Natural Language Engineering **1** (1): 1–39. Printed in the United Kingdom

A Semi-automatic and Low-cost Method to Learn Patterns for Named Entity Recognition†

M. MARRERO

Barcelona Supercomputing Center, Spain
E-mail: marrero.monica@gmail.com

J. URBANO

Delft University of Technology, The Netherlands
E-mail: urbano.julian@gmail.com

(*Received 17 January 2016; revised 18 April 2017; accepted 19 April 2017*)

Abstract

Named Entity Recognition is a basic task in Information Extraction that aims at identifying entities of interest within full text documents. The patterns used to recognize entities can be rule-based, as in the popular JAPE system. However, hand-crafting effective patterns is often difficult, and yet there is little research devoted to methods capable of learning human-readable patterns, possibly with arbitrary sets of features. In this paper we present a semi-automatic method to generate both regular expressions and a subset of the JAPE language. It does not need a corpus annotated beforehand. Instead, it employs active learning and combines clustering with an algorithm that finds alignments between symbols present in the entities discovered during the learning process. The method currently supports a fixed set of character features and an arbitrary set of token features, but it can incorporate other kinds of features as well. Through several experiments with an English corpus we show the ability of the method to generate effective patterns at a low annotation cost, and how it can successfully help in the annotation of brand new corpora.

1 Introduction

The dramatic rate at which digital information is generated requires in many cases to go beyond merely retrieving full documents in response to a query, for instance by extracting the exact pieces of information that users are looking for (Gantz and Reinsel 2012). The goal of Named Entity Recognition (NER) is to identify entities of interest in full documents, such as persons, locations and dates. NER is a basic area in Information Extraction, necessary for example to identify referents and their relations, or scenario compositions in order to satisfy specific user needs. It is a topic of research especially active in fields like Biomedicine (Nédellec, Bossy, Kim,

† This work was partially supported by the Spanish Government through a Juan de la Cierva fellowship and project MDM-2015-0502. We specially thank Jorge Morato and Sonia Sánchez for their advice, as well as the anonymous reviewers for their suggestions.

Kim, Ohta, Pyysalo and Zweigenbaum 2013; Marrero, Sánchez-Cuadrado, Urbano, Morato and Moreiro 2012) or Social Media Analysis (Li, Wei, Zhang and Zhou 2013; Ritter, Clark, Mausam and Etzioni 2011), and it is applied in other fields related to Information Management and Natural Language Processing, such as Semantic Annotation (Uren, Cimiano, Iria, Handschuh, Vargas-Vera, Motta and Ciravegna 2006; Reeve and Han 2005), Question Answering (Srihari and Li 1999), Ontology Population (Maedche and Staab 2001; Etzioni, Cafarella, Downey, Popescu, Shaked, Soderland, Weld and Yates 2005) and Opinion Mining (Pang and Lee 2007; Popescu and Etzioni 2005).

At the core of NER systems are patterns capable of recognizing the entities of interest. Even though statistical approaches are more frequently used to learn these patterns, hand-crafted rules still play an important role in real world applications, especially in the industry (Chiticariu and Reiss 2013). The need for traceable results, the lack of existing annotated data and the difficulty to create new annotations, as well as unclear or unstable specifications, are some of the reasons that favor rule-based approaches. Examples of successful rule-based systems are GATE with JAPE (Java Annotation Patterns Engine) (Cunningham et al. 2013), and Apache UIMA Ruta (Klueg, Toepfer, Beck, Fette and Puppe 2015). However, the user is required to learn specific formal languages to write these rules. Even though there are tools that help doing this, such as NooJ (Silberztein 2005) for grammars or Ultrapico Expresso for regular expressions, the user still has to review large amounts of text to debug these rules and possibly refine them for the corpus at hand.

A natural development is thus the creation of systems capable of generating recognition rules in an automatic fashion. However, currently there is little attention to such methods. Former methods employ a reduced set of features, and they seldom allow the use of custom features or features of different levels of granularity. In addition, these systems often learn in a supervised manner that requires large numbers of annotations to obtain effective patterns.

We have developed a new method to semi-automatically learn regular expressions and JAPE patterns that does not require any corpora annotated beforehand. Instead, it guides the user throughout the annotation process thanks to active learning techniques, thus reducing the overall annotation cost required to generate the patterns. The method uses features at both character and token levels to describe entities, and applies exactly the same algorithm to identify rules in both cases, thus simplifying its implementation and future extension to other types of feature. The performance of the method is evaluated with three experiments and a varied set of entity types. First, we show that it requires a small annotation effort to learn JAPE patterns that achieve good effectiveness scores compared to the state of the art. Second, we show that it can also be used to exhaustively annotate new corpora, sparing users the need to review large amounts of text. Third, we show that it can similarly learn very effective regular expressions from a small set of initial seeds.

The remainder of the paper is organized as follows. Section 2 presents an overview of NER and related work. Section 3 describes the proposed algorithm to identify rules for NER, and Section 4 describes how it is efficiently combined with clustering and active learning in order to learn valid rules. Section 5 describes general evaluation settings, and Sections 6, 7 and 8 present the experimental details and results. Finally, Sections 9 and 10 present a final discussion and finish with conclusions and lines for future research.

2 Named Entity Recognition

2.1 Problem Statement

The problem of Named Entity Recognition can be formulated as follows. Let $\mathcal{E}$ be the set of all entities of a particular type (e.g. person names, countries, dates), and let P be a pattern capable of recognizing all these entities. That is, $\mathcal{E} = L(\mathsf{P})$ where $L(\mathsf{P})$ is the language recognized by P, and thus the pattern recognizes the language of the particular entity type we are looking for and nothing else. The goal is to generate a pattern $\hat{\mathsf{P}}$ that estimates P from a set $\mathcal{E}^+ \subseteq \mathcal{E}$ of positive examples (i.e. known entities) and possibly a set $\mathcal{E}^- \nsubseteq \mathcal{E}$ of negative examples (i.e. text fragments that are not entities of the target type).

These patterns are based on different combinations of features that characterize entities in $\mathcal{E}$ and differentiate them from the rest of the text. These features have different degrees of granularity. They are often found at the character level (e.g. character classes) and at the token level (e.g. POS tag, morphological category), though we can also find discourse features. Let Σ be the alphabet of all characters. A character feature $f^\mathcal{C}$ is a function $f^\mathcal{C} : \Sigma \to \Sigma^\mathcal{C}$ that maps a character onto a symbol of a certain alphabet $\Sigma^\mathcal{C}$ specific of the feature. A token feature $f^\mathcal{T}$ is a function $f^\mathcal{T} : \Sigma^* \to \Sigma^\mathcal{T}$ that maps a sequence of characters (i.e. a token) onto a symbol in the feature output alphabet. Different NER systems can of course use different sets of features.

For our purposes, a pattern learning method for NER is thus defined as a function that estimates P from the sets $\mathcal{E}^+$ and $\mathcal{E}^-$ of positive and negative examples, sets $\mathcal{F}^\mathcal{C}$ and $\mathcal{F}^\mathcal{T}$ of character and token features, and a corpus $\mathcal{D}$ of documents. In this paper we propose a semi-automatic method to learn $\hat{\mathsf{P}}$, such that $L(\hat{\mathsf{P}}) \approx L(\mathsf{P})$. The accuracy of the pattern is measured in terms of precision and recall, that is, its ability to generate the same language as P.

2.2 Pattern Learning Methods

Different machine learning approaches can be found in the literature for the automatic creation of NER patterns. According to Sarawagi (2008), these can be broadly categorized as rule-based and statistical. While the former may produce a customized model designed to exploit specific characteristics of the task, the latter try to map the NER problem to some generic theoretical model that has to be fitted. Rule-based methods are easier to develop and interpret, while statistical methods are more robust to noisy unstructured text. Nonetheless, both rule-based and statistical learning methods are still used depending on the specificities of the extraction task (Sarawagi 2008). Chiticariu and Reiss (2013) show that rule-based methods are more frequent in industrial settings, while statistical methods are used more often in academia, probably because the former are easier to adopt, understand, debug and maintain in scenarios where requirements are changing.

We find several rule-based approaches in the literature. *Whisk* (Soderland 1999) and *Amilcare* (Ciravegna and Wilks 2003) calculate combinations of feature values and select the ones that best identify entities. *RAPIER* (Thompson, Califf and Money 1999) computes possible alignments between feature values and selects the ones with the best trade-off between accuracy and complexity of the pattern. Wu and Pottenger (2005) learn patterns by identifying sequences of features and iteratively expanding patterns from the most common feature value across entities. Li, Krishnamurthy, Raghavan, Vaithyanathan and Jagadish (2008) adapt regular expressions from annotated examples. Brauer, Rieger, Moca and Barczynski's method (2011) analyzes the prefixes and suffixes of the entities in order to determine character patterns. Finally, Nagesh and Chiticariu (2012) induce rules by applying a subset of operators from the declarative language AQL with basic features previously identified in the text.

Regarding statistical learning approaches, we find the use of Markov Models (Bikel, Miller, Schwartz and Weischedel 1997; Hachey, Alex and Becker 2005), Maximum Entropy Models (Borthwick, Sterling, Agichtein and Grishman 1998), Conditional Random Fields (Tomanek, Wermter and Hahn 2007; McCallum and Li 2003; Finkel, Grenager and Manning 2005), Support Vector Machines (Asahara and Matsumoto 2003; Shen, Zhang, Su, Zhou and Tan 2004; Vlachos 2008; Li, Bontcheva and Cunningham 2009), Neural Networks (Kazama and Torisawa 2007; Ratinov and Roth 2009) and Decision Trees (Sekine, Grishman and Shinnou 1998).

Some other works have studied the application of bootstrapping techniques for NER (Gupta and Manning 2014; Jones 2005; Nadeau 2007; Pasca, Lin, Bigham, Lifchits and Jain 2006; Etzioni et al. 2005), as well as other unsupervised techniques (Alfonseca and Manandhar 2002; Shinyama and Sekine 2004; Ritter et al. 2011). Finally, some recent works have studied hybrid approaches (Fersini, Messina, Felici and Roth 2014; Irmak and Kraft 2010; Liu et al. 2013; Nouvel et al. 2012).

2.3 Annotations

The initial sets $\mathcal{E}^+$ and $\mathcal{E}^-$ are usually part of the corpus, where these entities are already annotated. Statistical learning algorithms require a large number of such annotations (Settles 2012), which often requires the user to review very large amounts of text. As a result, these algorithms are seldom practical when there are none or just a few annotations available. Unfortunately, this is usually the case in NER when facing non-traditional types of entities (e.g. apparel brands, software applications, years of experience) or simply a brand new corpus for which known patterns are not suitable (Marrero, Urbano, Sánchez-Cuadrado, Morato and Gómez-Berbís 2013; Marrero, Sánchez-Cuadrado, Morato and Andreadakis 2009). In addition, these algorithms may generate patterns biased towards precision because $\mathcal{E}^+$ is normally outnumbered by $\mathcal{E}^-$, that is, there are many more negative examples than positive examples (Li et al. 2009).

Active learning techniques can reduce annotation costs, asking users to annotate only certain documents or text snippets throughout the learning process. The sets $\mathcal{E}^+$ and $\mathcal{E}^-$ are initially smaller, and they are augmented with the new annotations made during the learning process. Overall annotation costs are reduced by identifying the most informative or relevant text snippets to annotate. Active learning has been previously applied to both rule-based (Thompson et al. 1999; Soderland 1999; Wu and Pottenger 2005) and statistical learning methods (Hachey et al. 2005; Shen et al. 2004; Vlachos 2008; Li et al. 2009; Tomanek et al. 2007). However, even with active learning it is still necessary in many cases to have many initial annotations to obtain satisfactory results.

Some studies discuss different approaches to measure the effort required from the user (Settles 2012). For instance, Vijayanarasimhan and Grauman (2009) estimates annotation cost from the time it takes the user to annotate a few examples. Nevertheless, the most common measures of required user effort are the number of elements to review (e.g. number of tokens), and the number of actions required (e.g. change a label). A very important aspect in the first case is the length of the text units that users are given to annotate. In the case of NER, *RAPIER* (Thompson et al. 1999) uses full documents, *Whisk* (Soderland 1999) and Wu and Pottenger (2005) use predefined text snippets (assuming previous knowledge about the type of documents and the boundaries of the entities), Hachey et al. (2005) and Tomanek et al. (2007) use sentences, Shen et al. (2004) use the entities themselves, and Vlachos (2008) and Jones (2005) use tokens. Li et. al. (2009) compare the use of documents, text fragments and tokens with active learning. They observed that when the annotation units are text fragments or tokens instead of full documents, the performance of algorithms is significantly lower. But large documents do not necessarily contain more entities, so using full documents makes both overall annotation costs and algorithm performance more variable across document types.

Regarding the measurement of effort as the number of actions taken by the user, Culotta and Mccallum (2005) consider the actions of correcting the start boundary, end boundary, and type of an entity, as well as choosing from a list of suggestions with various boundaries. Others try to combine the number of elements to review and the number of actions. For instance, Ringger, Carmen, Haertel, Seppi, Londsdale, McClanahan, Carroll and Ellison (2008) and Haertel, Seppi, Ringger and Carroll (2008) apply an hourly cost model to the labeling of POS that depends on the number of tokens per sentence reviewed by the user, and the number of words whose label needs correcting.

2.4 Features

Entities may be recognized with regular expressions when their patterns respond to sequences of characters, such as in e-mail addresses, phone numbers and names of some genes and proteins. However, not even the typical NER entities (e.g. location, organization, person) can be recognized with only these features; custom token features are needed as well to describe more complex patterns. It is therefore important that a pattern learning method permits *generality* (i.e. to employ a diverse set of features to be able to recognize various entity types, even in different types of doc-

```
Phase: personsA
Input: Token Lookup
Rule: rule1
({Token.category == "NNP",
  Token.string ==~ "A.+",
  Lookup.majorType == "person_first"}) :pname
--> :pname.Person = {type = "first"}

Phase: doctorsA
Input: Token Person
Rule: rule1
({Token.string ==~ "Dr\\.?"}
 {Person.type == "first"}) :drname
--> :drname.Person = {title = "doctor"}
```

Fig. 1. Sample cascade grammar in JAPE notation. Each phase represents a finite state transducer, whose input alphabet is specified after `Input`. Each rule contains a left-hand-side with a pattern, the separator `-->`, and a right-hand-side with an action. The text matched by the pattern can be identified with a label (eg. `:pname`), which is used in the action to add certain annotations to that specific string. In the above example, the first phase recognizes person first names, made up of a token with morphosyntactic category `NNP`, whose initial character is `A` (`==~` indicates that the right-hand side is a regular expression), and is also included in the `person_first` gazetteer. The text matched by this pattern is annotated as `Person.type="first"`. The second phase uses these annotations to recognize first names that are preceded by a token matching the regular expression `Dr\\.?`, and annotates them as `Person.title="doctor"`.

ument) as well as *retargetability* (i.e. to include custom features, even at different granularity levels, to recognize specific or novel types of entities) (Freitag1998; Marrero et al. 2013). These capabilities have a direct impact on the learning method, which needs to deal with an arbitrarily large number of features and alphabet symbols. This is especially troublesome with features at the character level, because they are applied to each of the individual characters of the entities and lead to a high degree of variability. This is often the case with statistical learning methods, which very rarely work with character-level features and, when they do, they usually employ *n*-grams instead of single characters.

2.5 Representation of Patterns

An important difference between rule-based and statistical learning approaches is the transparency of the generated patterns. Statistical learning patterns are used as a black-box, because they are based on statistical models that are hard to interpret by humans. On the other hand, rule-based patterns can be more easily interpreted and analyzed, even during the learning process (Siniakov 2008). This further allows users to edit and refine patterns, possibly aided by computer tools (Chiticariu, Krishnamurthy, Li, Reiss and Vaithyanathan 2010). The need to represent features at different levels of granularity gave rise to a number of different ways and syntaxes to specify rule-based patterns in NER, from formal patterns such as regular expressions to ad-hoc patterns that represent predefined features. Unfortunately, neither regular expressions nor higher level grammars allow more than one input alphabet at once. On the other hand, languages for ad-hoc patterns, except maybe for recent developments like the UIMA language, lack standardization (Marrero and Urbano 2015).

Cascade grammars do allow us to use several types of features in the same pattern. The Common Pattern Specification Language (CPSL) (Appelt and Onyshkevych 1998) standardizes this representation, and it is widely used in Information Extraction tasks. It has been adopted and customized to different extents by different platforms (Boguraev 2004; Drozdzynski, Krieger, Piskorski, Schfer and Xu 2004; Rinaldi et al. 2005), *GATE* (Cunningham et al. 2013) being one of the most successful ones with its JAPE notation, which provides finite state transduction over GATE's document annotations. Each transducer consists of a set of pattern-action rules, the actions being new annotations over the matched text (see Fig. 1). The rule-based methods described in Section. 2.2 are in principle capable of generating some valid rules for this standard, but they usually employ only character features to generate regular expressions (Brauer et al. 2011; Li et al. 2008), or only token features, predefined (Soderland 1999; Thompson et al. 1999) or not (Nagesh and Chiticariu 2012; Ciravegna and Wilks 2003). There are some approaches like (Wu and Pottenger 2005) that use both types of features, but they can not be customized. The method we present in this paper is capable of generating cascade grammars with features that are more customizable than in other rule-based methods for NER, making it more general and retargetable.

3 Algorithm to Identify Potential Rules

In this section we describe the core of the proposed method to learn rules for NER: Section 3.1 first describes the type of patterns learned by the method, and Section 3.2 explains the algorithm to identify the rules that compose those patterns. Section 3.3 lists the features used by the method and how they are represented, and Section 3.4 finally describes how the validated rules are stored prior to the translation to JAPE notation. Afterwards, Section 4 will explain how this algorithm is efficiently combined with clustering and active learning, as well as the generation of the final JAPE patterns.

Throughout the remainder of the paper we will follow the simple scenario in Fig. 2 as the main example for demonstration purposes, though we will explicitly use other examples to illustrate some specific concepts. The set of initial positive examples contains two entities: $\mathcal{E}^+ = \{e_1 = \texttt{4:50␣pm}, e_2 = \texttt{10.55␣am}\}$.

3.1 Sequential Patterns

The most common criterion to recognize textual entities is the occurrence of the same symbol in all entities. For instance, the two entities in Fig. 2.a contain digits and a time expression (`am` or `pm`). The declarative languages UIMA, AQL or

a) Seed Entities

$$e1 = \textit{4:50␣pm} \qquad e_2 = \textit{10.55␣am}$$

b) Feature Matrices

Character Features

$$A_1^{\mathcal{C}} = \begin{bmatrix} \texttt{4} & \texttt{:} & \texttt{5} & \texttt{0} & \text{␣} & \texttt{p} & \texttt{m} \\ \texttt{Nd} & \texttt{Po} & \texttt{Nd} & \texttt{Nd} & \texttt{Zs} & \texttt{[Pp]} & \texttt{[Mm]} \\ \texttt{D} & \texttt{P} & \texttt{D} & \texttt{D} & \texttt{S} & \texttt{L} & \texttt{L} \end{bmatrix} \quad A_2^{\mathcal{C}} = \begin{bmatrix} \texttt{1} & \texttt{0} & \texttt{.} & \texttt{5} & \texttt{5} & \text{␣} & \texttt{a} & \texttt{m} \\ \texttt{Nd} & \texttt{Nd} & \texttt{Po} & \texttt{Nd} & \texttt{Nd} & \texttt{Zs} & \texttt{[Aa]} & \texttt{[Mm]} \\ \texttt{D} & \texttt{D} & \texttt{P} & \texttt{D} & \texttt{D} & \texttt{S} & \texttt{L} & \texttt{L} \end{bmatrix}$$

Token Features

$$A_1^{\mathcal{T}} = \begin{bmatrix} \texttt{<>} & \texttt{<time>} \\ \texttt{<italics>} & \texttt{<italics>} \end{bmatrix} \quad A_2^{\mathcal{T}} = \begin{bmatrix} \texttt{<>} & \texttt{<time>} \\ \texttt{<italics>} & \texttt{<italics>} \end{bmatrix}$$

c) Pattern Matrices

Character Patterns

$$B_1^{\mathcal{C}} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \text{␣} & \bullet & \bullet \\ \texttt{Nd} & \texttt{Po} & \texttt{Nd} & \texttt{Nd} & \bullet & \bullet & \texttt{[Mm]} \\ \bullet & \bullet & \bullet & \bullet & \bullet & \texttt{L} & \bullet \end{bmatrix} \quad B_2^{\mathcal{C}} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \text{␣} & \bullet & \bullet \\ \texttt{Nd} & \texttt{Nd} & \texttt{Po} & \texttt{Nd} & \texttt{Nd} & \bullet & \bullet & \texttt{[Mm]} \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \texttt{L} & \bullet \end{bmatrix}$$

Token Patterns

$$B_1^{\mathcal{T}} = \begin{bmatrix} \bullet & \texttt{<time>} \\ \texttt{<italics>} & \texttt{<italics>} \end{bmatrix} \quad B_2^{\mathcal{T}} = \begin{bmatrix} \bullet & \texttt{<time>} \\ \texttt{<italics>} & \texttt{<italics>} \end{bmatrix}$$

d) JAPE Pattern

```
Phase: char_phase
Input: Token SpaceToken
Rule: C_1
 ({Token.string ==~
   "\\p{Nd}\\p{Po}\\p{Nd}{2}"}
 ):token1
 ({SpaceToken.string == " "})
 ({Token.string ==~ "\\p{Ll}[Mm]"}
 ):token2
-->
 :token1.T_1_2 = {sT = "1"},
 :token2.T_1_2 = {sT = "2"}

Rule: C_2
 ({Token.string ==~
   "\\p{Nd}{2}\\p{Po}\\p{Nd}{2}"}
 ):token1
 ({SpaceToken.string == " "})
 ({Token.string ==~ "\\p{Ll}[Mm]"}
 ):token2
-->
 :token1.T_1_2 = {sT = "1"},
 :token2.T_1_2 = {sT = "2"}
```

```
Phase: token_phase
Input: Token T_1_2
Rule: T_1_2
 ({T_1_2.sT == "1",
   Token.fT2 == "italics"}
  {T_1_2.sT == "2",
   Token.fT1 == "time",
   Token.fT2 == "italics"}
 ):match
-->
 :match.entity =
  {type = "time_italics"}
```

Fig. 2. Sample application of the pattern learning method to recognize times in italic font: a) the set of initial seeds e_1 and e_2, b) their character feature matrices $A_i^{\mathcal{C}}$ and their token feature matrices $A_i^{\mathcal{T}}$ (see Section 3.3), c) the learned pattern matrices $B_i^{\mathcal{C}}$ and $B_i^{\mathcal{T}}$ for each feature matrix, containing the fixed rules (see Section 3.4), and d) the final JAPE pattern generated from the pattern matrices (see Section 4.3).

JAPE include operators to specify this type of conditions in the elements to analyze (e.g. `CONTAINS`), but patterns for NER are usually much more complex. They can be combined with other conditions to create sequential patterns, which are far more common in Natural Language Processing. These patterns consist of sequences of symbols that repeat in all the items of interest (Srikant and Agrawal 1996). These symbols may appear next to each other or with a number of symbols in between, which may be fixed or vary across items. In our example, if we represent the hour and time expressions as symbols, then they respond to a sequential pattern where those symbols are at a distance of 1 from each other. Note that patterns that rely only on one symbol could also be dealt with as sequences of two symbols, where one of them is the starting or the end symbol.

Markov Models and CRF implicitly model these dependencies among symbols in a sequence. Other statistical models such as SVM or Decision Trees rely on the features to add sequence information to the model (e.g. n-grams, left/right context features). Declarative languages include sequence conditions such as `AFTER` or `BEFORE`. Implicitly, systems based on regular expressions and transducers also search for sequences of symbols with the use of automata.

Examples of rule-based systems that learn sequential patterns are *Whisk* and *Amilcare*, which enumerate possible sequences of symbols and select the best ones based on several statistics. However these techniques may not be scalable in large search spaces, so domain information is usually employed to maintain the problem tractable. For instance, the search space could be pruned, as *Amilcare* does, or constrained with certain conditions as part of a known, customized model. For example, *RAPIER* looks for features that appear in the exact same position in all the entities. In that case, when two entities have the same value for a specific feature (string, POS and semantic) and at the same position, that value gets fixed. Otherwise, all the values appearing in the known entities at that position are accepted by the pattern. Using a customized model rather than a general classification model is specially relevant when, in addition to the scalability problem, we do not have enough data to statistically validate the patterns. In the latter case the user could provide some kind of feedback by means of active learning techniques, as we will see in Section 4.2.2, but the learning method should keep the interaction at a minimum, finding the most relevant patterns to ask based on the model we know.

3.2 Rule Vectors

Our pattern search space is composed of what we will call *rule vectors*. A rule vector is associated with a particular symbol from some feature, and it contains the positions of that symbol in *all* the known entities. In particular, if $\vec{z}$ is a rule vector related to a symbol α, each component $z_i \in \vec{z}$ is the position of α in the i-th known entity. At this point, let us assume that the symbol α is present just once in all entities; this restriction is relaxed in Section 4.2.1.

Example: with the entities in Fig. 2, a possible rule vector would be $(4, 2)$ for symbol `0`, meaning that the fourth character of the first entity and the second character of the second entity are both `0`. That vector suggests fixing those specific symbols in those specific positions as rules of the final pattern. □

We identify sequential patterns by comparing pairs of rule vectors, trying to determine how well they are aligned with each other. For two arbitrary vectors $\vec{u}$ and $\vec{v}$ that contain positions of two symbols α and β (not necessarily from the same feature), the ideal case would be such that $\vec{u} = \vec{v} + \delta$, meaning that all entities contain symbols α and β in the same order and at a distance of δ from each other. However, in practice the distance between sequential symbols is not always the same, so we define the following distance function between two rule vectors:

$$\Delta(\vec{u}, \vec{v}) = \sum_{i} |(v_{i+1} - u_{i+1}) - (v_i - u_i)|$$

Pairs of vectors with small distances are more likely to get fixed in the final pattern. Symbols that always appear at similar positions from the beginning or from the end of the entity can be identified by introducing two additional vectors. The first one, $\vec{z}_b$, contains only zeros to detect patterns with respect to the beginning of entities, that is, symbols that appear at a similar distance from the first symbol in each entity. The second vector, $\vec{z}_e$, contains the length of each entity and allows us to detect patterns with respect to their end, that is, symbols that appear at a similar distance from the last symbol of each entity.

For each feature, we iteratively calculate all rule vectors and compute their distance Δ with each of: i) all other possible rule vectors, ii) vectors $\vec{z}_b$ and $\vec{z}_e$, iii) all vectors already fixed in the final pattern and from the same feature, and iv) all vectors already fixed in the final pattern but from previous features. In the end, we select the rule vectors whose distance with other vectors was minimal in the whole set, as they are most likely to produce valid rules. We note that for two vector rules to actually define a sequential pattern, the symbols they represent must always appear in the same order in all entities. This means that if in any entity the symbols are swapped, the vectors are not even paired to compute their distance.

In addition, it generally does not make sense to fix a rule vector involving for instance the same character found in the body of some entities and in the context of some others. Because we do not force symbols to appear in the exact same position across entities, we look for rule vectors separately within the entities and within their contexts. The identification of rule vectors within the contexts is made by joining together the left and right context, so we are able to find sequential patterns involving matching symbols from both contexts. This approach would help finding symbols that appear sequentially in the left and right context of an entity, such as `<` and `>`, and would also let us identify symbols that could appear indistinctly in the left or right contexts in each entity.

3.3 Feature Matrices

As mentioned, a rule vector contains the positions of a certain symbol in all entities. These symbols are computed with different feature functions, which in our case are either at the character level or at the token level. The set $\mathcal{F}^{\mathcal{C}}$ of character features

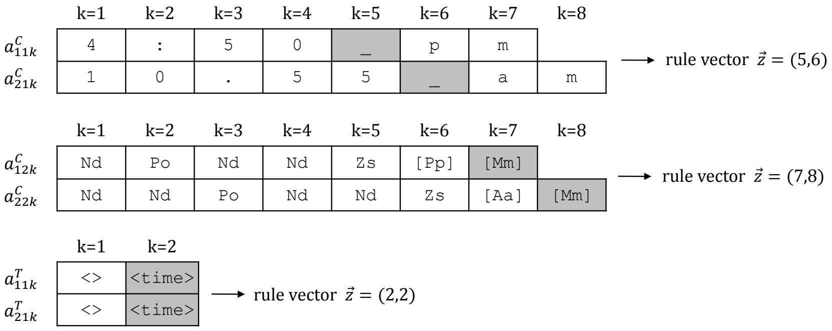
Table 1. *Custom groups of Unicode categories in the third character feature.*

Symbol	Grouped Unicode categories
`U`: Uppercase letter	`Lu`
`L`: Lowercase letter	`Ll`
`D`: Digit	`Nd`
`Y`: Symbol	`Sm, Sc, So`
`S`: Separator	`Cc, Zs`
`P`: Punctuation	`Pc, Pd, Ps, Pe, Pi, Pf, Po`
`O`: Others	rest of Unicode categories

is fixed to three feature functions similar to Brauer (2011). The first feature simply maps a character onto the character itself. The second feature maps characters onto their Unicode category, except for alphabetic characters, which are represented in case-insensitive form. The third one maps characters onto a custom group of Unicode symbols (see Table 1). The method allows the use of an arbitrary set $\mathcal{F}^{\mathcal{T}}$ of features at the token level.

All these features are represented in what we will call *feature matrices* (see Fig. 2.b). The character feature values for an arbitrary entity e_i are represented by a matrix $A_i^{\mathcal{C}}$, such that each row corresponds to a feature and each column corresponds to a character of the entity, that is, $a_{ijk}^{\mathcal{C}} \in A_i^{\mathcal{C}}$ is the value of the j-th feature with the k-th character of entity e_i. Similarly, a matrix $A_i^{\mathcal{T}}$ represents the token feature values of the entity.

Example: In our main example in Fig. 2, the fourth column of $A_2^{\mathcal{C}}$ represents the fourth character of the second entity. Because this character is *5*, the character feature values are $a_{2,1,4}^{\mathcal{C}} = \texttt{5}$, $a_{2,2,4}^{\mathcal{C}} = \texttt{Nd}$ and $a_{2,3,4}^{\mathcal{C}} = \texttt{D}$. As for tokens, the example uses two features: $f_1^{\mathcal{T}}$ assigns a semantic tag from a gazetteer, and $f_2^{\mathcal{T}}$ assigns a font style tag. Similarly, the second token of the second entity is *am*, which is represented as $a_{2,1,2}^{\mathcal{T}} = \texttt{<time>}$ and $a_{2,2,2}^{\mathcal{T}} = \texttt{<italics>}$ in the token feature matrix $A_2^{\mathcal{T}}$. From these feature matrices we may identify rule vectors. For example, with the first feature we may identify rule vector $(5, 6)$ for the space symbol ␣, and with the second feature we could identify vector $(7, 8)$ for symbol [`Mm`], among others. With the first token feature we would have vector $(2, 2)$ for symbol `<time>`.



	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8
$a^{\mathcal{C}}_{11k}$	4	:	5	0	_	p	m	
$a^{\mathcal{C}}_{21k}$	1	0	.	5	5	_	a	m

rule vector $\vec{z} = (5,6)$

	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8
$a^{\mathcal{C}}_{12k}$	Nd	Po	Nd	Nd	Zs	[Pp]	[Mm]	
$a^{\mathcal{C}}_{22k}$	Nd	Nd	Po	Nd	Nd	Zs	[Aa]	[Mm]

rule vector $\vec{z} = (7,8)$

	k=1	k=2
$a^{\mathcal{T}}_{11k}$	<>	<time>
$a^{\mathcal{T}}_{21k}$	<>	<time>

rule vector $\vec{z} = (2,2)$

□

In the initialization step of the learning method, we first compute all character feature matrices of the known entities, and group them together in a set $\mathcal{A}^{\mathcal{C}} = \{A^{\mathcal{C}}_1, A^{\mathcal{C}}_2, \dots\}$. In order to detect patterns involving the context of the entities as well, we also include in these matrices a fixed number of characters to the left and to the right of the entities; let these be the contexts for learning. In particular, we use a window of size θ_{wc} characters on each side, so the k indexes have an offset of θ_{wc}. The set $\mathcal{A}^{\mathcal{T}}$ with all token feature matrices is created similarly, but with an offset window of θ_{wt} tokens. In Fig. 2 we have set $\theta_{wc} = \theta_{wt} = 0$ for simplicity and therefore do not use any context in that example.

3.4 Pattern Matrices

Similarly to the feature matrices, we also use two *pattern matrices* for each known entity, in order to save the rule vectors that have been fixed during the process as part of the patterns. For each known entity e_i, matrix $B^{\mathcal{C}}_i$ will contain the symbols that have been fixed from the character features, and matrix $B^{\mathcal{T}}_i$ will contain the symbols fixed from the token features. Each component of these matrices corresponds to a rule vector learned for a particular feature and character (or token) of the entity. For instance, $b^{\mathcal{C}}_{ijk} = \alpha \in \Sigma^{\mathcal{C}}_j$ means that a rule was fixed for the k-th character of the i-th entity, forcing it to be α with the j-th character feature function. If no rule is found for some particular character (or token) and feature, the corresponding component is set to the wildcard $\bullet$.

Example: Fig. 2.c shows the character pattern matrices $B^{\mathcal{C}}_1$ and $B^{\mathcal{C}}_2$ in our main example. Row 1 indicates that the fifth character must be a space ␣ in positions 5 for e_1 and 6 for e_2, as a result of fixing rule vector $(5,6)$ in the previous example. The second row contains the values fixed by the rules found with the second feature, such as $(7,8)$. In particular, they indicate that we have a sequence of one or two digits (`Nd`), followed by a punctuation mark (`Po`), two more digits, and the character `M` in upper or lowercase at the end. Regarding the token features, we have that in both cases the second token of the entity must evaluate to `<time>` and `<italics>` with features one and two, respectively. The pattern also indicates that there is no specific rule for the first token with the first feature. □

After creating sets $\mathcal{A}^{\mathcal{C}}$ and $\mathcal{A}^{\mathcal{T}}$ with the feature matrices, the method also creates the set $\mathcal{B}^{\mathcal{C}}$ containing all character patterns, and then the set $\mathcal{B}^{\mathcal{T}}$ containing all token patterns. At that point, the patterns just contain $\bullet$ everywhere, meaning that no specific rule has been found yet. Learning of patterns can thus be seen here as the specialization of a wildcard matrix by means of rules that fix specific symbols from the feature matrices. In the end these pattern matrices will be converted to regular expressions or JAPE rules as detailed in Section 4.3.

4 Method to Learn Rules via Active Learning

A challenge we find to identify rules is that entities of the same type often respond to quite different patterns, and therefore trying to learn just one pattern for all entities is not effective. Instead, the method iteratively clusters entities by similarity (see Section 4.1), and learns new rules within each cluster by a call to procedure *LearnRules* (see Section 4.2). Some of those rules get fixed in the corresponding pattern matrices via active learning: different sets of potential entities are presented to the user for validation, and depending on the annotations the method decides whether the rule vectors should be fixed and thus become part of the pattern matrices (see Section 4.2.2). This process of entity clustering, rule identification and active learning is repeated with each character feature in $\mathcal{F}^{\mathcal{C}}$. After fixing in $\mathcal{B}^{\mathcal{C}}$ all rules found at the character level, we then proceed in the same way with each feature in $\mathcal{F}^{\mathcal{T}}$ to find rules at the token level and fix them in $\mathcal{B}^{\mathcal{T}}$. In the end, the method generates a pattern $\hat{\mathsf{P}}$ to recognize the entities of interest by calling function *JAPEGeneration* (see Section 4.3). At a very high level, the method thus works as follows:

Algorithm 1 Rule Learning Method

```
for all feature $f_j \in \mathcal{F}^{\mathcal{C}}$ do                    ▷ Iterate character features
    for all cluster $X \in \mathit{Clustering}(\mathcal{A}^{\mathcal{C}}, j, \theta_X)$ do
        $\mathit{LearnRules}(\mathcal{A}^{\mathcal{C}}, \mathcal{B}^{\mathcal{C}}, X, j, \theta_{wc})$
    end for
end for
for all feature $f_j \in \mathcal{F}^{\mathcal{T}}$ do                    ▷ Iterate token features
    for all cluster $X \in \mathit{Clustering}(\mathcal{A}^{\mathcal{T}}, j, \theta_X)$ do
        $\mathit{LearnRules}(\mathcal{A}^{\mathcal{T}}, \mathcal{B}^{\mathcal{T}}, X, j, \theta_{wt})$
    end for
end for
$\hat{\mathsf{P}} \leftarrow \mathit{JAPEGeneration}()$
```

In the following sections we discuss the clustering process, an efficient implementation of the rule identification algorithm from Section 3, active learning, and the generation of the final JAPE pattern.

4.1 Clustering

As mentioned above, we can not just simply look for rules that are met by all known entities (i.e. same sequence of symbols), because the resulting pattern would tend to generalize too much in order to recognize them all. The upside of identifying a single pattern is that we can maximize recall, but at the cost of reducing precision. Let us consider for instance the example of IP addresses with an initial set of entities following both the IPv6 format (8 groups of 4 hexadecimal digits separated by colons, such as `2001:0DBB:AC10:FE01:FE80:8432:4F71:8BA4`) and the IPv4 format (4 groups of numbers from 0 to 255 separated by dots, such as `192.168.1.10`). A single pattern could be a regular expression recognizing a sequence of 4 to 8 groups of 1 to 4 hexadecimal digits, all separated by a punctuation symbol (`[LD]{1,4}(P[LD]{1,4}){3,7}`). However, this pattern happens to recognize not only IPs but also MAC addresses (6 groups of 2 hexadecimal digits separated by colons, such as `00:1E:8C:33:82:B4`).

To avoid this situation and find a balance, for each feature to process we run a clustering algorithm with the row of the feature matrices corresponding to the current feature. Similar to (Nagesh and Chiticariu 2012), we use a complete-linkage agglomerative hierarchical clustering algorithm (Day and Edelsbrunner 1984), with the Levenshtein edit distance normalized by the entity length (Levenshtein 1966). Two entities will be clustered together only if their distance is smaller than a customizable threshold θ_X. The method will look for rule vectors within each cluster separately, leading to different patterns for the same entity type.

Example: in our main example in Fig. 2.b, the Levenshtein distance between the two entities is 5, 2 and 1 with the three character features respectively. Normalizing by entity length, they turn into 0.71, 0.29 and 0.14. If the clustering threshold is for instance set to $\theta_X = 0.3$, these entities would end up in the same cluster only with the second and third character features. With the token features they would be clustered together, because the distance is 0 in both cases. □

In our learning method, a cluster thus contains the indices of a subset of entities that are likely to produce new rules with the current feature (e.g. a cluster $X = \{x_1 = 4, x_2 = 5, x_3 = 9\}$ groups together entities e_4, e_5 and e_9).

4.2 Learning Rules

The procedure *LearnRules* is used to learn new rules within a given cluster and with a given feature, at the character or token level, indistinctly. First, it creates sets $\mathcal{A}$ and $\mathcal{B}$ by subsetting the entire feature and pattern matrix sets with the entities in the cluster X; here is where rules will be looked for. It iteratively finds new potential rules within the body of the entities and within their context, and then it runs the active learning phase to decide which are fixed. In each case, a set $\mathcal{Z}^{\mathcal{A}}$ of potential rule vectors is identified by calling function *FindRuleVectors* with the feature matrices (see Section 4.2.1). Then, a set $\mathcal{Z}^{\mathcal{B}}$ of previously fixed rules is similarly created from the pattern matrices. Finally, a set $\mathcal{Z}_{bdy}$ is created by function *RuleVectorsWithMin*Δ, which selects from $\mathcal{Z}^{\mathcal{A}}$ those vectors which

Algorithm 2 Pseudocode of the *LearnRules* procedure.

```
1: procedure $LearnRules(X, j, \theta)$
       $X$ : cluster
       $j$ : feature index
       $\theta$ : length of context window
2:   $\mathcal{A} \leftarrow \{A_i\}_{i \in X}$                  ▷ Subset of feature matrices from the cluster
3:   $\mathcal{B} \leftarrow \{B_i\}_{i \in X}$                  ▷ Subset of pattern matrices from the cluster
4:   $n \leftarrow |X|$
5:   $\mathcal{Z}' \leftarrow \emptyset$
6:   repeat
7:                                                  ▷ Get vector rules within bodies
8:     $\mathcal{Z}^{\mathcal{A}} \leftarrow FindRuleVectors(\mathcal{A}, j, \theta)$    ▷ Vectors with current feature
9:     $\mathcal{Z}^{\mathcal{B}} \leftarrow FindRuleVectors(\mathcal{B}, j, \theta)$    ▷ Vectors already fixed
10:    $\mathcal{Z}_{bdy} \leftarrow RuleVectorsWithMin\Delta(\mathcal{Z}^{\mathcal{A}}, \mathcal{Z}^{\mathcal{B}}, \vec{z}_b, \vec{z}_e)$ ▷ Select rules which minimize $\Delta$
11:
12:                                          ▷ Similarly, get vector rules within contexts
13:    $\mathcal{Z}^{\mathcal{A}} \leftarrow FindRuleVectors(\mathcal{A}, j, \theta)$
14:    $\mathcal{Z}^{\mathcal{B}} \leftarrow FindRuleVectors(\mathcal{B}, j, \theta)$
15:    $\mathcal{Z}_{ctx} \leftarrow RuleVectorsWithMin\Delta(\mathcal{Z}^{\mathcal{A}}, \mathcal{Z}^{\mathcal{B}}, \vec{z}_b, \vec{z}_e)$
16:
17:    $\mathcal{Z} \leftarrow \mathcal{Z}_{bdy} \cup \mathcal{Z}_{ctx}$
18:    if $\exists \vec{z} \in \mathcal{Z} : (z_1, \ldots, z_n) \notin \mathcal{Z}'$ then
19:      $ActiveLearning(\mathcal{Z}, \mathcal{A}, \mathcal{B}, X, j)$
20:    else
21:      $FixRules(\mathcal{Z}, \mathcal{A}, \mathcal{B}, j)$
22:    end if
23:    $n \leftarrow |X|$
24:    $\mathcal{Z}' \leftarrow \mathcal{Z}$
25:  until $\mathcal{Z}' = \emptyset$
26: end procedure
```

minimize the distance function Δ with other rule vectors. Note that this process is run separately for the body of the entities and their context, finally obtaining a set $\mathcal{Z}$ with all potential rules to validate.

Once $\mathcal{Z}$ is computed, we check whether any of the potential rule vectors has not been previously identified in the cluster (line 18). In that case, the user is asked to validate new entities selected by the *ActiveLearning* procedure (see Section 4.2.2). If with the new entities just validated the same rule vectors (or a subset of them) are identified again in the next iteration, the rules they contain get fixed in the patterns (line 21). When called, procedure *FixRules* adds to the pattern matrices in $\mathcal{B}$ all the new rules just found by fixing the symbols identified in each rule vector. In particular, for each rule $\vec{z} \in \mathcal{Z}$, it sets $B_{ijz_i} \leftarrow A_{ijz_i}$ (i.e. $A^{\mathcal{C}}_{x_ijz_i}$ when working with character features, or $A^{\mathcal{T}}_{x_ijz_i}$ with tokens). The procedure ends when no new rule vectors are found (line 25).

We note that the cluster is augmented when the user validates new entities during the active learning process. Therefore, the rule vectors found in some iteration may involve new entities that were not available in the previous iteration, so when comparing sets of rule vectors between successive iterations we must disregard the new entities just found. This is achieved with variable n, which tracks the size of the cluster: only the first n components of the rule vectors are compared in line 18, that is, only the first n entities in the cluster.

Example: in our example from Fig. 2.b, let us assume that the vector $(3, 4)$ is identified as a potential rule vector for symbol `5` in the first character feature. During the active learning process, the user validates a new entity $e_3 =$ `11.30␣PM`, so in the next iteration there is no rule vector involving symbol `5` anymore, because it is not contained in the new entity. Thus, the symbol `5` is not fixed. However, with the second character feature the potential rule vector $(7, 8, 8)$ is found for symbol `[Mm]`. During the active learning process the user validates new entities, all of which contain an `m` or an `M` in the last position, so the rule gets eventually fixed (see Fig. 2.c). □

At the end, the same *LearnRules* procedure is called one feature and cluster at a time, treating feature and pattern matrices in the same way, whether they represent character or token features. The only difference is that in the case of characters, if some rule is already fixed at position k, there is no point in trying to identify more rules in that same position with the next features, because features at the character level follow a hierarchy (e.g. a `3` with the first feature implies `Nd` and `D` with the second and third features). This inefficient and redundant situation is avoided after learning rules with the current feature by setting to $\bullet$ all symbols below that position in $A_i^{\mathcal{C}}$. This is not done in the token patterns because token features do not necessarily follow a hierarchy.

In the next subsections we explain in detail function *FindRuleVectors* and procedure *ActiveLearning*.

4.2.1 Finding Rule Vectors

We assumed in Section 3.2 the ideal case where a symbol can occur only once in each entity, but this is seldom true in practice. We may actually consider several rule vectors for the same symbol, each containing different combinations of symbol occurrences within the same entity. The problem of doing this is that the number of vectors for each symbol grows exponentially with the number of entities in the cluster. In general, with m being the length of the longest entity, a symbol α is repeated $O(m)$ times, and thus the number of vectors generated for α is $O(m^{|X|})$. Through the learning process we extend the cluster with new entities, so it could become large enough to make the algorithm impractical at some point. In order to reduce the complexity, we apply a heuristic that prioritizes the selection of symbol occurrences whose positions are closer to each other across entities. Function *FindRuleVectors* implements this heuristic to find new potential rule vectors within an arbitrary set of matrices $\mathcal{H}$, be them characters or tokens, entities or contexts, features or patterns (in the latter case, to get previously fixed rule vectors).

Since we only find rule vectors for symbols that appear in all entities, we can just iterate the unique symbols in the shortest entity of the cluster (i.e. with fewest columns in its matrix). If the symbol appears in all entities, we gather a set with

Algorithm 3 Pseudocode of the *FindRuleVectors* function.

```
function FindRuleVectors($\mathcal{H}, j$)
    $\mathcal{H}$ : matrices of the cluster (entities or contexts)
    $j$ : feature index
  $H^* \leftarrow \arg\min_{H_i \in \mathcal{H}} \mathrm{cols}(H_i)$                ▷ Shortest entity
  $\mathcal{Z} \leftarrow \emptyset$
  for all symbol $\alpha \in \bigcup_k \left(h^*_{jk}\right)$ do
    if $\alpha \neq \bullet$ and $\forall H_i \in \mathcal{H} :\ \exists k :\ h_{ijk} = \alpha$ then
      $\mathcal{K} \leftarrow \bigcup_{H_i \in \mathcal{H}} \{k \mid h_{ijk} = \alpha\}$        ▷ Positions with $\alpha$
      for all position $k \in \mathcal{K}$ do
        $\vec{z}^L \leftarrow \left(z_i^L = \min\left(\arg\min_{\{k' \mid h_{ijk'} = \alpha\}} |k - k'|\right)\right)$
        $\vec{z}^R \leftarrow \left(z_i^R = \max\left(\arg\min_{\{k' \mid h_{ijk'} = \alpha\}} |k - k'|\right)\right)$
        $\mathcal{Z} \leftarrow \mathcal{Z} \cup \left\{\vec{z}^L, \vec{z}^R\right\}$
      end for
    end if
  end for
  return $\mathcal{Z}$
end function
```

all its different positions across entities; note that if a symbol is equal to $\bullet$ we can skip it because it was marked as redundant during the character learning step (see Section 4.2). For each position k (note that we have $O(m)$ such positions), the algorithm computes the rule vector closest to k across entities. However, in some entities the symbol might appear twice and at the same distance from k (i.e. once on each side). We thus generate vectors $\vec{z}^L$ and $\vec{z}^R$ to keep the positions on the left and right sides of k, respectively.

Example: let us consider the following example where the j-th rows of the feature matrices contain the following symbols ($-$ represents an arbitrary symbol here):

	k=1	k=2	k=3	k=4	k=5	k=6
h_{1jk}	a	–	a	–	a	
h_{2jk}	a	–	a	–	a	
h_{3jk}	a	–	–	a		
h_{4jk}	a	–	–	–	a	–
h_{5jk}	a	–	–	–	a	
h_{6jk}	a	–	–	a		

The unique positions of symbol `a` are $\mathcal{K} = \{1, 3, 4, 5\}$. The rule vector closest to $k = 1$ is simply $(1, 1, 1, 1, 1, 1)$. Entities 3 to 6 do not contain symbol `a` at position $k = 3$, so we take the position that is closest, and when there is a tie we just take the positions closest from the left and from the right sides. The vector closest from the left side is $(3, 3, 4, 1, 1, 4)$, and the one closest from the right side is $(3, 3, 4, 5, 5, 4)$. The latter is also the vector from the left side for $k = 4$. The vector from the right side is $(5, 5, 4, 5, 5, 4)$, which turns out to be the vector from the left for $k = 5$. □

For each of the $O(m)$ positions of a symbol, *FindRuleVectors* returns two vectors at most (one from the left side and one from the right side), *regardless* of the number of entities in the cluster. The number of vectors returned is thus reduced from $O(m^{|X|})$ to $O(m)$. The time to generate them is $O(|X| \cdot m^2)$ because we compute $O(m)$ different vectors of $|X|$ dimensions, for each of which we compare $O(m)$ positions of the symbol. Therefore, the space complexity is independent of the number of entities, and the time complexity is linear with respect to it.

4.2.2 Active Learning

An active learning process is followed to validate the set of potential rule vectors $\mathcal{Z}$ identified during the learning step. We first generate a temporary JAPE pattern with procedure *JAPEGeneration* for each entity in the cluster, using both its character and token pattern matrices. They are used to gather a set $\mathcal{M}$ of text fragments from the unannotated corpus $\mathcal{D}$ that match the patterns as of this point, some of which will probably be entities and some of which will not. Until we have at least $\theta_{\mathcal{M}}$ such matches, we randomly select a document from the corpus and include all matches from the document that comply with any of the temporary JAPE patterns. From these matches we select two sets $\mathcal{P}$ and $\mathcal{R}$ that will allow us to increase the precision and recall of the patterns, respectively. With $\mathcal{P}$ we try to avoid fixing rules that may be wrong by finding at least one valid entity as counter-example, while with $\mathcal{R}$ we try to extend the set of valid rules by finding new valid entities that are different.

For precision, until we have at least $\theta_{\mathcal{P}}$ matches, we iterate each of the rule vectors we are validating and select the top θ_z matches from $\mathcal{M}$ that do *not* comply with it. To this end, these matches are previously ordered by their similarity with the entities we already know. The sorting criterion is the sum of two similarity functions: the first one counts the number of symbols that appear in both the match and its corresponding feature matrix, and the second one counts the number of potential rule vectors with which the match complies. These matches are then added to $\mathcal{P}$ and removed from $\mathcal{M}$. In the end, $\mathcal{P}$ contains matches that are very similar to the known entities in the cluster and comply with as many of the rule vectors we are validating except one.

For recall, we simply select the top $\theta_{\mathcal{R}}$ remaining matches from $\mathcal{M}$, sorted by the first function, that is, the matches that are the most similar to the entities we know. Note that the matches do not necessarily have the same length as the entities they come from, because they are only required to comply with the rules fixed so far. They are then sorted by their similarity with the original entities, so in the end we tend to select the matches whose length is similar but not necessarily the same.

Finally, all selected matches are presented to the user for validation, along with θ_{wv} characters to the left and right of the text fragments so that the user can correct the boundaries of a partially recognized entity and facilitate the validation task. Note that this context for validation is different from the contexts used to learn (θ_{wc} and θ_{wt}). All matches that are valid entities are added to sets $\mathcal{E}^+, \mathcal{A}^\mathcal{C}, \mathcal{A}^\mathcal{T}, \mathcal{B}^\mathcal{C}, \mathcal{B}^\mathcal{T}$ and the current cluster X; matches that are not entities are just added to $\mathcal{E}^-$ and ignored by the method. A valid entity from $\mathcal{P}$ will prevent us from fixing the rule vector it did not comply with, because that rule vector will not come up again in the next iteration within *LearnRules*. The effect of a valid entity from $\mathcal{R}$ is twofold: first, it might result in an even better set of rule vectors in the next iteration; second, it might end up in a different cluster for the next iteration, possibility leading to the creation of an additional branch of patterns for the same type of entities (e.g. different types of IP addresses).

4.3 JAPE Generation

Once all features have been iterated in search for rules, the final estimated pattern $\hat{\mathsf{P}}$ is generated in JAPE notation from the pattern matrices in $\mathcal{B}^{\mathcal{C}}$ and $\mathcal{B}^{\mathcal{T}}$. In order to avoid too general character patterns due to atypical entities, all positions for which no rule was found are set to their corresponding symbols in the second feature. Note that in the case of tokens we can not fix any default symbol because they do not follow a hierarchy like characters do.

The resulting pattern contains two JAPE phases: the first one involves character features, that we have called `char_phase`, and the second involves token features, that we have called `token_phase`. To avoid redundancy, we first group together all entities that share the same token pattern, and generate their JAPE rule in the *token phase*. Next, we group together all entities that share the same pattern with *both* character and token features. Each group will generate one JAPE rule in the `char_phase`, which will tag its tokens so they are recognized by the corresponding token-level JAPE rule in the `token_phase`. Rules in the `char_phase` are regular expressions created by concatenating the symbols fixed in the corresponding character pattern. Rules in the second phase simply concatenate the set of all fixed symbols in the token pattern, forcing all of them to appear for each token.

Example: in our main example in Fig. 2.c, both entities would be grouped together for the token pattern, because $B_1^{\mathcal{T}}$ and $B_2^{\mathcal{T}}$ are equal. This group turns into the JAPE rule `T_1_2`, which expects the first token to evaluate to `<italics>` with the second feature, and the second token to evaluate to `<time>` and `<italics>` with the first and second features, respectively (see Fig. 2.d-right). In terms of characters, our two entities are *not* grouped together because they do not have the same pattern, so in the first phase there is one JAPE rule for each of them. The first rule, `C_1`, matches a token with the regular expression `\p{Nd}\p{Po}\p{Nd}{2}`, followed by a space character and a token with the regular expression `\p{Ll}[Mm]`. The second rule, `C_2`, is the same except that it requires another digit in the first regular expression (see Fig. 2.d-left). □

5 Evaluation

Three different experiments were designed to evaluate the effectiveness and annotation cost of our method. The first and main experiment evaluates its ability to learn JAPE patterns. The second experiment assesses its utility to annotate a new corpus from scratch. The third and last experiment shows its ability to learn regular ex-

Table 2. *Number of entities in the 300 documents of the Jobs Postings Corpus.*

Entity Type	Instances	Entity Type	Instances
ID	300	Platform	686
Title	459	Application	606
Salary	141	Area	981
Company	291	Required years experience	167
Recruiter	325	Desired years experience	43
State	461	Required degree	81
City	639	Desired degree	21
Country	346	Post date	301
Language	849	Phone number	662

pressions. This section describes common evaluation settings, whereas Sections 6, 7 and 8 present the detailed design and results of each experiment.

5.1 Corpus, Features and Performance Measures

To evaluate our method with a varied range of entity types, we used the *Software Jobs Corpus*[1] (Califf 1998). It contains 300 documents in English, of about 313 tokens each, describing software related job postings. There are 17 different types of entities already annotated: *ID*, *title*, *salary*, *company*, *recruiter*, *state*, *city*, *country*, *language*, *platform*, *application*, *area*, *required years of experience*, *desired years of experience*, *required degree*, *desired degree*, and *post date*. For the third experiment, we annotated ourselves all the entities of type *phone number*. Table 2 shows the number of instances per entity in this corpus.

The Jobs corpus is widely used in Information Extraction (Lavelli et al. 2004), because it contains a well-defined and more varied selection of entity types than other corpora. In addition, some of these entities are recognizable with regular expressions, and many of them can be recognized fairly well without gazetteers. This is particularly relevant when evaluating a novel learning method, because the success of information extraction methods can be notably increased by using domain-specific knowledge resources such as gazetteers, dictionaries and taxonomies. Therefore, we explicitly do *not* use such resources in our experiments in order to center the evaluation on the performance of the proposed learning method and not on the performance of a specific set of knowledge resources, whose existence can not be presupposed and whose growth is seldom traceable (Siniakov 2008). Instead, we use a basic set of token features obtained with the *ANNIE* tool in the *GATE* framework, namely string, stem, POS tag, token type (`<word>`, `<number>` or `<punctuation>`) and case. Recall that the set of three character features is fixed in our method (see Section 3.3).

[1] `ftp://ftp.cs.utexas.edu/pub/mooney/ie-data/jobs300data.tar.gz`

Table 3. *Parameters configured for evaluation of the learning method.*

Parameter		Comment
θ_{wc}	$= 1$	Characters in each side of the context for learning.
θ_{wt}	$= 2$	Tokens in each side of the context for learning.
θ_X	$= 0.8$	Dissimilarity threshold for clustering.
$\theta_{\mathcal{M}}$	$= 40$k	Minimum number of random matches for active learning.
θ_z	$= 2$	Minimum number of matches to validate per potential rule.
$\theta_{\mathcal{P}}$	$= 10$	Minimum number of matches to validate for precision.
$\theta_{\mathcal{R}}$	$= 10$	Maximum number of matches to validate for recall.
θ_{wv}	$= 10$	Characters in each side of the context for validation.

In all three experiments we measure the effectiveness of the learning method with Precision, Recall and the F-measure. Because it learns patterns for each entity type separately, we compute the effectiveness for each of those types and then report the average across types (i.e. macro-average). In terms of cost, we follow the standard approach of reporting the number of elements that the user would have to review, which in our case are text fragments to validate as entities or not. Specifically, we report the number of entities and the number of tokens they contain. We include the initial set of seed entities in the annotation cost, as well as the $2 \cdot \theta_{wv}$ characters in the validation contexts of each match, rounded up to the next token, in order to help the user validate potential entities and adjust their boundaries if needed.

The goal in the first and third experiments is to generate effective patterns, so the method will be run on a training corpus to learn the patterns and they will then be evaluated on a test corpus. The goal in the second experiment is to fully annotate a new corpus, so the method will be trained and evaluated on the full corpus. Regarding the initial seeds, random selections of various sizes are made for each entity type separately. Note that the annotations in the training set, except for these initial seeds, are *not* used by the method in any way; they are iteratively discovered by the active learning process.

5.2 Initial Configuration and Baselines

As seen in Sections 3 and 4, our method depends on several parameters that control various aspects of the learning process, such as context windows, thresholds, etc. Table 3 summarizes the values of the parameters used in the three experiments. For details on the selection of these values, the reader is referred to the Appendix.

Given that the objective is to learn JAPE patterns of reasonable effectiveness at a low cost, we compare our learning method with a semi-supervised baseline that maximizes effectiveness while reducing cost, and an unsupervised baseline that minimizes cost. The semi-supervised baseline is a pattern learning method based on Support Vector Machines, adapted to deal with imbalanced training data and combined with active learning techniques (Li et al. 2009). In terms of features, they use the same basic set of token features we use. Their method also contains a post-processing step that discards entities of infrequent length, and chooses the sequence of entity types with the highest probability for a sequence of tokens (e.g. one entity type often preceded by another one). We note that our method can not apply the second correction because it is run *separately* for each entity type, so patterns are independent of other types that might be irrelevant to the user (and hence need not be annotated) or might not appear at all in the target corpus. To our knowledge, theirs are the best results to date of an active learning method, whether rule-based or statistical, applied to the Jobs corpus and without the use of gazetteers or dictionaries.

The unsupervised baseline is a rule-based bootstrapped method part of the Stanford CoreNLP framework (Gupta and Manning 2014). This method builds a dictionary from a set of initial seeds, and then iteratively learns patterns without any user intervention, adding to the dictionary the recognized fragments that ranked best according to a scoring function. In terms of features, they use the lemma and POS tag of the entity and its context, as well as their edit distance and distributional properties in the sets of positive and negative examples. In the end, all text fragments in the corpus that are found in the dictionary are labeled as entities.

In the remainder of the paper, we will refer to our method as RB-AL (for Rule-Based Active Learning), to the semi-supervised method of Li et.al. (2009) as SVM-AL (for SVM Active Learning), and to the unsupervised method of Gupta and Manning (2014) as BOOT (for Bootstrap).

6 Experiment 1: Generation of JAPE Patterns

In this experiment we evaluate the ability of our method to learn JAPE patterns and evaluate the trade-off between learning cost and effectiveness on a previously unseen corpus.

6.1 Design

We randomly split the full corpus in an unlabeled training set (two thirds) and a test set (one third) that remains unseen by the algorithm; results are reported over five such splits. In terms of initial seeds, we run the method with random sets of 2, 5, 10, 20, 50 and 100 entities from the training set and for each of the 17 entity types separately. Finally, we run 10 random trials in each case, leading to a total of 5,100 trials. Note that the annotations in the training set, except for the seeds, are *not* used by the method in any way; they are iteratively discovered by the active learning process.

For the assessment of effectiveness, we force the patterns to recognize the *exact* boundaries of the entities in order to be considered correct; if there is any discrepancy whatsoever the identification of the entity is considered incorrect regardless of how close it is to the true boundaries of the entity. For the assessment of cost, we report the total number of text fragments validated by the user during active learn-

Table 4. *Effectiveness and annotation cost (mean±95% CI) per entity type of the JAPE patterns learned by* RB-AL.

Seeds	2	5	10
F-score	0.44 ± 0.02	0.53 ± 0.02	0.57 ± 0.02
Precision	0.74 ± 0.02	0.71 ± 0.02	0.68 ± 0.02
Recall	0.35 ± 0.02	0.50 ± 0.02	0.57 ± 0.02
Cost (entities)	83 ± 4	157 ± 7	221 ± 9
Cost (tokens)	284 ± 16	603 ± 31	860 ± 40
Seeds	**20**	**50**	**100**
F-score	0.59 ± 0.02	0.59 ± 0.02	0.60 ± 0.02
Precision	0.64 ± 0.02	0.61 ± 0.02	0.60 ± 0.02
Recall	0.63 ± 0.02	0.68 ± 0.02	0.70 ± 0.01
Cost (entities)	295 ± 12	420 ± 19	532 ± 27
Cost (tokens)	1,172 ± 57	1,697 ± 88	2,166 ± 120

ing and the number of tokens they contained, including their validation context. Recall that the seeds are included in these costs.

In the case of the SVM-AL baseline, we compare with the results reported by Li et.al. (2009), though we must note two differences. First, they always use two fully annotated documents as seeds, so that all unannotated tokens are used as negative examples. Second, they train with the full corpus, using to test whatever (variable) portion is left without user validation. For the BOOT baseline, we ran their code with the same corpus splits and random sets of seeds as our method, for a total of 5,100 trials as well. Note that, because it is unsupervised, its annotation cost is due only to the seeds.

6.2 Results

Table 4 reports the average effectiveness scores of RB-AL over all the 5,100 trials. As expected, larger sets of initial seeds highly improve recall because we start with a larger variety of potential rules, but at the cost of more annotations because they need to be validated. On the other hand, precision decreases, but the average F-score still improves. It ranges between 0.44 (2 seeds) and 0.60 (100 seeds), with mean precision rates above 0.6 in all cases. The total annotation cost ranges between 83 and 532 entities, including seeds.

Fig. 3 shows the effectiveness-cost trade-off of our method and the two baselines. To properly compare our results with SVM-AL, we sum the annotation costs obtained independently for all 17 entity types found in the corpus. They report annotation costs using different annotation units in the active learning process: full

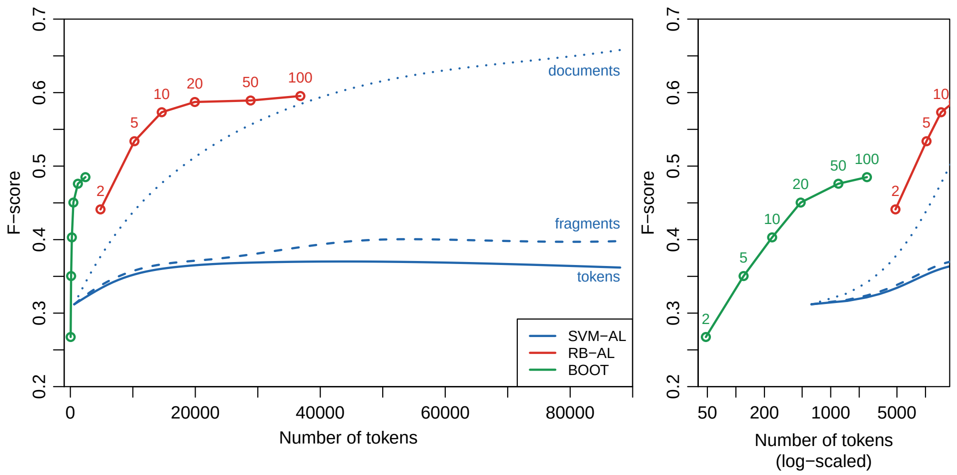


Fig. 3. Mean F-score by total cost (sum over all entity types) of Li et al.'s method (SVM-AL), Gupta and Manning's method (BOOT) and ours (RB-AL). Circles mark the means with 2, 5, 10, 20, 50 and 100 initial seeds, respectively. On left, 0 to 90,000 tokens. On right, 0 to 20,000 tokens (log-scaled), expanded to show detail of BOOT.

documents, 11-token fragments and single tokens. The left plot shows that even the average 0.44 F-score obtained with just 2 seeds outperforms their method when using text fragments and tokens as annotation unit, where their best results are approximately 0.41 and 0.37, respectively. Compared to their use of fragments, for the same annotation effort our method improves effectiveness between +35% (2 seeds) and +54% (100 seeds). Compared to their use of tokens, the relative improvement ranges between +35% (2 seeds) and +62% (100 seeds).

In the case where they use full documents as annotation unit, we see that our method still requires fewer annotations to achieve the same F-score. For instance, they need about 32,500 tokens to reach a score of 0.57, while our method requires 10 initial seeds and just 14,620 tokens (about 47 full documents), which is a −55% improvement in annotation effort. Even if we assumed that the user needs to annotate a full document in order to identify each seed, the approximate total cost would still be 57 documents, that is, a −45% improvement. On the other hand, while the average F-score of our method seems to converge to 0.60, the maximum achieved by SVM-AL is slightly above 0.65. However, in order to reach this point they require the annotation of about 78,000 tokens, or 83% of the corpus. Setting aside the great annotation cost, we note that in our case the training set comprises 67% of the full corpus, so we do not have access to that many documents to learn patterns in the first place.

Fig. 3 also plots the results for the BOOT baseline, showing that effectiveness also increases with the number of seeds. The mean F-scores start at 0.27 with 2 seeds, and similarly seem to converge to almost 0.50 with 100 seeds (we executed additional runs with 200 seeds and the improvement was of only +0.001). In terms of cost, it requires only the annotation of the initial seeds, which correspond to an average of 48 tokens (2 seeds) to 2,400 tokens (100 seeds).

In general, the figure evidences a clear trend in the effectiveness-cost trade-off. The unsupervised BOOT method is able to achieve mid-low F-scores for an exceptionally low cost, while on the other side of the spectrum we see the semi-supervised SVM-AL baseline achieving mid-high scores for a much larger annotation cost (they report and F-score of 0.81 in supervised mode). In the middle, our RB-AL method is able to reach medium effectiveness for a reasonably low annotation effort. In contrast to SVM-AL, an additional advantage of our method is that the learned patterns are human-readable (see Section 9.3). On the other hand, BOOT does generate human-readable patterns, but at the core of their method is the dictionary containing the seeds and the text fragments that were captured by these patterns and ranked best by the scoring function (Gupta and Manning 2014). The evaluation of BOOT using *only* the patterns yields an average F-score between 0.12 and 0.13, evidencing that, without the accompanying dictionaries, the patterns are not very effective on their own.

7 Experiment 2: Corpus Annotation

The previous experiment showed that the proposed method can learn patterns that adapt well to unseen parts of the corpus. Nonetheless, even patterns that do not generalize to other corpora or unseen parts of the same corpus can be of interest if they allow us to reduce the cost of editorial annotation. This is especially useful in the face of a brand new corpus or a new entity type that needs to be annotated as part of the creation of resources for the evaluation of IE algorithms. This second experiment evaluates the utility of our method in this scenario.

7.1 Design

This experiment is run in two phases. First, the method is given the initial seeds to learn patterns, which are then run over the whole corpus. Second, the user would have to review all text fragments matched by the patterns to ensure that they are correct. In this case we do not need separate training and test sets, because we actually want to annotate all documents in the corpus. As in the previous experiment, we use random sets of 2, 5, 10, 20, 50 and 100 entities for each of the 17 entity types separately. This is replicated 10 times, leading to a total of 1,020 trials. Again, the annotations in the corpus, except for the seeds, are *not* used by the method in any way.

Our goal this time is a high recall rate over all entities of the corpus. We thus report two annotation costs: the cost required to learn the patterns (learning cost), which includes the seeds, and the cost required to review the fragments captured by those patterns (review cost). Because of this review process, here we will also consider a partial match as correct because it will be reviewed, and possibly corrected, by the user anyway. For completeness, we still report the precision and F-scores obtained by the learning algorithm *before* the review process. Note that at the end of the review process the recall is the same, but the practical precision is actually 100%, that is, there will not be false positives.

For the BOOT baseline, we ran the code by Gupta and Manning (2014) with the same random sets of seeds as our method. The learning cost of BOOT is still due only to the seeds, but in this experiment there is also a review cost because the user would similarly check all labeled fragments. In this case we do not include the SVM-AL baseline because the experiments in Li et.al. (2009) do not provide information to compare with.

7.2 Results

Table 5 shows the effectiveness of our method and both the learning and review costs. Accounting for both costs we observe that on average just with 5 seeds we can already discover 61% of the entities of some type, with a total cost of about 600 entities, approximately equivalent to 2.5% of the corpus. With 50 initial seeds we can identify almost 90% of the entities, at the cost of reviewing slightly over 5% of the corpus. Between 50 and 100 seeds would be needed in order to identify more than 90% of them. As indicated by the precision scores, we finally remark that a large fraction of the matched fragments are indeed valid entities, and virtually all of these have correct boundaries. Even though they do not require any action from the user, we have still included them as part of the review cost.

In contrast, the BOOT baseline achieves a recall between 0.28 (2 seeds) and 0.65 (100 seeds), with total costs between approximately 190 and 500 entities; the mean precision across seeds remains nearly constant and at 0.50. In general, our method achieves better recall rates, but it also requires more annotation effort. However, this is expected in this experiment, because there is a correlation between recall and review cost: if we identify more entities, the user will still have to review them. Therefore, a better way to assess cost in this experiment is to consider only the false positives that the user would have to mark as such, during learning or reviewing, because they represent user effort spent in vain.

Fig. 4 shows somewhat similar trade-offs between recall and number of false positives. Still, BOOT has fewer false positives than RB-AL for the same recall rate. However, BOOT is clearly limited as to the maximum achievable recall, while our method is able to successfully identify nearly all the entities in the corpus. In order to reach this level, BOOT would apparently need several hundred seeds, which is effectively the same as manually annotating this corpus.

8 Experiment 3: Generation of Regular Expressions

There are two main reasons why the use of character features is interesting. First, and unlike some token features such as the POS, they can always be computed regardless of how much context there is available, allowing the user to obtain the seeds from a gazetteer or simply to make them up. Second, the resulting patterns are simply regular expressions, which is very appealing on its own because of their wide use. Therefore, in this section we briefly evaluate the capability of our method

Table 5. *Effectiveness and cost (mean±95% CI) when annotating an entity type in a new corpus using RB-AL.*

Seeds	2	5	10	20	50	100
F-score	0.50 ± 0.04	0.64 ± 0.04	0.67 ± 0.04	0.71 ± 0.04	0.71 ± 0.04	0.73 ± 0.04
Precision	0.83 ± 0.04	0.78 ± 0.05	0.73 ± 0.05	0.71 ± 0.05	0.66 ± 0.05	0.65 ± 0.05
Recall	0.43 ± 0.04	0.61 ± 0.04	0.71 ± 0.03	0.79 ± 0.03	0.88 ± 0.02	0.92 ± 0.01
Learning Cost (validated entities)	105 ± 13	187 ± 17	261 ± 24	344 ± 32	501 ± 53	631 ± 67
Review Cost (true positives, exact)	147 ± 22	204 ± 22	235 ± 23	251 ± 24	266 ± 28	246 ± 29
Review Cost (true positives, inexact)	4 ± 1	5 ± 1	7 ± 2	8 ± 2	8 ± 1	7 ± 1
Review Cost (false positives)	115 ± 55	219 ± 91	314 ± 105	319 ± 91	517 ± 146	522 ± 141
Total Cost (% corpus)	1.39 ± 0.30	2.46 ± 0.47	3.27 ± 0.57	3.72 ± 0.57	5.33 ± 0.91	5.87 ± 0.97

to generate this kind of patterns using seeds that may not be present in the training corpus.

8.1 Design

The target entity types must be exclusively recognizable with character features, so we only consider the *post date* of the job offer, and the contact *phone* and fax numbers. As usual, we run the method separately for each entity type, and the full corpus is again randomly split in two thirds for training and one third for testing; results are reported over five such splits. We always use 3 randomly selected seeds, either all found in the corpus (internal) or none (external). In the latter case, they are randomly chosen from the ones listed in Table 6; note that some of these follow a pattern that does not match any entity in the corpus. Finally, we run 10 random trials in each case, leading to a total of 200 trials.

We require the patterns to recognize the *exact* boundaries of the entities. For the assessment of cost, we report the total number of text fragments and the corresponding number of tokens validated during learning, including the validation contexts and the seeds.

Neither the SVM-AL nor the BOOT baselines can work exclusively with character

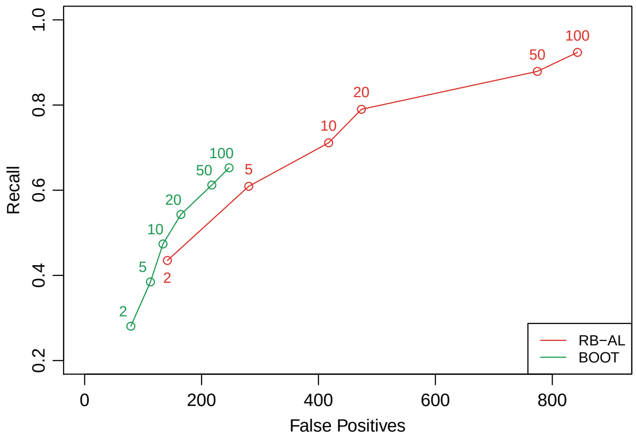


Fig. 4. Mean recall by false positives when annotating an entity type in a new corpus. Circles mark the means with different seed set sizes.

Table 6. *Entities used as external seeds in the third experiment; none of them are found in the corpus. Entities in the last two columns do not correspond to the same pattern as the entities in the corpus (e.g. months in post date entities always contain three letters, the first one in uppercase).*

	Same pattern	Different pattern	

Phone	`(821)444-8241`
	`(452)954-3377`
	`91-624-9115`
		`5554789632`
		`922 5554189`
		`698/458/2148`
		`958-523-ASCI`
			`916249-1155`
			`632 458 666 23`
			`632/945 7788`

Post date	`25 Feb 75`
	`14 May 85`
	`Apr 2010`
		`28 July 2000`
		`15 August 1989`
		`18 september 2020`
		`7 March 1977`
			`14 january 2033`
			`1 jun 85`
			`9 oct 2001`

features, and in particular BOOT does not allow the use of external seeds. Therefore, in this experiment we do not compare with baselines, but rather limit the analysis to how our method is affected by the use of internal or external seeds for the generation of regular expressions.

8.2 Results

Table 7 lists the results. The average F-scores are above 0.85 when using internal seeds, requiring the annotation of 53 and 20 entities (including seeds) for *phone* and *post date*, respectively. In the worst cases observed, the method required the

Table 7. *Effectiveness and annotation cost (mean±95% CI) of* RB-AL *when generating regular expressions from internal and external seeds.*

	Internal		External	
	Phone	Post date	Phone	Post date
F-score	0.93 ± 0.02	0.86 ± 0.04	0.80 ± 0.06	0.74 ± 0.08
Precision	0.92 ± 0.02	1	0.77 ± 0.05	0.92 ± 0.05
Recall	0.94 ± 0.02	0.77 ± 0.05	0.84 ± 0.07	0.68 ± 0.08
Cost (entities)	53 ± 3	20 ± 2	56 ± 10	52 ± 5
Cost (tokens)	300 ± 16	95 ± 8	291 ± 56	308 ± 37

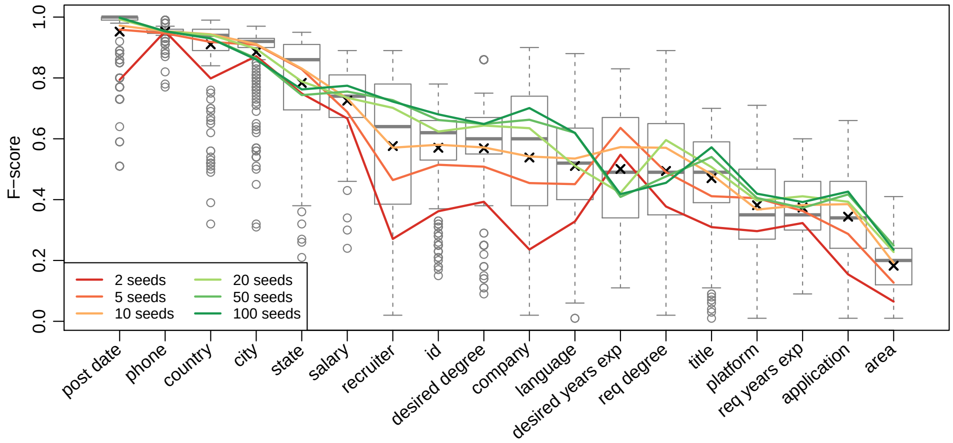


Fig. 5. Distributions of F-scores for each entity type and number of initial seeds. Entities are sorted by mean F-scored, marked with crosses.

annotation of 81 and 39 entities. F-scores drop by about 0.12 when using external seeds, but the average annotation cost is maintained at about 55 entities. In both cases, we thus obtain high effectiveness scores for a quite low annotation cost and with balanced precision and recall rates.

9 Discussion

9.1 Effectiveness

Fig. 5 shows the distributions of F-scores by entity type and initial number of seeds. We observe that entities clearly responding to character patterns, such as *post date* and *phone* number, obtain the best results (see Section 8). Other entities like *country*, *city*, *state* and *salary* also obtain high F-scores with only 2 seeds. If the number of seeds increases over 10, entities like *desired degree*, *recruiter*, *ID* and *company* get F-scores over 0.6 as well. As an example, we find that the *company* entities respond mainly to features like the case of the entity tokens, and the string and lemma of the contextual tokens. It is the case for example of the words `Inc` and `Corporation` following a large number of instances. In the case of *required/desired degree*, we find the word `in` after the degree level (e.g. `PhD/MS/BS in...`), except when they are `BSCS` or `MSCS`, because they already include the area of the title (eg. `BS/MS in Computer Science`).

The worst results are clearly obtained with the entity type *area*, though others like *application*, *platform* and *title* also get low F-scores between 0.2 and 0.4. The main problem in these cases is the amount of false positives due to the boundaries of the entities and their ambiguity. The same text fragment may appear multiple times within a document, referencing a different type of entity each time. For instance, in the sentence `...years experience with Oracle DBA`, `Oracle` is recognized as *application* and `DBA` as both *area* and *title*, just because it appears in the head of the job posting as `DBA's and Application Architects`. However, `Oracle DBA` also appears as *title* in other documents, and `Oracle` does as *platform*. In order to disambiguate these entities it would be useful to incorporate discourse features to the method. A possibility would be for instance to use information about the position of the text inside the document, or to exploit the occurrence of other entities as new features in the context or even nested within the entity.

9.2 Learning Cost

Throughout the paper we followed the standard approach in active learning to measure user effort in terms of annotation cost, that is, the total length of the text reviewed by the user. The correlation between this cost and the actual effort perceived by the user is a current topic of research in the area. Two additional factors that might impact this perception are the number of times the user is disrupted to validate new entities, and the interface employed to provide this feedback. With respect to such an interface, we tried to minimize effort by simplifying the required interaction to one of two actions: flag a proposed text fragment when it is not an entity, or correct its boundaries when they are not exact (this is seldom required, as seen in Table 5, "inexact boundaries"). This interface would be more similar to those offered by tools to aid in the annotation of corpora, where the automatic identification of potential elements of interest reduces the burden of annotating new documents (Reeve and Han 2005).

With respect to the number of active learning iterations, 80% of the times the method required less than 10 iterations per feature, and 50% of the times it required 4 or fewer iterations. As Fig. 6 shows, more than 35 iterations very seldom happened and are therefore considered outliers. Indeed, the four entity types mentioned above (*application*, *platform*, *area* and *title*) show again more extreme behavior. This is probably because, in addition to being ambiguous, their instances are longer than the others, and they are also more varied in terms of the feature values they produce. As a consequence, for these entities the method tends to produce more clusters

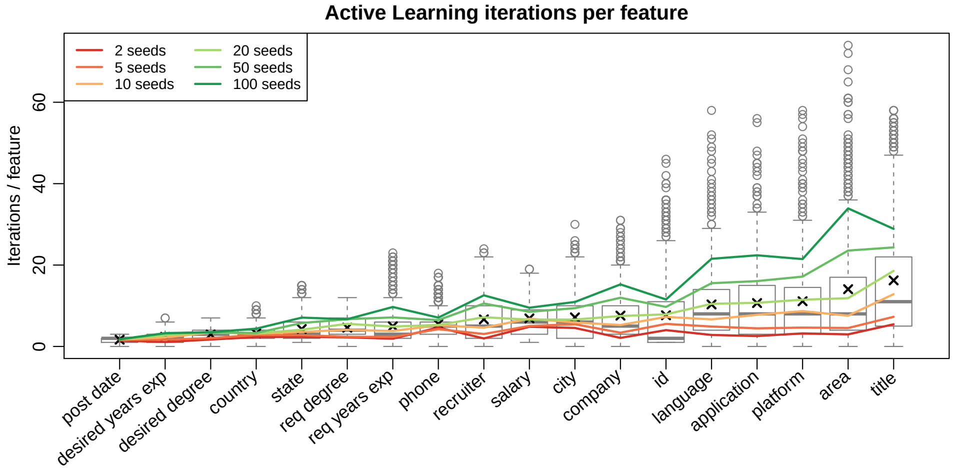


Fig. 6. Distributions of calls to the *ActiveLearning* procedure per feature, for each entity type and number of initial seeds. Entities are sorted by mean number of calls, marked with crosses.

and therefore more potential rules to validate separately during an active learning iteration. Similarly, *language* requires more iterations because its instances are also long and varied.

An important factor to consider is not only how often we require feedback from the user, but how efficiently we use that feedback, that is, how often an active learning iteration leads to potential rules getting fixed or new entities being discovered. Fig. 7-top shows that between 50% and 90% of the potential rules get accepted in any given active learning iteration, with an average of about 70%. Fig. 7-bottom similarly shows that in the majority of the iterations the method identifies at least one new entity in the training corpus. The notable exceptions are *desired degree*, probably because of the small number of instances, and some other types when using only 2 seeds. In general though, and unlike Fig. 6, the correlation with the number of seeds is weaker: more seeds mean more clusters and therefore more iterations, but it does not necessarily mean better rules per cluster. Indeed, Fig. 3 shows that increasing the number of seeds beyond 20 increases the annotation cost but not the F-scores, indicating that the method ends up learning similarly effective patterns but at a higher cost. This suggests the need for a more efficient use of the active learning process. One possible improvement might be to ask the user to validate text fragments for a minimum number of potential rules, possibly from different clusters, or to reduce the number of clusters by increasing the dissimilarity threshold θ_X. While the former could be applied in general, the latter would probably need to be adjusted for some entity types, because the choice of threshold has an impact on effectiveness in some cases (see Section 10).

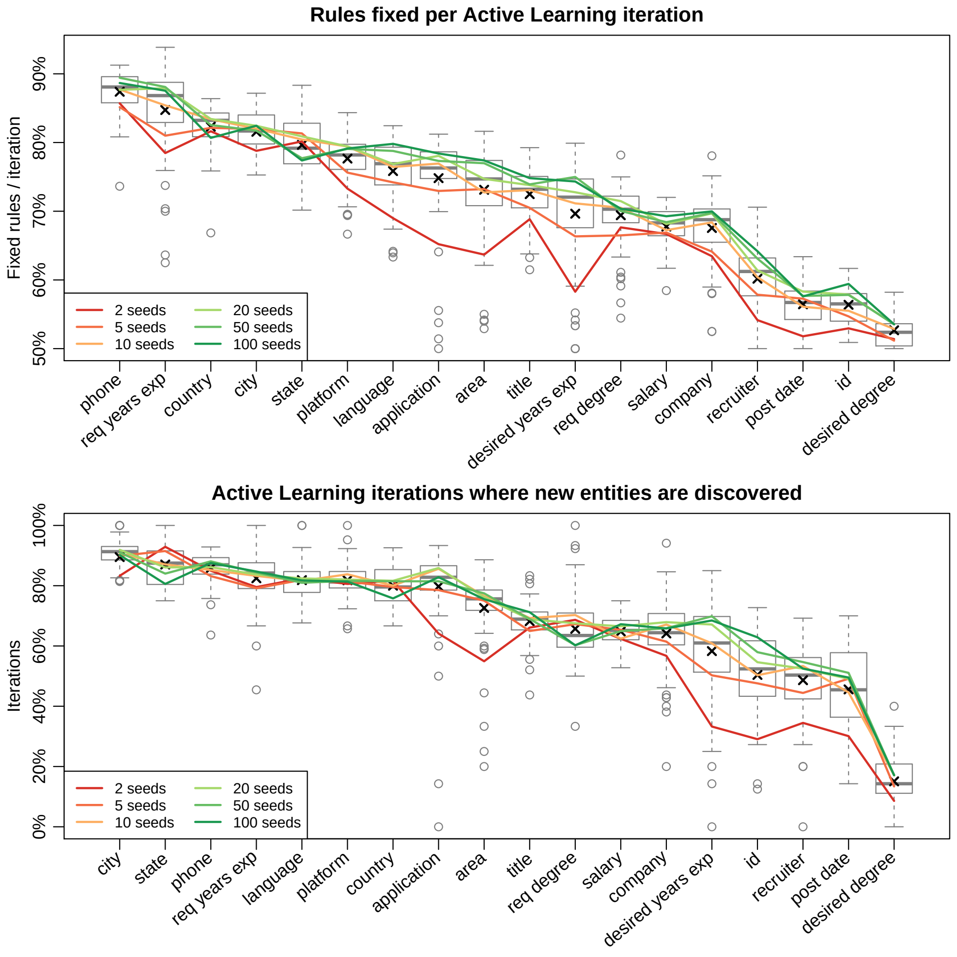


Fig. 7. Distributions of potential rules fixed during an active learning iteration (top), and iterations where at least a new entity was discovered (bottom), for each entity type and number of initial seeds. Entities are sorted by mean percentage, marked with crosses.

9.3 Readability of Patterns

An advantage of generating JAPE patterns is that the user has the possibility of reviewing them to correct, adapt or reuse rules. For example, in our case such an inspection allowed us to immediately spot a typo in an annotation of type *city* in document `job122387`, because of a rule generated to capture `Auston` in addition to `Austin`. However, the ability of the user to review a pattern is directly affected by its readability which, to some degree, depends on the number of rules it contains. The average number of JAPE rules in the token phase is 16 (median 9), whereas the average in the character phase is 22 (median 9). When mixed together in a single JAPE pattern, the same character rule can be linked to several token rules, which effectively means several instances of the same character rule (see Section 4.3). If we counted these as different ones, the average number of character rules per token rule is 2.3 (median 1.5). As expected, some entity types generate patterns with many more rules than others. In fact, 32% of all the JAPE rules learned in these experiments belong to entity types *area* and *title*.

The final patterns learned by our method are generated just by concatenating symbols, so they do not contain quantifiers to indicate repetitions. For instance, in Fig. 2.d rules `C_1` and `C_2` may be easily simplified by merging the regular expressions for the first token to `\p{Nd}{1,2}\p{Po}\p{Nd}{2}`. A similar example can be found for entity *required years experience*, where a single JAPE pattern is comprised of the rules `\p{D}`, `\p{D}\p{D}` and `\p{D}\p{Y}`, which capture instances like `7`, `10` and `4+`. These rules can be simplified to just `\p{D}{1,2}\p{Y}?`. Such a simplification would still be limited by the length of the positive examples used to learn. For instance, in order to generate a single pattern like `\p{D}{1,8}` to detect numbers, the method would need positive examples of lengths one to eight. In the extreme case of the Kleene star (e.g. `\p{D}*`), this simplification process would need to include heuristics because all positive examples are finite. Another consequence of generating patterns by concatenating symbols is that they do not contain alternations. For example, we find rules `US`, `USA` and `US\p{P}` for entity *country*, which capture different abbreviations for the United States. In this case, a single rule may be used instead: `US[A\p{P}]?`. Although these kinds of simplification may be found in the literature, we did not explore this line in the present work and leave its application to both the character and token phases for future work.

We finally note that some of the JAPE rules learned for the character phase could be avoided if we used gazetteers as token features, which we did not in order to center the evaluation on the performance of the learning method and not of the features. This would have a direct impact on the number of active learning iterations and hence on the number of rules as well. For instance, we find rules like `Win\p{D}\p{D}` and `Windows\p{S}\p{D}\p{D}` to capture different versions of the operating system, which could be easily detected with an appropriate gazetteer.

10 Conclusions and Future Work

We presented a semi-automatic and low-cost method to learn JAPE patterns and regular expressions for the recognition of named entities. Unlike patterns generated with statistical learning methods, rule-based patterns like JAPE are easily readable and understandable by humans, allowing users to refine them. In addition, they follow a standard specification that permits the use of arbitrary features, all of which has contributed to their wide acceptance and use in Information Extraction. Unlike other rule-based approaches, our method is able to generate more complex and customizable patterns by simultaneously analyzing features at different levels of granularity. The method proposed here generates these patterns semi-automatically from a corpus that does not need to be annotated beforehand. Instead, it just requires a small set of initial seeds that may or may not be contained in the corpus. Active learning techniques are used to guide the user in the annotation process by selecting appropriate text fragments for validation, thus reducing the overall cost required to generate patterns. The method uses features at the character level and allows the use of arbitrary features at the token level. Nonetheless, the same algorithm to identify rules is used in both cases and could be applied to other features in the same way.

We carried out several experiments with a variety of entity types and a basic set of features to evaluate the performance of the method. We showed that it can achieve good effectiveness levels with little annotation effort compared to unsupervised and semi-supervised state of the art. This effectiveness-cost trade-off suggests its use as a method to make annotations for new corpora or new entity types. In this scenario, we showed that the method was able to recognize more than 90% of all entities on average, requiring the user to review a very small fraction of the corpus. Further considering that the learned patterns can be manually refined by users, the proposed method is especially appealing to quickly gather annotations for supervised methods that can be more effective but require many more initial annotations.

Our future work will focus on the improvement of the clustering and rule selection algorithms, in particular of the functions to compute distances among entities and between rule vectors. Regarding the configuration for our experiments, here we fixed some parameters of the method, such as the clustering dissimilarity threshold. We plan to study how to adjust these parameters based on a preliminary analysis upon the initial seeds provided by the user, so that performance can be maximized for each entity type individually. Additionally, they need not be static during the learning process; they could be dynamically adapted based on the annotations made by the user via active learning. This self-adaptation ability could also lead to considering different stopping criteria for the learning step to avoid early convergence. Similarly, we may investigate how to better incorporate negative examples in the learning process to generate patterns with higher precision rates. Finally, it would be interesting to study the simplification of the final JAPE patterns. Even though this may be manually achieved by simple inspection, it would further contribute to their comprehension if this simplification was done automatically and transparently to the user.

Appendix

In terms of configuration, we recall that our method can be parameterized in several ways. Regarding the length of the contexts used for learning, we decided to use just $\theta_{wc} = 1$ character to the left and to the right of the entities when working with character features. This may allow us to detect patterns where the entities are enclosed with some kind of marks, such as < and > in e-mail addresses, parentheses, quotation marks, etc. In the case of tokens, we ran a preliminary experiment to figure out the most appropriate value for θ_{wt}. This experiment was also used to set the value for the dissimilarity threshold θ_X to cluster entities during the learning step. We ran a full-factorial experiment with context $\theta_{wt} \in \{1, 2, 3, 4, 6, 8\}$ tokens and threshold $\theta_X \in \{0.6, 0.8, 0.9\}$. For each context-threshold combination we ran

10 random trials with a set of 10 random seeds, and recorded both the F-score and the annotation cost in tokens.

The results showed that effectiveness is slightly affected by the context length: on average, the F-score increased by a factor of 0.008 ± 0.004 (mean±95% CI) with θ_{wt}. However, the annotation cost is more clearly affected, increasing by a factor of 55 ± 12. An Analysis of Variance (ANOVA) with post-hoc correction via Tukey's HSD shows that the cost when using a context of 2 tokens is not significantly higher than the most cost-efficient (1 token), and that the effectiveness is not significantly lower than the most effective (8 tokens), so we set the context length to $\theta_{wt} = 2$ tokens. A similar ANOVA analysis shows that both the F-score and the annotation cost decrease with the clustering dissimilarity threshold. A rough analysis by entity type shows that in some cases it is better to use a large threshold, while in other cases it is the other way around, or it just does not really affect performance. We decided to set the intermediate value $\theta_X = 0.8$. The interaction between context and threshold was not found to be statistically significant in any case.

For the active learning process, we set the minimum number of matches randomly sought throughout the corpus to $\theta_{\mathcal{M}} = 40,000$. To validate each potential rule vector, we select at least $\theta_z = 2$ of those matches, and the user validates $\theta_{\mathcal{R}} = 10$ and a minimum of $\theta_{\mathcal{P}} = 10$ matches overall. Finally, we show to the user the $\theta_{wv} = 10$ characters to the left and to the right of the candidate matches, under the assumption that she will be able to properly identify the entities whose boundaries are not exactly determined by the algorithm.

References

Alfonseca, E., and Manandhar, S. 2002. An unsupervised method for general named entity recognition and automated concept discovery. In *Proceedings of the 1st International Conference on General WordNet*, Mysore, India, pp. 34-43.

Appelt, D. E., and Onyshkevych, B. 1998. The Common Pattern Specification Language. In *Proceedings of the TIPSTER Text Program:Phase III.* Baltimore, Maryland, pp. 23-30.

Asahara, M., and Matsumoto, Y. 2003. Japanese Named Entity Extraction with Redundant Morphological Analysis. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1.* Edmonton, Canada, pp. 8-15.

Bikel, D. M., Miller, S., Schwartz, R., and Weischedel, R. 1997. Nymble: A High-Performance Learning Name-Finder. In*Proceedings of the fifth Conference on Applied Natural Language Processing.* Washington, DC, pp. 194-201.

Boguraev, B. K. 2004. Annotation-based finite state processing in a large-scale NLP architecture. In Nikolov et al. (ed.), *Recent Advances in Natural Language Processing III: Selected papers from RANLP 2003* John Benjamins Publishing, pp. 61-77.

Borthwick, A., Sterling, J., Agichtein, E., and Grishman, R. 1998. Exploiting Diverse Knowledge Sources via Maximum Entropy in Named Entity Recognition. In *Proceedings of the Sixth Workshop on Very Large Corpora.* Montreal, Canada, pp. 152-160.

Brauer, F., Rieger, R., Mocan, A., and Barczynski, W. M. 2011. Enabling Information Extraction by Inference of Regular Expressions from Sample Entities. In *Proceedings of the 20th Conference on Information and Knowledge Management.* Glasgow, United Kindgdom, pp. 1285-94

Califf, M.E. 1998. Relational learning techniques for natural language information extraction *PhD thesis, The University of Texas at Austin*

Chiticariu, L., Krishnamurthy, R., Li, Y., Reiss, F., and Vaithyanathan, S. 2010. Domain Adaptation of Rule-based Annotators for Named-entity Recognition Tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Massachusetts, USA, pp. 1002-12

Chiticariu, L., and Reiss, F. R. 2013. Rule-based Information Extraction is Dead! Long Live Rule-based Information Extraction Systems! In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Seattle, USA, pp. 827-32

Ciravegna, F., and Wilks, Y. 2003. Designing Adaptive Information Extraction for the Semantic Web in Amilcare. In Handschuh, S., and Staab, S. (ed.), *Annotation for the Semantic Web, Frontiers in Artificial Intelligence and Applications series, Vol. 96.* IOS Press, pp. 112-27.

Culotta, A., and Mccallum, A. 2005. Reducing labeling effort for structured prediction tasks. In *Proceedings of the Twentieth National Conference on Artificial intelligence.* Pittsburgh, Pennsylvania, pp. 746-51

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., et al. 2013. Developing Language Processing Components with GATE (a User Guide). Technical report. University of Sheffield Department of Computer Science.

Day, W. H. E. and Edelsbrunner, H. 1984. Efficient Algorithms for Agglomerative Hierarchical Clustering Methods. *Journal of Classification* 1(1):7-24.

Drozdzynski, W., Krieger, H.-U., Piskorski, J., Schäfer, U., and Xu, F. 2004. Shallow Processing with Unification and Typed Feature Structures: Foundations and Applications. *Künstliche Intelligenz* 1(1):17-23.

Etzioni, O., Cafarella, M. J., Downey, D., Popescu, A., Shaked, T., et al. 2005. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artificial Intelligence* 165(1):91-134.

Fersini, E., Messina, E., Felici, G., and Roth, D. 2014. Soft-Constrained Inference for Named Entity Recognition. *Information Processing and Management* 50(5):807-19.

Finkel, J. R., Grenager, T., and Manning, C. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics.* Ann Arbor, Michigan, pp. 363-70

Freitag, D. 1998. Toward General-Purpose Learning for Information Extraction Retargetability. In *Proceedings of the 17th International Conference on Computational Linguistics.* Montreal, Canada, pp. 404-8

Gantz, J. and Reinsel, D. 2012. The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. Technical report. IDC.

Gupta, S., and Manning, C.D. 2014. Improved Pattern Learning for Bootstrapped Entity Extraction. In *Proceedings of the 18th Conference on Computational Natural Language Learning.* Baltimore, USA, pp. 98-108

Hachey, B., Alex, B., and Becker, M. 2005. Investigating the Effects of Selective Sampling on the Annotation Task. In *Proceedings of the Ninth Conference on Computational Natural Language Learning* Ann Arbor, Michigan, pp. 144-51.

Haertel, R. A., Seppi, K. D., Ringger, E. K., and Carroll, J. L. 2008. Return on Investment for Active Learning. *NIPS Workshop on Cost-Sensitive Learning.*

Irmak, U., and Kraft, R. 2010. A Scalable Machine-learning Approach for Semi-structured Named Entity Recognition. In *Proceedings of the 19th International Conference on World Wide Web.* Raleigh, USA, pp. 461-70

Jones, R. 2005. *Learning to Extract Entities from Labelled and Unlabelled Text.* PhD thesis, Carnegie Mellon University.

Kazama, J., and Torisawa, K. 2007. A new perceptron algorithm for sequence labeling with non-local features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Prague, Czech Republic, pp. 315-24

Kluegl, P., Toepfer, M., Beck, P.-D., Fette, G., and Puppe, F. 2015. Uima ruta: Rapid development of rule-based information extraction applications. *Natural Language Engineering* 22(1):1-40.

Lavelli, A., Califf, M.E., Ciravegna, F., Freitag, D., Giuliano, C., Kushmerick, N., and Romano, L. 2004. IE evaluation: Criticisms and recommendations. *AAAI Workshop on Adaptive Text Extraction and Mining.* San Jose, California

Levenshtein, V. I. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady* 10(8):707-10.

Li, Y., Bontcheva, K., and Cunningham, H. 2009. Adapting SVM for Data Sparseness and Imbalance: A Case Study in Information Extraction. *Natural Language Engineering* 15(2):241-71.

Li, Y., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., and Jagadish, H. 2008. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Waikiki, Hawaii, pp. 21-30

Liu, X., Wei, F., Zhang, S., and Zhou, M. 2013. Named Entity Recognition for Tweets. *ACM Transactions on Intelligent Systems and Technology* 4(1):3.

Maedche, A. and Staab, S. 2001. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems* 16(2):72-9.

Marrero, M., Sánchez-Cuadrado, S., Morato, J., and Andreadakis, G. 2009. Evaluation of named entity extraction systems. *Research in Computing Science* 41:47-58.

Marrero, M., Sánchez-Cuadrado, S., Urbano, J., Morato, J., and Moreiro J. A. 2012. Information Retrieval Systems Adapted to the Biomedical Domain. Technical report. Arxiv.org

Marrero, M. and Urbano, J. 2015. Information Extraction Grammars. *European Conference on Information Retrieval.* Vienna, Austria

Marrero, M., Urbano, J., Sánchez-Cuadrado, S., Morato, J., and Gómez-Berbís, J. M. 2013. Named Entity Recognition: Fallacies, Challenges and Opportunities. *Journal of Computer Standards and Interfaces* 35(5):482-9.

McCallum, A., and Li, W. 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh Conference on Natural Language Learning.* Edmonton, Canada, pp. 188-91

Nadeau, D. 2007. *Semi-supervised named entity recognition: learning to recognize 100 entity types with little supervision.* PhD thesis, School of Information Technology and Engineering, University of Ottawa.

Nagesh, A. and Chiticariu, L. 2012. Towards Efficient Named-Entity Rule Induction for Customizability. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Jeju Island, Korea, pp. 128-38

Nouvel, D., Antoine, J.Y., Friburger, N. and Soulet, A. 2012. Coupling Knowledge-Based and Data-Driven Systems for Named Entity Recognition. In *Proceedings of the ACL Workshop on Innovative Hybrid Approaches to the Processing of Textual Data.* Avignon, France, pp. 69-77

Nédellec, C., Bossy, R., Kim, J.-D., Kim, J.-j., Ohta, T., et al. 2013. Overview of BioNLP Shared Task 2013. *ACL Workshop on BioNLP.* Sofia, Bulgaria, pp. 1-7

Pang, B. and Lee, L. 2007. Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval* 2(1-2):1-135.

Pasca, M., Lin, D., Bigham, J., Lifchits, A. and Jain. A. 2006. Organizing and searching the world wide web of facts-step one: the one million fact extraction challenge. In *Proceedings of the 21st National Conference on Artificial Intelligence.* Boston, Massachusetts, pp. 1400-5

Popescu, A.-M. and Etzioni, O. 2005. Extracting Product Features and Opinions from Reviews. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Vancouver, Canada, pp. 339-46

Ratinov, L. and Roth, D. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Conference on Natural Language Learning.* Boulder, Colorado, pp. 147-55

Reeve, L. H. and Han, H. 2005. Survey of Semantic Annotation Platforms. *ACM Symposium on Applied Computing.* Santa Fe, USA, pp. 1634-8

Rinaldi, F., Vasilakopoulos, A., Zervanou, K., Bernard, L., Zarri, G. P., et al. 2005. CAFETIERE: Conceptual Annotations for Facts, Events, Terms, Individual Entities, and RElations. Technical Report TR-U4.3.1. Parmenides Project IST-2001-39023.

Ringger, E., Carmen, M., Haertel, R., Seppi, K., Lonsdale, D., et al. 2008. Assessing the costs of machine-assisted corpus annotation through a user study. In *Proceedings of the International Conference on Language Resources and Evaluation.* Marrakech, Morocco, pp. 3318-24

Ritter, A., Clark, S., and Etzioni, O. 2011. Named Entity Recognition in Tweets: An Experimental Study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Edinburgh, United Kingdom, pp. 1524-34

Sarawagi, S. 2008. Information Extraction. *Foundations and Trends in Databases* 1(3):261-377.

Sekine, S., Grishman, R., and Shinnou, H. 1998. A Decision Tree Method for Finding and Classifying Names in Japanese Texts. In *Proceedings of the sixth Workshop on Very Large Corpora.* Montreal, Canada, pp. 171-8

Settles, B. 2012. Active Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6(1):1-114

Shen, D., Zhang, J., Su, J., Zhou, G., and Tan, C.-L. 2004. Multi-Criteria-Based Active Learning for Named Entity Recognition. In *Proceedings of the Annual Meeting of the ACL.* Barcelona, Spain, pp. 589-96

Shinyama, Y., and Sekine, S. 2004. Named entity discovery using comparable news articles. In *Proceedings of the International Conference on Computational Linguistics.* Geneva, Switzerland, pp. 848

Silberztein, M. 2005. NooJ: a linguistic annotation system for corpus processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Vancouver, Canada, pp. 10-11

Siniakov, P. 2008. *GROPUS-an Adaptive Rule Based Algorithm for Information Extraction.* PhD thesis, Free University of Berlin.

Soderland, S. 1999. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning* 34(1):233-72

Srihari, R. K. and Li, W. 1999. *Information Extraction Supported Question Answering.* Technical Report. Cymfony Inc.

Srikant, R. and Agrawal, R. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the International Conference on Extending Database.* Avignon, France, pp. 1-17

Thompson, C. A., Califf, M. E., and Mooney, R. J. 1999. Active Learning for Natural Language Parsing and Information Extraction. In *Proceedings of the International Conference on Machine Learning.* Bled, Slovenia, pp. 406-14

Tomanek, K., Wermter, J., and Hahn, U. 2007. An Approach to Text Corpus Construction which Cuts Annotation Costs and Maintains Reusability of Annotated Data. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* Prague, Czech Republic, pp. 486-95

Uren, V. S., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., et al. 2006. Semantic Annotation for Knowledge Management: Requirements and a Survey of the State of the Art. *J. Web Semantics* 4(1):14-28.

Vijayanarasimhan, S., and Grauman, K. 2009. What's It Going To Cost You? Predicting Effort vs. Informativeness for Multi-Label Image Annotations. In *Proceedings of the Confernce on Computer Vision and Pattern Recognition.* Miami, Florida, pp. 2262-9

Vlachos, A. 2008. A Stopping Criterion for Active Learning. *Computer Speech & Language* 22(3):295-312.

Wu, T., and Pottenger, W. M. 2005. A Semi-Supervised Active Learning Algorithm for Information Extraction from Textual Data. *Journal of the American Society for Information Science and Technology* 56(3):258-71.