UnixMan Corpus: A Resource for Language Learning in the Unix Domain

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

Abstract

We present a new resource, the UnixMan Corpus, for studying language learning it the domain of Unix utility manuals. The corpus is built by mining Unix (and other Unix related) man pages for parallel example entries, consisting of English textual descriptions with corresponding command examples. The commands provide a grounded and ambiguous semantics for the textual descriptions, making the corpus of interest to work on Semantic Parsing and Grounded Language Learning. In contrast to standard resources for Semantic Parsing, which tend to be restricted to a small number of concepts and relations, the UnixMan Corpus spans a wide variety of utility genres and topics, and consists of hundreds of command and domain entity types. The semi-structured nature of the manuals also makes it easy to exploit other types of relevant information for Grounded Language Learning. We describe the details of the corpus and provide preliminary classification results.

Keywords: Language Resources, Semantics, Grounded Learning,Semantic Parsing

1. Introduction

Recent work on Semantic Parsing has focused on using non-traditional forms of data supervision, such as structured meaning representations, to jointly learn language syntax and semantics. A variety of corpora had been developed for these studies, often centering on a particular domain such as Sports, Geography, Navigation Instructions, among others (for a review and examples, see (Mooney, R. , 2007)). Such corpora consist of parallel text and meaning pairs, and the ultimate goal is to learn how to translate unseen text examples to correct structured meaning representations. A large number of tools have been used to study this problem, often taking insights from statistical Machine Translation (Andreas al., 1990; Wong, Y., et al, 2006), and statistical parsing (Zettlemoyer, L., et al, 2007). Particularly impressive has been the high accuracy that many systems achieve on benchmark datasets since such datasets tend to be small, with training sets numbering in the hundreds of sentences.

One salient research direction within this community looks at learning with weak supervision, where the target meaning representations are non-linguistic and grounded somehow in the domain being modeled. Examples include learning to parse high-level temporal expressions from raw date stamps (Angeli et al., 20012), learning about navigation instructions from grounded map cues and features (Chen, D., R. Mooney, 2011; Artzi et al., 20012) and interpreting sportscaster commentary based on automatically acquired streams of event sequences from a simulated sports game (Chen, D., R. Mooney, 2008). In the latter study, for example, the exact semantic label for each comment is uncertain, in that comments in the training phase are paired with all events in the game occurring around the time of the commentary, making the annotation mirror the underlying ambiguous perceptual context. In such a case, the computer must learn with *ambiguous supervision*, a task strongly advocated in (Mooney, R. , 2008). In this experimental setting, little to no manual annotation effort is required, since the annotation is extracted from (possibly noisy) grounded features that are independent of the related text. There has also been a strong emphasis on extrinsically evaluating the resulting systems on actual down-stream tasks (e.g. executing real navigation instructions).

Despite the encouraging results achieved in Semantic Parsing, the available datasets have many shortcomings, chiefly related to their small size and limited scope. The Sportscaster Corpus (Chen, D., R. Mooney, 2008) mentioned above is limited to 9 types of relations and a few dozen entity types, and GeoQuery (Zelle, J., et al, 1996), another benchmark dataset, is limited to around 38 predicate types. More challenging datasets have been introduced (Chen, D., R. Mooney, 2011), even for doing large-scale open domain Semantic Parsing (Cai Q. , 2013), but these are still fairly scarce. In addition, the annotations in such datasets encode very little knowledge, making it hard to learn interesting generalizations about target concepts and relations, or other forms of world knowledge.

We believe that there is strong justification for developing new resources for Grounded Language Learning and Semantic Parsing. As part of this effort, this paper presents the UnixMan corpus, a resource built semi-automatically by mining Unix (and other Unix related)[1] utility manuals for example entries consisting of user generated English textual descriptions with corresponding code examples. These code examples provide an ambiguous and grounded meaning representation for the associated text. The grounded nature of the representations make it ultimately possible to execute the target commands from natural language input. Unlike in other related datasets, the corpus ranges over many different genres and topics, making the set of command and entity types quite large. Along with the parallel examples, we also extract surrounding information about the example code syntax, in addition to information about

[1] the Unix man pages can be viewed here: http://www.liv.ac.uk/Unixhelp/alphabetical/. In addition to the core Unix utilities, other types of manuals are included for utilities usually distributed with Unix (e.g. posgreSQL and C manuals)

FILENAME: reindexdb.txt
NAME: reindexdb
DESCRIPTION: reindex a PostgreSQL database
SYNOPSIS: // Specification of the command's syntax and argument types

```
reindexdb [connection-option...] [--table | -t table][--index | -i index] [dbname]
reindexdb [connection-option...] [--all | -a]
reindexdb [connection-option...] [--system | -s] [dbname]
```

EXAMPLES // English descriptions with typed command sequences

To reindex the database test

reindexdb=reindexdb **dbname**=test

To reindex the table foo and the index bar in a database named abcd:

reindexdb=reindexdb **--table**=--table **table**=foo **--index**=--index **index**=bar **dbname**=abcd

SEE ALSO REINDEX, ... // Functionally related utilities

Figure 1: Example extraction from the man page for *reindexdb*. *DESCRIPTION* provides a high-level description of the overall utility. *SYNOPSIS* gives a syntactic definition of how to use the command, and also specifies the types of switches or options the command takes. *EXAMPLES* contain pairs of text descriptions with example commands, which are segmented and marked according to the types given (if available) in the synopsis (shown in **bold=**, original example code shown to the right of "="). *reindexdb* in this case refers to the main command name, whereas other parts of the command constitute particular options/switches or arguments. *SEE ALSO* gives a pointer to related utilities.

relations between different types of utilities.
In what follows, we describe the details of the corpus. We also describe a pilot study on classifying command types, and show how surrounding semi-structured information can be used in classification.

2. UnixMan Corpus

2.1. Corpus Creation

Figure 1 provides an example extracted from the man page for the *reindexdb* utility. Using a initial collection of around 11,000 raw man pages, a small subset was automatically extracted based on whether each file contained a set of desired fields, including a text description, synopsis, and set of example entries (see Figure 1 for details). Since the man pages are not uniformly formatted, this small subset (including 1,585 files) was then manually reviewed in order to organize the different sections and verify that the content was relevant. For example, several files contained the text *To be added..* (and several variations thereof) in the *EXAMPLES* field, without any actual entries. We also extracted *SEE ALSO* sections from pages that have this section, which indicate relations between related types of utilities.
For each page, the *SYNOPSIS* section is used to type and segment the command sequences inside the example entries, which consist of a command (e.g. reindenxdb) along with zero or more arguments (i.e. flags, switches or external files). This was done by manually matching parts of the given command sequences to the types assigned in one or more of the synopses. In cases where the synopses are underspecified or ambiguous (e.g. if not all types of arguments are explicitly provided), the type *other* or *option* was assigned to an unknown term. In a few cases, when the type of a unknown constituent is obvious from the context (e.g. a pathname or file), a new type is generated and matched to that item. In the end, the typed examples were automatically matched against the synopses once more, in order to check for errors. In general, the typing procedure could have been done entirely automatically, but given the noisy nature of the man pages, we decided to do it manually in order to avoid subtle mistakes.
As summarized in Figure 2, this process resulted in 327 command types, and 605 types of unique command arguments. Although we have mainly focused on matching the example sequences with the synopses as closely as possible, it is possible to either refine or further abstract the representations. For example, many of the original man pages include a *DESCRIPTION* section where certain switches or options are further classified and defined, and one could use this information to arrive at a more fine-grained representation. Similarly, using the *SEE ALSO* sections, it is possible to underspecify command types by grouping them into clusters, which is further discussed below. In addition to our corpus, we are also releasing parts of the original man pages, which can be linked to particular command entries for extracting additional information.

2.2. Example Entries

Our main interest is in the *EXAMPLES* section entries, which contain high-level text descriptions with example commands. In a Grounded Learning setting, the commands

	English Text Entries			*Command Entries*		
	# Sentences	# Tokens	# Single Tokens	# Commands	# Options (Average/Comm)	Avg. Sen/Com
330 **Unix Pages**	914	2282	1182 (51%)	327	605 (3.37)	2.76
GeoQuery	880	282	80 (28%)	38 (*# Preds*)	—	129 *(Sen/Pred)*
Sportscaster	1872	427	151 (35%)	9 (*# Rels*)	2.20 (avg. MRs/sentence)	—

Figure 2: Some details of the UnixMan Corpus (top) compared with other benchmark datasets (below, both English): SportsCaster (Chen, D., R. Mooney, 2008) and GeoQuery (Zelle, J., et al, 1996). The column *#Single Tokens* refers to the number of tokens that occur only once in the overall dataset. Column *#Options* shows the number of unique command options/modifiers over all command types, whereas *Average/Comm* refers to the average number of command options associated with a command. Column *Avg. Sen/Com* refers to the average number of sentences per command type. Numbers for the other dat sets were computed by the first author, and in the Sportscaster case were computed after fixing small errors. See text for more details.

Commands in a SEE ALSO class	# Equivalence Classes
206 (62%)	55

Example Command Classes:

```
{slapauth,slapdn,slapcat,slapadd,
  slapindex,slapacl,slaptest,slapd}

{authopen,unzip,ditto,lsbom,
   pax,open,drutil,funzip,ls,
   zip,hdid,hdiutil,raidutil,
   tar,zipinfo}

{git-pull,git-fetch,git-merge,...}

{iotop,iosnoop,iopattern,iopending}

{lam,cut,paste}

{CREATE_INDEX,REINDEX,
     DROP_INDEX,ALTER_INDEX}

{ALTER_FOREIGN_DATA_WRAPPER,
     DROP_FOREIGN_DATA_WRAPPER,
     CREATE_FOREIGN_DATA_WRAPPER}

{crl,x509,req,pkcs7,crl2pkcs7}

{rcsmerge,rlog,ci,rcsdiff,rcsclean}
```

Table 1: Information about the number of commands from the total that appear in a *SEE ALSO* class, either by having this field in their man page, or by occurring in different man page under this section. These sets of commands were then organized into equivalence classes. Sample equivalence classes are shown above.

constitute the meaning of the associated text. In the example in Figure 1, the *reindex* command can be thought of as a relation that predicates over a number of option arguments, including *table name*, *index name*, and *dbname* (in the second example in Figure 1). Learning the meaning of unseen texts, in this context, can be done by learning correspondences between text phrases and command fragments, which is the goal of most Semantic Parsing learning algorithms.

In contrast to other corpora for Semantic Parsing, however, the UnixMan Corpus contains many more entities and predicate types. As detailed in Figure 2, there are 327 types of command relations, and around 605 command arguments, with the number of arguments per command ranging from zero to 18. Figure 2 provides a comparison with the benchmark datasets discussed above, which are more constrained. It is particularly striking to compare the average number of sentences per command type, shown in the *Avg.Sen/Com section*, with the same number for the GeoQuery corpus. As a consequence, standard Semantic Parsing algorithm might not perform well on the UnixMan corpus since there are much less training instances per command and the corpus has a much larger vocabulary (compare *# Tokens* and *# Single Tokens*).

Since the relevance of each command argument option is not annotated, a major challenge for learning is figuring out which options, if any, are being described in the text, and detecting general patterns among different option types. This challenge is comparable to the problem of *learning with ambiguous supervision* encountered in the SportsCaster corpus, where each text example has a number of possible representations (see comparison in Figure 2). For example, flags like *–table* and *–index* in Figure 1 often do not have an obvious mapping to parts of the related text, and are instead provided by convention or used to distinguish different arguments in the command sequence. In addition, the flag *–index* in one utility might have a different meaning in a different utility. In this sense, the transformation from language to the commands is highly ambiguous and noisy.

2.3. Surrounding Structure

In addition to the example entries, other (semi-structured) information in the extractions can be used for learning. As discussed in detail above, information in the *SYNOPSIS* makes it easy to segment and type command constituents in the examples. Bracketing is often used inside synopses to show dependencies between command arguments (e.g. there is a bracketing in Figure 1 showing a dependency between *–table* and *table*), and this could be used for structuring the provided command sequences. The *DESCRIPTION*

	# Training Sentences	# Testing Sentences (#Labels)	Accuracy
original (no preprocessing)	575	214 (214)	**0.420**
original with preproc.	575	214 (214)	**0.467**
original with descriptions	902	327 (327)	**0.428**
original with cluster classes	575	214 (136)	**0.532**
preproc. with cluster classes	575	214 (136)	**0.537**

Table 2: Results of the classification study. *Accuracy* refers to the overall accuracy. See text for a description of the different conditions.

section provides an additional textual description of each command, which might be used as additional training data for learning a lexicon. Though not all commands contain a *SEE ALSO* field, this can similarly be used for clustering commands based on their similar functionality. In Table 1, we provide information about the clusters that emerge when we use this information, and we use this clustering information in the classification study below.

3. Pilot Studies

3.1. Classifying Commands

To test the difficulty of the dataset, we performed a pilot classification study looking at classifying command types. We broke the dataset into a testing and training set by taking all commands that have more than one text example, and removed a random sentence from this set for testing. The training set was used to train the Stanford Max-Entropy classifier (Manning, 2003), using n-grams features (unigrams through trigrams, plus prefix and suffix n-grams). We then evaluated to see if the classifier could predict the command type of each unseen sentence. Although we are ultimately interested in parsing sentences to full and executable commands sequences, testing the classification of commands type seems like a reasonable first step, and is seemingly not trivial given the large number of command labels.

As detailed in Table 2, several variations of the overall dataset were evaluated. The *original* set is created by taking commands that have more than one sentence/command pair and leaving out one random sentence for evaluation. The set includes a total of 575 sentences, out of a total 914, and covers 214 label or command types. Since the testing set has exactly one example for each label/command, a random-guess baseline (i.e. choosing the same random command for every training instance) would be somewhere around 0.0046. This *original* set was also varied in terms of whether preprocessing was done, which included word stemming, and abstraction of directory token types to a normalized label. We also created a set, *original with descriptions*, in which the overall text descriptions from each man page (see Figure 1 and *DESCRIPTION*) was added to the training sets as additional evidence. This set allowed us to evaluate the entire dataset, since every command with the description text makes it have at least two text descriptions. Finally, in order to test the effect of using equivalence classes from *SEE ALSO* fields, we labelled the sentences in *original* that fit into the equivalence class with the same label, which reduced the number of labels from 214 to 136 (referred in Table 2 as *cluster classes)*.

Preprocessing helped raise the accuracy, as seen by comparing the *original* non-preprocessed set with the *original* preprocessed. This was to be expected, because this reduces the total number of words (by removing basic inflection, ect.), perhaps making it easier to learn generalizations. Classifying sentences based on cluster classes also significantly increased the results, showing that the information extracted from *SEE ALSO* is useful, and seems to properly categorize not only the command types but their respective sentence examples. Including the descriptions in training resulted in low accuracy, although it is hard to know its exact effect since the set of classes is larger.

3.2. Towards Semantic Parsing

Although the basic classification study only partially solves the interesting problems related to the dataset, it seems to indicate that the data is fairly controlled. In future work we will concentrate on trying to learn full command sequences, which is well beyond the scope of simple classification. By using some of the cluster information, and other surrounding semi-structured information, we expect it to be possible to automatically create training labels that allow to extract generalizations about how to express commands, filenames, etc.. As discussed above, a major difficulty for applying standard Semantic Parsing methods will be dealing with the small number of training instances and relatively large vocabulary.

One general question is whether the weak supervision provided by the command sequences can be used to learn more complex linguistic generalizations, for example related to quantifier scope and negation. To our knowledge, this topic has not received much discussion in the Semantic Parsing literature.

4. Conclusions

We presented a new resource for studies on Semantic Parsing and Grounded Language Learning, which we believe raises new challenges and overcomes some of the shortcomings in other standard datasets. We performed initial classification study on the dataset to investigate its difficulty, which indicates that it is relatively controlled. Future work will focus on applying Semantic Parsing techniques in order to learn more generalizations, and ultimately the grounded mappings from language to commands sequences.

5. Acknowledgements

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) on the project SFB 732, "Incremental Specication in Context". We thank Christian Scheible for providing helpful comments and suggestions.

6. References

Andreas,. J., Vlanchos, A. and Clark, S. 2013. Semantic Parsing as Machine Translation In *Proceedings of ACL*, pages 47–52, Sofia, Bulgaria.

Angeli,G., Manning, C. and Jurafsky, D. 2012. Parsing Time: Learning to Interpret Time Expressions. in *Proc. of NAACL*, pages 446–455, Montreal, Canada.

Artzi,Y. and Zettlemoyer, L. 2013. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. in *Transactions of the Association for Computational Linguistics, (1):29-62*

Cai,Q. and Yates, A. 2013. Semantic Parsing Freebase: Towards Open-domain Semantic Parsing. in *Proc. of *SEM*, pages 328-338, Atlanta, Georgia.

Chen, D. and Mooney, R. 2008. Learning to Sportscase: A Test of Grounded Language Acquisition. in *Proc. of ICML*, pages 128-135, Helsinki, Finland.

Chen, D. and Mooney, R. 2011. Learning to Interpret Natural Language Navigation Instructions from Observations. in *Proc. of AAAI*, pages 859-865. San Francisco, California.

Manning,C. and Klein,D. 2003. Optimization, maxent models, and conditional estimation without magic. in *Proc. of NAACL-Tutorials*

Mooney, R. 2007. Learning for Semantic Parsing. in *Proc. of CICLing*, pages 311-324. Mexico City, Mexico.

Mooney, R. 2008. Learning to Connect Language and Perception. in *Proc. of AAAI*, pages 1598-1601. Chicago, Illinois.

Wong, Y. and Mooney, R. 2008. Learning for Semantic Parsing with Statistical Machine Translation. in *Proc. of HLT-NAACL*, pages 439-446, New York City, New York.

Zelle, J. and Mooney,. R 1996. Learning to Parse Database Queries Using Inductive Logic Programming. in *Proc. of AAAI*, pages 1050-1055. Portland, Oregon.

Zettlemoyer, L. and Collins, M. 2007. Online learning of relaxed CCG grammars for parsing to logical form. in *Proc. of EMNLP-CoNLL*, pages 678-687, Prague, Czech Repubic.