# Stanford's 2014 Slot Filling Systems

**Gabor Angeli***, **Sonal Gupta***, **Melvin Jose***, **Christopher D. Manning***
**Christopher Ré***, **Julie Tibshirani***, **Jean Y. Wu***, **Sen Wu***, **Ce Zhang**†
*Stanford University; Stanford, CA 94305
{angeli, sonalg, melvinj, manning}@stanford.edu
{chrismre, jtibs, jeaneis, senwu}@stanford.edu
†University of Wisconsin; Madison, WI 53706
czhang@cs.wisc.edu

## Abstract

We describe Stanford's entry in the TAC-KBP 2014 Slot Filling challenge. We submitted two broad approaches to Slot Filling: one based on the DeepDive framework (Niu et al., 2012), and another based on the multi-instance multi-label relation extractor of Surdeanu et al. (2012). In addition, we evaluate the impact of learned and hard-coded patterns on performance for slot filling, and the impact of the partial annotations described in Angeli et al. (2014).

## 1 Introduction

We describe Stanford's two systems in the 2014 KBP Slot Filling competition. The first, and best performing system, is built on top of the DeepDive framework. We describe the system briefly below, and in more detail in Section 2; the central lesson we would like to emphasize from this system is that leveraging large computers allows for completely removing the information retrieval component of a traditional KBP system, and allows for quick turnaround times while processing the entire source corpus as a single unit. DeepDive offers a convenient framework for developing systems on these large computers, including defining the pre-processing pipelines (feature engineering, entity linking, mention detection, etc.) and then defining and training a relation extraction model.

The second system Stanford submitted is based around the MIML-RE relation extractor, following closely from the 2013 submission, but with the addition of learned patterns, and with MIML-RE trained fixing carefully selected manually annotated sentences. The central lesson we would like to emphasize from this system is that a relatively small annotation effort (10k sentences) over carefully selected examples can yield a surprisingly large gain in end-to-end performance on the Slot Filling task (4.4 $F_1$). We describe the two systems briefly below.

### 1.1 DeepDive

Stanford submitted two runs using the DeepDive framework for relation extraction (`http://deepdive.stanford.edu/`). The systems differed only in the threshold chosen for proposing relations. DeepDive is a framework for constructing trained systems whose goal is to make it easier to integrate domain-specific knowledge and user feedback. It runs in two phases: feature extraction, and statistical inference and learning. The first phase is implemented as steps in a pipeline, with intermediate results stored in a large distributed database. For example, once initialized with a *Sentences* table, mention detection runs a simple Python script on each row in the *Sentences* table and populates the resulting mentions in a *Mentions* table. Featurization can then take every pair of mentions from this mentions table, and run a featurization script to populate the features table. Importantly, the entire source corpus is processed as a single unit, without need for an information retrieval component.

Inference (and consequently learning) over the data is performed by *grounding* a factor graph via a set of database queries, and then running a Gibbs sampler to infer relation probabilities (Niu et al., 2011a). The model uses the distant supervision assumption, as in Mintz et al. (2009a).

Negative examples are constructed identically to the 2013 system, taking incompatible and incomplete relations with respect to the input knowledge base. The training corpus consists of only the source corpus for the year it is evaluated on – for the 2013 dev set and 2014 evaluation set, this consists of only the 2013 source corpus. In addition, snowball sampling over a large web corpus was used to collect additional named entity

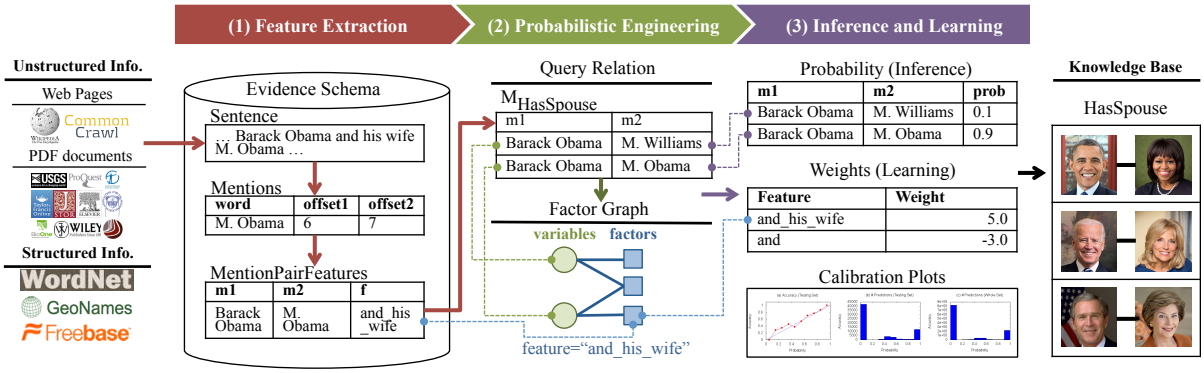


Figure 1: An overview of a system built with DeepDive that takes as input both structured and unstructured information and creates a knowledge base as the output. There are three phases of DeepDive's execution model: (1) Feature Extraction; (2) Probabilistic Engineering; and (3) Inference and Learning. Section 2 contains a more detailed walkthrough of these phases, and Figure 2 shows more details of how to conduct tasks in these three phases using SQL and script languages, e.g., Python.

triggers, and some high-precision relation triggers. We used Freebase as our source of training data, aligned to the set of KBP relations. Entity Linking was performed with a series of high-precision rules.

The system obtained an $F_1$ of 36.7 on the 2014 evaluation. In development runs, the system obtained an $F_1$ of 62 on the 2010 data, and 56 on the 2013 data.

### 1.2 MIML-RE

In addition to the DeepDive system, Stanford submitted three systems making use of the MIML-RE model. These are to a large extent similar to Stanford's submission last year, but with some key improvements.

First, the MIML-RE model was re-trained using selective human-annotated relation mention labels, as described in Angeli et al. (2014) and summarized in Section 4. Two submitted runs differ only by their use of the unmodified versus active learning models.

In addition, our model employed a somewhat extended set of hand-coded patterns, as well as a larger collection of automatically learned patterns (see Section 5). Two runs differ in their use of these patterns versus using only the MIML-RE relation extractor.

For training, we used both the 2010 and 2013 source corpora, as well as a 2013 dump of Wikipedia. For testing, we used only the official corpus for the evaluation year, as this was shown to perform better in the 2013 evaluation than using all three corpora.

The dev $F_1$ scores for each year for the best MIML-RE based system are as follows. Slot thresholds were trained on a per-relation basis to maximize the score on the 2013 development set. Our best MIML-RE based system obtained an $F_1$ of 32; active learning gained us an improvement of 4.4 $F_1$, while patterns gained us 1.9 $F_1$. Our scores on the development sets were:

| 2009 | 2010 | 2011 | 2012 | 2013 |
|---|---|---|---|---|
| 25.54 | 32.02 | 28.53 | 30.74 | 40.86 |

We proceed to describe our two approaches in more detail.

## 2 A Brief Overview of DeepDive

We briefly introduce the programming and execution model of DeepDive. There are three phases in DeepDive's execution model, as shown in Figure 1, and described below. All of the experiments on the KBP corpus were performed using a 1.5 TB memory computer; a distributed GreenPlum relational database (a distributed Postgres) was set up with 40 nodes on the computer. Therefore, it is relevant to point out that each of these phases occurs in parallel on 40 cores, and is run primarily if not entirely out of main memory.

**(1) Feature Extraction.** The input to this stage often contains both structured and unstructured information, and the goal is to produce a relational database that describes the features or signals of the data, which we call the *evidence schema*. We use the phrase evidence to emphasize that, in this stage, data is not required to be precisely correct; a fact which

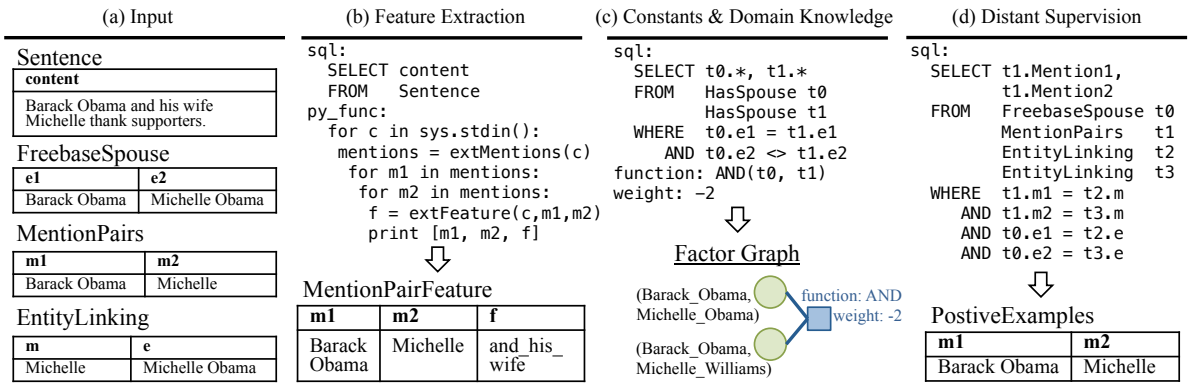


Figure 2: Illustration of popular operations in DeepDive. (a) Prepare data sets in relational form that can be used by DeepDive. (b) Generate labels using distant supervision with SQL; (c) Integrate constraints with SQL and logic functions; (d) Extract features with SQL and script languages (e.g., Python).

is very relevant for distantly supervised systems.

**(2) Probabilistic Engineering.** The goal of this phase is to transform the evidence schema into a probabilistic model, specifically a factor graph that specifies:

- The set of random variables that the user wants to model. To define the set of random variables in DeepDive is to create new relations called query relations, in which each tuple corresponds to one random variable. This operation can be done by using SQL queries on existing relations.
- How those random variables are correlated; e.g., "The mention 'Obama' refers to the president is correlated with the random variable that indicates whether 'Obama' is a person." To specify *how*, the user specifies a factor function. Although our submissions do not make use of more complex constraints, they are easy to encode in the framework. Figure 2(c) shows a rough example that describes the intuition that *"people tend to be married to only a single person."* One (of many ways) to say this is to say that there is some correlation between pairs of married tuples; i.e., using the logical function AND(t0, t1) returns 1 in possible worlds in which both married tuples are true and 0 in others. DeepDive then learns the "strength" of this correlation from the data, which is encoded as weight.[1] Here, $-2$ indicates that it is less likely that both married tuples are correct. This phase is also where the user can write logical constraints, e.g., hard functional constraints.

Our previous work has shown that this grounding phase (Niu et al., 2011b) can be a serious bottleneck if one does not use scalable relational technology. We have learned this lesson several times.

**(3) Inference and Learning.** This phase is largely opaque to the user: it takes the factor graph as input, estimates the weights, performs inference, and produces the output database along with a host of diagnostic information, notably calibration plots (see Fig. 3). More precisely, the output of DeepDive is a database that contains each random variable declared by the user with its marginal probability. For example, one tuple in the relation HasSpouse might be (Barack Obama, Michelle Obama), and ideally, we hope that DeepDive outputs a larger probability for this tuple as output.

## 2.1 Operations to Improve Quality in DeepDive

We describe three routine tasks that a user performs to improve a DeepDive based system.

[1] A weight is roughly the log odds, i.e., the $\log \frac{p}{1-p}$ where $p$ is the marginal probability of this random variable. This is standard in Markov Logic Networks (Richardson and Domingos, 2006), on which much of DeepDive's semantics are based.

**Feature Extraction.** DeepDive's data model allows the user to use any scripting language for feature extraction. Figure 2(b) shows one such example using Python. One baseline feature that is often used in relation extraction systems is the word sequence between mention pairs in a sentence (Mintz et al., 2009b; Hoffmann et al., 2010), and Figure 2(b) shows an example of extracting this feature. The user first defines the input to the feature extractor using an SQL query, which selects all available sentences. Then the user defines a UDF that will be executed for each tuple returned by the SQL query. In this example, the UDF is a Python function that first reads a sentence from STDIN, extracts mentions from the sentence, extracts features for each mention pair, and outputs the result to STDOUT. DeepDive will then load the output of this UDF to the MentionPairFeature relation.

**Constraints and Domain Knowledge.** One way to improve a KBP system is to integrate domain knowledge. DeepDive supports this operation by allowing the user to integrate constraints and domain knowledge as correlations among random variables, as shown in Figure 2(c).

Imagine that the user wants to integrate a simple rule that says "one person is likely to be the spouse of only one person." For example, given a single entity "Barack_Obama," this rule gives positive preference to the case where only one of (Barack_Obama, Michelle_Obama) and (Barack_Obama, Michelle_Williams) is true. Figure 2(c) shows one example of implementing this rule. The SQL query in Figure 2(c) defines a view in which each tuple corresponds to two relation candidates with the same first entity but different second entities. The function AND(t0, t1) defines the "type of correlation" among variables, and the weight "-2" defines the strength of the correlation. This rule indicates that the it is less likely that both (Barack_Obama, Michelle_Obama) and (Barack_Obama, Michelle_Williams) are true (i.e., when AND(t0, t1) returns 1). Typically, DeepDive is used to learn the weights from data.

**Distant Supervision.** Like the MIML-RE system, DeepDive is built on top of the distant supervision assumption, allowing more data to be incorporated easily.

As we have described, the user has at least the above three ways to improve the system and is free to use one or a combination of them to improve the system's quality. The question we address next is, *"What should the user do next to get the largest quality improvement in the KBP system?"*

## 3 Calibration Plots in the DeepDive

A DeepDive system is only as good as its features and rules. In the last two years, we have found that understanding which features to add is the most critical—but often the most overlooked—step in the process. Part of this process is the usual fine-grained error analysis; however, we have found that a macro-error analysis can be useful to guard against statistical errors and give an at-a-glance description of an otherwise prohibitively large amount of output.

In DeepDive, *calibration plots* are used to summarize the overall quality of the results. Because DeepDive uses a joint probability model, each random variable is assigned a marginal probability. Ideally, if one takes all the facts to which DeepDive assigns a probability score of $0.95$, then $95\%$ of these facts are correct. We believe that probabilities remove a key element: the developer reasons about features, not the algorithms underneath. This is a type of *algorithm independence* that we believe is critical.

DeepDive programs define one or more test sets for each relation, which are essentially a set of labeled data for that particular relation. This set is used to produce a calibration plot. Figure 3 shows an example calibration plot for the another DeepDive application: PaleoDeepDive, which provides an aggregated view of how the system behaves. By reading each of the subplots, we can get a rough assessment of the next step to improve our system. We explain each component below.

As shown in Figure 3, a calibration plot contains three components: (a) accuracy, (b) # predictions (test set), which measures the number of extractions in the test set with a certain probability; and (c) # predictions (whole set), which measures the number of extractions in the whole set with a certain probability. The test set is assumed to have labels so that we can measure accuracy, while the whole set does not.

**(a) Accuracy.** To create the accuracy histogram, we bin each fact extracted by DeepDive on the test set by the probability score assigned to each fact; e.g., we round to the nearest value in the set $k/10$ for $k = 1, \ldots, 10$. For each bin, we com-

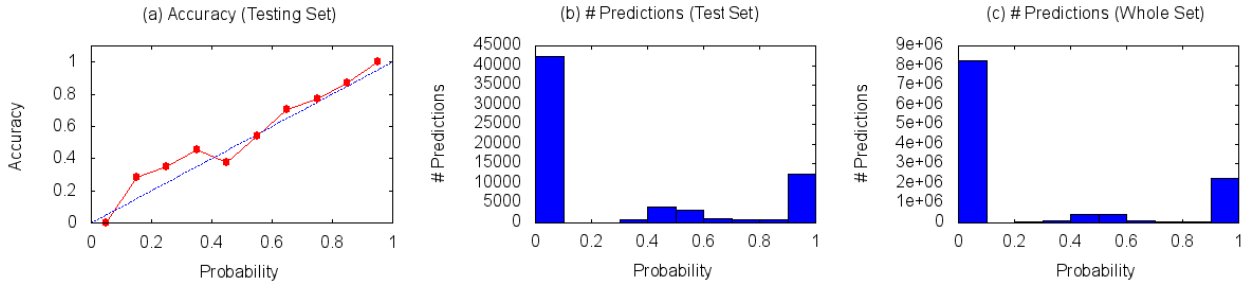


Figure 3: Illustration of calibration plots in DeepDive.

pute the fraction of those predictions that is correct. Ideally, this line would be on the (0,0)-(1,1) line, which means that the DeepDive-produced probability value is calibrated, i.e., it matches the *test-set accuracy*. For example, Figure 3(a) shows a curve for calibration. Differences in these two lines can be caused by noise in the training data, quantization error due to binning, or sparsity in the training data.

**(b) # Predictions (Testing Set).** We also create a histogram of the number of predictions in each bin. In a well-tuned system, the # Predictions histogram should have a "U" shape. That is, most of the extractions are concentrated at high probability and low probability. We do want a number of low-probability events, as this indicates DeepDive is considering plausible but ultimately incorrect alternatives. Figure 3(b) shows a U-shaped curve with some masses around 0.5-0.6. Intuitively, this suggests that there is some hidden type of example for which the system has insufficient features. More generally, facts that fall into bins that are not in (0,0.1) or (0.9,1.0) are candidates for improvements, and one goal of improving a KBC system is to "push" these probabilities into either (0,0.1) or (0.9,1.0). To do this, we may want to sample from these examples and add more features to resolve this uncertainty.

**(c) # Predictions (Whole Set).** The final histogram is similar to Figure 3(b), but illustrates the behavior of the system, for which we do not have any training examples. We can visually inspect that Figure 3(c) has a similar shape to (b); If not, this would suggest possible overfitting or some bias in the selection of the hold-out set.

Next, we proceed to describe our MIML-RE based KBP system; in particular, we describe the improvements to the system made since the 2013 entry.

## 4 Active Learning in MIML-RE

A key improvement in the 2014 KBP entry is the re-training of MIML-RE incorporating fixed labels for uncertain sentences. The approach is described in more detail in Angeli et al. (2014). We describe the process for incorporating these sentence-level statistics, and the selection criteria used for selecting sentences to annotate. This work yielded a net gain of 4.4 $F_1$ in the evaluation.

### 4.1 Training with sentence-level annotations

Following Surdeanu et al. (2012), MIML-RE is trained through hard discriminative Expectation Maximization, inferring the latent $z$ values in the E-step and updating the weights for both the $z$ and $y$ classifiers in the M-step. During the E-step, we constrain the latent $z$ to match our sentence-level annotations when available.

We describe three criteria for selection examples to annotate. The first – sampling uniformly – is a baseline for our hypothesis that intelligently selecting examples is important. For this criterion, we select mentions uniformly at random from the training set to annotate. The other two criteria rely on a metric for disagreement provided by QBC; we describe our adaptation of QBC for MIML-RE as a preliminary to introducing these criteria.

### 4.2 QBC For MIML-RE

We use a version of QBC based on bootstrapping (Saar-Tsechansky and Provost, 2004). To create the committee of classifiers, we re-sample the training set with replacement 7 times and train a model over each sampled dataset. We measure disagreement on $z$-labels among the classifiers using a generalized Jensen-Shannon divergence (McCallum and Nigam, 1998), taking the

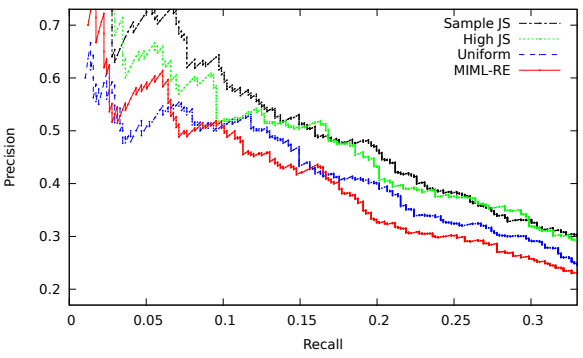


Figure 4: A comparison of models trained with various selection criteria on the evaluation of Surdeanu et al. (2012), all initialized with the corresponding supervised classifier.

average KL divergence of all classifier judgments.

We first calculate the mention-level confidences. Note that $z_i^{(m)} \in M_i$ denotes the latent variable in entity pair $i$ with index $m$; $\mathbf{z}_i^{(-m)}$ denotes the set of all latent variables except $z_i^{(m)}$:

$$p(z_i^{(m)}|\mathbf{y_i}, \mathbf{x_i}) = \frac{p(\mathbf{y_i}, z_i^{(m)}|\mathbf{x_i})}{p(\mathbf{y_i}|\mathbf{x_i})} = \frac{\sum_{\mathbf{z_i^{(-m)}}} p(\mathbf{y_i}, \mathbf{z_i}|\mathbf{x_i})}{\sum_{z_i^{(m)}} p(\mathbf{y_i}, z_i^{(m)}|\mathbf{x_i})}.$$

Notice that the denominator just serves to normalize the probability within a sentence group. We can rewrite the numerator as follows:

$$\begin{aligned}&\sum_{\mathbf{z_i^{(-m)}}} p(\mathbf{y_i}, \mathbf{z_i}|\mathbf{x_i}) \\ &= \sum_{\mathbf{z_i^{(-m)}}} p(\mathbf{y_i}|\mathbf{z_i}) p(\mathbf{z_i}|\mathbf{x_i}) \\ &= p(z_i^{(m)}|\mathbf{x_i}) \sum_{\mathbf{z_i^{(-m)}}} p(\mathbf{y_i}|\mathbf{z_i}) p(\mathbf{z_i^{(-m)}}|\mathbf{x_i}).\end{aligned}$$

For computational efficiency, we approximate $p(\mathbf{z_i^{(-m)}}|\mathbf{x_i})$ with a point mass at its maximum. Next, we calculate the Jensen-Shannon (JS) divergence from the $k$ bootstrapped classifiers:

$$\frac{1}{k} \sum_{c=1}^{k} \mathrm{KL}(p_c(z_i^{(m)}|\mathbf{y_i}, \mathbf{x_i}) || p_{\text{mean}}(z_i^{(m)}|\mathbf{y_i}, \mathbf{x_i})) \quad (1)$$

where $p_c$ is the probability assigned by each of the $k$ classifiers to the latent $z_i^{(m)}$, and $p_{\text{mean}}$ is the average of these probabilities. We use this metric to capture the *disagreement* of our model with respect to a particular latent variable. This is then used to inform our selection criteria.

| System | P | R | $F_1$ |
|---|---|---|---|
| Mintz++ | **41.3** | 28.2 | 33.5 |
| MIML + Dist | 38.0 | 30.5 | 33.8 |
| Supervised + SampleJS | 33.5 | **35.0** | **34.2** |
| MIML + Sup | 35.1 | 35.6 | 35.5 |
| MIML + Sup + SampleJS | 39.4 | 36.2 | 37.7 |

Table 1: A comparison of the best performing supervised classifier with other systems on the 2013 development set. The top section compares the supervised classifier with prior work. The lower section highlights the improvements gained from initializing MIML-RE with a supervised classifier.

We note that QBC may be especially useful in our situation as our objective is highly nonconvex. If two committee members disagree on a latent variable, it is likely because they converged to different local optima; annotating that example could help bring the classifiers into agreement.

The second selection criterion we consider is the most straightforward application of QBC – selecting the examples with the highest JS disagreement. This allows us to compare our criterion, described next, against an established criterion from the active learning literature.

### 4.3 Sample by JS Disagreement

We propose a novel active learning sampling criterion that incorporates not only disagreement but also *representativeness* in selecting examples to annotate. Prior work has taken a weighted combination of an example's disagreement and a score corresponding to whether the example is drawn from a dense portion of the feature space (e.g., McCallum and Nigam (1998)). However, this requires both selecting a criterion for defining density (e.g., distance metric in feature space), and tuning a parameter for the relative weight of disagreement versus representativeness.

Instead, we account for choosing representative examples by sampling without replacement proportional to the example's disagreement. Formally, we define the probability of selecting an example $z_i^{(m)}$ to be proportional to the Jensen-Shannon divergence in (1). Since the training set is an approximation to the prior distribution over examples, sampling uniformly over the training set is

| Slot | Dependency Pattern |
|---|---|
| per:employee_of | ENTITY (nsubjpass) ← dropped →(prep_from) {ner:/ORGANIZATION/}=FILL |
| per:country_of_death | {ner:/COUNTRY/}=FILL (nsubj) ← mourns →(dobj) ENTITY |
| org:founded_by | founders →(conj_and) {ner:/PERSON/}=FILL →(nn) ENTITY |

Table 2: Some examples of learned dependency patterns used by the system. The format of the patterns is in SemGrex format over Stanford Dependencies, where ENTITY and FILL are the entity head token and the slot fill head token respectively.

an approximation to sampling from the prior probability of seeing an input $x$. We can view our criterion as an approximation to sampling proportional to the product of two densities: a prior over examples $x$, and the JS divergence mentioned above.

### 4.4 Analysis of Selection Criteria

It's worth emphasizing at least three results from this line of work. First, that the selection criteria for annotating examples is very important for performance. Figure 4 shows the performance of the model using various selection criteria; we note that highJS and sampleJS noticeably outperform the uniform criterion. Second, that MIML-RE is very sensitive to initialization. To illustrate, the gains in Figure 4 disappear entirely if the model is initialized with Mintz++ (as in the original paper); gains only appear if the model is initialized from a supervised classifier. Lastly, it's worth noting that this supervised classifier, used on its own, performs surprisingly well on the 2013 development set. Table 1 shows the system's performance using distant supervision, vanilla MIML-RE (33.8 $F_1$ on 2013), and a supervised classifier trained from the SampleJS examples (34.2 $F_1$). This suggests that it may be reasonable for newcomers to the task to train a relatively simple classifier, and perhaps focus on downstream processes; although, of course, the best classifiers are still those which incorporate the examples into MIML-RE (37.7 $F_1$).

## 5 Learning Patterns to Augment MIML-RE

To extract slot fillers, we also used two types of patterns in our system: surface word lexico-syntactic patterns (such as, "X was married to Y") and dependency patterns (such as, "X (nsubj) ← marry → (dobj) Y"). Patterns often are useful for extracting slots with high precision, even though with low recall. Usually recall is increased by iteratively learning patterns by using the extractions learned in the previous iteration as labeled data. The lexico-syntactic patterns were hand written and the dependency patterns were both hand written and learned.

We learned the dependency patterns for each slot independently. For each slot, first, the the known entities and slot fillers were matched to sentences as described above. For each pair of entity head word and slot filler head word in a matched sentence, we extracted the shortest path between them in the dependency tree of the sentence as the candidate pattern. If any of the entity or the slot filler were tagged with a named entity tag, we included the restriction in the pattern. The candidate patterns were weighted by the measure:

$$\frac{Pos}{\sqrt{Neg + All + 2}}$$

where $Pos$ is the number of times the pattern matched the correct pair of entity and slot filler, $Neg$ is the number of the times the pattern extracted a pair of entity and slot filler that were related by another slot, and $All$ is the total number of times the pattern matched the sentences. Patterns above a certain threshold were selected. We learned a total of 1073 patterns for all slots. Due to lack of time, we ran the system on a small sample of the training data for one iteration. In future, we plan to extend this to run pattern learning for multiple iterations and improve pattern scoring using the approach of Gupta and Manning, 2014.

Some examples of the learned dependency patterns are given in Table 2.

## 6 Conclusion

We have described Stanford's two systems in the 2014 KBP competition: one based on DeepDive, and another around the MIML-RE relation extractor. We hope to convey at least two central messages: First, that there is an undervalued benefit to scaling computing resources proportional to the size of the task being solved. That is to say, using DeepDive to effectively harness large computers

to quickly iterate features and debug a KBP system is very valuable for creating top-performing systems. Second, that carefully annotating selected difficult examples can, with minimal cost and human intervention, yield large improvements in relation extraction accuracy.

## References


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

Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. 2010. Learning 5000 relational extractors. In *ACL*.

Andrew McCallum and Kamal Nigam. 1998. Employing EM and pool-based active learning for text classification. In *ICML*.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009a. Distant supervision for relation extraction without labeled data. In *ACL*.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009b. Distant supervision for relation extraction without labeled data. In *ACL*.

Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. 2011a. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *Proceedings of the VLDB Endowment*, 4(6):373–384.

Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. 2011b. Tuffy: Scaling up statistical inference in Markov logic networks using an RDBMS. *PVLDB*.

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

Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine Learning*.

Maytal Saar-Tsechansky and Foster Provost. 2004. Active sampling for class probability estimation and ranking. *Machine Learning*, 54(2):153–178.

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