ICE: Rapid Information Extraction Customization for NLP Novices

Yifan He and Ralph Grishman
Computer Science Department
New York University
New York, NY 10003, USA
{yhe,grishman}@cs.nyu.edu

Abstract

We showcase ICE, an Integrated Customization Environment for Information Extraction. ICE is an easy tool for non-NLP experts to rapidly build customized IE systems for a new domain.

1 Introduction

Creating an information extraction (IE) system for a new domain, with new vocabulary and new classes of entities and relations, remains a task requiring substantial time, training, and expense. This has been an obstacle to the wider use of IE technology. The tools which have been developed for this task typically do not take full advantage of linguistic analysis and available learning methods to provide guidance to the user in building the IE system. They also generally require some understanding of system internals and data representations. We have created ICE [the Integrated Customization Environment], which lowers the barriers to IE system development by providing guidance while letting the user retain control, and by allowing the user to interact in terms of the words and phrases of the domain, with a minimum of formal notation.

In this paper, we review related systems and explain the technologies behind ICE. The code, documentation, and a demo video of ICE can be found at http://nlp.cs.nyu.edu/ice/

2 Related Work

Several groups have developed integrated systems for IE development:

The extreme extraction system from BBN (Freedman et al., 2011) is similar in several regards: it is based on an extraction system initially developed for ACE[1], allows for the customization of entities and relations, and uses bootstrapping and active learning. However, in contrast to our system, it is aimed at skilled computational linguists.

The Language Computer Corporation has described several tools developed to rapidly extend an IE system to a new task (Lehmann et al., 2010; Surdeanu and Harabagiu, 2002). Here too the emphasis is on tools for use by experienced IE system developers. Events and relations are recognized using finite-state rules, with meta-rules to efficiently capture syntactic variants and a provision for supervised learning of rules from annotated corpora.

A few groups have focused on use by NLP novices:

The WIZIE system from IBM Research (Li et al., 2012) is based on a finite-state rule language. Users prepare some sample annotated texts and are then guided in preparing an extraction plan (sequences of rule applications) and in writing the individual rules. IE development is seen as a rule programming task. This offers less in the way of linguistic support (corpus analysis, syntactic analysis) but can provide greater flexibility for extraction tasks where linguistic models are a poor fit.

The SPIED system (Gupta and Manning, 2014) focuses on extracting lexical patterns for entity recognition in an interactive fashion. Our system, on

[1] https://www.ldc.upenn.edu/collaborations/past-projects/ace

Proceedings of NAACL-HLT 2015, pages 31–35,
Denver, Colorado, May 31 – June 5, 2015.

the other hand, aims at extracting both entities and relations. Furthermore, SPIED produces token sequence rules, while our system helps the user to construct lexico-syntactic extraction rules that are based on dependency paths.

The PROPMINER system from T. U. Berlin (Akbik et al., 2013) takes an approach more similar to our own. In particular, it is based on a dependency analysis of the text corpus and emphasizes exploratory development of the IE system, supported by search operations over the dependency structures. However, the responsibility for generalizing initial patterns lies primarily with the user, whereas we support the generalization process through distributional analysis.

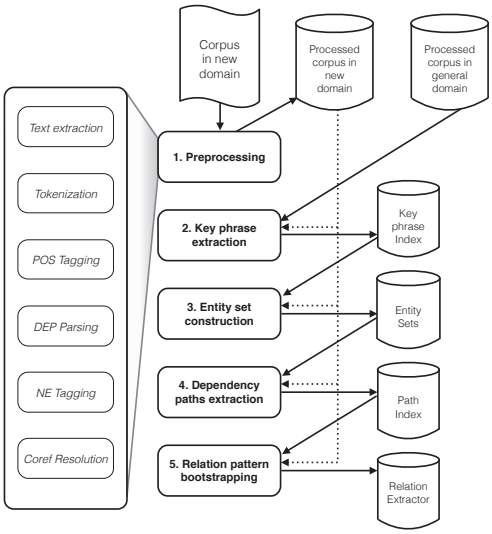

Figure 1: System architecture of ICE

3 System Description

3.1 Overall architecture

The information to be extracted by the IE system consists of user-specified types of entities and user-specified types of relations connecting these entities. Standard types of entities (people, organizations, locations, etc.) are built in; new entity types are defined extensionally as lists of terms. Relation types are captured in the form of sets of lexicalized dependency paths, discussed in more detail below.

For NLP novices, it is much easier to provide examples for what they want and make binary choices, than to come up with linguistic rules or comprehensive lists. ICE therefore guides users through a series of linguistic processing steps, presents them with entities and dependency relations that are potential seeds, and helps them to expand the seeds by answering yes/no questions.

Figure 1 illustrates the five steps of ICE processing: given a new corpus, preprocessing builds a cache of analyzed documents to speed up further processing; key phrase extraction and entity set construction build new entity types; dependency path extraction and relation pattern bootstrapping build new semantic relations.

3.2 Preprocessing

We rely on distributional analysis to collect entity sets and relation patterns on a new domain. Effective distributional analysis requires features from deep linguistic analysis that are too time-consuming to perform more than once. ICE therefore always preprocesses a new corpus with the Jet NLP pipeline[2] when it is first added, and saves POS tags, noun chunks, dependency relations between tokens, types and extents of named-entities, and coreference chains to a cache. After preprocessing, each of the following steps can be completed within minutes on a corpus of thousands of documents, saving the time of the domain expert user.

3.3 Key phrase extraction

In ICE, key phrases of a corpus are either nouns or multi-word terms. We extract multi-word terms from noun chunks: if a noun chunk has N adjectives and nouns preceding the head noun, we obtain $N + 1$ multi-word term candidates consisting of the head noun and its preceding i ($0 \leq i \leq N$) nouns and adjectives.

We count the absolute frequency of the nouns and multi-word terms and rank them with a ratio score, which is the relative frequency compared to a general corpus. We use the ratio score S_t to measure the representativeness of term t with regard to the given domain, as defined in Eq (1).

$$S_t = \frac{\#pos(t) \cdot \log^{\alpha}(\#pos(t))}{\#neg(t)} \quad (1)$$

where $\#pos(t)$ is the number of occurrences of term t in the in-domain corpus, $\#neg(t)$ is the number of

[2] http://cs.nyu.edu/grishman/jet/jet.html

occurrences of term t in the general corpus, and α is a user-defined parameter to favor either common or rare words, default to 0.

We present the user with a ranked list, where words or multi-word terms that appear more often in the in-domain corpus than in general language will rank higher.

3.4 Entity set construction

ICE constructs entity sets from seeds. Seeds are entities that are representative of a type: if we want to construct a DRUGS type, "methamphetamine" and "oxycodone" can be possible seeds. Seeds are provided by the user (normally from the top scoring terms), but if the user is uncertain, ICE can recommend seeds automatically, using a clustering-based heuristic.

3.4.1 Entity set expansion

Given a seed set, we compute the distributional similarity of all terms in the corpus with the centroid of the seeds, using the dependency analysis as the basis for computing term contexts. We represent each term with a vector that encodes its syntactic context, which is the label of the dependency relation attached to the term in conjunction with the term's governor or dependent in that relation.

Consider the entity set of DRUGS. Drugs often appear in the dependency relations *dobj(sell, drug)* and *dobj(transport, drug)* (where *dobj* is the direct object relation), thus members in the DRUGS set will share the features *dobj_sell* and *dobj_transport*. We use pointwise mutual information (PMI) to weight the feature vectors and use a cosine metric to measure the similarity between two term vectors.

The terms are displayed as a ranked list, and the user can accept or reject individual members of the entity set. At any point the user can recompute the similarities and rerank the list (where the ranking is based the centroids of the accepted and rejected terms, following (Min and Grishman, 2011)). When the user is satisfied, the set of accepted terms will become a new semantic type for tagging further text.

3.5 Dependency path extraction and linearization

ICE captures the semantic relation (if any) between two entity mentions by the lexicalized dependency path (LDP) and the semantic types of the two entities. LDP includes both the labels of the dependency arcs and the lemmatized form of the lexical items along the path. For example, for the sentence "[Parker] oversaw a sophisticated [crack cocaine] distribution business.", consider the parse tree in Figure 2. The path from "Parker" to "crack cocaine" would be *nsubj*$^{-1}$*:oversee:dobj:business:nn:distribution:nn*, where the $^{-1}$ indicates that the *nsubj* arc is being traversed from dependent to governor. The determiner "a" and the adjective modifier "sophisticated" are dropped in the process, making the LDP more generalized than token sequence patterns.

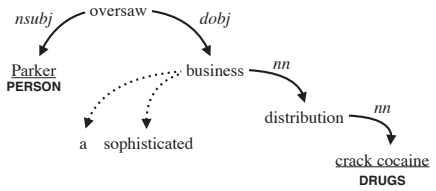

Figure 2: A parse tree; dotted relations ignored by LDP

We linearize LDPs before presenting them to the user to keep the learning curve gentle for NLP novices: given an LDP and the sentence from which it is extracted, we only keep the word in the sentence if it is the head word of the entity or it is on the LDP. The linearized LDP for the path in Figure 2 , "PERSON oversee DRUGS distribution business", is more readable than the LDP itself.

3.6 Bootstrapping relation extractors

3.6.1 Relation extractor

ICE builds two types of dependency-path based relation extractors. Given two entities and an LDP between them, the *exact* extractor extracts a relation between two entities if the types of the two entities match the types required by the relation, and the words on the candidate LDP match the words on an extraction rule. When the two nodes are linked by an arc in the dependency graph (i.e. no word but a type label on the LDP), we require the dependency label to match.

ICE also builds a *fuzzy* extractor that calculates edit distance (normalized by the length of the rule) between the candidate LDP and the rules in the rule set. It extracts a relation if the minimum edit dis-

tance between the candidate LDP and the rule set falls below a certain threshold (0.5 in ICE). We tune the edit costs on a development set, and use insertion cost 0.3, deletion cost 1.2, and substitution cost 0.8.

Fuzzy extractors with large rule sets tend to produce false positive relations. ICE therefore bootstraps both positive and negative rules, and requires that the candidate LDP should be closer to (the closest element in) the positive rule set than to the negative rule set, in order to be extracted by the fuzzy LDP matcher.

3.6.2 Bootstrapper

The learner follows the style of Snowball (Agichtein and Gravano, 2000), with two key differences: it bootstraps both positive and negative rules, and performs additional filtering of the top k ($k = 20$ in ICE) candidates to ensure diversity.

Starting with a seed LDP, the learner gathers all the pairs of arguments (endpoints) which appear with this LDP in the corpus. It then collects all other LDPs which connect any of these pairs in the corpus, and presents these LDPs to the user for assessment. If the set of argument pairs connected by any of the seeds is $\mathcal{S}$ and the set of argument pairs of a candidate LDP x is $\mathcal{X}$, the candidate LDPs are ranked by $\mid \mathcal{S} \cap \mathcal{X} \mid / \mid \mathcal{X} \mid$, so that LDPs most distributionally similar to the seed set are ranked highest. The linearized LDPs which are accepted by the user as alternative expressions of the semantic relation are added to the seed set. At any point the user can terminate the bootstrapping and accept the set of LDPs as a model of the relation.

Bidirectional bootstrapping. If the user explicitly rejects a path, but it is similar to a path in the seed set, we still bootstrap from the arg pairs of this path. We save all the paths rejected by the user as the negative rule set.

Diversity-based filtering. When presenting the bootstrapped LDPs, we require paths presented in the first ICE screen (top 20 candidates) to be distant enough from each other.

4 Experiments

We perform end-to-end relation extraction experiments to evaluate the utility of ICE: we start from plain text, extract named entities, and finally extract drug names and relations with models built by ICE. We collect approximately 5,000 web news posts from the U.S. Drug Enforcement Administration [3] (DEA) for our experiments.

	SELL			RESIDENT-OF		
	P	R	F	P	R	F
Fuzzy	0.60	0.22	0.32	0.68	0.51	0.58
-neg	0.59	0.22	0.32	0.55	0.51	0.53
Exact	0.92	0.10	0.18	0.72	0.47	0.57

Table 1: End-to-end relation extraction using small rule sets. *Fuzzy*: fuzzy match relation extractor with negative rule set; -neg: fuzzy match extractor without negative rule set; *Exact*: exact match extractor; P / R / F: Precision / Recall / F-score

	SELL			RESIDENT-OF		
	P	R	F	P	R	F
Fuzzy	0.46	0.36	0.40	0.56	0.53	0.55
-neg	0.31	0.38	0.34	0.30	0.56	0.39
Exact	0.76	0.20	0.32	0.75	0.53	0.62

Table 2: End-to-end relation extraction using large rule sets. Same configurations as Table 1

Entity Set Construction. In our first experiment, we extracted 3,703 terms from this corpus and manually identified 119 DRUGS names and 97 *law enforcement agent* (AGENTS) mentions, which we use as the "gold standard" sets. We then ran our customizer in the following manner: 1) we provided the entity set expansion program with two seeds ("methamphetamine" and "oxycodone" for DRUGS; "special agents" and "law enforcement officers" for AGENTS); 2) the program produced a ranked list of terms; 3) in each iteration, we examined the top 20 terms that had not been examined in previous iterations; 4) if a term is in the gold standard set, we added it to the expander as a positive seed, otherwise, we added it as a negative seed; 5) we continued the expansion with the updated seed set, repeating the process for 10 iterations. This process produced high-recall dictionary-based entity taggers (74% for

[3] http://www.justice.gov/dea/index.shtml

drugs, 82% for agents) in just a few minutes.

Relation Extraction. With the ICE-built DRUGS dictionary, we performed end-to-end extraction of two relations: SELL, in which a PERSON sells DRUGS, using "PERSON sell DRUGS" as seed, and RESIDENT-OF, which indicates that a PERSON resides in a GPE[4], using "PERSON of GPE" as seed. We manually annotated 51 documents from the DEA collection. There are 110 SELL relations and 45 RESIDENT-OF relations in the annotated corpus.

We first extracted small rule sets. For both relations, we asked a user to review the presented LDPs on the first screen (20 LDPs in total) and then ran bootstrapping using the expanded seeds. We did this for 3 iterations, so the user evaluated 60 LDPs, which took less than half an hour. We report the results in Table 1. Note that these are end-to-end scores, reflecting in part errors of entity extraction. After entity tagging and coreference resolution, the recall of entity mentions is 0.76 in our experiments.

We observe that fuzzy LDP match with negative rule sets obtains best results for both relations. If we remove the negative rule set, the precision of RESIDENT-OF is hurt more severely than the SELL relations. On the other hand, if we require exact match, the recall of SELL will decrease very significantly. This discrepancy in performance is due to the nature of the two relations. RESIDENT-OF is a relatively closed binary relation, with fewer lexical variations: the small RESIDENT-OF model covers around 50% of the relation mentions with 7 positive LDPs, so it is easier to rule out false positives than to further boost recall. SELL, in contrast, can be expressed in many different ways, and fuzzy LDP match is essential for reasonable recall.

We report experimental results on larger rule sets in Table 2. The large rule sets were bootstrapped in 3 iterations as well, but the user reviewed 250 LDPs in each iteration. The best score in this setting improves to 0.4 F-score for SELL and 0.62 F-score for RESIDENT-OF, as we have more LDP rules. The exact match extractor performs better than the fuzzy match extractor for RESIDENT-OF, as the latter is hurt by false positives.

[4]Geo-political entity, or GPE, is an entity type defined in Ace, meaning location with a government

5 Conclusion and Future Work

We described ICE, an integrated customization environment for information extraction customization and evaluated its end-to-end performance. We plan to explore more expressive models than LDP that can handle arbitrary number of arguments, which will enable ICE to build event extractors.

Acknowledgements

We thank Lisheng Fu, Angus Grieve-Smith, and Thien Huu Nguyen for discussions.

References

Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 85–94.

Alan Akbik, Oresti Konomi, Michail Melnikov, et al. 2013. Propminer: A workflow for interactive information extraction and exploration using dependency trees. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: Systems Demonstrations*, pages 157–162.

Marjorie Freedman, Lance Ramshaw, Elizabeth Boschee, Ryan Gabbard, Gary Kratkiewicz, Nicolas Ward, and Ralph Weischedel. 2011. Extreme extraction: machine reading in a week. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1437–1446.

Sonal Gupta and Christopher Manning. 2014. SPIED: Stanford pattern based information extraction and diagnostics. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 38–44.

John Lehmann, Sean Monahan, Luke Nezda, Arnold Jung, and Ying Shi. 2010. LCC approaches to knowledge base population at tac 2010. In *Proc. TAC 2010 Workshop*.

Yunyao Li, Laura Chiticariu, Huahai Yang, Frederick R Reiss, and Arnaldo Carreno-Fuentes. 2012. WizIE: a best practices guided development environment for information extraction. In *Proceedings of the ACL 2012 System Demonstrations*, pages 109–114.

Bonan Min and Ralph Grishman. 2011. Fine-grained entity refinement with user feedback. In *Proceedings of RANLP 2011 Workshop on Information Extraction and Knowledge Acquisition*.

Mihai Surdeanu and Sanda M. Harabagiu. 2002. Infrastructure for open-domain information extraction. In *Proceedings of the second international conference on Human Language Technology Research*, pages 325–330.