

How to Invest my Time: Lessons from Human-in-the-Loop Entity Extraction

Shanshan Zhang Lihong He Eduard C. Dragut Slobodan Vucetic
Department of Computer and Information Sciences
Temple University, PA, USA
{zhang.shanshan, lihong.he, edragut, vucetic}@temple.edu

ABSTRACT

Recognizing entities that follow or closely resemble a *regular expression* (regex) pattern is an important task in information extraction. Common approaches for extraction of such entities require humans to either write a regex recognizing an entity or manually label entity mentions in a document corpus. While human effort is critical to build an entity recognition model, surprisingly little is known about how to best invest that effort given a limited time budget. To get an answer, we consider an iterative *human-in-the-loop* (HIL) framework that allows users to write a regex or manually label entity mentions, followed by training and refining a classifier based on the provided information. We demonstrate on 5 entity recognition tasks that classification accuracy improves over time with either approach. When a user is allowed to choose between regex construction and manual labeling, we discover that (1) if the time budget is low, spending all time for regex construction is often advantageous, (2) if the time budget is high, spending all time for manual labeling seems to be superior, and (3) between those two extremes, writing regexes followed by manual labeling is typically the best approach. Our code and data is available at <https://github.com/nymph332088/HILRecognizer>.

CCS CONCEPTS

• **Computing methodologies** → **Information extraction; Active learning settings.**

KEYWORDS

entity extraction, regex, neural networks, human-in-the-loop

ACM Reference Format:

Shanshan Zhang Lihong He Eduard C. Dragut Slobodan Vucetic. 2019. How to Invest my Time: Lessons from Human-in-the-Loop Entity Extraction. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA*. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3292500.3330773>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6201-6/19/08... \$15.00

<https://doi.org/10.1145/3292500.3330773>

1 INTRODUCTION

Entity extraction occupies a prominent place in information retrieval. Named entity recognition, the most recognized entity extraction subtask, seeks to automatically identify substrings that represent specific people, locations, events, or organizations. Beside named entities, there is a large class of entities that are not “named,” such as expressions of *dates, times, email addresses, phone numbers, currencies, credit card numbers, social security numbers, measurements, and object properties*. These types of entities can often be expressed or approximated by a regular expression (regex) and are the focus of this work. They have drawn interest from several communities, including NLP [9, 17, 21], databases [17], data mining [1–4], and life sciences [20, 27].

Two common approaches to recognize regex-like entities are to (1) manually create a regex and (2) train a machine learning model, both of which have their advantages and disadvantages. Most programmers are familiar with regex and can write reasonably accurate entity recognizers with relatively little effort, without the need to use machine learning software. However, once a regex goes beyond a level of complexity, writing it requires a lot of time and experience and results in brittle recognizers, leading to a saying “Now you have two problems” [14]. Even a seemingly simple task of recognizing an email address apparently requires 6,500 characters [19]!

Machine learning (ML) approaches attempt to either infer a regex or create a regex-oblivious model. Regex learning approaches [3, 4, 17, 21, 26] require a set of substrings and focus on constructing a short regex recognizing the substrings. Similarly to manual construction of a regex, the existing approaches quickly end up in very long and brittle formulas and are not commonly used in practice [3]. In regex-oblivious approaches, the objective is to train a model such as a neural network (NN) from labeled substrings [28]. An advantage is that labeling does not need programming expertise. A disadvantage is that this approach requires a large set of labeled examples. In the rest of the paper, when we refer to ML methods we refer to the regex-oblivious approach.

In our recent work we proposed a human-in-the-loop (HIL) framework [28] that uses human effort to both write a regex and to manually label the documents. As will be elaborated in the methodology, a regex is used to scan a document corpus and produce weak labels to pretrain an NN. Then, manually labeled substrings are used to fine-tune the network. The results showed that fine tuning a pretrained NN is superior to training it from scratch. Thus, the results indicate that writing a regex before manual labeling is highly desirable. However, this conclusion does not take into account the time needed to create a regex and regex writing expertise. In this study, we consider the problem from a practical perspective, where a human is given

a fixed amount of time to interact with the ML system for entity recognition.

Time and expertise are critical factors in a HIL ML system such as the one we consider. Let us look at a potential real-life scenario. Let us imagine a data scientist Amy, faced with a challenge of extracting publication dates from a corpus of hundreds of thousands of news articles crawled from the Web given a one hour deadline. Just finding a single mention of date would include a lengthy scanning of articles and would make the task infeasible. Alternatively, Amy may remember that all articles are published in 2019 and write simple regex 2019 to identify date mentions with a high recall but low precision. Then, she glances over the extracted substrings surrounding mentions of 2019 and does one of the two things: (1) starts labeling the dates or (2) realizes that all dates seem to follow a particular pattern and proceeds to write a regex. Even if Amy starts writing a regex and proceeds with labeling, another question is whether she should spend a lot of time trying to improve the regex or stop and start with manual labeling. We are not aware of published results that may inform Amy how to efficiently invest her time. We set to gain insight into this problem in this work.

We make the following contributions in this paper:

- We propose a framework that recognizes an entity through character-wise classification. This is in contrast to our previous work [28] where we developed a HIL framework that detects if a text passage contains an entity through sequence-wise classification.
- We propose an algorithm for active selection of substrings.
- We perform a thorough characterization of the proposed framework on 5 entity recognition tasks.
- We perform a small-scale user study to obtain insight into the trade-offs between spending time to write regular expressions and spending time to manually label text fragments.

2 RELATED WORK

This section briefly reviews several lines of research we deem to be the most relevant to our work.

Regex Refinement and Inference. This line of research aims to (partially) automate regex construction. One research direction focuses on improving the precision and recall of initial regexes by identifying true or false matches [17, 21, 26]. They start from a user defined regex with either high recall, but low precision, [17] or high precision, but low recall [21], and search for an improved regex. In either case users need to create true positive and negative instances in the matching set of the initial regex. Some works seek to reduce human labeling efforts in this process, e.g., new matches of candidate regexes are automatically grouped into positives and negatives by comparing their context similarities to those of the generalized regex [26]. A different line of work attempts to induce regexes from positive and negative sample strings [6, 10, 11]. They do not require an input regex. The most recent approach uses genetic programming [3, 4]. All these efforts require human input, such as a set of examples, or an initial high precision regex, or manually labeled negative and positive matches of regexes. Human effort had not been explicitly quantified in this line of work.

Human Annotation Effort: An important area of research in ML seeks to reduce the human annotation effort, both in scale (i.e.,


amount of labeled instances) and form (e.g., weak labels). This is a broad area of research and we limit our coverage to the entity mining literature. One line of work uses solely weak labels to train NER models. Distant-LSTM-CRF [13], AutoNER [24], and SwellShark [12, 23] are examples of approaches in this category. String matching and (expert) rules are common means to generate weak labels.

Active learning aims to smartly involve human judgment in the training of a model. In NER, this follows a 2-iteration process. In the first iteration, the system samples sentences according to some heuristics, asks users to annotate them, and trains an initial NER model. In the second iteration, the system iteratively recruits unlabeled sentences by a scoring function for human annotations. The work in [7] uses the longest sentence selection heuristic in the first iteration and 12 scoring functions in the second iteration. In [25], a bag of initial models is trained with a hand-labeled seed data set. They use the disagreement among the bag of models measured by KL-divergence and f-complement for scoring.

We are not aware of any work in this space that couples weak supervision with active learning, as we propose in this work. In addition, although weak labeling is cheaper than standard labeling it still incurs human cost. This is largely ignored in the literature. We consider these factors in our framework.

3 PROPOSED FRAMEWORK

Problem definition Given a corpus of documents \mathcal{D} and an entity type \mathcal{E} , the objective is to create an accurate entity extractor for \mathcal{E} while limiting the total human effort within T minutes.

We give an overview of our solution in Figure 1 (left). The input of our framework is a document corpus \mathcal{D} and an entity type \mathcal{E} (e.g. phone number). It outputs an entity extractor. The proposed framework has several modules. The first module selects candidate substrings that are likely to contain an entity mention. Given the set of candidate substrings, the second module is responsible for weak labeling them. This is accomplished with a regex. The third module trains an NN for entity extraction using a set of labeled substrings. The fourth module selects a subset of substrings for human labeling. After the selected substrings are manually labeled by a human, they are fed back to module 3 for fine-tuning of the NN. We highlight the places where human effort is needed with a clock . A human expert invests time in three ways in the system. She (1) creates a regex for module 1 that selects candidate substrings with high recall and allowing for low precision, (2) creates a regex for module 2 that weakly labels the candidate substrings with relatively high precision and recall, and (3) manually labels an unlabeled substring in module 4 by highlighting substrings corresponding to an entity mention. We provide technical details in the remainder of this section.

3.1 Selection of Candidate Substrings

The core objective of our framework is to minimize human effort needed to build an entity extractor. As will be specified later in the paper, the user is expected to scan the corpus and recognize entity mentions. In a typical entity extraction scenario, the number of entity mentions in a corpus is relatively small compared to the total size of the corpus. Thus, it is very helpful to users if they automatically exclude portions of the corpus that do not contain entity mentions. Our main observation is that it is often possible to specify a simple

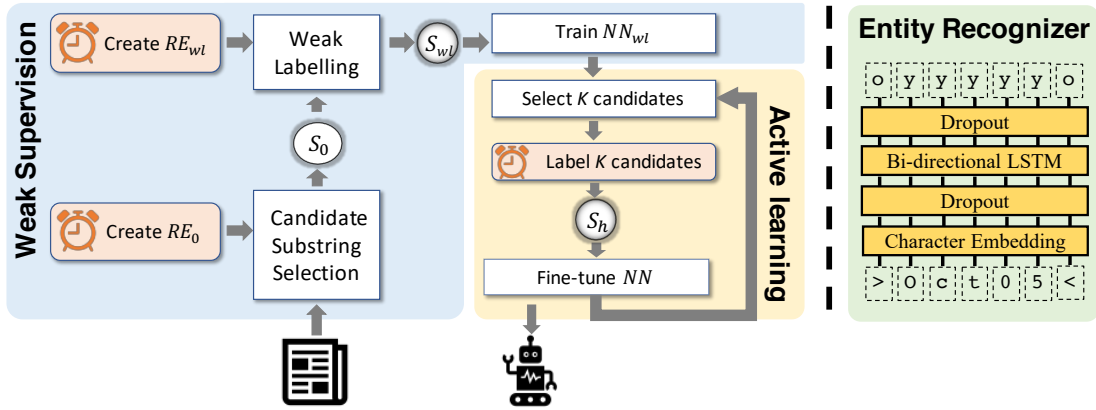


Figure 1: Overview of the solution framework(left) and deep learning architecture (right) used to train entity recognizer.

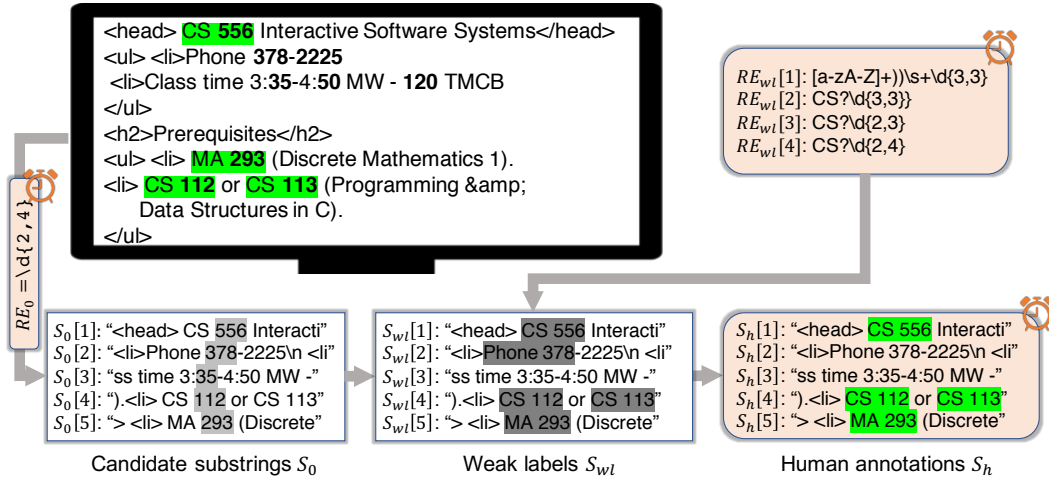


Figure 2: An example of course number recognition on a sample HTML document. Only 5 of the 9 S_0 candidate substrings are shown

regex that recognizes entities with a high recall (i.e., including most of the entities) and potentially a low precision (i.e., allowing a high fraction of false positives). For example, if our objective is to extract course numbers, a regex that recognizes two or more digits (e.g., $\backslash d\{2,4\}$) is likely to capture most of the course mentions in the U.S universities. It is evident that such regex also recognizes many strings that are not related to course numbers, thus resulting in low precision. We denote a regex used for recognition of candidate entities as RE_0 and call it the *candidate regex*.

For each match of the candidate regex in the text, we generate the *candidate substring* by expanding it with $\frac{L}{2}$ characters before and after it. L is selected to guarantee that the string fragment includes the entity mention and has sufficient context surrounding the mention. This helps determine if a substring contains the entity. We denote the set of candidate substrings obtained in this way from corpus \mathcal{D} by $S_0 = \{s_1, s_2, \dots, s_n\}$, where n is the number of strings RE_0 matches.

We need to emphasize that, in our framework, the role of RE_0 is to remove portions of the corpus that clearly do not contain entity

mentions. It is not critical that the precision of the candidate substrings is high. Instead, the emphasis is on encouraging the user to quickly come up with a simple RE_0 that has a good recall.

3.2 Weak Labeling

Given the set of candidate substrings S_0 , our objective is to label and use them to train an NN model for entity extraction. The NN has to predict each character in a candidate substring as either positive (belonging to an entity) or negative (not belonging to an entity). Given those character-wise predictions, the entity is identified as a string of consecutive positively labeled characters. Thus, it is possible that we may extract multiple entities from a single candidate substring.

A straightforward approach for labeling S_0 strings is to ask a user to manually label some or all of its substrings. As will be shown in the experimental results, this approach is inefficient when S_0 is very large. Instead, we propose a *weak labeling* approach aided by regex. In particular, we allow the user to provide a regex with a moderately high precision on S_0 . If an entity is well studied (e.g.,

date, email address), it is possible that one may find a good regex quickly by searching the web. Otherwise, it is assumed that the user is experienced enough with regexes and able to come up with a good regex in a reasonable amount of time. To aid the user, we can load a random subset of S_0 substrings into a freely available regex testing and debugging tool such as <https://regex101.com> [17]. We denote the resulting regex by RE_{wl} , where the subscript wl is an abbreviation for weak labeling.

Given RE_{wl} , we can automatically weakly label all the substrings in S_0 . S_{wl} denotes the resulting weakly labeled data set. We use S_{wl} to train an NN NN_{wl} . We expect that the recall and precision of NN_{wl} is comparable to RE_{wl} . The benefit of training NN_{wl} compared to directly using RE_{wl} , which is the traditional approach for entity extraction, is that NN_{wl} can be further fine-tuned and improved once manually labeled substrings become available.

3.3 Entity Recognizer

Given the set of labeled candidate substrings in S_{wl} , the next objective is to train an NN that classifies each character within a candidate substring. Suitable NN architectures include but are not limited to BiLSTM + CNNs [8], BiLSTM + CRF [16], and BiLSTM + CNNs + CRF [18]. In this study we use a BiLSTM + CNNs architecture illustrated in Figure 1 (on the right).

For a candidate substring with L characters, an embedding layer is used to map the l^{th} character ($l = 1, 2, \dots, L$) into a real valued vector e^l , where $e^l \in \mathbb{R}^p$ and p is the size of character embedding. Two or more Bidirectional LSTM (BiLSTM) layers are used to generate hidden vectors at each position l . One layer of BiLSTM contains two stacks of regular LSTM cells. The forward LSTM cells process the input string from the beginning to the end, while the backward LSTM cells go from the end to the beginning.

The loss function for the i^{th} string at the l^{th} position is defined as the cross entropy function. To train the deep learning model, we average the losses from all characters in the training set. We add a dropout layer after the embedding layer and each of the BiLSTM layers to avoid overfitting. To predict labels in a string during testing we assign each character to the class with the larger probability. We denote the vector of prediction of s_i as \hat{y}_i .

3.4 Fine-Tuning with Human Labels

The benefit of training an NN on weakly labeled data is that it can be fine-tuned and improved using manually labeled data. Assume we already trained NN_{wl} . The next questions are how many candidate substrings to manually label and how to select them from S_0 . In our framework, we ask the user to label K candidate substrings and then fine-tune the NN. We repeat the manual labeling and fine-tuning process until reaching the desired accuracy or the time limit.

The baseline approach to selection of substrings to be labeled is to select K substrings at random from S_0 . We refer to this selection algorithm as the **Random Querying (RQ)**. A more sophisticated approach is to use active learning, which attempts to select K substrings that result in the fastest increase in accuracy. Among the many active learning algorithms proposed in the ML literature [15], the ones based on the uncertainty principle have been the most successful over a large range of application. In particular, the examples on which the current predictor is more uncertain are more likely to

be selected. If a sigmoid neuron is used as output of an NN, we can interpret its output for the l^{th} character of string s as class probability, $p(y_l|s_l)$. The uncertainty of the prediction of the l^{th} character is defined as entropy $E(s_l) = -\sum_{k=0,1} -p(y_l = k|s_l) \log p(y_l = k|s_l)$. Higher entropy indicates high uncertainty.

To select the most uncertain subsequences, we need to aggregate the entropy over each subsequence. We denote uncertainty of a subsequence as $E(s)$. We consider several options for aggregation, e.g., averaged entropy, maximum entropy, and maximum entropy over a window (it is reported superior in [7]). While the differences are not large, we experimentally observed that maximum entropy is slightly better than the alternatives. We refer to the selection algorithm that picks the K most uncertain subsequences as the **Max Entropy (ME)**. When using the ME, it is possible that the most uncertain K substrings are highly redundant. Inspired by the idea of pre-clustering before active learning [22], we consider an alternative that first selects $M > K$ substrings from S_0 at random and then uses ME to select the most uncertain K substrings. We refer to this selection algorithm as the **Random then Max Entropy (RME)**.

3.5 Summary of the Framework

In Figure 2, we take course number as a running example and illustrate the intermediate data generated along the steps of our framework. We give the details in the following subsections. One notices that the user is involved in 3 steps of the algorithm: (1) creating candidate regex RE_0 , (2) creating weak labeling regex RE_{wl} , and (3) labeling candidate substrings. The total human effort is a sum of the efforts on those 3 steps. The effort for creation of RE_0 is assumed to be significantly smaller than for the other 2 steps. If we are given the time budget for human effort and assuming that the time to create RE_0 is negligible, an open question is how should the time be split between steps (2) and (3). We design our experiments to gain insight into the trade-offs between spending time to create a good regex and to manually label the candidate substrings.

The proposed framework also allows skipping one or more of the 3 steps. For example, instead of step (1), we can create the candidate substrings from all or from randomly selected substrings of length L . We can also decide to skip step (2) and train the first NN on the first K manually labeled candidate substrings selected from S_0 at random. We can refer to such an approach as the *cold start*. Finally, we can decide to skip step (3). In this case, we can decide to directly use RE_{wl} for entity extraction. We will evaluate all those scenarios in the experimental studies.

4 EXPERIMENTAL DESIGN

In this section we describe the entity extraction tasks we created to evaluate our framework and perform user studies.

4.1 Entity Recognition Tasks

We consider 5 entity recognition tasks in our experimental study:

- **DATETIME** recognition: we downloaded HTMLs of 6,000 English news articles published from August 24 - 30, 2017. They are randomly selected from 0.6 million articles in the Kaggle dataset¹. The task is to extract datetime in 2017 from the source HTML.

¹<https://www.kaggle.com/therohk/global-news-week>

Table 1: Summary of documents in the 5 entity recognition tasks.

| Domain | $ \mathcal{D} $ | Doc avg length (chars) | Number of entities in \mathcal{D} | Annot time (hrs) | Rate (ms/ch) | Entity avg length (chars) |
|---------------------|-----------------|---------------------------|--|---------------------|-----------------|------------------------------|
| DATETIME | 6,000 | 137,379 | 1,399* | 22 | 127 | 21.63 |
| BILLDATE | 600 | 27,518 | 3,085 | - | 100** | 12.63 |
| EMAILADDRESS | 602 | 1,284 | 2,206 | 16 | 74.5 | 21.92 |
| COURSENUMBER | 600 | 4,586 | 4,588 | 60 | 78.5 | 6.46 |
| PHONENUMBER | 3149 | 2674 | 2,018 | 150 | 65.9 | 13.64 |

* The number of entities in 6,000 strings in S_0 , one string per document.

** A reasonable guess of the human annotation rate.

- **BILLDATE** recognition: we downloaded 600 OCR scanned U.S. Congress bills². The task is to extract dates from the bills.
- **EMAILADDRESS** recognition: the task is to extract email addresses from 602 emails in the publicly available Enron email data set³.
- **COURSENUMBER** recognition: we downloaded 600 HTMLs from the 4 Universities Data Set at CMU (Web->KB) project⁴. The task is to extract course numbers from faculty and department web pages.
- **PHONENUMBER** recognition: we downloaded 3,149 documents, 2,000 of them from the 20 newsgroup dataset⁵ and the remaining 1,149 from the 4 universities data set. The task is to extract phone numbers from newsgroup messages and personal web pages.

We gathered various types of documents, ranging from HTML pages to OCR scanned documents. In Table 1, we list the basic statistics about documents in each of the 5 recognition tasks. The average length of a document varies a lot across the tasks, ranging from 1,284 in **EMAILADDRESS** to 137k in **DATETIME**.

4.2 Labeled Data for Evaluation

In order to allow comprehensive evaluation, we used student volunteers to manually label all documents in **EMAIL**, **COURSENUMBER** and **PHONENUMBER** corpuses. For documents in **BILLDATE** task, we knew the ground truth based on [3], so we did not use volunteers. For **DATETIME** task, we deemed too time consuming to label all the 6,000 documents; instead, we manually labeled one randomly selected substring from each document that contained 2017 in the center.

We list the number of entities in each corpus in Table 1. We also list the total time our volunteers took to annotate the corpus in each task and report the labeling rate as (milliseconds /character), which is calculated as the total time divided by the number of characters in the corpus. We assert that the labeling rate for **BILLDATE** task, is similar to that of **DATETIME** task. We observe that **COURSENUMBER**, **EMAILADDRESS**, **PHONENUMBER** are easier to label than **DATETIME**, and that the average entity length ranges from 6 to 20.

4.3 Accuracy Measures

We conduct all of our experiments using document-level 5-fold cross validation. We first divide documents into 5 subsets at random. We train the models on candidate substrings generated from documents in 4 out of the 5 subsets and test them on the candidate strings in the

remaining subset. The reported accuracies are averaged over the 5 repetitions within the cross-validation. We report position-level and entity-level accuracy.

Suppose there are N characters in the test set and denote the true labels for the i -th character as y_i and its prediction as \hat{y}_i . We define the positional precision (PosPrec), the positional recall (PosRecall) and the positional F1 (PosF1) as:

$$\begin{aligned} \text{PosPrec} &= \frac{\sum_{i=1}^N \mathbb{1}(y_i == 1 \wedge \hat{y}_i == 1)}{\sum_{i=1}^N \mathbb{1}(\hat{y}_i == 1)} \\ \text{PosRecall} &= \frac{\sum_{i=1}^N \mathbb{1}(y_i == 1 \wedge \hat{y}_i == 1)}{\sum_{i=1}^N \mathbb{1}(y_i == 1)} \quad (1) \\ \text{PosF1} &= 2 \cdot \text{PosPrec} \cdot \text{PosRecall} / (\text{PosPrec} + \text{PosRecall}) \end{aligned}$$

Entity level accuracies are calculated considering entities in the test strings. Suppose there are P ground truth entities in the test strings, denoted as E_{true} . After we get positional predictions for test strings, we extract the predicted entities as substrings of consecutively predicted positive characters. Assuming there are Q predicted entities in E_{pred} , we can calculate the size of their intersection $|E_{true} \cap E_{pred}|$. We define the entity precision (EntPrec), the entity recall (EntRecall) and the entity F1 (EntF1) as

$$\begin{aligned} \text{EntPrec} &= |E_{true} \cap E_{pred}| / |E_{pred}| \\ \text{EntRecall} &= |E_{true} \cap E_{pred}| / |E_{true}| \quad (2) \\ \text{EntF1} &= 2 \cdot \text{EntPrec} \cdot \text{EntRecall} / (\text{EntPrec} + \text{EntRecall}) \end{aligned}$$

5 FRAMEWORK CHARACTERIZATION

In this section, we study the proposed framework and its components, without focusing on user time.

5.1 RE_0 for Candidate Substring Extraction

The first step in the framework is creating the candidate regex RE_0 with high recall. We instructed one of the coauthors to come up with RE_0 for each of the 5 tasks in less than 5 minutes per task. Table 2 lists the resulting candidate regexes. For the **DATETIME** task, since the 6,000 documents are from August 24 - 30, 2017, the regex assumes that string 2017 occurs in all datetime entities listing year 2017.

Using the labels we collected for all the tasks (Table 1), we are able to evaluate the precision and recall of RE_0 defined as $\text{Prec} = \frac{\# \text{True entities hit by } RE_0}{\# \text{Total hit of } RE_0 \text{ in } \mathcal{D}}$, $\text{Recall} = \frac{\# \text{True entities hit by } RE_0}{\# \text{Total true entities in } \mathcal{D}}$, respectively.

²<http://machinelearning.inginf.units.it/data-and-tools>

³<https://www.cs.cmu.edu/~enron/>

⁴<http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/>

⁵www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html

Table 2: Summary of the RE_0 .

| Domain | RE_0 | Prec | Recall | $ S_0 $ | %Cov |
|---------------------|---------|--------|--------|---------|------|
| DATETIME | 2017 | 0.194* | 1.0* | 761,002 | 9.23 |
| BILLDATE | \d{2,4} | 0.105 | 0.980 | 72,258 | 43.8 |
| EMAILADDRESS | @ | 0.392 | 1.0 | 5,488 | 71.0 |
| COURSENUMBER | \d{2,4} | 0.112 | 0.969 | 43,623 | 79.2 |
| PHONENUMBER | \d{3,4} | 0.198 | 0.990 | 25,087 | 29.8 |

* Assumes all datetimes have "2017".

As can be seen in Table 2, the recall is very high on all 5 tasks, while the precision is quite low, as expected.

Each substring recognized by RE_0 becomes an anchor for a candidate substring. Each candidate substring is formed by taking $\frac{L}{2}$ characters preceding the start of the match and $\frac{L}{2}$ characters succeeding it. We set $L = 100$ in all experiments. As seen from Table 2, for **BILLDATE**, the total length of S_0 ($= 72,258 * 100$) is 43% of the original corpus ($= 6,000 * 27,518$) while it still contains 98% of all the true entity mentions. Thus, $|R_0|$ more than doubles efficiency of manual labeling.

5.2 RE_{wl} for Weak Labeling

For experiments in this section, we assume that RE_{wl} is already available and ask if pretraining an NN on weak labels obtained from RE_{wl} is beneficial. We collect regexes to instantiate RE_{wl} from the web and published papers. For **BILLDATE**, the regex is from [21]. We use the top five regexes from the Rege Library⁶ website for **EMAILADDRESS**. For **PHONENUMBER**, seven out of eight regexes are from [21], and the remaining one is from [17]. We use four regexes, one of which is borrowed from the results learned by ReLIE [17], and the remaining three are from [21] for **COURSENUMBER**. We use a disjunction of the collected regexes in each task as RE_{wl} . Those RE_{wl} are used on the candidate substrings to generate weak labels.

5.3 Hyperparameter Tuning and Settings

Deep learning is very sensitive to hyperparameters. A common approach to tune the hyperparameters is to explore several combinations of hyperparameters on validation data. However, since our framework relies on an iterative process that repeatedly fine-tunes an NN on an increasingly large set of labeled substrings, this standard approach is not feasible. Instead, we tune the hyperparameters on a subset of the weakly labeled data. We use the *random search* algorithm proposed in [5] that was shown to be more effective than the *grid search*. The best hyperparameters obtained in this way are used in all the experiments.

We set the activation function in the first fully connected layer to tanh and the batch to 512. We also add dropout layers after the embedding layer, the max pooling layer, and the first fully-connected layer to avoid overfitting, with the dropout rate set at 0.5. Our implementation is in PyTorch. With pre-defined RE_{wl} , we can tune the learning rate (lr), the dimension in the character embedding layer (emsize), the hidden unit size (nhidden) in BiLSTM layers and the number of BiLSTM layers (nlayers) by 5-fold cross validation using a random sample of weakly labeled data. We explore the

⁶<http://www.regexlib.com/>

following ranges for the hyperparameters: $\text{lr} = 2^{-k}$, $k \in [6, 7, \dots, 12]$, $\text{emsize} \in [20, 30, 40, 50, 60, 70, 80]$, $\text{nhidden} \in [50, 75, 100, 125, 150, 200]$ and $\text{nlayers} \in [1, 2, 5]$. We select a set of the best hyperparameters for each task.

We use 5 epochs to train an NN on weakly labeled data S_{wl} . For each round of fine-tuning, we use all previously collected manually labeled substrings S_h and train for 10 epochs over S_h . For selection of candidate substrings for labeling, we select them from a pool of 10,000 randomly selected candidate strings. We set the number of strings to be labeled in each iteration to $K = 20$. For **RME** selection algorithm, we set M to 500.

5.4 Impact of Weak Supervision

We evaluate 5 different approaches: (1) **Random**, which randomly predicts 0 or 1 for any character. (2) RE_{wl} , which uses regexes from Section 5.2 to recognize entity mentions. (3) $NN_{RE_{wl}}$, which is the NN pretrained on weak labels generated by RE_{wl} . (4) **RQ w/o (100)**, which is the NN trained directly with 100 randomly selected manually labeled candidate substrings. (5) **RQ w (100)**, which is an NN pretrained on weak labels and fine-tuned with 100 randomly selected labeled candidate substrings.

In the top half of Table 3, we see that weak supervision helps in two aspects. First, if we compare $NN_{RE_{wl}}$ and RE_{wl} , we notice that the weakly supervised model $NN_{RE_{wl}}$ preserves the precision and recall of the original regexes. Second, comparing **RQ w (100)** and **RQ w/o (100)**, we notice that pretraining on weak labels is superior. It is worth pointing out that, as expected, the positional accuracy is consistently higher than the entity-level accuracy.

5.5 Impact of Active Sampling

In this section, we evaluate the performance of 4 different sampling strategies described in Section 3.4. The first two are random sampling baselines, one being a cold start version without pretraining and another with pretraining on weak labels generated by regexes from Section 5.2. The last two are uncertainty-based, both using the pretraining. Table 3 reports EntF1 and PosF1 scores achieved by the 4 approaches after 1,000 labeled candidate substrings. First, we observe that pretraining an NN on weakly labeled data (**RQ w (1000)**) is superior to the cold start training (**RQ w/o (1000)**). Second, we observe that uncertainty-based sampling is superior to random sampling on all 5 tasks. **RME** is slightly better than **ME**.

In Figure 3, we illustrate how the accuracy changes with fine-tuning on the **DATETIME** task until one labels 1,000 candidate substrings. We zoom in the tails of the learning curves in the small embedded figures. The more time a user spends on annotation, the better the performance of the NNs. The two uncertainty-based approaches are superior to random sampling. The behavior is consistent on other 4 tasks (not shown due to space constraints).

From Table 3, we observe that we can achieve around 0.95 PosF1 in all 5 recognition tasks, however, the EntF1 is always worse than PosF1. It indicates that the NN entity recognizer predicts partially correct entities quite often. As an example, 75% of the partial matches in **EMAILADDRESS** task differ from the true entities by only 1-2 characters. This percentage is 35.3%, 30.9%, 36.1%, 30% on the other 4 tasks, respectively.

Table 3: Impact of weak supervision and active learning on the 5 tasks.

| Model Name | Human Annots | DATETIME | | BILLDATE | | EMAILADDRESS | | COURSENUMBER | | PHONENUMBER | |
|----------------------|--------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| | | EntF1 | PosF1 | EntF1 | PosF1 | EntF1 | PosF1 | EntF1 | PosF1 | EntF1 | PosF1 |
| Random | 0 | 0 | 0.1004 | 0 | 0.0331 | 0.0002 | 0.2543 | 0.0002 | 0.0446 | 0 | 0.1335 |
| RE_{wl} | 0 | 0.4340 | 0.7104 | 0.2827 | 0.3519 | 0.8811 | 0.9695 | 0.3926 | 0.4299 | 0.3182 | 0.5196 |
| $NN_{RE_{wl}}$ | 0 | 0.4411 | 0.7103 | 0.2825 | 0.3529 | 0.8819 | 0.9733 | 0.4078 | 0.4354 | 0.3138 | 0.5179 |
| RQ w/o (100) | 100 | 0.0449 | 0.8218 | 0.2853 | 0.6164 | 0.6914 | 0.9716 | 0.5311 | 0.6838 | 0.1418 | 0.5921 |
| RQ w (100) | 100 | 0.5061 | 0.8675 | 0.508 | 0.6196 | 0.962 | 0.9907 | 0.6876 | 0.7899 | 0.6007 | 0.7787 |
| RQ w/o (1000) | 1000 | 0.8376 | 0.9343 | 0.8226 | 0.9169 | 0.9903 | 0.9973 | 0.8408 | 0.9129 | 0.7972 | 0.8903 |
| RQ w (1000) | 1000 | 0.8644 | 0.9420 | 0.8452 | 0.9177 | 0.9898 | 0.9973 | 0.8451 | 0.9175 | 0.8253 | 0.8964 |
| ME (1000) | 1000 | 0.8696 | 0.9537 | 0.9534 | 0.9838 | 0.9940 | 0.9959 | 0.9234 | 0.9514 | 0.8943 | 0.9359 |
| RME (1000) | 1000 | 0.8879 | 0.9565 | 0.9537 | 0.9829 | 0.9951 | 0.9983 | 0.8765 | 0.9373 | 0.8964 | 0.9401 |

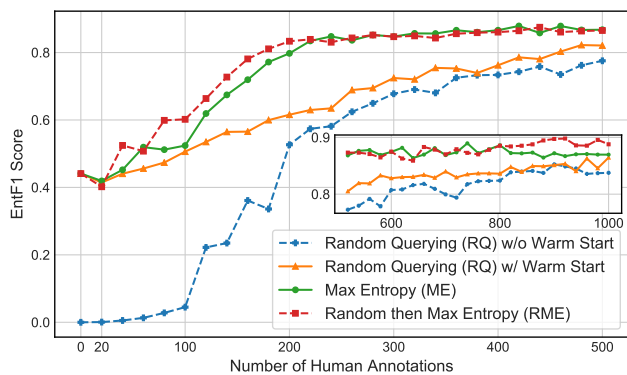


Figure 3: 4 query algorithms for DATETIME recognition.

From these results, we conclude that, given a regex, it is beneficial to pre-train an initial model with weak labels. We also conclude that uncertainty-based sampling of candidate substrings for labeling is superior to random sampling.

6 USER STUDIES

In our framework, a user can invest her effort in coming up with a regex and/or in manual labeling of the candidate substrings. In this section, we explore the tradeoffs between the two, assuming a user is given a limited time budget of T minutes. We ignore the time to come up with the candidate regex RE_0 , because it is much easier to come up with it than to come with a regex RE_{wl} that has both high recall and high precision.

6.1 Experimental Design

To experimentally explore the trade-off between creating a regex versus manual labeling of the candidate substrings, we collected data from 4 computer science students with different expertise in writing regexes. We use capital letters to represent the volunteers as M, C, J, and A. We asked the volunteers to create a regex for the **DATETIME** and **COURSENUMBER** tasks. For each task, we gave them 1,000 candidate substrings randomly selected from S_0 . We instructed the volunteers to use `{https://regex101.com}` environment to create and debug regexes. We gave the volunteers 40 minutes to create a regex for each task. We also asked them to submit their intermediate

regex after 5, 10, and 20 minutes of work. Eventually, we obtained regexes from volunteers C, J, A for both tasks and regexes from volunteer M only for the **DATETIME** task.

Unlike regex writing, labeling candidate substrings does not require much, if any expertise. We assume all of the volunteers are average people and can create string annotations at the same speed as listed in Table 1. Using those numbers, we are able to simulate a range of strategies a volunteer may take to help with the **DATETIME** and **COURSENUMBER** tasks within our framework. The first two strategies are two extremes, while the next 4 strategies are the trade-offs:

- **RegAll**. User spends all the time on constructing a regex. The final entity recognizer is the regex created after 40 minutes.
- **Label**. User immediately starts to annotate candidate strings selected from S_0 . NN is trained and fine-tuned using the labeled candidate substrings selected with Random Querying, as described in Section 3.4.
- **Reg5**. User spends the first 5 minutes on constructing RE_{wl} , which is used to pretrain an NN on weakly labeled data. Then, the user spends the remaining 55 minutes for annotating the candidate substrings selected using Random Querying.
- **Reg10**. The same as **Reg5**, but the user spends 10 minutes to construct RE_{wl} and 50 minutes for labeling.
- **Reg20**. The same as **Reg5**, but the user spends 20 minutes to construct RE_{wl} and 40 minutes for labeling.
- **Reg40**. The same as **Reg5**, but the user spends 40 minutes to construct RE_{wl} and 20 minutes for labeling.

Since there was no time for hyperparameter tuning in our real-time scenario, we had to select the hyperparameters in advance. Although it is not completely fair, in our experiments, we fixed all hyperparameters to a combination that appeared robust on all tasks in Section 5: $\text{lr} = 2^{-7}$, $\text{emsize} = 70$, $\text{nhidden} = 125$, $\text{nlayers} = 5$.

6.2 Experimental Results

Figure 4 shows the results for volunteer M, who had the most extensive expertise in writing regexes among our volunteers, on **DATETIME** recognition task. The figure shows EntF1 score as a function of time for the 6 different strategies. The figure allows us to compare different strategies for several different time budgets $T = [5, 10, 20, 40, 60]$.

The figure reveals several interesting observations. If the time budget is only 5 minutes, RE_{wl} generated by volunteer M is superior in accuracy to an NN trained on candidate substrings labeled

Table 4: The performance of the 6 strategies under different time budgets of volunteers C, J, and A for DATETIME and COUSENUMBER recognition. × means not applicable.

| Strategy | Volunteer C on DATETIME | | | | | Volunteer J on DATETIME | | | | | Volunteer A on DATETIME | | | | |
|---------------|-------------------------|--------------|--------------|--------------|--------------|-------------------------|--------------|--------------|--------------|--------------|-------------------------|--------------|-------------|--------------|--------------|
| | Time budget (min) | | | | | Time budget (min) | | | | | Time budget (min) | | | | |
| | 5 | 10 | 20 | 40 | 60 | 5 | 10 | 20 | 40 | 60 | 5 | 10 | 20 | 40 | 60 |
| RegAll | 0.002 | 0.412 | 0.456 | 0.593 | 0.593 | 0.0 | 0.0 | 0.001 | 0.061 | 0.061 | 0.002 | 0.003 | 0.005 | 0.011 | 0.011 |
| Label | 0.001 | 0.006 | 0.036 | 0.377 | 0.648 | 0.001 | 0.006 | 0.036 | 0.377 | 0.648 | 0.001 | 0.006 | 0.036 | 0.377 | 0.648 |
| Reg5 | × | 0.0 | 0.0 | 0.0 | 0.0 | × | 0.0 | 0.417 | 0.489 | 0.597 | × | 0.0 | 0.17 | 0.429 | 0.614 |
| Reg10 | × | × | 0.33 | 0.429 | 0.55 | × | × | 0.066 | 0.334 | 0.57 | × | × | 0.038 | 0.297 | 0.389 |
| Reg20 | × | × | × | 0.383 | 0.54 | × | × | × | 0.271 | 0.385 | × | × | × | 0.261 | 0.393 |
| Reg40 | × | × | × | × | 0.626 | × | × | × | × | 0.232 | × | × | × | × | 0.416 |

| Strategy | Volunteer C on COUSENUMBER | | | | | Volunteer J on COUSENUMBER | | | | | Volunteer A on COUSENUMBER | | | | |
|---------------|----------------------------|--------------|--------------|--------------|--------------|----------------------------|--------------|--------------|--------------|--------------|----------------------------|------------|--------------|--------------|--------------|
| | 5 | 10 | 20 | 40 | 60 | 5 | 10 | 20 | 40 | 60 | 5 | 10 | 20 | 40 | 60 |
| RegAll | 0.554 | 0.592 | 0.532 | 0.338 | 0.338 | 0.145 | 0.184 | 0.19 | 0.19 | 0.19 | 0.563 | 0.6 | 0.675 | 0.702 | 0.702 |
| Label | 0.046 | 0.361 | 0.652 | 0.761 | 0.815 | 0.046 | 0.361 | 0.652 | 0.761 | 0.815 | 0.046 | 0.361 | 0.652 | 0.761 | 0.815 |
| Reg5 | × | 0.591 | 0.682 | 0.75 | 0.796 | × | 0.433 | 0.654 | 0.751 | 0.795 | × | 0.571 | 0.66 | 0.72 | 0.777 |
| Reg10 | × | × | 0.679 | 0.734 | 0.771 | × | × | 0.611 | 0.732 | 0.798 | × | × | 0.651 | 0.698 | 0.769 |
| Reg20 | × | × | × | 0.733 | 0.796 | × | × | × | 0.684 | 0.797 | × | × | × | 0.681 | 0.727 |
| Reg40 | × | × | × | × | 0.707 | × | × | × | × | 0.681 | × | × | × | × | 0.699 |

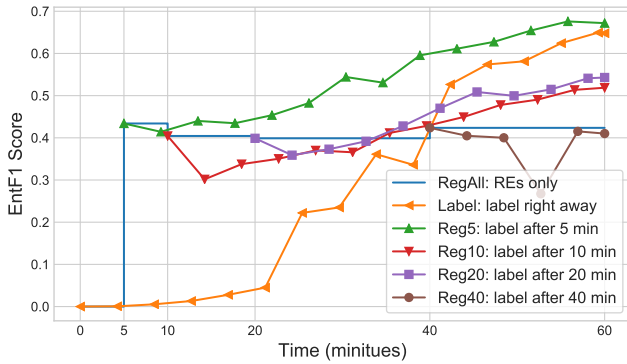


Figure 4: Time efficiency of volunteer M on DATETIME task.

within 5 minutes. After 10 minutes, NNs become superior to using RE_{wl} only. The best trade-off between regex writing and labeling is achieved by **Reg5**. It seems that placing an extensive effort in improving RE_{wl} does not pay off: the regex created after 5 minutes is comparable in accuracy to the one created after 40 minutes. **Label** is not competitive initially, but after 60 minutes it catches up with the overall best **Reg5**.

To examine the generalizability of the conclusions with volunteers and different recognition tasks, we repeated the analysis with volunteer C, J, A on tasks **DATETIME** and **COUSENUMBER** in Table 4.

From Table 4 we see that volunteer C is different from volunteer M. While C’s RE_{wl} produced after 5 minutes is not accurate, there is a steady increase in C’s EntF1 accuracy after 10, 20, and 40 minutes. The observed accuracy after 40 minutes is higher than that of volunteer M. **RegAll** is better than the rest until the 50 minute mark is reached. After 50 minutes, the **Label** becomes more accurate than any regex-based approach. **Reg5** is extremely inaccurate. This is a surprising finding because, unlike **Label**, we do not observe any accuracy improvement with the increase in number of labeled candidate substrings. It seems that the NN pretrained using RE_{wl}

created by volunteer C after 5 minutes has properties that prevent successful fine-tuning with labeled substrings.

Table 4 also shows results for volunteers J and A on task **DATETIME**. One can observe that neither volunteer manages to come up with a good regex in 40 minutes. Interestingly, the overall behavior of these 2 volunteers is more similar to volunteer M than to volunteer C. **Reg5** is the best overall in the first 40 minutes. **Label** becomes competitive with **Reg5** after around 40 minutes. **Reg10, 20, 40** show that it does not pay off to spend a large amount of time on writing and refining RE_{wl} .

Table 5 provides an insight into the differences between RE_{wl} produced after 5 minutes by volunteers C, J, and A. None of the volunteers is able to create an accurate RE_{wl} . This is expected knowing that they were exposed to 1,000 strings with 100-character lengths and asked to write a regex within only 5 minutes. Interestingly, volunteer C created a very specific regex with precision 1 and very low recall, while volunteers J and A created regexes with very low precision and recall. Volunteer C’s RE_{wl} had only 34 matches in S_0 , which meant that the resulting weakly labeled data set S_{wl} was extremely imbalanced with virtually all negative labels. We hypothesize that the high imbalance resulted in a very poorly pretrained NN, to the extent that it could not have been improved by fine-tuning. Unlike volunteer C, although volunteers J and A were not more successful with regexes, their RE_{wl} resulted in a more balanced weakly labeled data set, that allowed successful fine-tuning.

Table 4 shows that we obtain comparable results on the **COUSENUMBER** task with volunteers C, J, and A. An overall theme emerges from the user study and can be summarized as:

- If the time budget is less than 40 minutes, it is useful to spend a few minutes to construct RE_{wl} for weak labeling and the remaining allotted time for labeling.
- If the time budget is over 40 minutes, the weak labeling step could potentially be skipped and it might be sufficient to focus all effort on labeling of candidate substrings.

Table 5: Regexes created by C, J and A after the first 5 minutes for the DATETIME task.

| Source | Regex and their properties | | | |
|--------|---|---------|-----------|-------|
| C | <code>\d{4}-\d{1,2}-\d{1,2}\d{1,2}:\d{1,4}-\d{1,4}</code> | | | |
| J | <code>([0-9][0-9][0-9][0-9] 0-9 0-9)\n/([0-9][0-9]\n 0-9 0-9)</code> | | | |
| A | <code>(Monday Tuesday Wednesday Thursday Friday Saturday Sunday){0,1}\s*[0-9]{1,2}</code> | | | |
| | No. of matches in S_0 | EntPrec | EntRecall | EntF1 |
| C | 34 | 1.0 | 0.001 | 0.001 |
| J | 161,299 | 0.0 | 0.0 | 0.0 |
| A | 3,795,149 | 0.001 | 0.027 | 0.002 |

Limitation of our study. Before concluding the section, we point out that a limitation of our study is that we ignore the time needed to train and fine-tune an NN. Our assumption is that the training is instantaneous. We use a standard PC with a single GeForce GTX 1080 Graphics Card in our actual experiments. For all tasks, excluding **DATETIME**, pretraining an NN on S_{wl} took in the range of 20 minutes to an hour, and it took almost 2 hours for **DATETIME**. Each round of fine-tuning on all data sets ranged from 2 to 20 minutes. Thus, it appears that the user would waste time waiting for an NN to be pretrained and fine-tuned. However, this limitation is not necessarily a fatal flaw of our study due to several reasons: (1) a user could proceed with manual labeling and regex construction while waiting for NN training, (2) a user could switch to some other task while waiting, (3) the training time could be significantly improved if it were implemented on a more powerful computer system, (4) our study did not focus on training speedups, and it is possible that with some tuning the training time could be further reduced.

7 CONCLUSIONS

We investigated the problem of entity extraction, where entities follow or closely resemble patterns described using regular expressions. Industrial strength entity recognizers for this class of entities employ regex. Regex is either manually crafted or learned. The main drawbacks of regexes are that they tend to be complex to achieve high coverage, are difficult to maintain, and are not resilient to noise, such as typos. In the wake of data deluge, deep learning algorithms are an attractive alternative, but they require large amount of human annotated data. We propose a framework that combines the advantages of regexes and deep learning, coupled with weak supervision and active learning.

We conducted extensive experiments with data from 5 application domains: email, course number, phone number, datetime, and bill date. We also conducted a user study with 4 volunteers. The experiments showed that we can build ML models that are regex-oblivious, achieve high accuracy, and are resilient to small noise. The user study provided interesting insights about the trade-offs between constructing regexes and manually labeling the unlabeled text.

ACKNOWLEDGMENTS

This work was supported in part by the following grants: U.S. NSF BigData 1546480 and U.S. NIH R21CA202130.

REFERENCES

- [1] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Marco Mauri, Eric Medvet, and Enrico Sorio. 2012. Automatic generation of regular expressions from examples with genetic programming. In *GECCO*. 1477–1478.
- [2] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Eric Medvet, and Enrico Sorio. 2014. Automatic synthesis of regular expressions from examples. *Computer* 47, 12 (2014), 72–80.
- [3] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. 2016. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering* 28, 5 (2016), 1217–1230.
- [4] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. 2018. Active learning of regular expressions for entity extraction. *IEEE Transactions on cybernetics* 48, 3 (2018), 1067–1080.
- [5] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *JMLR* 13, Feb (2012), 281–305.
- [6] Falk Brauer, Robert Rieger, Adrian Mocan, and Wojciech M Barczynski. 2011. Enabling information extraction by inference of regular expressions from sample entities. In *CIKM*. ACM, 1285–1294.
- [7] Yukun Chen, Thomas A Lasko, Qiaozhu Mei, Joshua C Denny, and Hua Xu. 2015. A study of active learning methods for named entity recognition in clinical text. *Journal of biomedical informatics* 58 (2015), 11–18.
- [8] Jason PC Chiu and Eric Nichols. 2015. Named entity recognition with bidirectional LSTM-CNNs. *arXiv preprint arXiv:1511.08308* (2015).
- [9] Robert A Cochran, Lorin D’Antoni, Benjamin Livshits, David Molnar, and Margus Veenas. 2015. Program boosting: Program synthesis via crowd-sourcing. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 677–688.
- [10] François Denis. 2001. Learning regular languages from simple positive examples. *Machine Learning* 44, 1-2 (2001), 37–66.
- [11] Henning Fernau. 2009. Algorithms for learning regular expressions from positive data. *Information and Computation* 207, 4 (2009), 521–541.
- [12] Jason Fries, Sen Wu, Alex Ratner, and Christopher Ré. 2017. SwellShark: A Generative Model for Biomedical Named Entity Recognition without Labeled Data. *arXiv preprint arXiv:1704.06360* (2017).
- [13] Athanasios Giannakopoulos, Claudiu Musat, Andreea Hossmann, and Michael Baeriswyl. 2017. Unsupervised aspect term extraction with b-lstm & crf using automatically labelled datasets. *arXiv preprint arXiv:1709.05094* (2017).
- [14] IQAndreas. 2014. What is meant by “Now you have two problems”? <https://softwareengineering.stackexchange.com/questions/223634/what-is-meant-by-now-you-have-two-problems>
- [15] Anita Krishnakumar. 2007. *Active learning literature survey*. Technical Report. Technical reports, University of California, Santa Cruz. 42.
- [16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360* (2016).
- [17] Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and HV Jagadish. 2008. Regular expression learning for information extraction. In *EMNLP*. 21–30.
- [18] Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bidirectional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354* (2016).
- [19] Rachel Millner. 2008. Four Regular Expressions to Check Email Addresses. <https://tinyurl.com/y93bomtvt>
- [20] Maureen A. Murtaugh, Bryan Smith Gibson, Doug Redd, and Qing Zeng-Treitler. 2015. Regular expression-based learning to extract bodyweight values from clinical notes. *Journal of Biomedical Informatics* 54 (2015), 186 – 190.
- [21] Karin Murthy, P Deepak, and Prasad M Deshpande. 2012. Improving recall of regular expressions for information extraction. In *WISE*. Springer, 455–467.
- [22] Hieu T Nguyen and Arnold Smeulders. 2004. Active learning using pre-clustering. In *ICML*. ACM, 79.
- [23] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. In *Advances in neural information processing systems*. 3567–3575.
- [24] Jingbo Shang, Liyuan Liu, Xiang Ren, Xiaotao Gu, Teng Ren, and Jiawei Han. 2018. Learning Named Entity Tagger using Domain-Specific Dictionary. *arXiv preprint arXiv:1809.03599* (2018).
- [25] Yanyao Shen, Hyokun Yun, Zachary C Lipton, Yakov Kronrod, and Animashree Anandkumar. 2017. Deep Active Learning for Named Entity Recognition. *arXiv preprint arXiv:1707.05928* (2017).
- [26] Stanley Simoes, P Deepak, Munu Sairamesh, Deepak Khemani, and Sameep Mehta. 2018. Content and Context: Two-Pronged Bootstrapped Learning for Regex-Formatted Entity Extraction.. In *AAAI*.
- [27] Qing T. Zeng, Sergey Goryachev, Scott Weiss, Margarita Sordo, Shawn N. Murphy, and Ross Lazarus. 2006. Extracting principal diagnosis, co-morbidity and smoking status for asthma research: evaluation of a natural language processing system. *BMC Medical Informatics and Decision Making* 6, 1 (26 Jul 2006), 30.
- [28] Shanshan Zhang, Lihong He, Slobodan Vucetic, and Eduard Dragut. 2018. Regular Expression Guided Entity Mention Mining from Noisy Web Data. In *EMNLP*. 1991–2000.