Learning to Rank Adaptively for Scalable Information Extraction

Pablo Barrio
Columbia University
pjbarrio@cs.columbia.edu

Gonçalo Simões
INESC-ID and IST, Universidade de Lisboa
goncalo.simoes@tecnico.ulisboa.pt

Helena Galhardas
INESC-ID and IST, Universidade de Lisboa
helena.galhardas@tecnico.ulisboa.pt

Luis Gravano
Columbia University
gravano@cs.columbia.edu

ABSTRACT

Information extraction systems extract structured data from natural language text, to support richer querying and analysis of the data than would be possible over the unstructured text. Unfortunately, information extraction is a computationally expensive task, so exhaustively processing all documents of a large collection might be prohibitive. Such exhaustive processing is generally unnecessary, though, because many times only a small set of documents in a collection is useful for a given information extraction task. Therefore, by identifying these useful documents, and not processing the rest, we could substantially improve the efficiency and scalability of an extraction task. Existing approaches for identifying such documents often miss useful documents and also lead to the processing of useless documents unnecessarily, which in turn negatively impacts the quality and efficiency of the extraction process. To address these limitations of the state-of-the-art techniques, we propose a principled, learning-based approach for ranking documents according to their potential usefulness for an extraction task. Our low-overhead, online learning-to-rank methods exploit the information collected during extraction, as we process new documents and the fine-grained characteristics of the useful documents are revealed. Then, these methods decide when the ranking model should be updated, hence significantly improving the document ranking quality over time. Our experiments show that our approach achieves higher accuracy than the state-of-the-art alternatives. Importantly, our approach is lightweight and efficient, and hence is a substantial step towards scalable information extraction.

1. INTRODUCTION

Information extraction systems are complex software tools that discover structured information in natural language text. For instance, an information extraction system trained to extract tuples for an *Occurs-in*(*NaturalDisaster*, *Location*) relation may extract the tuple <tsunami, Hawaii> from the sentence: "*A tsunami swept the coast of Hawaii.*" Having information in structured form enables more sophisticated querying and data mining than what is possible over the natural language text. Unfortunately, information extraction is a time-consuming task. A state-of-the-art information extraction system to extract *Occurs-in* tuples may take more than two months to process a collection containing about 1 million documents. Since document collections routinely contain several millions of documents, improving the efficiency and scalability of the extraction process is critical, even over highly parallel computation environments.

Interestingly, extracting a relation of interest with a properly trained information extraction system rarely requires processing all documents of a collection: Many times only a small set of documents produces tuples for a given relation, because relations tend to be topic-specific, in that they are associated mainly with documents about certain topics. For example, only 1.69% out of the 1.03 million documents in collections 1-5 from the TREC conference[1] produce *Occurs-in* tuples when processed with a state-of-the-art information extraction system and, not surprisingly, most of these documents are on environment-related topics. If we could identify the small fraction of documents that lead to the extraction of tuples, we would extract all tuples while decreasing the extraction time by over 90% without any need to change the information extraction system.

To identify the documents that produce tuples for an extraction task, which we refer to as the *useful* documents, existing techniques (e.g., QXtract [2], PRDualRank [14], and FactCrawl [7]) are based on the observation that such documents tend to share words and phrases that are specific to the extraction task at hand. For example, documents containing mentions of earthquakes—hence useful for the *Occurs-in* relation—many times include words like "richter" or "hypocenter." These words and phrases can then be used as keyword queries, to retrieve from the collection the (hopefully useful) documents that the extraction system will then process. To discover these words and phrases, a critical step in the process, these techniques analyze a sample of documents from the collection of interest. The size of this document sample is necessarily small to keep the overhead of the querying approach at reasonable levels.

Unfortunately, small document samples are unlikely to reflect the typically large variations in language and content

[1]http://trec.nist.gov/data.html

that useful documents for an extraction task may exhibit. For example, a document sample for *Occurs-in* may not include any documents on (relatively rare) volcano eruptions, and hence these techniques may fail to derive queries such as [lava] or ["sulfuric acid"] that would retrieve relevant, volcano-related documents. As a result, the queries from existing techniques may suffer from low recall during extraction. Furthermore, precision is also compromised: standard keyword search identifies documents whose topic is relevant to the queries, without considering their relevance to the information extraction task at hand.

To alleviate the precision-related issue above, FactCrawl [7] moves a step beyond keyword search: after retrieving documents with sample-derived keyword queries, FactCrawl reranks the documents according to a simple function of the number and "quality"—based on their F-measure [27]—of the queries that retrieved them, thus helping prioritize the extraction effort. However, FactCrawl exhibits two key weaknesses: (i) for document retrieval and ranking, FactCrawl relies on queries derived, once and for all, from a document sample, and hence suffers from the sample-related problems discussed above; (ii) for document ranking, FactCrawl relies on a coarse, query-based document scoring approach that is not adaptive (i.e., the scoring function does not change as we observe new documents). Therefore, this approach does not benefit from the information that is captured as the extraction process progresses.

In this paper, we advocate an adaptive document ranking approach that addresses the above limitations of the state-of-the-art techniques. Specifically, we propose a principled, efficient learning-to-rank approach that prioritizes documents for an information extraction task by combining: (i) online learning [30], to train and adapt the ranking models incrementally, hence avoiding computationally expensive retrains of the models from scratch; and (ii) in-training feature selection [17], to identify a compact, discriminative set of words and phrases from the documents to train ranking models effectively and efficiently. Importantly, our approach revises the document ranking decisions periodically, as the ongoing extraction process reveals (fine-grained) characteristics of the useful documents for the extraction task at hand. Our approach thus manages to capture, progressively and in an adaptive manner, the heterogeneity of language and content typically exhibited by the useful documents, which in turn leads to information extraction executions that are substantially more efficient—and effective—than those with state-of-the-art approaches, as we will see. In summary, we present an end-to-end document ranking approach for effective and efficient information extraction in an adaptive, online, and principled manner. Our main contributions are:

- Two low-overhead ranking algorithms for information extraction based on learning-to-rank strategies. These algorithms perform online learning and in-training feature selection (Section 3.1).
- Two techniques to detect when adapting the ranking model for information extraction is likely to have a significantly positive impact on the ranking quality (Section 3.2).
- An experimental evaluation of our approach using multiple extraction tasks implemented with a variety of extraction approaches (Sections 4 and 5). Our approach has low overhead and manages to achieve higher accuracy than the state-of-the-art approaches, and hence is a substantial step towards scalable information extraction.

2. BACKGROUND AND RELATED WORK

Information extraction systems extract structured information from natural language text. For instance, an extraction system properly trained to extract tuples of an *Occurs-in*(*NaturalDisaster*, *Location*) relation might extract the tuple <tsunami, Hawaii> from the sentence: "*A tsunami swept the coast of Hawaii.*" These systems often rely on computationally expensive processing steps and, consequently, processing all documents exhaustively becomes prohibitively time consuming for large document collections [2]. Ideally, we should instead focus the extraction effort on the *useful* documents, namely, the documents that produce tuples when processed with the information extraction system at hand.

As a crucial task, information extraction optimization approaches (e.g., Holistic-MAP [31]) choose a document selection strategy to identify documents that are likely to be useful. State-of-the-art approaches for such document selection (e.g., QXtract [2], PRDualRank [14], and FactCrawl [7]) are based on the observation that useful documents for a specific relation[2] tend to share distinctive words and phrases. Discovering these words and phrases is challenging because: (i) many extraction systems rely on off-the-shelf, black-box components (e.g., named entity recognizers), from which we cannot extract relevant words and phrases directly; and (ii) machine learning techniques for information extraction do not generally produce easily interpretable models, which complicates the identification of relevant words and phrases. QXtract learns these words and phrases through document classification: after retrieving a small document sample, QXtract automatically labels each document as useful or not by running the extraction system of interest over these documents. QXtract can thus learn that words like "richter" or "hypocenter" are characteristic of some of the useful documents for *Occurs-in*. Then, QXtract uses these learned words and phrases as keyword queries to retrieve (other) potentially useful documents (see Figure 1). More recent approaches (e.g., FactCrawl [7] and PRDualRank [14]) adopt similar retrieval-based document selection strategies.

QXtract issues queries to the standard keyword search interface of document collections in order to retrieve potentially useful documents for extraction. Such keyword search interface, unfortunately, is not tailored for information extraction: the documents that are returned for a keyword query are ranked according to how well they match the query and not on how useful they are for the underlying information extraction task [7]. For example, the query [tornado] for the *Occurs-in* relation returns only 145 useful documents among the top-300 matches from our validation split of the New York Times annotated corpus[3] (see Section 4) using Lucene[4], a state-of-the-art search engine library.

FactCrawl [7] moves a step beyond keyword search and reranks the retrieved documents to prioritize the extraction effort (see Figure 1). Specifically, FactCrawl scores documents proportionally to the number and quality of the queries that retrieve them. FactCrawl determines the quality of each learned query—and of the query generation method that

[2]Our approach is not applicable over open information extraction scenarios (e.g., [4]) where most documents often contribute tuples to the open-ended extraction task.
[3]http://catalog.ldc.upenn.edu/LDC2008T19
[4]http://lucene.apache.org/

was used to generate the query—in an initial step, once and for all, by retrieving a small number of documents with the query and running them through the extraction system in question. With this initial step, FactCrawl derives: (i) for each query q, the F-measure $F_\beta(q)$, where β is a parameter that weights precision over recall; and (ii) for each query generation method m, the average $F_\beta^{avg}(m)$ of the F_β value of all queries generated with method m. During the extraction process, after retrieving documents with a set Q_d of queries learned via a query generation method m, FactCrawl re-ranks the documents according to a scoring function $S(d) = \sum_{q \in Q_d} F_\beta(q) \cdot F_\beta^{avg}(m)$. FactCrawl's document re-ranking process improves the efficiency of the extraction, since the documents more likely to be useful are processed earlier. However, FactCrawl exhibits two key weaknesses: (i) for document retrieval and ranking, just as QXtract (see discussion above), FactCrawl relies on queries derived, once and for all, from a small initial document sample, and hence may miss words and phrases relevant to the information extraction task at hand; and (ii) for document ranking, FactCrawl relies on a coarse, query-based document scoring approach that is not adaptive, and hence does not benefit from the wealth of information that is captured as the extraction process progresses.

Adaptive models have been used for information extraction in a variety of ways. Early influential systems for large-scale information extraction, such as DIPRE [10] and Snowball [1], have relied on bootstrapping to adapt to newly discovered information. Starting with a small number of "seed" tuples for the extraction task of interest, these systems learn and iteratively improve extraction patterns and, simultaneously, build queries from the tuples that they discover using these patterns. However, these systems are not suitable for our problem for two main reasons. First, techniques based on bootstrapping often exhibit far-from-perfect recall, since it is difficult to reach all tuples in a collection by using previously extracted tuples as queries [2, 19]. Second, extraction systems are many times "black box" systems, which impedes the alteration of their extraction decisions. Other approaches (e.g., [12]) have relied on label propagation: starting with labeled and unlabeled examples, these approaches propagate the given labels to the unlabeled examples based on some example similarity computation. Such label propagation approaches are not beneficial for our extraction scenario, where the extraction system has already been trained and we can obtain new labels (i.e., useful or not) for previously unseen documents automatically by running the extraction system over them.

3. ONLINE ADAPTIVE RANKING

We now propose an end-to-end document ranking approach for scalable information extraction (see Figure 2) that addresses the limitations of the state of the art. Our approach prioritizes documents for an information extraction task—with a corresponding already-trained information extraction system—based on principled, efficient learning-to-rank approaches that exploit the full contents of the documents (Section 3.1). Additionally, our approach revises the ranking decisions periodically as the extraction process progresses and reveals (fine-grained) characteristics of the useful documents for the extraction task at hand (Section 3.2). Our approach thus manages to capture, progressively and in an adaptive manner, the heterogeneity of language and

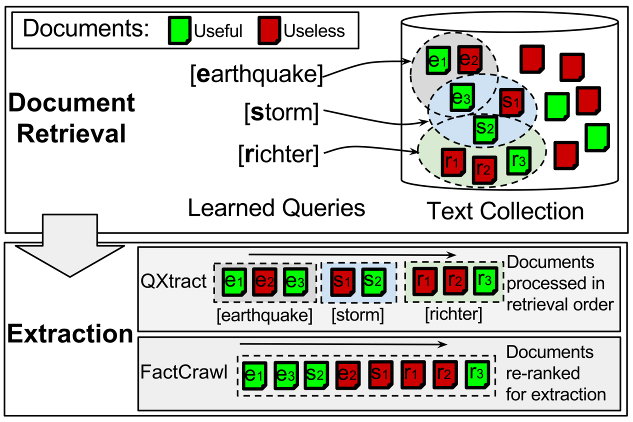


Figure 1: QXtract and FactCrawl.

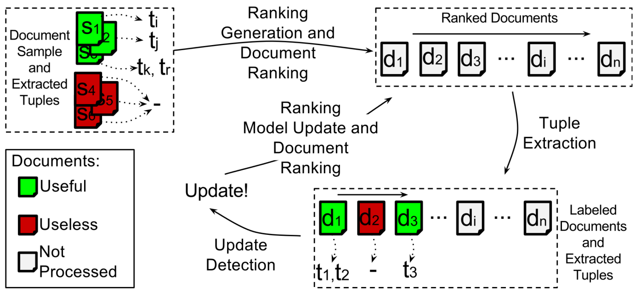
content typically exhibited by the useful documents, which leads to extraction processes substantially more efficient—and effective—than those with state-of-the-art approaches, as we will show experimentally in Sections 4 and 5.

3.1 Ranking Generation

To prioritize the information extraction effort, by focusing on the potentially useful documents for the extraction system at hand, we follow a learning-to-rank approach (see Ranking Generation step in Figure 2). Similarly to state-of-the-art query-generation and ranking efforts (see Section 2), we obtain a small document sample and automatically "label" it with the information extraction system, without human intervention. We use the documents in this sample, with their words as well as the attribute values of tuples extracted from them as features, to train an initial document ranking model. After the initial document ranking is produced, we start processing documents, in order, with the information extraction system (see Tuple Extraction step in Figure 2).[5] Unfortunately, the initial ranking model is generally far from perfect, because it is learned from a necessarily small document sample. So our approach periodically updates and refines the ranking model (see Update Detection step in Figure 2), as new documents are processed and the characteristics of the useful documents are revealed, as we will discuss in detail in Section 3.2.

Unfortunately, state-of-the-art approaches for learning to rank [23] are problematic for our document ranking setting for two main reasons. First, such approaches tend to be computationally expensive [29], so updating and revising the ranking model continuously over time, as new documents are processed, would result in an unacceptably high overhead in the extraction process. Second, such approaches tend to require a relatively small feature space [3]. In contrast, in our ranking setting the feature space, including the document words and attributes of extracted tuples, is vast; furthermore, the feature space continues to grow as new documents are processed. Therefore, we need to develop unconventional learning-to-rank techniques for our ranking prob-

[5]The pool of documents to process is either the full document collection, for collections of moderate size over which we have full access, or, alternatively, the documents retrieved with queries learned from the document sample. In Sections 4 and 5, we discuss this issue further and experimentally study these two scenarios.



Document Sample and Extracted Tuples
S1
S2
S3
S4
S5
S6
ti
tj
tk, tr
-
Ranking Generation and Document Ranking
Ranked Documents
d1 d2 d3 ··· di ··· dn
Tuple Extraction
Ranking Model Update and Document Ranking
Update!
Update Detection
Labeled Documents and Extracted Tuples
t1,t2
-
t3
Documents:
Useful
Useless
Not Processed

Figure 2: Our adaptive learning-to-rank approach for information extraction.

lem, to address the above two limitations of state-of-the-art approaches in an effective and efficient manner and without compromising the quality of the ranking models that we produce.

To address the efficiency limitation of learning-to-rank approaches, and to update the document ranking model efficiently, we rely on *online learning* [8]. Using online learning, we can train the ranking model incrementally, one document at a time. Therefore, we can continuously adapt the ranking model as we process new documents, without having to retrain it from scratch. To adapt online learning to our problem, the main challenge is to define an update rule for the model—to be triggered when we observe new documents along the extraction process—that is simple enough to be efficient but, at the same time, sophisticated enough to produce high-quality models. From among the most robust online learning approaches [8], the updates based on Pegasos gradient steps [30] are particularly well suited for our approach because of their efficiency and accuracy. Specifically, Pegasos gradient steps provide update rules that guarantee that learning techniques based on Support Vector Machines (SVM), the basis for some of the best-performing learning-to-rank approaches, learn high-quality models efficiently.

To address the feature-set limitation of learning-to-rank approaches, and to handle large (and expanding) feature sets, we rely on *in-training feature selection* [17]. In a nutshell, with in-training feature selection the learning-to-rank algorithm can efficiently identify the most discriminative features, out of a large and possibly expanding feature set, during the training of the document ranking model and without an explicit feature selection step. To do so, we rely on a sparse representation of the vectors that represent the feature weights, to discard all features with zero value. Therefore, our objective is to penalize models that rely on a large number of features with non-zero weight. Interestingly, we can rely on *regularization* [6] to control the feature weight distribution in our learned models: regularization penalizes models that have undesirable properties such as having many features with non-zero weights, so we can use it for in-training feature selection and also to avoid overfitting. In our approach, we rely on a linear combination of two regularization methods, usually called elastic-net regularization [35], which integrates: (i) the ℓ_1-norm regularization [32], which tends to learn models where only a small subset of the features have non-zero weights; and (ii) the ℓ_2-norm regularization, which produces high-quality models by avoiding overfitting. This combination is necessary because the ℓ_1-norm regularization does not perform well when the number of documents is smaller than the feature space [35], which is the case during early phases of the extraction process.

We now propose two learning-to-rank strategies, BAgg-IE and RSVM-IE, that overcome the limitations of state-of-the-art learning-to-rank approaches by integrating online learning and in-training feature selection, as discussed above.

BAgg-IE: Our first strategy incorporates online learning and in-training feature selection into a binary classification scheme where documents are ranked according to their assigned label and prediction confidence. Since binary classifiers optimize the accuracy of label assignment instead of the instance order, they are not optimized for ranking tasks [18]. For this reason, BAgg-IE adopts a more robust approach that exploits multiple binary classifiers based on bootstrapping aggregation, or bagging [9]. With this approach, the label assignments and confidence predictions derive from the aggregation of the answers of a committee of classifiers, rather than from an individual classifier. The intuition behind BAgg-IE is that each classifier is able to evaluate distinct aspects of the documents, thus collectively mitigating the limitations of each individual classifier. We adapt SVM-based binary classifiers [20] to support online learning and in-training feature selection. For online learning, our algorithm is based on Pegasos, in which each text document is a training instance and, hence, we update the model one document at a time. For in-training feature selection, each classifier in BAgg-IE combines the SVM binary classification problem with the regularization components of the elastic-net regularization framework that we discussed earlier, thus yielding the following learning problem to solve:

$$\arg\min_{\mathbf{w},b} \lambda_{All}\left(\frac{\lambda_{L2}}{2}\|\mathbf{w}\|_2+(1-\lambda_{L2})\|\mathbf{w}\|_1\right)+\sum_{(\mathbf{d},y)\in\mathcal{S}}\ell(y\langle\mathbf{w},\mathbf{d}\rangle+b)$$

where b is the bias factor, ℓ is the hinge loss function, $\ell(t) = max(0, 1-t)$, and $\|\mathbf{w}\|_1$ and $\|\mathbf{w}\|_2$ are the ℓ_1 and ℓ_2-norms of the weight vector (i.e., the regularization components), respectively. Moreover, λ_{All} is the parameter that weights the regularization component over the loss function, and λ_{L2}, $0 \leq \lambda_{L2} \leq 1$, is the parameter that weights the ℓ_2-norm regularization over the ℓ_1-norm regularization.

The committee in BAgg-IE consists of three classifiers[6], trained over disjoint splits of the documents, which leads to different feature spaces for each, and with balanced labels (i.e., same number of useful and useless documents). Finally, to obtain the score of a text document we sum over the normalized scores of each classifier $s(\mathbf{d}) = \frac{1}{1+e^{-(\mathbf{w}^\top\mathbf{d}+b)}}$, which accounts for the differences in the feature weights of each classifier. In this equation, $\mathbf{w}$ and b are the weight vector and bias factor, respectively, of the classifier.

In summary, BAgg-IE addresses the ranking problem as an optimized classification problem. In contrast, our second technique, RSVM-IE, which we describe next, adopts a principled learning-to-rank approach natively.

RSVM-IE: Our second learning-to-rank strategy is based on RankSVM [21], a popular and effective pairwise learning-to-rank approach. Just as we did for BAgg-IE, we need to modify RankSVM's original optimization problem so that it incorporates in-training feature selection and, in turn, suits our ranking problem. In a nutshell, RankSVM scores the

[6]Additional classifiers would slightly improve performance at the expense of substantial overhead.

documents via a linear combination of the document features: the score of a document d is $s(d) = \sum_i w_i \cdot d_i$, where w_i is the weight of feature i and d_i is the value of feature i in document d. The objective of RankSVM is then to find the set of weights $\mathbf{w} = \{w_1, ..., w_n\}$ that is optimized to determine, in a pair of documents, if a document is more relevant than the other document. To achieve this, RankSVM learns the feature weights by comparing the features of useful and useless documents in pairs: each pair includes a useful and a useless document, and the label indicates whether the useful document is the first document in the pair.

By integrating the in-training feature selection discussed above into the original RankSVM formulation, we obtain the following optimization problem to solve for RSVM-IE:

$$\arg\min_{\mathbf{w}} \lambda_{All}(\frac{\lambda_{L2}}{2}\|\mathbf{w}\|_2 + (1-\lambda_{L2})\|\mathbf{w}\|_1) + \sum_{(i,j)\epsilon\mathcal{P}} \ell(\mathbf{w}^\top(\mathbf{d}_i - \mathbf{d}_j))$$

where all variables are defined as for BAgg-IE, and $\mathbf{d}_i$ and $\mathbf{d}_j$ represent a useful and a useless document, respectively. For online learning, and in contrast to BAgg-IE, which uses the individual documents in the Pegasos scheme, the training examples are the pairs of useful and useless documents that the extraction process observes, which is known as *Stochastic Pairwise Descent* [29].

Unlike BAgg-IE, RSVM-IE is designed from the ground up to address a ranking task, so we expect it to outperform BAgg-IE. Moreover, we expect the overhead of RSVM-IE to be substantially lower than that of BAgg-IE, since BAgg-IE maintains multiple learned models (i.e., the classifiers in the committee). This overhead becomes noticeable when the models are frequently updated. Next, we explain our approach to decide when an update of the ranking models is desirable during the extraction process, thus reducing the overall document re-ranking overhead.

3.2 Update Detection

As we mentioned in Section 3.1, our adaptive extraction approach revises the ranking decisions periodically, to account for the new observations gathered along the extraction process. To determine when to update the ranking model (and, correspondingly, the document ranking), we introduce the *Update Detection* step (see Figure 2). To make this decision, we analyze whether the features of recently processed documents differ substantially from those in the ranking model. If this is the case, then we trigger a new ranking generation step (Section 3.1), which uses the recently processed documents as additional training examples. The new training examples often reveal novel features, or lead to adjusting the weight of known features, which in turn helps to more effectively prioritize the yet-unprocessed documents.

One possible approach for update detection is through feature shifting detection techniques [16]. Feature shifting predicts whether the distribution of features in a (test) dataset differs from the distribution of the features in the training data. Unfortunately, most feature shifting techniques are problematic: First, they rely on computationally expensive algorithms (e.g., kernel-based one-class SVM classifier [16]), thus incurring substantial overhead when applied repeatedly. Second, these techniques only detect changes in existing features, so they do not handle well the evolving feature space in our problem. Thus, the features that do not appear in the ranking would not be considered in the comparison, unless we re-train the kernel-based classifier from scratch, which would be prohibitively expensive.

As efficient alternatives, we introduce two update detection approaches, namely, Top-K and Mod-C. Top-K evaluates a reduced set of highly relevant features, determined independently from the ranking model, whereas Mod-C directly manipulates the low-level characteristics of the ranking model to detect changes in the feature space.

Top-K: Our first approach exploits the fact that the predicted usefulness of the documents in the current ranking varies the most when the highly influential features in the ranking model change. For instance, if the word "lava" becomes more frequent along the processed useful documents in our *Occurs-in* example, this feature will become (temporarily) more relevant than others. In that case, the predicted usefulness of documents that include such word should increase accordingly to be prioritized over other documents. Based on this observation, Top-K compares the K most influential features in the current ranking against the K most influential features according to the recently processed documents, and triggers an update when the difference between these two sets exceeds a given threshold τ, determined experimentally, as we explain in Section 4. Overall, Top-K consists of two key steps: (i) *feature selection*, which selects the K most influential features; and (ii) *feature comparison*, which measures the distance between two sets of features. To perform feature selection, we choose the K features with highest weight in an SVM-based linear classifier trained—and subsequently updated—on the same features (i.e., words and tuple attributes) as the ranking algorithm. To perform feature comparison, we compute a generalized version of the Spearman's Footrule[7] [22], which considers the relative position of the features and their weights. According to this measure, the difference between feature weights will be higher when heavily weighted features change positions.

As discussed, Top-K maintains its own set of relevant features according to an SVM-based binary classifier. The advantage of this approach is that it makes Top-K independent of the ranking technique. However, the relevant features in this classifier may differ from those in the ranking model [18]. In our *Occurs-in* example, for instance, a trained RankSVM model weighted the word "northern" as a top-20 feature, whereas a linear SVM model trained on the same documents weighted "northern" almost neutrally. Such discrepancies in the feature relevance may cause updates that have little impact on the document ranking or, alternatively, may lead to missing necessary updates because important features are not being evaluated. We now introduce Mod-C, which works directly with the ranking models, to capture feature relevance directly.

Mod-C: The techniques in Section 3.1 learn ranking models that consist of a vector of numeric weights, where each weight represents the captured relevance of one feature. We can then use a vector similarity metric, such as cosine similarity [24], to measure the difference between the relevance of features in two similar ranking models. Our second technique, Mod-C, exploits this observation and compares the current ranking model to an "updated" ranking model that also includes some of the recently processed documents. This

[7]The generalized version of the Spearman's Footrule that we use is given by $\sum_i w_i \cdot \left|\sum_{j:j\leq i} w_j - \sum_{j:\sigma(j)\leq\sigma(i)} w_j\right|$, where $\sigma(i)$ is the rank of feature i and w_i is its weight.

updated ranking model includes only a fraction ρ of the recently processed documents, since including all of these documents would incur substantial overhead. To compare the ranking models, Mod-C depends on a metric suitable for the ranking model (e.g., cosine similarity for RSVM-IE) and a threshold α, determined experimentally as we explain in Section 4, that needs to be exceeded to trigger an update. In our cosine similarity example, α would indicate the maximum allowed angle between ranking models, hence triggering an update when this angle is exceeded. Mod-C is thus able to handle the real relevance of features, crucial to precisely decide when an update in the ranking model will improve the current document ranking.

In summary, we propose two update detection techniques that decide efficiently when it is beneficial to revise the ranking decisions to adaptively improve the extraction process.

4. EXPERIMENTAL SETTINGS

We now describe the experimental settings for the evaluation of our adaptive ranking approach:

Datasets: We used the *NYT Annotated Corpus* [28], with 1.8 million New York Times articles from 1987 to 2007. We split this corpus into a training set (97,258 documents), a development set (671,457 documents), and a test set (1,086,944 documents). We evaluated different combinations of techniques and parameters on the development set. We ran the final experiments on the test set. Additionally, we used collections 1-5 from the TREC conference[8] to generate the queries for the query-based sample generation that we explain later in this section.

Document Access: As mentioned in Section 3.1, we consider two document-access scenarios: In the *full-access* scenario, we rank all documents in a (moderately sized) document collection. In contrast, in the (more realistic) *search interface access* scenario, we retrieve the documents to rank through keyword queries. We evaluate our ranking approach over both scenarios. For the search interface access scenario we learn the queries following QXtract (Section 2) to retrieve an initial pool of documents. Also, we provide a search interface over our collection using the Lucene indexer, to retrieve additional documents as the extraction process progresses: after each ranking update, we use the top-100 features of the updated ranking model as individual text queries to retrieve additional (potentially) useful documents.

Relations: Table 1 shows the broad range of relations from different domains that we extract for our experiments, with the number of useful documents for each relation in the test set. Our relations include *sparse* relations, for which a relatively small fraction of documents (i.e., less than 2% of the documents) are useful, as well as *dense* relations.

Information Extraction Techniques: We selected the extraction approach for each relation to include a variety of extraction approaches (e.g., both machine learning and rule-based approaches, as well as techniques with varying speed). Specifically, we considered different entity and relation extractors for each relation, and selected the best performing combination. However, for diversity, whenever we had ties in performance, we selected the (arguably) less common contender (e.g., a pattern-based approach to extract organizations and Maximum Entropy Markov Model [25], or MEMM, for natural disasters):

[8]http://trec.nist.gov/data.html

Relation	**Useful Documents**
Person–Organization Affiliation (PO)	185,237 (16.95%)
Disease–Outbreak (DO)	847 (0.08%)
Person–Career (PC)	458,294 (42.16%)
Natural Disaster–Location (ND)	18,370 (1.69%)
Man Made Disaster–Location (MD)	15,837 (1.46%)
Person–Charge (PH)	19,237 (1.77%)
Election–Winner (EW)	5,384 (0.50%)

Table 1: Relations for our experiments.

- For the Person–Organization Affiliation relation we used Hidden Markov Models [13] and automatically generated patterns [34] as named entity recognizers for Person and Organization, respectively. We used SVM [15] to extract the relation.
- For the Disease–Outbreak relation we used dictionaries and manually crafted regular expressions as named entity recognizers for Disease and Temporal Expression, respectively. We used the distance between entities to predict if they are related.
- For the remaining relations, we used Stanford NER[9] to find Person and Location entities, a MEMM [25] to find Natural Disasters, and Conditional Random Fields [26] to find the remaining entities. Then, we used the Subsequence Kernel [11] to identify relations between these entities.

Development Toolkits: We used the following off-the-shelf libraries: (i) Lingpipe[10], for rule-based named entity extraction; (ii) OpenNLP[11], for word and sentence segmentation; (iii) E-txt2db[12] and Stanford NER, to train and execute named entity extractors based on machine learning; and (iv) REEL[13] [5], to train relation extraction models.

Sampling Strategies: We compared two techniques to collect the initial document sample for our ranking techniques (Section 3.1):

- *Simple Random Sampling (SRS):* SRS picks 2,000 documents at random from the collection (only for the *full-access* scenario).
- *Cyclic Query Sampling (CQS):* CQS iterates repeatedly over a list of queries and collects the unseen documents from the next K documents that each query retrieves until it collects 2,000 documents. We learned 5 lists of queries using sets of 10,000 random documents (5,000 useful and 5,000 useless) from the TREC collection by applying the SVM-based method in QXtract [2].

Ranking Generation Techniques: We evaluated our ranking generation techniques from Section 3.1. To obtain the best parameters for these techniques, we performed several experiments over our development set, varying λ_{All} and λ_{L2}. The parameter values that we determined experimentally are as follows: for *BAgg-IE*, $\lambda_{All} = 0.5$ and $\lambda_{L2} = 0.99$; while for *RSVM-IE*, $\lambda_{All} = 0.1$ and $\lambda_{L2} = 0.99$. Setting $\lambda_{L2} = 0.99$ results in an ℓ_1-norm weight of $1 - \lambda_{L2} = 0.01$. This weight in turn results in models with 10 times fewer features—which are hence 10 times faster—than models that only use the ℓ_2-norm. Higher ℓ_1-norm weights would lead to

[9]http://nlp.stanford.edu/software/CRF-NER.shtml
[10]http://alias-i.com/lingpipe/
[11]http://opennlp.apache.org/
[12]http://web.ist.utl.pt/ist155840/etxt2db/
[13]http://reel.cs.columbia.edu/

lower-quality ranking models, as discussed in Section 3.

We also evaluated the following (strong) baselines:

- *FactCrawl (FC):* FC corresponds to our implementation of FactCrawl [7], as described in Section 2.
- *Adaptive FactCrawl (A-FC):* We produced a new version of FC that re-ranks the documents. Specifically, to make FC more competitive with our adaptive ranking strategies, A-FC recomputes the quality of the queries, and re-ranks the documents with these new values after each document is processed. In addition, A-FC learns new queries and retrieves more documents before every re-ranking step.

(We evaluated other approaches, such as QXtract [2] and PRDualRank [14], but do not discuss them further because FactCrawl dominated the alternatives that we considered.)

Update Detection Techniques: We evaluated our update detection techniques from Section 3.2:

- *Top-K:* We set $K = 200$, which experimentally led to high coverage of the relevant features and small overhead in feature comparison. We set $\tau = \varepsilon \cdot K$, where ε indicates how much each feature can change without impacting the ranking. We experimented with several values of τ and finally picked $\tau = 0.5$ ($\varepsilon = 0.0025$).
- *Mod-C:* We evaluated several combinations of ρ and α: the best value for ρ is 0.1, while the best angle values for α are 5° and 30° for RSVM-IE and BAgg-IE, respectively.

We also compared against the following baselines:

- *Wind-F:* We implemented a naïve approach for update detection that updates the ranking model after processing a fixed number of documents. We experimented with several values and observed no substantial differences. We report our results for updating 50 times along the extraction process, which leads to updates after 13,429 and 21,739 documents for the validation and test sets, respectively.
- *Feat-S:* We implemented an efficient version of feature shifting [16] using an online one-class SVM based on Pegasos [30]. We used a Gaussian kernel with $\gamma = 0.01$ and $k = 6$, as suggested in [16]. Finally, we triggered an update when the geometrical difference $F = 1 - S$ exceeded a threshold $\tau = 0.55$. Since the features of the documents after each update tend to fluctuate, we only run Feat-S after processing 700 new documents or more.

Executions and Infrastructure: We ran all experiments over a cluster with 60 machines with a uniform configuration: Intel Core i5-3570 CPU @ 3.40 GHz processors, with 8 GB of RAM, and OS Debian GNU/Linux 7 (wheezy). We used multiple independent processes to test our approach with different configurations. We executed each experiment five times with different samples (i.e., five different random samples and five different sets of initial sample queries), to account for the effect of randomness in the results, and report the average of these executions.

Evaluation Metrics: We use the following metrics:

- *Average recall* is the recall of the extraction process (i.e., the fraction of useful documents in the collection that have been processed) at different points during the extraction (e.g., after processing x% of the documents) and averaged over all executions of the same configuration.
- *Average precision* is the mean of the precision values at every position of the ranking [33], averaged over all the executions of the same configuration.
- *Area Under the ROC (AUC)* is the area under the curve of the true positive rate as a function of the false positive rate, averaged over all the executions of the same configuration.
- *CPU time* measures the CPU time consumed for extracting and ranking the documents.

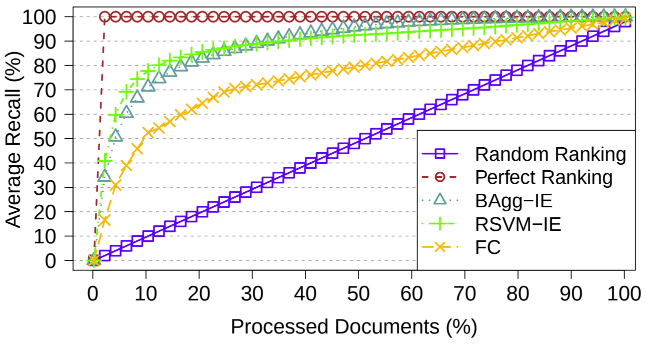


Figure 3: Average recall for Person–Charge for different base ranking generation techniques.

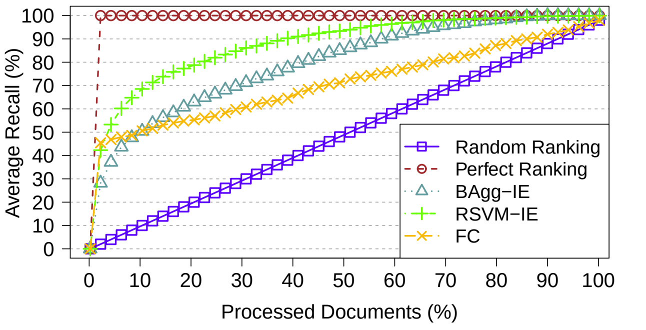


Figure 4: Average recall for Disease–Outbreak for different base ranking generation techniques.

5. EXPERIMENTAL RESULTS

We now present the results of the experimental evaluation of our adaptive ranking approach. We tuned the configuration of all components of our approach (i.e., the sampling strategy, the learning-to-rank approach, and the update detection approach) by exhaustively considering all possible combinations over the development set and selecting the best such combination. In the discussion below, for clarity, we consider the configuration choices for each component separately. Later, for the final evaluation of our approach over the test set and against the state-of-the-art ranking strategies, we use the best configuration according to the development set experiments.

Impact of Learning-To-Rank Approach: To understand the impact of using our learning-to-rank approach, we first evaluate our techniques of Section 3.1, *without the adaptation step*, against FC over the development set. Figure 3 shows the average recall for the Person–Charge relation for the full-access scenario. (For reference, we also show the performance of a random ordering of the documents, as well as of a perfect ordering where all useful documents are ahead of the useless ones.) Both RSVM-IE and BAgg-IE consistently outperform FC. Interestingly, RSVM-

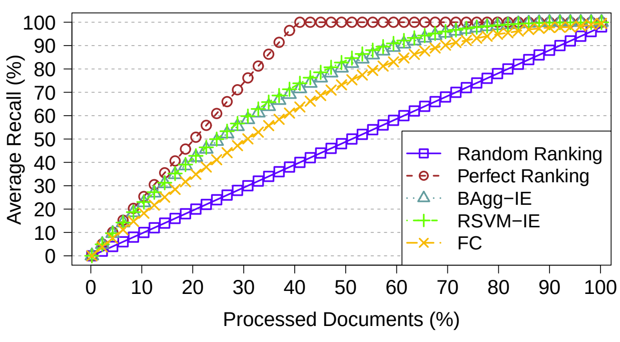


Figure 5: Average recall for Person–Career for different base ranking generation techniques.

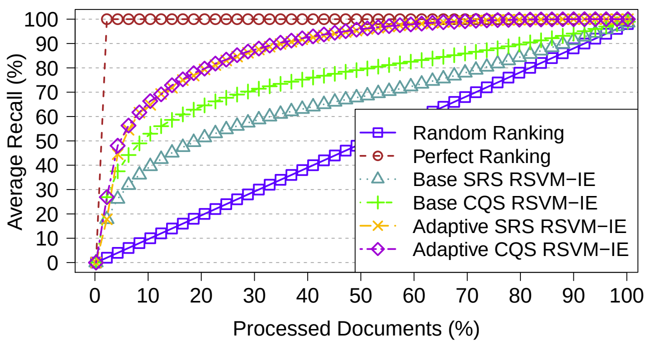


Figure 6: Average recall for Man Made Disaster–Location with different sampling techniques for the base and adaptive versions of RSVM-IE.

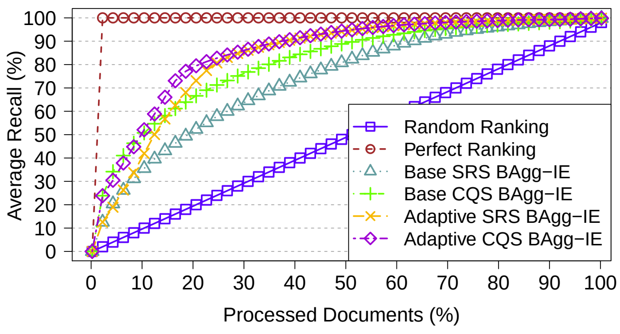


Figure 7: Average recall for Man Made Disaster–Location with different sampling techniques for the base and adaptive versions of BAgg-IE.

IE performs better in early phases of the extraction, while BAgg-IE performs better in the later phases, which agrees with our intuition from Section 3.1: RSVM-IE is at its core a ranking optimization technique, while BAgg-IE is based on classifiers. BAgg-IE separates useful from useless documents, thus obtaining high-accuracy in the middle of the extraction process, which in turn leads to high recall later on. We observed similar results for most of the relations (e.g., Figures 4 and 5 show the results for Disease–Outbreak and Person–Career respectively). However, RSVM-IE performs better than BAgg-IE for sparse relations, so RSVM-IE is preferable for such relations even in later phases of the extraction process (see Figure 4). Overall, even without an adaptation step, our techniques outperform the state-of-the-art ranking technique FC.

Impact of Sampling Strategies: To understand the impact of different sampling techniques to learn the initial ranking model, we compared RSVM-IE and BAgg-IE using the SRS and CQS sampling techniques (Section 4). Figure 6 shows the average recall for the Man Made Disaster–Location relation in the full-access scenario for RSVM-IE, both without the adaptation step (denoted with keyword "Base" in the plot) as well as with adaptation (denoted with keyword "Adaptive"). (The results for BAgg-IE were analogous; see Figure 7.) Using CQS, a sophisticated sampling technique, has a generally positive impact relative to using the (simpler) SRS strategy. The only exceptions were the dense relations, namely, Person–Organization and Person–Career, for which a simple random sample typically includes a wide variety of useful documents, thus leading to high-quality models.

Impact of Adaptation: We claimed throughout this paper that refining the document ranking along the extraction process significantly improves its efficiency. To support this claim, Figure 6 shows the average recall of RSVM-IE for the Man Made Disaster–Location relation for the full-access scenario. (The results for BAgg-IE are analogous, although the difference between the sampling techniques is higher than for RSVM-IE; see Figure 7.) These results show that by adapting the ranking model learned by RSVM-IE and, correspondingly, the document ranking, we significantly improve the efficiency of the extraction process. For example, Figure 6 shows that the adaptive versions of RSVM-IE can reach 70% of the useful documents after processing only 10% of the collection, whereas the base (non-adaptive) versions only reached 40% and 50% of the useful documents, for SRS and CQS, respectively. This same behavior was replicated by almost all relations. Additionally, as shown in Figure 6, the sampling technique does not have a significant impact anymore when we incorporate the adaptation step. Nevertheless, we observed that the results of average precision and AUC (see Table 2) are generally better for CQS than for SRS, since CQS leads to processing more useful documents at early stages of the extraction process.

Finally, we evaluated the number of new features incorporated into the ranking model during the adaptation step. In early stages of the extraction process, an average of 200 (or about 25% of the total number of features in the previous models) are incorporated; a similar number of features is removed in each adaptation step. However, in later stages, this behavior changes as the models become more stable. Specifically, the number of incorporated and removed features drops to 10 after each adaptation step. These results show that while the initial adaptation steps significantly impact the ranking model, the later ones are insignificant. Therefore, it is important to properly schedule the adaptation step to avoid insignificant updates to the ranking model.

Impact of Update Detection: To evaluate the update detection techniques that we introduced in Section 3.2, we fix the document sampling to SRS, and evaluate the tech-

	Base SRS		Base CQS		Adaptive SRS		Adaptive CQS	
Rel.	A. Precision	AUC	A. Precision	AUC	A. Precision	AUC	A. Precision	AUC
PO	33.6±0.9%	76.7±1.0%	37.9±1.0%	77.7±0.9%	**44.2±0.3%**	**82.7±0.1%**	43.6±0.3%	**82.7±0.1%**
DO	2.3±1.1%	88.2±2.2%	3.1±0.6%	87.9±0.9%	3.0±1.0%	97.0±0.1%	**3.8±0.6%**	**97.1±0.1%**
PC	80.2±0.4%	86.9±0.2%	79.2±0.5%	86.5±0.4%	**84.2±0.2%**	**89.9±0.1%**	84.1±0.2%	**89.9±0.1%**
ND	6.1±1.1%	64.0±3.5%	13.1±0.9%	64.3±3.2%	10.2±0.9%	**85.5±0.2%**	**16.4±0.8%**	85.4±0.2%
MD	7.3±1.8%	67.4±3.2%	13.6±1.2%	76.6±3.6%	12.9±1.4%	88.6±0.2%	**17.2±0.8%**	**89.2±0.1%**
PH	28.6±0.7%	89.7±1.4%	28.1±1.1%	87.3±1.6%	33.0±0.6%	**95.5±0.0%**	**33.4±0.6%**	95.4±0.0%
EW	6.6±4.0%	79.5±8.6%	10.2±0.8%	84.6±1.4%	9.4±3.2%	94.9±0.5%	**12.6±0.6%**	**95.3±0.1%**

Table 2: Comparison of the impact of different document sampling techniques on the ranking quality for all the relations with the base and adaptive versions of RSVM-IE for the full-access scenario.

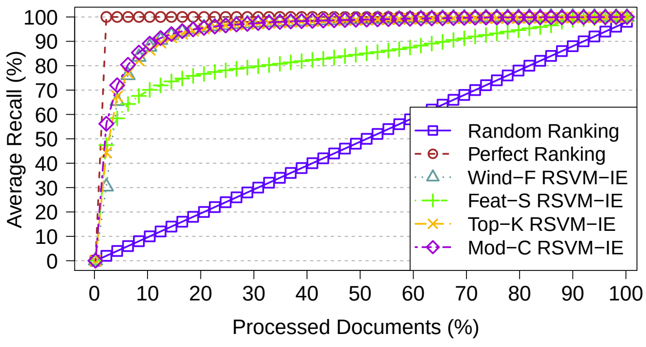


Figure 8: Average recall for Election–Winner for different update methods with RSVM-IE.

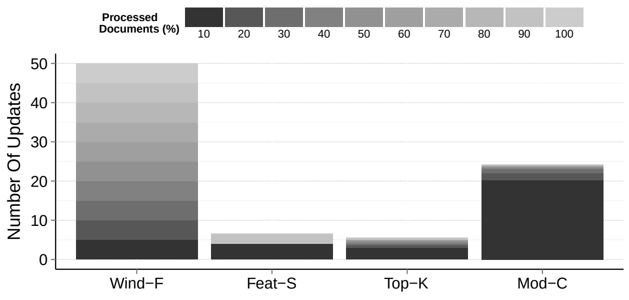


Figure 9: Distribution of updates for different techniques over the Election–Winner relation with RSVM-IE. (Darker shades represent earlier stages of the extraction process.)

niques according to their impact on the extraction process, distribution of updates, and overhead. Figure 8 shows the results of RSVM-IE for the Election–Winner relation for the full-access scenario. (The behavior for the other relations is analogous.) The Feat-S technique performed poorly in comparison to others, because Feat-S stops performing updates when the features observed in the data stabilize with respect to its kernel-based definition of shifting. For this reason, Feat-S misses late updates that prioritize other still poorly ranked useful documents. In addition, we observe that both Top-K and Mod-C produce consistently better results than Wind-F, especially at early stages of the extraction process, thus leading to high recall early in the extraction process. Overall, we show that both Top-K and Mod-C are robust alternatives for update detection in terms of ranking quality.

We also studied the distribution of updates across the extraction process, to understand the behavior of Top-K and Mod-C. Figure 9 shows the number of updates that each technique performs at different stages of the extraction process. Top-K and Mod-C tend to update much more frequently in early stages, where almost all documents carry new evidence of usefulness, than in later stages. For instance, most of the updates are performed while processing the first 10% of the collection. This behavior leads to ranking models that stabilize soon, since they are able to overcome the usual lack of training data in the initial document samples. Interestingly, despite the density of updates early in the process, the overall number of updates of Top-K and Mod-C remains smaller than that of Wind-F, since our techniques avoid unnecessary updates in late phases of the extraction process.

Additionally, we observed the percentage of features that are added to and eliminated from the models: the adaptation steps triggered by Top-K and Mod-C incorporate a consistent percentage of new features (i.e., about 10% per adaptation step) throughout the extraction process. This behavior significantly differs from that of Wind-F, which incorporates a large fraction of new features in early phases of the extraction process but only a small fraction of features later on: Top-K and Mod-C only perform an update when it will have a significantly positive impact on the model.

Update Technique	CPU Time per Document
Wind-F	0.01±0.00 ms
Feat-S	5.72±0.29 ms
Top-K	1.89±0.71 ms
Mod-C	0.32±0.10 ms

Table 3: Average CPU time to perform update detection after processing each document.

Finally, to evaluate the impact on efficiency of the update detection techniques, we calculated the overhead per document in terms of average CPU time, which we summarize in Table 3. As expected, Wind-F incurs negligible overhead (roughly 0.01 ms per document), since it only keeps a counter of the processed documents, whereas Feat-S incurs the highest overhead (5.72 ms per document). Our two techniques, Top-K (1.89 ms per document) and Mod-C (0.32 ms per document), exhibit a substantial difference in terms of efficiency, since the overhead of Top-K is dominated by the use of the binary classifier, as we discussed in Section 3.2. In conclusion, and considering also the quality results, Mod-C consistently outperforms the other techniques.

Scalability of our Approach: To understand how our

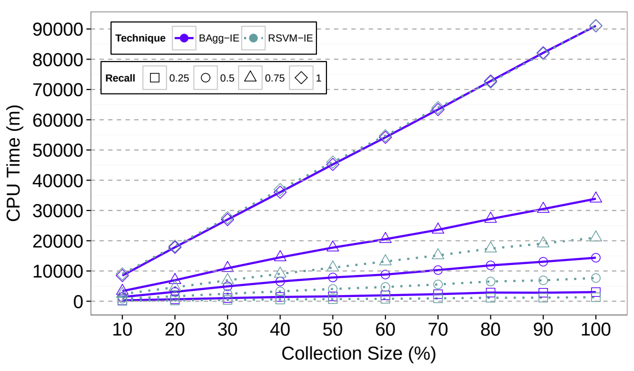


Figure 10: Average CPU time of our techniques as a function of the collection size for different target recall values, for the Natural Disaster–Location relation.

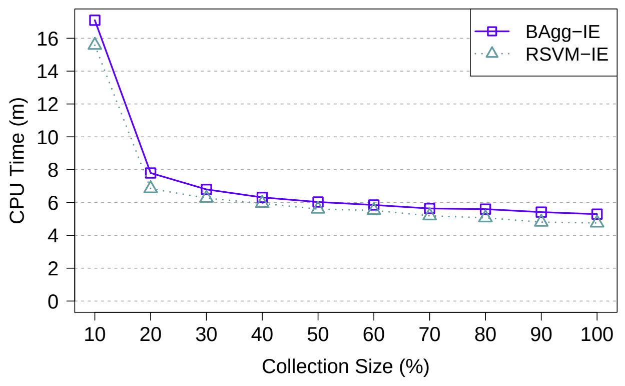


Figure 11: Average CPU time to find a target number of documents (i.e., the number of useful documents in the subset with 10% of the collection) for the Person–Organization Affiliation relation, as a function of the collection size.

strategies scale with the document collection size, we produced 10 subsets of the test collection with different sizes (from 10% to 100% of the total collection) and we measured (i) the time overhead for producing the ranking and (ii) the extraction time needed to reach a (fixed) target number of useful documents in each subset. Figure 10 shows how the size of the collection affects the CPU time needed to perform the ranking and extraction tasks with our techniques for the Natural Disaster–Location relation: the CPU time needed to perform an extraction task with our techniques grows approximately linearly with the collection size, which is desirable. Additionally, Figure 11 shows—for the Person–Organization Affiliation relation—that the time needed to find and process a target number of useful documents significantly drops as we increase the size of the collection. In this figure, the target number of useful documents corresponds to that in the subset of the collection that only contains 10% of the documents. As shown, the time becomes almost constant when the number of useful documents in the subset is large enough for the ranking to reach the target number at very early phases of the extraction process.

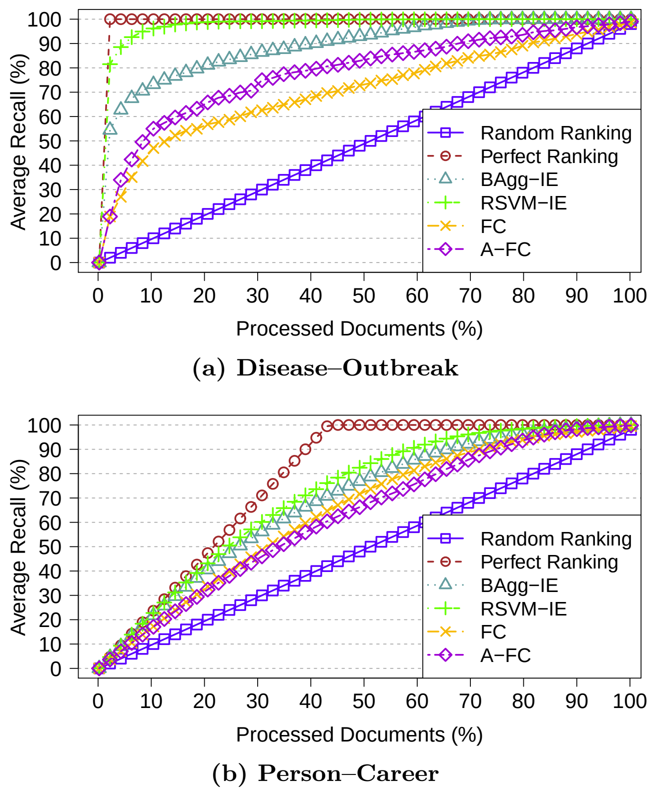


Figure 12: Average recall for different ranking approaches in the full-access scenario.

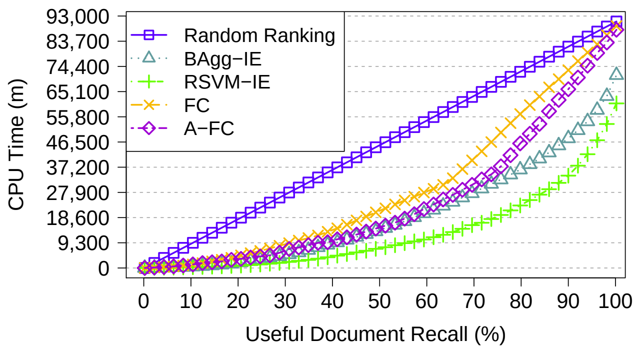
Comparison with State-of-the-Art Ranking Strategies: We now compare our best performing ranking approaches with the state-of-the-art approaches discussed in Section 4. We selected the best configuration for RSVM-IE and BAgg-IE according to the previous experiments, which involve CQS sampling and Mod-C update detection. Then, we ran this configuration over the test set to compare with FC and A-FC. We performed this experiment in the search interface access scenario as well, with similar conclusions. We compare the techniques on ranking quality and efficiency.

Table 4 shows the average precision and AUC of the four techniques that we compare, for all relations and over the full access scenario: RSVM-IE and BAgg-IE generally outperform the FactCrawl baselines by a large margin, and RSVM-IE consistently outperforms BAgg-IE. Interestingly, our adaptive version of FactCrawl, A-FC, does not exhibit the same significant improvement compared to FC that we observed between the adaptive and base versions of RSVM-IE and BAgg-IE above: A-FC is unable to properly model the usefulness of the documents when new features emerge, since it only relies on a small number of features.

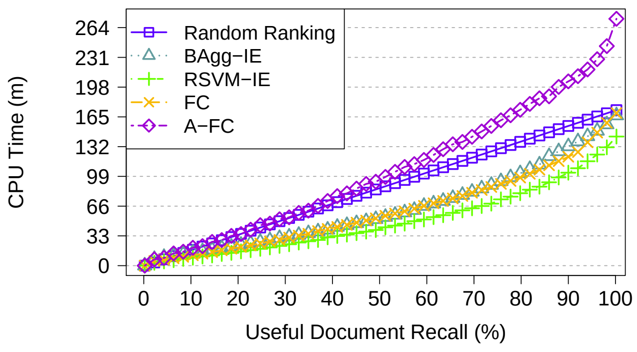
To understand the effects of the relation characteristics, we studied the performance of the techniques over both sparse (Figure 12a) and dense (Figure 12b) relations. The performance gap is more evident for sparse relations than it is for dense relations: The vocabulary around mentions of sparse relations tends to be reduced and specific, which makes it easier to model and prioritize the useful documents. Conversely, dense relations are scattered across diverse documents, thus co-occurring with a large variety of words, which makes it difficult to select a set of features that

	BAgg-IE		RSVM-IE		FC		A-FC	
Rel.	A. Precision	AUC	A. Precision	AUC	A. Precision	AUC	A. Precision	AUC
PO	40.5±0.9%	78.2±0.6%	**45.7±0.3%**	**82.4±0.1%**	29.0±0.9%	68.9±0.5%	30.5±0.6%	71.9±0.8%
DO	3.5±1.3%	89.7±0.3%	**8.3±0.2%**	**98.2±0.1%**	1.5±0.4%	71.5±11.4%	1.6±0.4%	78.8±5.4%
PC	79.2±0.4%	83.7±0.4%	**85.1±0.1%**	**88.6±0.1%**	66.3±1.1%	76.3±0.4%	63.2±1.0%	72.9±0.5%
ND	10.2±1.4%	78.4±0.5%	**18.9±0.6%**	**85.8±0.1%**	6.0±0.4%	67.8±1.5%	7.1±0.4%	72.9±0.2%
MD	10.8±2.1%	81.4±1.2%	**17.0±0.1%**	**88.0±0.0%**	3.8±0.4%	67.1±1.7%	4.1±0.4%	69.9±1.5%
PH	22.3±2.6%	90.5±2.1%	**33.8±0.3%**	**95.1±0.0%**	10.0±1.5%	74.6±2.8%	11.0±1.2%	78.9±1.5%
EW	9.6±0.6%	90.2±0.2%	**15.5±0.3%**	**95.4±0.1%**	2.4±0.2%	78.1±1.5%	2.6±0.2%	80.5±1.3%

Table 4: Comparison of the rankings generated by different techniques for the full-access scenario.



(a) Natural Disaster–Location



(b) Person–Organization Affiliation

Figure 13: CPU time to obtain a target recall value.

precisely identifies useful documents. Regardless, RSVM-IE and BAgg-IE still outperform the other techniques, since they are able to handle feature spaces of variable sizes.

We evaluate efficiency by measuring the time—including both ranking and extraction time—that each technique requires to achieve different values of recall. We show the results for two relations that exhibit substantially different extraction times according to their respective information extraction system: (i) Natural Disaster–Location, which takes an average of 6 seconds per document (Figure 13a); and (ii) Person–Organization Affiliation, which takes an average of 0.01 seconds per document (Figure 13b). RSVM-IE outperforms the others, in agreement with our earlier findings. The results for Person–Organization Affiliation are, in contrast, slightly different. For this fast extraction task, the overhead of the ranking technique can be problematic since it may easily become larger than the extraction time per se. We can observe such behavior for A-FC, which is less efficient than a random ranking technique with no overhead: A-FC (and, correspondingly, FC) relies on features that are expensive to compute [7], which is problematic for the adaptive case. However, the other techniques behave similarly as for the more expensive relations, with RSVM-IE resulting in the most efficient extraction process. Interestingly, for extraction tasks that incur lengthier extraction time, as is the case for Natural Disaster–Location, the quality of the ranking has a higher impact on efficiency than for other extraction tasks.

Overall, our experiments show that RSVM-IE outperforms all other techniques in all settings and extraction tasks. More specifically, RSVM-IE produces better rankings, while incurring very little overhead. Finally, when combined with Mod-C, RSVM-IE achieves much lower extraction times than the alternative strategies that we studied. Indeed, even with fast information extraction systems, adaptively ranking documents with RSVM-IE remained the best choice. Additionally, we evaluated the scalability of our techniques and confirmed that as the size of the collection grows, so does the positive impact of our approach, making it a substantial step towards scalable information extraction.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented an adaptive, lightweight document ranking approach for information extraction. Our approach enables effective and efficient information extraction over large document collections. Specifically, our approach relies on learning-to-rank techniques that learn in a principled way the fine-grained characteristics of the useful documents for an extraction task of interest. Our techniques incorporate (i) online learning algorithms, to enable a principled, efficient, and continuous incorporation of new relevant evidence as the extraction process progresses and reveals the real usefulness of documents; and (ii) in-training feature selection, to enable the learning of ranking models that rely on a small, discriminative set of features. Our experiments show that our approach exhibits higher recall and precision than state-of-the-art approaches, while keeping the overhead low. Overall, our document ranking approach is a substantial step towards scalable information extraction.

As future work, we plan to study how to estimate the recall of the alternative document ranking approaches for an information extraction task of interest. Through such estimates, we could in turn estimate the extraction cost, as a function of the number of processed documents, to achieve a target recall value with each ranking approach. We could then explore the recall-extraction cost tradeoff in a robust, quantitative manner, and substantially enhance recent optimization efforts for information extraction programs (e.g., [31]) by integrating our approach as an alternative document selection technique. As another direction for future work, we plan to continue studying our document ranking approaches along other dimensions, so that, for example, we can characterize ranking models according to the diversity of the tuples that they tend to produce. Finally, we aim at exploring parallelization approaches that, combined with the ranking-based approach described in this paper, can further speed up the execution of information extraction systems over large volumes of text data.

Acknowledgments: This material is based upon work supported by the National Science Foundation under Grant IIS-08-11038 and by *Fundação para a Ciência e a Tecnologia*, under Projects UID/CEC/50021/2013 and EXCL/EEI-ESS /0257/2012 (Datastorm), and Ph.D. Grant SFRH/BD /61393/2009. This work was also supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D11PC20153. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

7. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *ACM-DL*, 2000.

[2] E. Agichtein and L. Gravano. Querying text databases for efficient information extraction. In *ICDE*, 2003.

[3] B. Bai, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger. Learning to rank with (a lot of) word features. *Information Retrieval*, 13(3):291–314, 2010.

[4] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, 2007.

[5] P. Barrio, G. Simões, H. Galhardas, and L. Gravano. REEL: A relation extraction learning framework. In *JCDL*, 2014.

[6] C. M. Bishop. *Pattern recognition and machine learning*. Springer-Verlag, 2006.

[7] C. Boden, A. Löser, C. Nagel, and S. Pieper. FactCrawl: A fact retrieval framework for full-text indices. In *WebDB*, 2011.

[8] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*, 2010.

[9] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[10] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB*, 1998.

[11] R. C. Bunescu and R. J. Mooney. Subsequence kernels for relation extraction. In *NIPS*, 2005.

[12] J. Chen, D. Ji, C. L. Tan, and Z. Niu. Relation extraction using label propagation based semi-supervised learning. In *COLING/ACL*, 2006.

[13] A. Ekbal and S. Bandyopadhyay. A Hidden Markov Model based named entity recognition system: Bengali and Hindi as case studies. *Lecture Notes in Computer Science*, 4815:545–552, 2007.

[14] Y. Fang and K. C.-C. Chang. Searching patterns for relation extraction over the web: Rediscovering the pattern-relation duality. In *WSDM*, 2011.

[15] C. Giuliano, A. Lavelli, and L. Romano. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *EACL*, 2006.

[16] A. Glazer, M. Lindenbaum, and S. Markovitch. Feature shift detection. In *ICPR*, 2012.

[17] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[18] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*. MIT Press, 2000.

[19] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. To search or to crawl?: Towards a query optimizer for text-centric tasks. In *SIGMOD*, 2006.

[20] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML*, 1998.

[21] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, 2003.

[22] R. Kumar and S. Vassilvitskii. Generalized distances between rankings. In *WWW*, 2010.

[23] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3:225–331, 2009.

[24] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[25] A. McCallum, D. Freitag, and F. C. N. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *ICML*, 2000.

[26] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *CONLL*, 2003.

[27] C. J. V. Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, 1979.

[28] E. Sandhaus. The New York Times Annotated Corpus. In *Linguistic Data Consortium*, 2008.

[29] D. Sculley. Large scale learning to rank. In *NIPS Workshop on Advances in Ranking*, 2009.

[30] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *ICML*, 2007.

[31] G. Simões, H. Galhardas, and L. Gravano. When speed has a price: Fast information extraction using approximate algorithms. In *PVLDB*, 2013.

[32] Y. Tsuruoka, J. Tsujii, and S. Ananiadou. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In *ACL*, 2009.

[33] A. Turpin and F. Scholer. User performance versus precision measures for simple search tasks. In *SIGIR*, 2006.

[34] C. Whitelaw, A. Kehlenbeck, N. Petrovic, and L. Ungar. Web-scale named entity recognition. In *CIKM*, 2008.

[35] H. Zou and T. Hastie. Regularization and variable selection via the Elastic Net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.