Stanford-UBC at TAC-KBP

Eneko Agirre[†], Angel X. Chang[‡], Daniel S. Jurafsky[‡],
Christopher D. Manning[‡], Valentin I. Spitkovsky[‡], Eric Yeh[‡]
[†] IXA NLP Group, University of the Basque Country, Donostia, Basque Country
[‡] Computer Science Department, Stanford University, Stanford, CA, USA

Abstract

This paper describes the joint Stanford-UBC knowledge base population system. We developed several entity linking systems based on frequencies of backlinks, training on contexts of anchors, overlap of context with the text of the entity in Wikipedia, and both heuristic and supervised combinations. Our combined systems performed better than the individual components, which situates our runs better than the median of participants. For slot filling, we implemented a straightforward distant supervision system, trained using snippets of the document collection containing both entity and filler from Wikipedia infoboxes. In this case our results are below the median.

1 Introduction

The TAC 2009 Knowledge Base Population track includes two tasks, entity linking and slot filling. We participated in the first task with a robust system that combines several techniques inspired by Word Sense Disambiguation (WSD) literature [AE06]. We developed several entity linking systems based on frequencies of backlinks, training on contexts of anchors, and overlap of context with the text of the entity in Wikipedia. These techniques were combined using heuristic and supervised methods. We thought that robustness would be an important feature, especially in this first edition of the task, where only a small amount of development data was available. The construction of a dictionary for string-article mappings was a centerpiece of our system, given the subtle issues involved when preprocessing Wikipedia.

For the slot filling task we implemented a straightforward distant supervision system [MBSJ09], trained using snippets of the document collection containing both entity and fillers from Wikipedia infoboxes.

The paper is structured as follows. We first introduce the sections about entity linking. The second section presents the method to construct the dictionary approaches, most of which do not use any context. Section three presents the supervised system. Section four presents the knowledge based system. Section five presents the methods used to combine the rest of the systems. The following section presents the slot filling system. Finally section seven presents the conclusions.

2 Dictionary

Our first task was to construct a rudimentary static mapping from strings to relevant Wikipedia articles. We scored each association using the number of times that the string appeared as the anchor-text of an article's incoming hyperlinks. Note that such dictionaries can be used directly to disambiguate any of the dictionary's keys by simply returning a highest-scoring article.

To construct the dictionaries, we augmented the official KB with all English pages from the March 6th, 2009 Wikipedia dump and all English Wikipedia pages in a recent Google crawl; from each, to the extent possible, we extracted Wikipedia titles, URLs and redirects.

2.1 Remapping

To collect titles and URLs that in fact refer to the same article, we first remapped all entries not in the official KB to URLs using Wikipedia's canonicalization algorithm. We then connected any two URLs that appeared together either in an official Wikipedia redirect or that were redirected at crawltime. For each title and URL, we then extracted a connected component (which could be quite large). From each component, we chose a single representative, with preference given first to members of the official KB,

then to non-redirect pages from the Wikipedia dump, followed by redirect pages, and finally pages that did not appear in the KB or the Wikipedia dump. Within each preference category, we resolved ties lexicographically. Here is an example of a typical cluster merged to be represented by the bolded entry:

Route_102_(Virginia_pre-1933)	State_Route_102_(Virginia_1928)
State_Route_102_(Virginia_1928-1933)	State_Route_102_(Virginia_pre-1933)
State_Route_63_(Virginia_1933)	State_Route_63_(Virginia_1933-1946)
State_Route_63_(Virginia_1940)	State_Route_63_(Virginia_pre-1946)
State_Route_758_(Lee_County,_Virginia)	**Virginia_State_Route_758_(Lee_County)**

All of the entries above are taken to refer to the bolded URL. In what follows, we use the single representative from each cluster.

2.2 Core Dictionary

The core dictionary maps arbitrary strings to sorted lists of (remapped) Wikipedia URLs and associated scores. There are five possible scores, and each string-URL pair must at least have one: booleans $\{\texttt{c}, \texttt{d}, \texttt{t}\}$ and rationals $\{\texttt{w}, \texttt{W}\}$. The presence of booleans in a mapping from <str> to <url> indicates that:

c) *clarification* — either <str> or <str> (...) is the title of some page in the cluster represented by <url>;
d) *disambiguation* — a disambiguation page titled <str> links to some page in the cluster represented by <url>;
t) *title* — a page titled <str> is in the cluster represented by <url>.

The rational w:x/y, $x \neq 0 < y$ indicates that of the y inter-Wikipedia links with anchor-text <str>, x pointed to a page represented by <url>; similarly, W:u/v, $u \neq 0 < v$, indicates that of the v external links into Wikipedia with anchor-text <str>, u pointed to a page represented by <url>.

The list of URLs for each string is sorted in decreasing order by $(x + u)/(y + v)$, taking $x = 0$ when w is absent, $u = 0$ when W is absent, and $(x + u)/(y + v) \equiv 0$ when both are absent, with ties resolved lexicographically.

We refer to this dictionary as exact (EXCT), as it contains precisely the strings found using the methods outlined above. For example, for the string Hank Williams, it returns the following eight associations:

0.997642	Hank_Williams	W:936/938 c d t w:756/758
0.00117925	Your_Cheatin'_Heart	W:2/938
0.000589623	Hank_Williams_(Clickradio_CEO)	c w:1/758
0.000589623	Hank_Williams_(basketball)	c w:1/758
0	Hank_Williams,_Jr.	d
0	Hank_Williams_(disambiguation)	c
0	Hank_Williams_First_Nation	d
0	Hank_Williams_III	d

2.3 Fuzzy Flavors

In addition to exact lookups in the core dictionary, we entertain two less strict views. In both cases, an incoming string now maps to a *set* of keys, with their lists of scored URLs merged in the intuitive way: a boolean score is present in the merged score if it is present in any of the original scores for the URL; rational scores are merged using the formula $a/b + c/d \to (a + c)/(b + d)$.

LNRM: We form the lower-cased normalized version $l(s)$ of a string s by canonicalizing its UTF-8 characters, eliminating diacritics, lower-casing the UTF-8 and throwing out all ASCII-range characters that are not alpha-numeric. If the resulting string $l(s)$ is empty, then s maps to no keys; otherwise, it maps to all keys k such that $l(s) = l(k)$, with the exception of $k = s$, to exclude the key already covered by EXCT.

Thus, $l(\texttt{Hank Williams}) = \texttt{hankwilliams} = l(\texttt{HANK WILLIAMS}) = l(\texttt{hank williams})$, etc. The combined contribution of all but the original string with this signature yields five suggestions (most of which overlap with EXCT's):

0.952381	Hank_Williams	W:20/21 c t
0.047619	I'm_So_Lonesome_I_Could_Cry	W:1/21
0	Hank_Williams_(Clickradio_CEO)	c
0	Hank_Williams_(basketball)	c
0	Hank_Williams_(disambiguation)	c

FUZZ: We define a metric $d(s, s')$ to be the byte-level Levenshtein edit-distance between strings s and s'. If $l(s)$ is empty, then s maps to no keys once again; otherwise, it maps to all keys k, $l(k)$ not empty, that minimize $d(l(s), l(k)) > 0$. Note that this again excludes $k = s$, as well as any keys covered by LNRM.

For Hank Williams, some strings whose signature is exactly one byte away include Tank Williams, Hanks Williams, hankwilliam, and so on. Together, they recommend just three articles (two of which were already covered by EXCT / LNRM):

0.631579	Tank_Williams	c t w:12/12
0.315789	Hank_Williams	W:6/7
0.0526316	Your_Cheatin'_Heart	W:1/7

2.4 Other Flavors

In addition to the core dictionary, we experimented with a number of alternatives, two of which proved useful enough to be included in the final submissions.

GOOG: This "dictionary" queries the Google search engine using the directive `site:en.wikipedia.org` and the string `<str>`, keeping only the URLs that start with `http://en.wikipedia.org/wiki/` and using inverse ranks for scores.

CAPS: This context-sensitive "dictionary" queries a repository of Wikipedia pages using Lucene[1]. In addition to the string `<str>`, it includes all non-trivially capitalized words in the document containing `<str>` for context via the construct

`<str>` AND (`<str>` OR `<context>`),

which boosts the articles containing *both* `<str>` and `<context>`, and using the scores returned by Lucene.

3 Supervised Disambiguation

In addition to plain lookups, we applied machine learning techniques to do supervised disambiguation of entities. For each target string, we trained a multiclass classifier to distinguish the more relevant of the possible articles. Using the text surrounding the target string in the document, this classifier scored each possible entity among the mappings provided by the dictionary.

Inter-Wikipedia links and the surrounding snippets of text comprised our training data. When one article linked to another, we assumed that the anchor text was synonymous with the linked Wikipedia entity (as collapsed by the remapper, described above). Since the dictionary could return entities having no incoming Wikipedia links for the target string, we restricted the list of possible entities to those with actual links for the exact (raw, unnormalized) target string. Our snippets were spans of text consisting of 100 tokens to the left and 100 to the right of a link. These spans were extracted from the October 8th, 2008 English Wikipedia dump.

Depending on the number of Wikipedia spans that contained the target string as linked text, we sometimes supplemented our training data with spans that linked to a possibly related entity:

- Spans with exact matches (same anchor-text as the target string) if there were more than 1,000 such spans. For instance, since there were over 3,000 Wikipedia spans linking from the string `Pittsburgh`, we simply used just those Wikipedia spans for training.
- Spans with all possible entities for the target string if there were fewer than 1,000 spans with exact string matches. For example, for `ASG`, there were only 13 Wikipedia spans linking from the exact string `ASG`, so we also used spans linking to `Abu Sayyaf`, `ASG (band)`, as well as other entities associated with the string `ASG` as training data.

From the spans, we extracted the following features for training:

- the anchor text;
- the lemmas in the span;
- lemma for noun/verb/adjective in a 4 token window around the anchor text;
- lemma and word for noun/verb/adjective before and after the anchor text;
- word/lemma/POS bigram and trigrams around the anchor text.

For example, the Wikipedia article for `SuperFerry` has the following span of text that links the string `Abu Sayyaf` to the URL `Abu_Sayyaf`:

```
On February 27 , 2004 , SuperFerry 14 was bombed by the Abu Sayyaf terrorists killing 116 people .
It was considered as the worst terrorist attack in the Philippines .
```

We would use this span (actually a longer version, covering 100 tokens to the left and also to the right of `Abu Sayyaf`) as a training instance of `ASG` mapping to `Abu_Sayyaf` with the following features:

[1] `http://lucene.apache.org/`

anchor text		unigram Abu_Sayyaf
lemmas in the span		win_cont_lem_context terrorist win_cont_lem_context kill ...
lemma for noun/verb/adjective in a 4 token window around the anchor text		win_cont_lem_4w be win_cont_lem_4w bomb win_cont_lem_4w kill win_cont_lem_4w people win_cont_lem_4w terrorist
lemma and word for noun/verb/adjective before the anchor text	noun (lemma)	prev_N_lem SuperFerry
	noun (word)	prev_N_wf SuperFerry
	verb (lemma)	prev_V_lem bomb
	verb (word)	prev_V_wf bombed
lemma and word for noun/verb/adjective after the anchor text	adjective (lemma)	post_J_lem bad
	adjective (word)	post_J_wf worst
	noun (lemma)	post_N_lem terrorist
	noun (word)	post_N_wf terrorists
	verb (lemma)	post_V_lem kill
	verb (word)	post_V_wf killing
bigrams around the anchor text	lemma before	big_lem_func_+1 the Abu_Sayyaf
	lemma after	big_lem_cont_-1 Abu_Sayyaf terrorist
	POS before	big_pos_+1 DT J
	POS after	big_pos_-1 J N2
	word before	big_wf_func_+1 the Abu_Sayyaf
	word after	big_wf_cont_-1 Abu_Sayyaf terrorist
trigrams around the anchor text	lemma before	trig_lem_func_+1 by the Abu_Sayyaf
	lemma around	trig_lem_cont_0 the Abu_Sayyaf terrorist
	lemma after	trig_lem_cont_-1 Abu_Sayyaf terrorist kill
	POS before	trig_pos_+1 P-ACP DT J
	POS around	trig_pos_-1 J N2 VVG
	POS after	trig_pos_0 DT J N2
	word before	trig_wf_func_+1 by the Abu_Sayyaf
	word around	trig_wf_cont_0 the Abu_Sayyaf terrorists
	word after	trig_wf_cont_-1 Abu_Sayyaf terrorists killing

We trained SVMs using these binary features with a linear kernel, producing one classifier for each target string that had more than one possible entity. Below is the break-down of target strings into ambiguous and non-ambiguous classes. For the evaluation corpus, we trained supervised classifiers for 213 out of the 514 unique strings.

Corpus	*Target Strings*	*No Entities*	*One Entity*	*Ambiguous*
Sample v2.0	67	29	22	16
DevData	73	3	36	34
Evaluation	514	103	198	213

4 Knowledge Based Disambiguation

Our third method for disambiguating entity mentions employed Wikipedia as a knowledge rich resource for obtaining candidate entities — a natural fit, since the target entities were themselves drawn from it. To this end, we simply treated Wikipedia as a standard IR repository, with each article a document. The mention and surrounding context were then treated as a document query against this repository, with cosine similarities between its and the articles' TF-IDF vectors used for scoring.

The intent here is similar to the use of corpus based WSD techniques such as the Lesk algorithm [Les86]: some form of overlap with the gloss for a given sense is used to identify the likeliest match. For example, observing a mention of Lee and Jefferson Davis in the context, the mention would be more indicative of Robert E. Lee than of Bruce Lee.

4.1 Resource Preparation

We used the October 8th, 2008 English Wikipedia dump stripped of non-content pages, such as discussions and redirects. To further reduce noise, we kept only the articles with a content area of at least 100 tokens. We then zoomed in on the primary text intended for the readers by removing elements contained in MediaWiki markup (e.g. infoboxes) and using the display values of links, instead of the canonical link target names, when available.

We then indexed the resulting text using Lucene and queried its index to identify matching articles.

We also explored generating a Lucene index over just the content in the provided Knowledge Base, as it was also derived from Wikipedia. However, its performance on the development set was poor compared to using the entire Wikipedia dump (subject to the described filtering), and we did not pursue this avenue further.

4.2 Querying and Scoring

With the resource in place, for a given mention and document pair, we employed several methods to formulate a query against it. The following variations depend on the amount of surrounding context that could be pulled in:

- The mention alone, without the surrounding context.
- The last occurrence of the mention in the text, with a span of 25 tokens to the left and to the right of the mention.
- The concatenation of all matching mentions, and their 25 token spans, in the text.
- A window of 1,000 tokens around the last mention in the text.

Preliminary results on a development set showed that using the concatenation of all occurrences of the mention and their 25 word contexts performed best, and this was used for test set queries.

After issuing the query, we filtered the returned list of Wikipedia articles and similarity scores, retaining only those covered by the dictionary. The rationale here was to match the same set of articles under consideration by the other methods.

5 Combination

After constructing the dictionary, the supervised and the knowledge-based modules, we combined them into the final system in three different ways, each corresponding to a submitted run:

1. *cascade of dictionaries* (`run1`) — an overall context-independent score for each entity, defaulting to LNRM (lower-cased normalized) in case of an EXCT (exact match) miss;
2. *heuristic voting* (`run2`) — a hand-tuned linear combination of GOOG (Google ranks), the cascaded dictionary, plus the knowledge-based and supervised systems;
3. *optimized mix* (`run3`) — a linear combination of positive weights applied to the scores from many components, fitted to the `Sample v2.0` and `DevData` corpora.

For each of the combined methods, we have a system that, for each target mention, provides a ranked list of potential entities (i.e., Wikipedia article titles) corresponding to the target string. If the top-rated entity is in the KB, the system returns its ID; otherwise, or if no entities were suggested, it returns NIL.

We now describe these different combination methods in more detail.

5.1 Cascade of Dictionaries

For `run1`, we combined the dictionaries in a simple cascade to get overall, context-independent scores. For each target mention, we provided a ranked list of potential matches by using the following heuristic:

- If the EXCT dictionary has a match, use scores from the EXCT dictionary;
- else if the LRNM dictionary has a match, use scores from the LNRM dictionary;
- else if the FUZZ dictionary has a match, use scores from the FUZZ dictionary;
- otherwise, do not suggest anything.

Initial evaluation indicated that the FUZZ dictionary was very noisy, so for `run1`, we submitted results using the cascade of dictionaries consisting of EXCT and LNRM only.

5.2 Heuristic Voting

For `run2`, we combined context-free and context-sensitive methods using a voting heuristic based on the following systems:

- cascaded dictionary (EXCT → LNRM → FUZZ);
- Google ranks;
- knowledge-based system;
- supervised system (for ambiguous exact matches only).

Here, we began with the dictionaries (EXCT/LNRM/FUZZ), together with rankings from Google (GOOG) to get a list of possible entities for the target string. From each component, we took its top 1,000 entities and filtered out what wasn't on the

list. Next, we calculated the combined score for an entity by taking the sum of the inverse ranks from the different components, yielding a merged ranked list of potential matches.

5.3 Optimized Mix

For `run3`, we used a positive linear combination of the scores provided by various components, combining individual scores $\{c, d, t, W, w\}$ from the EXCT dictionary, the overall scores from the LNRM, FUZZ, GOOG and CAPS dictionaries, the scores from the knowledge-based system and the inverse ranks from the supervised system.

Taking the highest scoring guess as the answer, we used conjugate gradient to find the weights that optimized the fraction of correct answers and exponentiated parameters to keep the final weights positive, once again training on the `Sample v2.0` and `DevData` corpora.

5.4 Summary of Results

The tables below show results for `Sample v2.0`, `DevData` and the final test corpus. Note that the cascade of dictionaries (`run1`) performs surprisingly well, considering that it ignores context. For `DevData`, there is little ambiguity and the dictionaries achieve 96.6% accuracy. Adding context yields only small improvements, with the heuristic combination performing better than the linear fit. The small gain attained by the latter may be due to the limited amount of training data or to differences between training and test sets.

Note that the linear combination results for the `Sample v2.0` and `DevData` are over-fitted. We include them only for completeness.

	Micro			Macro		
	119 queries	75 KB	44 NIL	31 entities	18 KB	13 NIL
run1	0.8655	0.8400	0.9091	0.8349	0.7712	0.9231
run2	0.8908	0.9333	0.8182	0.8781	0.9196	0.8205
run3	0.8403	0.8933	0.7500	0.8015	0.8224	0.7724

Table 1: Entity Linking Results for TAC 2009 KBP Sample Corpus v2.0.

	Micro			Macro		
	297 queries	210 KB	87 NIL	42 entities	41 KB	1 NIL
run1	0.9495	0.9571	0.9310	0.9607	0.9614	0.9310
run2	0.9663	0.9857	0.9195	0.9822	0.9837	0.9195
run3	0.9663	0.9857	0.9195	0.9644	0.9654	0.9195

Table 2: Entity Linking Results for TAC 2009 KBP DevData.

	Micro			Macro		
	3904 queries	1675 KB	2229 NIL	560 entities	182 KB	378 NIL
run1	0.7485	0.6949	0.7887	0.6851	0.5535	0.7485
run2	0.7884	0.7588	0.8107	0.7003	0.6081	0.7446
run3	0.7510	0.7325	0.7649	0.6861	0.5792	0.7375

Table 3: Entity Linking Results for TAC 2009 KBP Evaluation Corpus.

6 Slot Filling

We tried a straightforward strategy for slot filling, designed around distant supervision [MBSJ09]. To train the relation-extraction system, we first produced the training examples for slots, as follows:

- Obtain entity-slot-filler triples from infoboxes;
- Map infoboxes into required KB slots;

- Assign a named-entity (NE) type or a closed list to each slot;
- Obtain text fragments from documents where the entity and filler occur close to each other.

We then trained classifiers and applied them to obtain new entity-slot-filler triples, as follows:

- Train a classifier for each slot using the text fragments gathered (see above);
- Obtain mentions to the target entities from the document collection which included a named entity of the required type or a value from closed list;
- Run each of the classifiers on those mentions;
- For each entity-slot-filler triple in positively classified mentions, record number of instances and average weight;
- For each entity-slot, return the top-scoring filler (if slot is single-valued) or the top five fillers (if multi-valued) pair.

Finally, we applied the cascade of dictionaries to disambiguate the filler and return a knowledge-base entity ID or NIL.

The development of the system did not involve manual curation of data, except assigning named entity classes (e.g., date, person) or closed lists of fillers (e.g., religions, countries) to each slot.

Below, we first present the details of how we prepared the slot information, then how we extracted the textual fragments (spans) of entity occurrences, followed by the method to train the classifiers. The application of the classifier to produce the slot filling results is explained next. Finally, we summarize the obtained results.

6.1 Slot Preparation

To prepare training data for the slot classifiers, we extracted entity-slot-filler triples from Wikipedia infoboxes using the provided mapping.

As part of slot preparation, we categorized different slots based on the expected NE type: ORG, PER, LOC, DATE, and NUMBER. The NE type is used to help assign ambiguous infobox values to the appropriate slot, as well as to identify potential fillers for a text fragment for a slot. Some slots, such as gpe:currency did not match an NE type we had available, so we used a closed list to help identify potential fillers; for org:website, we used regular expressions; we did nothing for per:title.

NE (ORG)	gpe:subsidiary_orgs, org:alternate_names, org:founded_by, org:member_of, org:members, org:parents, org:shareholders, org:subsidiaries, per:employee_of, per:member_of, per:schools_attended
NE (PER)	gpe:top_employees, org:founded_by, org:shareholders, org:top_members/employees, per:alternate_names, per:children, per:other_family, per:parents, per:siblings, per:spouse
NE (LOC)	gpe:capital, org:headquarters, per:place_of_birth, per:place_of_death
NE (DATE)	gpe:established, org:dissolved, org:founded, per:date_of_birth, per:date_of_death
NE (NUMBER)	gpe:population, org:number_of_employees/members, per:age
Closed List	gpe:currency, gpe:political_parties, org:political/religious_affliation, per:cause_of_death, per:charges, per:origin, per:religion
RegExp	org:website
NIL	per:title

Table 4: Mapping of slot to NE type or closed list.

Due to the ambiguity and noisiness of the infobox to slot mapping, we processed the infobox values for the entity-slot-filler triple as follows:

- Run a named-entity recognition and classification system [FGM05] on the entity itself to determine if the entity is ORG/PER/LOC. We map the NE type LOC to correspond to the TAC KBP entity type GPE. Because the slots are specific for the three entity types, we can safely ignore any entity that is not ORG/PER/LOC. Besides, note that some entities in the knowledge base have been tagged as UNK by the organizers (instead of ORG/PER/GPE). We also run NER on this to determine the entity type.
- Run NER on infobox fillers to extract fillers for ambiguous slots. The mapping from the Wikipedia infobox to the TAC KBP slots can be ambiguous. For instance, the Wikipedia infobox "born" can map to both date_of_birth and place_of_birth.

  Carrie Underwood
  Born March 10, 1983 (1983-03-10) (age26) Muskogee, Oklahoma, USA

  In these cases, the NER system will determine that "March 10, 1983" is a DATE and "Muskogee, Oklahoma, USA" is a LOC. We then map "March 10, 1983" to the date_of_birth slot and "Muskogee, Oklahoma, USA" to the place_of_birth slot.

After obtaining the entity-slot-filler triples, we extract spans from the document base for training and development.

6.2 Span Extraction

Development spans were drawn from the TAC KBP Entity Linking Sample Corpus, while training spans were drawn from the entire document base. We indexed the document base using Lucene, and used its `SpanQuery` and `SpanNearQuery` functionality to extract the spans.

For training spans, we used the known entity and filler pairs, and looked for occurrences of these in the document base. Exact string match is used for both the entities and fillers. We looked for spans with up to 10 tokens between the entity and filler, and five words to surrounding the entity and filler. The spans are of the form:

```
5w entity 0-10w filler 5w
5w filler 0-10w entity 5w
```

where Nw corresponds to N words/tokens; for the middle span, this ranged from zero to ten.

Note that because we look for exact matches for the entity and filler, we miss spans that contain variations of the entity or filler strings e.g.

Carrie Marie Underwood was born to Stephen and Carole Underwood in Muskogee, Oklahoma.

For target entities for slot filling, we extracted spans that matched the string of the entity exactly. These spans are of the form:

```
30w entity 30w
```

Similarly, we miss spans that use different names for the target entity.

6.3 Training the Classifiers

For each slot, we trained a binary classifier that takes a text fragment with the entity and potential filler and decides whether or not the potential filler is an actual filler for the slot. We used a simple regularized logistic regression model trained on the entity-slot-filler spans extracted from the document base.

For positive examples, we used spans containing the known entity and filler pairs based on slots derived from Wikipedia infoboxes. To avoid misleading infoboxes, we only used spans that had an entity type matching the entity type of the slot.

We had two types of negative examples:

- Spans from other slots matching the entity type (up to twice the number of positive examples if available);
- Generated negative spans. We ran NER and the closed lists of terms over the training spans from Wikipedia to get potential negative examples. We then filtered out the spans with an entity-filler that matched the correct entity-filler for the slot. The remaining instances were used as negative examples.

To extract features from the spans, we annotated and transformed the span text as follows:

- Replace the specific entity and filler with placeholders (`LFILLER`, `RENTITY` if filler is to left of entity; `LENTITY`, `RFILLER` if filler is to the right of entity);
- Append tags identifying whether the words come from Left Window (`L`), Between Window (`B`), or Right Window (`R`);
- Append direction (filler to right (`R`) or left of entity (`L`)).

We then took the annotated spans and extracted n-grams to use as features.

Slot:	`per:parents`
Entity:	Shirley Phelps-Roper
Filler:	Fred Phelps
Span:	One of the defendants , Shirley Phelps-Roper , is an attorney and church member whose father , Fred Phelps , helped establish Westboro in 1955
Annotated Span:	RLOne RLof RLthe RLdefendants , LENTITY , RBis RBan RBattorney RBand RBchurch RBmember RBwhose RBfather , RFILLER , RRhelped RRestablish RRWestboro RRin RR1955
Features:	n-grams (for $n = 1, \ldots, 4$) of the annotated span.

These allow us to learn features such as:

`(4-SW#-RBdaughter-RBof,+)`	0.5091
`(4-SW#-RBfather,+)`	0.4469
`(4-SW#-RBdaughter,+)`	0.4233
`(4-SW#-RBfather-RFILLER,+)`	0.4100
`(4-SW#-LBdaughter,+)`	0.3679

However, they do not allow us to capture the longer distance relationships, as shown in the example. In addition, some of the other features learned can be very noisy. We left these issues for future work.

Once the classifiers have been trained, we used them to evaluate and determine the most likely fillers for the target entities. Using the spans extracted from the document base for each entity, we identify potential fillers using NER, a regular expression, or a closed list of strings (see Table 4). After identifying potential fillers, we eliminated any spans that did not match the training data. More specifically, we kept only those spans matching the following format:

```
5w entity 0-10w filler 5w
5w filler 0-10w entity 5w
```

For each potential filler, we use the trained binary classifier for the slot to determine whether or not the filler is correct. For fillers that are identified positively, we aggregate over all spans with that filler to get the average score for the filler. For each entity-slot, this yields a ranked list of potential fillers. We select the top-scoring filler for single-valued slots and top five for list-valued slots. As a last step, we check existing slot values in the knowledge base and remove any fillers that are already present. For single-valued slots, we assume the knowledge base is correct and return NIL if the slot is already populated.

6.4 Summary of Results

Due to the limited time remaining for this task, we were not able to tune our system. For `run1`, we submitted a basic version that was too aggressive in guessing slot fillers, resulting in low scores for both NIL and non-NIL scores; `run2` unexpectedly produced the same results as `run1`. For `run3`, we used more negative examples (as described earlier). This gave a small boost on accuracy of NILs for list-valued slots.

	`run1`, `run2`	`run3`
SF-value score (all slots)	0.355	0.373
single-valued slots		
Accuracy (all 255 slots)	0.392	0.384
Accuracy (39 non-NIL slots)	0.179	0.179
Accuracy (216 NIL slots)	0.431	0.421
list-valued slots		
Average F-score (all 499 slots)	0.317	0.363
Average F-score (129 non-NIL slots)	0.141	0.148
Average F-score (370 NIL slots)	0.378	0.438

Table 5: TAC 2009 KBP Slot Filling Results.

7 Conclusions

We have described the joint Stanford-UBC knowledge base population system. We developed several entity linking systems based on frequencies of backlinks, training on contexts of anchors, overlap of context with the text of the entity in Wikipedia, and both heuristic and supervised combinations. Our combined systems performed better than the individual components, which situates our runs better than the median of participants. For slot filling, we implemented a straightforward distant supervision system, trained using snippets of the document collection containing both entity and filler from Wikipedia infoboxes. In this case our results are below the median.

The construction of a good dictionary which would encompass all possible strings that could be used to refer to any entity and article in Wikipedia is one of the cornerstones of our entity linking system. In addition to list all possible string-article pairs, we also included counts for anchors which attested the string-article pair. These counts already provide a very good performance, with 75% accuracy, which is surprising given the fact that it does not make use of the context. A good dictionary is also a requirement for effective supervised and knowledge based systems. All in all, the use of context by the supervised and knowledge based systems improves the performance to 79%.

Our entity linking system is able to disambiguate any string in Wikipedia. In fact, we did disambiguate the given strings with regards to all relevant articles in Wikipedia (in contrast to focusing on just the entities in the given Knowledge Base). We think this is a key feature in order to have good performance with NIL.

We also observed a similar effect when comparing the entity linking performance of knowledge based Lucene resources generated from Wikipedia as a whole versus one generated just from the provided Knowledge Base. We believe that having this

extra information can help reduce ambiguity about when a mention should be considered NIL, by giving a more concrete notion of what the space of competing concepts looks like, instead of just focusing on the Knowledge Base alone.

We developed our systems from scratch in a short time, and given the fact that it was our first attempt on these tasks we are satisfied with the results. We plan to continue our work on both tasks, finetuning the algorithms and including more sophisticated techniques.

Acknowledgements

This work was carried out while Eneko Agirre was visiting Stanford with a grant from the Ministry of Science. We thank Oier Lopez de Lacalle and David Martinez for the script to extract features.

References

[AE06] Eneko Agirre and Philip Edmonds, editors. *Word Sense Disambiguation: Algorithms and Applications*, volume 33 of *Text, Speech and Language Technology*. Springer, July 2006.

[FGM05] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 363–370, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[Les86] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *SIGDOC '86: Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26, New York, NY, USA, 1986. ACM.

[MBSJ09] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of ACL-IJCNLP 2009*, 2009.