Large-scale Semantic Parsing via Schema Matching and Lexicon Extension

Qingqing Cai
Temple University
Computer and Information Sciences
qingqing.cai@temple.edu

Alexander Yates
Temple University
Computer and Information Sciences
yates@temple.edu

Abstract

Supervised training procedures for semantic parsers produce high-quality semantic parsers, but they have difficulty scaling to large databases because of the sheer number of logical constants for which they must see labeled training data. We present a technique for developing semantic parsers for large databases based on a reduction to standard supervised training algorithms, schema matching, and pattern learning. Leveraging techniques from each of these areas, we develop a semantic parser for Freebase that is capable of parsing questions with an F1 that improves by 0.42 over a purely-supervised learning algorithm.

1 Introduction

Semantic parsing is the task of translating natural language utterances to a formal meaning representation language (Chen et al., 2010; Liang et al., 2009; Clarke et al., 2010; Liang et al., 2011; Artzi and Zettlemoyer, 2011). There has been recent interest in producing such semantic parsers for large, heterogeneous databases like Freebase (Krishnamurthy and Mitchell, 2012; Cai and Yates, 2013) and Yago2 (Yahya et al., 2012), which has driven the development of semi-supervised and distantly-supervised training methods for semantic parsing. Previous purely-supervised approaches have been limited to smaller domains and databases, such as the GeoQuery database, in part because of the cost of labeling enough samples to cover all of the logical constants involved in a domain.

This paper investigates a reduction of the problem of building a semantic parser to three standard problems in semantics and machine learning: supervised training of a semantic parser, schema matching, and pattern learning. Figure 1 provides a visualization of our system architecture. We apply an existing supervised training algorithm for semantic parsing to a labeled data set. We apply schema matching techniques to the problem of finding correspondences between English words w and ontological symbols s. And we apply pattern learning techniques to incorporate new (w, s) pairs into the lexicon of the trained semantic parser.

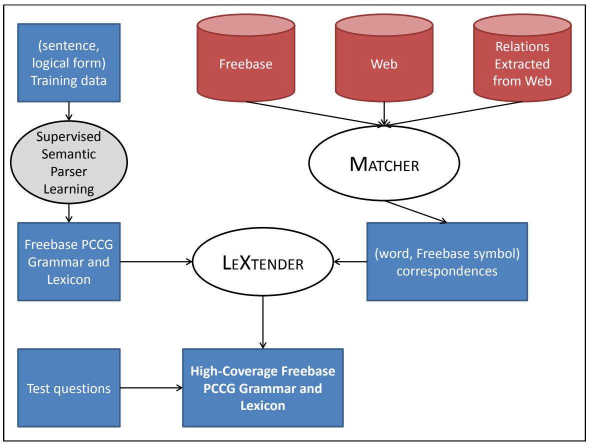
(sentence, logical form) Training data
Freebase
Web
Relations Extracted from Web
Supervised Semantic Parser Learning
MATCHER
Freebase PCCG Grammar and Lexicon
LEXTENDER
(word, Freebase symbol) correspondences
Test questions
High-Coverage Freebase PCCG Grammar and Lexicon

Figure 1: We reduce the task of learning a large-scale semantic parser to a combination of 1) a standard supervised algorithm for learning semantic parsers; 2) our MATCHER algorithm for finding correspondences between words and database symbols; and 3) our LEXTENDER algorithm for integrating (word, database symbol) matches into a semantic parsing lexicon.

This reduction allows us to apply standard techniques from each problem area, which in combination provide a large improvement over the purely-supervised approaches. On a dataset of 917 questions taken from 81 domains of the Freebase database, a standard learning algorithm for semantic parsing yields a parser with an F1 of 0.21, in large part because of the number of logical symbols that appear during testing but never appear during training. Our techniques can extend this parser to new logical symbols through schema matching, and yield a semantic parser with an F1 of 0.63 on the same task. On a more challenging task where training and test data are divided so that all logical constants in test are never observed during training, our approach yields a semantic parser with an F1 of 0.6, whereas the purely supervised approach cannot parse a single test question correctly. These results indicate that it is possible to automatically extend semantic parsers to symbols for which little or no training data has been observed.

The rest of this paper is organized as follows. The next section discusses related work. Section 3 describes our MATCHER algorithm for performing schema matching between a knowledge base and text. Section 4 explains how we use MATCHER's schema matching to extend a standard semantic parser to logical symbols for which it has seen no labeled training data. Section 5 analyzes the performance of MATCHER and our semantic parser. Section 6 concludes.

2 Previous Work

Two existing systems translate between natural language questions and database queries over large-scale databases. Yahya *et al.* (2012) report on a system for translating natural language queries to SPARQL queries over the Yago2 (Hoffart et al., 2013) database. Yago2 consists of information extracted from Wikipedia, WordNet, and other resources using manually-defined extraction patterns. The manual extraction patterns pre-define a link between natural language terms and Yago2 relations. Our techniques automate the process of identifying matches between textual phrases and database relation symbols, in order to scale up to databases with more relations, like Freebase. A more minor difference between Yahya *et al.*'s work and ours is that their system handles SPARQL queries, which do not handle aggregation queries like `argmax` and `count`. We rely on an existing semantic parsing technology to learn the language that will translate into such aggregation queries. On the other hand, their test questions involve more conjunctions and complex semantics than ours. Developing a dataset with more complicated semantics in the queries is part of our ongoing efforts.

Krishnamurthy and Mitchell (2012) also create a semantic parser for Freebase covering 77 of Freebase's over 2000 relations. Like our work, their technique uses distant supervision to drive training over a collection of sentences gathered from the Web, and they do not require any manually-labeled training data. However, their technique does require manual specification of rules that construct CCG lexical entries from dependency parses. In comparison, we fully automate the process of constructing CCG lexical entries for the semantic parser by making it a prediction task. We also leverage synonym-matching techniques for comparing relations extracted from text with Freebase relations. Finally, we test our results on a dataset of 917 questions covering over 600 Freebase relations, a more extensive test than the 50 questions used by Krishnamurthy and Mitchell.

Numerous methods exist for comparing two relations based on their sets of tuples. For instance, the DIRT system (Lin and Pantel, 2001) uses the mutual information between the (X, Y) argument pairs for two binary relations to measure the similarity between them, and clusters relations accordingly. More recent examples of similar techniques include the Resolver system (Yates and Etzioni, 2009) and Poon and Domingos's USP system (Poon and Domingos, 2009). Our techniques for comparing relations fit into this line of work, but they are novel in their application of these techniques to the task of comparing database relations and relations extracted from text.

Schema matching (Rahm and Bernstein, 2001; Ehrig et al., 2004; Giunchiglia et al., 2005) is a task from the database and knowledge representation community in which systems attempt to identify a "common schema" that covers the relations defined in a set of databases or ontologies, and the mapping between each individual database and the common schema. Owing to the complexity of the general case, researchers have resorted to defining standard similarity metrics between relations and attributes, as well as machine learning algorithms for learning and predicting matches between relations (Doan et al., 2004; Wick et al., 2008b; Wick et al., 2008a; Nottelmann and Straccia, 2007; Berlin and Motro, 2006). These techniques consider only matches between relational databases, whereas we apply these ideas to matches between Freebase and extracted relations. Schema matching in the database sense often considers complex matches between relations (Dhamanka et al., 2004), whereas as our techniques are currently restricted to matches involving one database relation and one relation extracted from text.

3 Textual Schema Matching

3.1 Problem Formulation

The textual schema matching task is to identify natural language words and phrases that correspond with each relation and entity in a fixed schema for a relational database. To formalize this task, we first introduce some notation.

A *schema* $S = (E, R, C, I)$ consists of a set of entities E, a set of relations R, a set of categories C, and a set of instances I. Categories are one-argument predicates (*e.g.*, $\texttt{film}(e)$), and relations are two- (or more-) argument predicates (*e.g.*, $\texttt{directed\_by}(e_1, e_2)$). Instances are known tuples of entities that make a relation or category true, such as `film(Titanic)` or `directed_by(Titanic, James Cameron)`. For a given $r \in R$ (or $c \in C$), $I_S(r)$ indicates the set of known instances of r in schema S (and likewise for $I_S(c)$). Examples of such schemas include Freebase (Bollacker et al., 2008) and Yago2 (Hoffart et al., 2013). We say a schema is a *textual* schema if it has been extracted from free text, such as the Nell (Carlson et al., 2010) and ReVerb (Fader et al., 2011) extracted databases.

Given a textual schema T and a database schema D, the textual schema matching task is to identify an alignment or *matching* $M \subset R_T \times R_D$ such that $(r_T, r_D) \in M$ if and only if r_T can be used to refer to r_D in normal language usage. The problem would be greatly simplified if M were a 1-1 function, but in practice most database relations can be referred to in many ways by natural language users: for instance, `film_actor` can be referenced by the English verbs "played," "acted," and "starred," along with morphological variants of them. In addition, many English verbs can refer to several different relations in Freebase: "make" can refer to `computer_processor_manufacturer` or `distilled_spirits_producer`, among many others. Our MATCHER algorithm for textual schema matching handles this by producing a confidence score for every possible (r_T, r_D) pair, which downstream applications can then use to reason about the possible alignments.

Even worse than the ambiguities in alignment, some textual relations do not correspond with any database relation exactly, but instead they correspond with a projection of a relation, or a join between multiple relations, or another complex view of a database schema. As a simple example, "actress" corresponds to a subset of the Freebase `film_actor` relation that intersects with the set $\{x\colon \texttt{gender}(x, \texttt{female})\}$. MATCHER can only determine that "actress" aligns with `film_actor` or not; it cannot produce an alignment between "actress" and a join of `film_actor` and `gender`. These more complex alignments are an important consideration for future work, but as our experiments will show, quite useful alignments can be produced without handling these more complex cases.

3.2 Identifying candidate matches

MATCHER uses a generate-and-test architecture for determining M. It uses a Web search engine to issue queries for a database relation r_D consisting of all the entities in a tuple $t \in I_D(r_D)$. 1000 tuples for each r_D are randomly chosen for issuing queries. The system then retrieves matching snippets from the search engine results. It uses the top 10 results for each search engine query. It then counts the frequency of each word type in the set of retrieved snippets for r_D. The top 500 non-stopword word types are chosen as candidates for matches with r_D. We denote the candidate set for r_D as $C(r_D)$.

MATCHER's threshold of 500 candidates for $C(r_D)$ results in a maximum possible recall of just less than 0.8 for the alignments in our dataset, but even if we double the threshold to 1000, the recall improves only slightly to 0.82. We therefore settled on 500 as a point with an acceptable upper bound on recall, while also producing an acceptable number of candidate terms for further processing.

3.3 Pattern-based match selection

The candidate pool $C(r_D)$ of 500 word types is significantly smaller than the set of all textual relations, but it is also extremely noisy. The candidates may include non-relation words, or other frequent but unrelated words. They may also include words that are highly related to r_D, but not actually corresponding textual relations. For instance, the candidate set for `film_director` in Freebase includes words like "directed," but also words like "film," "movie," "written," "produced," and "starring." We use a series of filters based on synonym-detection techniques to help select the true matching candidates from $C(r_D)$.

	Pattern	Condition	Example
1.	"r_T in E"	r_T ends with "-ed" and E has type `datetime` or `location`	"founded in 1989"
2.	"r_T by E"	r_T ends with "-ed"	"invented by Edison"
3.	"r_T such as E"	r_T ends with "-s"	"directors such as Tarantino"
4.	"E is a(n) r_T"	all cases	"Paul Rudd is an actor"

Table 1: Patterns used by MATCHER as evidence of a match between r_D and r_T. E represents an entity randomly selected from the tuples in $I_D(r_D)$.

The first type of evidence we consider for identifying true matches from $C(r_D)$ consists of pattern-matching. Relation words that express r_D will often be found in complex grammatical constructions, and often they will be separated from their entity arguments by long-distance dependencies. However, over a large corpus, one would expect that in at least some cases, the relation word will appear in a simple, relatively-unambiguous grammatical construction that connects r_T with entities from r_D. For instance, entities e from the relationship `automotive_designer` appear in the pattern "designed by e" more than 100 times as often as the next most-common patterns, "considered by e" and "worked by e."

MATCHER use searches over the Web to count the number of instances where a candidate r_T appears in simple patterns that involve entities from r_D. Greater counts for these patterns yield greater evidence of a correct match between r_D and r_T. Table 1 provides a list of patterns that we consider. For each r_D and each $r_T \in C(r_D)$, MATCHER randomly selects 10 entities from r_D's tuples to include in its pattern queries. Two of the patterns are targeted at past-tense verbs, and the other two patterns at nominal relation words.

MATCHER computes statistics similar to pointwise mutual information (PMI) (Turney, 2001) to measure how related r_D and r_T are, for each pattern p. Let c(p, r_D, r_T) indicate the sum of all the counts for a particular pattern p, database relation, and textual relation:

$$f_p(r_T, r_D) = \frac{\mathrm{c}(p, r_D, r_T)}{\sum\limits_{r'_D} \mathrm{c}(p, r'_D, r_T) * \sum\limits_{r'_T} \mathrm{c}(p, r_D, r'_T)}$$

For the sum over all r'_D, we use all r'_D in Freebase for which r_T was extracted as a candidate.

One downside of the pattern-matching evidence is the sheer number of queries it requires. Freebase currently has over 2,000 relations. For each r_D, we have up to 500 candidate r_T, up to 4 patterns, and up to 10 entities per pattern. To cover all of Freebase, MATCHER needs $2,000 \times 500 \times 4 \times 10 = 40$ million queries, or just over 1.25 years if it issues 1 query per second (we covered approximately one-quarter of Freebase's relations in our experiments). Using more patterns and more entities per pattern are desirable for accumulating more evidence about candidate matches, but there is a trade-off with the time required to issue the necessary queries.

3.4 Comparing database relations with extracted relations

Open Information Extraction (Open IE) systems (Banko et al., 2007) can often provide a large set of extracted tuples for a given r_T, which MATCHER can then use to make much more comprehensive comparisons with the full tuple set for r_D than the pattern-matching technique allows.

MATCHER employs a form of PMI to compute the degree of relatedness between r_D and r_T. In its simplest form, MATCHER computes:

$$PMI(r_T, r_D) = \frac{|I_D(r_D) \cap I_T(r_T)|}{|I_D(r_D)| \cdot |I_T(r_T)|} \quad (1)$$

While this PMI statistic is already quite useful, we have found that in practice there are many cases where an exact match between tuples in $I_D(r_D)$ and tuples in $I_T(r_T)$ is too strict of a criterion. MATCHER uses a variety of approximate matches to compute variations of this statistic. Considered as predictors for the true matches in M, these variations of the PMI statistic have a lower precision, in that they are more likely to have high values for incorrect matches. However, they also have a higher recall: that is, they will have a high value for correct candidates in $C(r_D)$ when the strict version of PMI does not. Table 2 lists all the variations used by MATCHER.

Statistics for (r_T, r_D)
$s_\kappa(r_T, r_D) = \frac{\sum_{t_D \in I_D(r_D)} \sum_{t_T \in I_T(r_T)} \kappa(t_D, t_T)}{\lvert I_D(r_D)\rvert \cdot \lvert I_T(r_T)\rvert}$
$s'_\kappa(r_T, r_D) = \frac{s_\kappa(r_T, r_D)}{\sum_{r'_D} s_\kappa(r'_D, r_T)}$
$s''(r_T, r_D) = \frac{\lvert I_T(r_T)\rvert}{\lvert I_D(r_D)\rvert}$

Table 2: MATCHER statistics: for each κ function for comparing two tuples (given in Table 3), MATCHER computes the statistics above to compare r_D and r_T. The PMI statistic in Equation 1 corresponds to s_κ where κ =strict match over Φ =full tuples.

$\kappa(t_1, t_2)$ for comparing tuples t_1, t_2
strict match: $\begin{cases} 1, \text{if } \Phi(t_1) = \Phi'(t_2) \\ 0, \text{otherwise.} \end{cases}$
type match: $\begin{cases} 1, \text{if } \forall_k cat(\Phi(t_1)_k) \\ \qquad = cat(\Phi'(t_2)_k) \\ 0, \text{otherwise.} \end{cases}$

Table 3: MATCHER's κ functions for computing whether two tuples are similar. cat maps an entity to a category (or type) in the schema. MATCHER has a different κ function for each possible combination of Φ and Φ' functions, which are given in Table 4.

MATCHER uses an API for the ReVerb Open IE system[1] (Fader et al., 2011) to collect $I(r_T)$, for each r_T. The API for ReVerb allows for relational queries in which some subset of the entity strings, entity categories, and relation string are specified. The API returns all matching triples; types must match exactly, but relation or argument strings in the query will match any relation or argument that contains the query string as a substring. MATCHER queries ReVerb with three different types of queries for each r_T, specifying the types for both arguments, or just the type of the first argument, or just the second argument. Types for arguments are taken from the types of arguments for a potentially matching r_D in Freebase. To avoid overwhelming the ReVerb servers, for our experiments we limited MATCHER to queries for the top 80 $r_T \in C(r_D)$, when they are ranked according to frequency during the candidate identification process.

$\Phi(t)$ for tuple $t = (e_1, \ldots, e_n)$
$\forall_i e_i$ (projection to one dimension)
$(e_1, \ldots, e_n)$ (full tuple)
$\forall_{\sigma(\cdot)}(e_{\sigma(1)}, \ldots, e_{\sigma(n)})$ (permutation)

Table 4: MATCHER's Φ functions for projecting or permuting a tuple. σ indicates a permutation of the indices.

3.5 Regression models for scoring candidates

Pattern statistics, the ReVerb statistics from Table 2, and the count of r_T during the candidate identification step all provide evidence for correct matches between r_D and r_T. MATCHER uses a regression model to combine these various statistics into a score for (r_T, r_D). The regression model is a linear regression with least-squares parameter estimation; we experimented with support vector regression models with non-linear kernels, with no significant improvements in accuracy. Section 5 explains the dataset we use to train this model. Unlike a classifier, MATCHER does not output any single matching M. However, downstream applications can easily convert MATCHER's output into a matching M by, for instance, selecting the top K candidate r_T values for each r_D, or by selecting all (r_T, r_D) pairs with a score over a chosen threshold. Our experiments analyze MATCHER's success by comparing its performance across a range of different values for the number of r_T matches for each r_D.

4 Extending a Semantic Parser Using a Schema Alignment

An alignment between textual relations and database relations has many possible uses: for example, it might be used to allow queries over a database to be answered using additional information stored in an extracted relation store, or it might be used to deduce clusters of synonymous relation words in English. Here, we describe an application in which we build a question-answering system for Freebase by extending a standard learning technique for semantic parsing with schema alignment information.

As a starting point, we used the UBL system

[1] http://openie.cs.washington.edu/

developed by Kwiatkowski *et al.* (2010) to learn a semantic parser based on probabilistic Combinatory Categorial Grammar (PCCG). Source code for UBL is freely available. Its authors found that it achieves results competitive with the state-of-the-art on a variety of standard semantic parsing data sets, including Geo250 English (0.85 F1). Using a fixed CCG grammar and a procedure based on unification in second-order logic, UBL learns a lexicon Λ from the training data which includes entries like:

Example Lexical Entries

New York City $\vdash NP : \texttt{new\_york}$
neighborhoods in $\vdash$
$S\backslash NP/NP : \lambda x \lambda y . \texttt{neighborhoods}(x, y)$

Example CCG Grammar Rules

$X/Y : f \quad Y : g \Rightarrow X : f(g)$
$Y : g \quad X\backslash Y : f \Rightarrow X : f(g)$

Using Λ, UBL selects a logical form z for a sentence S by selecting the z with the most likely parse derivations y: $h(S) = \arg\max_z \sum_y p(y, z|x; \theta, \Lambda)$. The probabilistic model is a log-linear model with features for lexical entries used in the parse, as well as indicator features for relation-argument pairs in the logical form, to capture selectional preferences. Inference (parsing) and parameter estimation are driven by standard dynamic programming algorithms (Clark and Curran, 2007), while lexicon induction is based on a novel search procedure through the space of possible higher-order logic unification operations that yield the desired logical form for a training sentence.

Our Freebase data covers 81 of the 86 core domains in Freebase, and 635 of its over 2000 relations, but we wish to develop a semantic parser that can scale to all of Freebase. UBL gets us part of the way there, by inducing a PCCG grammar, as well as lexical entries for function words that must be handled in all domains. It can also learn lexical entries for relations r_D that appear in the training data. However, UBL has no way to learn lexical entries for the many valid (r_T, r_D) pairs that do not appear during training.

We use MATCHER's learned alignment to extend the semantic parser that we get from UBL by automatically adding in lexical entries for Freebase relations. Essentially, for each (r_T, r_D) from MATCHER's output, we wish to construct a lexical entry that states that r_T's semantics resembles $\lambda x \lambda y . r_D(x, y)$. However, this simple process is complicated by the fact that the semantic parser requires two additional types of information for each lexical entry: a syntactic category, and a weight. Furthermore, for many cases the appropriate semantics are significantly more complex than this pattern.

To extend the learned semantic parser to a semantic parser for all of Freebase, we introduce a prediction task, which we call *semantic lexicon extension*: given a matching M together with scores for each pair in M, predict the syntactic category Syn, lambda-calculus semantics Sem, and weight W for a full lexical entry for each $(r_T, r_D) \in M$. One advantage of the reduction approach to learning a semantic parser is that we can automatically construct training examples for this prediction task from the other components in the reduction. We use the output lexical entries learned by UBL as (potentially noisy) examples of true lexical entries for (r_T, r_D) pairs where r_T matches the word in one of UBL's lexical entries, and r_D forms part of the semantics in the same lexical entry. For (r_T, r_D) pairs in M where r_D occurs in UBL's lexical entries, but not paired with r_T, we create dummy "negative" lexical entries with very low weights, one for each possible syntactic category observed in all lexical entries. Note that in order to train LEXTENDER, we need the output of MATCHER for the relations in UBL's training data, as well as UBL's output lexicon from the training data.

Our system for this prediction task, which we call LEXTENDER (for Lexicon eXtender), factors into three components: $P(Sem|r_D, r_T, score)$, $P(Syn|Sem, r_D, r_T, score)$, and $P(W|Syn, Sem, r_D, r_T, score)$. This factorization is trivial in that it introduces no independence assumptions, but it helps in designing models for the task. We set the event space for random variable Sem to be the set of all lambda calculus expressions observed in UBL's output lexicon, modulo the names of specific Freebase relations. For instance, if the lexicon includes two entries whose semantics are $\lambda x \lambda y$. film_actor(x, y) and $\lambda x \lambda y$. book_author(x, y), the event space would include the single expression in which relations film_actor and book_author were replaced by a new variable: $\lambda p \lambda x \lambda y . p(x, y)$. The final semantics for a lexical entry is then constructed by substituting r_D for p, or more formally, by a function application $Sem(r_D)$. The event space for Syn consists of all syntactic categories in UBL's output lexicon, and W ranges over $\mathbf{R}$.

LEXTENDER's model for Sem and Syn are Naïve Bayes classifiers (NBC), with features for the part-of-speech for r_T (taken from a POS tagger), the suffix of r_T, the number of arguments of r_D, and the argument types of r_D. For Syn, we add a feature for the predicted value of Sem. For W, we use a linear regression model whose features are the score from MATCHER, the probabilities from the Syn and Sem NBC models, and the average weight of all lexical entries in UBL with matching syntax and semantics. Using the predictions from these models, LEXTENDER extends UBL's learned lexicon with all possible lexical entries with their predicted weights, although typically only a few lexical entries have high enough weight to make a difference during parsing. Pruning entries with low weights could improve the memory and time requirements for parsing, but these were not an issue in our experiments, so we did not investigate this further.

5 Experiments

We conducted experiments to test the ability of MATCHER and LEXTENDER to produce a semantic parser for Freebase. We first analyze MATCHER on the task of finding matches between Freebase relations and textual relations. We then compare the performance of the semantic parser learned by UBL with its extension provided by LEXTENDER on a dataset of English questions posed to Freebase.

5.1 Experimental Setup

Freebase (Bollacker et al., 2008) is a free, online, user-contributed, relational database (www.freebase.com) covering many different domains of knowledge. The full schema and contents are available for download. The "Freebase Commons" subset of Freebase, which is our focus, consists of 86 domains, an average of 25 relations per domain (total of 2134 relations), and 615,000 known instances per domain (53 million instances total). As a reference point, the GeoQuery database — which is a standard benchmark database for semantic parsing — contains a single domain (geography), 8 relations, and 880 total instances.

Examples

1. What are the neighborhoods in New York City?
   λx . $\texttt{neighborhoods}(\texttt{new\_york}, x)$
2. How many countries use the rupee?
   $\texttt{count}(x)$. $\texttt{countries\_used}(\texttt{rupee}, x)$
3. How many Peabody Award winners are there?
   $\texttt{count}(x)$. $\exists y$. $\texttt{award\_honor}(y) \wedge$
   $\texttt{award\_winner}(y, x) \wedge$
   $\texttt{award}(y, \texttt{peabody\_award})$

Figure 2: Example questions with their logical forms. The logical forms make use of Freebase symbols as logical constants, as well as a few additional symbols such as `count` and `argmin`, to allow for aggregation queries.

Our dataset contains 917 questions (on average, 6.3 words per question) and a meaning representation for each question written in a variant of lambda calculus[2]. 81 domains are represented in the data set, and the lambda calculus forms contain 635 distinct Freebase relations. The most common domains, `film` and `business`, each took up no more than 6% of the overall dataset. Several examples are listed in Fig. 2. The questions were provided by two native English speakers. No restrictions were placed on the type of questions they should produce, except that they should produce questions for multiple domains. By inspection, a large majority of the questions appear to be answerable from Freebase, although no instructions were given to restrict questions to this sort. We also created a dataset of alignments from these annotated questions by creating an alignment for each Freebase relation mentioned in the logical form for a question, paired with a manually-selected word from the question.

5.2 Alignment Tests

We measured the precision and recall of MATCHER's output against the manually labeled data. Let M be the set of (r_T, r_D) matches produced by the system, and G the set of matches in the gold-standard manual data. We define

[2] The data is available from the second author's website.

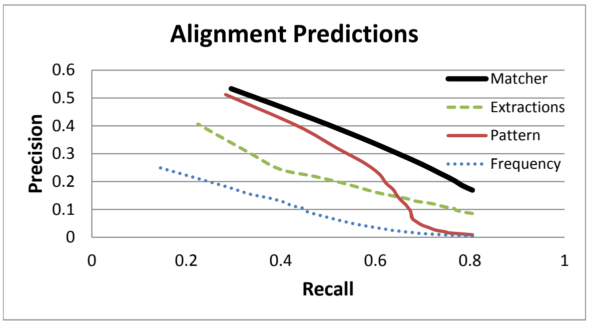

Figure 3: MATCHER's Pattern features and Extractions features complement one another, so that in combination they outperform either subset on its own, especially at the high-recall end of the curve.

precision and recall as:

$$P = \frac{|M \cap G|}{|M|}, R = \frac{|M \cap G|}{|G|}$$

Figure 3 shows a Precision-Recall (PR) curve for MATCHER and three baselines: a "Frequency" model that ranks candidate matches for r_D by their frequency during the candidate identification step; a "Pattern" model that uses MATCHER's linear regression model for ranking, but is restricted to only the pattern-based features; and an "Extractions" model that similarly restricts the ranking model to ReVerb features. We have three folds in our data; the alignments for relation r_D in one fold are predicted by models trained on the other two folds. Once all of the alignments in all three folds are scored, we generate points on the PR curve by applying a threshold to the model's ranking, and treating all alignments above the threshold as the set of predicted alignments.

All regression models for learning alignments outperform the Frequency ranking by a wide margin. The Pattern model outperforms the Extractions model at the high-precision, low-recall end of the curve. At the high-recall points, the Pattern model drops quickly in precision. However, the combination of the two kinds of features in MATCHER yields improved precision at all levels of recall.

5.3 Semantic Parsing Tests

While our alignment tests can tell us in relative terms how well different models are performing, it is difficult to assess these models in absolute terms, since alignments are not typical applications that people care about in their own right. We now compare our alignments on a semantic parsing task for Freebase.

In a first semantic parsing experiment, we train UBL, MATCHER, and LEXTENDER on a random sample of 70% of the questions, and test them on the remaining 30%. In a second test, we focus on the hard case where all questions from the test set contain logical constants that have never been seen before during training. We split the data into 3 folds, making sure that no Freebase domain has symbols appearing in questions in more than one fold. We then perform 3-fold cross-validation for all of our supervised models. We varied the number of matches that the alignment model (MATCHER, Pattern, Extractions, or Frequency) could make for each Freebase relation, and measured semantic parsing performance as a function of the number of matches.

Figure 4 shows the F1 scores for these semantic parsers, judged by exact match between the top-scoring logical form from the parser and the manually-produced logical form. Exact-match tests are overly-strict, in the sense that the system may be judged incorrect even when the logical form that is produced is logically equivalent to the correct logical form. However, by inspection such cases appear to be very rare in our data, and the exact-match criterion is often used in other semantic parsing experimental settings.

The semantic parsers produced by MATCHER+LEXTENDER and the other alignment techniques significantly outperform the baseline semantic parser learned by UBL, which achieves an overall F1 of 0.21 on these questions in the 70/30 split of the data, and an F1 of 0 in the cross-domain experiment. Purely-supervised approaches to this data are severely limited, since they have almost no chance of correctly parsing questions that refer to logical symbols that never appeared during training. However, MATCHER and LEXTENDER combine with UBL to produce an effective semantic parser. The best semantic parser we tested, which was produced by UBL, MATCHER, and LEXTENDER with 9 matches per Freebase relation, had a precision of 0.67 and a recall of 0.59 on the 70/30 split experiment.

The difference in alignment performance between MATCHER, Pattern, and Extractions carries over to semantic parsing. MATCHER drops in F1 with more matches as additional matches tend to be low-quality and low-probability, whereas Pat-

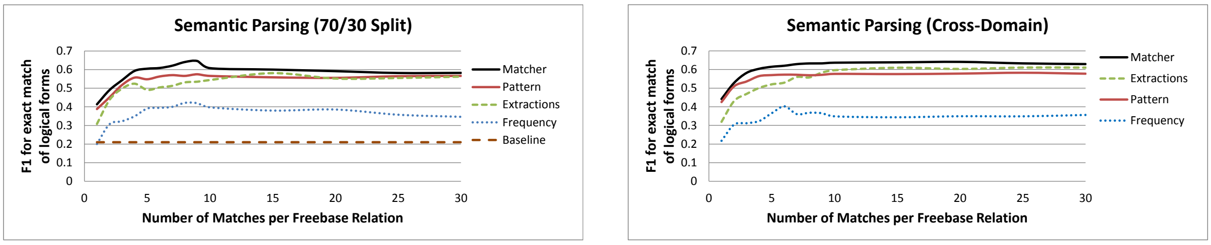

Figure 4: Semantic parsers produced by UBL+MATCHER+LEXTENDER outperform the purely-supervised baseline semantic parser on a random 70/30 split of the data (left) by as much as 0.42 in F1. In the case of this split and in the case of a cross-domain experiment (right), UBL+MATCHER+LEXTENDER outperforms UBL+Pattern+LEXTENDER by as much as 0.06 in F1.

tern and Extractions keep improving as more low-probability alignments are added. Interestingly, the Extractions model begins to overtake the Pattern model in F1 at higher numbers of matches, and all three models trend toward convergence in F1 with increasing numbers of matches. Nevertheless, MATCHER clearly improves over both, and reaches a higher F1 than either Pattern or Extractions using a small number of matches, which corresponds to a smaller lexicon and a leaner model.

To place these results in context, many different semantic parsers for databases like GeoQuery and ATIS (including parsers produced by UBL) have achieved F1 scores of 0.85 and higher. However, in all such tests, the test questions refer to logical constants that also appeared during training, allowing supervised techniques for learning semantic parsers to achieve strong accuracy. As we have argued, Freebase is large enough that is difficult to produce enough labeled training data to cover all of its logical constants. An unsupervised semantic parser for GeoQuery has achieved an F1 score of 0.66 (Goldwasser et al., 2011), impressive in its own right and slightly better than our F1 score. However, this parser was given questions which it knew *a priori* to contain words that refer to the logical constants in the database. Our MATCHER and LEXTENDER systems address a different challenge: how to learn a semantic parser for Freebase given the Web and a set of initial labeled questions.

6 Conclusion

Scaling semantic parsing to large databases requires an engineering effort to handle large datasets, but also novel algorithms to extend semantic parsing models to testing examples that look significantly different from labeled training data. The MATCHER and LEXTENDER algorithms represent an initial investigation into such techniques, with early results indicating that semantic parsers can handle Freebase questions on a large variety of domains with an F1 of 0.63.

We hope that our techniques and datasets will spur further research into this area. In particular, more research is needed to handle more complex matches between database and textual relations, and to handle more complex natural language queries. As mentioned in section 3.1, words like "actress" cannot be addressed by the current methodology, since MATCHER assumes that a word maps to a single Freebase relation, but the closest Freebase equivalent to the meaning of "actress" involves the two relations `film_actor` and `gender`. Another limitation is that our current methodology focuses on finding matches for nouns and verbs. Other important limitations of the current methodology include:

- the assumption that function words have no domain-specific meaning, which prepositions in particular can violate;
- low accuracy when there are few relevant results among the set of extracted relations;
- and the restriction to a single database (Freebase) for finding answers.

While significant challenges remain, the reduction of large-scale semantic parsing to a combination of schema matching and supervised learning offers a new path toward building high-coverage semantic parsers.

References

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping Semantic Parsers from Conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. 2007. Open information extraction from the web. In *IJCAI*.

Jacob Berlin and Amihai Motro. 2006. Database schema matching using machine learning with feature selection. In *Advanced Information Systems Engineering*. Springer.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1247–1250.

Qingqing Cai and Alexander Yates. 2013. Semantic Parsing Freebase: Towards Open-Domain Semantic Parsing. In *Second Joint Conference on Lexical and Computational Semantics*.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. 2010. Toward an Architecture for Never-Ending Language Learning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI 2010)*.

David L. Chen, Joohyun Kim, and Raymond J. Mooney. 2010. Training a Multilingual Sportscaster: Using Perceptual Context to Learn Language. *Journal of Artificial Intelligence Research*, 37:397–435.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, 33(4):493–552.

J. Clarke, D. Goldwasser, M. Chang, and D. Roth. 2010. Driving semantic parsing from the world's response. In *Computational Natural Language Learning (CoNLL)*.

R. Dhamanka, Y. Lee, A. Doan, A. Halevy, and P. Domingos. 2004. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *SIGMOD*.

A. Doan, J. Madhavan, P. Domingos, and A. Halevy. 2004. Ontology Matching: A Machine Learning Approach. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 397–416. Springer-Verlag.

M. Ehrig, P. Haase, N. Stojanovic, and M. Hefke. 2004. Similarity for ontologies-a comprehensive framework. In *Workshop Enterprise Modelling and Ontology: Ingredients for Interoperability, PAKM*.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying Relations for Open Information Extraction. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. 2005. Semantic schema matching. *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 347–365.

D. Goldwasser, R. Reichart, J. Clarke, and D. Roth. 2011. Confidence driven unsupervised semantic parsing. In *Association for Computational Linguistics (ACL)*.

Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Artificial Intelligence*, 194:28–61, January.

Jayant Krishnamurthy and Tom Mitchell. 2012. Weakly Supervised Training of Semantic Parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing Probabilistic CCG Grammars from Logical Form with Higher-order Unification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

P. Liang, M. I. Jordan, and D. Klein. 2009. Learning semantic correspondences with less supervision. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*.

D. Lin and P. Pantel. 2001. DIRT – Discovery of Inference Rules from Text. In *KDD*.

Henrik Nottelmann and Umberto Straccia. 2007. Information retrieval and machine learning for probabilistic schema matching. *Information processing & management*, 43(3):552–576.

Hoifung Poon and Pedro Domingos. 2009. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, EMNLP '09, pages 1–10, Stroudsburg, PA, USA. Association for Computational Linguistics.

E. Rahm and P.A. Bernstein. 2001. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350.

P. D. Turney. 2001. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Procs. of the Twelfth European Conference on Machine Learning (ECML)*, pages 491–502, Freiburg, Germany.

M. Wick, K. Rohanimanesh, A. McCallum, and A.H. Doan. 2008a. A discriminative approach to ontology mapping. In *International Workshop on New Trends in Information Integration (NTII) at VLDB WS*.

M.L. Wick, K. Rohanimanesh, K. Schultz, and A. McCallum. 2008b. A unified approach for schema matching, coreference and canonicalization. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*.

Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. 2012. Natural Language Questions for the Web of Data. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Alexander Yates and Oren Etzioni. 2009. Unsupervised methods for determining object and relation synonyms on the web. *Journal of Artificial Intelligence Research (JAIR)*, 34:255–296, March.