Towards Efficient Named-Entity Rule Induction for Customizability

Ajay Nagesh[1,2]
[1]IITB-Monash Research Academy
ajaynagesh@cse.iitb.ac.in

Ganesh Ramakrishnan
[2]IIT Bombay
ganesh@cse.iitb.ac.in

Laura Chiticariu
IBM Research - Almaden
chiti@us.ibm.com

Rajasekar Krishnamurthy
IBM Research - Almaden
rajase@us.ibm.com

Ankush Dharkar
SASTRA University
ankushdharkar@cse.sastra.edu

Pushpak Bhattacharyya
IIT Bombay
pb@cse.iitb.ac.in

Abstract

Generic rule-based systems for Information Extraction (IE) have been shown to work reasonably well out-of-the-box, and achieve state-of-the-art accuracy with further domain customization. However, it is generally recognized that manually building and customizing rules is a complex and labor intensive process. In this paper, we discuss an approach that facilitates the process of building customizable rules for Named-Entity Recognition (NER) tasks via rule induction, in the Annotation Query Language (AQL). Given a set of basic features and an annotated document collection, our goal is to generate an initial set of rules with reasonable accuracy, that are interpretable and thus can be easily refined by a human developer. We present an efficient rule induction process, modeled on a four-stage manual rule development process and present initial promising results with our system. We also propose a simple notion of extractor complexity as a first step to quantify the interpretability of an extractor, and study the effect of induction bias and customization of basic features on the accuracy and complexity of induced rules. We demonstrate through experiments that the induced rules have good accuracy and low complexity according to our complexity measure.

1 Introduction

Named-entity recognition (NER) is the task of identifying mentions of rigid designators from text belonging to named-entity types such as persons, organizations and locations (Nadeau and Sekine, 2007). Generic NER rules have been shown to work reasonably well-out-of-the-box, and with further domain customization (Chiticariu et al., 2010b), achieve quality surpassing state-of-the-art results. Table 1 summarizes the quality of NER rules out-of-the-box and after domain customization in the GATE (Cunningham et al., 2011) and SystemT (Chiticariu et al., 2010a) systems, as reported in (Maynard et al., 2003) and (Chiticariu et al., 2010b) respectively.

System	**Dataset**	**$F_{\beta=1}$**	
		Generic	**Customized**
GATE	ACE2002	57.8	82.2
SystemT	ACE 2005	57.32	88.95
	CoNLL 2003	64.15	91.77
	Enron	76.53	85.29

Table 1: Quality of generic vs. customized rules.

Rule-based systems are widely used in enterprise settings due to their *explainability*. Rules are transparent, which leads to better explainability of errors. One can easily identify the cause of a false positive or negative, and refine the rules without affecting other correct results identified by the system. Furthermore, rules are typically easier to understand by an IE developer and can be customized for a new domain without requiring additional labeled data.

Typically, a rule-based NER system consists of a combination of four categories of rules (Chiticariu et al., 2010b): (1) *Basic Feature (BF)* rules to identify components of an entity such as first name and last name. (2) *Candidate definition (CD)* rules to identify complete occurrences of an entity by combining the output of multiple BF rules, e.g., first name followed by last name is a person candidate. (3) *Candidate refinement (CR)* rules to refine candidates generated by CD rules. E.g., discard candidate persons contained within organizations. (4) *Consolidation rules (CO)* to resolve overlapping candidates generated by multiple CD and CR rules.

A well-known drawback that influences the adoptability of rule-based NER systems is the manual effort required to build the rules. A common approach to address this problem is to build a generic NER extractor and then customize it for specific domains. While this approach partially alleviates the problem, substantial manual effort (in the order of several person weeks) is still required for the two stages as reported in (Maynard et al., 2003; Chiticariu et al., 2010b). In this paper, we present initial work towards facilitating the process of building a generic NER extractor using induction techniques.

Specifically, given as input an annotated document corpus, a set of BF rules, and a default CO rule for each entity type, our goal is to generate a set of CD and CR rules such that the resulting extractor constitutes a good starting point for further refinement by a developer. Since the generic NER extractor has to be manually customized, a major challenge is to ensure that the generated rules have good accuracy, and, at the same time, that they are *not too complex*, and consequently *interpretable*.

The main contributions in this paper are

1. An *efficient* system for NER rule induction, using a highly expressive rule language (AQL) as the target language. The first phase of rule induction uses a combination of *clustering* and *relative least general generalization (RLGG)* techniques to learn CD rules. The second phase identifies CR rules using a propositional rule learner like *JRIP* to learn accurate compositions of CD rules.
2. Usage of induction biases to enhance the interpretability of rules. These biases capture the expertise gleaned from manual rule development and constrain the search space in our induction system.
3. Definition of an initial notion of extractor complexity to quantify the interpretability of an extractor and to guide the process of adding induction biases to favor learning *less complex* extractors. This is to ensure that the rules are easily customizable by the developer.
4. Scalable induction process through usage of SystemT, a state-of-the-art IE system which serves as a highly efficient theorem prover for AQL, and performance optimizations such as clustering of examples and parallelizing various modules (E.g.: propositional rule learning).

Roadmap We first describe preliminaries on SystemT and AQL (Section 3) and define the target language for our induction algorithm and the notion of rule complexity (Section 4). We then present our approach for inducing CD and CR rules, and discuss induction biases that would favor interpretability (Section 5), and discuss the results of an empirical evaluation (Section 6). We conclude with avenues for improvement in the future (Section 7).

2 Related Work

Existing approaches to rule induction for IE focus on rule-based systems based on the cascading grammar formalism exemplified by the Common Pattern Specification Language (CPSL) (Appelt and Onyshkevych, 1998), where rules are specified as a sequence of basic features that describe an entity, with limited predicates in the context of an entity mention. Patel et al. (2009) and Soderland (1999) elaborate on top-down techniques for induction of IE rules, whereas (Califf and Mooney, 1997; Califf and Mooney, 1999) discuss a bottom-up IE rule induction system that uses the *relative least general generalization (RLGG)* of examples[1]. However, in all these systems, the expressivity of the rule-representation language is restricted to that of capturing sequence information. As discussed in Section 3, contextual clues and higher level rule interactions such as *filtering* and *join* are very difficult, if not impossible to express in such representations without resorting to custom code. Learning higher level interactions between rules has received little attention. Our technique for learning higher level interactions is similar to the induction of *ripple down rules* (Gaines and Compton, 1995), which, to the best of our knowledge, has not been previously applied to IE. A framework for refining AQL extractors based on an annotated document corpus described in (Liu et al., 2010). We present complementary techniques for inducing an initial extractor that can be automatically refined in this framework.

3 Preliminaries

SystemT is a declarative IE system based on an algebraic framework. In SystemT, developers write rules in AQL. To represent annotations in a docu-

[1]Our work also makes use of RLGGs but computes these generalizations for clusters of examples, instead of pairs.

```
R1:   create view Caps as
         extract regex /[A-Z](\w|-)+/ on D.text as match from Document D;

R2:   create view First as
         extract dictionary 'FirstNameGazeteer' on D.text as match from Document D;

R3:   create view Last as
         extract dictionary 'LastNameGazeteer' on D.text as match from Document D;

R4:   create view PersonFirst as
         select F.match as match
         from First F, Caps C
         where Equals(F.match, C.match);

R5:   create view PersonFirstLast as
         select CombineSpans(F.match, L.match) as match
         from First F, Last L, Caps C
         where FollowsTok(F.match, L.match, 0, 0) and Equals(L.match, C.match);

R6:   create view PersonCandidate as
         (select R.match from PersonFirst R)
         union all
         (select R.match from PersonFirstLast R);

R7:   create view PersonInvalid as
         select P.match
         from PersonCandidate P, Organization O
         where Overlaps (P.match, O.match);

R8:   create view PersonAll as
         (select R.match from PersonCandidate R)
         minus
         (select R.match from PersonInvalid R);

R9:   create view Person as
         select R.match
         from PersonAll R
         consolidate on R.match using 'ContainedWithin';
```

Complexity of extractor C(E) = 15 + C(Organization)

Figure 1: Example *Person* extractor in AQL

ment, AQL uses a simple relational data model with three types: a *span* is a region of text within a document identified by its "begin" and "end" positions; a *tuple* is a fixed-size list of spans; a *relation*, or *view*, is a multi-set of tuples, where every tuple in the view must be of the same size.

Figure 1 shows a portion of a *Person* extractor written in AQL. The basic building block of AQL is a *view*. A *view* is a logical description of a set of tuples in terms of (i) the document text (denoted as a special view called *Document*), and (ii) the contents of other views, as specified in the *from* clauses of each statement. Figure 1 also illustrates five of the basic constructs that can be used to define a view, and which we explain next. The complete specification can be found in the AQL manual (IBM, 2012). In the paper, we will use 'rules' and 'views' interchangeably.

The *extract* statement specifies basic character-level extraction primitives such as regular expression and dictionary matching over text, creating a tuple for each match. As an example, rule R_1 uses the *extract* statement to identify matches (*Caps* spans) of a regular expression for capitalized words.

The *select* statement is similar to the SQL select statement but it contains an additional *consolidate on* clause (explained further), along with an extensive collection of text-specific predicates. Rule R_5 illustrates a complex example: it selects *First* spans immediately followed within zero tokens by a *Last* span, where the latter is also a *Caps* span. The two conditions are specified using two join predicates: *FollowsTok* and *Equals* respectively. For each triplet of *First*, *Last* and *Caps* spans satisfying the two predicates, the *CombineSpans* built-in scalar function in the select clause constructs larger *PersonFirstLast* spans that begin at the begin position of the *First* span, and end at the end position of the *Last* (also *Caps*) span.

The *union all* statement merges the outputs of two or more statements. For example, rule R_6 unions person candidates identified by rules R_4 and R_5.

The *minus* statement subtracts the output of one statement from the output of another. For example, rule R_8 defines a view *PersonAll* by filtering out *PersonInvalid* tuples from the set of *PersonCandidate* tuples. Notice that rule R_7 used to define the view *PersonInvalid* illustrates another join predicate of AQL called *Overlaps*, which returns true if its two argument spans overlap in the input text. Therefore, at a high level, rule R_8 removes person candidates that overlap with an *Organization* span. (The *Organization* extractor is not depicted in the figure.)

The *consolidate* clause of a *select* statement removes selected overlapping spans from the indicated column of the input tuples, according to the specified policy (for instance, 'ContainedWithin'). For example, rule R_9 retains *PersonAll* spans that are not contained in other *PersonAll* spans.

Internally, SystemT compiles an AQL extractor into an executable plan in the form of a graph of *operators*. The formal definition of these operators takes the form of an algebra (Reiss et al., 2008), similar to relational algebra, but with extensions for text processing. The decoupling between AQL and the operator algebra allows for greater rule expressivity because the rule language is not constrained by the need to compile to a finite state transducer, as in grammar systems based on the CPSL standard. In fact, join predicates such as *Overlaps*, as well as filter operations (*minus*) are extremely difficult to express in CPSL systems such as GATE without an escape to custom code (Chiticariu et al., 2010b). In addition, the decoupling between the AQL specification of *"what"* to extract from *"how"* to extract it, allows greater flexibility in choosing an efficient execution strategy among the many possible operator graphs that may exist for the same AQL extractor. Therefore, extractors written in AQL achieve orders of magnitude higher throughput (Chiticariu et al., 2010a).

4 Induction Target Language

Our goal is to automatically generate NER extractors with good quality, and at the same time, manageable complexity, so that the extractors can be further refined and customized by the developer. To this end, we focus on inducing extractors using the subset of AQL constructs described in Section 3. We note that we have chosen a small subset of AQL constructs that are sufficient to implement several common operations required for NER. However, AQL is a much more expressive language, and incorporating additional constructs is subject to our future work.

In this section we describe the building blocks of our target language, and propose a simple definition for measuring the complexity of an extractor.

Target Language. The components of the target language are as follows, and summarized in Table 2.

Basic features (BF): *BF views* are specified using the *extract* statement, such as rules R_1 to R_3 in Figure 1. In this paper, we assume as input a set of *basic features*, consisting of dictionaries and regular expressions.

Candidate definition (CD): *CD* views are expressed using the *select* statement to combine BF views with join predicates (e.g., *Equals*, *FollowsTok* or *Overlaps*), and the *CombineSpans* scalar function to construct larger candidate spans from input spans. Rules R_4 and R_5 in Figure 1 are example CD rules. In general, a CD view is defined as: "*Select all spans constructed from view$_1$, view$_2$, ..., view$_n$, such that all join predicates are satisfied*".

Candidate refinement (CR): *CR* views are used to discard spans output by the CD views that may be incorrect. In general, a CR view is defined as: "*From the list of spans of view$_{valid}$ subtract all those spans that belong to view$_{invalid}$*". *view$_{valid}$* is obtained by joining all the positive CD clues on the *Equals* predicate and *view$_{invalid}$* is obtained by joining all the negative overlapping clues with the *Overlaps* predicate and subsequently 'union'ing all the negative clues. (*e.g.*, similar in spirit to rules R_6, R_7 and R_8 in Figure 1, except that the subtraction is done from a single view and not the *union* of multiple views).

Consolidation (CO): Finally, a *select* statement with a fixed *consolidate* clause is used for each entity type to remove overlapping spans from CR views. An example CO view is defined by rule R_9 in Figure 1.

Extractor Complexity. Since our goal is to generate extractors with manageable complexity, we must introduce a quantitative measure of extractor complexity, in order to (1) judge the complexity of the extractors generated by our system, and (2) reduce the search space considered by the induction system.

To this end, we define a simple complexity score that is a function of the number of rules, and the number of input views to each rule of the extractor. In particular, we define the *length of rule R*, denoted as $L(R)$, as the number of input views in the *from* clause(s) of the view. For example, in Figure 1, we have $L(R_4) = 2$ and $L(R_5) = 3$, since R_4 and R_5 have two, and respectively three views in the *from* clause. Furthermore, $L(R_8) = 2$ since each of the two inner statements of R_8 has one *from* clause with a single input view. The complexity of BF rules (e.g., R_1 to R_3) and CO rules (e.g., R_9) is always 1, since these types of rules have a single input view. We define the *complexity of extractor E*, denoted as $C(E)$ as the sum of lengths of all rules of E. For example, the complexity of the *Person* extractor from Figure 1 is 15, plus the length of all rules involved in defining *Organization*, which are omitted from the figure.

Our simple notion of rule length is motivated by existing literature in the area of database systems (Abiteboul et al., 1995), where the *size* of a conjunctive query is determined only by the number of atoms in the body of the query (e.g., items in the FROM clause), and it is independent on the number of join variables (i.e., items in the WHERE clause), or the size of the head of the query (e.g., items in the SELECT clause). As such, our notion of complexity is rather coarse, and we shall discuss its shortcomings in detail in Section 6.2. However, we shall show that the complexity score significantly reduces the search space of our induction techniques leading to

Phase name	AQL statements	Prescription	Rule Type
Basic Features	`extract`	Off-the-shelf, Learning using prior work (Riloff, 1993; Li et al., 2008)	Basic Features Definition
Phase 1 (Clustering and RLGG)	`select`	Bottom-up learning (LGG), Top-down refinement	Development of Candidate Rules
Phase 2 (Propositional Rule Learning)	`select, union all, minus`	RIPPER, Lightweight Rule Induction	Candidate Rules Filtering
Consolidation	`consolidate, union all`	Manually identified consolidation rules, based on domain knowledge	Consolidation rules

Table 2: Phases in induction, the language constructs invoked in each phase, the prescriptions for inducing rules in the phase and the corresponding type of rule in manual rule development.

simpler and smaller extractors, and therefore constitutes a good basis for more comprehensive measures of interpretability in the future.

5 Induction of Rules

Since the goal is to generate rules that can be customized by humans, the overall structure of the induced rules must be similar in spirit to what a developer following best practices would write. Hence, the induction process is divided into multiple phases. Figure 2 shows the correspondence between the phases of induction and the types of rules. In Table 2, we summarize the phases of our induction algorithm, along with the subset of AQL constructs that comprise the language of the rules learnt in that phase, the possible methods prescribed for inducing the rules and their correspondence with the stages in the manual rule development.

Our induction system generates rules for two of the four categories, namely CD and CR rules as highlighted in Figure 2. We assume that we are given the BFs in the form of dictionaries and regular expressions. Prior work on learning dictionaries (Riloff, 1993) and regular expressions (Li et al., 2008) could be leveraged to semi-automate the process of defining the basic features.

We represent each example, in conjunction with relevant background knowledge in the form *first order horn clauses*. This background knowledge will serve as input to our induction system. The first phase of induction uses a combination of *clustering* and *relative least general generalization (RLGG)* (Nienhuys-Cheng and Wolf, 1997; Muggleton and Feng, 1992) techniques. At the end of this phase, we have a number of CD rules. In the second phase, we begin by forming a structure called the *span-view table*. Broadly speaking, this is an attribute-value table formed by all the views induced in the first phase along with the textual spans generated by them. The attribute-value table is used as input to a propositional rule learner such as *JRIP* to learn accurate compositions of a useful (as determined by the learning algorithm) subset of the CD rules. This forms the second phase of our system. The rules learnt from this phase are the CR rules. At various phases, several induction biases are introduced to enhance the interpretability of rules. These biases capture the expertise gleaned from manual rule development and constrain the search space in our induction system.

The *hypothesis language* of our induction system is *Annotation Query Language (AQL)* and we use *SystemT* as the *theorem prover*. SystemT provides a very fast rule execution engine and is crucial in our induction system as we test multiple hypotheses in the search for the more promising ones. AQL provides a very expressive rule representation language that is proven to be capable of encoding all the paradigms that any rule-based representation can encode. The dual advantages of *rich rule-representation* and *execution efficiency* are the main motivation behind our choice.

We discuss our induction procedure in detail next.

5.1 Basic Features and Background Knowledge

We assume that we are provided with a set of dictionaries and regular expressions for defining all our basic *create view* statements. R_1, R_2 and R_3 in Figure 1 are such basic view definitions. The basic views are compiled and executed in SystemT over the training document collection and the resulting spans are represented by equivalent predicates in first order logic. Essentially, each training example is represented as a definite clause,

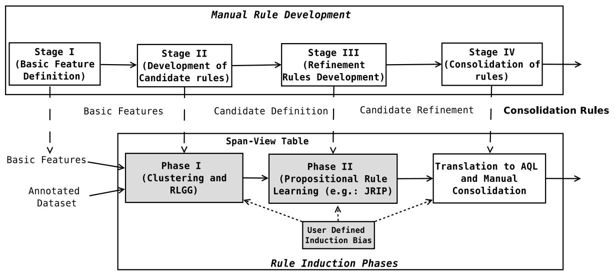

Manual Rule Development
Stage I (Basic Feature Definition)
Stage II (Development of Candidate rules)
Stage III (Refinement Rules Development)
Stage IV (Consolidation of rules)
Basic Features
Candidate Definition
Candidate Refinement
Consolidation Rules
Span-View Table
Basic Features
Annotated Dataset
Phase I (Clustering and RLGG)
Phase II (Propositional Rule Learning (e.g.: JRIP)
Translation to AQL and Manual Consolidation
User Defined Induction Bias
Rule Induction Phases

Figure 2: Correspondence between Manual Rule development and Rule Induction.

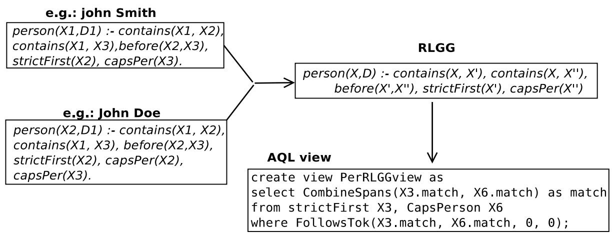
that includes in its body, the basic view-types encoded as background knowledge predicates. To establish relationships between different background knowledge predicates for each training example, we define some additional *"glue predicates"* such as **`contains`** and **`before`**.

5.2 Induction of Candidate Definition Rules

Clustering Module. We obtain non-overlapping clusters of examples within each type, by computing similarities between their representations as definite clauses. We present the intuition behind this approach in Figure 3 which illustrates the process of taking two examples and finding their generalization. It is worthwhile to look at generalizations of instances that are similar. For instance, two token person names such as *Mark Waugh* and *Mark Twain* are part of a single cluster. However, we would not be able to generalize a two-token name (*e.g.*, *Mark Waugh*) with another name consisting of initials followed by a token (*e.g.*, *M. Waugh*). Using a wrapper around the hierarchical agglomerative clustering implemented in LingPipe[2], we cluster examples and look at generalizations only within each cluster. Clustering also helps improve efficiency by reducing the computational overhead, since otherwise, we would have to consider generalizations of all pairs of examples (Muggleton and Feng, 1992).

RLGG computation. We compute our CD rules as the *relative least general generalization (RLGG)* (Nienhuys-Cheng and Wolf, 1997; Muggleton and Feng, 1992) of examples in each cluster. Given a set of clauses in first order logic, their RLGG is the least generalized clause in the

[2]http://alias-i.com/lingpipe/demos/tutorial/cluster/read-me.html



```
person(X1,D1) :- contains(X1, X2),
contains(X1, X3),before(X2,X3),
strictFirst(X2), capsPer(X3).
```

```
person(X2,D1) :- contains(X1, X2),
contains(X1, X3), before(X2,X3),
strictFirst(X2), capsPer(X2),
capsPer(X3).
```

```
person(X,D) :- contains(X, X'), contains(X, X''),
      before(X',X''), strictFirst(X'), capsPer(X'')
```

```
create view PerRLGGview as
select CombineSpans(X3.match, X6.match) as match
from strictFirst X3, CapsPerson X6
where FollowsTok(X3.match, X6.match, 0, 0);
```

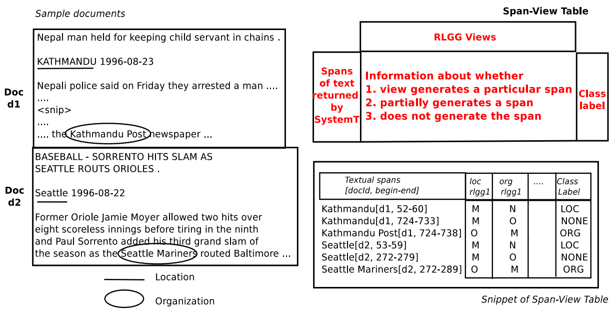
Figure 3: Relative Least General Generalization

subsumption lattice of the clauses relative to the background knowledge (Nienhuys-Cheng and Wolf, 1997). RLGG is associative, and we use this fact to compute RLGGs of sets of examples in a cluster. The RLGG of two bottom clauses as computed in our system and its translation to an AQL view is illustrated in Figure 3. We filter out noisy RLGGs and convert the selected RLGGs into the equivalent AQL views. Each such AQL view is treated as a CD rule. We next discuss the process of filtering-out noisy RLGGs. We interchangeably refer to the RLGGs and the clusters they represent .

Iterative Clustering and RLGG filtering. Since clustering is sub-optimal, we expected some clusters in a single run of clustering to have poor RLGGs, either in terms of complexity or precision. We therefore use an iterative clustering approach, based on the *separate-and-conquer* (Fürnkranz, 1999) strategy. In each iteration, we pick the clusters with the best RLGGs and remove all examples covered by those RLGGs. The best RLGGs must have precision and number of examples covered above a pre-specified threshold.

5.3 Induction of Candidate Refinement Rules

Span-View Table. The CD views from phase 1 along with the textual spans they generate, yield the *span-view table*. The rows of the table correspond to the *set of spans* returned by all the CD views. The columns correspond to the set of CD view names. Each span either belongs to one of the named entity types (PER, ORG or LOC) or is none of them (NONE); the type information constitutes its class label (see Figure 4 for an illustrated example). The cells in the table correspond to either a match (M) or a no-match (N) or partial/overlapping match (O) of a span generated by a CD view. This attribute-value table is used as input to a propositional rule learner


Sample documents
Doc d1
Doc d2
Location
Organization
Span-View Table
RLGG Views
Spans of text returned by SystemT
Information about whether
1. view generates a particular span
2. partially generates a span
3. does not generate the span
Class label
Snippet of Span-View Table

Nepal man held for keeping child servant in chains .

KATHMANDU 1996-08-23

Nepali police said on Friday they arrested a man ....
....
<snip>
....
.... the Kathmandu Post newspaper ...

BASEBALL - SORRENTO HITS SLAM AS
SEATTLE ROUTS ORIOLES .

Seattle 1996-08-22

Former Oriole Jamie Moyer allowed two hits over eight scoreless innings before tiring in the ninth and Paul Sorrento added his third grand slam of the season as the Seattle Mariners routed Baltimore ...

Textual spans [docId, begin-end]	loc rlgg1	org rlgg1	....	Class Label
Kathmandu[d1, 52-60]	M	N		LOC
Kathmandu[d1, 724-733]	M	O		NONE
Kathmandu Post[d1, 724-738]	O	M		ORG
Seattle[d2, 53-59]	M	N		LOC
Seattle[d2, 272-279]	M	O		NONE
Seattle Mariners[d2, 272-289]	O	M		ORG

Figure 4: Span-View Table

like *JRIP* to learn compositions of CD views.

Propositional Rule Learning. Based on our study of different propositional rule learners, we decided to use *RIPPER* (Fürnkranz and Widmer, 1994) implemented as the *JRIP* classifier in weka (Witten et al., 2011). Some considerations that favor JRIP are (i) absence of rule ordering, (ii) ease of conversion to AQL and (iii) amenability to add induction biases in the implementation.

A number of syntactic biases were introduced in JRIP to aid in the interpretability of the induced rules. We observed in our manually developed rules that CR rules for a type involve interaction between CDs for the same type and negations (not-overlaps, not matches) of CDs of the other types. This bias was incorporated by constraining a JRIP rule to contain only positive features (CDs) of the same type (say PER) and negative features (CDs) of only other types (ORG and LOC, in this case).

The output of the JRIP algorithm is a set of rules, one set for each of PER, ORG and LOC. Here is an example rule: `PER-CR-Rule` $\Leftarrow$ `(PerCD = m) AND (LocCD != o)` which is read as : *"If a span matches PerCD and does not overlap with LocCD, then that span denotes a PER named entity"*. Here `PerCD` is $\{$`[FirstName` $\wedge$ `CapsPerson][LastName` $\wedge$ `CapsPerson]`$\}$ [3] and `LocCD` is $\{$`[CapsPlace` $\wedge$ `CitiesDict]`$\}$. This rule filters out wrong person annotations like *"Prince William"* in *Prince William Sound*. (This is the name of a location but has overlapped with a person named entity.) In AQL, this effect can be achieved most elegantly by the `minus` (filter) construct. Such an AQL rule will filter all those occurrences of *Prince William* from the list of persons that overlap with a city name.

Steps such as clustering, computation of RLGGs, JRIP, and theorem proving using SystemT were parallelized. Once the CR views for each type of named entity are learnt, many forms of consolidations (COs) are possible, both within and across types. A simple consolidation policy that we have incorporated in the system is as follows: union all the rules of a particular type, then perform a *contained within* consolidation, resulting in the final set of consolidated views for each named entity type.

6 Experiments

We evaluate our system on CoNLL03 (Tjong Kim Sang and De Meulder, 2003), a collection of Reuters news stories. We used the CoNLL03 training set for induction and report results on the CoNLL03 test collection.

The basic features (BFs) form the primary input to our induction system. We experimented with three sets of BFs:

*Initial set(**E1**)*: The goal in this setup is to induce an initial set of rules based on a small set of reasonable BFs. We use a conservative initial set consisting of 15 BFs (5 regular expressions and 10 dictionaries).

*Enhanced set (**E2**)*: Based on the results of E1, we identify a set of additional domain independent BFs[4]. Five views were added to the existing set in E1 (1 regular expression and 4 dictionaries). The goal is to observe whether our approach yields reasonable accuracies compared to generic rules developed manually.

*Domain customized set (**E3**)*: Based on the knowledge of the domain of the training dataset (CoNLL03), we introduced a set of features specific to this dataset. These included sports-related person, organization and location dictionaries[5]. These views were added to the existing set in E2. The intended goal is to observe what are the best possible accuracies that could be achieved with BFs customized to a particular domain.

The set of parameters for iterative clustering on which the accuracies reported are : the precision threshold for the RLGGs of the clusters was 70%

[3]Two consecutive spans where the 1st is FirstName and CapsPerson and the 2nd is LastName and CapsPerson.

[4]E.g., the feature *preposition dictionary* was added in E2 to help identify organization names such as *Bank of England*.

[5]Half of the documents in CoNLL03 are sports-related.

	Train			Test			
Type	**P**	**R**	**F**	**P**	**R**	**F**	**C(E)**
E1 (Initial set)							
PER	88.5	41.4	56.4	92.5	39.4	55.3	144
ORG	89.1	7.3	13.4	85.9	5.2	9.7	22
LOC	91.6	54.5	68.3	87.3	55.3	67.8	105
Overall	90.2	35.3	50.7	89.2	33.3	48.5	234
E2 (Enhanced set)							
PER	84.7	52.9	65.1	87.5	49.9	63.5	233
ORG	88.2	7.8	14.3	85.8	5.9	11.0	99
LOC	92.1	58.6	71.7	88.6	59.1	70.9	257
Overall	88.6	40.7	55.8	88.0	38.2	53.3	457
E3 (Domain customized set)							
PER	89.9	57.3	70.0	91.7	56.0	69.5	430
ORG	86.9	50.9	64.2	86.9	47.5	61.4	348
LOC	90.8	67.0	77.1	84.3	67.3	74.8	356
Overall	89.4	58.7	70.9	87.3	57.0	68.9	844

Table 3: Results on CoNLL03 dataset with different basic feature sets

and the number of examples covered by each RLGG was 5. We selected the top 5 clusters from each iteration whose RLGGs crossed this threshold. If there were no such clusters then we would lower the precision threshold to 35% (half of the threshold). When no new clusters were formed, we ended the iterations.

6.1 Experiments and Results

Effect of Augmenting Basic Features. Table 3 shows the accuracy and complexity of rules induced with the three basic feature sets E1, E2 and E3, respectively [6]. The overall F-measure on the test dataset is 48.5% with E1, it increases to around 53.3% with E2 and is highest at 68.9% with E3. As we increase the number of BFs, the accuracies of the induced extractors increases, at the cost of an increase in complexity. In particular, the recall increases significantly across the board, and is more prominent between E2 and E3, where the additional domain specific features result in recall increase from 5.9% to 47.5% for ORG. The precision increases slightly for PER, but decreases slightly for LOC and ORG with the addition of domain specific features.

Comparison with manually developed rules. We compared the induced extractors with the manually developed extractors of (Chiticariu et al., 2010b), heretofore referred to as *manual extractors*. (For a detailed analysis, we obtained the extractors from the authors). Table 4 shows the accuracy and complexity of the induced rules with E2 and E3 and the manual extractors for the generic domain and, respectively, customized for the CoNLL03 domain. (In the table, ignore the column *Induced (without bias)*, which is discussed later). Our technique compares reasonably with the manually constructed generic extractor for two of the three entity types; and on precision for all entity types, especially since our system generated the rules in 1 hour, whereas the development of manual rules took much longer [7]. Additional work is required to match the manual customized extractor's performance, primarily due to shortcomings in our current target language. Recall that our framework is limited to a small subset of AQL constructs for expressing CD and CR rules, and there is a single consolidation rule. In particular, advanced constructs such as dynamic dictionaries are not supported, and the set of predicates to the Filter construct supported in our system is restricted to predicates over other concepts, which is only a subset of those used in (Chiticariu et al., 2010b). The manual extractors also contain a larger number of rules covering many different cases, improving the accuracy, but also leading to a higher complexity score. To better analyze the complexity, we also computed the average rule length for each extractor by dividing the complexity score by the number of AQL views of the extractor. The average rule length is 1.78 and 1.87 for the induced extractors with E2 and E3, respectively, and 1.9 and 2.1 for the generic and customized extractors of (Chiticariu et al., 2010b), respectively. The average rule length increases from the generic extractor to the customized extractor in both cases. On average, however, an individual induced rule is slightly smaller than a manually developed rule.

Effect of Bias. The goal of this experiment is to demonstrate the importance of biases in the induction process. The biases added to the system are broadly of two types: (i) Partition of basic features based on types (ii) Restriction on the type of CD views that can appear in a CR view. [8] Without

[6]These are the results for the configuration with bias.

[7] (Chiticariu et al., 2010b) mentions that customization for 3 domains required 8 person weeks. It is reasonable to infer that developing the generic rules took comparable effort.

[8]For e.g., person CR view can contain only person CD views as positive clues and CD views of other types as negative clues.

(i) many semantically similar basic features (especially, regular expressions) would match a given token, leading to an increase in the length of a CD a rule. For example, in the CD rule [FirstNameDict][CapsPerson $\wedge$ CapsOrg]} (*"A FirstNameDict span followed by a CapsPerson span that is also a CapsOrg span"*), CapsPerson and CapsOrg are two very similar regular expressions identifying capitalized phrases that look like person, and respectively, organization names, with small variations (e.g., the former may allow special characters such as '-'). Including both BFs in a CD rule leads to a larger rule that is unintuitive for a developer. The former bias excludes such CD rules from consideration.

The latter type of bias prevents CD rules of one type to appear as positive clues for a CR rule of a different type. For instance, without this bias, one of the CR rules obtained was `Per` $\Leftarrow$ `(OrgCD = m) AND (LocCD != o)` (*"If a span matches OrgCD and does not overlap with LocCD, then that span denotes a PER named entity"*. Here `OrgCD` was {`[CapsOrg][CapsOrg]`} and `LocCD` was {`[CapsLoc` $\wedge$ `CitiesDict]`}. The inclusion of an Organization CD rule as a positive clue for a Person CR rule is unintuitive for a developer.

Table 4, shows the effect (for E2 and E3) on the test dataset of disabling and enabling bias during the induction of CR rules using JRIP. Adding bias improves the precision of the induced rules. Without bias, however, the system is less constrained in its search for high recall rules, leading to slightly higher overall F measure. This comes at the cost of an increase in extractor complexity and average rule length. For example, for E2, the average rule length decreases from 2.17 to 1.78 after adding the bias. Overall, our results show that biases lead to less complex extractors with only a very minor effect on accuracy, thus biases are important factors contributing to inducing rules that are understandable and may be refined by humans.

Comparison with other induction systems. We also experimented with two other induction systems, Aleph[9] and ALP[10], a package that implements one of the reportedly good information extraction algorithms (Ciravegna, 2001). While induction in Aleph was performed with the same target language as in our approach, the target language of ALP is JAPE, which has been shown (Chiticariu et al., 2010b) to lack in some of the constructs (such as minus) that AQL provides and which form a part of our target language (especially the rule refinement phase). However, despite experimenting with all possible parameter configurations for each of these (in each of E1, E2 and E3 settings), the accuracies obtained were substantially (30-50%) worse and the extractor complexity was much (around 60%) higher when compared to our system (with or without bias). Additionally, Aleph takes close to three days for induction, whereas both ALP and our system require less than an hour.

6.2 Discussion

Weak and Strong CDs reflected in CRs. In our experiments, we found that varying the precision and complexity thresholds while inducing the CDs (*c.f* Section 5) affected the F1 of the final extractor only minimally. But reducing the precision threshold generally improved the precision of the final extractor, which seemed counterintuitive at first. We found that CR rules learned by JRIP consist of a strong CD rule (high precision, typically involving a dictionary) and a weak CD rule (low precision, typically involving only regular expressions). The strong CD rule always corresponded to a positive clue (match) and the weak CD rule corresponded to the negative clue (overlaps or not-matches). This is illustrated in the following CR rule: `PER` $\Leftarrow$ `(PerCD = m) AND (OrgCD != o)` where `(PerCD` is {`[CapsPersonR] [CapsPersonR` $\wedge$ `LastNameDict]`} and `(OrgCD` is {`[CapsOrgR][CapsOrgR][CapsOrgR]`}. This is posited to be the way the CR rule learner operates – it tries to learn conjunctions of weak and strong clues so as to filter one from the other. Therefore, setting a precision threshold too high limited the number of such weak clues and the ability of the CR rule learner to find such rules.

Interpretability. Measuring interpretability of rules is a difficult problem. In this work, we have taken a first step towards measuring interpretability using a coarse grain measure in the form of a simple notion of complexity score. The complexity is very helpful in comparing alternative rule sets based on

[9]A system for inductive logic programming. See http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html

[10]http://code.google.com/p/alpie/

		Chiticariu et al. 2010b				Induced (With Bias)				Induced (Without Bias)			
		P	**R**	**F**	**C(E)**	**P**	**R**	**F**	**C(E)**	**P**	**R**	**F**	**C(E)**
Generic (E2)	*PER*	82.2	60.3	69.5	945	87.5	49.9	63.5	233	85.8	53.7	66.0	476
	ORG	75.7	17.5	28.5	1015	85.8	5.9	11.0	99	74.1	15.7	25.9	327
	LOC	72.2	86.1	78.6	921	88.6	59.1	70.9	257	85.9	61.5	71.7	303
	Overall	75.9	54.6	63.5	1015	88.0	38.2	53.3	457	84.2	43.5	57.4	907
Customised (E3)	*PER*	96.3	92.2	94.2	2154	91.7	56.0	69.5	430	90.7	60.3	72.4	359
	ORG	91.1	85.1	88.0	2154	86.9	47.5	61.4	348	90.4	46.8	61.7	397
	LOC	93.3	91.7	92.5	2154	84.3	67.3	74.8	356	83.9	69.1	75.8	486
	Overall	93.5	89.6	91.5	2160	87.3	57.0	68.9	844	87.8	58.7	70.4	901

Table 4: Comparison of induced rules (with and without bias) and manually developed rules. (CoNLL03 test dataset)

the number of rules, and the size of each rule, but exhibits a number of shortcomings described next. First, it disregards other components of a rule besides its from clause, for example, the number of items in the select clause, or the where clause. Second, rule developers use semantically meaningful view names such as those shown in Figure 1 to help them recall the semantics of a rule at a high-level, an aspect that is not captured by the complexity measure. Automatic generation of meaningful names for induced views is an interesting direction for future work. Finally, the overall structure of an extractors is not considered. In simple terms, an extractor consisting of 5 rules of size 1 is indistinguishable from an extractor consisting of a single rule of size 5, and it is arguable which of these extractors is more interpretable. More generally, the extent of this shortcoming is best explained using an example. When informally examining the rules induced by our system, we found that CD rules are similar in spirit to those written by rule developers. On the other hand, the induced CR rules are too fine-grained. In general, rule developers group CD rules with similar semantics, then write refinement rules at the higher level of the group, as opposed to the lower level of individual CD views. For example, one may write multiple CD rules for candidate person names of the form $\langle First\rangle\langle Last\rangle$, and multiple CD rules of the form $\langle Last\rangle$, $\langle First\rangle$. One would then union together the candidates from each of the two groups into two different views, e.g., *PerFirstLast* and *PerLastCommaFirst*, and write filter rules at the higher level of these two views, e.g., *"Remove PerLastCommaFirst spans that overlap with a PerFirstLast span"*. In contrast, our induction algorithm considers CR rules consisting of combinations of CD rules directly, leading to many semantically similar CR rules, each operating over small parts of a larger semantic group (see rule in Section 6.1). This results in repetition, and qualitatively less interpretable rules, since humans prefer higher levels of abstraction and generalization. This nuance is not captured by the complexity score which may deem an extractor consisting of many rules, where many of the rules operate at higher levels of groups of candidates to be more complex than a smaller extractor with many fine-grained rules. Indeed, as shown before, the complexity of the induced extractors is much smaller compared to that of manual extractors, although the latter follow the semantic grouping principle and are considered more interpretable.

7 Conclusion

We presented a system for efficiently inducing named entity annotation rules in the AQL language. The design of our approach is aimed at producing accurate rules that can be understood and refined by humans, by placing special emphasis on low complexity and efficient computation of the induced rules, while mimicking a four stage approach used for manually constructing rules. The induced rules have good accuracy and low complexity according to our complexity measure. While our complexity measure informs the biases in our system and leads to simpler, smaller extractors, it captures extractor interpretability only to a certain extent. Therefore, we believe more work is required to devise a more comprehensive quantitative measure for interpretability, and refine our techniques in order to increase the interpretability of induced rules. Other interesting directions for future work are introducing more constructs in our framework, and applying our techniques to other languages.

References

S. Abiteboul, R. Hull, and V. Vianu. 1995. *Foundations of Databases*. Addison Wesley Publishing Co.

Douglas E. Appelt and Boyan Onyshkevych. 1998. The common pattern specification language. In *TIPSTER workshop*.

Mary Elaine Califf and Raymond J. Mooney. 1997. Applying ilp-based techniques to natural language information extraction: An experiment in relational learning. In *IJCAI Workshop on Frontiers of Inductive Logic Programming*.

Mary Elaine Califf and Raymond J. Mooney. 1999. Relational learning of pattern-match rules for information extraction. In *AAAI*.

Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick R. Reiss, and Shivakumar Vaithyanathan. 2010a. Systemt: an algebraic approach to declarative information extraction. In *ACL*.

Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. 2010b. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *EMNLP*.

Fabio Ciravegna. 2001. (lp)2, an adaptive algorithm for information extraction from web-related texts. In *In Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. 2011. *Text Processing with GATE (Version 6)*.

J. Fürnkranz and G. Widmer. 1994. Incremental reduced error pruning. pages 70–77.

Johannes Fürnkranz. 1999. Separate-and-conquer rule learning. *Artif. Intell. Rev.*, 13(1):3–54, February.

B. R. Gaines and P. Compton. 1995. Induction of ripple-down rules applied to modeling large databases. *J. Intell. Inf. Syst.*, 5:211–228, November.

IBM, 2012. *IBM InfoSphere BigInsights - Annotation Query Language (AQL) reference*. `http://publib.boulder.ibm.com/infocenter/bigins/v1r3/topic/com.ibm.swg.im.infosphere.biginsights.doc/doc/biginsights_aqlref_con_aql-overview.html`.

Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and H. V. Jagadish. 2008. Regular expression learning for information extraction. In *EMNLP*.

Bin Liu, Laura Chiticariu, Vivian Chu, H. V. Jagadish, and Frederick R. Reiss. 2010. Automatic rule refinement for information extraction. *Proc. VLDB Endow.*, 3:588–597.

Diana Maynard, Kalina Bontcheva, and Hamish Cunningham. 2003. Towards a semantic extraction of named entities. In *In Recent Advances in Natural Language Processing*.

Stephen Muggleton and C. Feng. 1992. Efficient induction in logic programs. In *ILP*.

D. Nadeau and S. Sekine. 2007. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30:3–26.

Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. 1997. *Foundations of Inductive Logic Programming*.

Anup Patel, Ganesh Ramakrishnan, and Pushpak Bhattacharyya. 2009. Incorporating linguistic expertise using ilp for named entity recognition in data hungry indian languages. In *ILP*.

Frederick Reiss, Sriram Raghavan, Rajasekar Krishnamurthy, Huaiyu Zhu, and Shivakumar Vaithyanathan. 2008. An algebraic approach to rule-based information extraction. In *ICDE*.

Ellen Riloff. 1993. Automatically constructing a dictionary for information extraction tasks. In *AAAI*.

Stephen Soderland. 1999. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34:233–272.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: language-independent named entity recognition. In *HLT-NAACL*.

Ian H. Witten, Eibe Frank, and Mark A. Hall. 2011. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Amsterdam, 3rd edition.