

This item is likely protected under Title 17 of the U.S. Copyright Law. Unless on a Creative Commons license, for uses protected by Copyright Law, contact the copyright holder or the author.

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

Please provide feedback

Please support the ScholarWorks@UMBC repository by emailing scholarworks-group@umbc.edu and telling us what having access to this work means to you and why it's important to you. Thank you.

HLTCOE Approaches to Knowledge Base Population at TAC 2009

Paul McNamee, Mark Dredze, Adam Gerber, Nikesh Garera,
Tim Finin, James Mayfield, Christine Piatko, Delip Rao, David Yarowsky, Markus Dreyer
Human Language Technology Center of Excellence
The Johns Hopkins University
Baltimore, MD 21218, USA
paul.mcnamee@jhuapl.edu

Abstract

The HLTCOE participated in the entity linking and slot filling tasks at TAC 2009. A machine learning-based approach to entity linking, operating over a wide range of feature types, yielded good performance on the entity linking task. Slot-filling based on sentence selection, application of weak patterns and exploitation of redundancy was ineffective in the slot filling task.

1 Introduction

The Human Language Technology Center of Excellence (HLTCOE) participated in both the Entity Linking and Slot Filling tasks of the Knowledge Base Population track at TAC 2009. This paper describes the systems we built for each of these tasks, lists our results and comments on some of the strengths and weaknesses of our approaches.

2 Entity Linking

We cast entity linking as a supervised machine learning problem. Each entity linking query comprises a query document and a mention string found within that document. We represent each query by a D dimensional vector \mathbf{x} , where $\mathbf{x} \in \mathbb{R}^D$. The entity linking task requires us to select, for each example, a single KB node y , where $y \in \mathbb{Y}$, a set of possible KB nodes for this query. Note that \mathbb{Y} may contain all nodes in the KB, although in practice we filter this set considerably before learning. We assume at most one correct KB node for each example, so the i th entity linking example is given by the pair $\{\mathbf{x}_i, y_i\}$.

To evaluate each candidate KB node in \mathbb{Y} we create feature functions dependent on both the example \mathbf{x} and the KB node y . The feature function $f_j(\mathbf{x}, y)$ returns the value for the j th feature dependent on the query and a candidate KB node. We define the feature functions used by our system in the next section.

We take a maximum margin approach to learning. We desire that the correct KB node y for an example \mathbf{x} receives a higher score than all other possible KB nodes $\hat{y} \in \mathbb{Y}, \hat{y} \neq y$. This learning constraint is equivalent to the ranking SVM algorithm of Joachims (Joachims, 2002). For the ranking SVM, we define an ordered pair constraint for each of the incorrect KB nodes \hat{y} and the correct node y , such that $\text{score}(y) \geq \text{score}(\hat{y}) + \gamma$, where γ is the margin. We then use the learning package SVM^{rank} to solve the optimization problem.¹ For learning we use a linear kernel, set the slack parameter C as 0.01 times the number of training examples, and take the loss function as the total number of swapped pairs summed over all training queries.

So far we have assumed that each example has a correct KB entry; however, many queries will not match any KB entries (*i.e.*, the correct system response for the query is NIL). One approach to predicting NIL answers is to set a threshold, returning NIL if the top answer's score is below the threshold. This approach has several disadvantages. First, we need to set this threshold manually. Second, the ranking SVM constraints are relative only within a single example; the magnitude of the scores cannot be compared across examples. Third, a threshold sets a uniform standard across all examples, whereas

¹http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

in practice we may have reasons to favor a `NIL` prediction in a given example.

We integrate `NIL` prediction into the learning process by augmenting \mathbb{Y} to include the answer `NIL`. We then add a feature function that returns 1 only if $y = \text{NIL}$. This enables the algorithm to learn a threshold for `NIL` answers by tuning the weight given to the `NIL` feature function. Additionally, we can add arbitrary features for the `NIL` label.

2.1 Features

Since SVMs robustly learn with many features, we added a large number of features to our system (about 200 atomic features). The features were computed for each candidate query/KB pair that passed an initial triage phase. Since we used a linear kernel, we explicitly combined certain features (*e.g.*, acronym-match AND known-alias) to model correlations. This included combining each feature with the predicted type of the entity, allowing the algorithm to learn prediction functions specific to each entity type. With feature combinations, the total number of features used by the learner was 26,569.

We organized these features into 55 classes for the purpose of internal analysis, which are summarized in Table 1. Due to space restrictions we provide high level feature descriptions in the following sections.

2.1.1 Triage Features

While feature extraction and classification are relatively fast, considering all nodes in the KB for every query is prohibitively expensive. Instead, we opted for a two-phase process. We first compute a limited set of triage features and select candidate KB nodes based on these features alone. Our second phase then applies the supervised learning approach described above, extracting the full set of features for the smaller set of KB candidate nodes, then ranking them using the learned ranking function. In our experiments, these triage features served to identify a candidate list for each query that was three to four orders of magnitude smaller than the full set of KB nodes.

The goal for our triage features was to achieve high recall (*i.e.*, to miss few correct entries) and to generate reasonably sized candidate sets. Nearly all of these features were based simply on the strings of the query name q and the KB node name k .

On our development dataset, described below, we attained a recall of 98.8% in this candidate identification phase. Many of our misses had to do with inexact acronyms; for example, we missed: ABC (Arab Banking; ‘Corporation’ is missing), ASG (Abu Sayyaf; ‘Group’ is missing), and PCF (French Communist Party; in French the order of the prenominal adjectives is reversed). We also missed International Police (Interpol) and Becks (David Beckham; Mr. Beckham and his wife are collectively referred to as ‘Posh and Becks’).

String Equality. If the query name q and KB node name k are identical, this is a strong indication that perhaps q might be linked to k . Of course this may not be the case, particularly because names are ambiguous.

The KB is derived from Wikipedia where article titles are distinct; as a consequence, node name strings in the KB are unique. Similar or identical names that refer to distinct entities are often qualified with parenthetical expressions or short clauses. As an example, E0001618 “London, Kentucky” is distinguished from E0026729 “London, Ontario”, E0104817 “London, Arkansas”, E0104817 “London (novel)”, and E0397283 “London”.

Other string equality features were used in the candidate ranking phase, such as whether names are equivalent after some transformation. For example, “Baltimore” and “Baltimore City” are exact matches after removing a common GPE word like city; “University of Vermont” and “University of VT” match if VT is expanded to Vermont.

Approximate String Matching. Name strings occurring in article text may resemble KB node names but may not be exact matches. Thresholds on Dice coefficients comparing sets of letters or of letter bigrams were used, as were nearly exact matches using left or right Hamming distance. These were useful for detecting minor spelling variations or mistakes.

KB nodes were also considered if q was wholly contained in k , or vice-versa. This was useful for handling ellipsis (*e.g.*, “United States Department of Agriculture” vs. “Department of Agriculture”).

In the candidate ranking phase, additional fuzzy matches were considered. The ratio of the recursive longest common subsequence (Christen, 2006)

<i>Type</i>	<i>Features</i>	<i>Details</i>
acro	4	Acronym match.
alias	3	Names match using alias lists.
bbnalias	1	An alias match using a list provided by BBN Technologies.
bs	6	Basic string similarity metrics. Includes Hamming distance, Dice scores using skip bigrams or regular trigrams, ratio of longest common subsequence to total string length, and string equality.
bs2	5	Additional string similarity metrics, including equivalence with acronym expansion or string equality with organization and geographic terms removed (<i>e.g.</i> , Inc., Corp., Province).
dice	2	Dice coefficient for words in the KB facts or KB text and the query document text.
facts	3	Presence of KB facts words in near query name mention or in query document.
filt1	1	Triage feature. Exact string match.
filt2	1	Triage feature. Query or KB name contained in the other (<i>e.g.</i> , <i>Nationwide</i> and <i>Nationwide Insurance</i>).
filt3	1	Triage feature. Query name matches first letters of KB name (<i>e.g.</i> , <i>OA</i> and <i>Olympic Airlines</i>).
filtalias	3	Triage feature. PER, ORG, and GPE specific alias lists derived from Wikipedia.
filter	3	Triage feature. Strong string similarity: character Dice score > 0.X, skip bigram Dice score > 0.X, or Hamming distance <= 2.
goognil	2	Top Wikipedia pages ranked by Google for query name are missing from KB.
googtop	2	KB node corresponds to Wikipedia page ranked in top pages ranked by Google in Web search for query name.
googwiki	3	KB node corresponds to Wikipedia page ranked in top pages ranked by Google in search of query name at en.wikipedia.org.
kbtext	1	Query name appears early in KB article text.
kbtype	4	An assignment of 'PER, ORG, GPE, or RARE' based on manually compiled list for the classes listed in the knowledge base.
kname	5	Attributes about KB node name, such as having a comma, having parenthesis, or containing words indicative of a GPE or organization (<i>e.g.</i> , Inc., Corp.)
ldctype	4	The entity type provided for the KB node in the knowledge base (one of PER, ORG, GPE, or UKN).
name	4	Word-based name matching between query and KB name, such as sharing an 'organization' word (<i>e.g.</i> , <i>company</i> , bank, team, hospital etc.).
namedoc	4	Variations of KB node name appearing in the query article.
namedocp	7	Variations of KB node name appearing in the query article, where KB node name is first normalized, such as by removing a parenthetical part of the name (<i>e.g.</i> , Mike_Quigley_(footballer) would match <i>Mike Quigley</i>).
nand	1	A good string similarity match exists, but isn't the current KB node under consideration.
nil, nilon	3	Whether the current pair is a NIL candidate.
nilstat	21	The max, mean, and difference between max and mean for 7 atomic features for all KB candidates considered for the given query.
nilvec	4	Whether no, one, or several candidate nodes have matching names. (These feature are intended to give negative evidence for choosing the NIL candidate.)
nilvec2	3	Whether no, one, or several candidate nodes have matching names using fuzziier matching than with <i>nilvec</i> .
qname	4	Attributes about query name, such as containing words indicative of a GPE or organization.
qrytext	3	Whether the KB name appears near the query name or in the query article at all.
serifcovall	6	Coverage of the mentions identified in the query article by the SERIF tagger in KB facts or KB article text.
serifcovqry	9	Coverage of the mentions identified by SERIF in the query article and which are deemed coreferent with the query, in KB facts or KB article text.
serifq	26	Type or subtype assigned by SERIF to the query name in the article.
stcremp	4	One of four buckets for name matching based on Dice scores using skip bigrams.
tfidf	8	Cosine similarity between KB facts or text and the query article using TF/IDF weighting. Also information about the relative ranking of the KB node.
tfidfctx	2	Cosine similarity between KB facts or text and the local context of the query name in the provided article, using TF/IDF weighting.
trans	12	A similarity measure based on an probabilistic finite state transducer recognize the names as being accepted variants of one another using a PER/ORG/GPE-trained model.
wt	4	A good match using Wikitology.
wtattr	12	Attributes about the link graph for the Wikipedia page corresponding to the KB node (<i>e.g.</i> , number of inlinks).
wtnil	4	Whether any KB node is a top match based on Wikitology.

Table 1: Major classes of features.

to the shorter of q or k is effective at handling some deletions or word reorderings (e.g., “John Adams” and “John Quincy Adams”, or “Li Gong” and “Gong Li”). Checking whether all of the letters of q are found in the same order in k can be indicative (e.g., “Univ Maryland” would match “University of Maryland”).

Acronyms. Consideration of acronyms enables matches between “MIT” and “Madras Institute of Technology” or “Ministry of Industry and Trade.”

Aliases. Many aliases or nicknames are non-trivial to guess. For example JAVA is the stock symbol for Sun Microsystems, and “Ginger Spice” is a stage name of Geri Halliwell. We used multiple lists, including class-specific lists (i.e., for PER, ORG, and GPE) lists based on Wikipedia redirects.

2.1.2 Document Features

In the candidate ranking phase we considered features that used document information from either the document d provided with the query or the text present in the KB node (KB text).

Entity Mentions. Some features were simply based on presence of names, that is, whether q was found in the KB text, or whether k was present in d . Additionally we ran a named-entity tagger and relation finder, SERIF (Boschee et al., 2005), identified name and nominal mentions that were deemed co-referent with q in d , and tested whether these nouns were present in the KB text.

KB Facts. KB nodes contain infobox attributes (or facts); we tested whether these words of the facts were present in d , both locally to a mention of q , or anywhere in the provided article.

Document Similarity. We measured document similarity between d and the KB text in two ways: using cosine similarity with TF/IDF weighting (Salton and McGill, 1983); and using the Dice coefficient over bags of words. IDF values were approximated using counts from the Google 5-gram dataset as by Klein and Nelson (2008)

2.1.3 KB Node Attributes

Entity Classification. KB nodes contained classes (i.e., the original Wikipedia Infobox class) which can be assigned to PER, ORG, GPE, or *other*. The

Type	Reference KB	COE types
PER	14.0%	21.8%
ORG	6.8%	7.2 %
GPE	14.2%	19.4%
UKN	64.9%	-
BAD	-	38.3%
RARE	-	13.3%

Table 2: Entity classes in the reference KB.

reference knowledge base provided a type, which we used as a feature. However, the types in the reference KB were incomplete. Table 2 shows that only 35% of KB nodes were assigned a tag of PER, ORG, or GPE; the remainder were typed UKN (unknown).

Working through classes by frequency of occurrence we created an independent list that assigned types to 87% of nodes. In addition to PER, ORG, and GPE, we labelled many classes as BAD, meaning they could not be a PER, ORG, or GPE. This was helpful for discouraging selection of eponymous nodes named after famous entities (e.g., E0381570 “Michael Collins (film)” vs. E046674, the Irish political figure; or, the former U.S. president vs. E0194013 “John F. Kennedy International Airport”). We marked the remaining uncategorized classes as RARE.

Popularity. Though for some text genres it may be a dangerous bias to prefer common entities, it seemed helpful for this task to estimate measures of popularity. We did this in two ways. Our first approach, which was based on intrinsic properties of the KB nodes, used the fact that the nodes were derived from a snapshot of Wikipedia. We associated with each KB node several graph-theoretic properties of its corresponding Wikipedia page, namely, the number of in-links and out-links, and the page length (in bytes). These served as a weak approximation to PageRank. Our second approach used a PageRank calculation – we submitted the query string to Google and used the ranks of Wikipedia pages in the list of results as an attribute for their corresponding KB nodes.

NIL Indications. Some features can indicate whether it is likely or unlikely that there is a matching KB node for a query. For example, if many candidates have good name matches, it is likely that one of them is correct. Conversely, if no node has high

node-text/article similarity, or overlap between facts and the article text, it is likely that the entity is absent from the KB.

FST Model of Name Equivalence. Another measure of surface similarity between the query and a candidate was computed by training finite-state transducers similar to those described in Dreyer et al. (2008). These transducers assign a score to any string pair by summing over all alignments and scoring all contained character n -grams; we used n -grams of length 3 and less. The scores are combined using a global log-linear model. Since different spellings of a name may vary considerably in length (e.g., *J Miller* vs. *Jennifer Miller*) we eliminated the limit on consecutive insertions used in previous applications (Dreyer et al., 2008).² We also added a latent variable to the model that determines the direction in which the transducer is being used.

Wikilogy. Documents can be indexed with human or machine generated metadata consisting of keywords or categories in a domain-appropriate taxonomy. Using a system called *Wikilogy*, Syed et al. (2008) investigated use of ontology terms obtained from the explicit category system in Wikipedia as well as relationships induced from the hyperlink graph between related Wikipedia pages in. Following this approach we computed top-ranked categories for the input articles in the entity linking queries and used this information as features. If none of the candidate KB nodes had corresponding highly-ranked Wikilogy pages, we used this as a feature for the `NIL` condition (i.e., absent from the KB).

2.2 Feature Combination

We identified several features that we believed to be particularly useful; we then created combination features from the cross-product of this set with the entire list of features. The combination features included a feature for entity classification; a popularity feature based on Google’s rankings; document comparison using TF/IDF; coverage of co-referential nouns in the KB node text; and name similarity. The combinations were cascaded, so it was possible to

²Without such a limit, the objective function may diverge for certain parameters of the model; we detect such cases and learn to avoid them during training.

end up with a feature that represented, for example, *kbtype-is-per AND google-rank-is-low AND high-tfidf-score AND low-name-similarity*. The combined features increased the number of features from about 200 to roughly 26,000.

2.3 Training Data

Since our approach to entity linking is based on supervised learning we needed to obtain training data. To facilitate query generation we developed a tool to search the document collection and the KB, and to output training data in a suitable format. About a half-dozen members of our research group donated training queries. We also used the 119 sample queries provided by LDC as part of LDC2009E20. In total we trained our models using 1615 examples, of which 539 (33.4%) were PER, 618 (38.3%) were ORG, and 458 (28.4%) were GPE. 1301 of the examples (80.5%) were found in the KB, matching 300 unique entities; 314 examples (19.4%) were not present in the KB.

2.4 Experimental Results

We split our development data (N=1615) into two parts, *dev train* (N=908) and *dev test* (N=707) for in-house development; however all of the exemplars were used to construct the final model that we used for our submitted runs.

We submitted three runs for the entity linking task. The first run (COE-1) used our entire feature set. The second and third runs removed several features each, deletion of which had improved performance on our development dataset. COE-2 did not use the following feature classes: *conjoin-strcmpMedHiStrCmp*, *nand*, and *conjoin-aliasSomeAlias* (See Table 1). COE-3 did not use: *wattr* and *nil*.

Use of our entire feature set (run COE-1) yielded the best performance on the official evaluation metric, micro-averaged accuracy measured against all 3904 queries. Tables 3 and 4 give performance for all three runs averaged over all queries, broken out by presence in the KB (non-NIL vs. NIL).

Micro-average accuracies of roughly 80% were obtained for the test set for all three submissions. Prediction of absent entities (~85%) was higher than linkage of entities present in the KB (~70%), a trend that was also apparent in the best and median runs.

	<i>Best</i>	<i>Median</i>	<i>COE-1</i>	<i>COE-2</i>	<i>COE-3</i>
All	0.8217	0.7108	0.7984	0.7951	0.7941
non-NIL	0.7725	0.6352	0.7063	0.7164	0.6639
NIL	0.8919	0.7891	0.8677	0.8542	0.8919

Table 3: Micro-averaged accuracy for our submitted runs compared to best and median performance.

	<i>Best</i>	<i>Median</i>	<i>COE-1</i>	<i>COE-2</i>	<i>COE-3</i>
All	0.7704	0.6861	0.7695	0.7665	0.7704
non-NIL	0.6696	0.5335	0.6097	0.6160	0.5593
NIL	0.8789	0.7446	0.8464	0.8390	0.8721

Table 4: Macro-averaged accuracy for our submitted runs compared to best and median performance.

Our overall scores are substantially above median and near the top score reported. Run *COE-3* had the best overall macro-averaged accuracy, but was 2.8% below the top score using micro-averages. In comparison, a strawman approach of always predicting that entities were absent from the KB (*i.e.*, *NIL*) would receive an overall score of 0.5710.

Our candidate selection phase was the same for all three runs and we considered the correct answer 98.6% of the time; this is consistent with performance on our in-house development set. Some of the difficult cases that we failed to consider include: EL1687 (*Iron Lady*) which refers metaphorically to Yulia Tymoshenko; EL2885 (*PCC*), the Spanish-origin acronym for the Cuban Communist Party; and, EL2974 (*Queen City*), a former nickname for the city of Seattle, Washington.

There was a wide range in the number of candidates we considered per query. The mean was 76, but the median was 15 and the maximum 2772.³ In about 10% of cases there were four or fewer candidates and in 10% of cases there were more than 100 candidate KB nodes.

Among the entity types we observed that ORGs were more difficult to associate, which we attribute to the greater variation and complexity in the naming of organizations, along with the fact that they can be named after persons or locations. Table 5 breaks the results for COE-1 down by entity type.

77% of the distinct GPEs in the queries were present in the KB, but for PERs and ORGs these percentages were significantly lower, 19% and 30%

³For the query *Texas*.

<i>Class</i>	<i>N</i>	<i>All</i>	<i>non-NIL</i>	<i>NIL</i>
PER	627	0.8309	0.7098	0.9140
ORG	2710	0.7804	0.6367	0.8662
GPE	567	0.8483	0.8771	0.7750
All	3904	0.7984	0.7063	0.8677

Table 5: COE-1 performance by entity type.

<i>Class</i>	<i>Subgroups</i>
acronym	acro
alias	alias, bbnalias
baseline	filt1, filt2, filt3, fitalias, filter
facts	facts
graph	wtattr
nes	namedoc, namedocp, qrytext, serifcovall, serifcovqry, serifq
nil	goognil, nil, nilon, nilstat, nilvec, nilvec2
popularity	goog, googtop, googwiki
string	bs, bs2, kname, name, nand, qname, strcmp
text	dice, kbtext, tfidf, tfidfctxt
transducer	trans
type	kbtype, ldctype
wikit	wt, wtnil

Table 6: Grouping classes of features.

respectively.

2.5 Feature Effectiveness

To estimate the contributions of certain types of features, we first group features into a smaller set of classes (Table 6). We then perform two analyses. Our additive study starts by using only our triage features, and measures the change when adding each of the feature groups. (No feature combinations are used in the additive study.) Our ablative study starts using all of the features and measures the change when subtracting each feature group.

Feature Addition. Table 7 shows the changes that occur when different groups of features are added to our baseline, or triage set of features. The baseline condition, which only includes features based on string similarity or aliases, is not effective at finding correct alignments when target entities are present in the KB; the non-NIL percentage is only 46.21%. Inclusion of features based on analysis of named-entities, popularity measures (*e.g.*, Google rankings), and text comparisons (*e.g.*, query and kb document similarities) provided the largest gains.

<i>Class</i>	<i>All</i>	<i>non-NIL</i>	<i>NIL</i>
baseline	0.7264	0.4621	0.9251
acronym	0.7316	0.4860	0.9161
alias	0.7226	0.5081	0.8838
facts	0.6965	0.5558	0.8022
graph	0.6629	0.5248	0.7667
nes	0.7661	0.7181	0.8022
nil	0.7303	0.4884	0.9121
popularity	0.7597	0.7421	0.7730
string	0.6970	0.5099	0.8376
text	0.7313	0.6699	0.7775
transducer	0.7016	0.4812	0.8672
type	0.7139	0.5015	0.8735
wikit	0.7318	0.4549	0.9399

Table 7: Additive analysis: micro-averaged accuracy.

Feature Ablation. Table 8 reports the micro-averaged accuracy when feature groups are removed from our learning algorithm.

The overall changes are fairly small, roughly plus or minus 1%; however changes in non-NIL precision are larger, about plus or minus 5%. The relatively small degree of change indicates that there is considerable redundancy across our feature groupings.

In several cases, performance would have been improved by removing features - removing all feature combinations would have improved overall performance to 81.05%, apparently by gaining on non-NIL for a small decline on NIL detection.

3 Slot Filling

To explore the properties of the slot filling task and evaluate its difficulty, we built a multi-phase system. In the first phase, we identify sentences in the document collection that are likely to be relevant to the given query. The next phase extracts candidate answers from the selected sentences. We then filter and process these answers to address the requirements of the task, *e.g.*, limiting single slot answers, providing document support for an answer, etc.

Our system aggressively pruned sentences to ensure only the most relevant sentences were used; nonetheless, we produced many incorrect answers, both false positives and false negatives. In particular, in generating our official submission, a configuration error caused us to over-produce sentences for extraction, which in turn led to the inclusion of many sentences that were not relevant to the queries. After

<i>Class</i>	<i>All</i>	<i>non-NIL</i>	<i>NIL</i>
acronym	0.7969	0.7051	0.8659
alias	0.8025	0.7331	0.8546
baseline	0.7853	0.7278	0.8286
facts	0.7874	0.7206	0.8376
graph	0.8035	0.6943	0.8856
nes	0.7841	0.6740	0.8630
nil	0.7941	0.7224	0.8479
popularity	0.7569	0.6519	0.8359
string	0.7874	0.7540	0.8125
text	0.8076	0.7499	0.8511
transducer	0.7951	0.7475	0.8309
type	0.7843	0.6961	0.8506
wikit	0.7976	0.7284	0.8497
conjoin Alias	0.7989	0.7075	0.8677
conjoin Popularity	0.8058	0.7296	0.8632
conjoin NE	0.7964	0.7218	0.8524
conjoin String	0.7994	0.7260	0.8546
conjoin Text	0.7915	0.7313	0.8367
conjoin Type	0.7966	0.7337	0.8439
No Combinations	0.8105	0.7755	0.8367
No ablation	0.7984	0.7063	0.8677

Table 8: Ablation analysis: micro-averaged accuracy.

our official submission, we corrected the parameter settings and reran our system to produce a more constrained set of relevant sentences. We include both our official scores and scores for our subsequent corrected run.

3.1 Document Filtering

The goal of our filtering stage is to find all segments of text in the document collection that are relevant to the provided queries. We select relevant segments on a sentence level, determining whether each sentence is related or un-related to the query entity. We begin by indexing the document collection, then searching all relevant documents using the entity name as a query. We look for documents that match the entity name exactly. We would have preferred to use our entity linking system both for this task and for sentence selection below; however, shortness of time prevented us from doing so.

Our next task is to identify the sentences in the matched documents that mention the entity. Again, we relied on heuristics here instead of our entity linking system. We developed several approaches to selecting relevant sentences including, from most to least restrictive: 1) only select sentences that contain an exact reference to the entity; 2) return sen-

tences that refer to the entity (we detect references to entities using SERIF and select coreference chains within the document that include an exact match for the entity); and 3) return sentences in close proximity to the entity (*i.e.*, those that occur just before or after a sentence mentioning the entity). We experimented with all three and found that while the first yields a small number of sentences, it ensures relatively higher precision. Our official runs mistakenly contained sentences using the least restrictive of these filters, including some sentences that mistakenly passed through our filters unchecked. Our unofficial run described below contains sentences from the first filter only.

3.2 Answer Extraction

The second phase of our system extracts candidate answers from sentences that are relevant to the query entity. Since we used a restrictive sentence selector for the first pass, we opt for a more aggressive extraction system. Rather than rely on high precision patterns, we apply low precision patterns, using domain models to extract answers of the correct type.

To generate patterns, we used the knowledge base as a source of seed knowledge. We identified sentences in the document collection containing both the name of an entity and a fact from that entity’s KB entry. This produced (noisy) training sentences for each slot. We then extracted patterns for these slots using the pattern construction approach of Garera and Yarowsky (2009). We sorted these patterns based on their frequency of match to the sample sentences. The resulting patterns were then manually curated to select patterns that appeared to be of a high quality.

Next, we constructed a domain model for each slot. For many of the slots, this entailed creating a list of valid answers for the slot, such as religion, political party, education, etc. To generate these lists we selected answers that occurred multiple times in the knowledge base for each slot. Again, these lists were manually pruned to ensure only high-quality values.

Using these two resources, we constructed a rule based extraction system for each slot. Rules used a span identifier to detect which part of the sentence contained a likely answer. Options for detecting spans included using the patterns crafted above,

the context for manually selected keywords related to each slot, contexts of given regular expressions, the context of the entity mention, or the entire sentence. For domain models, slots used either a fixed list of slot fills (as described above), a typed named entity, a number, a date within a predefined range, or a URL. For named entity detection we used SERIF. We created slot extractors by pairing these two components, which then aggressively searched the provided sentences for possible answers. Table 9 provides a list of the answer span identifiers and domain models used for each slot.

As an example, consider the slot *title* for a person entity type. Our system extracted terms that appeared near the query entity in the sentence that also appeared in our list of person titles. For the entity George Kennedy and sentence: “It’s a mystery to me why (U.S.) publishers think people will pay more for less, especially when the online world offers so many alternatives, said George Kennedy, a professor emeritus at the University of Missouri School of Journalism who spent the summer in London.” our system extracted the candidate answer “professor emeritus” since it appeared in our list of titles. The advantage to such an approach is that it is robust to changes in sentence syntax and surrounding terms, although in some cases the aggressiveness of the extractions leads to incorrect answers.

Another example slot is *website* for organizations. We use predefined patterns, such as *visit X* and then look for URLs near the pattern, such as in “For more information on International Monetary Fund, visit <http://www.imf.org>.”

3.3 Answer Post-Processing

After extracting candidate answers, we processed them to conform to the track guidelines. We used simple heuristics to make decisions at this stage; there is clearly room for more intelligent processing.

Our processing included removing answers that are stopwords, those that appear in hand-crafted blacklists and those that have high string similarity with the slot fill of the existing KB node for the entity. To avoid redundancy in our answers we merged candidate answers that had high string similarity with other candidate answers for the same entity. Finally, we selected the candidate answer that we extracted the most times for single value slots

<i>Type</i>	<i>Slot</i>	<i>Span Identifier</i>	<i>Domain Model</i>	
PER	Age	Keywords	Numbers	
	Alternate Names	Patterns	Named Entity (PER)	
	Cause of Death	Patterns	List	
	Charges	Patterns	List	
	Children	Patterns	Named Entity (PER)	
	Date of Birth	Keywords	Date	
	Date of Death	Keywords	Date	
	Employee of	Patterns	Named Entity (ORG, GPE)	
	Member of	Patterns	Named Entity (ORG, GPE)	
	Origin	Query	List	
	Parents	Patterns	Named Entity (PER)	
	Other Family	Patterns	Named Entity (PER)	
	Place of Birth	Patterns	Named Entity (GPE, LOC, Fac)	
	Religion	Sentence	List	
	Residences	Patterns	Named Entity (GPE, LOC)	
	Siblings	Patterns	Named Entity (PER)	
	Spouse	Patterns	Named Entity (PER)	
	Title	Query	List	
	ORG	Alternate Names	Patterns	Named Entity (ORG)
		Dissolved	Regex	Date
Founded by		Patterns	Named Entity (PER)	
Founded		Regex	Date	
Headquarters		Patterns	Named Entity (GPE, LOC)	
Member of		Patterns	Named Entity (ORG, GPE)	
Members		Patterns	Named Entity (PER, ORG)	
Number of Employees		Patterns	Number	
Parents		Patterns	Named Entity (ORG)	
Political Religious Affiliation		Query	List	
Shareholders		Patterns	Named Entity (PER, ORG)	
Subsidiaries		Patterns	Named Entity (ORG)	
Top Members Employees		Patterns	Named Entity (PER)	
Website		Sentence	URL	
GPE		Alternate Names	Patterns	Named Entity (GPE)
	Capital	Patterns	List	
	Currency	Patterns	List	
	Established	Regex	Date	
	Political Parties	Patterns	List	
	Population	Patterns	Number	
	Subsidiary ORGs	Patterns	Named Entity (ORG)	
	Top Employees	Patterns	Named Entity (PER)	

Table 9: The answer span identifier and domain model used for each slot.

	Single		List		<i>SF</i> Value
	<i>Slot</i>	<i>NN</i>	<i>Slot</i>	<i>NN</i>	
NIL base	0.847	0	0.741	0	0.794
Median	0.514	0.154	0.439	0.141	0.461
Best	0.816	0.436	0.742	0.292	0.779
hltcoe1	0.765	0	0.518	0.035	0.641
hltcoe2	0.722	0	0.450	0.052	0.586
fixed	0.816	0	0.496	0.035	0.656
oracle	0.859	0	0.692	0.073	0.776

Table 10: Evaluation of slot-filling performance compared to best, median and the strong NIL baseline, including two unofficial runs. Columns are for single and list slots, including evaluation on Non-NIL (NN) only.

and took at most the top three frequently extracted answers for list slots.

3.4 Experimental Results

Our two official runs used the approach described above. `hltcoe1` used strict passage filtering, while `hltcoe2` used less strict filtering.

A coding error for our official runs let many non-relevant sentences through that should have been removed. We corrected the filter and created new runs, which we evaluated by counting answers unjudged by NIST as incorrect. The `fixed` run used the strict filter we had intended to apply to our official submission; it selected sentences containing exact matches to the query entity name, or those with a mention that SERIF deemed coreferential with an exact match.⁴ We also did an `oracle` experiment, using only those sentences containing both the exact query string and the exact string answer found in the `qrels`. About seventy percent of the correctly assessed answers occur in such sentences. While both these runs are “improved” by the official scoring, they only did so because they returned fewer answers, and thus approached the strong NIL baseline.

4 Conclusions

Our machine learning approach to entity linking worked well on the TAC 2009 data. While there are domains for which our current feature set will be less directly applicable, nonetheless we see our framework as a good one for the entity linking task

⁴SERIF output was available only for about half of the documents in the collection.

at large. We are currently improving the speed of our system to allow its use in slot filling and other language processing tasks.

Our approach to slot filling was a disappointment on these data. Our overall approach had worked well in other situations (Garera and Yarowsky, 2009). Significant factors that limited our performance on the TAC 2009 task included the low occurrence of slot fills in the document collection, heuristic selection of mentions that matched the query entity, sentence selection that was too broad, noisy patterns, patterns that did not identify both ends of the relation being extracted, impoverished domain models, heuristic combination of slot fills, and coding errors. We are performing further failure analysis on our slot filling system, and are reengineering it to ameliorate some of the problems we have identified.

References

- E. Boschee, R. Weischedel, and A. Zamanian. 2005. Automatic information extraction. In *First International Conference on Intelligence Analysis*, pages 2–4.
- Peter Christen. 2006. A comparison of personal name matching: Techniques and practical issues. Technical Report TR-CS-06-02, Australian National University.
- Markus Dreyer, Jason Smith, and Jason Eisner. 2008. Latent-variable modeling of string transductions with finite-state methods. In *EMNLP*, pages 1080–1089.
- Nikesh Garera and David Yarowsky. 2009. Structural, transitive and latent models for biographic fact extraction. In *European Chapter of the Association for Computational Linguistics (EACL)*.
- Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Knowledge Discovery and Data Mining (KDD)*.
- Martin Klein and Michael L. Nelson. 2008. A comparison of techniques for estimating IDF values to generate lexical signatures for the web. In *WIDM '08: Proceeding of the 10th ACM workshop on Web information and data management*, pages 39–46, New York, NY, USA. ACM.
- Gerard Salton and Michael McGill. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company.
- Zareen Syed, Tim Finin, and Anupam Joshi. 2008. Wikipedia as an ontology for describing documents. In *Proceedings of the Second International Conference on Weblogs and Social Media*. AAAI Press.