W

Web Information Extraction

Laura Chiticariu, Marina Danilevsky, Howard Ho, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

Information extraction; Text analytics

Definition

Information extraction (IE) is the process of automatically extracting structured pieces of information from unstructured or semi-structured text documents. Classical problems in information extraction include *named-entity recognition* (identifying mentions of persons, places, organizations, etc.) and *relationship extraction* (identifying mentions of relationships between such named entities). Web information extraction is the application of IE techniques to process the vast amounts of unstructured content on the Web. Due to the nature of the content on the Web, in addition to named-entity and relationship extraction, there is growing interest in more complex tasks such as extraction of reviews, opinions, and sentiments.

Historical Background

Historically, information extraction was studied by the Natural Language Processing community in the context of identifying organizations, locations, and person names in news articles and military reports [15]. From early on, information extraction systems were based on the *knowledge engineering approach* of developing carefully crafted sets of *rules* for each task. These systems view the text as an input sequence of symbols, and extraction rules are specified as regular expressions over the lexical features of these symbols. The formalism underlying these systems is based on cascading grammars and the theory of finite-state automata. One of the earliest languages for specifying such rules is the Common Pattern Specification Language (CPSL) developed in the context of the TIPSTER project [2]. To overcome some of the drawbacks of CPSL resulting from a sequential view of the input, the AfST system [4] uses a more powerful grammar that views its input as an object graph.

Beginning in the mid-1990s, as the unstructured content on the Web continued to grow, information extraction techniques were applied in building popular Web applications. One of the earliest such uses of information extraction was in the context of *screen scraping* for online comparison shopping and data integration applications. By manually examining a number of sample pages, application designers would develop ad hoc rules and regular expressions to eke out relevant pieces of information (e.g., the name

L. Liu, M.T. Özsu (eds.), *Encyclopedia of Database Systems*,
DOI 10.1007/978-1-4899-7993-3_459-2

of a book, its price, the ISBN number, etc.) from multiple Web sites to produce a consolidated Web page or query interface. Recently, more sophisticated IE techniques have been employed on the Web to improve search result quality, guide ad placement strategies, and assist in reputation management [13, 20].

Scientific Fundamentals

Knowledge-engineered rules have the advantage that they are easy to construct in many cases (e.g., rules to recognize prices, phone numbers, zip codes, etc.), easier to debug and maintain when written in a high-level rule language, and provide a natural way to incorporate domain or corpus-specific knowledge. However, such rules are extremely labor-intensive to develop and maintain. An alternative paradigm for producing extraction rules is the use of learning-based methods [12]. Such methods work well when training data is available in plenty and the extraction tasks are hard to encode manually. Finally, there has been work on the use of complex statistical models, such as Hidden Markov Models and Conditional Random Fields, where the rules of extraction are implicit within the parameters of the model [17].

For ease of exposition, the rule-based paradigm is used here to present the core concepts of Web information extraction. Rule-based extraction programs are called *annotators* and their output is referred to as *annotations*. The two central concepts of rule-based extraction are **rules** and **spans**. A span corresponds to a substring of the document text, represented as a pair of offsets *(begin,end)*. A rule is of the form $A \leftarrow P$ (Fig. 1), where A is an annotation (specified within angled brackets) and P is a pattern specification. Evaluating the rule associates any span of text that matches pattern P with the annotation A.

The description of information extraction is organized around four broad categories of extraction tasks: *entity extraction*, *binary relationship extraction*, *complex composite extraction*, and *application-driven extraction*. The first two categories, while relevant and increasingly used in Web applications, are classical IE tasks that have been extensively studied in the literature even before the advent of the Web.

Category 1. Entity extraction. This refers to the identification of mentions of named entities (such as persons, locations, organizations, phone numbers, etc.) in unstructured text. While the task of entity extraction is intuitive and easy to describe, the corresponding annotators are fairly complex and involve a large number of rules and carefully curated dictionaries. For example, a high-quality annotator for person names would involve several tens of rules to capture all of the different conventions, shorthand, and formats used in person names all over the world.

Example 1 As an illustrative example, consider the simple annotator shown in Fig. 1 for recognizing mentions of person names. The annotator uses a CPSL-style cascading grammar specification. Assume that the input text has already been tokenized and is available as a sequence of ⟨Token⟩ annotations. Rules R_4, R_5, and R_6 are the lowest level grammar rules since they operate only on the input ⟨Token⟩ annotations. Each of these rules attempts to match the text of a token against a particular regular expression and produces output annotations whenever the match succeeds. R_6 states that a span of text consisting

⟨Person⟩ ← ⟨Salutation⟩ ⟨CapsWord⟩ ⟨CapsWord⟩ (R_1)
⟨Person⟩ ← ⟨PersonDict⟩ ⟨PersonDict⟩ (R_2)
⟨Person⟩ ← ⟨CapsWord⟩ ⟨PersonDict⟩ (R_3)
⟨Salutation⟩ ← ⟨Token⟩[≃ "`Mr.|Mrs.|Dr.| ...`"] (R_4)
⟨CapsWord⟩ ← ⟨Token⟩[text() ≃ "`[A-Z][a-z]*`"] (R_5)
⟨PersonDict⟩ ← ⟨Token⟩[text() ≃ "`Michael|Richard|Smith|....`"] (R_6)

Web Information Extraction, Fig. 1 Simple rules for identifying person names

of a single token whose text matches a dictionary of person names ("Michael," "Richard," etc.) is a ⟨PersonDict⟩ annotation. R_4 similarly defines ⟨Salutation⟩ annotations ("Dr.," "Prof.," "Mr.," etc.) and R_5 defines annotations consisting of a single token beginning with a capital letter. Finally, R_1, R_2, and R_3 are the higher-level rules of the cascading grammar since they operate on the annotations produced by the lower-level rules. For example, R_1 states that a salutation followed by two capitalized words is a person name. This rule will recognize names such as "Dr. Albert Einstein" and "Prof. Michael Stonebraker."

Category 2. Binary relationship extraction. This refers to the task of associating pairs of named entities based on the identification of a relationship between the entities. For instance, Example 2 describes the task of extracting instances of the *CompanyCEO* relationship, i.e., finding pairs of person and organization names such that the person is the CEO of the organization.

Example 2 Assume that mentions of persons and organizations have already been annotated (as in Category 1 above) and are available as ⟨Person⟩ and ⟨Organization⟩ annotations respectively. Fig. 2 shows rules that identify instances of the *CompanyCEO* relationship. Rule R_7 looks for pairs of ⟨Person⟩ and ⟨Organization⟩ annotations such that the text between these annotations satisfies a particular regular expression. The regular expression that is used in this example will match any piece of text containing the phrase "CEO of" with an optional comma before the phrase and an arbitrary amount of whitespace separating the individual words of the phrase. Thus, it will correctly extract an instance of this relationship from the text "Ginni Rometty, CEO of IBM." As with Example 1, the rules presented here are merely illustrative, and high-quality relationship annotators will involve significantly more rules.

Category 3. Complex composite extraction. The presence of vast amounts of user generated content on the Web (in blogs, wikis, discussion forums, and microblogs) has engendered a new class of complex information extraction tasks. The goal of such tasks is the extraction of more complex concepts, such as reviews, opinions, and sentiments. Annotators for such complex tasks are characterized by two main features: (i) the use of entity and relationship annotators as submodules to be invoked as part of a higher level extraction workflow, and (ii) the use of aggregation-like operations in the extraction process.

The task of discovering and analyzing sentiment present in text is an example of a complex composite extraction. Extracting individual instances of sentiment mentions typically consists of five subtasks: (1) entity extraction, (2) buzz extraction, (3) sentiment extraction, (4) target detection, and (5) aspect extraction.

Example 3 Consider the task of extracting sentiment for a class of smartphone products. First, the *entity extraction* step identifies all the mentions of the smartphones. Next, *buzz extraction* identifies the class of smartphones of interest. *Sentiment extraction* then uses the output of the buzz extraction to identify positive or negative sentiment mentions, including detecting the *target* of the sentiment (the smartphone mention with which the sentiment is associated). For example, in the sentence, "I prefer iPhone 5s to Galaxy S5," a positive sentiment is associated with the target "iPhone 5s" and a negative sentiment is associated with "Galaxy S5." In addition, *aspect extraction* provides fine-grained analysis of product review feedback. For instance, "I like its large screen, but not its short battery life," describes two aspects of a phone, each with a different

⟨CompanyCEO⟩ ← ⟨Person⟩ RegExp(''\s+,?\s+CEO\s+of\s+'') ⟨Organization⟩ (R_7)
⟨CompanyCEO⟩ ← ⟨Organization⟩ RegExp(''\s+CEO\s+'') ⟨Person⟩ (R_8)

Web Information Extraction, Fig. 2 Simple rules for identifying CompanyCEO relationship instances

sentiment. Depending on the application domain, the individual sentiment mentions may need to be aggregated within each individual document, or across all documents to answer questions such as, "what is the overall sentiment of a particular reviewer towards the product?" and, "what is the overall sentiment of all the reviewers?"

Category 4. Application-driven extraction. The last category of extraction tasks covers a broad spectrum of scenarios where IE techniques are applied to perform extraction unique to the needs of a given Web application.

Example 4 As an example, consider information that can be extracted from the large amount of financial data publicly available on the web. Such data appears in various forms, including regulatory filings, blogs, news feeds, government reports and research articles. This mass of mostly unstructured documents contains many nuggets of high-value information related to the financial performance of organizations, their relationships, and key relevant events. An automatic process to extract relevant information from such a large number of documents is necessary for the performance of sophisticated analysis. Such extraction pipelines are often constructed hierarchically, consisting of several components where each component in turn is constructed from more narrowly specified subcomponents. In particular, consider analyzing a table within a PDF document, which are a common source of information on the Internet. The PDF file must first be converted to text before subsequent analysis is possible. To extract a table structure from a PDF document, an OCR engine is often required to provide position information for text blocks on the page. Rules can be written to use the position information to identify the table, table title, and row and column headers; to logically merge tables across multiple pages; and to extract the actual cell contents. The information thus extracted from tables can then be combined with information found in the free text to gain a sophisticated understanding of the present entities and relations.

Key Applications

As the vast majority of information on the Web is in unstructured form, there is growing interest, within the database, data mining, and information retrieval communities, in the use of information extraction for Web applications. Many research projects in the areas of *search*, *online communities*, and *Web analytics*, as well as a wide variety of business needs, already employ IE techniques to analyze unstructured data. The common theme in all of these applications is that the information required to solve the problem is hidden inside unstructured text, and necessitate the use of IE to process the input text and produce a structured representation.

Web Search Today, web search engines employ IE techniques to recognize key entities associated with a web page, such as persons, locations, or organizations. Search engines use this semantically richer understanding of the contents of a page to drive corresponding improvements to their search ranking and ad placement strategies. Entity information is likewise extracted from user queries to better understand the user's information need, enabling the search engine to improve the relevance of search results, and to even return a precise answer to questions such as, "What time is it in California?"

Social Media Sentiment Analysis With the ubiquity of social media in the lives of consumers, there is constant interest in techniques to reliably extract reviews, opinions, and sentiments about products and services from the content present in online communities. The extracted information can be used to guide business decisions related to product placement, targeting, and marketing. To follow *Example 3* from the previous section, smartphone companies would benefit from sentiment-related extraction techniques to discover how their products are perceived by customers.

Machine Data Analysis Information extraction techniques can help analyze machine-generated logs. System logs are generally a mix of struc-

tured information related to the system health and performance over time, and text comments recorded by the system or by human administrators, about particular events that occurred. This data may hold useful knowledge, such as potential relationships between the occurrence of suspicious events and degradation of system performance. In order to discover such relationships, IE techniques must first be used to extract key event information from the text comments, transforming them into structured temporal features.

Domain-Specific Analysis The application of IE techniques extends to a variety of other circumstances. Consider again *Example 4* that characterizes financial data, with interesting knowledge spread across many different types of documents, such as regulatory filings, news articles, and government reports; and hidden in titles, section headings, tables and free text. In order to, for instance, build a counterparty graph of the U.S. financial sector, IE approaches are necessary to identify all of the relevant entities – banks, directors, customers, loans, and securities – as well as how all those different entities are related to each other [7].

The complex nature of these extraction tasks and the heterogeneous and noisy nature of the data pose interesting challenges, and IE techniques will be invaluable components of many research and business applications.

Future Directions

While the area of information extraction has made considerable progress since its inception, several important challenges still remain open. A few of these challenges that are under active investigation in the research community are described below.

Expressivity

With the expansion of both the domain and applicability of information extraction, the expressivity demands of information extraction systems also increases. Expressivity refers to how versatile and sophisticated a system can handle various tasks. We list a few future directions for increasing expressivity.

Parser-based IE Pattern matching has been a workhorse for information extraction, but due to the variability of natural language expressions, it is tedious and error-prone to construct all the relevant patterns for a particular relationship. Shallow parsing, also known as part-of-speech tagging, may be used to help increase the reliability of annotators. Deep parsing constructs a parse tree from a sentence, with semantically meaningful relations among the nodes. Such parse trees may be used as a more robust and versatile alternative to regular expressions. As a simplified example, a pattern "Subject:⟨Organization⟩ Verb:{acquire, buy} Object:⟨Organization⟩" may be used in an "acquisition annotator", replacing a number of similar rules written with regular expressions. Since deep parsing is usually much slower than regular expression matching, performance optimization will be an important issue. For applications in different domains, the ability for users to customize the grammar used in parsing may also become important.

Handling documents in different formats Documents in formats other than plain text or markup languages present additional challenges. In particular, PDF processing remain an area of active research. Other types of data may contain information worth extracting (e.g., text data in graphic or image formats). Such IE tasks would benefit from a system capable of expressing the relevant features at a high level.

Noisy data Social media increasingly contains text that is much noisier than well-written newspaper articles. However, robust and effective information extraction from such data is still highly desirable. One approach for handling noisy data is to build a model of generating the noisy data via corruption of clean data. For example, in tweets, the phrase "I am going to see" is often written as "gonna see". A trained model may be able to "normalize" (reverse the corruption in) the noisy data [31].

Robustness against different writing styles Data from different sources, such as news articles, financial reports and tweets can be written in very different styles while conveying the same information. Generalizing the parser-based IE and the normalization method mentioned above, it is desirable to have systems that can construct annotators that are independent of such variations in style, while still able to extract the essential information.

Extensibility

The extensibility of an IE system can be measured along three axes: system, semantic, and multilingual.

System Extensibility Given the increasing demand for web-scale IE for various domains, it is important that a single IE system is flexible enough to both provide all required functionalities, and easily support new functionalities. As such, an IE system should be able to benefit from third-party libraries to expand any limitation of its native capabilities. To do so, the system needs to provide extension points (e.g., via user-defined functions) to allow easy integration of third-party libraries to provide new or enhanced functionalities (e.g., pdf conversion or text normalization) without compromising the systems resource management or scalability.

Semantic Extensibility The same annotator, when applied over different domains or data sources, may require domain or data-specific customizations. The exact definition of the same information extraction task may also differ from application to application. Taking sentiment analysis as an example, when performing sentiment analysis over Twitter messages, it is crucial to properly handle informal language pervasive on Twitter, such as slang ("*lol*", "*bff*") and tweet-specific syntaxes (e.g. hashtags, retweet, and reply). When performing the same task over financial research reports, it is necessary to consider domain-specific sentiment expressions ("*Sell EUR/CHF at market for a decline to 1.31*" expresses negative sentiment about the Euro but positive sentiment about the Swiss Franc). Even within the same domain, different applications may have varied requirements for the same extraction task. For instance, "I like Target's website better," would be considered to be positive sentiment by a brand management application for Target, but negative by applications used by Target's competitors.

Rather than requiring that a sentiment analysis solution be built from scratch for different use cases, an extensible IE system should support the development of the same core sentiment analysis package for a generic domain, which can be adapted and customized to handle different data, domain, and application-specific nuances with minimal additional effort. The viability of this approach has been demonstrated by [10], showing that it is possible to produce high quality extractors over different domains based on adaptations of the same core generic extractor.

Multilingual Support Given the wide variety of languages used on the web, multilingual support is crucial for any IE system aiming to support web-scale IE. Specifically, an IE system should provide native multilingual support (e.g., tokenization) for as many languages as possible and allow its users to build new annotators or extend existing ones for these languages. In addition, it should provide pre-built annotators to provide out-of-the-box extraction support for as many languages as possible. Finally, it should be able to take advantage of third-party libraries to extend its existing multilingual capability. For instance, providing a public API to enable the integration of third-party tokenizers and POS taggers enables users to develop extractors for languages that have yet to be natively supported by the system.

Scalability

As complex information tasks move to operating over Web-size datasets, scalability has become a major focus [14]. In particular, IE systems need to scale to both large numbers of input documents (dataset size), and many rules and patterns (annotator complexity). Two approaches that address scalability are improving the performance of low-level operations that dominate

execution time, and choosing efficient orders for evaluating annotator rules.

Low-Level Primitives In many IE systems, low-level text operations like tokenization, regular expression evaluation and dictionary matching dominate execution time. Speeding up these operations leads to direct improvements of both the number of documents the system can handle and the number of basic operations the system can afford to perform on a given document.

- When an annotator uses a large number of dictionaries, evaluating each one separately is expensive, as the document text is tokenized once per dictionary evaluation. To address this, techniques are being explored to efficiently evaluate multiple dictionaries with a single pass over the document text.
- Evaluating complex regular expressions over every document is an expensive operation. One active area of research is the design of faster regular expression matchers for special classes of regular expressions. A different technique under investigation is the design of "filter" regular expression indexes. These indexes can be used to quickly eliminate most of the documents that do not contain a match.
- IE rules may use approximate string matching to improve the recall of dictionary-based rules, or to allow for variations in spelling. These operations can be very CPU- and memory-intensive. Recent research projects have developed algorithms and data structures to support efficient approximate string matching [29].
- Even with indexing and the best software-based implementations of low-level operations, some applications still require additional throughput. Another active area of research explores offloading operations like regular expression evaluation onto a dedicated hardware accelerator, such as a field-programmable gate array [3].

High-Level Additional Optimization As web information extraction moves towards Categories 3 and 4 (more complex tasks) there is more opportunity for improving efficiency [8, 23, 28]. Traditional relational optimizations, such as join reordering and pushing down selection predicates, can be beneficial. There are also additional text-specific optimizations that may yield significant performance gains:

- *Restricted Span Evaluation* optimizes an annotator to evaluate a particular rule on a subset of the document text – the text spans output by some other previously applied rule – rather than the complete text.
- *Conditional Evaluation* optimizes an annotator to evaluate a particular rule conditional on whether the evaluation of a previously applied rule satisfies certain predicates (e.g., it produced at least one annotation).

As in traditional database query optimization, effective selectivity estimation is an important component of any optimization strategy for IE rules. These rules often include text-specific selection and join predicates, as well as basic extraction primitives such as dictionary matching, which can vary widely in selectivity depending on the characteristics of the target text and the configuration of the predicate itself. Techniques for estimating the selectivity of these operations include random sampling and synopsis data structures [30]. Overall, systems that combine efficient evaluation of lower-level primitives with cost-based optimization at a higher level have reported an order of magnitude improvement in execution time [9, 28].

Usability

Usability refers to the ease of building an annotator to solve a specific extraction task. Recent rule-based systems [5, 9, 28] have adopted a declarative approach to information extraction, allowing the developer to specify *what* the annotator result should be, while leaving the choice of control flow for *how* to generate the result to the underlying system's optimizer. Declarative languages thus focus on building complex extractors without worrying about performance, thus increasing

the usability of an IE system. However, even with declarative languages, extractor development remains a labor-intensive process. Open challenges in reducing the human effort involved in developing extractors include the learning of IE rules, and developing tooling for non-programmers.

Learning of IE rules Historically, both rule-based and machine learning-based approaches have been applied to solving IE problems, but the efforts have remained largely separate, each with its own advantages and disadvantages [11]. Machine learning-based IE systems require less effort to develop, but are difficult to comprehend and may require large amounts of labeled data in order to perform well. Rule-based IE systems are easy to comprehend, maintain, debug, and optimize, rely less on labeled data, but require more human effort to develop. A combination of focused machine learning algorithms for learning IE rules would significantly reduce the human effort, needing only to consider good candidate rules suggested by the system, without going through the manual "trial and error" development process. At the same time, by using an IE language as a target language, the comprehensibility, maintainability and runtime performance benefits of rule-based approaches are preserved. There are several open challenges in this direction:

- **Learning basic primitives**: A lot of manual effort in developing IE rules is focused in specifying basic features, such as dictionaries or regular expressions. Therefore, machine learning algorithms focused on learning these primitives, such as generating regular expressions [6] or dictionaries [18, 24] from positive examples, and refining regular expressions from positive and negative examples [19], are very important.
- **Learning rules**: Given a set of basic primitives, such as regular expressions or dictionaries, and labeled data, it is also possible to induce a complete set of IE rules [22]. With an expressive rule language, a major challenge is preventing the system from learning arbitrarily complex rule sets, which would be difficult to understand or maintain. Research directions include devising measures for rule complexity, which may help constrain the search space, and creating visualization tools to help developers utilize automatically generated rules.
- **Learning for domain adaptation**: Domain adaptation refers to the task of adapting an existing IE system to a new data source or application. Recent work [21,25] explores determining the specific portions of an existing rule set responsible for many false positives and refining specific operators, such as span-based predicates or dictionary-based operators, to improved precision of the overall rule set. Open challenges include generating rule refinements for an expanded set of operators, and also improving the recall of the extractor.

Tooling for non-programmers Existing research on providing development support for building extractors has been traditionally focused on programmers. Recent work in this area ranges from tools for automatically generating explanations of false positives [26, 27] or false negatives [16], to higher-level abstractions catered towards specific information extraction tasks such as entity extraction [10] or relation extraction [1]. Such higher-level languages constitute one step forward in enabling less experienced programmers to build IE systems. A more audacious goal is enabling non-programmers to build IE programs without writing a single line of code. Open challenges include designing a visual language that is simple yet sufficiently powerful to build non-trivial extractors, and presenting the user with a list of automatically built extractors that may be relevant to a task.

Cross-References

- ▸ Information Extraction
- ▸ Languages for Web Data Extraction
- ▸ Metasearch Engines
- ▸ Probabilistic Databases
- ▸ Query Optimization

- Text Analytics
- Text Mining
- Web Advertising
- Web Data Extraction System
- Wrapper Induction

Recommended Reading

1. Akbik A, Konomi O, Melnikov M. Propminer: a workflow for interactive information extraction and exploration using dependency trees. In:ACL (conference system demonstrations). 2013.
2. Appelt DE, Onyshkevych B. The common pattern specification language. In: TIPSTER. 1998.
3. Atasu K, Polig R, Hagleitner C, Reiss FR. Hardware-accelerated regular expression matching for high-throughput text analytics. In: FPL. IEEE; 2013. p. 1–7.
4. Boguraev B. Annotation-based finite state processing in a large-scale NLP architecture. In: RANLP. 2003.
5. Bohannon P, Merugu S, Yu C, Agarwal V, DeRose P, Iyer AS, Jain A, Kakade V, Muralidharan M, Ramakrishnan R, Shen W. Purple sox extraction management system. : SIGMOD Rec. 2008;37(4):21–27.
6. Brauer F, Rieger R, Mocan A, Barczynski WM. Enabling information extraction by inference of regular expressions from sample entities. In: CIKM. 2011.
7. Burdick D, Hernández M, Ho H, Koutrika G, Krishnamurthy R, Popa L, Stanoi IR, Vaithyanathan S, Das S. Extracting, linking and integrating data from public sources: a financial case study. : IEEE Data Eng Bull. 2011;34(3):60–67.
8. Cafarella MJ, Etzion O. A search engine for natural language applications. In: WWW. 2005.
9. Chiticariu L, Krishnamurthy R, Li Y, Raghavan S, Reiss F, Vaithyanathan S. Systemt: an algebraic approach to declarative information extraction. In: ACL. 2010.
10. Chiticariu L, Krishnamurthy R, Li Y, Reiss F, Vaithyanathan S. Domain adaptation of rule-based annotators for named-entity recognition tasks. In: EMNLP. 2010.
11. Chiticariu L, Li Y, Reiss FR. Rule-based information extraction is dead! long live rule-based information extraction systems! In: EMNLP. 2013.
12. Cohen W, McCallum A. Information extraction from the world wide web. In: KDD. 2003.
13. Cunningham H. Information extraction, automatic. In: Encyclopedia of language and linguistics. 2nd ed. Elsevier; Amsterdam. 2005.
14. Doan A, Ramakrishnan R, Vaithyanathan S. Managing information extraction: state of the art and research directions. In: SIGMOD. 2006.
15. Grishman R, Sundheim B. Message understanding conference-6: a brief history. In: COLING. 1996.
16. Huang J, Chen T, Doan A, Naughton JF. On the provenance of non-answers to queries over extracted data. In: PVLDB;1(1):736–747
17. Lafferty J, McCallum A, Pereira F. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: ICML. 2001.
18. Li Y, Chu V, Blohm S, Zhu H, Ho H. Facilitating pattern discovery for relation extraction with semantic-signature-based clustering. In: CIKM. 2011.
19. Li Y, Krishnamurthy R, Raghavan S, Vaithyanathan S, Jagadish HV. Regular expression learning for information extraction. In: EMNLP. 2008.
20. Li Y, Krishnamurthy R, Vaithyanathan S, Jagadish H. Getting work done on the web: supporting transactional queries. In: SIGIR. 2006.
21. Liu B, Chiticariu L, Chu V, Jagadish HV, Reiss F. Automatic rule refinement for information extraction.: PVLDB. 2010;3(1):588–97.
22. Nagesh A, Ramakrishnan G, Chiticariu L, Krishnamurthy R, Dharkar A, Bhattacharyya P. Towards efficient named-entity rule induction for customizability. In: EMNLP-CoNLL. 2012.
23. Reiss F, Raghavan S, Krishnamurthy R, Zhu H, Vaithyanathan S. An algebraic approach to rule-based information extraction. In: ICDE. 2008.
24. Riloff E. Automatically constructing a dictionary for information extraction tasks. In: AAAI. 1993.
25. Roy S, Chiticariu L, Feldman V, Reiss F, Zhu H. Provenance-based dictionary refinement in information extraction. In: SIGMOD. 2013.
26. Sarma AD, Jain A, Bohannon P. Building a generic debugger for information extraction pipelines. In: CIKM. 2011.
27. Sarma AD, Jain A, Srivastava D. I4e: interactive investigation of iterative information extraction. In: SIGMOD. 2010.
28. Shen W, Doan A, Naughton J, Ramakrishnan R. Declarative information extraction using datalog with embedded extraction predicates. In: VLDB. 2007.
29. Wandelt S, Deng D, Gerdjikov S, Mishra S, Mitankin P, Patil M, Siragusa E, Tiskin A, Wang W, Wang J, Leser U. State-of-the-art in string similarity search and join. SIGMOD Rec. 2014;43(1):64–76.
30. Wang DZ, Wei L, Li Y, Reiss F, Vaithyanathan S. Selectivity estimation for extraction operators over text data. In: ICDE. 2011.
31. Zhang C, Baldwin T, Ho H, Kimelfeld B, Li Y. Adaptive parser-centric text normalization. In: ACL (1). 2013. p. 1159–68.