Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022), pages 524–533
Marseille, 20-25 June 2022

VALET: Rule-Based Information Extraction for Rapid Deployment

Dayne Freitag, John Cadigan, Robert Sasseen, Paul Kalmar
SRI International
9988 Hibert Street, Suite 203
San Diego, California 92131, USA
firstname.lastname@sri.com

Abstract

We present VALET, a framework for rule-based information extraction written in Python. VALET departs from legacy approaches predicated on cascading finite-state transducers, instead offering direct support for mixing heterogeneous information—lexical, orthographic, syntactic, corpus-analytic—in a succinct syntax that supports context-free idioms. We show how a handful of rules suffices to implement sophisticated matching, and describe a user interface that facilitates exploration for development and maintenance of rule sets. Arguing that rule-based information extraction is an important methodology early in the development cycle, we describe an experiment in which a VALET model is used to annotate examples for a machine learning extraction model. While learning to emulate the extraction rules, the resulting model generalizes them, recognizing valid extraction targets the rules failed to detect.

Keywords: information extraction, rule-based methods

1. Introduction

Under the current state of the art, achieving peak accuracy in sentence-level information extraction (IE) involves training supervised models on substantial quantities of text annotated at the phrase level. Whether the objective is named entity or concept recognition, relation extraction, event extraction, or some slot-filling endeavor, the basic recipe is the same: extraction elements must be identified and labeled in context, their boundaries marked, and phrasal elements optionally drawn into typed relations with each other. Once enough examples have been annotated in this fashion, the literature describes a large number of machine learning models that can be trained to perform extraction at accuracies ranging from strong to adequate, e.g., (Ma and Hovy, 2016; Zhang et al., 2018; Wadden et al., 2019). To the extent that an extraction problem is nuanced, e.g. exhibits high linguistic variability or requires supra-lexical interpretation, data requirements are increased, necessitating the employment of a team of annotators, often governed by detailed guidelines to maximize annotation consistency. Thus, the IE development process can be quite expensive, often too expensive for many applications that might benefit from special-purpose extraction.

The models that result from this process are also chronically limited. Over the decades during which IE has been a focus of study, the field has emphasized problem formulations of broad applicability. For example, recognizing mentions of named entities and certain concepts, such as dates or monetary amounts, is a necessary component of many applications of text understanding. However, all such work must commit to specific genres (e.g., newswire) and levels of granularity, necessarily limiting the applicability of any resulting model. Generally, it is not difficult to find practical problems of text understanding for which use of such community models is problematic, whether because the formalization assumed by the model is an awkward fit to the problem at hand, or because the text of interest is distinctive enough to occasion significant degradation when a model is applied. This problem is most obvious for canonical formulations such as event extraction, where the types of extraction targets are limited in number and exhibit strong domain dependency. For example, the ACE formulation of this problem, which defined 33 event types (LDC, 2005), was strongly influenced by use cases relevant to national defense and intelligence analysis, and therefore provides limited value to many other use cases involving the same genres.

The upshot of these challenges is that IE is practically unavailable for many problems of text understanding. While a methodology exists to produce accurate extraction systems, its application is beyond the reach of many real-world budgets. We argue that, as a practical matter, *agility* in deployment is an important complement to and at least as important as the field's long-standing concern with *accuracy*. Various approaches to IE offer different trade-offs between these two concerns. In particular, rule-based approaches typically yield higher accuracies than those involving machine learning early in the development cycle, their relative inferiority only becoming evident after substantial volumes of annotated data have been amassed. Despite this, research into rule-based methods has stalled with few exceptions.

In the interest of reviving interest in these methods, we offer the following contributions:

- We introduce VALET (*Very Agile Language Extraction Toolkit*), an open-source framework implemented in Python, with a rule syntax that

flexibly incorporates various linguistic sources of evidence—orthographic, lexical, corpus-analytic, and syntactic—enabling the succinct definition of efficient extractors. [1]

- We illustrate the application of this framework to a problem of considerable practical interest and describe an interactive front end that greatly enhances the speed and convenience with which existing extraction capabilities can be maintained and new ones implemented.

- And we present the results of a study exploring the use of VALET as a means to perform annotation at scale in the training of robust machine learning extractors. We maintain that the ultimate promise of rule-based extraction is not as a replacement for machine learning extractors, but as a component in future hybrid frameworks that lower the cost of entry and increase the agility of IE deployment.

2. Related Work

Lowering the cost of deploying IE through machine learning has been and continues to be an area of active research. Contextual language models trained on large text collections lower the learning curve for many NLP tasks, a benefit that extends to IE (Shi and Lin, 2019). A range of approaches have been proposed to further reduce training overheads, including prototype learning (Han et al., 2018; Fritzler et al., 2019; Huang et al., 2020) and transfer learning (Wang et al., 2018; Huang et al., 2018; Yang and Katiyar, 2020). In some cases, these methods make it possible to achieve impressive competence in a new task with a few training examples. But note that such approaches, inasmuch as they often transfer extraction knowledge from known target types or from highly resourced domains to adjacent ones, do not eliminate the need for annotation. And it is often questionable whether the resulting models are sufficiently performant for downstream use without supplementation.

Much work on rule-based IE has been influenced by the FASTUS system (Hobbs et al., 1997) and the subsequent common pattern specification language (CPSL) for rule-based extraction (Appelt and Onyshkevych, 1998), which emphasized extraction through "cascaded" finite-state transducers, an implementation commitment motivated by a desire to avoid the expense of interpreting context-free rules. A potentially negative side-effect of this commitment is an emphasis on phased annotation; rules operate on the annotations deposited by lower-level rules. This rule tiering limits the flexibility of languages in this class, while increasing the level of required expertise and engineering overhead. Despite these disadvantages, a number of subsequent efforts have either implemented CPSL directly, such as in the GATE project's JAPE system (Thakker et al., 2009), or proposed alternative languages implementing cascaded transduction (Piskorski et al., 2004; Chang and Manning, 2014; Kluegl et al., 2016).

The VALET language, while offering access to the same kinds of processing, imposes no restrictions on how information derived from distinct forms of processing (e.g., tagging, named entity recognition, embeddings, etc.) should be used or combined. VALET shares this focus on uniform access to diverse sources of evidence with the Odin framework (Valenzuela-Escárcega et al., 2016), though VALET rules are more modular, relying heavily on rule composition through named reference. The typical result, in our opinion, is more succinct, more intelligible rule sets. VALET also supports context-free constructs, including recursive rules, relying on the rule author to control extraction expense through the use of selective rule components. The automated exploitation of such sources of efficiency has been the focus of several papers (Khaitan et al., 2008; Reiss et al., 2008), including the more recent Odinson framework (Valenzuela-Escárcega et al., 2020), which offers efficient search against extraction output over large corpora.

We believe that the value of rule-based IE will ultimately be maximized through its incorporation into hybrid systems that use rules for exploration and early development, and statistical methods or machine learning for robust deployment. A notable alternative to such an approach is the use of "labeling functions," as exemplified in the Snorkel framework (Ratner et al., 2017). Rule authoring frameworks like VALET are compatible with such work, offering an arguably more convenient means to implement labeling functions than through code, while applying to a larger class of extraction problems.

Type	Syntax	Description
import	`<-`	enables modularization
token class	`:`	defines token classes
phrase	`->`	matches subsequences
parse	`^`	matches parse segments
coordinator	`~`	manipulates matches
frame	`$`	composite extractions

Table 1: The types of statements understood by VALET.

3. VALET Language

A VALET rule set consists of a sequence of statements, each containing three elements: a *name*, a *statement separator*, and an *expression* defining the behavior to be associated with the name. The statement separator is a bit of syntax that controls how the expression should be evaluated.

As shown in Table 1, VALET recognizes six types of

[1] github.com/SRI-AIC/valet

Type	Example
regex	`/^[a-z]/i`
set	`{ a an the }i`
radius	`{ king queen }0.5`
layer	`pos[NN NNP]`
reference	`&myclass`

Table 2: The types of atomic token class expressions understood by VALET.

expression. For all types but *import*, the effect of interpreting a statement is to create an *extractor* and associate it with the indicated name. Once defined, an extractor may be applied to a *token sequence*, yielding a stream of zero or more *matches*, each covering some contiguous span in the input. In practice, a token sequence is often a sentence, but VALET is agnostic to the provenance and form of its input, and has been fruitfully applied to non-sentential sequences, such as those found in table cells. We now describe each of the types in Table 1 in turn.

3.1. Imports

Import statements make the extractors defined in an external file available in the current context. References to imported rules must be prefixed with the import name using dot notation. For example:

```
ner <- ner.vrules
person -> @ner.persons
```

This rule sequence has the effect of loading the extractors defined in the file named `ner.vrules` and associating them with the package name `ner`. The second statement defines a recognizer for person names that simply delegates to the extractor named `persons` in `new.vrules`. (See Section 3.3 for more information on this kind of inter-rule reference.)

3.2. Token Classes

The ability to succinctly define flexible token classes, drawing on orthography, lexical membership, embeddings, and syntax, affords VALET much of its power. A *token class* extractor yields matches covering individual tokens, usually words or units of punctuation, but also dependency labels in some contexts (see Section 3.4).

Token classes are represented by Boolean expressions (with operators `and`, `or`, and `not`, and supporting parenthesized subexpressions), the atomic elements of which are listed in Table 2. The *regex* and *set* expressions provide match by regular expression and explicit enumeration, respectively. The optional terminal 'i' qualifier in each case indicates case insensitivity. A *set* expression can optionally be prefixed with 'f' and specify a file name to load an external token lexicon.

A *radius* expression uses the syntax for *set* with a numeric suffix. Using an optionally loaded embedding, its effect is to match any input tokens within the indicated Euclidean radius of any of the words listed in the expression.

The *layer* expression is used to access token-level annotations provided by third-party analyzers (typically, but not necessarily, general-purpose NLP libraries). The example in the table matches any token tagged as a singular noun by inspecting labels collected in the `pos` layer. Similarly, the expression

```
pname : ner[S-PER]
```

matches any single-token person name identified by an underlying NER model. Note that here, as in other cases where VALET relies on external analysis, it accesses that analysis verbatim. Thus, rules predicated on one NLP engine may suffer degradation if another engine following a discrepant annotation scheme is substituted.

As shown in the table, a previously defined token class can be incorporated by reference, enabling the construction of legible expressions of arbitrary complexity. For example:

```
cap : /^[A-Z/
job : { writer scientist }0.8
title : &cap and &job
```

3.3. Phrase Expressions

Phrase expressions employ a regular expression syntax that is applied to token rather than character sequences. VALET recognizes the standard regular expression operators ('?' for optional components, '*' and '+' for Kleene closures, '|' for alternation) and understands parenthesized subexpressions. In other respects, elements of an expression are presumed to represent literal tokens, unless they are identifiers prefixed by '&' or '@', in which case they refer to separately defined token classes or arbitrary named extractors, respectively. This named co-reference between phrase expressions not only implements a context-free semantics, but is also VALET's mechanism for submatch capture (see Section 3.5).

An example will hopefully make this clear. Consider the rules:

```
pname : ner[B-PER I-PER]
honor : { Dr Mr Mrs Ms }
person -> &pname+
titled -> &honor .? @person
```

The `titled` extractor in this example recognizes person mentions prefixed by an honorific, incorporating a separate phrase extractor for names by reference. Note that unlike in standard character-level regular expressions, the optional '.' has no special significance and matches period characters in the input literally.

3.4. Parse Expressions

Parse expressions are identical in form and interpretation to phrase expressions with one crucial difference: they are applied to the graph returned by dependency parsing or a comparable process. A subsequence matches a particular parse expression if its start and end tokens are joined by a path whose arc labels match the expression. By default, this matching is insensitive to dependency direction; the element `dobj` yields matches from verbs to their direct objects and from objects to their governing verbs. This approach is suitable for almost all practical uses, but expression elements may be prefixed by '\' and '/' to restrict matching to parent-child and child-parent dependencies, respectively.

A simple example should illustrate how this capability works in practice. The following expression extracts all extents covering a verb and any of its direct objects, assuming the annotation engine uses Universal Dependencies version 2 [2]:

```
direct_objects ^ obj conj*
```

Note that because there is no directional restriction on `obj`, this expression also generates a match from the first direct object to its governing verb. Given this behavior, and more generally the misalignment between matching over a non-linear structure and linear subsequence extraction, parse expressions mainly become useful when mixed with other language elements, as will be made clear in the next section.

3.5. Coordinators

Coordinators greatly expand the expressiveness and utility of the Valet language, providing various transformations of match streams that are difficult or impossible to achieve with the constructs described so far. A coordinator accepts one or more match streams and transforms them into an output stream through a number of available operations, such as comparison, submatch selection, unification, etc.

Coordinators are defined using the following syntax:

```
name ~ expression
```

Table 3 lists the types of expressions that may occur in this context. Where *name* occurs in the table, the name of an extractor must be specified. Where *strm* (for "match stream") is indicated, an expression may contain one of three things: '_', a special token representing a match spaning the entire input sequence (e.g., a whole sentence); the name of an extractor, which is then applied to the input sequence to generate matches; or a nested coordinator expression. The *win* parameter calls for a non-negative integer representing a token window. Finally, `invert` is an optional literal argument that has the effect of inverting the interpretation of a coordinator in which it appears. We now briefly describe the interpretation of these operators.

[2] `https://universaldependencies.org/`

`match(`*name*`, `*strm*`)`
`filter(`*name*`, `*strm*`, invert?)`
`prefix(`*name*`, `*strm*`, invert?)`
`suffix(`*name*`, `*strm*`, invert?)`
`near(`*name*`, `*win*`, `*strm*`, invert?)`
`precedes(`*name*`, `*win*`, `*strm*`, invert?)`
`follows(`*name*`, `*win*`, `*strm*`, invert?)`
`contains(`*strm*`, `*strm*`)`
`overlaps(`*strm*`, `*strm*`)`
`union(`*strm*`, ...)`
`inter(`*strm*`, ...)`
`diff(`*strm*`, ...)`
`connects(`*name*`, `*strm*`, `*strm*`)`
`select(`*name*`, `*strm*`)`

Table 3: Coordinator expressions available in VALET.

The `match` and `filter` have similar behavior, each applying a named extractor to a stream of matches indicated by the second argument. While `match` returns any matches of the named extractor within the bounds of some input match, `filter` passes through any input matches that the indicated named extractor matches (or doesn't match if `invert` is specified). For example, if `pname` matches fully qualified person names and `honor` matches honorifics, then the following rule matches only person names that include honorifics:

```
hname ~ filter(honor, pname)
```

Inclusion of the optional `invert` argument would yield names *lacking* honorifics. Both `prefix` and `suffix` have the same signature as `filter`, but require that the named extractor generate a match that immediately precedes or follows (rather than contains) an input match, respectively.

The operators `near`, `precedes`, and `follows` are proximity tests, passing through any matches on the input stream occurring in the proximity of the named extractor, the degree of proximity controlled by *win*. When `invert` is indicated, non-proximal matches are instead passed through. The `contains` and `overlaps` operators yield any matches in their first argument that enclose or overlap matches in their second argument.

The operators `union`, `inter`, and `diff` perform the union, intersection, and set difference, respectively, on one or more input streams. For these operators, element equality corresponds to match extent: two matches are equal if they have the same input sequence and the same start and end tokens. Thus, `inter` and `diff` essentially pass through any matches in the first stream that have the same extent as matches found in all of (none of, respectively) the subsequent streams. When `union` and `inter` encounter distinct matches of equal extent, only the first of the two compared matches is passed through, but the resulting record

maintains a pointer to the other match, so that it can be subsequently selected (see below).

The `connects` operator yields any matches of the extractor named in its first argument that connects two matches from each of its input streams. A connection is observed if the extractor matches a sequence starting within the extent of a match in the first stream and ending within the extent of a match in the second stream. Although this operator is agnostic to the type of the named extractor, it is most useful when applying parse extractors, especially in combination with the `select` operator, described next. Please refer to Section 5 for a typical combination of `connects` and `select`.

As matches are generated by potentially deep cascades of named extrators, VALET tracks the contributions that lead to the generation of a given match. Applied to the stream of matches in its second argument, the `select` operator yields a stream of contributing submatches made by the named extractor. An example should make clear the value of this capability. Suppose we have defined a token class called `hire` (e.g., using a radius expression), a parse expression called `dobj` that connects transitive verbs to their objects, and a phrase extractor for person names called `name` built on top of the `ner` layer. Consider the expression:

```
hire_whom ~ select(name,
  connects(dobj, hire, name))
```

This expression generates a stream of names in an object relation to a hiring verb.

3.6. Frames

Finally, the `frame` operator generates composite objects suitable for downstream applications. Its use is perhaps best illustrated by extending the previous example. Suppose `nsubj` is a parse extractor matching subject-verb dependencies. A typical frame usage is the following:

```
hsubj ~ select(hire,
  connects(nsubj, name, hire))
hobj ~ select(hire,
  connects(dobj, hire, name))
hiring ~ union(hsubj, hobj)
hframe $ frame(hiring,
  employer=hsubj name,
  employee=hobj name)
```

As this example suggests, the `frame` operator uses a previously named extractor as an anchor, then applies submatch selection to associate contributing submatches with arbitrarily named fields. Here, the `union` operator is used to unify instances of hiring that include both a hirer and a hiree. The `frame` then selects the respective submatches, treating each of the two slots as optional. Note that this submatch selection takes the form of a space-separated list of named extractors, representing a kind of submatch selection path. In this example, to populate the `employer` field, we select first any contributing `hsubj` phrase (which covers any name in a subject relation to a verb of hiring), then select the implicated `name` match.

4. User Interface and Instrumentation

A critical piece of the VALET framework is its user interface (UI), depitcted in Figure 1, which supports exploration and interactive rule authoring. The interface is divided into three panes arranged vertically—from top to bottom the *rules*, *text*, and *control* panes. At invocation time, the user posits a rules file and corpus, individual documents from which are displayed in the text pane. When the name of an extractor in the rules pane is clicked on, the system highlights the corresponding matches in the text pane, to develop rule sets incrementally. The buttons on the lower left enable the user to move through the corpus, either sequentially or by scanning forward to the next match of a selected rule.

The VALET UI provides additional features that make access to some of the relatively opaque functions more convenient. When the user clicks on a word in the text pane, the UI displays information about that word from the underlying NLP stack[3], such as the part of speech, lemma, and NER tag. Similarly, when the user drags over two words in a sentence, the system to reports the labels on the dependency path between the two words. Through a separate process that precedes UI invocation, the user can generate lexical embeddings and compute a distance matrix between words, persisted in a format that VALET understands. If such information is present, the user can summon a list of similar words by shift-clicking on a word in the text pane. To create a custom token class from this list, the user simply selects those entries in this list that are suitable and clicks a button to paste the corresponding token class definition into the rules pane.

VALET understands a number of common corpus persistence formats, such as directory of plain text files, single file with one document per line, CSV, CoNLL, etc., all represented as subclasses of a class that provides corpus iteration, document segmentation, and tokenization. The size of this API is quite small, making it straightforward to assimilate new corpus formats.

5. A Sample Model

To illustrate how the features described in Section 3 combine to support the implementation of sophisticated extraction capabilities, we present a walkthrough of a sample rule set. Table 4 shows a short set of rules from a project on procedural language. In many domains (e.g., food recipes), texts that contain instructions are dominated by imperative constructions

[3] VALET is integrated with two third-party stacks that can be used interchangeably: Spacy (`https://spacy.io`) and Stanza (`https://stanfordnlp.github.io/stanza/`). We are currently integrating the SRL model from AllenNLP (`https://allennlp.org`).

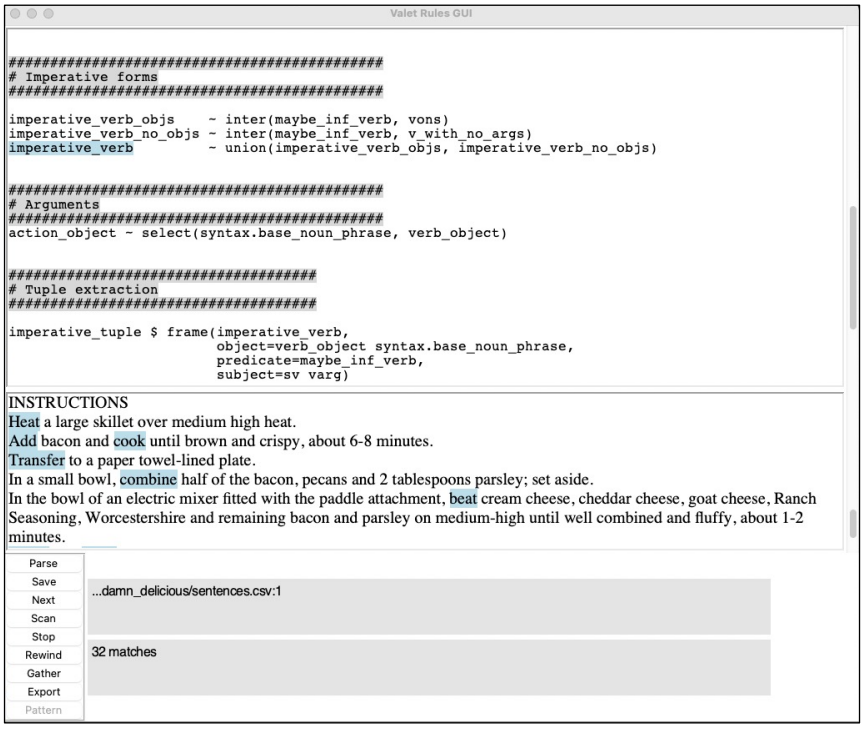

Figure 1: The VALET user interface.

(e.g., "Chop the onions"). The rules in the table aim to identify these constructions and extract the verbs and their corresponding objects, as a step toward extracting the entire procedure.

The statement in Line 1 imports a built-in rule set intended to provide convenient access to various syntactic features. Invocations of rules defined in this module can be easily identified in the rest of the listing by looking for identifies containing the `syntax` prefix. An example reference can be seen in Line 5, which has the effect of renaming a token class defined in `syntax`. The preceding statement (Line 4) is more selective, matching verbs in an infinitive inflection.

The two coordinator expressions and Lines 8 and 11 each match verbs based on their syntactic neighborhood as defined by the dependency structure in which they participate. The first expression relies on the built-in `ROOT` extractor, which matches the head word of the dependency parse, selecting those head words that are uninflected verbs. The second identifies verbs with direct objects by selecting the verb component (using the `select` coordinator) of verb-object relations (identified using the `connects` coordinator). Note that the corresponding objects remain accessible to rules built on this expression (e.g., in the expression at Line 19).

The statement on Line 16 takes the intersection of the streams the previous two statements generate, yielding only uninflected clausal head verbs *with* objects. Finally, in Line 19, the verbs and their objects are picked out from this stream and given names for the convenience of downstream code. Note that if a verb has multiple objects, the rule `syntax.objects`, and correspondingly `verb_object`, generates multiple matches for a single verb. The frame operator collapses such matches into a single frame with a list-valued `object` slot.

6. Valet for Accelerated Annotation

For simple extraction problems, problems of limited scope exhibiting low linguistic variability, the framework we have described makes it possible to deploy competetent information extraction sufficient for many downstream uses. Perhaps more important, initial development of such capabilities almost always involves lower overhead than under a conventional regime of systematic annotation and training of machine learning models. Using the features that VALET provides, a trained practitioner often can achieve F1 scores in hours that would take days or weeks under conventional model development. Of course, for many non-trivial problems, rule authoring eventually hits a point of diminishing returns, particularly when "edge cases" start to dominate the set of legitimate targets that the rules do not yet sanction. A supervised model trained on sufficient data then often yields more robust detection.

This complementarity in strengths—the speed of rule deployment and the robustness of supervised models—suggests a hybrid approach to extractor development. Specifically, is it possible to avoid some of the overhead of model development by using extraction rules for annotation and training of extractors? Here, we pro-

```
syntax <- syntax.vrules

# Some useful classes
inf_verb : pos[VB]
any_verb : &syntax.verb

# Uninflected main verbs
main_inf_verb ~ inter(ROOT, inf_verb)

# A verb-object relation
verb_object   ~ select(any_verb,
                       connects(syntax.objects, any_verb,
                                syntax.base_noun_phrase))

# Imperatives with objects
imp_verb_obj  ~ inter(main_inf_verb, verb_object)

# Final frame
imperative_tuple $ frame(imp_verb_obj,
                         object=verb_object syntax.base_noun_phrase,
                         predicate=imp_verb_obj))
```

Table 4: A sample rule set for extracting imperative verbs and their objects.

vide anecdotal evidence that this is a viable approach, particularly that models trained in this fashion learn to extract accurately *and* generalize effectively, detecting valid extraction targets that the "seed" rules don't cover.

Prior work provides evidence that this approach is promising, though the specific conditions under which it adds value remain to be investigated. The state of the art in open information extraction (Open IE) involves a supervised model (Cui et al., 2018) trained on sentences annotated by earlier pattern-based approaches to the problem (Mausam, 2016), yielding more robust performance. Of course, because the scope of Open IE is extremely broad, it remains to be demonstrated that narrower extraction problems can be addressed in this fashion.

6.1. Extracting Scientific Claims

As part of a project seeking to document scientific progress in research on solar materials, we implemented a set of VALET rules to extract "claims" from a large collection of abstracts from the Web of Science[4], a corpus consisting of about 160K abstracts on solar energy research from 1968 to 2014. In our framing, a claim is a sentence-level relation between a quantitative *measurement* (consisting of a numeric *value* and a *unit* of measurement) and a *metric*, the quantity being measured. In the snippet, "*The overall light-to-electric energy conversion yield is 7.1-7.9 percent in simulated solar light*," the value, unit, and metric are "7.1-7.9", "percent", and "light-to-electric energy conversion yield", respectively. Note that this definition does not take into account the epistemic status of the statement, which would be required to distinguish true claims from, say, known facts.

In the context of that effort, we implemented and refined a set of rules for this problem (some 19 phrasal extraction rules and a few large synonym sets for relation elements such as units of measure). This extraction model was then used to harvest claims at scale and to plot progress on key metrics over the years covered by the corpus.

6.2. Experiments

In a fashion typical of many practical applications of information extraction, this effort proceeded without a formal definition of the extraction target, and the resulting model was evaluated by inspection. As it approached its final form, it was applied to increasingly large subsets of the corpus and its extractions reviewed for accuracy, with some ad hoc spot checking to increase confidence that no important claims were missed. The usual result of such a process is a precise extractor that is limited in recall to an uncertain degree. Of interest, therefore, are approaches that can build on this model, yielding improved recall without degradation in precision.

[4] https://clarivate.com/webofsciencegroup/solutions/web-of-science/

	P	R	F1
Metric	0.940	0.970	0.955
Measurement	0.932	0.963	0.947

Table 5: Precision, recall, and F1 on the test set

Here, we report on experiments conducted well after the project that motivated this model (the *reference model*) had ended. We seek to answer two questions: Taking the reference model as *authoritative*[5], can we use it to train a model that exhibits comparable precision? And does this trained model make useful errors of commission, finding legitimate claim expressions not sanctioned by the reference model?

To answer these questions, we applied VALET to the entire set of abstracts that were the focus of earlier rule development (some 160K abstracts—more than a million sentences—divided into 60% train, 20% validation, and 20% test) using its predictions to train a sequence labeler for metrics and measurements.

Our model was developed with the FLAIR toolkit, which enables experimentation with various embedding types and sequence tagging architectures (Akbik et al., 2018). We used the GloVE embeddings supported by the framework, which have 100 dimensions and a vocabulary of 400K. Our architecture followed default settings; we used bidirectional LSTMs with word-wise dropout rate of 0.05, a locked dropout rate of 0.5 and a hidden size of 256. The output of the LSTM is fed into a CRF layer which predicted BIO tags for the claim metrics and values (Lafferty et al., 2001). Training included learning rate scheduling targeting loss on the development set with a patience factor of 3 epochs and an annealing factor of 0.5. We used a batch size of 512 and stopped after 77 epochs. The model which performed the best on the development set was selected.

The results are displayed in Table 5. We can see that the model achieves strong performance on this task, though recall exceeds prediction. As noted before, this effect is potentially useful, inasmuch as rule-based methods are typically recall limited. In this case, precision errors are extractions that the rules do not sanction, but they may in fact correspond to legitimate "claims" that the rules miss.

To arrive at some qualitative insights into this matter, we performed a small post-hoc analysis. Selecting 30 false positives at random, we tabulated the number that fit our definition of claim, counting 17 in this category. Admittedly, this kind of analysis is vulnerable to investigator bias, but the result does reinforce the impression that useful generalization can be derived through this approach. Errors in this data are often due to overgeneralization of the metric entity to include other qualifying expressions attached to the measurement (e.g., "frequency range of *0.58-2.52 THz* at *room temperature*," measurement and metric displayed in *green* and *blue*, respectively).

On the other hand, useful generalization takes some interesting forms. We find that our corpus is "contaminated" with articles from biology and clinical chemistry. A non-trivial fraction of the false positives come from these domains (e.g., "an *average SAR* of *0.26 W/kg*"). The model also admits metric expressions containing editorial qualifiers (e.g., "a *promising power conversion efficiency* of *2.4%*). Finally, it exhibits greater inclusiveness in the processing of the text found between the two claim elements and greater flexibility with respect to order, as in "the *epitaxial layer thickness* becomes *less than 200 nm*" and "*2.5%* (*electrical power/incident light power*)".

7. Conclusion

Machine learning has largely eclipsed rule-based methods in academic work on information extraction. While this trend is easy to explain—machine learning models are typically more robust in the limit and offer certain methodological advantages—we argue that this development has had negative consequences, such as a focus on "popular" problems and a paucity of options for practitioners on a fixed budget. In compensation, we offer VALET, a framework that makes it possible to stand up competent sentence-level extraction for a wide range of problems at minimal expense.

We have argued that the early-development superiority of rule-based methods points in the direction of hybrid methodologies, in which the time to deployment of accurate machine learning extractors is greatly reduced through rule-driven annotation. We have presented evidence that such an approach is viable. Much work remains in this area, including careful quantification of accuracy as a function of effort. More interestingly, we believe that the authoring of rules can and should result from mixed initiative, that machine learning can be used to facilitate the many small actions and decisions a rule author takes in producing an effective rule set. In its current form, effective use of VALET or any comparable rule interpretation framework requires not only a working knowledge of rule language constructs, but also a certain level of NLP sophistication. We believe that the right incorporation of machine learning as a facilitator, rather than as the sole predictor, will ultimately make it possible to deploy non-trivial extraction for new problems in the course of an afternoon.

8. Acknowledgements

This material is based on work supported by the Office of Naval Research under Contract No. N000014-20-C-2037. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Office of Naval Research.

[5]In other words, the only data "annotation" is provided by applying the reference model, and any divergence of the learner from the reference model is measured as error.

9. Bibliographical References

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *Proceedings of the 27th international conference on computational linguistics*, pages 1638–1649.

Appelt, D. E. and Onyshkevych, B. (1998). The common pattern specification language. Technical report, SRI International.

Chang, A. X. and Manning, C. D. (2014). TokensRegex: Defining cascaded regular expressions over tokens. *Stanford University Computer Science Technical Reports. CSTR*, 2:2014.

Cui, L., Wei, F., and Zhou, M. (2018). Neural open information extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 407–413.

Fritzler, A., Logacheva, V., and Kretov, M. (2019). Few-shot classification in named entity recognition task. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 993–1000.

Han, X., Zhu, H., Yu, P., Wang, Z., Yao, Y., Liu, Z., and Sun, M. (2018). Fewrel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4803–4809.

Hobbs, J. R., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M., and Tyson, M. (1997). FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. In *Finite-State Language Processing*, pages 383–406. MIT Press.

Huang, L., Ji, H., Cho, K., Dagan, I., Riedel, S., and Voss, C. (2018). Zero-shot transfer learning for event extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2160–2170.

Huang, J., Li, C., Subudhi, K., Jose, D., Balakrishnan, S., Chen, W., Peng, B., Gao, J., and Han, J. (2020). Few-shot named entity recognition: A comprehensive study. *arXiv preprint arXiv:2012.14978*.

Khaitan, S., Ramakrishnan, G., Joshi, S., and Chalamalla, A. (2008). Rad: A scalable framework for annotator development. In *2008 IEEE 24th International Conference on Data Engineering*, pages 1624–1627. IEEE.

Kluegl, P., Toepfer, M., Beck, P.-D., Fette, G., and Puppe, F. (2016). UIMA Ruta: Rapid development of rule-based information extraction applications. *Natural Language Engineering*, 22(1):1–40. Publisher: Cambridge University Press.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of Eighteenth International Conference on Machine Learning*.

LDC, (2005). *ACE (Automatic Content Extraction) English Annotation Guidelines for Events*. Linguistic Data Consortium.

Ma, X. and Hovy, E. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074.

Mausam, M. (2016). Open information extraction systems and downstream applications. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence*, pages 4074–4077.

Piskorski, J., Schäfer, U., and Xu, F. (2004). Shallow processing with unification and typed feature structures–foundations and applications. *Künstliche Intelligenz*, 1(1):17–23.

Ratner, A. J., Bach, S. H., Ehrenberg, H. R., and Ré, C. (2017). Snorkel: Fast training set generation for information extraction. In *Proceedings of the 2017 ACM international conference on management of data*, pages 1683–1686.

Reiss, F., Raghavan, S., Zhu, H., Reiss, F., Raghavan, S., Krishnamurthy, R., Zhu, H., and Vaithyanathan, S. (2008). An algebraic approach to rule-based information extraction. In *In ICDE*, pages 933–942. IEEE.

Shi, P. and Lin, J. (2019). Simple bert models for relation extraction and semantic role labeling. *arXiv preprint arXiv:1904.05255*.

Thakker, D., Osman, T., and Lakin, P. (2009). GATE JAPE grammar tutorial.

Valenzuela-Escárcega, M. A., Hahn-Powell, G., and Surdeanu, M. (2016). Odin's runes: A rule language for information extraction. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 322–329.

Valenzuela-Escárcega, M. A., Hahn-Powell, G., and Bell, D. (2020). Odinson: A Fast Rule-based Information Extraction Framework. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2183–2191, Marseille, France, May. European Language Resources Association.

Wadden, D., Wennberg, U., Luan, Y., and Hajishirzi, H. (2019). Entity, relation, and event extraction with contextualized span representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5784–5789.

Wang, Z., Qu, Y., Chen, L., Shen, J., Zhang, W., Zhang, S., Gao, Y., Gu, G., Chen, K., and Yu, Y. (2018). Label-aware double transfer learning for cross-specialty medical named entity recognition. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1–15.

Yang, Y. and Katiyar, A. (2020). Frustratingly simple few-shot named entity recognition with struc-

tured nearest neighbor learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6365–6375.

Zhang, Y., Qi, P., and Manning, C. D. (2018). Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2205–2215.