Bootstrapped Self Training for Knowledge Base Population

Gabor Angeli, Victor Zhong, Danqi Chen, Arun Chaganty, Jason Bolton,
Melvin Johnson Premkumar, Panupong Pasupat, Sonal Gupta, Christopher D. Manning
Stanford University; Stanford, CA 94305
{angeli, vzhong, danqi, chaganty, jebolton}@stanford.edu
{melvinj, ppasupat, sonal, manning}@stanford.edu

Abstract

A central challenge in relation extraction is the lack of supervised training data. Pattern-based relation extractors suffer from low recall, whereas distant supervision yields noisy data which hurts precision. We propose bootstrapped self-training to capture the benefits of both systems: the precision of patterns and the generalizability of trained models. We show that training on the output of patterns drastically improves performance over the patterns. We propose self-training for further improvement: recall can be improved by incorporating the predictions from previous iterations; precision by filtering the assumed negatives based previous predictions. We show that even our pattern-based model achieves good performance on the task, and the self-trained models rank among the top systems.

1 Introduction

The landscape of relation extractors can be clustered into three general categories: (1) pattern-based methods, which suffer from difficulties with recall; (2) supervised methods, which suffer from a lack of training data; and (3) distantly supervised methods, which suffer from excessive noise in their training data. We present an approach – bootstrapped self-training – to relation extraction which mitigates the weaknesses of each of these approaches while preserving much of their benefits.

In this setup, we begin with a manually defined rule-based relation extractor, built using less than a person-week of total development time. This extractor is inherently high precision but low recall. We then use the predictions of this extractor over a large corpus to train a first iteration of statistical models. The predictions of these models can then be fed into this same self-training loop to train subsequent iterations of progressively higher-precision models. This can be thought of as a hard-EM-like procedure: from a smart initialization, we infer labels on our large unlabeled corpus. Then, from the statistics gained from this corpus we re-train our parameters to get a better sense of their true distribution.

The obvious danger of such an approach is the possibility of *semantic drift*, where the self-trained models drift progressively away from their intended label semantics. We mitigate this in two ways: first, the patterns are always included in the true labels. Assuming the initial patterns account for a non-negligible fraction of the labels, this at least partially anchors the distribution of extractions. Second, and perhaps more importantly, we train our extractors with a realistic proportion of positives to negatives – that is, we train with many more *no relation* examples than examples that express a relation. This, of course, requires our negative examples to be very clean. We get these negatives by removing from the candidate negatives (i.e., all type checked relation mentions) any example which is extracted by a *high recall* extractor. In our case, we used the union of all of our self-trained extractors as this high recall extractor.

For our submissions, we ran a conservative shallow version of our self-training procedure. From our pattern-based extractor's output, we trained a number of relation extractors. We then removed from our negative set all "negatives" which were predicted by the union of these extractors. We then re-trained our extractors off of the patterns output and our new negatives. Initial experiments showed modest improvements from running self-training for more iterations.

We show that our methods achieve good performance on both slotfilling and cold-start knowledge base population, presented in detail in Section 5. A union of our extractors ranks second and third in KBP 2015 for the slotfilling and KB track respectively. We further show that the pattern-based relation extractor alone achieves the highest precision of any system that achieves non-negligible recall, and ranks above median on the task (5^{th} excluding Stanford's other systems). Lastly, we show that model combination, in the flavor of Rajani et al. (2015), can improve precision of hop0 queries by 18 F_1 with only a 5 F_1 drop in recall.

2 System Design and Architecture

Traditionally, slot filling systems have been structured as a pipeline beginning with an information retrieval system, and then passing through an entity linker, a relation extractor, and finally a consistency component. This setup has the advantage of being resource-light; however, we have observed a number of practically important disadvantages:

1. Analytics on the corpus and the model – even simple analytics such as histograms of predicted relations, or the number of occurrences of an entity in the corpus – become slow and difficult to code.

2. An amount of recall is immediately lost at the initial information retrieval step.

3. The approach is abusive to the IR system: if a document contains multiple candidate relation mentions, the system must re-retrieve the document for each relation mention.

4. IR is a tool for querying *textual mentions*, whereas we are nearly always querying for *entities*. Even trivial morphological variations of names can confuse the IR system (e.g., *Beyonce* instead of *Beyonc'e*).

For these reasons, we built our KBP system around a relational database, similar in spirit to DeepDive (Niu et al., 2012). At the center of this framework are two tables, described below, both populated by running Stanford CoreNLP (Manning et al., 2014) and the Illinois Wikifier (Ratinov et al., 2011) on the 2010 and 2013 source corpora. In addition to the named entity tags from CoreNLP, we extract tags for additional relevant named entities (e.g., job titles, nationalities) via a set of gazetteers. Lastly, we run a retrained Stanford NER model for detecting job titles, trained off of the output of our gazetteers.

sentences A table of sentences in the KBP source corpus. This includes part-of-speech tags, lemmas, named entity tags, and dependency parses.

mentions A table of entity mentions, along with their named entity tag and canonical entity link. The canonical entity link of coreferent mentions in a document are constrained to be the same, but there is not other explicit notion of coreference. A canonical entity link is represented as the Wikipedia title of the entity, if one exists. If not, it is the surface form of the canonical coreference mention is used. In the case of dates and numbers, the Timex value and canonical numeric form are used respectively.

From these two core tables we can derive most of the "data munging" operations in knowledge base population directly in SQL. For instance, finding candidate relation mentions is a simple join on the mention table with itself, subject to constraints on occurring in the same sentence, and the named entity tags type-checking. Distant supervision is as simple as joining this relation mention table with a known knowledge base.[1]

Relation extraction then becomes a simple matter of taking as input such a relation mention table, joined with its provenance sentence, and outputting the relations expressed by each row of the table. In practice, all of our extractors are implemented as command-line programs taking as input a TSV on standard input (one example per row), and outputting a stream of relations for those rows.

Lastly, a coherent knowledge base is constructed from these relation mention predictions directly in the database through an SQL script. For the KB track, this knowledge base is submitted directly for evaluation. For the slotfilling track, the database is used as the input to a consistency and inference component – as in Angeli et al. (2014a) – and the output of that component is submitted for evaluation.

The advantages of this setup are many fold:

1. SQL is a powerful language for probing the corpus. Furthermore, since the output of our extractors are also in the database, we can easily view and debug against not only specific classification decisions, but also the aggregate statistics of the classifier.

[1] Although none of our extractors are distantly supervised, we do load a range of knowledge bases (Freebase, KBP, Google) into the database for filtering negatives.

2. Every candidate relation mention in the corpus is considered.

3. The query planner is able to most efficiently retrieve the data passed into the extractors.

4. Querying on *entities* in addition to textual mentions is trivial; furthermore, aggregate statistics on entity linking are practically invaluable during debugging.

5. Evaluation no longer performs relation extraction – evaluating on new query entities becomes a set of database queries.

2.1 Supporting Infrastructure

The work flow described in this section requires an unconventionally large software and hardware infrastructure. The database queries can be nearly linearly sped up through the use of distributed databases, such as Greenplum. This allows us to run on a 20 core machine with a huge speed improvement. Furthermore, the expensive joins described in the previous section are orders of magnitude more efficient when executed in memory. In practice, the submissions in this paper were run on a machine with 790GB of memory backed by a 1.2TB PCI-E solid state drive. Although this appears extravagant at first glance, we found the savings in runtime and debug time from this setup to justify the investment.

2.2 Development Data

Development of all of the models was evaluated against a number of development sets. First, the annotations from the 2013 KBP evaluation were propagated back into the database, linking to the mentions table. This was then used to create a supervised relation-mention level development set in the same format as the training and test data for the extractors.

This enabled two important types of debugging. Trivially, it provides a development set for assessing the quality of the extractors on human annotated data. However, it also for the first time enables approximately assessing *recall* errors in the system – these are examples which could not be linked to any known mention. For instance, even with approximate span matching only 71% of correct judgments could be mapped back to any mention pair at the time of submission.

This development data does, however, have the problem of being biased towards positive examples, and therefore less sensitive to precision errors than it should be. Furthermore, it does not distinguish between entity linking errors and relation extraction errors. Therefore, we also make use of the 2010 – 2013 data for a slower end-to-end evaluation of the system. The 2014 cold start data was used for model combination. The 2014 slotfilling data was unused.

2.3 Class Skew

An important emergent property of the system is the increased importance of precision over recall. Whereas IR serves as a sort of regularizer, ensuring that only potentially-sane sentences are proposed to the classifier, in our case every relation mention is a candidate for classification. This results in an increased sensitivity to misclassifications of sentences as exhibiting a known relation – particularly for common entities.

In some ways, this is a concern more broadly for NLP systems. In real world applications, classifiers are often applied to problems with a single dominant class; however, these systems are nearly always trained on a balanced or near balanced data set. In this paper, we approach this problem using our bootstrapped self training approach (see Section 4) – a simple and empirically effective approach.

3 Relation Extractors

At the core of our system is a set of relation extraction models. In total, Stanford trained the following 6 models, described in more detail in this section:

- **patterns**: The hand-coded *Semgrex* and *Tokensregex* patterns.
- **openie**: The Stanford OpenIE system (Angeli et al., 2015).
- **website**: A website extractor, based on approximate website UR match.
- **altnames**: An alternate names extractor, based entirely on entity linking output.
- **supervised**: A learned supervised logistic regression classifier.
- **neural**: A learned supervised bidirectional LSTM classifier.

3.1 Patterns

The base system for our bootstrapping process is a pattern-based relation extractor. The motivation behind collecting these patterns is to ensure that we capture the simple cases for relation extraction. As the amount of data in our corpus grows, it becomes more and more likely that the relations we are interested expressed straightforwardly somewhere in the corpus in a way that patterns can capture.

The patterns were constructed by hand from four sources: (1) the patterns used in Stanford's previous KBP submissions (Angeli et al., 2014a); (2) the patterns used in Roth et al. (2013); (3) the patterns from the DeepDive system described in Angeli et al. (2014a); and (4) a set of new patterns developed for this year. This yielded a total of 4528 Tokensregex patterns (Chang and Manning, 2014) and 169 Semgrex patterns.

2417 of the Tokensregex patterns were collected semi-automatically by taking common short n-grams between people and organizations, aiming to capture a long-tail of job titles that were missed in the named entity annotations by the job title gazetteer. For example, x, *incoming president of the* y. These yielded a small but noticeable improvement of around 1 F_1 on our dev set.

The remaining patters developed for this year were constructed by hand by hill-climbing on the errors found in the relation mention oriented development data described in Section 2.2, and evaluating on the 2010 – 2013 end-to-end evaluations. Especially for this extractor, the ability to rapidly evaluate over the entire corpus was invaluable to the development pipeline. In all, approximately 40 man-hours of work was expended writing patterns over the course of 2 months.

3.2 Open IE

The Stanford Open IE system was run over the corpus, and the resulting extractions mapped back into the KBP relation schema. This mirrors the evaluation in Angeli et al. (2015). This extractor also serves as a base case for self training.

3.3 Alternate Names

Alternate names are unique in that they are not usually explicit in the text, but rather are inferred via coreference chains. We therefore create a special-case alternate names extractor to handle these cases. In particular, for every canonical entity in our database, we collect all of the surface forms of that entity in the mentions table. In cases where there are more than 50 unique surface forms, we ignore the entity – it is likely a sink for entity linking mistakes. Otherwise, we consider any surface form which links at least twice to the canonical entity to be a name for the entity. Any pairs of surface forms in the same document which are considered names for the entity by this criteria are considered alternate names.

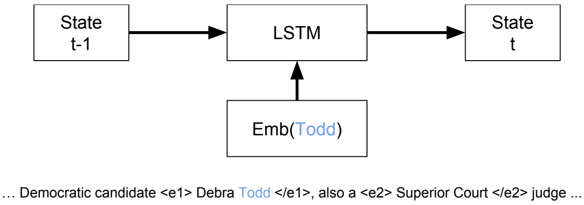
State
t-1
LSTM
State
t
Emb(Todd)
... Democratic candidate <e1> Debra Todd </e1>, also a <e2> Superior Court </e2> judge ...

Figure 1: At some time step t, the LSTM has read up to the word *Todd.* The sentence fragment read so far is represented by current state, denoted by "State t."

Importantly, note that this process is expressed declaratively in under 100 lines of SQL, and executes in a matter of minutes over the full 150 million sentence source corpus. On the 2010 dev evaluation, this yields 33% recall at 33% precision for organization alternate names, and 24% recall at 47% precision for person alternate names.

3.4 Websites

Although it's possible to extract a company's website using a conventional relation extractor, in practice it is more reliable to do approximate name matching on the website domain and organizations in the document. We match a domain with an organization if either the edit distance between the organization and the domain is sufficiently small, or the domain is a near acronym of the organization's name. On the 2010 dev evaluation, this yields 38% recall at 67% precision.

3.5 Traditional Extractor

The first of two trained models is a traditional relation extractor, implemented using l_1 regularized logistic regression. Since we do not have access to constituency trees, we cannot run the featurizer of Surdeanu et al. (2012); we therefore define our own, taking many of the same features.

3.6 LSTM Extractor

In addition to traditional relation classification models in which we manually select features such as dependency patterns, we propose a recurrent neural networks based model to automatically learn feature representations useful for the task. Recurrent networks have recently proven to

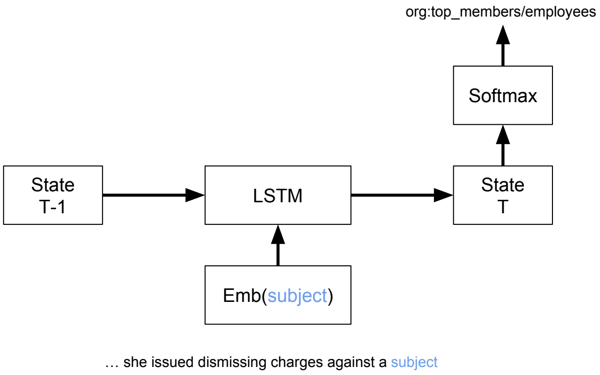
org:top_members/employees
Softmax
State
T-1
LSTM
State
T
Emb(subject)
... she issued dismissing charges against a subject

Figure 2: At time step T, the LSTM has read the entire sentence. The final hidden state, denoted "state T," encodes the entire sentence, and is multiplied with another weight matrix (not shown) to produce a distribution over the valid states.

be very successful for the relation classification, achieve near state of the art results on the SemEval 2010 relation classification task.

For our model, we employ a long short term memory (LSTM) based sequence classification model as shown in figures Figure 1 and Figure 2. The model takes as input the sentence and predicts as output the relation. At each time step, the model takes in the current word in the input sentence and looks up an associated embedding for the word. This embedding, along with the previous hidden state, are then used to compute the current hidden state. Finally, the hidden state at the last time step (eg. after the model has read the entire sentence) is used to compute a distribution over the set of possible relations. We train the network using stochastic gradient descent. In addition, we use Adagrad to adapt the learning rate per parameter and Dropout to regularize the network.

We make two task-specific modifications to simplify the training procedure:

- The input sentence is modified such that each of the entity is surrounded by position marker such that the model is aware of their respective locations
- The set of possible relations is reduced to the small subset of all relations that is possible given the NER tag of the subject entity and the NER tag of the object entity

4 Training

Ignoring alternate names and websites, the system has two extractors which do not need to be trained: patterns and Open IE. We run the union of these two high-precision systems on the entire source corpus, and collect the output predictions as a training dataset. In this way, we are creating a *bootstrapped* system.

We then sample 1000 examples from this dataset for each of the 41 relations, yielding 41,000 examples total. This is added to the supervised data released in Angeli et al. (2014b), 5000 *assumed negatives* from each of the 41 relations (205,000 total negative examples), and a small corpus of additional hand-annotated relations (7,858). Assumed negatives for a relation are relation mentions which: (1) type check with the given relation type, and (2) occur in neither the last iteration's predicted output, or any knowledge base.

We then train the traditional and LSTM extractors, and treat this output as the new training dataset. Predictably, more iterations of this training improves recall and degrades precision. For our submissions, based on dev results, we purely optimize precision: after a first iteration of training the learned systems, we re-generate the assumed negatives (recall that these depend on the output of the previous iteration), but continue to train only on the patterns as positive examples. A clear avenue of future exploration is assessing the tradeoff between precision and recall from both increasing the iterations of self-training, and changing the ratio of positive and negative training examples.

4.1 Model Combination

Taking inspiration from the model combination results in Rajani et al. (2015), we train our own model combination over the 6 extractors described in Section 3. We train a classifier taking as input a candidate relation produced by at least one of the extractors, and as output a judgment for whether to keep that prediction or not. We train the model combination system using four features:

1. An indicator for the number of extractors which proposed the relation.
2. An indicator for each extractor which proposed the relation.
3. An indicator for the relation being proposed.
4. An indicator for the combination of the relation being proposed and the system proposing it.

For training data, we then hand-annotated the predictions of our system on the 2014 hop0 and hop1 evaluations. The hop1 evaluations were annotated assuming perfect hop0 output. The 2014 hop0 data was used to train our hop0 model combination, and the hop1 data (assuming perfect hop0 output) was used to train the hop1 classifier.

5 Results

Stanford submitted five runs to the slotfilling track, and two runs two runs to the KB track. These runs were meant to capture different points in the precision / recall tradeoff by including different combinations of systems. In addition, for the slotfilling track, some of the systems used the model combination component.

A summary of our slotfilling submissions is given below.

Stanford1 A trained model combination of all systems for both hop0 and hop1.

Stanford2 A high precision system (patterns, openie, website, altnames) for hop0, and model combination for hop1.

Stanford3 A high recall system: the union of all models for both hop0 and hop1.

Stanford4 The hand-coded patterns alone, for both hop0 and hop1.

Stanford5 Same as Stanford1, but disallowing guessing of relations we have evidence for in the larger KBP corpus.

Our preliminary results in the slot filling track are given in Table 1. In addition, Stanford made two submissions to the cold-start knowledge base track:

Stanford1 A high precision system (patterns, openie, website, altnames) for both hop0 and hop1.

Stanford2 A high recall system: the union of all models for both hop0 and hop1.

Our preliminary results in the KB track are given in Table 2

A few observations can be made from these results. Most saliently, the high-recall systems consistently outperform the high-precision counterparts. This suggests a mismatch between our development and our test data – common for live competitions like KBP – and lends support the to hypothesis that further iterations of self-training to improve recall would further benefit the systems.

However, it's also worth noting that the high-precision systems are both high-precision and retain a reputable recall. The pattern extractor alone achieves nearly 60% precision with a recall above 10% – unmatched by any similar system.

Furthermore, the hop0 model combination finds what is very likely the correct tradeoff spot between precision and recall – it recovers much of the recall of the high recall system, while maintaining nearly the precision of patterns. This shows a clear promise for model combination, although the results also suggest that the challenge of finding the right training data for hop1 model combination remains outstanding.

Lastly, we observe that accuracy drops fairly substantially between the slotfilling track and the KB track. This suggests that the consistency and inference components in the slotfilling track are in fact very useful. An interesting area of future research would be to incorporate these sorts of constraints not on a per-query basis, but across the entire predicted knowledge base.

6 Conclusion

We have shown that a pattern-based relation extractor can perform competitively in the KBP slotfilling competition; and, more importantly, that a bootstrapped self-trained relation extractor built on top of these patterns is very competitive at the task. In addition, we've made the case for approaching the engineering task of KBP by doing as much computation as possible in a distributed relational database.

Acknowledgements

The authors gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) Deep Exploration and Filtering of Text (DEFT) Program under Air Force Research Laboratory (AFRL) contract no. FA8750-13-2-0040. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DARPA, AFRL, or the US government.

References

Gabor Angeli, Arun Chaganty, Angel Chang, Kevin Reschke, Julie Tibshirani, Jean Y. Wu, Osbert Bas-

		Hop 0			Hop 1			All		
System		P	R	F_1	P	R	F_1	P	R	F_1
Stanford1	(model combination)	52.2	20.1	29.1	26.4	4.6	7.8	46.8	14.4	22.0
Stanford2	(high precision)	58.5	14.0	22.6	32.2	4.5	7.8	51.8	10.5	17.4
Stanford3	(high recall)	34.3	**25.8**	**29.5**	21.0	**25.1**	**22.8**	27.9	**25.6**	**26.7**
Stanford4	(patterns only)	**59.5**	13.3	21.7	**40.8**	5.9	10.4	**54.3**	10.6	17.7

Table 1: A summary of Stanford's performance on the 2015 slot filling preliminary evaluation.

		Hop 0			Hop 1			All		
System		P	R	F_1	P	R	F_1	P	R	F_1
Stanford1	(high precision)	**53.0**	11.6	19.0	**36.7**	5.0	8.8	**48.7**	9.1	15.4
Stanford2	(high recall)	28.5	**21.9**	**24.8**	15.2	**25.4**	**19.0**	21.0	**23.2**	**22.1**

Table 2: A summary of Stanford's performance on the 2015 KB preliminary evaluation.

tani, Keith Siilats, and Christopher D. Manning. 2014a. Stanford's 2013 KBP system. In *TAC-KBP*.

Gabor Angeli, Julie Tibshirani, Jean Y. Wu, and Christopher D. Manning. 2014b. Combining distant and partial supervision for relation extraction. In *EMNLP*.

Gabor Angeli, Melvin Johnson Premkumar, and Christopher D. Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *ACL*.

Angel X Chang and Christopher D Manning. 2014. TokensRegex: Defining cascaded regular expressions over tokens. Technical report, Technical Report CSTR 2014-02, Department of Computer Science, Stanford University.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Feng Niu, Ce Zhang, Christopher Ré, and Jude W Shavlik. 2012. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. In *VLDS*, pages 25–28.

Nazneen Fatema Rajani, Vidhoon Viswanathan, Yinon Bentor, and Raymond J. Mooney. 2015. Stacked ensembles of information extractors for knowledge-base population. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*, pages 177–187.

Lev Ratinov, Dan Roth, Doug Downey, and Mike Anderson. 2011. Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1375–1384. Association for Computational Linguistics.

Benjamin Roth, Tassilo Barth, Michael Wiegand, Mittul Singh, and Dietrich Klakow. 2013. Effective slot filling based on shallow distant supervision methods. *TAC-KBP*.

Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. 2012. Multi-instance multi-label learning for relation extraction. In *EMNLP*.