting records a specific entity (e.g., “bill gates”) and its occurrence in the corpus (e.g., 5th word in document d2). With entity position recorded in the inverted index, the searcher can support a set of proximity checking operations: for example, for the input of Before[“founded”, \#person], the searcher will return all the \#person entities which appear before the keyword “founded” in some snippets.

Based on such an entity-aware searcher, we design our relational entity search framework in Figure 4: a set of relation-specific ranking models is learned in the offline training phase in advance; and in the online searching phase, such learned models would be chosen to answer the input query according to the target relation.

In the offline training phase, given a set of known relations, we utilize the entity-aware searcher to obtain candidate entities and construct a training corpus. Taking the relation FounderOf(“microsoft”, {“bill gates”, “paul allen”}) as an example, we will use the entity-aware searcher to find a list of \#person entities (e.g., “bill gates”) that ever co-occur with the term “microsoft,” and associate each candidate entity with a set of snippets. Then, “bill gates” and “paul allen” will be labeled as positive, and other entities as negative, leaving all the snippets unlabeled.

With only such coarse entity-level annotations, we are mandated to learn a relation-specific ranking function \( R(e, s) \), which takes an entity \( e \) and its supporting snippets \( s \) as input, and produces a ranking score to rank the candidate entities according to a set of features defined over \( e \) and \( s \) with respect to the target relation \( r \). We name such a learning problem as the **distantly supervised ranking** problem. As the name indicates, first, we need to explore the indirect supervision defined on entities; second, as a practical ranking problem, the ranking function \( R \) must be executable online. Before we give the formal definition and solution of the **distantly supervised ranking** problem, we would first discuss the properties a practical ranking function \( R \) should have for efficient online execution purpose.

The **online searching phase** will execute the learned ranking model against new queries. As shown in Figure 4, given a query, e.g., FounderOf(“facebook”, \( e \)), the entity-aware searcher will look for the related \#person entities and snippets, then a relation-specific entity ranker will predict the ranking among the retrieved entities.

As an online system, efficiency is the main concern: the major bottleneck arises in fetching the ranking features. Assuming that we are interested in a snippet feature indicating whether the keyword “founded” appears before the entity, denoted as \( \text{BeforeEntity} \[ “founded” \] \), there are two different approaches to obtain such a feature, as shown in Figure 3. The first approach, based on document checking, needs to fetch all snippets first, and then extract features from the detailed snippet contents. Although straightforward, it is quite inefficient as it requires a large number of I/O operations, especially when the number of snippets is large. In this work, we would adopt the second approach which takes advantage of inverted indexes: with all positions of keywords and entities recorded in an inverted index, \( \text{BeforeEntity} \[ “founded” \] \) can be calculated for all snippets by joining the “founded” index and the \#person index.

To adapt to the inverted index-based approach, the ranking model can only use “indexable” features, or more specifically, the ones that are supported by the entity-aware searcher. Figure 5 illustrates all the features we designed for our framework. In general, the features can be classified into two types: entity features \( f^e_k \), e.g., \( tf(e) \), and snippet features \( f^s_k \), e.g., \( \text{BeforeEntity} \[ “founded” \] (s) \). The ranking function \( R \) will rely on such features to estimate the relevance of an entity to the given query. For example, \( \text{BeforeEntity} \[ “founded” \] (s_{11})=1 \), which means “founded” occurs before “bill gates” in \( s_{11} \), indicates that \( s_{11} \) may serve as an evidence supporting the relation FounderOf(“microsoft”, “bill gates”). The value of entity features can be obtained when we are joining the query index with the entity index as shown in Figure 2; while to get the snippet features, we need to perform additional join with the selected entities as shown in Figure 3, which would be the major bottleneck of efficiency.

To achieve fast online retrieval, we need to limit the number of snippet features, which requires additional index checking. We set \( M \) to be the maximal number of the selected snippet features. As a result, the designed snippet features should yield relatively high recall to cover more snippets. Such requirement prohibits us from adopting the high-precision low-recall features in traditional information extraction work (e.g., [16] used all words between the query and the entity as one feature, like “which was recently founded by”). Conversely, we limit each \( w \) to be unigram (e.g., “founded”) or bigram (“founded by”) in our feature design.

The relation-specific ranking model is the focus of this paper. In the next section, we will formally study the problem of how to effectively learn such models with indirect supervision, i.e., the **distantly supervised ranking** problem.

### B. Distantly Supervised Ranking

In this section, we would formally define the **distantly supervised ranking** problem.
With respect to a target relation \( r \), e.g., FounderOf(), we are given a set of training queries. For each query \( q \), e.g., “microsoft” (to simplify the description in the following discussion, we only use this query for illustration purpose), we obtain a list of entities \( e = \{e_1, ..., e_{|e|}\} \) (e.g., “bill gates” and “paul allen”) returned by the entity-aware searcher; each entity \( e_i \) is associated with a set of snippets \( s_i = \{s_{i1}, ..., s_{ij}|s_i|\} \). \( s_{ij} \) is defined as a fixed-length text fragment where entity \( e_i \) and query \( q \) co-occur. We assign a label \( y_i \) to each \( e_i \): if the relation \( r(e_i, q) \) is true, \( e_i \) is labeled as positive (\( y_i = 1 \)); otherwise negative (\( y_i = 0 \)). Entity \( e_i \) and snippet \( s_{ij} \) are characterized by a set of ranking features \( f_{k}^{(e)} \) and \( f_{k}^{(s)} \) defined in Figure 5.

For efficiency concern, we have to limit the number of snippet features that require additional index checking. Denoting \( \mathcal{F}_{all} = \{f_{k}^{(e)}, ..., f_{k}^{(s)}|\mathcal{F}_{all}\} \) as the whole candidate snippet feature set, the final model can only choose \( M \) snippet features \( \mathcal{F} \subseteq \mathcal{F}_{all} \).

As our objective, we aim to learn a ranking function \( R(e_i, s_i) \) for the target relation \( r \), such that the candidate entities are ordered by the confidence of being in the true relation. Using the language of probability, \( R(e_i, s_i) \) can be described as \( P(y_i = 1|e_i, s_i) \), the probability of \( e_i \) being true given entity features \( f_{k}^{(e)} \), and snippet features \( f_{k}^{(s)} \in \mathcal{F} \). As a result, the goal of distantly supervised ranking is to find the optimal ranking function in the form of \( P(y_i = 1|e_i, s_i) \), which can correctly rank the annotated entities for the target relation, based on a limited size of snippet features satisfying \( |\mathcal{F}| \leq M \).

**IV. PATTERN-BASED FILTER NETWORK**

There are two challenges embedded in the distantly supervised ranking problem: one is **accuracy**, namely we need to filter noise inherited in the coarse entity-level annotations so as to estimate an accurate ranking function \( R(e_i, s_i); \) another is **efficiency**, that is we can only employ a small amount of “indexable” features when solving the problem.

As discussed in Section I, to address the two challenges, we would rely on the redundancy ranking principle — intuitively, a positive entity will have more relevant snippets as evidences; furthermore, as different relevant snippets will have different contributions (e.g., a snippet with small \( dis \) represents strong relatedness between the query and entity, and thus tends to be more important), we hypothesize that the positiveness of an entity should depend on the summation of the contribution, rather than the number, of its relevant snippets. If we define \( h_i = \{h_{i1}, ..., h_{ij}|h_i|\} \) where \( h_{ij} \in \{0, 1\} \) denotes whether \( s_{ij} \) is an evidence snippet and \( a_{ij} \) measures the contribution of \( s_{ij} \) to \( e_i \), the redundancy ranking principle could be formalized as follows.

**Definition 1 (Redundancy Ranking Principle):** Given entities \( e_i \) and \( e_j \) with the same entity features, we have

\[
\sum_{s_{ij} \in s_i \wedge h_{ij} = 1} a_{ij} > \sum_{s_{kl} \in s_k \wedge h_{kl} = 1} a_{kl} \quad \rightarrow \quad P(y_i = 1|h_i, e_i, s_i) > P(y_j = 1|h_j, e_j, s_j)
\]

where \( a_{ij} \) is defined as the weighted summation of the chosen snippet features of \( s_{ij} \),

\[
a_{ij} = \sum_{f_{k}^{(s)} \in \mathcal{F}} w_{k}^{(s)} f_{k}^{(s)}(s_{ij})
\]

where \( w_{k}^{(s)} \) is the importance weight of snippet features.

As the prerequisite to apply the redundancy ranking principle, one needs to determine which snippets should serve as evidences, i.e., \( h_{ij} = 1 \). As proposed in Section I, we utilize indicative patterns, some common patterns that are indicative of the target relation \( r \) (e.g., FounderOf()) between the entity \( e \) and query \( q \), to identify the evidence snippets. Formally, we define \( \mathcal{P} = \{P_1, ..., P_p\} \) as the indicative pattern set, where each indicative pattern is a binary function \( P_k(s_{ij}) \subseteq \{0, 1\} \) defined over the chosen snippet features \( \mathcal{F} \), e.g., \( P_k(s_{ij}) := \text{BeforeEntity}[^{\text{[founded by]}}] \wedge \text{dis} < 5 \). Assuming such an indicative pattern set \( \mathcal{P} \) is given (we will discuss how to learn \( \mathcal{P} \) in the later section), we can predict if a snippet \( s_{ij} \) is a positive evidence by matching it against \( \mathcal{P} \) as follows.

**Definition 2 (Evidence Snippet):** For a given set of indicative patterns \( \mathcal{P} \), snippet \( s_{ij} \) is an evidence snippet (i.e., \( h_{ij} = 1 \)) if and only if \( \exists P_k \in \mathcal{P}, P_k(s_{ij}) = 1 \). We use \( s_i^{(P)} := \{s_{ij} \in s_i | \exists P_k \in \mathcal{P}, P_k(s_{ij}) = 1\} \leq s_i \) to denote the evidence snippet set for \( e_i \).

As discussed in Section I, we can address the two challenges based on the redundancy ranking principle defined upon the notion of evidence snippet: as such indicative patterns commonly exist in relevant snippets across different positive entities, noisy snippets can be filtered by the redundancy of indicative patterns; in addition, the diversity of such patterns is usually small, which renders us the opportunity to limit the number of patterns without hurting the accuracy.

As a result, there are two objectives clearly stated in the redundancy ranking principle to address the two challenges: first, filtering noisy snippets, and second, scoring the positive snippets based on chosen evidences. To fulfill this modeling assumption, we propose Pattern-based Filter Network (PFNet), which consists of \( |e| \) tree-structured Markov networks [14], as shown in Figure 6. The proposed PFNet forms a two-layer tree network.
structure and each layer corresponds to one particular modeling objective:

- **Noise Filtering**: At the leaf layer (bottom), PFNet selects a set of indicative patterns $P$ to build up factor $\psi_{ij}(h_{ij})$ for effectively distinguishing noisy snippets, i.e., $P(h_{ij}|s_i)$.

- **Evidence Aggregation**: At the root layer, to predict $y_i$, i.e., the conditional probability of $P(y_i|h_i,e_i,s_i)$, PFNet uses $\tau_i(y_i)$ to model the features from entity $e_i$ itself, and $\phi_{ij}(y_i,h_{ij})$ for modeling the redundancy within snippets. In the aggregation of snippet redundancy, $\phi_{ij}(y_i,h_{ij})$ only aggregate the contribution $a_{ij}$ of chosen evidence snippets $s_i$ with $h_{ij} = 1$.

As a result, each tree in PFNet specifies the conditional probability of $P(y_i|h_i|e_i,s_i)$ for an entity $e_i \in e$ being positive (e.g., those underlined $e_i$ in Figure 1) and the snippet $s_i$ being an evidence snippet by the factors of $\psi_{ij}(h_{ij})$, $\phi_{ij}(y_i,h_{ij})$ and $\tau_i(y_i)$, i.e.,

$$P(y_i, h_i|e_i, s_i) = P(y_i|h_i,e_i,s_i)P(h_i|s_i) \propto \tau_i(y_i) \prod_{j=1} \phi_{ij}(y_i,h_{ij}) \psi_{ij}(h_{ij})$$  \hspace{1cm} (3)

In the following discussion, we will illustrate the design of each layer in PFNet in detail.

### A. Noise Filtering

As required by the redundancy ranking principle, to avoid distraction from noisy snippets, we use indicative patterns to identify evidence snippets, encoded in $\psi_{ij}(h_{ij})$. In a standard Markov network, factors are defined via exponential functions. However, due to the positiveness of an exponential function, all snippets, including the noisy ones, will have non-zero probabilities of being an evidence, especially when the volume of noisy snippets is larger than the evidence snippets. This will bias the prediction of $y_i$ for entity $e_i$. Moreover, this setting can hardly scale up to a large data set: one has to repeatedly infer $h_{ij}$ when estimating the model, which makes the training process prohibitively slow. As a result, we adopt a “hard filter” design for $\psi_{ij}(h_{ij})$ as,

$$\psi_{ij}(h_{ij}) = \begin{cases} 1 & \text{if } s_{ij} \in s_{(P)} \land h_{ij} = 1 \lor \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (4)

where $\psi_{ij}(h_{ij})$ is modeled as a binary factor to emphasize the effect of matching against the indicative pattern set $P$.

### B. Evidence Aggregation

The root layer of PFNet models the prediction of $y_i$, which indicates if $e_i$ is a positive entity (thus an answer to the query). Specifically, we want to specify the interaction between $y_i$ and the latent variables $h_{ij}$ and observations $(e_i$ and $s_i)$ by factors $\tau_i(y_i)$ and $\phi_{ij}(y_i,h_{ij})$.

Factor $\tau_i(y_i)$ is designed to capture the contribution from the general entity features on $e_i$. One property that $\tau_i(y_i)$ should follow is that when we failed to find any evidence for $e_i$, it is more likely to be a negative entity. Given a set of indicative patterns $P$, denote $e^{(P)}$ as the set of covered entities, where each entity should have at least one evidence snippet. Formally, we define $e^{(P)} := \{ e_i \in e|s_i \notin (P) \neq \emptyset \}$. If $e_i \notin e^{(P)}$, we should give $y_i = 1$ a small probability; otherwise, the prediction of $y_i$ depends on its features $f_k(e)$. To achieve this requirement, we define $\tau_i(y_i)$ as,

$$\tau_i(1) = \begin{cases} \varepsilon & \text{if } e_i \notin e^{(P)} \\ \exp[\sum_k w_k(e) f_k(e)] & \text{if } e_i \in e^{(P)} \end{cases}$$ \hspace{1cm} (5)

where $\varepsilon$ is a small positive constant. Note that, we set $\tau_i(0) = 1$, since only the relative value between $\tau_i(1)$ and $\tau_i(0)$ matters in the Markov network.

$\phi_{ij}(y_i,h_{ij})$ explicitly encodes the aggregation of snippet contribution in the redundancy ranking principle. First, for noisy snippets (with $h_{ij} = 0$), we set $\phi_{ij}(0, 0) = \phi_{ij}(1, 0) = 1$, indicating the existence of noisy snippets should not affect the prediction of $y_i$. Second, for an evidence snippet (with $h_{ij} = 1$), how $s_i$ affects $P(y_i = 1|h_i, e_i, s_i)$ depends on its contribution $a_{ij}$. Similar to Eq. 5, we use the exponential function to incorporate $a_{ij}$ in $\phi_{ij}(1, 1)$ and set $\phi_{ij}(0, 1)$ to be 1 as

$$\phi_{ij}(y_i,h_{ij}) = \begin{cases} \exp(a_{ij}) & \text{if } y_i = 1 \land h_{ij} = 1 \\ 1 & \text{otherwise} \end{cases}$$  \hspace{1cm} (6)

We can observe that Eq. 6 strictly reflects the redundancy ranking principle, i.e., the resulting $P(y_i = 1|h_i, e_i, s_i)$ is a function monotonically increase with $\sum_{s_i \in s_i \land h_{ij} = 1} a_{ij}$. By definition in Eq. 3, we have

$$P(y_i = 1|h_i, e_i, s_i) = \frac{\tau_i(1) \prod_j \phi_{ij}(1, h_{ij})}{\tau_i(1) \prod_j \phi_{ij}(1, h_{ij}) + \tau_i(0) \sum_j \phi_{ij}(0, h_{ij})}$$

$$\frac{\tau_i(1) \exp[\sum_{s_i \in s_i \land h_{ij} = 1} a_{ij}]}{1 + \tau_i(1) \exp[\sum_{s_i \in s_i \land h_{ij} = 1} a_{ij}]}$$  \hspace{1cm} (7)

which clearly satisfies the monotonic property.

### C. Objective of PFNet

The graph structure described in Figure 6 with factors designed in Eq. 4, Eq. 5 and Eq. 6 together define our PFNet model, which represents the conditional probability $P(y_i|h_i,e_i,s_i)$, as defined in Eq. 3. Due to the lack of detailed snippet-level labels $h_{ij}$ in training data, the only supervision we have comes from $P(y_i|e_i, s_i)$. To obtain the representation of $P(y_i|e_i, s_i)$, we need to marginalize $P(y_i|h_i,e_i,s_i)$ over all possible configurations of $h_i$:

$$P(y_i|e_i, s_i) = \sum_{h_i} P(y_i|h_i,e_i,s_i) \propto \sum_{h_i} \tau_i(y_i) \prod_{j=1} \phi_{ij}(y_i,h_{ij}) \psi_{ij}(h_{ij})$$  \hspace{1cm} (8)

Because of the product term in Eq. 8, only one specific $h_i$ configuration leads to a non-zero $P(y_i|h_i,e_i,s_i)$ (assign 1 only
where \( \hat{h}_i(\mathbf{P}) \) is the configuration of \( h_i \) corresponding to the given \( \mathbf{P} \), and for each \( \hat{h}_ij(\mathbf{P}) \in \hat{h}_i(\mathbf{P}) \), \( \hat{h}_ij(\mathbf{P}) = 1 \) if \( s_{ij} \in s_i(\mathbf{P}) \) and \( \hat{h}_ij(\mathbf{P}) = 0 \) otherwise.

By substituting \( \tau_i(y_i) \) and \( \phi_{ij}(y_i, 1) \) in Eq. 9 with Eq. 5 and Eq. 6, we obtain the complete formula of \( P(y_i = 1|e_i, s_i) \) as,

\[
P(y_i = 1|e_i, s_i) = \left\{ \frac{\frac{\hat{h}_i(\mathbf{P})}{1 + e^{-\frac{1}{\lambda} - s_i(e_i, \mathbf{F}, \mathbf{P})}}}{e_i \notin e(\mathbf{P})} e_i \in e(\mathbf{P}) \right. \tag{10}
\]

where \( \bar{w} \) is a feature weighting vector including \( w_k^{(c)} \) and \( w_k^{(s)} \), and \( d_i(\bar{w}, \mathbf{F}, \mathbf{P}) \) is defined as follows,

\[
d_i(\bar{w}, \mathbf{F}, \mathbf{P}) = \sum_k w_k^{(c)} f_k^{(c)}(e_i) + \sum_{s_{ij} \in s_i(\mathbf{P})} w_k^{(s)} f_k^{(s)}(s_{ij}) \tag{11}
\]

Eq. 10 clearly illustrates the probability of an entity being positive depends on its own features \( f^{(c)} \) and the snippet features \( f^{(s)} \) from the associated evidence snippets: if \( e_i \) is an entity without any evidence snippet, \( P(y_i = 1|e_i, s_i) \) equals to a small constant probability \( \frac{1}{e^{\lambda} + 1} \); otherwise, \( P(y_i = 1|e_i, s_i) \) depends on \( d_i(\bar{w}, \mathbf{F}, \mathbf{P}) \), the weighted summation of features from \( e_i \) and its evidence snippets \( s_{ij} \in s_i(\mathbf{P}) \), incorporated in a logistic function.

With \( P(y_i = 1|e_i, s_i) \) defined in Eq. 10, following the maximal likelihood principle, our objective is to find the optimal set of feature set \( \mathbf{F} \), indicative pattern set \( \mathbf{P} \), and feature weighting \( w_k^{(c)} \) and \( w_k^{(s)} \), which maximizes the log likelihood \( \mathcal{L}(\mathbf{F}, \bar{w}, \mathbf{P}; y) \) over the annotated entities for the relation selection.

Note that in entity ranking task, the data set is usually heavily unbalanced: given a query, there are a lot more negative entities than positive ones. Therefore, we set a cost parameter \( \lambda > 1 \) to emphasize the importance of positive entities. Thus, the objective function \( \mathcal{L}(\mathbf{F}, \bar{w}, \mathbf{P}; y) \) is given as follows,

\[
\arg\max_{\mathbf{F}, \bar{w}, \mathbf{P}} \mathcal{L}(\mathbf{F}, \bar{w}, \mathbf{P}; y) = \arg\max_{\mathbf{F}, \bar{w}, \mathbf{P}} \sum_{e_i \in e(\mathbf{P})} \lambda y_i \log \frac{e}{1 + e} + (1 - y_i) \log \frac{1}{1 + e} + \sum_{e_i \in e(\mathbf{P})} \lambda y_i \log \frac{e - d_i(\bar{w}, \mathbf{F}, \mathbf{P}) + 1}{e - d_i(\bar{w}, \mathbf{F}, \mathbf{P}) + 1} + (1 - y_i) \log \frac{e - d_i(\bar{w}, \mathbf{F}, \mathbf{P}) + 1}{e - d_i(\bar{w}, \mathbf{F}, \mathbf{P}) + 1} \tag{12}
\]

subject to \( |\mathbf{F}| \leq M \)
Algorithm 1: Algorithm for Learning PFNet

Input: $y_i, f_k^{(c)}, F_{all}, M$.
Output: $F \subseteq F_{all}, P, \hat{w}$

1: $F^{(0)} \leftarrow \emptyset$, $P^{(0)} \leftarrow \emptyset$, $t \leftarrow 1$.
2: while $|F| \leq M$ do
3: $l_{best} \leftarrow -\infty$, $f_{best} \leftarrow null$, $\pi_{best} \leftarrow null$.
4: for $f_k^{(s)} \in F_{all}$ do
5: Optimize $\hat{w}$ to calculate $l_i^{(f)}(f_k, 0)$ according to Eq. 13.
6: if $l_i^{(f)}(f_k, 0) \geq l_{best}$ then
7: $l_{best} \leftarrow l_i^{(f)}(f_k, 0)$, $f_{best} \leftarrow f_k$, $\pi_{best} \leftarrow 0$.
8: end if
9: Optimize $\hat{w}$ compute $l_i^{(f)}(f_k, 1)$ according to Eq. 14.
10: if $l_i^{(f)}(f_k, 1) \geq l_{best}$ then
11: $l_{best} \leftarrow l_i^{(f)}(f_k, 1)$, $f_{best} \leftarrow f_k$, $\pi_{best} \leftarrow 1$.
12: end if
13: end for
14: $F^{(t)} \leftarrow F^{(t-1)} \cup \{f_{best}\}$
15: if $\pi_{best} = 1$ then
16: $P^{(t)} \leftarrow P^{(t-1)} \cup \{f_{best}\}$
17: end if
18: $t \leftarrow t + 1$.
19: end while
20: return $F, P, \hat{w}$.

To calculate Eq. 13 and Eq. 14 in each iteration, we use an inner loop to search the optimal $\hat{w}$ by gradient ascend, according to their gradients defined as follows,

$$\frac{\partial \log \mathcal{L}}{\partial w_k^{(c)}} = \sum_{e_i \in d(P)} [y_i - P(y_i = 1 | e_i, s_i)] f_k^{(c)} (e_i)$$

(15)

$$\frac{\partial \log \mathcal{L}}{\partial w_k^{(s)}} = \sum_{e_i \in d(P)} [y_i - P(y_i = 1 | e_i, s_i)] \sum_{s_{ij} \in s_i^{(p)}} f_k^{(s)} (s_{ij})$$

(16)

Note that such an optimization procedure is still inefficient since we have to search for the optimal $\hat{w}$ for every candidate feature. To further improve the efficiency, we decide to relax it: when testing a new feature $f_k^{(s)}$, we keep $\hat{w}$ from the last iteration unchanged, and only use the inner loop to find the optimal $w_k^{(s)}$ for the new feature. The whole vector $\hat{w}$ will be updated only after the best feature is added. Such a strategy avoid updating the whole vector $\hat{w}$ during feature testing.

V. EXPERIMENT

A. Experiment Setting

We chose around 100 million general English Web pages (about 20%) from the ClueWeb09 dataset as our testbed. To retrieve entities, we used our previously built system [22] (distributed over 30 nodes) as the entity-aware searcher to index the dataset. A list of entity types were extracted and indexed, including general entities (e.g., \#person, \#organization and \#location) extracted by Stanford NER toolkit [10].

When choosing the evaluation collection, we consider three criteria: first, the chosen collections should cover different target types to evaluate the model’s capability over different types of relations; second, they should have different quality of evidence snippets in order to show the influence of noise on ranking performance; third, they should also include less popular queries to validate the capability of PFNet on those collections with less redundancy.

Based on the above consideration, we collected six sets of different relations, two from Freebase (FounderOf() and PublisherOf()), three from Wikipedia (WriterOf(), PlaceOfBirth() and PlaceOfDeath()), and one from the Mathematics Genealogy Project \(^1\) (GraduateFrom()). As shown in Figure 7, they cover three different target types; in particular, the GraduateFrom() collection is much smaller than others, because the queries used in the collection are the names of graduate students, which occur much less frequently on the Web than the queries from Wikipedia and Freebase.

We can take the FounderOf() relation as an example to illustrate our procedure for collecting the evaluation data with entity-level annotations. Given the relation FounderOf(“microsoft,” \{"“bill gates,” “paul allen,”\}) we used the selected entry terms from this target relation as queries (e.g., “microsoft”) to search in our ClueWeb09 dataset, and retrieved all the entities matching the required target type (e.g., \#person for FounderOf()) together with the associated snippets. The retrieved entities will be labeled accordingly, e.g., “bill gates” and “paul allen” are labeled as positive and all the others as negative.

PFNet has two free parameters, i.e., \(\lambda\) and \(\varepsilon\), to be tuned. We used 5-fold cross validation to find the optimal parameters through extensive experiments for each collection. We used \(M = 10\) in all the datasets (later we would investigate the impact of feature size), and used 5-fold cross validation to confirm the confidence of the comparison with standard t-test.

B. Performance Comparison with Baselines

We employed five baseline methods to validate the performance of the proposed PFNet. First, to demonstrate the necessity of learning relation-specific ranking functions, we compared PFNet with EntityRank [6], a general ranking algorithm without distinguishing entity types. Second, as our task is formalized as a distantly supervised problem, we compared PFNet with Multi-Instance Learning (MIL) – a distant supervision model introduced in [19]. We designed two versions of MIL as baselines: MIL-All using all features, and MIL-IG using only top \(M\) snippet features chosen by the standard information gain criterion [21], to demonstrate the effectiveness of the feature selection component in PFNet. Third, since our task can also be viewed as a learning-to-rank problem, we

\(^1\)http://genealogy.math.ndsu.nodak.edu/
compared PFNet with SVMRank, a state-of-the-art learning to rank model [13]. Similar to MIL, we employed both SVMRank-All and SVMRank-IG as baselines. The details of five baseline models are introduced as follows,

1) EntityRank [7]: this work aims at designing a general entity ranking function. Different from PFNet, EntityRank relies on user-specified keywords to represent the target relation (e.g., “microsoft founder #person,” where “founder” is treated as a keyword like “microsoft”), and thus, does not require any training data. To conquer the possible variants of keywords used to describe the relation, we employed the patterns learned by PFNet as keywords instead of manually designating as in the original work.

2) Multi-instance Learning with All Features (MIL-All) [19]: this work shares similar motivation as ours, which aims to extract relations from text data according to distant supervision. To overcome noise, MIL assumes there has to be at least one positive snippet for a positive relation, and no positive snippet for a negative relation. Different from PFNet, MIL does not explicitly model redundancy. To make a fair comparison, we fed MIL with all features defined in PFNet.

3) Multi-instance Learning with Information Gain (MIL-IG): As our problem requires to use only a small number of features, MIL-IG uses a standard information gain criterion to choose the top $M$ features. To calculate information gain for each feature, we assume all snippets in positive entities are positive and those in negative entities are negative. The chosen features are then fed into MIL for learning purpose.

4) SVMRank with All Features (SVMRank-All): this baseline treats the distantly supervised ranking problem as a standard learning to rank problem. To adapt SVMRank, we construct the feature vector for each entity by inte-

5) SVMRank with Information Gain (SVMRank-IG): Similar to MIL-IG, this baseline only uses top $M$ features chosen by information gain criterion.

Since the relational entity search task is mostly search-oriented, we employ standard information retrieval performance metrics [12], e.g., NDCG@k, MAP and Precision@k, as our evaluation criteria. As Figure 7 shows, there are only one or two positive entities for most of queries, such as “facebook founder” and “publisher of starcraft,” we calculate NDCG@5 and Precision@3 as the performance metrics. The ranking performance of our proposed PFNet and all the other five baseline methods on the six collections are shown in Figure 8.

The results demonstrate that PFNet outperforms all the baselines in six collections. In particular, PFNet achieves encouraging improvement of NDCG@5 and MAP against the runner-ups in the PublisherOf (NDCG@5 +23.7%, MAP +24.9%), WriterOf (NDCG@5 +22.0%, MAP +23.1%), PlaceOfBirth (NDCG@5 +14.4%, MAP +17.1%), PlaceOfDeath (NDCG@5 +14.4%, MAP +10.7%) collections (p-value<0.05 in all cases). We also observe that the improvement of PFNet decreases in the GraduateFrom (NDCG@5 +6.3%, MAP +5.9%) and FounderOf (NDCG@5 +9.1%, MAP +13.4%) collections. By analyzing the results, we find that PFNet’s performance gain is closely related to the percentage of noisy snippets in the positive entities. To verify this, we sampled 300 snippets from positive entities in each collection, and manually labeled each snippet as evidence or noise, as demonstrated in Figure 9. The result shows that on those more noisy collections (more than 70% noisy snippets), i.e., PublisherOf(), WriterOf(), PlaceOfBirth() and PlaceOfDeath(), PFNet achieves significant improvement; while for the collections of FounderOf() and GraduateFrom(), since they are less noisy, the baseline methods can already achieve satisfactory results. Such results confirm that noise filtering plays an important role in our distantly supervised ranking problem.

From the comparisons in Figure 8 we can clearly notice the advantage of PFNet over the baseline methods. MIL-All does not perform well in our data sets because it is developed for traditional information extraction task and heavily depends on a large number of high-precision low-recall features to distinguish noise. Without leveraging redundancy, it fails when only light-weighted features are available. Although EntityRank uses all the patterns learned by PFNet, it works in a relation-independent manner and does not work well. For the SVMRank-All baseline, though it models the concept of redundancy, it uses all snippets in training without distinguishing the noise. By comparing SVMRank-IG with
SVMRank-All, and MIL-IG with MIL-All, we can find that the information-gain-based feature selection does improve the ranking performance; however, the improvement is limited. The reason is that when calculating the information gain of each feature, we assume all snippets in positive entities are positive, which makes information gain criterion vulnerable to noisy snippets. Comparing to SVMRank-IG and MIL-IG, PFNet explicitly depends on the automatically learned indicative patterns to filter out the noisy snippets and thus achieves better performance.

C. Influence of Redundancy on Ranking Performance

In this section, we will investigate the influence of redundancy on the model’s ranking performance. In particular, we are interesting in two questions. First, in the training phase, how crucial the redundancy is for learning a robust and accurate model? Second, in the searching phase, will a ranking model’s performance vary on the queries of different redundancy? Note that although five of our collections come from Wikipedia and Freebase, the collections still contain a lot of unpopular queries that are associated with only a small number of snippets.

To analyze how much data is needed in the training phase, we fixed 60 queries in each collection as testing data, and varied the number of training queries from 30 to 150. The ranking performance of PFNet over different size of training relations is demonstrated in Figure 10. The result shows that PFNet benefits from more training relations; whereas, the performance improvement differs in different categories. In particular, from 30 to 150 relations, PFNet achieves significant improvement in PublisherOf (NDCG@5 +17.4%), WriterOf (NDCG@5 +19.7%), PlaceOfBirth (NDCG@5 +19.2%) and PlaceOfDeath (NDCG@5 +50.3%) collections. While the improvement gets diminished in FounderOf (NDCG@5 +10.7%) and GraduateFrom (NDCG@5 +8.6%) collections. Referring to Figure 9, we notice that such difference is also highly related with the noise level of the collection. For collections containing less noisy snippets (i.e., FounderOf() and GraduateFrom()), the learning task is relatively easier. Therefore, PFNet can achieve promising performance with a small number of training relations and does not benefit much from more training data; while for noisy collections, more training data is beneficial. In general, we observe that when the relation size increases to more than 120, the ranking performance becomes stable. It demonstrates that a median size of training data (e.g., 120 queries) is generally sufficient for PFNet to estimate a good ranking model.

To evaluate the ranking performance of PFNet on queries of different redundancy, we classified queries into 3 categories: low-popularity (where positive entities have less than 10 snippets), medium-popularity (10 to 80 snippets) and high-popularity (more than 80 snippets). We sampled 20 queries from each category to construct 3 testing sets and used the remaining queries as training data. We used SVMRank-IG, which is the runner-up in Figure 8, as the baseline. The performance of PFNet and SVMRank-IG over queries of different popularity is demonstrated in Figure 11. The result fits our intuition: both PFNet and SVMRank-IG achieve better performance on high-popularity queries comparing to low-popularity ones. The explanation is intuitive: popular entities are more likely to contain evidence snippets compared with the uncommon entities. As a result, the redundancy ranking principle renders both PFNet and SVMRank-IG the capability of deriving more confident results for the popular entities.

However, exploiting redundancy does not necessarily lead to the conclusion that PFNet would completely fail for low-popularity queries. As Figure 11 shows, PFNet achieves comparable performance over low-popularity queries comparing to the high-popularity ones, e.g., GraduateFrom (-8%) and PlaceOfBirth (-15%); while for the FounderOf() relation, low-popularity queries even performs better (+8%). By analyzing the result, we observe that PFNet achieves better performance on the collections with more covered entities (the entities which are associated with at least one evidence snippet matched by the identified indicative patterns). As demonstrated in Figure 12, both FounderOf() and GraduateFrom() have all low-popularity entities covered, where PFNet performs well over the low-popularity queries. And from another perspective, more importantly, such results demonstrate that the patterns identified by PFNet are general and effective, which are able to discover uncommon entities and achieve promising performance on the low-popularity queries as well.

By comparing PFNet with SVMRank-IG, we can find that PFNet outperforms SVMRank-IG in all categories and PFNet achieves more improvement over low-popularity and medium-
popularity queries compared with high-popularity queries (average improvement on low-popularity queries: 45.9%, medium-popularity queries: 38.3%, high-popularity queries: 16.2%). The result demonstrates redundancy helps the model combat with noisy snippets, and thus even the baseline method (SVMRank-IG) could benefit from it. The comparison also demonstrates the capability of PFNet in handling low-popularity and medium-popularity queries.

D. Performance Comparison with Different Feature Size

In this section, we are going to investigate the effect of the number of features in our PFNet ranking framework. There are two important aspects to be assessed: first, how would number of chosen features affect the final ranking performance; second, how would the number of chosen features affect the execution efficiency.

Figure 13 shows PFNet’s NDCG@5 performance with different number of features. We find that NDCG@5 improves a lot from 1 feature to 10 features: the improvement is over 3% for the PlaceOfBirth() relation, and over 10% for all other relations. While from 10 features to 20 features, the improvement gets diminished, less than 5% for all the relations. By analyzing the results, we find that the results are heavily affected by the “diversity” of a relation representation, i.e., how many different patterns are used to characterize the relation. For example, the PlaceOfBirth() relation is usually described by a small number of patterns such as “born in” and “birth place,” and therefore the model can achieve quite satisfactory results with less than 5 features. While for the GraduateFrom relation, people usually do not explicitly mention “graduated from” but rather use a variety of other patterns such as “studied in,” “got phd from” and so on. As the result, PFNet benefits more from using a larger set of features in this kind of relations.

In addition to the effect on the ranking performance, we also need to analyze the impact of the number of selected features on the ranking efficiency, because the more features we select the more indexes we have to check during the online execution phase. In this experiment, we sampled 60 queries from each data set and calculated the average execution time over all the collections in Figure 14. In PFNet, the overhead increases linearly with the size of selected features. By considering the performance improvement gained from the additional patterns (in Figure 13), a median size of indicative patterns (e.g., \( M = 10 \)) would be a good trade-off.

We also compared our choice of the inverted-index-based feature fetching approach to the document-checking-based approach, which was introduced in Section II, in Figure 15. The result demonstrates that the document-checking approach is much less efficient than inverted index checking (about 2 to 3 times slower). Such significant efficiency improvement confirms the necessity of choosing only indexable features in our online relational entity search task.

E. Case Study

To give an intuitive illustration of relational entity search task and diagnose the quantitative ranking performance, in Figure 16, we perform case studies by showing the top 3 entities returned by PFNet for some typical queries in our data set. We observe that PFNet can often return “positive” entities in front, but they are not listed in Freebase. For example, “bill bowerman,” as a co-founder of “nike inc,” is not recorded in Freebase, but PFNet ranks it at the 2nd position. The same case is for “michael bolton” to the query “the dock of the bay.”

Besides, we also observe some limitations of PFNet. First, PFNet can not handle ambiguous queries: “wolfgang vogit” is ranked at the first place by PFNet for query “kompakt.”

---

**Fig. 13.** Ranking performance of PFNet with different feature number.

**Fig. 14.** Average execution time per query with different feature number.

**Fig. 15.** Efficiency comparison (in ms) between two feature fetching methods: inverted-index-based (10 patterns) and document-checking-based.

**Fig. 16.** Query example and top-3 returned entities (underlined entities are ground-truth from Freebase).

**Fig. 17.** Discovered indicative patterns.
however he was the founder of another “kompakt,” a company different from the one founded by “michael mayer.” Second, PFNet fails to distinguish the synonyms of the positive entities. For example, “phil h knight” is ranked higher than the ground-truth “phil knight” simply because the former is more frequently mentioned in the corpus; and with respect to the query “terrel davis,” “san diego,” “california” and “american” are all correct answers but only differ in resolution. We can expect that some post-processing techniques, e.g., entity name disambiguation, can further improve the ranking performance of PFNet.

Besides, since it is hard to directly evaluate the quality of the identified indicative patterns by PFNet, we list some of those patterns in Figure 17 for qualitative analysis purpose. From the result, we find that most of the patterns match our intuition of the tasks: for example, in the PublisherOf() relation, the model extracts “published by” and “released by” to capture the different types of “publisher” in Web pages. Some patterns that do not directly indicate the target relation are also discovered, e.g., “grew,” “graduated” for PlaceOfBirth() relation. And for the WriterOf() relation, PFNet also discovered patterns such as BeforeEntity[“director”]. By looking into the data, we find that the data set contained many movie names as queries, and for a large portion of such movies, the director is also the writer of that movie (e.g., “bruce lee” is the writer and director of the movie “game of death”).

VI. CONCLUSION

In this paper, we proposed a novel framework for the relational entity search task, where we focused on the distantly supervised ranking problem – given only coarse entity-level annotations, how to effectively derive a set of relation-specific ranking functions for online searching purpose. As our solution, we proposed PFNet, a Markov network model based on limited size of “indexable” features. By exploiting the redundancy within evidence snippets, PFNet automatically selects a set of indicative patterns for noise filtering and estimates the relation-specific ranking models in a unified way. Our experiment results on six different collections from CuleWeb09 dataset confirm the effectiveness of PFNet.

As we have discussed in the experiment, with the current annotation strategy, “positive” entities might be mistakenly annotated as “negative” because of the limited coverage of existing knowledge base. As a result, directly labeling them as negative might introduce inconsistency into model learning. It would be interesting to study how to model such unlabeled entities by exploiting the dependency among different entities in a more appropriate way. Moreover, we also found that the current framework cannot deal with synonyms and entities at different resolutions very well. As our future work, we will investigate the entity disambiguation technique in relational entity search framework to further improve the ranking results.

REFERENCES