Slot-Filling by Substring Extraction at TAC KBP 2012 (Team Papelo)

Christopher Malon
Dept. of Machine Learning
NEC Laboratories America
malon@nec-labs.com

Bing Bai
Dept. of Machine Learning
NEC Laboratories America
bbai@nec-labs.com

Kazi Saidul Hasan
HLT Research Institute
The University of Texas at Dallas
saidul@hlt.utdallas.edu

Abstract

Motivated by the general question-answering problem, we implement a system for slot-filling based on substring extraction. Rather than restricting to classification of named entity pairs, our system learns to extract arbitrary substrings of text as predicted slot fills. We build two classifiers of this form–one logistic regression classifier using a typical sparse feature vector of syntactic and semantic features (submitted), and a convolutional neural network classifier using dense features learned through a language model (not submitted). In either case, the pipeline is completed with a coreference system and an answer validation module, which rejects answers in improper form for a particular slot. This is our first TAC submission, and our performance was challenged by the distant supervision assumption and the differences between Wikipedia and newswire data.

1 Introduction

Slot-filling is an important NLP task for Knowledge Base Population, but our interest is in using slot-filling as a step towards a question answering system. With this goal in mind, our approach differs slightly from previous work. One crucial distinction is that a question-answering system should be able to answer *any* question about a reading, not just a finite set of pre-determined questions (slots). Answering arbitrary questions, of course, relies on interpreting a natural language question, which we do not address here, but it also relies on extracting arbitrary answers from a reading or a corpus. For example, "why" questions will never be answered by a single named entity.

Like most other teams, we still train separate classifiers for every slot. However, our classifiers learn to extract arbitrary substrings of text, not just named entities or fixed patterns. The classifier considers a representation of the entire parse tree of a sentence at once, in addition to a special tag for each leaf, which tells it whether any named entity is considered coreferent to the query entity. Although some heuristics are used to clean up classifier output, the classifier itself uses very general syntactic features, which don't favor one type of lexical entity over another.

This approach is not mature, and our performance probably suffered from the missing heuristics and our relatively cautious approach to distant supervision, which we discuss later. However, we are encouraged by strong performance if the distant supervision requirement is relaxed, for example if the classifier is tested on Wikipedia data after training it on the same kind of data. Although some kind of NLP pipeline is inevitable in the slot-filling task, we hope that this work contributes towards a method to slot-filling that better extends to general QA.

2 Related work

NEC has developed an approach to NLP problems based upon convolutional neural networks (CNN), known as the Semantic Extraction Neural Network Architecture (SENNA) (Collobert et al., 2011). Although CNN may be more widely known for two-dimensional image classification problems, CNN in one dimension effectively learn features from a slid-

ing window across tokens of text, where each token is represented by a dense feature vector in a learned lookup table that models the words in the language. Such a language model was proposed in (Bengio and Ducharme, 2001). SENNA has achieved state-of-the-art performance in part-of-speech tagging, named entity recognition, and chunking, and competitive performance in semantic role labeling and parsing. Moreover, at classification time, SENNA is 200 times faster at POS tagging than state-of-the-art systems such as (Shen et al., 2007) and a hundred times faster at semantic role labeling (Koomen et al., 2005).

We were anxious to extend SENNA to handle even higher-level semantic tasks. Others have taken up sentiment analysis (Socher et al., 2011b) and paraphrase detection (Socher et al., 2011a) using recursive neural networks, and word sense disambiguation (Bordes et al., 2012) using energy-based models of learned embeddings. Slot-filling could be a prime semantic task for the neural network approach, although its dependence on many other tasks in an NLP pipeline (such as retrieval) makes it a less pure problem. One contribution of this paper is to define a classifier for slot-filling guided by the approach of SENNA. Particularly, it represents the entire parse tree of a sentence as features for labeling each token. Long-range syntactic relationships were at first invisible to convolutional neural networks, which never saw beyond a sliding window of words that tracked the current word for tagging. When SENNA was extended for semantic role labeling, far away tokens contributed to classification via a feature that summarizes all the words in the sentences, taken with their relative distances from the current word. Inspired by this approach, we craft features that summarize an entire parse tree.

One novelty of the CNN approach is that it uses only dense feature vectors to encode linguistic information. Our baseline approach, which is the one we ultimately submitted, uses very similar features in a more traditional sparse setting. As in (Surdeanu et al., 2011), we train logistic regression classifiers based on a sparse, binary feature vector indicating the tokens and patterns that occur, forgoing the language model.

3 Pipeline

Although we rely mainly on a trained classifier to extract slot fills, some other modules are needed to prepare the input and clean up the output of this classifier.

3.1 Document Retrieval

Each query is taken literally, and used in up to three forms as a query for a standard Lucene server (the included Demo server, with standard analyzer and default scorer), which has indexed the documents in the official contest corpus. In the first form, the query is quoted. In the second, each token of the query is taken as a required term. In the third, the query is passed without quoting.

Assuming the first (strictest) query returns at least fifty documents, its results are processed in the sequel. Otherwise, the second, somewhat weaker, form is tried. If at least fifty documents are returned, those are processed. If not, the results of the third query are processed.

Maximally, the top 1000 retrieved documents are processed. If many matches exist, ones with lower TF-IDF matches against the query may be truncated.

3.2 Tokenization and Parsing

We use Stanford's "CoreNLP" system (Lee et al., 2011), (Raghunathan et al., 2010) to perform the lowest level tasks of sentence splitting and tokenization, but not parsing. For higher level syntactic analysis (part-of-speech tagging, named entity recognition, and parsing), we use SENNA.

We run the coreference analyzer included in Stanford CoreNLP to extract mentions and form coreference chains. This analyzer achieved state-of-the-art performance on CoNLL tasks (Raghunathan et al., 2010). We have modified the code to operate on the result of SENNA's faster parser, rather than Stanford's own parser.

Before these steps, newsgroup documents are filtered from the retrieved results, as they are likely to present challenges for later modules. Particularly, ASCII art in a signature could result in sentences with huge numbers of tokens but no meaningful words, resulting in long, fruitless searches for Stanford's coreference module.

3.3 Special Mention

For a given query, it is natural to find sentences that do not mention the query entity exactly and yet provide answers for specific slots. Let us use the following example for illustration.

> "Barack Obama is the 44th and current President of the United States. He is the first African American to hold the office. President Obama is a graduate of Columbia University and Harvard Law School."

Given the query "Barack Obama", we have answers to slots "per:origin" (African American) and "per:schools_attended" (Columbia University and Harvard Law School) in the 2nd and 3rd sentences, respectively. However, the query (i.e., "Barack Obama") does not occur exactly in either of these two sentences. To extract answers from such sentences, it will be useful if the classifier has a clue that both "He" and "President Obama" refer to the query "Barack Obama".

Motivated by this, we run Stanford's co-reference resolver on every supporting document for a given query and extract mention chains from those documents. Ideally, for the text above, "Barack Obama" (1st sentence), "He" (2nd), and "President Obama" (3rd) will be part of the same chain. Once we have all such mention chains from a document, we find a chain that refers to the query entity. Heuristically, a chain refers to the query entity if it has at least one mention that is either an exact match, or a substring, or an acronym of the query string. We then tag each word in the document. We tag the words in the mentions that refer to the query entity using IBES format (*e.g.*, B-ENTITY, I-ENTITY etc). We assign an 'O' tag to each of the remaining words. Finally, we use this tag as a feature for each word.

3.4 Classification

At this point, classifiers are run on the preprocessed documents. The query is no longer needed, as its role is served by the special mention tagging.

One classifier is needed per slot. The classifier gives a prediction for every word, indicating whether the word should be extracted to fill the slot or not. The classifiers use features of the given word, relative to features of the other words in the same sentence, as tagged by the previous modules. Details of the features and the classifers appear in later sections.

3.5 Answer Validation

We apply a post-processing step to validate the slot values predicted by the classifier. Since many of the 42 pre-defined slots are associated with values of specific types (*e.g.*, "per:spouse", "org:top_members_employees" are always names of persons), it makes sense to check if a predicted slot value refers to an entity of the type the slot is associated with. Consequently, we apply a set of rules to validate classifier responses and try to improve our system's precision in this process.

Once the classifier predicts a list of candidate token sequences as potential responses for a particular slot, we validate each of these token sequences in our post-processing module. First, the module lists the entity type(s) associated with the slot. For instance, for "org:alternate_names", the only acceptable answers are organization names. A slot may be associated with several entity types as well. For example, "org:shareholders" can have both person and organization names as potential answers.

Given the list of entity types for a slot, we then check each token sequence (predicted by the classifier) and determine if it matches (or overlaps with) an entity sequence of the desired type(s) in the text. To accomplish this, we use SENNA's named entity tag predictions to mark the spans of person, organization, and location entities in the text. Additionally, we heuristically define token sequence patterns that may have answers for a slot that accepts a date or a number as an answer. Any sequence of nouns with at least one number can potentially be a sequence for a date. Sequence of numbers are marked since they can be answers to "org:number_of_employees_members" or "per:age". For anything else that can not be fit to these types, we mark any sequence of nouns and adjectives (obtained from SENNA's part-of-speech tags) as a valid sequence.

Once we mark these valid sequences for a particular slot in the text, we check if a predicted token sequence exactly fits or overlaps with a marked se-

quence. In case of overlaps, we adjust the boundary of the predicted token sequence so that it exactly fits the sequence in the text. We execute this step for each token sequence predicted for a slot and keep the valid ones (i.e., exact fits or overlaps) only. We then sort these valid token sequences based on their average token score (assigned by the classifier). Note that, this score is not changed even in a case when we fix the boundary of the token sequence (in case of an overlap).

This validation process can help us get rid of invalid sequences, but it certainly can not handle redundant answers, which we like to remove from our list of valid token sequences as well. To do this, we check if two valid token sequences have a sub-string or acronym relationship between them. If so, we remove the one with the lower average token confidence from the pool of valid sequences. Additionally, we also check if two valid token sequences match after we remove punctuation symbols and spaces from both of them and keep the one with higher confidence. This step gets rid of answers like "U.S.A." or "U. S. A." when we already have "USA" as a response. However, we do not apply this process for alternate names (for both persons and organizations) since they often have sub-string and/or acronym relationships among themselves. For such slots, we only keep those valid token sequences each of which has either a sub-string or an acronym relationship with the query string.

Of the remaining token sequences, the one with the highest average score is submitted as the slot-fill. For multiple-valued slots, the scores are thresholded, and up to four results are taken.

4 Classifiers and Features

We consider two approaches to classifying words for slot fills. One is a convolutional neural network based on a dense vector representation of each word, learned through an unsupervised task as a language model. The other is a logistic regression model trained on sparse feature vectors. The logistic regression model was the one officially submitted to the contest.

In either case, the features are crafted to represent tags of the word currently being classified, in addition to information about the rest of sentence. In particular, the relationship to other nodes in the parse tree is encoded.

4.1 Logistic Regression

For the logistic regression classifier, each possible value of the following tags is considered a Boolean feature:

- Single-valued features:
  - Part of speech
  - The word itself (we index 100,000 words), or NIL if the word is not indexed
  - The special mention tag (I, B, E, S, or O), indicating whether the word is part of a mention believed to be in a coreference chain with the query
  - Capitalization feature
- Multiple-valued features:
  - Parse tree node descriptors (see below); one feature is contributed by each node in the parse tree
  - Other words in the sentence, with relative distances from the current word
  - Other words in the sentence, without relative distances

Each feature group has a distinct encoding, so the feature representing the current word is different from the feature representing the current word as part of the bag of words of the sentence.

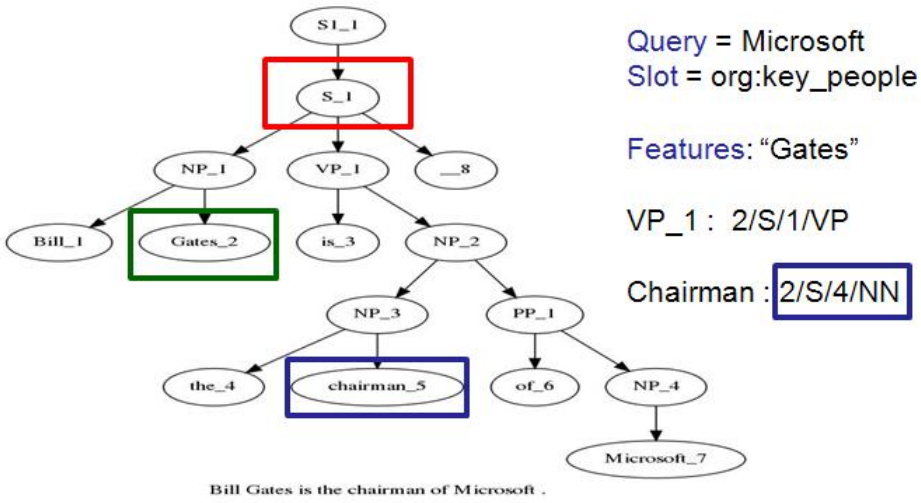
Parse tree nodes are represented by a 4-tuple

$$(\text{Node type}, \text{Pivot type}, \text{Up}, \text{Down})$$

that represents the path in the parse tree from the current word to the node in question. Such a path has a *pivot*, which is the node in the path with the minimal depth from the root of the parse tree. The *node type* and *pivot type* are given by Penn Treebank constituent labels for nonterminal nodes (see (Gildea and Jurafsky, 2002), Table 23), or by the part of speech, for terminal nodes (words). *Up* and *Down* refer to the length of the path from the current node (a terminal node) to the pivot, and the length of the path from the pivot to the node in question.

A single such feature represents each possible 4-tuple, and one feature is contributed by every node in the parse tree. Figure 1 illustrates an example.

Figure 1: Parse Tree Features.



The other words in the sentence contribute two features each: one bag-of-words feature, and one feature consisting of a pair such as (-2, "of"), which means that the word "of" occurred in the same sentence, two words before the word currently being classified.

For the logistic regression classifier, we restricted to the most frequent 10,000 such features. Each word obtains an average of 146 true Boolean features in this manner. Feature vectors are L^2 normalized to unit length, and the classifier is trained with LibLinear (Fan et al., 2008), using L^2-regularized logistic regression and training in the primal space.

4.2 Convolutional Neural Networks

The convolutional neural network is trained with similar information, encoded differently.

Fundamentally, the CNN classifies dense feature vectors from a frame of five words around the current word. Each word's feature vector has two parts—one representing the word itself, and another summarizing other parse tree nodes, from the perspective of the current word. Figure 2 summarizes this architecture.

The word feature vector consists of the same tags, but learned embeddings are associated with each tag. Fifty-dimensional embeddings of the words themselves are borrowed from SENNA, as they have already been trained in a semi-supervised manner to be useful for many syntactic tasks; see (Collobert et al., 2011) for the details of this training.

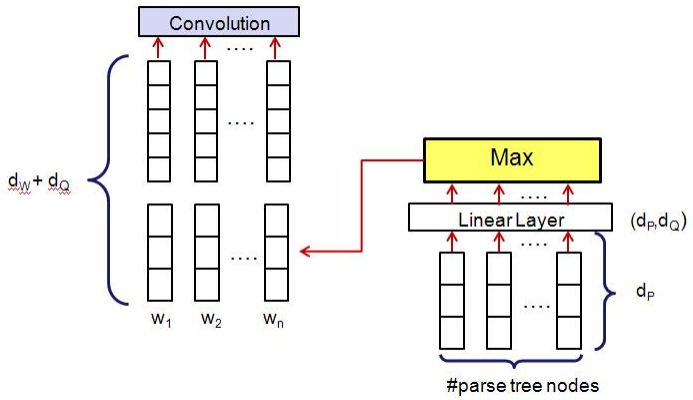
Additionally, each part of speech tag is assigned a random embedding in a six dimensional space. The capitalization feature and special mention tag are each encoded in a single dimension.

Each word vector is concatenated with a tree feature vector, representing the rest of the parse tree from the perspective of the current node. The tree feature vector is computed on the basis of node vectors from each node in the parse tree. The node vector encodes the integers Up and $Down$ from the node descriptor in a single dimension each, and uses a random six–dimensional embedding for the possible values of *Node type* and *Pivot type*.

The node feature vectors are each transformed into a 15-dimensional space through a learned linear layer, at the bottom of our neural network. These results are aggregated by the max function in each dimension, and the output becomes the tree feature vector.

Higher levels of the convolutional neural network act on the concatenation of the word vector and tree vector for each node. First, a convolution over a sliding window of five words transforms these vectors, now 73-dimensional, into a 30-dimensional space. Padding vectors are added to the beginning and end of the sentence, so that these results may be obtained at every position in the sentence, even if the sliding

Figure 2: Convolutional Neural Network for Slot Filling.



window extends beyond the beginning or end.

After squashing the outputs with a sigmoid function, a final linear layer maps the convolution outputs to two dimensions. Let these dimensions have standard basis vectors $\delta_0 = (1, 0)$ and $\delta_1 = (0, 1)$. The classifiers are trained to output the negative log probabilities of the word belonging to the slot fill, by setting the loss against an input vector $\vec{x}$ with ground truth δ_i as:

$$L(\vec{x}, \delta_i) = -\log \frac{e^{-x_i}}{\sum_j e^{-x_j}}. \quad (1)$$

Stochastic gradient descent is used to backpropagate this error through the entire network, down to the word and node type embeddings, which remain fixed.

5 Training and Results

In development, we used three ground truth sources for training and evaluation. Each was distributed to all participants.

1. **TAC 2009 KBP Reference Knowledge Base**. Two-thirds of the articles was used for training, and the remaining one-third was reserved for testing. Infoboxes were used to provide ground truth data, as described below. Sentences from the same article as the infobox were searched for occurrences of the slot fill values.

2. **TAC 2010 Evaluation Slot Filling Queries**. Sentences from the articles containing useful slot fills (as indicated by the LDC judgments) were searched for slot fill values. This data set was used only for training.

3. **TAC 2011 Evaluation Slot Filling Queries**. This data set was used in the same way as the 2010 data, but reserved for testing.

The TAC 2009 KBP Reference Knowledge Base consists of Wikipedia articles, and the ground truth data is extracted from Wikipedia infoboxes, using the mappings distributed by LDC. These mappings take 938 of the Wikipedia infobox slots, such as `actor:birthplace`, and map them to one or more TAC KBP slot names, such as `per:country_of_birth`. In this particular example, the same infobox is mapped to `per:stateorprovince_of_birth` as well, so it is necessary to decide later which part of the Infobox value pertains to which slot.

We normalize infobox values as follows, when necessary:

- Patterns resembling a date are normalized into the form expected by the contest (*e.g.* 2012-04-XX). For these slots, information not matching the patterns is discarded.

- We consult the 2010 CIA World Factbook (Central Intelligence Agency, 2010) for a list of provinces and associated countries. Using this list, we can take text for a Wikipedia infobox such as `actor:birthplace`, identify text matching a known province or country, and use any remaining text as a possible city.

Table 1: Ground truth data for development.

Name	Ground Truth	Text
Wiki	TAC 2009 KBP Reference KB (from Wikipedia)	Same Articles
2010	TAC 2010 Evaluation Slot Filling Queries	LDC Positive Documents
2011	TAC 2011 Evaluation Slot Filling Queries	LDC Positive Documents

We follow a distant supervision heuristic (Mintz et al., 2009) to ground these slot fills with support in the text, but our approach differs slightly from other teams because we do not determine (Entity, Slot, Slot Fill) triples after reading the infoboxes. A normalized infobox value is still a string of text, possibly longer than the appropriate slot fill value. For example, the infobox value mapped to `per:title` for Edward Wilkerson is *bandleader and composer and musician*. Many teams (*e.g.* (Surdeanu et al., 2011)) have relied on named entity recognition to extract relevant triples from an infobox value, but that approach does not work here. Instead, we look for the longest match of the text against the infobox that contains contains useful text. The notion of useful depends on the slot type:

- Number-valued slots should contain a number
- Date-valued slots should contain a date
- Organization, location, or person slots should contain a named entity
- Other slots should contain an adjective or noun

Thus we obtain a set of substrings of articles as positive slot fills for the 42 slots.

For the TAC 2009 KB reference corpus, the same Wikipedia articles are searched to produce the distantly supervised ground truth data. For the TAC 2010 and TAC 2011 query data, each document previously judged by LDC to have a correct slot fill is used as training data for the corresponding entity.

Recall that the classifier ultimately will classify each word of every sentence of the retrieved documents (up to 1000) as a possible slot fill. In training, the task is simplified because we do not process any documents that are completely irrelevant: either the document provides a positive answer to some slot fill (TAC 2010 or TAC 2011), or the document is the Wikipedia article about the entity itself (TAC 2009 Reference).

By far, we can obtain more ground truth labels from Wikipedia than from the previous contest data, which consisted of 80–100 entities each. However, the style of Wikipedia articles is different from most of the newswire articles in the evaluation corpus. Furthermore, not every slot is well-represented by Wikipedia infoboxes; `per:cause_of_death` occurs in an infobox only twice, and only one of those has support in the associated text.

For reference, Table 2 compares the accuracy of the classifiers trained with Wikipedia (2009 reference corpus) alone, the 2010 queries alone, and both together, in predicting slot fills for the 2011 queries. These results represent the probabilities of tokens being correctly classified as positive or negative, not the accuracy of the final slot fill (which may require choosing among several positively-classified substrings from several articles). Ultimately, training on a combination of the TAC 2009 (Wikipedia) and TAC 2010 query data was most effective.

Table 3 compares the best slots (by F1 score) when the logistic regression classifier trained and tested on Wikipedia, to the best slots when it is trained and tested on actual query data (2010 and 2011). The fact that the top five slots are disjoint suggests that the examples captured by the respective distant supervision, techniques or the styles of the documents, are quite different. This is another motivation for combining the training sets.

Like the logistic regression classifier, the convolutional neural network was trained on combined Wikipedia (2009 reference corpus) and 2010 query data, and tested on 2011 query data. Table 4 shows the results on the final phrases submitted, after the classifier outputs have gone through answer validation. As above, only articles known to have a useful slot fill are classified, so the F1 scores are not comparable to actual team submissions, but they are comparable to each other. The convolutional neural network improves upon the F1 score of the logistic

Table 2: Token-level performance on slot fills: Wikipedia versus Query data. Median over 42 slots. Logistic regression classifier.

Training	Testing	F1: Median	F1: 75th Pct	F1: 90th Pct	F1: 95th Pct
Wiki (trn)	Wiki (tst)	.184	.395	.542	.642
Wiki (trn)	2011	0	.031	.108	.185
2010	2011	0	.010	.160	**.316**
Wiki (trn) + 2010	2011	0	**.064**	**.185**	.221

Table 3: Top slots: Wikipedia versus Query data. Token-level results. Logistic regression classifier.

F1	Slot (Wikipedia)	F1	Slot (Query)
.922	per:charges	.667	per:age
.914	per:date_of_birth	.414	org:website
.645	org:alternate_names	.319	org:top_members/employees
.579	per:date_of_death	.250	per:parents
.549	per:alternate_names	.160	per:siblings

regression model by 77%.

The convolutional neural network was not designed in time for our official submission, so our submitted run "papelo1" used the logistic regression model. On the TAC 2012 regular slot filling task, Papelo1 achieved F1=.062, with a recall of .050 and a precision of .081. Suppressing all NIL outputs would have raised the F1 score to .088, by improving recall to .097.

The non-NIL submissions from the other teams were released as data for the Slot Filling Validation task. Of our 77 correct non-NIL slot fills, 27.4% were strings that were not submitted by another team. Thus it may be useful to combine our techniques with those of other systems. If provenance is included, most teams (not runs) have entirely unique sets of correct answers. For this reason, tuning a system's hyperparameters to optimize performance on an answer key derived from past submissions may be dangerous.

6 Conclusion

We introduced two new systems for slot filling based on very general representations of syntactic features, using a traditional logistic regression model and a new convolutional neural network model. Our performance in TAC 2012 was below the median, although the CNN, achieving 77% better performance on the 2011 task used for development, may have surpassed the median F1=.099 if it had been done in time. We showed a lack of generalization between training data sets, indicating that the choice of training corpus and distant supervision technique may be as important as any consideration in designing the classifier.

References

Y. Bengio and R. Ducharme. 2001. A neural probabilistic language model. In *Advances in Neural Information Processing Systems (NIPS)*.

A. Bordes, X. Glorot, J. Weston, and Y. Bengio. 2012. Joint learning of words and meaning representations for open-text semantic parsing. *Journal of Machine Learning Research*.

U. S. Central Intelligence Agency. 2010. *The World Factbook 2010*. U. S. Government Printing Office.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2461–2505.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. 2008. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

D. Gildea and D. Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288.

P. Koomen, V. Punyakanok, D. Roth, and W. Yih. 2005. Generalized inference with multiple semantic role labeling systems (shared task paper). In *Conference on Computational Natural Language Learning (CoNLL)*.

Table 4: Results of the complete system. Systems are trained on the 2009 Wikipedia reference corpus plus 2010 query data.

Classifier	Test Data	Recall	Precision	F1
Logistic Regression	2011 (Useful docs)	.061	.293	.101
CNN	2011 (Useful docs)	.138	.253	.179
Logistic Regression	2012 (Official)	.050	.081	.062

H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky. 2011. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the CoNLL-2011 Shared Task*.

M. Mintz, S. Bills, R. Snow, and D. Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *ACL-IJCNLP*.

K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning. 2010. A multi-pass sieve for coreference resolution. In *EMNLP-2010*.

L. Shen, G. Satta, and A. Joshi. 2007. Guided learning for bidirectional sequence classification. In *Meeting of the Association for Computational Linguistics (ACL)*.

R. Socher, E. Huang, J. Pennington, A. Ng, and C. Manning. 2011a. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems (NIPS)*.

R. Socher, J. Pennington, E. Huang, A. Ng, and C. Manning. 2011b. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

M. Surdeanu, S. Gupta, J. Bauer, D. McClosky, A. Chang, V. Spitkovsky, and C. Manning. 2011. Stanford's distantly-supervised slot-filling system. In *Text Analysis Conference (TAC)*.