Supplementary Material: Learning Semantic Correspondences in Technical Documentation

Kyle Richardson and **Jonas Kuhn**
Institute of Natural Language Processing
University of Stuttgart
{kyle,jonas}@ims.uni-stuttgart.de

1 Dataset Information

We report additional details about our dataset collection.

1.1 Credits

Standard Library Documentation Figure 1 shows additional details of the different standard datasets, including pointers to the original source of the documentation, the standard library version numbers, and other details. To our knowledge, none of these datasets, excluding the Java Standard Library set from (Deng and Chrupała, 2014), have been used for the types of NLP experiments that we describe in the main paper.

Technical Manuals We also show information about the Unix dataset first reported in (Richardson and Kuhn, 2014). We are the first to report semantic parsing results using this dataset, since their original paper is only about the data resource.

1.2 Java Documentation Comparison

While we use the data (including the original splits) from Deng and Chrupała (2014) and apply some of the same baseline models, we made a few changes to their original evaluation, thus making our results not directly comparable. First, they train their models on the full documentation string, which we found to not have any advantage over just using the first sentence in the description (as we do for all other datasets). Secondly, they evaluate by ranking each example over a subset of the components in the validation/test splits, whereas we rank over all known components in the overall API (as also done in all other cases). We also made small differences changes to how tokenization is done in order to match the other datasets.

Despite these differences, as well as differences in the implementations of the baseline translation models, we were still able to reproduce (and in some cases improve) their original results.

1.3 Non-English Datasets

The documentation collections in languages other than English come exclusively from the PHP documentation set. Currently, these languages include: French, Spanish, Japanese, Russian, Turkish, and German (see paper for more details). More information about the translations, which are derived from the English original documentation set, can be found here: `http://doc.php.net/tutorial/`

1.4 Document Feature Information

Figure 1. shows information about the additional document-level features associated with each dataset.

Class information Some documentation sets include assertions about related functions or utilities, in the form of `see also` sections or links to other parts of the API. Such information can also be found in quick reference manuals or language cheat sheets available online, as well as from html structure.This information is used to define features in our discriminative model (see details in main paper).

Examples are shown in Figure 2. Since the datasets differ in terms of resources, the features used for each dataset are shown in gray.

Parameter descriptions In many datasets, the function documentation include additional textual descriptions of the function parameters. For the baseline translation models, these can add these fragmented pairs to the parallel training data. We also use this information as features in our discriminative model.

Return descriptions Similarly, some documentation also contains textual descriptions of return values, which can be used in the same way as described above.

Dataset	Version	Source(s)		Document Features: class info.	param.	return	section
Java	SE 6.0	(Deng and Chrupała, 2014)	**main data**	see-also	✓	✓	✓
Ruby	2.3.0	`ruby-doc.org/core-2.3.0/`	**main data**	fun. links	✓	✗	✗
		`ruby-doc.org/stdlib-2.3.0/`	**main data**				
PHP	3.0	`php.net/download-docs.php`	**main data**	see-also	✓	✓	✓
Python	2.7.11	`docs.python.org/2.7/library/`	**main data**	✗	✗	✗	✓
		`docs.python.org/2/reference/`	**background**	html	–	–	–
		`docs.python.org/2.7/library/` (*numpy*)	**num. library**	html	✓	✓	✗
Elisp	25.1	`gnu.org/software/emacs/manual`	**main data**	✗	✓	✗	✓
		`wikemacs.org/wiki/Emacs_Lisp_Cheat_Sheet`	**background**	html	–	–	–
		`github.com/magnars/s.el` (*s.el*)	**string library**	html	✗	✗	✗
		`github.com/magnars/dash.el` (*dash.el*)	**list library**	html	✗	✗	✗
Haskell	4.8.1	`hackage.haskell.org/package/base-4.8.1.0`	**main data**	✗	✗	✗	✓
Clojure	1.7	`clojure.org/api/api`	**main data**	✗	✗	✗	✓
		`clojuredocs.org/`	**main data**	see-also	✗	✗	✗
		`clojure.org/api/cheatsheet`	**background**	html	–	–	–
		`github.com/weavejester/medley` (*medley*)	**fun. library**	✗	✗	✗	✗
		`github.com/Raynes/fs` (*fs*)	**file-sys library**	✗	✗	✗	✗
		`github.com/ztellman/gloss` (*gloss*)	**byte library**	✗	✗	✗	✗
		`github.com/clj-time/clj-time` (*clj-time*)	**time library**	✗	✗	✗	✗
C	2.24	`gnu.org/software/libc/manual/`	**main data**	✗	✗	✗	✓
		`en.cppreference.com/w/c`	**main data**	html	✓	✓	✓
MIT Scheme	9.2	`https://www.gnu.org/software/mit-scheme`	**main data**	html	✗	✗	✗
Unix	–	(Richardson and Kuhn, 2014)	**main data**	see-also	✓	✗	✓

Figure 1: Further details of our corpus collection, including any background resources (**background**) or third party libraries (shown under double line) that were used. The last column shows additional document features used in our experiments. *Class* refers to information about general classes of functions, and *param* and *return* specify if the additional textual description of parameter values and return values (respectively) are included.

Language	Example Classes
Ruby	{`logger.info, logger.warn, logger.fatal, logger.debug, ...`}
Elisp	{`sin, cos, tan, asin, acos, atan, exp, log, log10, ...`}
Unix	{`iotop,iosnoop,iopattern,iopending, ...`}
PHP	{`ps_close_image, ps_open_image_file, ps_place_image, ...`}
C	{`UINT8_MAX UINT16_MAX UINT32_MAX UINT64_MAX UINT_FAST8_MAX, ...`}

Figure 2: Example classes, or abstract groupings of symbol types, extracted using document-level information.

Section descriptions Section descriptions are module or class level descriptions.

2 Implementation Details

We report some details about our implementation.

Preprocessing For programming languages that use camel case (e.g., Java), the function representations we normalized representations by replacing camel case boundaries with white space (e.g., `myFunction` → `my function`). Similarly, languages that use other common word delimiters (e.g., - in the lisp languages) are preprocessing in a similar fashion.

Haskell type declarations are converted into a linear form, or a polish notation, by converting function and container symbols into predicates (e.g., `fn ::(a,b) -> a` would be normalized as: `fn $tuple$2 a b a`). A similar idea is used in (Andreas et al., 2013). An example Haskell representation and tree is show in Figure 3.

Pseudo lexicons When training the translation models, we found that improvements can be achieved for latin character language datasets by adding matching component terms to the parallel training data. Such as idea is similar to the use of NP-lists in semantic parsing (e.g., Andreas et al. (2013)), and is a common trick used in other MT tasks (MacCartney et al., 2008).

Learning Parameters Standardly, all hyper parameters in Algorithm 2 are tuned to validation sets (i.e., number of iterations, learning rate, ..). Early stopping is done by monitoring training performance after each iteration using validation sets. Following (Zettlemoyer and Collins, 2009), the full learning rate α in line 7 is defined as $\alpha = \frac{\alpha_0}{1+c*o}$, where $o = i + t * n$, thus making c and

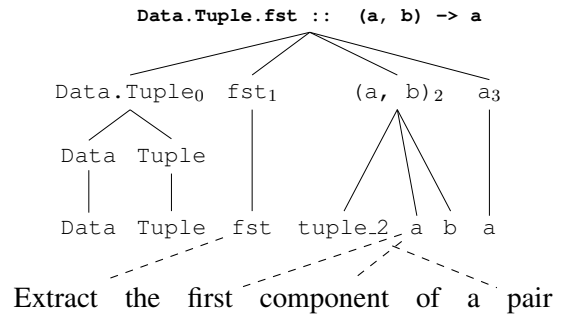

Figure 3: An example Haskell linear representation and tree structure.

α_0 the hyper-parameters.

3 Feature Details

Figure 5-6 show all the feature used. Since our datasets differ in terms of background resources, the green shows the features used in the top models for each dataset. While we did not do rigorous feature testing for this paper, we did notice that the Hiero rules sometimes negatively effect the results, and removed these feature accordingly.

Feature Selection A greedy, backward search selection method is employed for some datasets where overfitting seemed to be an issue. This is done in the following way: after training a complete model, features or templates that lead to incorrect predictions on the validation are greedily removed, and those whose removal increases the accuracy on the validation are shut off. The model is then retrained used the resulting selected set of features or templates. More details are documented in our source code release.

Phrase Features Phrase features are extracted for each input-ouput pair using standard word-based phrase heuristics over symmetric alignments (i.e., alignments from text $\rightarrow$ components, components $\rightarrow$ text). We use the *grow-diag* heuristic in all cases, see background in (Koehn, 2009).

Hierarchical phrase rules are extracted using a SAMT style rule extraction procedure (Zollmann and Venugopal, 2006). This can be informally described in the following way: when extracting phrases of the form $X \rightarrow$ `english word span|||foreign side` (using standard methods), sub alignments on each side that match tree patterns are replaced with NTs that match the name of the tree. The LHS of the rule X is also replaced, with with the tree name or the rule that results from combining the inner tree patterns using a small glue grammar. For example:

`function` $\rightarrow$ this function [`function`] `|||` [`function`]

Each symbol in brackets is a inner alignment-tree pattern. Given a text and component pair, a CKY-style chart procedure is used for finding all rules subject to the alignment.

4 Reproducibility

We are releasing all data reported on in this paper, as well as all the software used to complete the experiments. Please check the first author's webpage (http://www.ims.uni-stuttgart.de/institut/mitarbeiter/kyle), and the following Github (https://github.com/yakazimir) for more information.

References

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In *in Proceedings of ACL-2013*. pages 47–52.

Huijing Deng and Grzegorz Chrupała. 2014. Semantic approaches to software component retrieval with English queries. In *Proceedings of LREC-14*. pages 441–450.

Philipp Koehn. 2009. *Statistical Machine Translation*. Cambridge University Press.

Bill MacCartney, Michel Galley, and Christopher D Manning. 2008. A phrase-based alignment model for natural language inference. In *Proceedings of EMNLP-2008*. pages 802–811.

Kyle Richardson and Jonas Kuhn. 2014. UnixMan corpus: A resource for language learning in the Unix domain. In *Proceedings of LREC-2014*.

Luke S. Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of ACL-2009*. pages 976–984.

Andreas Zollmann and Ashish Venugopal. 2006. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*. pages 138–141.

id	descriptions	Java	Ruby	PHP$_{en}$	Python	Elisp	Haskell	Clojure	C	Unix	Scheme
1	model rank positions	■	■	■	■	■	■	■	■	■	■
2	english unigrams								■		
3	foreign unigrams								■		
4	e/f unigram pairs	■	■	■	■	■	■	■	■	■	■
5	# unigram matches	■	■	■	■	■	■	■	■	■	■
6	# unigram containments	■	■	■	■	■	■	■	■	■	■
7	type of unigram matches	■	■	■	■	■	■	■	■	■	■
8	# bigram matches	■	■	■	■	■	■	■	■	■	■
9	# bigram containments	■	■	■	■	■	■	■	■	■	■
10	foreign output length								■		
11	tree position of unigram contain.	■	■	■	■	■	■	■	■	■	■
12	tree position bigram matches	■	■	■	■	■	■	■	■	■	■
13	viterbi alignment pos.	■	■	■	■	■	■	■	■	■	■
14	tree pos. of alignment	■	■	■	■	■	■	■	■	■	■
15	phrase instances	■	■	■	■	■	■	■	■	■	■
16	# known phrases								■		
17	# matching phrases	■	■	■	■	■	■	■	■	■	■
18	# phrase containments	■	■	■	■	■	■	■	■	■	■
19	tree position of phrases	■	■	■	■	■	■	■	■	■	■
20	tree position matching phrases	■	■	■	■	■	■	■	■	■	■
21	tree position phrase contain.	■	■	■	■	■	■	■	■	■	■
22	tree position phrase overlap	■	■	■	■	■	■	■	■	■	■
23	size of phrase word overlap.	■	■	■	■	■	■	■	■	■	■
24	size of english phrase in matched	■	■	■	■	■	■	■	■	■	■
25	size of foreign phrase in matched	■	■	■	■	■	■	■	■	■	■
26	size of english phrases								■		
27	size of foreign phrases								■		
28	size of english overlapping phrases								■		
29	size of foreign overlapping phrases								■		
30	hiero phrase rule		■	■	■	■	■	■	■	■	■
31	# known hiero rules		■	■	■	■	■	■	■	■	■
32	# hiero rules with reordering		■	■	■	■	■	■	■	■	■
33	type of hiero reordering		■	■	■	■	■	■	■	■	■
34	english sides of hiero rules		■	■	■	■	■	■	■	■	■
35	foreign side of hiero rules		■	■	■	■	■	■	■	■	■
36	# unknown hiero rules		■	■	■	■	■	■	■	■	■
37	unigram pair in description	■	■	■	■	■	■	■	■	■	■
38	# of pairs in description	■	■	■	■	■	■	■	■	■	■
39	unigram pair in abstract class		■	■		■	■	■		■	■
40	unigram in see-also pair	■	■	■	■	■		■	■	■	■
41	unigram in see-also pair match	■	■	■	■	■	■	■	■	■	■
42	tree position of item in description	■	■	■	■	■	■	■	■	■	■
43	foreign abstract classes seen		■	■		■	■	■		■	■
44	type of english unigrams in descriptions	■	■	■	■	■	■	■	■	■	■
45	type of foreign words with descriptions	■	■	■	■	■	■	■	■	■	■
46	tree position of see-also pair	■	■	■	■	■	■	■	■	■	■
47	13+37	■	■	■	■	■	■	■	■	■	■
48	13+14+37								■		
49	5+13+14+37	■	■	■	■	■	■	■	■	■	■
50	5+14+37	■	■	■	■	■	■	■	■	■	■
51	15+40	■	■	■	■	■		■	■	■	■
52	31+40		■	■	■	■	■	■	■	■	■
53	english phrases and abstract classes	■	■	■	■	■	■	■	■	■	■
54	english phrases in descriptions	■	■	■	■	■	■	■	■	■	■
55	hiero english side and see-also		■	■	■	■		■	■	■	■
56	hiero foreign side and see-also		■	■	■	■		■	■	■	■

Figure 4: Description of features for our English datasets. Green shading shows that a particular feature was used in best model.

id	descriptions	PHP$_{fr}$	PHP$_{es}$	PHP$_{ja}$	PHP$_{ru}$	PHP$_{tr}$	PHP$_{de}$
1	model rank positions	■	■	■	■	■	■
2	english unigrams						
3	foreign unigrams						
4	e/f unigram pairs	■	■	■	■	■	■
5	# unigram matches	■	■	■	■	■	■
6	# unigram containments	■	■	■	■	■	■
7	type of unigram matches	■	■	■	■	■	■
8	# bigram matches	■	■	■	■	■	■
9	# bigram containments	■	■	■	■	■	■
10	foreign output length						
11	tree position of unigram contain.	■	■	■	■	■	■
12	tree position bigram matches	■	■	■	■	■	■
13	viterbi alignment pos.	■	■	■	■	■	■
14	tree pos. of alignment	■	■	■	■	■	■
15	phrase instances	■	■	■	■	■	■
16	# known phrases						
17	# matching phrases	■	■	■	■	■	■
18	# phrase containments	■	■	■	■	■	■
19	tree position of phrases	■	■	■	■	■	■
20	tree position matching phrases	■	■	■	■	■	■
21	tree position phrase contain.	■	■	■	■	■	■
22	tree position phrase overlap	■	■	■	■	■	■
23	size of phrase word overlap.	■	■	■	■	■	■
24	size of english phrase in matched	■	■	■	■	■	■
25	size of foreign phrase in matched	■	■	■	■	■	■
26	size of english phrases						
27	size of foreign phrases						
28	size of english overlapping phrases						
29	size of foreign overlapping phrases						
30	hiero phrase rule	■	■	■	■	■	■
31	# known hiero rules	■	■	■	■	■	■
32	# hiero rules with reordering	■	■	■	■	■	■
33	type of hiero reordering	■	■	■	■	■	■
34	english sides of hiero rules	■	■	■	■	■	■
35	foreign side of hiero rules	■	■	■	■	■	■
36	# unknown hiero rules	■	■	■	■	■	■
37	unigram pair in description	■	■	■	■	■	■
38	# of pairs in description	■	■	■	■	■	■
39	unigram pair in abstract class	■	■	■	■	■	■
40	unigram in see-also pair	■	■	■	■	■	■
41	unigram in see-also pair match	■	■	■	■	■	■
42	tree position of item in description	■	■	■	■	■	■
43	foreign abstract classes seen	■	■	■	■	■	■
44	type of english unigrams in descriptions	■	■	■	■	■	■
45	type of foreign words with descriptions	■	■	■	■	■	■
46	tree position of see-also pair	■	■	■	■	■	■
47	13+37	■	■	■	■	■	■
48	13+14+37						
49	5+13+14+37	■	■	■	■	■	■
50	5+14+37	■	■	■	■	■	■
51	15+40	■	■	■	■	■	■
52	31+40	■	■	■	■	■	■
53	english phrases and abstract classes	■	■	■	■	■	■
54	english phrases in descriptions	■	■	■	■	■	■
55	hiero english side and see-also	■	■	■	■	■	■
56	hiero foreign side and see-also	■	■	■	■	■	■

Figure 5: Feature information for our non-English datasets.