Identifying Domain Adjacent Instances for Semantic Parsers

James Ferguson University of Washington jfferg@cs.washington.edu
Janara Christensen Google Inc.
Edward Li Google Inc
Edgar Gonzàlez Google Inc
{jchristensen,gavinbelson,edgargip}@google.com

Abstract

When the semantics of a sentence are not representable in a semantic parser's output schema, parsing will inevitably fail. Detection of these instances is commonly treated as an out-of-domain classification problem. However, there is also a more subtle scenario in which the test data is drawn from the same domain. In addition to formalizing this problem of *domain-adjacency*, we present a comparison of various baselines that could be used to solve it. We also propose a new simple sentence representation that emphasizes words which are unexpected. This approach improves the performance of a downstream semantic parser run on in-domain and domain-adjacent instances.

Air Travel Domain	
Example In-Domain Predicates	
$buyTicket$	Buy ticket LGA to SFO on 3/12
$flightStatus$	What's the status of my SF flight?
$switchFlight$	Change it to the 8am SFO flight
$cancelFlight$	Cancel my flight to SFO
$awardTravel$	I want to fly to SFO with miles
Example Domain-Adjacent Predicates	
$changeSeat$	Change my seat to 23A
$milesUpgrade$	Upgrade my flight with my miles
$arrivalGate$	Gate that my SFO flight arrives at
$mileageStatus$	What's my miles status
Example Out-of-Domain Predicates	
$transferMoney$	Transfer $200 to checking
$addTimer$	Add a timer for 3 minutes
$restaurantSearch$	Thai restaurants in SF
$scheduleMeeting$	Set up a 9am meeting with Amy

Figure 1: In this example, an **air travel** semantic parser is trained on data containing in-domain predicates. A test instance cannot be parsed correctly if it contains any domain-adjacent or out-of-domain predicates.

1 Introduction

Semantic parsers map text to logical forms, which can then be used by downstream components to fulfill an action. Consider, for example, a system for booking air travel, in which a user provides natural language input, and a downstream subsystem is able to make or cancel flight reservations. Users of the system typically have a general understanding of its purpose, so the input will revolve around the correct topic of **air travel**. However, they are unlikely to know the limits of the system's functionality, and may provide inputs for which the expected action is beyond its capabilities, such as asking to change seats on a flight reservation. Because the logical schema is designed with fulfillment in mind, no logical form can capture the semantics of these sentences, making it impossible for the parser to generate a correct parse. Any output the parser generates will cause unintended actions to be executed downstream. For example, asking to change seats might be misparsed and executed as changing flights. Instead, the parser should identify that this input is beyond its scope so the condition can be handled.[1] In this paper, we formalize this pervasive problem, which we call *domain-adjacent instance identification*.

While this task is similar to that of identifying *out-of-domain* input instances (e.g., **banking** with respect to **air travel**), it is much more subtle — the instances come from roughly the same domain as the parser's training examples, and thus use very similar language. Domain adjacency is a property with respect to the parser's output schema, independent of the data used to train it.

In this paper, we formalize this task, and propose a simple approach for representing sentences in which words are weighted by how likely they are to differentiate between in-domain and domain-adjacent instances. Note that while this approach can also be applied to out-of-domain

[1]In the final page of the paper, we suggest a few immediate downstream system behaviors when a domain-adjacent instance is identified, but others have investigated the related problem of teaching the system new behavior (Azaria et al., 2016).

instances, in this paper we are interested in its performance on domain-adjacent instances. We describe an evaluation framework for this new task and, finally, evaluate our proposed method against a set of baselines, comparing performance on the domain-adjacent classification problem and a downstream semantic parsing task.

2 Problem Setting

A semantic parser can be seen as a function φ that maps sentences x in a natural language $\mathcal{L}$ to logical forms $y \in \mathcal{Y}$. Assuming the existence of an oracle parser $\hat{\varphi}$, the problem we propose in this paper is that of determining, for a given test instance x, whether it belongs to the *domain* Φ of $\hat{\varphi}$, i.e., if its semantics can be encoded in the schema $\mathcal{Y}$.

In real-world usage, the input sentences x will be generated by a human user, who associates the capabilities of the parser to a particular topic (e.g., **air travel**). Thus most of the $x \in \mathcal{L} \setminus \Phi$ will share topic with the $\hat{x} \in \Phi$. Because of the similarity between these x and $\hat{x}$, we call this task *identification of domain-adjacent instances*.

3 Approach

Our goal is to identify input instances whose semantics are not representable in the parser's output schema, and we assume only an in-domain dataset is available at training time. Our approach is based on determining similarity to these training instances. We split the task in two parts: 1) encode the sentences to a compact representation that preserves the needed information, and 2) given these representations, identify which sentences are so dissimilar that they are unlikely to be parseable with any schema that covers the training set.

3.1 Sentence Representation

Among recent work in distributional semantics, averaging the word vectors to represent a sentence (Wieting et al., 2016; Adi et al., 2017) has proven to be a simple and robust approach. However, we have an intuition that words which are *unexpected* in their context given the training data may be a strong signal that an instance is domain-adjacent. To incorporate this signal, we propose a weighted average, in which the weight corresponds to how unexpected the word is in its context. For example, given in-domain predicates from Figure 1, in the domain-adjacent sentence "Upgrade my flight to SFO with my miles", *upgrade* should receive a much higher weight than *flight* or *SFO*.

Our weighting scheme is as follows: We use the cosine distance between the expected ($\bar{v}_i$) and the actual ($\hat{v}_i$) domain-specific word embedding at a given position (i) in a sentence to compute its weight: $w_i = 1 - cos(\bar{v}_i, \hat{v}_i)$. The expected word embedding is computed using the context embeddings, $\bar{v}_i = \sum_{j=i-c, j\neq i}^{i+c} \hat{v}_j$, where $\hat{v}_j$ is a domain-specific word embedding, in a window of size c around position i. Intuitively, w_i represents how surprising the word is in the context.

Since our training set is too small to directly learn domain-specific embeddings, we learn a mapping from general pre-trained embeddings. We train a continuous bag-of-words model (Mikolov et al., 2013) in which we pass pre-trained embeddings (v_i) instead of 1-hot vectors, as input to the embedding layer. The layer thus learns a mapping from pre-trained to domain-specific embeddings ($\hat{v}_i$). We use this mapping to compute new embeddings for words that are missing from the training set. Only words that do not have pre-trained embeddings are ignored.

Finally, for a sentence with n words, we take the weighted average of the pre-trained embeddings of the words in the sentence, using the weights from above: $S = (\sum_{i=1}^{n} w_i v_i) \,/\, (\sum_{i=1}^{n} w_i)$.

This approach assigns high weight to words that differ significantly from what is expected based on the training data. By combining these weights with the pre-trained word embeddings, we allow the model to incorporate external information, improving generalization beyond the training set.

3.2 Domain-Adjacent Model

A number of techniques can be applied to predict whether a sentence is domain-adjacent from its continuous representation. Of the methods we tried, we found k-nearest neighbors (Angiulli and Pizzuti, 2002) to perform best: to classify a sentence, we calculate the average cosine distance between its embedding and its k nearest neighbors in the training data, and label it domain-adjacent if this value is greater than some threshold. This simpler model relies more heavily on the external information brought in by pre-trained word embeddings, while more complex models seem to overfit to the training data.

Basketball	*numGamesPlayed*
Blocks	*length*
Calendar	*startTime*
Housing	*size*
Publications	*venue*
Recipes	*preparationTime*
Restaurants	*starRating*
Social	*educationStartDate*, *employmentEndDate*

Table 1: Predicates excluded from training and considered domain-adjacent. Domains have 5-20 predicates.

4 Evaluation

In this section, we introduce an evaluation framework for this new task. We consider training and test sets from a single domain, with only the latter containing domain-adjacent instances. Test instances are classified individually, and we measure performance on in-domain/domain-adjacent classification and semantic parsing.

4.1 Dataset and Semantic Parser

We simulate this setting by adapting the OVERNIGHT dataset (Wang et al., 2015). This dataset is composed of queries drawn from eight domains, each having a set of seven to eighteen distinct semantic predicates. Queries consist of a crowd-sourced textual sentence and its corresponding logical form containing one or more of these domain-specific semantic predicates.

For each domain, we select a set of predicates to exclude from the logical schema (see Table 1), and remove all instances containing these predicates from the training set (since they are now domain-adjacent). We then train a domain-adjacent model and semantic parser on the remaining training data and attempt to identify the domain-adjacent examples in the test data. We use the train/test splits from Wang et al. (2015). In all experiments, we use the SEMPRE parser (Berant et al., 2013).

4.2 Baselines

Because this is a novel task, and results are not comparable to previous work, we report results from a variety of baseline systems. The first baseline, CONFIDENCE, identifies instances as domain-adjacent if the semantic parser's confidence in its predictions is below some threshold.

The remaining baselines follow the two-part approach from Section 3. AUTOENCODER is inspired by Ryu et al. (2017)'s work on identifying *out-of-domain* examples. For the sentence representation, this method uses a bi-LSTM with self-attention, trained to predict the semantic predicates, and concatenates the final hidden state from each direction as the sentence representation. An autoencoder is used as the domain-adjacent classifier.

The remaining methods use the nearest neighbor model discussed in Section 3.2. For sentence representations, we include baselines drawn from different neural approaches. In CBOW, we simply average the pre-trained word embeddings in the sentence. In CNN, we train a two-layer CNN with a final softmax layer to predict the semantic predicates for a sentence. We concatenate the mean pooling of each layer as the sentence representation. In LSTM, we use the same sentence representation as in AUTOENCODER, with the nearest neighbor domain-adjacent model. Finally, SURPRISE is the approach presented in Section 3.1.

4.3 Direct Evaluation

We first directly evaluate the identification of domain-adjacent instances: Table 2 reports the area under a receiver operating characteristic curve (AUC) for the considered models (Fawcett, 2006). SURPRISE generally performs the best on this evaluation; and, in general, the simpler models tend to perform better, suggesting that more complex approaches tune too much to the training data.

Qualitatively, for domains where the SURPRISE model performs better, it places higher weight on words we would consider important for distinguishing domain-adjacent sentences. For example in "show me recipes with longer preparation times than rice pudding" from **Recipes**, "longer" and "preparation" have the highest weights. In **Social**, there are two in-domain predicates (*employmentStartDate* and *educationEndDate*) which use very similar wording to those that are domain-adjacent, making it difficult to isolate surprising words. The weights in this domain seem to instead emphasize unusual wordings such as "soonest" in "employees with the soonest finish date".

4.3.1 Ablation Analysis

In order to determine the contribution of each one of the components of SURPRISE, we performed an ablation analysis comparing the following modifications of the method: CBOW, as described above, using an unweighted average of pre-trained embeddings; FREQUENCY, using a

	Basketball	Blocks	Calendar	Housing	Publications	Recipes	Restaurants	Social
AUTOENCODER	0.801	0.479	0.766	0.781	0.874	0.722	0.581	0.774
CONFIDENCE	0.660	0.738	0.697	0.648	0.631	0.651	0.730	0.573
CBOW	0.743	0.782	0.662	0.910	0.884	0.670	0.911	0.675
CNN	**0.916**	0.654	**0.862**	0.792	0.908	0.505	0.840	**0.813**
LSTM	0.826	0.571	0.741	0.912	0.827	0.487	0.593	0.754
SURPRISE	0.755	**0.827**	0.817	**0.933**	**0.978**	**0.758**	**0.941**	0.545

Table 2: AUC for domain-adjacent instance identification, using KNN as the domain-adjacent model.

	Basketball	Blocks	Calendar	Housing	Publications	Recipes	Restaurants	Social
CBOW	0.743	0.782	0.662	0.910	0.884	0.670	0.911	**0.675**
FREQUENCY	0.656	0.703	0.771	0.884	0.887	0.667	0.834	0.591
PRETRAINED	0.612	0.636	0.512	0.819	0.842	0.526	0.858	0.538
SURPRISE	**0.755**	**0.827**	**0.817**	**0.933**	**0.978**	**0.758**	**0.941**	0.545

Table 3: AUC for domain-adjacent instance identification, using ablated versions of SURPRISE with KNN.

weighted average of pre-trained embeddings, with weights based on inverse document frequency in the training set; PRETRAINED, using the surprise schema but with weights determined using pre-trained embeddings; and the full SUPRISE as presented above. Each approach adds one component (weighting, surprise-based weights, and domain-specific embeddings) with respect to the previous one.

The results of the experiment are shown in Table 3. We can see that FREQUENCY performs slightly worse than CBOW and PRETRAINED performs even worse than that. We can conclude that the combination of the weighting schema and the tuned vectors is what makes SUPRISE effective.

4.4 Downstream Task Evaluation

We next evaluate how including the domain-adjacent predictions affects the performance of a semantic parser. In a real setting, when the semantic parser is presented with domain-adjacent input that is beyond its scope, the correct behavior is to label it as such so that it can be handled properly by downstream components. To simulate this behavior, we set the gold parse for domain-adjacent instances to be an empty parse, and automatically assign an empty parse to any instance that is identified as domain-adjacent. We report accuracy of the semantic parser with 20% domain-adjacent test data. We include two additional models: NOFILTER, in which nothing is labeled domain-adjacent, and ORACLE, in which all the domain-adjacent instances are correctly identified. For each baseline requiring a threshold, we set it such that 3% of the instances in the dev set would be marked as domain-adjacent (intuitively, this represents the error-tolerance of the system).

Table 4 shows the results for this experiment. In general, the relative performance is similar to that in the direct evaluation (e.g. SURPRISE tends to do well on most domains, but performs poorly on BASKETBALL and SOCIAL in particular). However, in this evaluation, misclassifying an instance as domain-adjacent if the semantic parser would have accurately parsed it is worse than misclassifying the instance if the semantic parser could not have accurately parsed it. For example, in SOCIAL we can thus infer that SURPRISE is marking some instances as domain-adjacent that would otherwise be accurately parsed as the performance there is actually worse than for NOFILTER.

5 Related Work

Domain-adjacency identification is a new task, but relatively little effort has been devoted to even the related task of identifying out-of-domain instances (i.e., from completely separate domains) for semantic parsers. Hakkani-Tur et al. (2015) approached the problem by clustering sentences based on shared subgraphs in their general semantic parses; Ryu et al. (2017) classify sentences with autoencoder reconstruction error.

Prior distributional semantics work to create compact sentential representations generated specific embeddings for downstream tasks (Kalchbrenner et al., 2014; Kim, 2014; Socher et al., 2013). Recently, work has focused on domain-

	Basketball	Blocks	Calendar	Housing	Publications	Recipes	Restaurants	Social
NOFILTER	0.358	0.294	0.617	0.461	0.511	0.570	0.626	0.355
ORACLE	0.558	0.494	0.817	0.661	0.711	0.770	0.826	0.555
AUTOENCODER	0.413	0.268	0.581	0.447	0.463	0.530	0.543	0.417
CONFIDENCE	0.389	0.306	0.644	0.472	0.525	0.568	0.665	0.360
CBOW	0.344	0.295	0.634	0.515	0.621	**0.575**	0.722	0.358
CNN	**0.452**	0.324	0.674	0.488	0.573	0.570	0.605	**0.446**
LSTM	0.385	0.314	0.622	0.495	0.581	0.547	0.612	0.363
SURPRISE	0.356	**0.371**	**0.679**	**0.570**	**0.668**	0.554	**0.764**	0.345

Table 4: Accuracy for a semantic parser evaluated on a test set in which 20% is domain adjacent.

independent embeddings, learned without downstream task supervision. Kiros et al. (2015), Hill et al. (2016), and Kenter et al. (2016) learn representations by predicting the surrounding sentences. Wieting et al. (2016) use paraphrases as supervision. Mu et al. (2017) represent sentences by the low-rank subspace spanned by the embeddings of the words in them; Arora et al. (2017) use a weighted average of word embeddings, with their projection onto the first principal component across all sentences in the corpus removed.

Another relatively sparse area of related work is handling the domain-adjacent instances once they have been identified. The simplest thing to do is to return a generic error. For user-facing applications, one such message might state that the system can't handle that specific query. Azaria et al. (2016) approach this problem by having the user break down the domain-adjacent instance into a sequence of simpler textual instructions and then attempting to map those to known logical forms.

6 Conclusion

Identifying domain-adjacent instances is a practical issue that can improve downstream semantic parsing precision, and thus provide a smoother and more reliable user experience. In this paper, we formalize this task, and introduce a novel sentence embedding approach which outperforms baselines. Future work includes exploring alternative ways of incorporating information outside of the given training set and experimenting with various combinations of semantic parsers and upstream domain-adjacency models. Another area of future research is how the underlying system should recover when domain-adjacent instances are detected.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *Proceedings of ICLR 2017*.

Fabrizio Angiulli and Clara Pizzuti. 2002. Fast outlier detection in high dimensional spaces. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 15–27. Springer.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of ICLR 2017*.

Amos Azaria, Jayant Krishnamurthy, and Tom M. Mitchell. 2016. Instructable intelligent personal agent. In *Proceedings of AAAI 2016*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of EMNLP 2013*.

Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874.

Dilek Hakkani-Tur, Yun-Cheng Ju, Geoff Zweig, and Gokhan Tur. 2015. Clustering novel intents in a conversational interaction system with semantic parsing. In *Proceedings of INTERSPEECH 2015*.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In *Proceedings of NAACL-HLT 2016*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of ACL 2014*.

Tom Kenter, Alexey Borisov, and Maarten de Rijke. 2016. Siamese CBOW: Optimizing word embeddings for sentence representations. In *Proceedings of ACL 2016*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of EMNLP 2014*.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2015. Skip-thought vectors. In *Proceedings of NIPS 2015*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of ICLR 2013*.

Jiaqi Mu, Suma Bhat, and Pramod Viswanath. 2017. Representing sentences as low-rank subspaces. In *Proceedings of ACL 2017*.

Seonghan Ryu, Seokhwan Kim, Junhwi Choi, Hwanjo Yu, and Gary Geunbae Lee. 2017. Neural sentence embedding using only in-domain sentences for out-of-domain sentence detection in dialog systems. *Pattern Recognition Letters*, 88(C):26–32.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP 2013*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of ACL 2015*.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Towards universal paraphrastic sentence embeddings. In *Proceedings of ICLR 2016*.