The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Content and Context: Two-Pronged Bootstrapped Learning for Regex-Formatted Entity Extraction

Stanley Simoes
Indian Institute of Technology Madras
stanley@cse.iitm.ac.in

Deepak P
Queen's University Belfast
deepaksp@acm.org

Munu Sairamesh
Indian Institute of Technology Madras
musram@gmail.com

Deepak Khemani
Indian Institute of Technology Madras
khemani@iitm.ac.in

Sameep Mehta
IBM Research - India
sameepmehta@in.ibm.com

Abstract

Regular expressions are an important building block of rule-based information extraction systems. Regexes can encode rules to recognize instances of simple entities which can then feed into the identification of more complex cross-entity relationships. Manually crafting a regex that recognizes all possible instances of an entity is difficult since an entity can manifest in a variety of different forms. Thus, the problem of automatically generalizing manually crafted seed regexes to improve the recall of IE systems has attracted research attention. In this paper, we propose a bootstrapped approach to improve the recall for extraction of regex-formatted entities, with the only source of supervision being the seed regex. Our approach starts from a manually authored high precision seed regex for the entity of interest, and uses the matches of the seed regex and the context around these matches to identify more instances of the entity. These are then used to identify a set of diverse, high recall regexes that are representative of this entity. Through an empirical evaluation over multiple real world document corpora, we illustrate the effectiveness of our approach.

Introduction

Notwithstanding recent advancements in learning-based IE systems, rule-based IE systems remain more popular in the industry simply because rules are easier to understand and maintain (Chiticariu, Li, and Reiss 2013). A core task in IE systems is entity extraction, which involves identifying instances of entities in unstructured and semi-structured text. In rule-based IE systems, regular expressions (regexes) provide an elegant means of characterizing entities since most entities have an underlying pattern. Phone numbers, email addresses, and social security numbers are some examples.

A good regex for a chosen entity would accurately match most, if not all, of its instances, but manually designing such a regex is often laborious. A human expert tasked with coining a good regex for an entity would typically think of a few of its instances and try to generalize them to a regex. While such a regex would likely match a large chunk of instances, it might miss out on some rather intuitive variants. For example, a human expert would typically author `1-\d{3}-\d{3}-\d{4}` for extracting phone numbers in the US. However, this regex misses out on phone number instances such as *1-800-FOR-SALE* and *1-800-COMPANY* which are phone number representations of the kind regularly used by marketing agencies (Murthy, P., and Deshpande 2012). Brauer et al. (2011) cite a scenario involving invoice numbers from a particular domain, where the human would think of instances such as *2010.08338* and *2000.348* as examples, to author the regex `20\d{2}\.\d+`. But this regex does not extract other valid invoice numbers such as *2008-035465* and *11-093*, as pointed out therein. Additionally, they observe that entity types that are relevant to IE within industry settings often follow a strict underlying syntactical pattern.

Further, regexes also need to be refined continuously to cover new variants of entities that did not previously exist. For example, until a few years ago, the regex `9\d{9}` was appropriate to extract mobile numbers in India. Today, these mobile numbers also start with *7* or *8*, necessitating the generalization of the first digit. Microprocessor product identifiers form another such example. Until the end of the last decade, the regex `i\d-\d{3}` would identify every Intel Core i5 and i7 microprocessor. Since then, newer models such as *i5-3340S*[1] and *i7-4770TE*[2] have been released, necessitating regex refinement through generalization to cover them. In such cases, it would prove beneficial to have a system that assists the human by automatically learning and suggesting (i) instances that may have been overlooked, or better yet, (ii) meaningful regexes themselves. The human can then author regexes that better cover the entity using her domain knowledge in conjunction with the learned instances or refined regexes.

One direction towards finding high precision and high recall regexes would be to infer such regexes from labeled data (Brauer et al. 2011; Bartoli et al. 2016). Alternatively, a seed regex can be progressively specialized (Li et al. 2008) or generalized (Murthy, P., and Deshpande 2012), optionally soliciting user feedback in the process. Among such refinements, generalization is harder since it requires taking a call on new/unseen matches. Thus, it has attracted research attention, mainly employing active learning (Murthy, P., and Deshpande 2012; Bartoli et al. 2017).To the best of our knowledge, our work is the first approach that does not require explicit instance labeling for the regex generalization problem.

Our Contribution: This paper presents a two-stage approach for enhancing the recall of regex-based entity extraction systems. In the first stage, we take a high precision seed regex as input, and

[1]https://en.wikipedia.org/wiki/List_of_Intel_Core_i5_microprocessors

[2]https://en.wikipedia.org/wiki/List_of_Intel_Core_i7_microprocessors

	INPUT			OUTPUT		METHOD
	RX	INS-POS	INS-NEG	RX	INS	
Riloff and Jones (1999)	✗	✓	✗	✗	✓	iterative pattern-based bootstrapping
Thelen and Riloff (2002)	✗	✓	✢	✗	✓	iterative pattern-based bootstrapping
Sarmento et al. (2007)	✗	✓	✗	✗	✓	co-occurrence based set expansion
Li et al. (2008)	✓	✓	✓	✓	✗	greedy hill climbing
Babbar and Singh (2010)	✓	✓	✗	✓	✗	iterative regex relaxation & clustering
Brauer et al. (2011)	✗	✓	✗	✓	✓	prefix and suffix automata
Murthy, P., and Deshpande (2012)	✓	✓	✓	✓	✓	best-first search, active learning
Bartoli et al. (2014)	✗	✓	✓	✓	✗	genetic programming
Gupta and Manning (2014)	✗	✓	✢	✗	✓	iterative pattern-based bootstrapping
Gupta and Manning (2015)	✗	✓	✓	✗	✓	iterative pattern-based bootstrapping
Qadir et al. (2015)	✗	✓	✗	✗	✓	non-iterative pattern-based set expansion
Bartoli et al. (2016)	✗	✓	✗	✓	✗	genetic programming
Bartoli et al. (2017)	✗	✓	✓	✓	✓	genetic programming, active learning
Our Approach	✓	✗	✗	✓	✓	iterative bootstrapping

RX (INS) stand for regex (instances).
INS-POS and INS-NEG denote explicit instance supervision, standing for positive and negative instances respectively.
✓ indicates presence, ✗ absence, and ✢ optional.

Table 1: Overview of related work.

employ bootstrapping to learn new instances similar to the matches by the seed, leveraging a specified document corpus. In the second stage, we use these learned instances to recommend a set of high precision, high recall regexes from the generalization space of the seed regex, thus equipping the human with regex refinements that are more representative of the entity in question. With minimal manual tuning, these regexes can be deployed in entity extraction systems. We illustrate, through a variety of empirical analyses, that our method outperforms current methods appropriate for the task.

Roadmap: We first review related work relevant to our task, followed by the formal problem statement. We then introduce relevant background, followed by a description of our proposed approach. Next, we detail our empirical evaluation, followed by conclusions.

Related Work

A high-level overview of representative related work categorized according to input and output specifications appear in Table 1. All techniques either start with a seed regex or a set of labeled instances or both. It may be noted that a seed regex implies a set of labeled instances, since the matches of the seed regex are indicative of the entity to be modeled. Similarly, a regex output consistently implies a set of their matches in the output. It is however worth noting that literature differs slightly in the treatment of seed regex matches in the input side; while all seed regex matches are considered as correct entity instances in Murthy, P., and Deshpande (2012), Li et al. (2008) allows for the possibility that some of the seed regex matches are not valid instances of the entity. Thus, we indicate requirement of explicit (i.e., not implied by the input regex, if any) positive and negative instance supervision in Table 1, denoted by INS-POS and INS-NEG respectively.

Regex Refinement Techniques: The techniques that make use of a seed regex differ in the type of adaptation they apply on the seed regex, with all existing methods using labeled instances in addition to the seed regex. Li et al. (2008) specializes the seed regex, whereas Babbar and Singh (2010) and Murthy, P., and Deshpande (2012) use generalization as the adaptation technique. Apart from the adaptation method, the techniques differ in the way they exploit the reference document dataset used to guide the search process. In contrast to methods that require labeled instance data in addition to the seed regex, our approach is modeled to work without such instance labelling, and thus requires no extra information other than the seed regex and a reference document dataset.

Regex Learning Techniques: A second group of techniques, all of which employ supervision, consider learning a regex from scratch using a set of positive and negative entity instances in text; these methods expect more comprehensive supervision to offset the non-usage of seed regexes. While Brauer et al. (2011) use automata to characterize prefix and suffix patterns of positive instances, Bartoli et al. (2014) use genetic programming to infer a regex from positive and negative instances. Bartoli et al. (2016) improve the regex learning by learning regexes with disjunctions and lookaround operators. In a recent work, they consider usage of limited supervision in an active learning approach (Bartoli et al. 2017). Our problem differs from this family of techniques too in the non-usage of supervision in the form of labeled instances.

Non-regex Match Set Expansion Techniques: The third category of methods do not use regexes in the input or output, and instead start with a set of labeled instances and output a larger set of instances that better characterize the entity. Riloff and Jones (1999) introduced bootstrapping as an approach for accurate identification of entity instances using context around labeled instances, later extended in Thelen and Riloff (2002). On the other hand, Sarmento et al. (2007) use co-occurrence as a heuristic to formulate a solution. Gupta and Manning (2014) leverage unlabeled data to model and score patterns for accurate instance identification. This was extended in Gupta and Manning (2015) to perform collective instance identification of a set of unrelated entities through mutual supervision. It may be noted that this work considers instances of *other* entities as negative supervision to facilitate learning of instances for a particular entity, and is thus not applicable to scenarios that focus on a single entity at a time. Qadir et al. (2015) propose a pattern-based set expansion method using an external semantic similarity oracle, targeting short texts from sources such as microblogs. In general, this oracle may not always be available. *In the interest of comparing our match set expansion module against the state-of-the-art for semi-supervised (single) entity instance identification,*

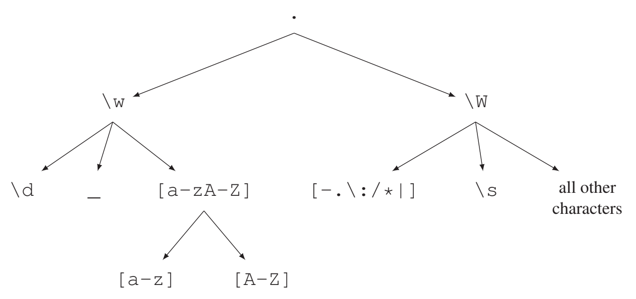

Figure 1: Character class hierarchy

we use Gupta and Manning (2014) as the baseline method in our empirical evaluation.

Problem Statement

Consider a document corpus $\mathcal{D}$ and a manually crafted seed regex r_{seed} that is intended to capture instances of a particular entity. We use $\mathcal{M}(r, \mathcal{D})$ to denote the set of all matches of the regex r within the document corpus $\mathcal{D}$; thus, $\mathcal{M}(r_{seed}, \mathcal{D})$ denotes the set of seed regex matches in $\mathcal{D}$. We are interested in two tasks:

1. **Expansion of Matches:** Identifying a larger set of entity instances than the matches by the seed, i.e., $\mathcal{M}(r_{seed}, \mathcal{D})$. We will use $\mathcal{M}_{exp}$ to denote the expanded set of matches.
2. **Regex Recommendation:** Identifying generalizations of r_{seed}, as $\mathcal{R}$, each of which can more comprehensively characterize the entity in question, by way of their matches covering more entity instances than $\mathcal{M}(r_{seed}, \mathcal{D})$ while maintaining a high precision. The human user can then choose from among regexes in $\mathcal{R}$ for more accurate entity extraction.

In both the above tasks, our intent is to ensure a broader coverage of the entity while retaining a high precision (i.e., ensuring that very few incorrect instances get covered in the process). As in Murthy, P., and Deshpande (2012), we assume that r_{seed} is highly precise in that, all its matches are correct. This is intuitively true of manually crafted regexes since they are meant to capture the correct instances that the regex author is able to recollect.

Evaluation: It is desired and expected that $\mathcal{M}_{exp}$ and $\mathcal{M}(r, \mathcal{D})$ $(\forall r \in \mathcal{R})$ would cover more correct matches[3] (i.e., correct instances of the entity) than $\mathcal{M}(r_{seed}, \mathcal{D})$; our evaluation focuses on quantifying the enhancement in coverage of correct matches and ensuring the exclusion of incorrect matches. Since we are more interested in evaluating the more general task, our full comparative evaluation is performed on the set expansion task. As stated previously, we expect $recall(\mathcal{M}_{exp}) > recall(\mathcal{M}(r_{seed}, \mathcal{D}))$, given the nature of the set expansion task. To ensure exclusion of incorrect matches, we would like $\mathcal{M}_{exp}$ to contain as few incorrect matches as possible. Thus, our quality measure is simply the precision, recall, and F-score of $\mathcal{M}_{exp}$ when compared against a gold standard set of correct entity instances.

Background

Our method, much like previous work (e.g., Murthy, P., and Deshpande (2012)), uses a regex generalization space to guide the set expansion and recommendation tasks. We now illustrate the generalization space, being essential background to describe our approach.

[3]We consistently use the term *matches* to refer to text fragments that are matched by a regex. Since we expect the regex to characterize an entity of interest, these matches are highly correlated with entity *instances* as well.

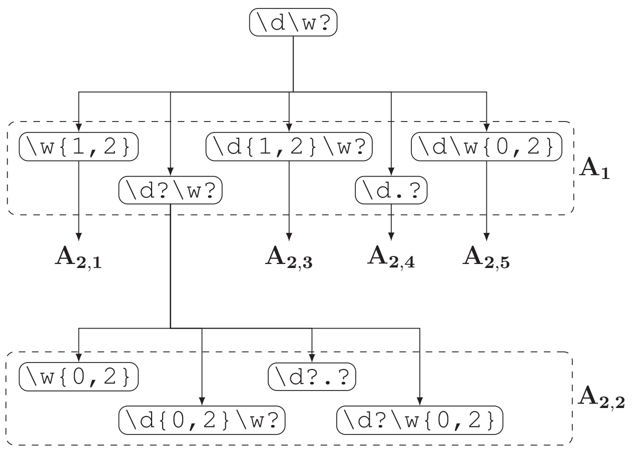

Figure 2: Generating regexes in $\mathcal{G}^2_{\texttt{\d\w?}}$

A regex is made up of two basic units: the character class, and the quantifier. The character class specifies the domain of characters, while the quantifier specifies the number of times a character in the preceding character class occurs. For example, in the regex `\d{2,4}`, the character class is `\d` and the quantifier is `{2,4}`, with `2` being the lower bound and `4` the upper bound. This regex matches all sequences of digits of lengths 2, 3, or 4. As in previous literature (Murthy, P., and Deshpande 2012; Li et al. 2008), we restrict ourselves to the class of regexes that are sequences of character class - quantifier pairs.

Our approach for generalization follows the framework in Murthy, P., and Deshpande (2012), where each generalization step is modeled as replacing a single regex unit (which is either a character class or a quantifier) by its generalization. In a slight departure from the framework, we first *flatten* the seed regex to form a regex with just `{1,1}` or `{0,1}` quantifiers. Accordingly, `\d{1,2}` is flattened to `\d{1,1}\d{0,1}`. This flattening allows for more fine-grained generalizations due to the design of the generalization space. This quantifier flattening at each character class level is not feasible in the case of grouped character class sequences, such as those in `(\d[a-z]){1,3}` or `([a-z]|[0-9]){5}`. In those cases, we retain the grouping in the generalization process while generalizing the character classes within them individually. The actual generalization step design is as follows: if the choice of the regex unit is a character class, the generalization involves replacing it by its parent in the hierarchy shown in Figure 1. As evident from this hierarchy, every character class has at most one generalization, and the root character class `.` cannot be generalized any further. The quantifier is generalized either by decreasing its lower bound by 1, or increasing its upper bound by 1. Thus, a single generalization of `\d{1,1}\d{1,1}` would produce `\d{1,2}\d{1,1}` if the choice of regex unit is the first quantifier and the choice of generalization is rightward (a leftward generalization of the same quantifier would yield `\d{0,1}\d{1,1}`), whereas a generalization on the second character class would yield `\d{1,1}`. A regex can be generalized multiple times, each generalization step involving a regex unit which itself may be the output of a previous step. This leads us to the generalization space.

Definition 1 d-Depth Generalization Space ($\mathcal{G}^d_r$): *All regexes that can be arrived at by <u>at most</u> d generalization steps from a regex r form the d-depth generalization space of r.*

For example, `\w{0,2}` is in `\d\w?` since two generalization steps can affect the transformation in question. Figure 2 illustrates the generation of regexes in the 2-depth generalization space of the regex `\d\w?`. To avoid clutter in the illustration, we re-aggregate the flattened parts as and when possible, using standard regex notation. The regexes in $\mathbf{A_1}$, along with the root regex are in the 1-depth generalization space $\mathcal{G}^1_{\texttt{\textbackslash d\textbackslash w?}}$. Each of the five regexes in $\mathbf{A_1}$ are further generalized yielding $\mathbf{A_{2,1}}$, $\mathbf{A_{2,2}}$, $\mathbf{A_{2,3}}$, $\mathbf{A_{2,4}}$, and $\mathbf{A_{2,5}}$, from which the regexes in $\mathbf{A_{2,2}}$ are shown in the figure. The 2-depth generalization space of `\d\w?` is the collection of every regex in $\mathbf{A_{2,1}}$, $\mathbf{A_{2,2}}$, $\mathbf{A_{2,3}}$, $\mathbf{A_{2,4}}$, and $\mathbf{A_{2,5}}$, as well as `\d\w?`. It may be noted that r itself is contained in every generalization space, i.e., $\forall d, r \in \mathcal{G}^d_r$ holds.

Our Approach

We now present our method[4] for match set expansion and regex recommendation. Our technique first addresses the match set expansion task, forming an expanded set of matches $\mathcal{M}_{exp}$, which is in turn used to identify a ranked list of regexes that maximizes the coverage of matches within the expanded set using a diversity-conscious greedy formulation. Accordingly, we first describe our core method, that for match set expansion, followed by the greedy approach for regex recommendation.

Overview of Match Set Expansion

Our method starts with the set of seed regex matches, i.e., $\mathcal{M}(r_{seed}, \mathcal{D})$, and progressively builds the expanded set $\mathcal{M}_{exp}$ over the course of many iterations, without using any supervision in the form of labeled instances. The technique starts with two sets of matches, denoted as $\mathcal{P}$ and $\mathcal{N}$ meant to contain correct matches of the entity (i.e., positives) and incorrect matches (i.e., negatives) respectively. $\mathcal{P}$ is initialized to $\mathcal{M}(r_{seed}, \mathcal{D})$ while $\mathcal{N}$ covers matches of all other regexes in $\mathcal{G}^d_{r_{seed}}$ (i.e., $\big(\cup_{r \in \mathcal{G}^d_{r_{seed}}} \mathcal{M}(r, \mathcal{D})\big) - \mathcal{M}(r_{seed}, \mathcal{D})$), reflecting the initial belief that the regex author accurately characterized the entity to kick-off the bootstrapped approach. Over iterations, $\mathcal{P}$ is progressively enlarged by eating into matches from $\mathcal{N}$. Specifically, each iteration selects a small number of matches that are judged to be *'similar'* to instances in $\mathcal{P}$ based on both *content* of the matches as well as their *context*, to be included in $\mathcal{P}$. Thus, two elements are central to our method:

- **Modeling:** The representation of each match as a combination of *content* and *context* features.
- **Similarity Judgment:** The assessment of $\mathcal{P}$-likeness of matches, to decide on which of the matches would get included in $\mathcal{P}$ in each iteration. We accomplish this using a logistic regression model.

The third element, the quantum of the change (from $\mathcal{N}$ to $\mathcal{P}$) in each iteration, is modeled as a parameter to allow the user to choose an appropriate trade-off in the aggressive-conservative spectrum. We now detail the modeling and similarity judgment parts in separate subsections herein, followed by the overall algorithm.

Modeling Matches

We represent each match in $\mathcal{N} \cup \mathcal{P}$ using a vector of four features, of which two model the *content* of the match and the other two model the *lexical context* of the match. Each feature uses a different way of quantifying the $\mathcal{P}$-likeness of each match.

Content Modeling: Both the content features use edit distance modeling (Levenshtein 1966) to score a match against the current estimates of $\mathcal{P}$ and $\mathcal{N}$ (at the iteration). Consider a match $m \in (\mathcal{N} \cup \mathcal{P})$; we define $\mathcal{E}_t(m)$ as the set of matches in $\mathcal{N} \cup \mathcal{P}$ that are within t edit distance of m. For example, *CS402* would be part of $\mathcal{E}_2(\mathit{CS413})$ since two character changes can transform *CS413* to *CS402*. The $\mathcal{P}$-likeness feature of m via t-edit distance modeling is specified as the fraction of $\mathcal{E}_t(m)$s that are within the current estimate of $\mathcal{P}$. Thus:

$$\mathit{Content\text{-}Score}_t(m) = \frac{|\mathcal{E}_t(m) \cap \mathcal{P}|}{|\mathcal{E}_t(m)|} \tag{1}$$

Our two content features are simply $\mathit{Content\text{-}Score}_2(m)$ and $\mathit{Content\text{-}Score}_3(m)$.

Context Modeling: For most entities, the context or the lexical neighborhood on either side of a match provides critical evidence to help determine the correctness of the match. For example, in a text fragment "*... please do call me on 0123456789 or leave a text therein, ...*", the few words in the left and right of the actual phone number instance (i.e., *0123456789*) hold valuable evidence in support of the correctness of the match. In a way, even if the actual phone number were redacted in the document, it would be easy for a human user to guess that the redacted fragment was a phone number instance making use of the left and right contexts. Our context modeling within each iteration takes the matches in the current state of the $\mathcal{P}$ set and builds language models (Jurafsky and Martin 2000, Chapter 6) to characterize the left (i.e., pre) and right (i.e., post) context of these matches. Accordingly, we build two unigram language models $L_{\mathcal{P}}$ and $R_{\mathcal{P}}$. $L_{\mathcal{P}}$ is built over the five tokens[5] to the left of each match in $\mathcal{P}$. In the earlier example, each of the words $\{please, do, call, me, on\}$ contribute to building the language model $L_{\mathcal{P}}$. Similarly, $\{or, leave, a, text, therein\}$ all feed into building $R_{\mathcal{P}}$. We perform add-one smoothing on both $L_{\mathcal{P}}$ and $R_{\mathcal{P}}$ for better generalizability. The two context features for each match is then just the scoring of the left (right) context of the match m ($\in (\mathcal{N} \cup \mathcal{P})$) against $L_{\mathcal{P}}$ ($R_{\mathcal{P}}$).

$$\mathit{Context\text{-}Score}_L(m) = \sum_{w \in LC(m)} ln(L_{\mathcal{P}}(w)) \tag{2}$$

$$\mathit{Context\text{-}Score}_R(m) = \sum_{w \in RC(m)} ln(R_{\mathcal{P}}(w)) \tag{3}$$

where $LC(m)$ and $RC(m)$ denote the left and right contexts of the match m. The log-sum (instead of product) prevents floating-point underflow that is likely while multiplying small unigram probabilities.

With the log-sum quantifying context support, we have one score for each context, and two content scores. In short, the $\mathcal{P}$-likeness of each match m is represented as a vector of four features:

$$\begin{aligned}[\mathit{Content\text{-}Score}_2(m), \mathit{Content\text{-}Score}_3(m),\\ \mathit{Context\text{-}Score}_L(m), \mathit{Context\text{-}Score}_R(m)]\end{aligned} \tag{4}$$

Similarity Judgment and $\mathcal{P}/\mathcal{N}$ Memberships

We would now like to learn a model based on the four-feature representation to characterize the current estimates (i.e., estimates at the start of the iteration) of positive and negative matches (i.e., $\mathcal{P}$ and $\mathcal{N}$). Given the binary nature of the variable of interest (i.e., $\mathcal{P}/\mathcal{N}$membership) to characterize, we learn a logistic regression model (Hosmer, Jr., Lemeshow, and Sturdivant 2013) since that is well suited to the task of modeling a binary dependent variable. It may be noted that we use *pseudo*-supervision in learning a logistic regression model since we use the current estimates of $\mathcal{P}$ and $\mathcal{N}$

[4]Source code available at https://github.com/stanleyts/ContentNContext

[5]Qadir et al. (2015) uses six tokens for creating contextual patterns, but we found five to be empirically sufficient.

as labelings, with $\mathcal{P}$ corresponding to label 1, and $\mathcal{N}$ to label 0. In particular, these labels derived from the current $\mathcal{P}/\mathcal{N}$estimates reflect the current belief, and are different from labeled information in supervised learning tasks where they reflect the ground truth.

The logistic regression model takes 5-tuples in the form of the four features along with the $\mathcal{P}/\mathcal{N}$membership, and learns a model that can quantify the $\mathcal{P}$-likeness of any match specified using the four features. In simple terms, the logistic regression model learns a weight w_f for each feature f and a bias b (together forming the model parameters) such that the $\mathcal{P}$-likeness of a match can be quantified using its features as:

$$\mathcal{P}\text{-likeness}(m) = \frac{1}{1 + e^{-(b + \sum_{f \in F} w_f \times f(m))}} \tag{5}$$

where F denotes the set of four features and $f(m)$ denotes the value of feature f for the match m. It may be noted that unlike supervised tasks, we do not distinguish between training and test sets. The model parameters are learned across matches in $\mathcal{N} \cup \mathcal{P}$, followed by a quantification of $\mathcal{P}$-likeness over all matches in $[(\mathcal{N} \cup \mathcal{P}) - \mathcal{M}(r_{seed}, \mathcal{D})]$.

At each iteration, the top-k $\mathcal{P}$-like matches from $[(\mathcal{N} \cup \mathcal{P}) - \mathcal{M}(r_{seed}, \mathcal{D})]$ are chosen for inclusion in the $\mathcal{P}$ set for the next iteration. All the seed regex matches (i.e., $\mathcal{M}(r_{seed}, \mathcal{D})$) remain within $\mathcal{P}$; this fixation reflects the assumption that the manually crafted regex is highly precise. All the remaining matches form the estimate of $\mathcal{N}$ for the next iteration. At each iteration, k is steadily increased, leading to a progressive enlargement of $\mathcal{P}$ with respect to $\mathcal{N}$ as indicated earlier. The usage of a simple model such as logistic regression along with avoidance of large values of k make the method reasonably robust to model overfitting.

Match Set Expansion Approach

We now outline the match set expansion approach in entirety in Algorithm 1. As described earlier, our approach uses a bootstrapped iterative approach to expand the set of matches from the initial set (i.e., $\mathcal{M}(r_{seed}, \mathcal{D})$) to progressively include matches from other regexes in the generalization space of r_{seed}. Lines 5-11 indicate the steps within each iteration, following the description presented earlier. While we omit describing the details again, it may be noted that the k within each iteration is enlarged by a fixed fraction (denoted by p) of the seed matches; this ensures that $\mathcal{P}$ increases within each iteration. A low-value of p adopts a cautious expansion approach, whereas a high-value of p leads to a more aggressive approach. The quantum of overall expansion may also be controlled by the number of iterations num.

Algorithm 1 MATCH-SET-EXPANSION

```
Input: $r_{seed}$, $\mathcal{D}$, $\mathcal{G}^d_{r_{seed}}$
Parameters: $p$, $num$
Output: $\mathcal{M}_{exp}$
 1: $\mathcal{P} \leftarrow \mathcal{M}(r_{seed}, \mathcal{D})$
 2: $\mathcal{N} \leftarrow \big(\cup_{r \in \mathcal{G}^d_{r_{seed}}} \mathcal{M}(r, \mathcal{D})\big) - \mathcal{M}(r_{seed}, \mathcal{D})$
 3: $k = |\mathcal{M}(r_{seed}, \mathcal{D})| \times p$
 4: repeat
 5:     Learn $L_\mathcal{P}$ and $R_\mathcal{P}$ using current $\mathcal{P}$
 6:     Use $L_\mathcal{P}$, $R_\mathcal{P}$, and current $\mathcal{P}$ and $\mathcal{N}$ to estimate the
        four-feature representation of all matches in $\mathcal{N} \cup \mathcal{P}$
 7:     Train a logistic regression model and estimate the
        model parameters
 8:     $top\text{-}k$ = top-$k$ $\mathcal{P}$-like non-seed matches using the lo-
        gistic regression model
 9:     $\mathcal{N} \leftarrow (\mathcal{N} \cup \mathcal{P}) - (top\text{-}k \cup \mathcal{M}(r_{seed}, \mathcal{D}))$
10:     $\mathcal{P} \leftarrow top\text{-}k \cup \mathcal{M}(r_{seed}, \mathcal{D})$
11:     $k \mathrel{+}= |\mathcal{M}(r_{seed}, \mathcal{D})| \times p$
12: until $num$ iterations are not completed
13: $\mathcal{M}_{exp} = \mathcal{P}$
```

Regex Recommender

Having built an expanded set of matches $\mathcal{M}_{exp}$, we would like to use that to construct a ranked list of generalized regexes from $\mathcal{G}^d_{r_{seed}}$ to cater to scenarios where the human user can choose regexes instead of simply using the expanded set of matches. Our idea is to choose regexes from $\mathcal{G}^d_{r_{seed}}$ that cover a lot of matches in $\mathcal{M}_{exp}$, while covering very few other matches, to reflect the estimation that $\mathcal{M}_{exp}$ is the set of *correct* matches of the intended entity. At the same time, however, we would also want to avoid a lot of very *similar* regexes at the top of the ordered output list. Consequently, we choose a popular diversity conscious ranking framework, Maximum Marginal Relevance (MMR) (Carbonell and Goldstein 1998), to model our regex recommender on. MMR offers a simple recipe to progressively choose items in a ranked output, which, adapted to our context, is as follows:

$$Nxt = \operatorname*{arg\,max}_{r \in \mathcal{G}^d_{r_{seed}} - \mathcal{R}} \big[\lambda \cdot Sim_1(r, \mathcal{M}_{exp}) - (1-\lambda) \cdot Sim_2(r, \mathcal{R})\big] \tag{6}$$

where $\mathcal{R}$ is the set of regexes already chosen for the output. λ is a weighting parameter controlling the trade-off between relevance and diversity, which we set to 0.7 as recommended in Carbonell and Goldstein (1998) to emphasize relevance over diversity. We start with $\mathcal{R} = \phi$ and progressively find the next result, Nxt, and add it to $\mathcal{R}$, forming an ordered result set in the process. $Sim_1(.,.)$ is a relevance function that estimates the *goodness* of r, whereas $Sim_2(.,.)$ feeds into a penalty term that ensures that regexes similar to already chosen ones are not picked, thus forming the diversity criterion. We model the Sim functions as follows:

$$Sim_1(r, \mathcal{M}_{exp}) = \frac{|\mathcal{M}(r, \mathcal{D}) \cap \mathcal{M}_{exp}|}{|\mathcal{M}_{exp}|} \tag{7}$$

$$Sim_2(r, \mathcal{R}) = \frac{|\mathcal{M}(r, \mathcal{D}) \bigcap (\cup_{r' \in \mathcal{R}} \mathcal{M}(r', \mathcal{D}))|}{|\mathcal{M}(r, \mathcal{D}) \bigcup (\cup_{r' \in \mathcal{R}} \mathcal{M}(r', \mathcal{D}))|} \tag{8}$$

Informally, $Sim_1(.,.)$ measures the recall of r against $\mathcal{M}_{exp}$, whereas $Sim_2(.,.)$ is modeled as the Jaccard similarity between the matches of r and the collective matches across the regexes in $\mathcal{R}$. The result set cardinality can be controlled easily by stopping after choosing the desired number of regexes in the output.

Computational Costs

In contrast to online tasks such as IE that benefit from real-time responses, the match set expansion and regex recommender tasks are offline tasks that enable fine tuning the rule-based IE system at design time. However, it may be noted that such a setting assumes that the character of document data that the system needs to be tested over remains similar in character to that of the reference document dataset used in the match set expansion and regex recommender tasks. We now briefly discuss the computational costs of our methods. With typical seed regexes only containing up to tens of regex units, computing the regex set $\mathcal{G}^d_{r_{seed}}$ takes up to a few thousands of operations, making it a very lightweight step. Identifying matches (and associated left/right contexts) for all regexes in $\mathcal{G}^d_{r_{seed}}$ in $\mathcal{D}$, denoted by $\mathbb{M}$ ($= \cup_{r \in \mathcal{G}^d_{r_{seed}}} \mathcal{M}(r, \mathcal{D})$), takes time linear in the size of the document corpus with the aid of an automaton. Learning the language models over the fixed-size left and right contexts of matches in $\mathbb{M}$ is evidently linear in $|\mathbb{M}|$; so is the

ENTITY	r_{seed}	TASK	$\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{M}(r_{seed}, \mathcal{D})\rvert$
DATE	`\d{2}/\d{2}/\d{2}`	DATE$_{\text{MIDEAST}}$	talk.politics.mideast	1k	7
		DATE$_{\text{WEBKB}}$	WebKB	2k	86
		DATE$_{\text{ENRON}}$	Enron	100k	25654
PHONE NUMBER	`\(\d{3}\)\d{3}-\d{4}`	PHONE$_{\text{FORSALE}}$	misc.forsale	1k	88
COURSE NUMBER	`CS\d{3}`	COURSE$_{\text{WEBKB}}$	WebKB	2k	1348
PHONE NUMBER	`\d{3}-\d{3}-\d{4}`	PHONE$_{\text{ENRON}}$	Enron	100k	28994

Table 2: Extraction Tasks Overview. The seed regex for DATE is common on all corpora.

TASK	PRECISION				RECALL				F-SCORE				
	FREQ	GM$_{10}$	GM$_{1\%}$	OURS	$\mathcal{M}(r_{seed}, \mathcal{D})$	GM$_{10}$	GM$_{1\%}$	OURS	$\mathcal{M}(r_{seed}, \mathcal{D})$	FREQ	GM$_{10}$	GM$_{1\%}$	OURS
DATE$_{\text{MIDEAST}}$	0.009	0.009	0.045	**0.351**	0.072	0.072	0.072	**0.948**	0.135	0.017	0.016	0.055	**0.513**
DATE$_{\text{WEBKB}}$	0.032	0.186	0.479	**1.000**	0.251	0.436	0.330	**0.857**	0.402	0.063	0.261	0.391	**0.923**
DATE$_{\text{ENRON}}$	0.038	**0.982**	0.965	0.913	0.608	0.619	0.624	**0.887**	0.756	0.072	0.759	0.758	**0.900**
PHONE$_{\text{FORSALE}}$	0.462	0.224	0.517	**0.984**	0.169	0.257	0.236	**0.483**	0.289	0.621	0.239	0.324	**0.648**
COURSE$_{\text{WEBKB}}$	0.070	0.672	0.633	**0.994**	0.342	0.348	0.349	**0.855**	0.509	0.131	0.459	0.450	**0.919**
PHONE$_{\text{ENRON}}$	0.118	**0.977**	0.970	0.766	0.684	0.687	0.687	**0.830**	0.812	0.211	**0.807**	0.805	0.796

Table 3: Comparative evaluation of accuracies of match set expansion algorithms

estimation of context features using the learned language models. The Levenshtein automaton from Schulz and Mihov (2002) makes $\mathcal{E}_t(m)$ estimation over all matches in $\mathbb{M}$ linear in $|\mathbb{M}|$. The linearity in $|\mathbb{M}|$ holds for the logistic regression training as well as the partial sort (Martınez 2005) in Line 8; note that k is a fraction of the size of the seed regex matches, which is much smaller than $\mathbb{M}$. With all the intra-iteration steps being linear, the sequence of iterations for the match set expansion is in $\mathcal{O}(num \times |\mathbb{M}|)$. Given the regex matches identified in the match expansion phase, the final regex recommender phase is much simpler and is in $\mathcal{O}(|\mathcal{G}^d_{r_{seed}}|)$ for fixed output sizes. We search over the generalization space $\mathcal{G}^d_{r_{seed}}$ much like in Murthy, P., and Deshpande (2012); heuristically limiting the size of $\mathcal{G}^d_{r_{seed}}$ by excluding unpromising generalization sub-trees would help scale the search to higher values of d.

Experiments

Experimental Setup

Extraction Tasks: We perform our empirical evaluation on a variety of extraction tasks over multiple real-world document corpora as shown in Table 2. The talk.politics.mideast and misc.forsale corpora are taken from the 20 Newsgroups dataset[6], whereas the Enron corpus is a random subset of 100k documents from the Enron Email Dataset[7]. The WebKB corpus[8] is another popular document dataset.

Baselines: Our empirical evaluation uses three baseline methods. A simple baseline, FREQ, labels all matches by regexes in $\mathcal{G}^d_{r_{seed}}$ as correct. GM$_{10}$ is the approach from Gupta and Manning (2014) using the recommended expansion rate of 10 instances per iteration. GM$_{1\%}$ is the adaptation of the same method, using the same expansion rate as our method, which is 1%. As stated earlier, the method from Gupta and Manning (2014) requires a set of positive instances to start the learning; we set it to $\mathcal{M}(r_{seed}, \mathcal{D})$.

Gold Standard: Our evaluation requires instances labeled as correct, for each extraction task. Since manual annotation of each token over thousands of documents is prohibitively expensive, we got labelings done by human annotators in two phases. In the first phase, our annotators labeled every regex in $\mathcal{G}^d_{r_{seed}}$ as either correct or incorrect with respect to the entity being considered. In the second phase, all matches of these correct regexes were shown to the annotators for getting labelings at the level of matches. Regexes labeled correct in the first phase and then found to have incorrect matches were specialized to regex(es) that span only correct matches. Thus, all correct regexes match only correct instances; these regexes are listed in the supplementary material. The matches by such correct regexes are used as the gold standard instances for evaluation purposes. The annotators also labeled the instances learned by the baselines, and the correct ones were added to the gold standard instances.

Parameters used: Our method uses three parameters: d, num, and p. We set these to 4, 150, and 1%, unless otherwise stated. We separately study the performance of our method across variations in these parameters.

Comparison of Match Set Expansion Stage

Table 3 illustrates the results from our comparative evaluation. As outlined earlier, we are interested in a good trade-off between precision and recall, and thus, a high F-score. FREQ peaks on recall with very low precision, whereas just considering $\mathcal{M}(r_{seed}, \mathcal{D})$ as the results results in the reverse behavior; since these were observed to achieve 1.0 on recall and precision respectively, those metrics are not shown in the table for brevity. Our method (**OURS**) beats the baselines on five of six extraction tasks, posting F-scores as much as 0.9 on three of them. On the sixth, it is seen to trail the leading method very closely. This illustrates the effectiveness of our modeling and establishes our method as the preferred method for the task.

Discussion: The baseline approach from Gupta and Manning (2014) identifies patterns of context words to characterize correct matches. This coarse-grained pattern-based framework does not allow discriminating between matches that bear context differences, but fall within the same pattern(s). Our context modeling is more fine-grained, with scoring depending on all words in the context window. We believe the enhanced effectiveness of our method is explained by the nature of our context modeling and blending it with content modeling in a logistic regression framework, as against just averaging the scores as Gupta and Manning (2014).

[6]http://qwone.com/~jason/20Newsgroups/

[7]https://www.cs.cmu.edu/~./enron/

[8]http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-51/www/co-training/data/

Parameter Sensitivity Analysis

We now study the trends across variations on the various parameters, varying one parameter at a time, with others fixed.

Generalization Depth (d): The space of regexes and their matches, and consequently the search space of candidates from which our result set is constructed, increases with generalization depth. Figure 3a plots the F-score over varying generalization depths, showing that the F-score is quite stable over varying values of d, with a notable increase in the COURSE$_{\text{WebKB}}$ F-score when d is increased to 4. This indicates that our method is able to leverage the increased search space to identify more of the correct instances.

Number of Iterations (num): Figure 3b indicates the F-score trends over varying number of iterations. The high-level trend is that of increasing F-score with increasing number of iterations, until a point at which incorrect matches start getting labeled as correct resulting in a snowball effect that incentivizes labeling incorrect matches as correct. The peak F-score is reached in the 150-300 iterations window, whereas the DATE$_{\text{MidEast}}$ is seen to peak much earlier. This was found to be so since most correct matches were learned within the first few iterations, largely due to the very small number of seed regex matches (Ref: Table 2) that encompasses very little evidence about the correct entity instances for the bootstrapped approach to exploit.

Expansion Rate (p): Figure 3c shows how our method fares over iterations for expansion rates of 0.5%, 1.0%, 1.5%, 2.0%, 5.0%, and 7.5% on the COURSE$_{\text{WebKB}}$ task. Other tasks were seen to show similar trends, and have been omitted for brevity. This parameter is semantically similar to the number of iterations parameter in that it determines the amount of learning; p determines the quantum of learning within an iteration, whereas num controls the overall quantum across iterations. Further, trends on p convey trends on k as well, k being a function of p (Algo 1 Lines 3, 11). The variations between expansion rates starts declining at low values, indicating that it is best to select an expansion rate lesser than 2%.

Discussion: It is notable from Figure 3b that F-score > 0.9 - i.e., high precision along with high recall - was achieved on all but one task (PHONE$_{\text{ForSale}}$'s peak F-score was 0.85) for some value of num, indicating that a heuristic to identify an ideal task-customized stopping point would lead to a virtually thorough solution to the match set expansion task. Notwithstanding the possibility that such an ideal stopping point heuristic might be elusive, the high F-score peaks are illustrative of the effectiveness of our modeling.

Regex Recommender Results

Table 4 shows the regex output of our method. The regexes may be seen as largely meaningful; for example, our method is able to generalize the two digit day and month fields of the regex to cover single digit values as well. The seed for the PHONE$_{\text{ForSale}}$ task was meant to recognize only those phone numbers that have the area code within brackets; this bracket requirement is seen to be relaxed in the first and the third output regexes. The generalizations to encompass alphanumeric phone numbers outlined in the Introduction section were seen to appear slightly lower down in the list (we restricted the table to the top-3 regexes for brevity). The COURSE$_{\text{WebKB}}$ task seed regex which matches only CS courses are seen to be generalized to regexes that capture other department courses. Our generalization method is limited to make generalizations one regex unit at a time, and thus needs to generalize on the pair brackets - for example, the $\backslash($ and $\backslash)$ in the PHONE$_{\text{ForSale}}$- separately. While such issues can be solved by engineering the generalization space with domain knowledge to force synchronized generalization of corresponding pairs of units, our method's generalizations are broadly in the correct direction nevertheless.

Conclusions and Future Work

We considered the task of enhancing the coverage of entity instances for regex-based rule IE systems, where the only user input is in the form of a high precision seed regex. In particular, we attempt to perform the learning in the absence of instance supervision (besides that implicitly provided by the seed regex) as used in previous work for the task. We designed an iterative method that leverages signals from the content as well as the lexical context of the matches of the seed regex in correlation with the generalization space of the seed regex, in identifying newer instances of the entity. Our empirical analyses over multiple real-world document corpora confirm the effectiveness of our method over techniques from literature, in identifying correct instances more accurately.

Our method assumes that the input seed regex is 100% precise; it would be interesting to study the effectiveness of our method with this assumption relaxed. The natural next step for this work is to broaden the generalization approach to address higher-level IE tasks such as rule-driven relationship discovery where the seed patterns would straddle token and even sentence boundaries. A future direction for extending this algorithm is to enrich the logistic regression model with features exploiting context pattern motifs (Gupta and Manning 2014), distributed representation of words (Mikolov et al. 2013), and the graph nature of the generalization space. We are considering expanding this framework to cover simple entities that span multiple tokens involving POS tags, such as noun phrases comprising one or more contiguous nouns.

Acknowledgments

Special thanks to Prof. Hema Murthy (Dept. of CSE, IIT Madras) for providing us with server facilities. We also thank the anonymous reviewers for their insightful comments.

References

Babbar, R., and Singh, N. 2010. Clustering based approach to learning regular expressions over large alphabet for noisy unstructured text. In *Proceedings of the Fourth Workshop on Analytics for Noisy Unstructured Text Data*, AND 2010, 43–50. Association for Computing Machinery.

Bartoli, A.; Davanzo, G.; De Lorenzo, A.; Medvet, E.; and Sorio, E. 2014. Automatic synthesis of regular expressions from examples. *IEEE Computer* 47(12):72–80.

Bartoli, A.; De Lorenzo, A.; Medvet, E.; and Tarlao, F. 2016. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering* 28(5):1217–1230.

Bartoli, A.; De Lorenzo, A.; Medvet, E.; and Tarlao, F. 2017. Active learning of regular expressions for entity extraction. *IEEE Transactions on Cybernetics* PP(99):1–14.

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 ACM International Conference on Information and Knowledge Management*, CIKM 2011, 1285–1294. Association for Computing Machinery.

Carbonell, J. G., and Goldstein, J. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR 1998, 335–336. Association for Computing Machinery.

Chiticariu, L.; Li, Y.; and Reiss, F. R. 2013. Rule-based information extraction is dead! long live rule-based information extraction systems! In *Proceedings of the 2013 Conference on Empiri-*

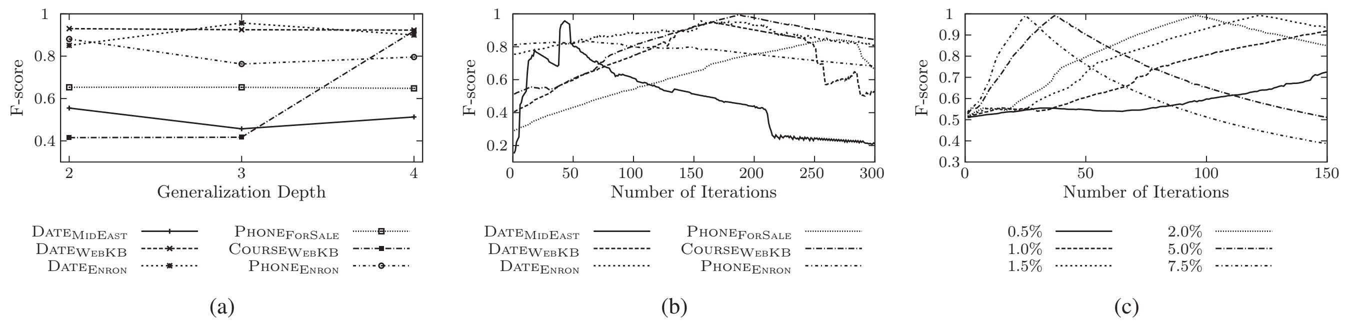

Figure 3: Effect of (a) generalization depth, (b) number of iterations, and (c) expansion rate on the F-score of $\mathcal{M}_{exp}$

TASK	DATE$_{MidEast}$	DATE$_{WebKB}$	DATE$_{Enron}$
r_{seed}	\d{2}/\d{2}/\d{2}	\d{2}/\d{2}/\d{2}	\d{2}/\d{2}/\d{2}

$\mathcal{R}$	\d{**1**,2}/**?**\d{**1**,2}/**?**\d{2}
	\d{**1**,2}/**?**\d{2}**[-.\:/*|]**\d{2,**3**}
	\d{2}/**?**\d{2}**\W**\d{2}
		\d{**1**,2}/\d{**1**,2}/\d{2}
		\d{2}/\d{**1**,2}/\d{2}
		\d{**1**,2}/\d{2}/\d{2}
			\d{**1**,2}**[-.\:/*|]**\d{**1**,2}**[-.\:/*|]**\d{2}
			\d{**1**,2}/\d**.**/**?**\d{2}
			\d{**1**,2}/\d{**1**,2}/\d{**1**,2}

TASK	PHONE$_{ForSale}$	COURSE$_{WebKB}$	PHONE$_{Enron}$
r_{seed}	\(\d{3}\)\d{3}-\d{4}	CS\d{3}	\d{3}-\d{3}-\d{4}

$\mathcal{R}$	\(**?**\d{3}**\W**\d{3}**\W**\d{4}
	\(\d{3}\)**\W?**\d{3}**\W**\d{4}
	\(**?**\d{**1**,3}**\W**\d{3}-\d{4}
		C**?[a-zA-Z]**{1,**2**}\d{3}
		[a-zA-Z]{1,**2**}S**?**\d{3}
		CS**\w**{1,**3**}\d**\w**
			\d{3}**\W**{1,**2**}\d{3}**[-.\:/*|]**\d{4}
			\d{**1**,3}-**?**\d{3}-**?**\d{4}
			\d{3}**\W**{1,**2**}\d{3}-\d{**3**,4}

Table 4: Top-3 recommended regexes. The generalized units are boldfaced and underlined.

cal Methods in Natural Language Processing, EMNLP 2013, 827–832. Association for Computational Linguistics.

Gupta, S., and Manning, C. D. 2014. Improved pattern learning for bootstrapped entity extraction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, CoNLL 2014, 98–108. Association for Computational Linguistics.

Gupta, S., and Manning, C. D. 2015. Distributed representation of words to guide bootstrapped entity classifiers. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT 2015, 1215–1220. Association for Computational Linguistics.

Hosmer, Jr., D. W.; Lemeshow, S.; and Sturdivant, R. X. 2013. *Applied Logistic Regression*. Wiley Series in Probability and Statistics. Hoboken, New Jersey: John Wiley & Sons, Inc., third edition.

Jurafsky, D., and Martin, J. H. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall Series in Artificial Intelligence. Upper Saddle River, New Jersey: Prentice Hall, 1st edition.

Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics – Doklady* 10(8):707–710.

Li, Y.; Krishnamurthy, R.; Raghavan, S.; Vaithyanathan, S.; and Jagadish, H. V. 2008. Regular expression learning for information extraction. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, EMNLP 2008, 21–30. Association for Computational Linguistics.

Martınez, C. 2005. Forty years of quicksort and quickselect: a personal view. In *Algorithms Seminar, 2002–2004*, 101–104. INRIA.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems*, NIPS 2013, 3111–3119. Curran Associates, Inc.

Murthy, K.; P., D.; and Deshpande, P. M. 2012. Improving recall of regular expressions for information extraction. In *Proceedings of the 13th International Conference on Web Information Systems Engineering*, WISE 2012, 455–467. Springer-Verlag.

Qadir, A.; Mendes, P. N.; Gruhl, D.; and Lewis, N. 2015. Semantic lexicon induction from twitter with pattern relatedness and flexible term length. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI 2015, 2432–2439. AAAI Press.

Riloff, E., and Jones, R. 1999. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, AAAI/IAAI 1999, 474–479. AAAI Press / The MIT Press.

Sarmento, L.; Jijkoun, V.; de Rijke, M.; and Oliveira, E. 2007. "more like these": Growing entity classes from seeds. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*, CIKM 2007, 959–962. Association for Computing Machinery.

Schulz, K. U., and Mihov, S. 2002. Fast string correction with levenshtein automata. *International Journal on Document Analysis and Recognition* 5(1):67–85.

Thelen, M., and Riloff, E. 2002. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, EMNLP 2002, 214–221. Association for Computational Linguistics.