Event Extraction Using Distant Supervision

Kevin Reschke* kreschke@stanford.edu

December 14, 2012

1 Introduction

The purpose of this paper is to explore a distant supervision approach to event extraction—that is, the extraction of template-based facts about events from unstructured text. In a distantly supervised system, training texts are labeled automatically (and noisily) by leveraging an existing database of known facts. This approach has been applied successfully to the extraction of binary relations such as a person's employer or a film's director (e.g., Surdeanu et al., 2011), but it has not previously been applied to event extraction.

Concretely, I develop a system which extracts airplane crash events from a corpus of news documents using Wikipedia infoboxes as a source for distant supervision.[1] The news corpus is a collection of newswire texts spanning 1988 to the present.[2] I selected 80 plane crash infoboxes in that time frame from Wikipedia: 36 for training; 8 for development; 40 for testing. An example is shown in Table 1. At training time, facts from the 36 training infoboxes are used to automatically label training sentences from the news corpus. At test time, the system takes the flight number of a test infobox as input and produces values for the seven template slots as output.

The paper is structured as follows. First I detail the event extraction process and my distant supervision approach. Then I describe a series of experiments testing various models within this framework.

Table 1: Sample plane crash infobox.

Slot Type	**Slot Value**
Flight Number	Flight 967
Operator	Armavia
Aircraft Type	Airbus A320-211
Crash Site	Alder-Sochi Airport, Black Sea
Passengers	105
Crew	8
Fatalities	113
Injuries	0
Survivors	0

2 Event Extraction

At test time, event extraction has three steps. 1) Candidate Generation: run Named-Entity Recognition software[3] on relevant documents from the corpus to identify candidate mentions. In this setting, I use the flight number as a proxy for document relevance; if a document has the string "Flight x", then the document is considered relevant to the Flight x plane crash event. 2) Mention Classification: Classify candidate mentions based on contextual features (surrounding unigrams, syntactic dependencies, etc). The label space is the set of slot types in the event template, plus NIL for mentions which don't fit any slot. 3) Label Aggregation: Merge labels from different mentions of the same value to produce final slot value predictions. For the bulk of this paper I assume Exhaustive Aggregation—that is, all non-NIL labels are included in the final prediction. But see Section 4.6 for an improved aggregation scheme.

As an example of the test time procedure, suppose we are extracting facts about the crash of Flight 13, and we identify the candidate *Mississippi* in the sentence "*Flight 13 crashed in Mississippi.*" A properly trained mention classifier will give this mention the label ⟨CrashSite⟩ based on the words *crashed* and *in* which precede it. Now suppose that over all of the mentions of *Mississippi*, three mentions were classified as ⟨CrashSite⟩, two mentions were classified as NIL, and one mention was (incorrectly) classified as ⟨Operator⟩. Label aggregation will gives us the final predictions that *Mississippi* is both the crash site and the operator of Flight 13.

3 Distant Supervision

The mention classification step mentioned above requires a trained classifier—this is where distant supervision comes in. In a fully supervised approach, we would

*I thank Mihai Surdeanu, Martin Jankowiak, David McClosky, and Christopher Manning for guidance on this project.

[1] http://en.wikipedia.org/wiki/Help:Infobox

[2] I use Gigaword-5, Tipster-1, Tipster-2, and Tipster-3. See www.ldc.upenn.edu/.

[3] Stanford CoreNLP NER: nlp.stanford.edu/software/CRF-NER.shtml

have humans label a set of mentions and train a classifier on those gold labels, but in this case supervision comes indirectly from a set of training infoboxes.

How are infoboxes mapped to text labels? Consider first the relation extraction setting for which distant supervision was first introduced (Mintz et al. 2009). In distant supervision for relation extraction, the training set is a database of binary relations such as ⟨*Steve Jobs*, *Apple*⟩ for the *FounderOf* relation. Training sentences are labeled by the following rule: if both entities appear in a single sentence, that sentence is a positive instance of the relation; otherwise it is NIL. This rule ensures that we apply the label *FounderOf* to the sentence "*Steve Jobs co-founded Apple in 1976*," but not to random, unrelated mentions of *Apple*.

Unfortunately, for event extraction, this sentence level rule doesn't work. We might think of template slots as binary relations between the slot value and the flight number, but as we see below, slot values often occur in isolation.

- The plane went down in central **Texas**.
- **10** died and **30** were injured in yesterdays tragic incident.

Instead, I adopt a document-level heuristic. Given a slot value and flight number pair from a training infobox, if the slot value occurs in the same document as the flight number, mark the mention as a positive example for that slot type. Since we're using the presence of the flight number as a heuristic for document relevance, this is equivalent to only labeling mentions that occur in documents relevent to the training event.

Named entities that occur in a relevant document but don't match any slot values are given NIL labels. After the process is complete, NIL examples are subsampled, resulting in a training set with a 50/50 split between NIL and non-NIL examples.

Due to the heuristic nature of this noisy labeling scheme, the resulting training examples are extremely noisy. In fact, training data noise is a hallmark of distant supervision. Noise is prevalent in the relation extraction setting—for example, any sentence containing both *Apple* and *Steve Jobs* will be marked with the *FounderOf* relation, even the sentence "*Steve Jobs was fired from Apple in 1985.*" Likewise there are many false labelings in the event extraction setting, such as when an airline's name is mentioned, but the sentence has nothing to do with that airline being the operator in the target crash event. In fact, by manually checking 50 examples from each slot type, I found that 39% were wrong. This high degree of noise is a central challenge to the distant supervision approach, and will be a theme that resurfaces in the experiments that follow.

4 Experiments

Having introduced the general framework for distantly supervise event extraction, in this section I present experiments testing various models in this framework. For all test-set scores that I present, the model has been tuned to maximize F_1-Score on the 8-infobox dev-set.

4.1 Experiment 1: Simple Local Classifier

First I use multi-class logistic regression to train a model which classifies each mention independently, using the noisy training data described above. Features include the mention's part of speech, named entity type, surrounding unigrams, incoming and outgoing syntactic depencies, the location within the document and the mention string itself.[4] For example, the *Mississippi* example from Section 2 might have the following binary features: LexIncEdge-prep_in-crash-VBD, UnLexIncEdge-prep_in-VBD, PREV_WORD-in, 2ndPREV_WORD-crash, NEType-LOCATION, Sent-NEType-ORGANIZATION, etc.

I compare this local classifier with a majority class baseline. Table 2 shows the distribution of labels in the distantly generated training data. The majority baseline simply picks the majority class for each named entity type: ⟨Site⟩ for locations, ⟨Operator⟩ for organizations, and ⟨Fatalities⟩ for numbers.

Table 2: Label frequency in noisy training data.

Label	**Frequency**	**Named Entity Type**
<NIL>	19196	
Site	10365	LOCATION
Operator	4869	ORGANIZATION
Fatalities	2241	NUMBER
Aircraft_Type	1028	ORGANIZATION
Crew	470	NUMBER
Survivors	143	NUMBER
Passengers	121	NUMBER
Injuries	0	NUMBER

To compare performance on the final slot prediction task, I define precision and recall as follows. Precision is the number of correct guesses over the total number of guesses. Recall is the number of slots correctly filled over the number of findable slots. A slot is *findable* if its true value appears somewhere as a candidate mention. In other words, we don't penalize the extraction model for missing a slot that either wasn't in the corpus or didn't occur under our heuristic notion of *relevant* document.

[4]Parsing, POS tagging, and NER: Stanford Core NLP. nlp.stanford.edu/software/corenlp.shtml

Table 3: Performance of Local Classifier vs. Majority baseline.

	Precision	Recall	F_1-Score
Maj. Baseline	0.026	0.237	0.047
Local Classifier	**0.159**	**0.407**	**0.229**

Table 4: Accuracy of local classifier by slot type

Site	8/50 = 0.16
Operator	5/25 = 0.20
Fatalities	7/35 = 0.20
Aircraft_Type	4/19 = 0.21
Crew	15/170 = 0.09
Survivors	1/1 = 1.0
Passengers	11/42 = 0.26
Injuries	0/0 = NA

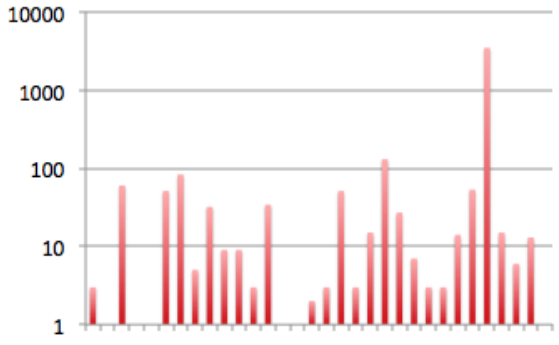
Some slots, such as ⟨CrashSite⟩, can have multiple values. This metric only requires identifying one or more of them.

The performance of the local and majority classifiers are shown in table 3. The test set contained 40 test infoboxes for a total of 135 findable slots. The local classifier significantly outperforms the baseline. Table 4 breaks down the accuracy of the local classifier by slot type.

4.2 Experiment 2: Training set bias

Recall that during distant supervision, training examples are generated for a training infobox for every document relevent to that infobox. Figure 1 shows the frequency of relevant documents for each infobox. We see that most infoboxes selected a hundred or so documents, but one event in particular had several thousand relevant documents. (Incidentally, this high frequency event is the crash of Pan Am Flight 103—a.k.a. the Lockerbie bombing). Consequently a large portion of the noisy training examples are due to this single event.

Figure 1: Relevant documents per training infobox



To see whether this imbalance hurts system performance, I trained a new local classifier using only 200 documents for Flight 103. This improved dev-set precision slightly (0.26 vs. 0.30) but hurt recall (0.13 vs. 0.30). However, this effect appears to be a consequence of less training data, not training set imbalance. I trained another model using all 3000+ Flight 103 documents but only using 24 training events instead of 32. The results were comparable (prec:0.33; rec:0.13).

4.3 Experiment 3: Sentence relevance

With the simple local classifier described in Section 4.1, a lot of errors come from sentences that are irrelevant to the event. For example, *Northwest Airlines* was classified as ⟨Operator⟩ in the sentence below, but in fact neither the sentence nor Northwest Airlines had any relevance to the target plane crash event.

- Clay Foushee, vice president for flying operations for **Northwest Airlines**, which also once suffered from a coalition of different pilot cultures, said the process is "long and involved and acrimonious."

These errors would be mitigated if we could eliminate irrelevant sentences from consideration during mention classification. To this end, I trained a binary sentence relevance classifier over unigram and bigram features. Like the mention models, the relevance classifier was distantly supervised—during noisy labeling, a training sentence was marked relevant if it contained at least one slot value from one event.

Two new local models incorporate this sentence relevance signal. In LocalWithHardSent, all mentions from non-relevant sentences are classified NIL. In LocalWithSoftSent, sentence relevance is used as a feature in mention classification.

The test-set results for these new models are shown in Table 5. Surprisingly, the new models significantly underperform the simple local model. One explanation is that the distant supervision for sentence relevance was just too noisy to train a good classifier. Still, it is surprising that this *hurt* performance. If we can't get a signal from sentence relevance, we would expect LocalWithSoftSent to ignore the relevance feature, not to perform worse.

Table 5: Classifiers using sentence relevance.

	Precision	Recall	F_1-Score
Local Classifier	**0.16**	**0.41**	**0.23**
LocalWithHardSent	0.12	0.24	0.16
LocalWithSoftSent	0.11	0.23	0.15

I also applied sentence relevance to the model upgrades described in Sections 4.4 and 4.5. In every case, sentence relevance hurt performance.

4.4 Experiment 4: Pipeline model

So far I have presented only local models which classify mentions independently, but in reality there are dependencies between mention labels. For example, ⟨Crew⟩ and ⟨Passenger⟩ go together; ⟨Site⟩ often follows ⟨Site⟩; and ⟨Fatalities⟩ never follows ⟨Fatalities⟩:

- **4** crew and **200** passengers were on board.
- The plane crash landed in **Beijing**, **China**.
- * **20** died and **30** were killed in last Wednesday's crash.

In this experiment, I compare a pipeline model with the simple local model. In the pipeline model, mentions in a sentence are classified sequentially. At each step, the label of the previous non-NIL mention is used as a feature for the current mention. At training time, this is the previous mention's noisy "gold" label. At test time, this is the classifier's output on the previous mention.

The pipeline model boosted recall, but took a slight hit on precision. Table 6 shows test-set results. A qualitative analysis of the pipeline model's feature weights revealed that the classifier learned the patterns mentioned above, as well as others. However, this wasn't enough to significantly improve performance.

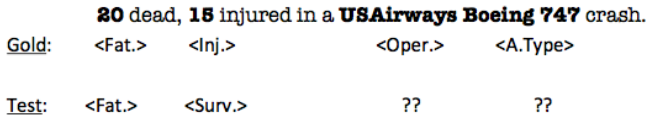
Table 6: Performance of Pipeline Model

	Precision	**Recall**	**F_1-Score**
Local Model	**0.159**	0.407	**0.229**
Pipeline Model	0.154	**0.422**	0.226

4.5 Experiment 5: Joint model (Searn)

There is a common problem with pipeline models which may explain the performance reported above: pipeline models propagate error. Consider the example in Figure 2. At training time, *USAirways* has the feature PREV-LABEL-INJURY. But suppose that at inference time, we mislabel *15* as ⟨Survivors⟩. Now *USAirways* has the feature PREV-LABEL-SURVIVOR, and we are in a feature space that we never saw in training. Thus we are liable to make the wrong classification for *USAirways*. And if we make the wrong decision there, then again we are in an unfamiliar feature space for *Boeing 747* which may lead to another incorrect decision.

Figure 2: Error propagation in pipeline classification.



This error propagation is particularly worrisome in our distant supervision setting due to the high amount of noise in the training data. To extend the example, suppose instead that at distant supervision time, *15* was given the incorrect "gold" label ⟨Fatalities⟩. Now at test time, we might correctly classify *15* as ⟨Injuries⟩, but this will put us in an unseen feature space for subsequent decisions because *USAirways* saw ⟨Fatalities⟩ at training time, not ⟨Injuries⟩.

An ideal solution to this error propagation problem should do two things. First, it should allow suboptimal local decisions that lead to optimal global decisions. For the previous example, this means that our choice for *15* should take into account our future performance on *USAirways* and *Boeing 747*. Second, models of sequence information should be based on actual classifier output, not gold labels. This way we won't be in an unfamiliar feature space each time our decision differs from the gold label.

In essence, we want a joint mention model—one which optimizes an entire sequence of mentions jointly rather than one at a time. To this end, I applied the Searn algorithm (Daumé, 2006) to mention classification. Searn stands for *Search-based Structured Prediction*. At a high level, Searn is an iterative solution to the following chicken-and-egg problem: we want a set of decision costs based on an optimal global policy; and we want a global policy to be learned from these decision costs. A sketch of the algorithm is given in Figure 3. The *current hypothesis* is an interpolation of the optimal policies learned at each iteration. The algorithm is seeded with an initial policy that simply chooses gold labels (akin to a pipeline approach). At each iteration, the hypothesis moves away the gold policy, and ultimately this inital policy is dropped from the final hypothesis.

At each iteration, Searn requires a cost-sensitive classifier. For this I follow Vlachos and Craven (2011) in using the algorithm in Cramer et al. (2006), which amounts to a passive-aggressive multiclass perceptron. Searn has a number of hyperparameters. By hill climbing on my development set, I arrived at the following settings: 4 Searn iterations; 8 perceptron epochs per iteration; interpolation $\beta = 0.3$; perceptron *aggressiveness* $= 1.0$.

The test-set results comparing Searn to the pipeline and local models are shown in Table 7. Searn clearly

Figure 3: The Searn algorithm for mention classification.

A Searn Iteration:

For a mention *m* in sentence s:

Step 1: Classify all mentions left of *m* using current hypothesis.

Step 2: Use these decision as features for *m*.

Step 3: Pick a label *y* for *m*.

Step 4: Given *m* = *y*, classify all mentions right of *m* with current hypothesis

Step 5: Compare whole sentence to gold labels.

Step 6: Set cost for *m* = *y* to #errors

Repeat 2-5 for all possible labels.

Repeat 1-6 for all mentions and sentence.

You now have a cost vector for each mention. Train a Cost Sensitive Classifier then interpolate with current hypothesis.

Table 7: Performance Searn, Pipeline and Local models.

	Precision	**Recall**	**F_1-Score**
Local Model	0.159	0.407	0.229
Pipeline Model	0.154	**0.422**	0.226
Searn Model	**0.213**	**0.422**	**0.283**

dominates. A likely explanation is that Searn was able to model the inter-mention dependencies described in Section 4.4 while avoiding the error propagation endemic to the pipeline model.

4.6 Experiment 6: Noisy-OR Aggregation

As described in Section 2, the final step in event extraction, after mention classification, is label aggregation. So far I have assumed exhaustive aggregation—as long as at least one mention of a value gets a particular slot label, we use that value in our final slot-filling decision. Intuitively, this approach is suboptimal, especially in a noisy data environment where we are more likely to misclassify the occassional mention. In fact, a proper aggregation scheme can act as fortification against noise induced misclassifications.

With this in mind, I adopted a Max Aggregation scheme: when multiple non-NIL labels occur for mentions of a particular value, choose the label that occurs most often. Interestingly, this scheme had little effect (+0.01 precision) on system performance. It turns out the scenario with multiple non-NIL labels is relatively rare. Instead, it was most common to see mention labels split between a single non-NIL label and NIL. In this case, Exhaustive and Max Aggregation always return the non-NIL label, but we would prefer a scheme that can select NIL under the right cirumstances.

To achieve this I use Noisy-OR Aggregation. The key idea is that classifiers gives us distributions over labels, not just hard assignments. A simplified example is given below.

- Stockholm ⟨NIL:0.8; Site: 0.1, Crew:0.01, etc.⟩
- Stockholm ⟨Site: 0.5; NIL: 0.3, Crew:0.1, etc.⟩

Given a distribution over labels ℓ for each mention m in M, the set of mentions for a particular candidate value, we can compute Noisy-OR for each label as follows.

$$NoisyOr(\ell) = P(\ell|M) = 1 - \prod_{m \in M} 1 - Pr(\ell|m)$$

If the Noisy-OR of a label is above some threshold, we use the label—otherwise we return NIL. I found 0.9 to be an optimal threshold by tuning on the development set. Table 8 shows test-set results comparing Noisy-OR and Exhaustive Aggregation on the simple local classifier. We see that Noisy-OR improves precision while decreasing recall. (This is to be expected as Noisy-OR is strictly more conservative—i.e. NIL-prefering—than Exhaustive Aggregation). In terms of F_1-Score, Noisy-OR Aggregation is the better method.

Table 8: Two Label Aggregation schemes applied to Local Model.

	Precision	**Recall**	**F_1-Score**
Exhaust. Agg.	0.16	**0.41**	0.23
Noisy-OR. Agg.	**0.19**	0.39	**0.26**

5 Conclusion

I have presented a distant supervision approach to event extraction using plane crash events as a test bed and described how various models perform in the framework. In future work, I will explore better notions of document relevance (to replace the naive "contains flight number" heuristic), and I will look into better methods for applying sentence relevance to the system. Additionally, I will apply this framework to the MUC-4 shared event extraction task to compare distantly supervised event extraction with the fully supervised state of the art.

References: Koby Crammer, Ofer Dekel, Joseph Keshet, Shai ShalevShwartz, and Yoram Singer. (2006). Online passiveaggressive algorithms. Journal of Machine Learning Research, 7:551585. • Hal Daumé (2006). Practical Structured Learning Techniques for Natural Language Processing. PhD Thesis (USC). • Mihai Surdeanu, Sonal Gupta, John Bauer, David McClosky, Angel X. Chang, Valentin I. Spitkovsky, Christopher D. Manning (2011). Stanford's Distantly-Supervised Slot-Filling System. Proceedings of the TAC-KBP Workshop. • Mike Mintz, Steven Bills, Rion Snow, Daniel Jurafsky (2009) Distant supervision for relation extraction without labeled data. ACL/AFNLP: 1003-1011. • Andreas Vlachos, Mark Craven (2011). Search-based structured prediction applied to biomedical event extraction. In Proceedings of the Fifteenth Conference on Computational Natural Language Learning. Association for Computational Linguistics:49-57.