

FactCrawl: A Fact Retrieval Framework for Full-Text Indices

Christoph Boden, Alexander Löser, Christoph Nagel, Stephan Pieper
University of Technology Berlin
Einsteinufer 17
10587 Berlin

[firstname.lastname]@tu-berlin.de

ABSTRACT

We present FactCrawl, a framework for retrieving structured, factual information leveraging the full-text index of a search engine. The framework applies an approximation algorithm to solve problem of retrieving all facts in a document collection using a minimal set of keywords while minimizing cost. The search engine is queried with automatically generated keywords, the results are re-ranked according to our fact score and documents are forwarded to a fact extractor. Keywords are determined using structural, syntactic, lexical and semantic information from sample documents. We estimate the fact score of a document by combining the observations of keywords in the document. We report results of an experimental evaluation over 20 fact extractors on a Reuters NIST corpus with 731,752 pages. Our experiments demonstrate that FactCrawl more than doubles recall in an online query scenario and nearly halves processing costs in an archive scenario, compared to existing approaches.

Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: *Text Analytics*

General Terms

Algorithms, Performance, Experimentation and Theory.

Keywords

Fact retrieval model, Knowledge harvesting

1. INTRODUCTION

Transforming unstructured data into structured information is an essential challenge for many Web users and business analysts. For instance, possible applications are market research, trend analysis and business intelligence in general. FactCrawl provides the first building block, the retrieval of relevant pages that are likely to contain a fact in an information extraction pipeline.

For example, users are often faced with the problem of searching the Web for lists or missing values of a spread sheet. Imagine a scenario where a company needs factual information about the criminal record of managers in the company. This information is highly important for the company, but might not be published in textual or structured form on ‘information aggregators’ in the Web, such as ‘Wikipedia.com’, or ‘Trueknowledge.com’. Rather, this rare factual information is hidden in unstructured Web text on a few Web pages of News Agencies or Blogs. Discovering these

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WebDB Workshop 2011, June 12-16, 2011, Athens, Greece.
Copyright 2010 ACM 1-58113-000-0/00/0010...\$10.00.

facts in pages in the Web is a tedious task, since Web search engines do not return aggregated factual information for such an informational query. The heuristic of a search engine user is: typing in keywords as queries, extracting relevant facts from the top-k documents. In our approach we aim to automate this process.

Our contributions: FactCrawl emulates a search engine user’s behavior to solve an inverse retrieval problem. We present a framework which implements the iterative fact retrieval process outlined in Figure 1. The framework allows plugging in different methods to sample documents, to observe potential keywords and to estimate the usefulness of keywords. In our framework we re-implemented QXtract [2], a prior approach to solve the fact retrieval problem, as well as five new feature generation methods that utilize lexical, syntactic, statistical and semantic models to generate keyword queries. We introduce an adapted version of relevance feedback [16] to re-rank the search results, so that the pages that are most likely to contain extractable facts are ranked highest. We present an experimental evaluation on a corpus of over 731,752 Reuters news articles which shows that FactCrawl clearly outperforms QXtract.

This paper is organized as follows: We outline related work in the fields of information extraction and keyword query generation (Section 2). After that we discuss the theoretical base and derive our fact retrieval model (Section 3). Next, we evaluate the performance of our algorithms (Section 4), summarize our work and propose future work (Section 5).

2. RELATED WORK

We discuss relevant related work in the areas of focused fact retrieval from full text indexes and Open Information Extraction.

Keyword query generation with QXtract: The authors of QXtract [2][19] pioneered work on generating keyword phrases for fact retrieval from full-text indices. The system executes the following stages in a one time learning process: sample seed documents, observe/score phrases in these documents, query a search engine with the most promising phrases and forward the

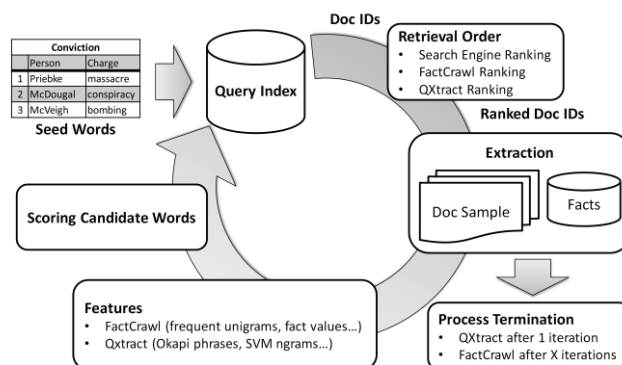


Figure 1. Iterative document ranking and extraction process

retrieved pages to an information extractor.

Sample seed documents. QXtract requires a small set of manually specified ‘seed’ facts to retrieve an initial set of sample pages for training. The size of this sample influences the precision of the learning process; the authors suggest between 100 and 2500 pages. After sampling, the pages are forwarded to the extractor to identify which documents are relevant for the extraction task.

Observe and score phrases. QXtract utilizes three classification approaches for observing and scoring keyword phrases from the documents in the seed sample: An SVM-based classifier, the OKAPI system [13] and the rule-based classifier Ripper [14] are trained on the set of seed documents. Each approach returns a list of terms, ordered by their significance for classifying relevant and irrelevant documents. The authors of QXtract propose a threshold based technique to assemble and select relevant phrases as keyword queries out of these terms.

Determine document retrieval order. Next, QXtract retrieves a ranked set of documents from the index for each generated keyword query, where the queries are selected from the three lists in a round robin fashion. All documents retrieved for a phrase are forwarded to the fact extractor. The process continues with selecting the next highest ranked phrases from each list and forwarding the resulting documents to the extraction service. It terminates once no more phrases exist in any list. The authors evaluated their approach on the two fact extractors “CompanyHeadquarter” and “DiseaseOutbreak” and achieved a recall of about 20% while processing 10% of the documents in the test corpus.

Phrase generation from SQL queries: Authors of [3] extract and rank potential keywords from attribute names and values of a structured query. They apply a greedy approach that selects terms based on their relevance measures *informativeness* and *representativeness*.

*Our approach makes use of a similar idea: We trust that Web page authors develop a set of common words and grammatical structures to express important factual information in natural language.

Self-supervised keyword generation: In [8] we published an approach that applies a self-supervised keyword generation method to extract meaningful phrases which often correlate with factual information in sentences. Our approach utilizes open information extraction techniques to generalize fact-type-independent lexico-syntactic patterns from sentences [4]. We reuse these patterns as one potential feature generation type in this work; for details see Section 3.3.

Fact extraction: Multiple projects, such as CIMPLE [15], AVATAR [20], DIAL [22], DoCQS [21], PurpleSox [23] describe an information extraction plan with an abstract, predicate-based rule language. Our approach tries to ‘re-engineer’ common patterns from observing fact extractors and other patterns that appear frequently with factual information on a page as features.

3. FactCrawl RETRIEVAL FRAMEWORK

The main goal of our framework is to be able to mine all the facts contained in a document collection while processing as few documents as possible. To solve this task we adapt keyword based document retrieval systems, such as Web search engines, to retrieve only those pages that contain a fact.

3.1 Problem Statement

Retrieve facts with function RFACT. Information retrieval systems such as Web search engines usually index document

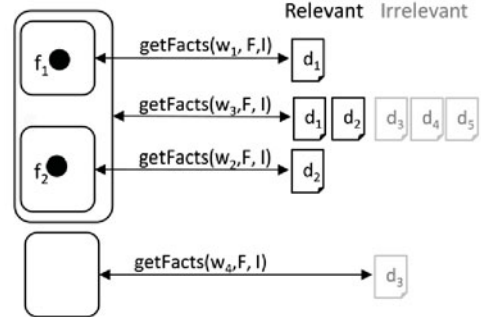


Figure 2. Individual keywords w_i are used to retrieve documents d_i . Relevant documents contain facts f_i that can be extracted by an information extractor. These facts are said to be reachable with a keyword w_i that led to the retrieval of the relevant document containing the fact.

collections with inverted indices. Words w out of a vocabulary W are mapped to documents d of a document collection D , where documents are usually treated as a bags of words w . A keyword query with a word w executed against an index I will thus return a set of documents which contain the keyword: $D_w = \{d : D \mid w \in d\}$. Unfortunately such an index does not allow querying the system with a keyword w to retrieve a set of facts $\{f\}$ contained in the documents. To overcome this missing functionality we define a new function $RFACT(W, F, I) = \{f_i\}$ which retrieves a set of facts f_i of fact type F . It takes as input a set of initial seed keywords W and accesses a specified index I over the document collection. See Figure 2 for an illustrative example.

Determine relevant words for RFACT. Given a specific keyword $w \in W$ the operation $r_{fact}(w, F, I)$ retrieves documents $d \in D_w$ which contain the keyword w . Next, the documents D_w are forwarded to an extractor which returns the facts contained in these documents. The cost for this operation is composed as follows:

$$COST(r_{fact}(w, F, I)) = \sum_{d \in D_w} Retrieve(d) + Extract(d, F)$$

Note that the cost of extraction clearly dominates the cost of retrieval for a document, as information extractors still need up to multiple seconds to process a document. Since the function $r_{fact}(w, F, I)$ will be executed once for each word in the set of seed keywords, the overall cost sums up to:

$$COST(RFACT(W, F, I)) = \sum_{w \in W} COST(r_{fact}(w, F, I))$$

Retrieve a minimal subset of relevant documents. Given the universe X of all facts of type F contained in a document collection D accessible via index I , we want to minimize the cost $COST(RFACT(W_{min}, F, I))$ of retrieving a full cover of the universe while using a minimal subset of documents that contain all the facts in a collection. To be able to solve this problem one needs to estimate the amount of extractable facts in a document and process the most promising documents first. QXtract essentially uses the search engine rank as an estimate and processes the documents in the order of their rank. However, the QXtract experiments show that this estimate is rather inaccurate and many irrelevant documents have to be processed by the information extractor. In our approach we attempt to generate a more accurate estimate of the relevance of documents and re-rank all retrieved documents before they are forwarded to the extractor.

```

RFACT( $W_{Seed}$ ,  $F$ ,  $I$ )
1.  $k \leftarrow 100$ ;  $n \leftarrow 10$ ;  $Q \leftarrow \emptyset$ ;  $X \leftarrow \emptyset$ ;
2.  $D_{Sample} \leftarrow \emptyset$ ;  $D_{Relevant} \leftarrow \emptyset$ ;
// Phase 1: Collect documents
3. foreach  $m \in \text{FeatureGenerationMethod}$ 
4.   foreach  $q \in m(D_{Sample}, D_{Relevant}, W_{Seed}, F, X, k)$ 
5.      $ID_{Doc} \leftarrow \text{SEARCH}(q, I)$ ;
6.      $D_{Crawled} \leftarrow \text{RETRIEVE}(\text{TOP}(ID_{Doc}, n))$ ;
7.      $D_{Sample} \leftarrow D_{Sample} \cup D_{Crawled}$ ;
8.      $Q \leftarrow Q + \langle q, m, ID_{Doc} \rangle$ ;
9.     foreach  $d \in D_{Crawled}$ 
10.       $X_{New} \leftarrow \text{EXTRACT}(d, F)$ ;
11.      if  $|X_{New}| > 0$ 
12.         $D_{Relevant} \leftarrow D_{Relevant} + d$ ;
13.       $X \leftarrow X \cup X_{New}$ ;
// Phase 2: Re-rank documents
14.  $ID_{Score} \leftarrow \emptyset$ ;
15. foreach  $\langle q, m, ID_{Doc} \rangle \in Q$ 
16.   foreach  $id \in ID_{Doc}$ 
17.      $score_{id} = \text{FACTSCORE}(id, q, m)$ ;
18.     if  $id \notin ID_{Score}$ 
19.        $D_{Scored} \leftarrow D_{Scored} + \langle ID_{Score}, score \rangle$ ;
20.     else
21.        $score_{id} \leftarrow score_{id} + \text{getScore}(ID_{Score}, id)$ ;
22.      $\text{UPDATE}(ID_{Score}, id, score_{id})$ ;
// Phase 3: Collect facts
23.  $D_{Ranked} \leftarrow \text{SORTDESC}(D_{Scored})$ ;
24. while (budget)
25.    $X \leftarrow X \cup \text{EXTRACT}(\text{RETRIEVE}(\text{NEXT}(D_{Ranked})), F)$ ;
26.    $\text{DECREASE}(\text{budget})$ ;
27. return  $X$ 

```

Figure 3. Pseudo code of the RFACT algorithm

The next sections provide a detailed description of our solution to solve this problem.

3.2 FactCrawl’s Fact Retrieval Framework

FactCrawl is a generic framework for evaluating different feature generation methods. We evaluate the usefulness of each feature and use this information to estimate the relevance of retrieved documents. Figure 3 lists the main algorithm of the framework in pseudo code which implements three phases:

Phase 1: Iteratively evaluate feature generation methods. We start with an empty set of sample documents D_{Sample} in line 2 and a small set of initial, potentially relevant, words W_{Seed} . For instance, these words could be derived from few user-defined tuples [2] or from schema information of the fact extractor [3]. In the first pass of the feature evaluation loop, we populate D_{Sample} by querying the search engine with the provided set of seed keywords W_{Seed} . These documents are forwarded to the information extractor to obtain any facts contained in the sample and to label each documents as either relevant or irrelevant. Our main goal is to identify discriminative keywords that can be used to identify documents that contain extractable facts of type F . To achieve this we evaluate different feature generation methods m to extract promising words or phrases of single words from the relevant documents of the sample. A detailed review of the different feature generation methods used by FactCrawl is given in Section 3.3.

For each subsequent feature generation method m we proceed as follows: we obtain the top k feature candidates q from the sample documents in conjunction with the facts X extracted from them (line 4). Next we query the index to receive all document IDs (ID_{Doc}) matching the query (line 5) and retrieve the top n ranked documents (line 6), which are then added to the sample documents (line 7). The important information which document

IDs where retrieved with query q of feature type m is stored in our buffer Q (line 8). We once again forward all n new pages to the extractor to obtain the contained facts (line 10). If facts could be extracted from a document, they are added to the fact buffer X and the document is marked as relevant (lines 11-13). This procedure is repeated for all surveyed feature categories m .

The goal of this phase is to obtain reasonable statistics on the effectiveness of feature candidates that can ultimately be used to score documents according to their usefulness for fact extraction. Furthermore, an extensive list of all potentially relevant document IDs (ID_{Doc}) is obtained in the process and stored in the buffer Q .

Phase 2: Re-Rank documents according to their relevance for fact extraction. As outlined in Section 3.1, we are trying to estimate the number of facts that can potentially be extracted from a document. We make this judgment based on prior evidence, namely we have a comprehensive graph that links already evaluated keyword queries q to the documents that have been retrieved with them and the facts in X that could be extracted from these documents. This setting is illustrated in Figure 2.

Scoring the effectiveness of a keyword query. In Phase 2 we iterate through each query q , estimate its effectiveness and finally score the documents based on their estimated usefulness (lines 16-22). The effectiveness of a query q is estimated with the F-Measure[11]:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

which is the weighted harmonic mean of precision P and recall R , a well-known measure in the context of information retrieval. It resembles the ability of a query to retrieve many facts (high recall) and only relevant pages that contain facts (high precision). We define precision as the fraction of relevant documents retrieved via query q :

$$P(q) = \frac{|D_q \cap D_{Relevant}|}{|D_{Sample}|}$$

We estimate the recall as the fractions of facts $f \in X$ which have been extracted via query q :

$$\hat{R}(q) = \frac{|X_q \cap X|}{|X|}$$

To obtain the final *EvidenceValue* of a query q we weight its F-measure with the average F-Measure over all queries from this feature generation method m :

$$EvidenceValue(q) = F_{\beta}^{avg}(m) \cdot F_{\beta}(q)$$

To obtain the final score of a document, we aggregate the *EvidenceValues* for each keyword query q that retrieved the document d :

$$FactScore(d) = \sum_{q \in Q_d} EvidenceValue(q)$$

Phase 3: Extract facts from re-ranked documents. After all the documents have been scored the entire list of documents is sorted (line 23). The pages are forwarded to the information extractor in the order of their relevance until the budget is exhausted or all pages have been processed (lines 24-26).

3.3 Evaluated Feature Generation Methods

Previous approaches, such as QXtract [2], execute only one sampling phase to bootstrap a set of seed documents. We extend this approach and adaptively expand the set of sample documents with six different methods. We start with the methods *user defined seed facts* and *fact-type name* to create an initial set of

sample documents. Both methods do not need sample documents as input and can thus bootstrap the sampling process. Next we evaluate four methods to extract potential keyword queries indicating a fact. The methods are executed in an order such that the methods requiring the most sample documents are executed last.

User defined seed facts. Similar to QXtract, this method utilizes handcrafted facts, such as {"SAP AG", "Walldorf"} or {"IBM Research", "Hawthorne"} as keyword queries.

Fact-type name. As proposed by the authors of [3], this method exploits the semantics of the fact-type name. We convert this name into a string representation, e.g. we tokenize the representation of the fact-type "CompanyProduct" into the conjunctive query "company AND product".

Attribute values of retrieved facts. We use the attribute values of facts extracted in phase one to identify further potential lexical expressions for the same fact-type, e.g. Company = {"Oracle", "Microsoft", ... }.

Frequent unigrams. Additionally we extract the top-k most frequent unigrams from relevant documents and use them as new search queries. These terms help us to retrieve all relevant documents, thus increasing recall while only modestly decreasing precision.

Significant phrases. We extract the most frequently collocated pairs of words from relevant documents. For observing these frequent collocations we use the methods of [12] and [24].

Lexico-syntactic fact-type classifier. Recent work in techniques for open information extraction [4] shows that most semantic relations (facts) between two entities are expressed utilizing only few syntactic patterns, such as $\langle E \rangle VRB \langle E \rangle$ or $\langle E \rangle VRB, PRP \langle E \rangle$. In our previous work we presented an efficient self-supervised lexico-syntactic classifier [8]. We identify sentences that contain a fact and obtain its part-of-speech (POS) tags. Next, we apply lexico-syntactic patterns from [4] to extract phrases representative for a certain fact-type. Finally, we rank these phrases according to their discriminativeness [5].

4. COMPARATIVE EVALUATION

In this Section we report our results on a comparative evaluation of prototype implementations for FactCrawl and QXtract.

4.1 Evaluation Framework

Reuters NIST corpus as data set. Our evaluation framework utilizes the Reuters News Corpus English (NIST). This standard evaluation corpus consists of 731,752 news documents from 1996 to 1997. Such a large corpus is impossible to label manually. Therefore we utilized the OpenCalais Extractor [1] to label

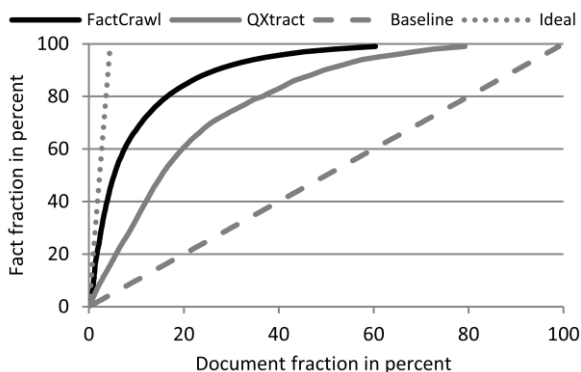


Figure 4. Average fact recall for 20 fact-types

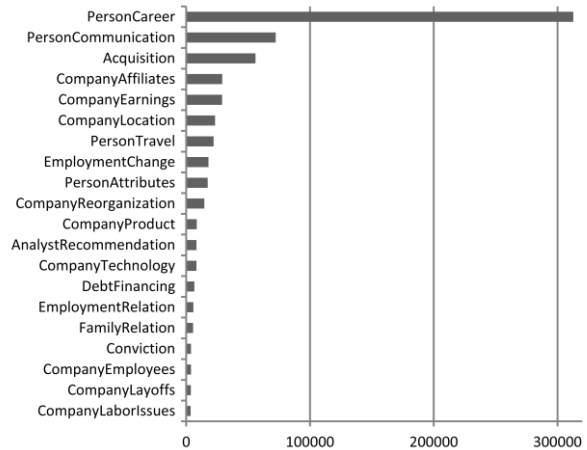


Figure 5. Long tailed fact distribution for 20 fact-types in the NIST evaluation corpus; the X-axis depicts relevant documents. The whole corpus consists of 731,752 documents.

entities and facts in the documents. Overall, this extractor extracted 866,684 entities and 1,798,875 facts out of 70 different fact-types. For our evaluation we selected 20 fact-types from the domains person, product and company, see Figure 5 for details. The attributes of these fact-types vary in their number and type. Most fact-types are rare and only appear in a small fraction of the documents. Therefore generating patterns to retrieve these pages is a highly relevant task.

Evaluation measurements. We define $FactRecall(n)$ as the fraction of all facts X_{all} of a specific fact-type that have been extracted after processing n documents.

$$FactRecall_{system}(n) = \frac{|X_{all} \cap X_{after\ n\ docs}|}{|X_{all}|}$$

Parameter setting for FactCrawl and QXtract. Following the work of [2] we utilize five seed tuples to sample 100 documents. We set the β value of the F-Measure to 0.5 to give a stronger weight to the precision of the top $k = 100$ keyword queries based on the top $n = 10$ documents. Automatically tuning these parameters is a subject in our future work.

Setup: We implemented FactCrawl and QXtract [19] in Java for the in-memory database HSQLDB [10]. For efficiently searching on the document collection we created a Lucene index [9] on the evaluation corpus. We conducted experiments on a Desktop PC (Windows 7) with 8 GB RAM and a Quad Core CPU with 3 GHz.

4.2 Evaluation Results

Different cost budgets may appear in application scenarios. To gain some deeper insights into the meaning of different recall levels, we identified two common application scenarios:

Online query scenario. In this scenario [26][17][25] it is crucial to reduce costs for retrieving and for extracting pages, while retaining high fact recall. Figure 4 shows the percentage of found facts, averaged over measurements for 20 fact-types. FactCrawl drastically reduces the cost for extracting in an online query scenario. FactCrawl only forwards 4% of the document collection which contain 45% of available facts, while QXtract processes 15% of the documents to reach the same recall. Likewise, after processing 10% of the document collection, FactCrawl already reaches a recall of 70%, while QXtract achieves only 32%.

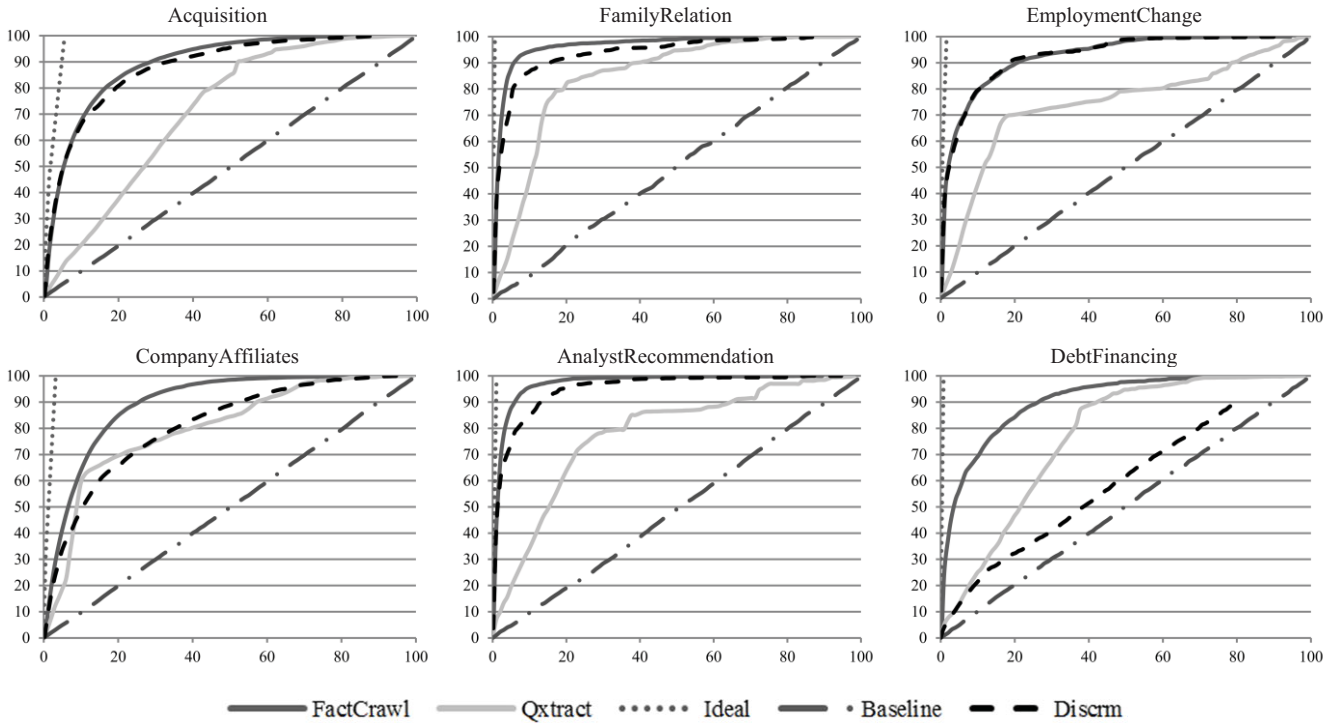


Figure 6. Evaluation results for six fact-types: The X-axes of the diagrams display the percentage of processed documents. Y-axes show the percentage of extracted facts. The system results lie between the ideal curve, which resembles perfect foresight where only relevant documents are forwarded to the extractor and the baseline where documents are forwarded in a random order. Discrm describes another competitive strategy that only uses generated keyword explained in Section 3.3.

Archive scenario. In this scenario [27][7][20][25] we consider a digital archive or a search engine operator that creates a knowledge base by extracting facts from a very large pool of documents, such as the Web. In this scenario a complete fact recall is crucial, while extraction cost should remain low. Both, FactCrawl and QXtract, discover 100% of available facts on our collection. Hence, both approaches observe features and generate queries against the full text index which effectively return all relevant documents. However, FactCrawl forwards only 37% of the entire document collection to the fact extractor, while QXtract forwards more than 60%.

4.3 Discussion

For each document FactCrawl’s retrieval model combines observed words and phrases into a single value. This fact score indicates how likely a document contains facts and is independent from the scoring model of the search engine. We re-rank the documents based on this score and process the most promising documents first. On the other hand, QXtract’s feature extraction methods do not merge information from collected features into a single score.

Effectiveness of FactCrawl’s score. Figure 4 shows the aggregated results averaged over all 20 fact-types we evaluated. We compare the recall curves of QXtract and FactCrawl with a Baseline scenario in which documents are selected in a random order and an ideal scenario where only relevant documents are processed. This figure illustrates that our approach of introducing new features and re-ranking the documents outperforms QXtract. Our recall curve shows a steeper increase than that of QXtract up to a recall level of about 70%.

Effectiveness of FactCrawl’s lexico-syntactic features. Figure 6 shows the results for six exemplary experiments for fact-types from the domains *person* and *company*. The curve labeled Discrm shows FactCrawl using only the lexico-syntactic classifier as a

feature generation method. The feature set of QXtract is limited to features from Ripper, SVM and OKAPI. This limitation prevents QXtract from extracting compound phrase structures as keywords, such as structures which combine verbs and noun phrases with pronouns, preposition and predicates in relevant sentences. Recent research [4] shows that it is often these combined features, which represent the most discriminative phrases in sentences that contain a fact. FactCrawl benefits from high precision feature extraction methods that can efficiently extract these compound phrases. For instance, our lexico-syntactic classifier [8] extracts compound phrases that represent predicates in English sentences, such as *sale of*, *workers in* or *to issue* (see also Figure 7). In addition, FactCrawl considers additional feature-types to reach missing pages for some rare fact-types, such as *CompanyAffiliates* and *DebtFinancing*. Our closer inspection of generated keywords revealed that noun phrases are particularly effective.

Effectiveness of FactCrawl’s iterative sampling strategy. Because of the additional feature types, FactCrawl observes a richer set of words per processed document. FactCrawl leverages

Fact-type	FactCrawl (Top 5)	QXtract(Top 5)
Acquisition	acquisition of, sale of, to sell, purchase of, to acquire	share, offer, rival, trade, deal
Analyst-Recommendation	coverage of, rating on, analyst, recommendation on, stock to	stock, bear stock, month stock, stock price, shares
CompanyAffiliates	employs, employees, workers in, people, jobs	company, percent strike, company united, minister company, industry united
DebtFinancing	exchange commission, shelf, subordinated, to issue, securities	million, percent primary, issue friday, million percent, million yield
EmploymentChange	named, executive, resigned, elected, become	broadcaster, executive, told, dollar, conference
FamilyRelation	friend, estranged, former, children, married	former, house family, chief time, divorced, love

Figure 7: Top-5 keywords for fact-types of Figure 6

this additional information in the sampling phase to generate new seed words and is able to sample more relevant documents for succeeding feature extracting iterations. Contrary, the training sample of QXtract is based on user defined seed words only. Depending on the ‘representative power’ of these words for the selected fact-type, the initial document sample may contain very few relevant documents. Consequently, QXtract cannot observe a sufficiently focused set of highly discriminative words with the same size of seed documents. Our results confirm this intuition: Compared to FactCrawl, costs for processing the entire document collection (archive scenario) are almost twice as high for QXtract.

5. CONCLUSIONS AND FUTURE WORK

Scaling fact extraction for large, indexed document collections is inherently difficult. Ideally, fact extractors leverage information from the index to process only relevant documents that are likely to contain a fact. We presented the FactCrawl framework which extracts words and phrases from few relevant documents with lexical, syntactic and statistical models in an iterative fashion. Next, the framework queries the index with observed words to retrieve further candidate documents. We re-rank the documents based on our fact score. This score combines information from observed features to determine the likelihood of a document containing facts. Finally, we forward the documents to the extractor in this order.

We present a comparative evaluation for 20 different fact-types over a standard NIST corpus. Our experiments show, that FactCrawl retrieves relevant documents more accurately than QXtract. For an online query and an archive scenario FactCrawl drastically reduces costs, compared to previous approaches [2][8].

Our framework is open for additional feature types and different fact scoring models. For instance, we will investigate the effectiveness of recent methods that consider the pattern relation duality [18]. Another orthogonal direction of work is the design of supervised classifiers [17] that filter out irrelevant pages. Potentially one could implement operators for cleansing, merging, verifying, aggregating or ranking [6][7] extracted facts to create concise, complete and relevant results.

Acknowledgements: The research leading to these results has received funding from the European Union’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n° ICT-2009-5-257859, ‘Risk and Opportunity management of huge-scale BUSiness community cooperation’ (ROBUST) and from grant agreement n° ICT-2009-4-1 270137 ‘Scalable Preservation Environments’ (SCAPE).

6. REFERENCES

- [1] OpenCalais. <http://opencalais.com> (Last visited 02/03/11).
- [2] Agichtein, E., Gravano, L. Querying text databases for efficient information extraction. ICDE 2003: 113–124.
- [3] Liu, J., Dong, X., Halevy, A.Y. Answering Structured Queries on Unstructured Data. WebDB 2006.
- [4] Etzioni, O., Banko, M., Soderland, S., Weld, D.S. 2008. Open information extraction from the Web. Commun. ACM 51(12): 68-74.
- [5] Fung, G., Yu, J., Lu, H. 2002. Discriminative Category Matching: Efficient Text Classification for Huge Document Collections. ICDM 2002: 187-194.
- [6] Kasneci et al. 2008. The YAGO-NAGA approach to knowledge discovery. SIGMOD 2008 Row 37(4): 41-47.
- [7] Jain, A., Doan, A., Gravano, L. 2008. Optimizing SQL Queries over Text Databases. ICDE. IEEE Computer Society, Washington, DC, 636-645.
- [8] Löser, A., Nagel, C., Pieper, S. Augmenting Tables by Self-Supervised Web Search. BIRTE Workshop at VLDB 2010.
- [9] Lucene. <http://lucene.apache.org/> (Last visited 02/23/11).
- [10] HSQLDB. <http://hsqldb.org/> (Last visited 02/23/11).
- [11] Van Rijsbergen, C. J. Information Retrieval. Butterworths. London. 1979.
- [12] LingPipe. <http://alias-i.com/lingpipe/> (Last visited 03/04/11).
- [13] Robertson, S. 1990. On term selection for query expansion. Journal of Documentation, volume 46, 1990.
- [14] Cohen, W. W. 1995. Fast effective rule induction. In International Conference on Machine Learning 1995.
- [15] Shen et al. 2008. Toward best-effort information extraction. SIGMOD 2008. ACM, New York, NY, 1031-1042.
- [16] Manning, C., Raghavan, P., Schütze, H. Introduction to Information Retrieval. Cambridge University Press, 2008.
- [17] Boden, C., Häfele, T., Löser, A. Classification Algorithms for Web Text Filtering. DaLi Workshop at ICDE 2011.
- [18] Fang, Y., Chang, K.C. Searching patterns for relation extraction over the web: rediscovering the pattern-relation duality. In Proceedings of WSDM. 2011, 825-834.
- [19] <http://code.google.com/p/qxtract/> (Last visited 02/21/2011).
- [20] Chiticariu et al. SystemT. An Algebraic Approach to Declarative Information Extraction. ACL 2010: 128-137.
- [21] Zhou M., Cheng T., Chang K. C. DoCQS: a prototype system for supporting data-oriented content query. SIGMOD Conference 2010: 1211-1214.
- [22] Feldman R., Regev Y., Gorodetsky M. A modular information extraction system. Intell. Data Anal. 12(1): 51-71 (2008).
- [23] Bohannon et al. Purple SOX Extraction Management System. SIGMOD Record, Volume 37, Issue 4, p.21-27 (2008)
- [24] Manning C.D. and Hinrich Schuetze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press. Chapter 5: Collocations.
- [25] Jain A., Srivastava D. Exploring a Few Good Tuples from Text Databases. ICDE 2009: 616-627.
- [26] Chu et al. A Relational Approach to Incrementally Extracting and Querying Structure in Unstructured Data. VLDB 2007: 1045-1056.
- [27] Ipeirotis P.G., Agichtein E., Jain P., Gravano L. To search or to crawl?: towards a query optimizer for text-centric tasks. SIGMOD Conference 2006: 265-27.