RAD: A Scalable Framework for Annotator Development

Sanjeet Khaitan #$1, Ganesh Ramakrishnan *2, Sachindra Joshi *3, Anup Chalamalla *4

#*InfoSpace Inc.*
India Subsidiary, Bangalore
INDIA
1sanjeet.khaitan@infospace.com

**IBM India Research Lab*
New Delhi
INDIA
2ganramkr@in.ibm.com
3jsachind@in.ibm.com
4achalama@in.ibm.com

$*Work done while at IBM India Research Lab*

***Abstract*— Developments in semantic search technology have motivated the need for efficient and scalable entity annotation techniques. We demonstrate RAD: a tool for Rapid Annotator Development on a document collection. RAD builds on a recent approach [1] that translates entity annotation rules into equivalent operations on the inverted index of the collection, to directly generate an annotation index (which can be used in search applications). To make the framework scalable, we use an industrial strength indexer, Lucene [2] and introduce some modifications to its API.**

The index also serves as a suitable representation for making quick comparisons with an indexed ground truth of annotations on the same collection to evaluate precision and recall of the annotations. RAD achieves at least an order of magnitude speedup over the standard approach of annotating a document-at-a-time as adopted by GATE [3]. The speedup factor increases with increase in the size of the collection, making RAD scalable. We cache intermediate results from the index operations, enabling quick update of the annotation index as well as speedy evaluation when rules are modified. This makes RAD suitable for rapid and interactive development of annotators.

I. Motivation

Developments in semantic search technology [4], [5], [6] have motivated the need for efficient and scalable entity annotation techniques that can be deployed on a large scale. Entity annotation involves associating one of several well-defined types with token sequences in unstructured documents. Example types are ‘person name’, ‘organization’, ‘place’, ‘date’, *etc.*

In this paper, we will restrict our attention to rule-based entity annotators. A rule for entity annotation is a pair consisting of a *pattern* and a *type*. A *pattern* defines the sequence of tokens that need to be identified and the corresponding *type* denotes the annotation type for the token sequence. In this paper, we only consider patterns that are regular expressions over tokens and over properties of tokens such as (i) which dictionaries they belong to, (ii) their part of speech and (iii) orthographic properties.

Manually developing a set of rules that perform annotation of a document collection to a desired level of precision and recall requires an iterative and interactive process wherein effects of slight changes to rules can be measured quickly. To enable this, it is desirable to have an annotator development framework that has the following properties:
(a) The annotator should perform close to real time annotation of reasonably large data sets (around 1 GB).
(b) The annotator should take advantage of the fact that typically, in an interactive rule development scenario, only slight modifications are made to rule sets. An example modification is the addition of rules having common subexpressions with already evaluated rules. Benefits of such modifications could be reaped through techniques such as caching of intermediate rule operations.
(c) It should be possible to perform real time quantitative evaluation of the annotations produced by the rules against gold standard annotations (in terms of measures such as precision and recall).
(d) The framework should also provide a user interface to enable real time qualitative evaluation of the rules through visualization of changes in annotations as rules are modified.

Most current rule-based techniques [3], [7] for this task operate at the document level, such that each rule is evaluated against one document at a time. The computational complexity of this approach varies linearly with the number of documents and the cost for annotating each document, which could be prohibiting for large document corpora. Another drawback of this approach is that the entire document collection needs to be reprocessed for every minor modification made in the set of rules.

Recently we proposed [1] a framework (henceforward referred to as *IndexAnnot*) for annotating a document collection with typed entities by working solely on its inverted index. The framework yielded an order of magnitude speedup over a stateof-the-art document-based annotator [3]. The improvement achieved was even more pronounced when annotations had to be produced for a set of rules that were incrementally added.

In this demo, we present a system (RAD) for Rapid Annotator Development. The underlying annotator rule engine of RAD is based on the *IndexAnnot* [1] approach, which is efficient and scalable. We cache results of intermediate annotation operations to enable speedy updates to the annotation index whenever slight changes are made to the rule base. We use an industrial strength annotator, Lucene [2] and appropriately extend its API to support the addition of new index entries (in our case annotation types), to the index of an existing document collection. Further, if the collection comes with an accompanying set of gold standard annotations (the ground truth), we create index entries for those annotations. We support quick quantitative evaluation of the rule-based annotations (in terms of precision and recall) by a simple linear comparison of the postings lists for the rule-based annotations against the postings lists for the gold standard annotations.

II. Background: Entity annotation using inverted index operations

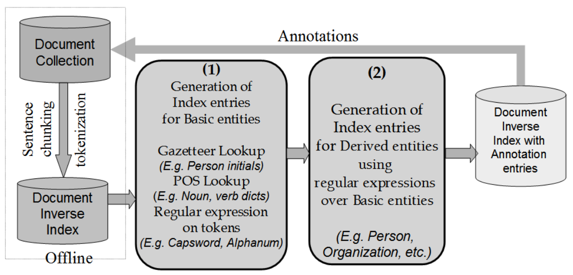

Document Collection
Annotations
Sentence chunking
tokenization
(1)
Generation of Index entries for Basic entities
Gazetteer Lookup
(E.g. Person initials)
POS Lookup
(E.g. Noun, verb dicts)
Regular expression on tokens
(E.g. Capsword, Alphanum)
(2)
Generation of Index entries for Derived entities using regular expressions over Basic entities
(E.g. Person, Organization, etc.)
Document Inverse Index with Annotation entries
Document Inverse Index
Offline

Fig. 1. Back-end process in the RAD tool

Figure 1 shows the process for entity annotation presented in [1], which forms the backend process in the RAD tool. A given document collection $\mathcal{D}$ is tokenized and segmented into sentences. The tokens are stored in an inverted index I. The inverted index I has an ordered list $\mathcal{U}$ of the unique tokens u_1, u_2, $..u_W$ that occur in the collection, where W is the number of tokens in I. Additionally, for each unique token u_i, I has a postings list $L(u_i) =< l_1, l_2, \ldots l_{cnt(u_i)} >$ of locations in $\mathcal{D}$ at which u_i occurs. $cnt(u_i)$ is the length of $L(u_i)$. In the approach described in [1], we designed each entry posting list entry l_k to have four fields: (1) a document identifier, $l_k.did$, (2) a sentence identifier, $l_k.sid$, (3) the begin position of the particular occurrence of u_i, $l_k.first$ and (4) the end position of the same occurrence of u_i, $l_k.last$. The entries in a postings list are sorted first by did, then by sid, followed by $first$ and finally by $last$.

The input grammar used in [1] is the same as that used for named entity annotations in GATE [3]. The GATE architecture for text engineering uses the Java Annotations Pattern Engine (JAPE) for its information extraction task. JAPE is a pattern matching language. [1] supports two classes of properties for tokens that are required by grammars such as JAPE: (1) orthographic properties such as an uppercase character followed by lower case characters, and (2) dictionaries (gazetteers) to which a token or a token sequence belongs. Examples of dictionaries are ‘location’ and ‘person name’. The set of tokens along with entity types specified by either of these two properties are referred to as *Basic Entities*. The instances of basic entities specified by orthographic properties must be single tokens. However, instances of basic entities specified using dictionary containment properties can be token sequences.

A. Generation of postings list for basic and derived entities

The module (1) of the system in Figure 1, identifies postings lists for each basic entity type. These postings lists are entered as index entries in I for the corresponding types. For example, if the input rules require tokens/token sequences that satisfy *Capsword* or *Location Dictionary* properties, a postings list is created for each of these basic types. The postings lists of tokens satisfying the same orthographic property are merged. A rule for NE annotation may require a token to satisfy multiple properties such as *Location Dictionary* as well as *Capsword*. The postings list for tokens that satisfy multiple properties are determined by performing an operation $parallelint(L, L')$ over the postings lists of the corresponding basic entities. The $parallelint(L, L')$ operation returns a postings list such that each entry in the returned list occurs in both L as well as L'.

The module (2) of the system shown in Figure 1 identifies instances of each annotation type, by performing index-based operations on the postings lists of *basic entity* types and other tokens. The different operation types and the sequence of their application will be discussed in the remainder of this section. Every intermediate postings list generated by module (2) has a field Opt. When any expression in the JAPE grammar is optional (or has the ’*’ or ’?’ operators associated), the value of Opt for the corresponding postings list is set to ‘true’.

Operations on Postings Lists: (a) $merge(L_1, L_2, \ldots, L_n)$: Returns a postings list such that each entry in the returned list occurs at least in one of the lists L_1 or L_2 .... or L_n. The Opt field of the resultant list is a disjunction of the Opt fields of the input lists.

(b) $consint(L, L')$: Returns a postings list such that each entry in the returned list points to a token sequence which consists of two consecutive subsequences $@sa$ and $@sb$ within the same sentence, such that, the list L has an entry for $@sa$ and L' has an entry for $@sb$. If either of L or L' has its Opt field set to true, the resultant list is merged with the corresponding optional postings list(s).

B. Inverted Index-based Annotation using an AND/OR Tree

Each annotation pattern (which is a regular expression over basic entities) is first translated into an AND/OR Tree [1]. An AND/OR Tree specifies a bottom-up sequence of $consint$

and $merge$ operations on postings lists to obtain a postings list for the annotation type at the root. Associated with each node in the tree is a regular expression and a postings list of all the matches in the collection for that node's regular expression. There are two types of nodes: AND node where the output list is computed from the consecutive intersection ($consint$) of the lists of two children nodes and OR node where the output list is computed by merging the lists of all the children nodes. Additionally, each node has two binary properties: Opt and $selfLoop$. The first property is set if the regular expression being matched is of the form 'R?', where '?' denotes that the regular expression R is optional. The second property is set if the regular expression is of the form 'R+', where '+' is the kleen operator denoting one or more occurrences. For the case of 'R*', both properties are set. The AND/OR Tree is recursively built by scanning the regular expression from left to right and identifying every sub-regular expression for which a sub-tree could be built. Details of the algorithm that builds the AND/OR Tree from a regular expression are provided in [1].

III. Description of the System

The goal of annotator development is generating a set of rules that annotate a document collection to a desired level of precision and recall. This development requires an iterative and interactive process [5] wherein effects of slight changes to rules can be measured quickly. RAD has been developed keeping this goal in mind.

Figure 2 shows the high-level architecture of the RAD tool. The tool uses the technique for annotation described in [1] and gives an order of magnitude speedup over the document-at-a-time paradigm, enabling the quick annotation of any pre-indexed large document collection.

Given a document collection (D), a Lucene-based inverted index (BI) is first created for the collection. The rules and dictionaries for NEA are stored in a rule-base (RB). Using the technique described in [1], rules and dictionaries in the rule base are matched directly against the base inverted index to create index entries for named entities in the annotation index (AI).

If the document collection comes with an accompanying set of gold standard annotations (the ground truth G), these annotations can be stored in the gold annotation index (GI). The advantage of creating the gold annotation index is that precision and recall for the rule-base can be quickly computed by simply comparing the entries in the annotation index against the entries in the gold index. The base inverted index, the annotation index and the gold index are just logically separate parts of a common Lucene index (LI). All the indexes are stored using Lucene [2]. Some details regarding the integration of Lucene are provided in Section III-A.

RAD comes with a user interface (UI) that has the following facilities:
(a) It allows the user to specify the document collection and create a base inverted index.
(b) It has an editor for creating and modifying the rule base and the rules therein.
(c) It facilitates the rapid creation of annotation index using a set of specified rule base and base inverted index.
(d) Given the gold annotation set for a document collection, it allows the user to populate the gold annotation index.
(e) It quickly evaluates the precision and recall for a specified rule base and a document collection by comparing the resultant annotation index against the gold index.
(f) The user interface enables the user to browse through annotations for individual documents, and help refine rules by providing diagnosis of mistakes in the annotations.

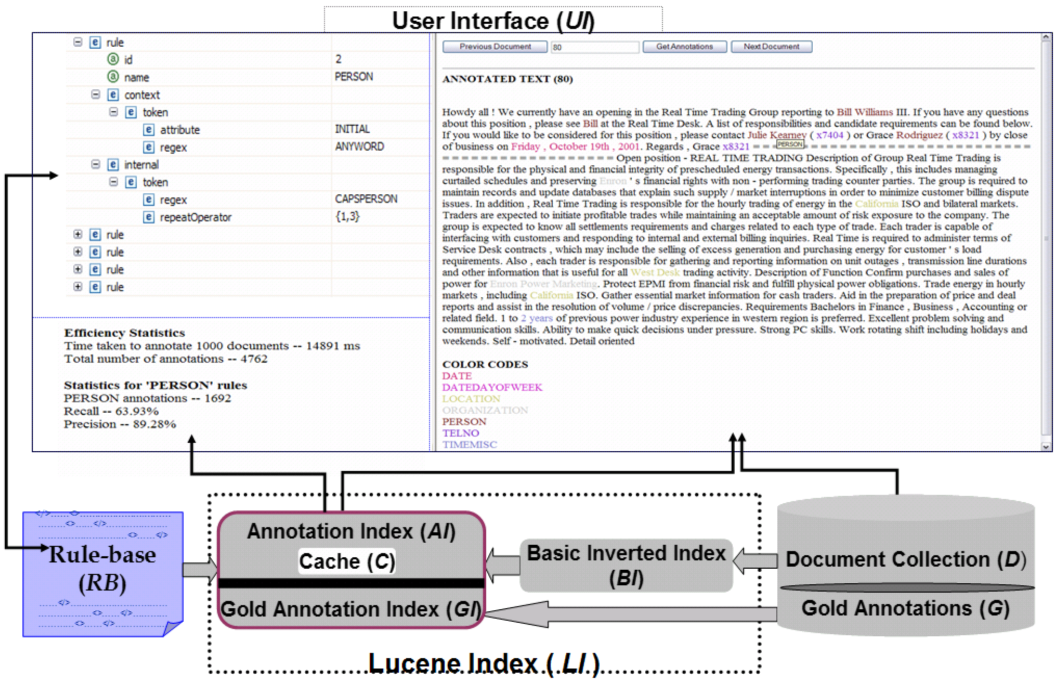
For a set of 8 rules across 4 NE types, the process of named entity annotation directly using the inverted index yields speedup factors of 8 and 13 over GATE for the Reuters and Enron corpora respectively. The annotation index also holds a cache (C) of intermediate postings lists generated from previous index operations. The cache enables the reuse of results of common sequences of operations on postings lists across different NE rules. The cache also enables rapid update of the annotation index when named entity rules are modified. *E.g.*, when NEA is performed for an additional set of 4 rules, the speedup factors increase to 23 and 37 for the Reuters and Enron corpora respectively.

A. Index Storage using Lucene

Lucene [2] is an open source search engine project written in JAVA. It provides API for creating and managing inverted indexes. For every unique term T, it maintains a postings list $loc_1, loc_2, \ldots loc_n$, where each location loc_i contains two elements doc_i and pos_i, which are the document number and the term position in the document respectively. The annotation postings lists are very similar to those of the terms postings lists except that an annotation can span several words in a document. In order to store annotation information in Lucene's inverted index format we split the annotation postings list into two. The first postings list contains the begin positions of the annotations and the second stores the end positions. It is easy to see that a linear merge algorithm will yield the actual postings list for the annotation. Given the index for an existing collection, Lucene does not provide any API for introducing new terms whose postings lists refer to the same collection. We extended the Lucene API to enable the addition of new terms (which are annotation types in our case).

IV. Demonstration

Our demonstration will showcase the RAD system using two document collections (i) the publicly available Enron data set consisting of about 250,000 email messages and (b) a combination of Reuters-21578 and the 20 Newsgroups data sets. We illustrate using some homegrown rules for 4 types, *viz.*, Person name, Organization, Location and Date. We also demonstrate the speedup for incremental



```
rule
    id                    2
    name                  PERSON
    context
        token
            attribute     INITIAL
            regex         ANYWORD
    internal
        token
            regex         CAPSPERSON
            repeatOperator {1,3}
rule
rule
rule
rule
```

Efficiency Statistics
Time taken to annotate 1000 documents -- 14891 ms
Total number of annotations -- 4762

Statistics for 'PERSON' rules
PERSON annotations -- 1692
Recall -- 63.93%
Precision -- 89.28%

80

ANNOTATED TEXT (80)

Howdy all ! We currently have an opening in the Real Time Trading Group reporting to Bill Williams III. If you have any questions about this position , please see Bill at the Real Time Desk. A list of responsibilities and candidate requirements can be found below. If you would like to be considered for this position , please contact Julie Kearney (x7404) or Grace Rodriguez (x8321) by close of business on Friday , October 19th , 2001. Regards , Grace x8321 = = = PERSON = Open position - REAL TIME TRADING Description of Group Real Time Trading is responsible for the physical and financial integrity of prescheduled energy transactions. Specifically , this includes managing curtailed schedules and preserving Enron ' s financial rights with non - performing trading counter parties. The group is required to maintain records and update databases that explain such supply / market interruptions in order to minimize customer billing dispute issues. In addition , Real Time Trading is responsible for the hourly trading of energy in the California ISO and bilateral markets. Traders are expected to initiate profitable trades while maintaining an acceptable amount of risk exposure to the company. The group is expected to know all settlements requirements and charges related to each type of trade. Each trader is capable of interfacing with customers and responding to internal and external billing inquiries. Real Time is required to administer terms of Service Desk contracts , which may include the selling of excess generation and purchasing energy for customer ' s load requirements. Also , each trader is responsible for gathering and reporting information on unit outages , transmission line durations and other information that is useful for all West Desk trading activity. Description of Function Confirm purchases and sales of power for Enron Power Marketing. Protect EPMI from financial risk and fulfill physical power obligations. Trade energy in hourly markets , including California ISO. Gather essential market information for cash traders. Aid in the preparation of price and deal reports and assist in the resolution of volume / price discrepancies. Requirements Bachelors in Finance , Business , Accounting or related field. 1 to 2 years of previous power industry experience in western region is preferred. Excellent problem solving and communication skills. Ability to make quick decisions under pressure. Strong PC skills. Work rotating shift including holidays and weekends. Self - motivated. Detail oriented

COLOR CODES
DATE
DATEDAYOFWEEK
LOCATION
ORGANIZATION
PERSON
TELNO
TIMEMISC

Fig. 2. High Level Architecture of the RAD tool

annotation by introducing new rules for the 4 types. We then present how RAD could help improve the quality of rules quickly and interactively, with the help of gold annotations. We show a side-by-side comparison with the GATE annotator.

REFERENCES

[1] G. Ramakrishnan, S. Balakrishnan, and S. Joshi. (2006) Entity annotation based on inverse index operations.

[2] Apache. Apache lucene: A high performance, full featured text search engine. [Online]. Available: http://lucene.apache.org

[3] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "GATE: A framework and graphical development environment for robust NLP tools and applications," in *Proceedings of ACL*, 2002.

[4] S. Dill, N. Eiron, D. Gibson, D. Gruhl, and R. G. et. al., "Semtag and seeker: bootstrapping the semantic web via automated semantic annotation," in *Proceedings of the WWW*, 2003.

[5] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu, "Avatar semantic search: a database approach to information retrieval," in *Proceedings of the ACM SIGMOD*, 2006.

[6] S. Chakrabarti, K. Puniyani, and S. Das, "Optimizing scoring functions and indexes for proximity search in type-annotated corpora," in *Proceedings of WWW*, 2006.

[7] D. Appelt, J. Hobbs, J. Bear, D. Israel, M. Kameyama, D. Martin, K. Myers, and M. Tyson, "Sri intnl fastus system: Muc-6 test results and analysis," in *Proc. of MUC*, 1995.