A Clustering Approach to Semantic Decoding

Hui Ye and Steve Young

Cambridge University Engineering Department
Trumpington Street, Cambridge, England, CB2 1PZ
hy216@eng.cam.ac.uk, sjy@eng.cam.ac.uk

Abstract

This paper presents a novel algorithm for semantic decoding in spoken language understanding systems. Unlike conventional semantic parsers which either use hand-crafted rules or statistical models trained from fully annotated data, the proposed approach uses an unsupervised sentence clustering technique called Y-clustering to automatically select a set of exemplar sentences from a training corpus. These exemplars are combined with simple sentence-level semantic annotations to form templates which are then used for semantic decoding. The performance of this approach was evaluated in the travel domain using the ATIS corpus. Training is fast and cheap, and the results are significantly better than those achieved using HMM-based or stack-based statistical parsers.

1. Introduction

Spoken dialogue systems require robust decoders to extract the required semantic information from the automatically recognised user inputs. Existing approaches to semantic decoding are either rule-based or statistical. Rule-based systems typically require hand-crafted rules which might then be augmented with corpus statistics (e.g. MIT's TINA [1], CMU's PHOENIX [2], and SRI's Gemini [3] systems). Whilst good performance is often achieved using this approach, rule-based parsers are normally expensive to build and hard to transplant from one application to another. Furthermore, they can still degrade badly in the face of high speech recognition error rates and unexpected or ill-formed input sentences.

In contrast, the statistical approach seeks to automatically train parsers from semantically annotated sentences in the hope of building more robust decoders with less effort. An early example is AT&T's finite state semantic tagger in which a HMM is used to assign semantic concepts to words [4]. More sophisticated models have been proposed since that can handle hierarchical structure such as the hierarchical Hidden Understanding Model [5], the hierarchical Hidden Markov Model [6] and the Hidden Vector State (HVS) model [7]. However, these models are complex and typically require every training utterance to be semantically annotated.

In some sense, both the rule-based and the statistical approaches mentioned above treat semantic decoding as a classical parsing problem. An alternative would be to treat the problem as a straightforward pattern recognition problem in which all sentences which share the same semantic annotation are deemed to be members of the same class. However, if sentences were classed according to their semantics, this would still require all training sentences to be semantically annotated. Our experience in building statistical decoders suggests that several thousand such sentences are needed to build a robust decoder and the provision of this data is time-consuming and often prohibitive.

This paper describes a simple and straightforward approach to robust semantic decoding which is fully automatic and which requires relatively few training sentences to be annotated, and only at the sentence level. Furthermore, the approach works surprisingly well, outperforming our best statistical parser on the ATIS corpus. The key idea is to first cluster sentences into classes without knowledge of their semantics and then assign a single semantic annotation to each class. To decode, the unknown input sentence is assigned the semantics associated with the class (or classes) to which it most closely matches.

The remainder of the paper is organised as follows. The sentence clustering algorithm, which we call Y-clustering, is described next in section 2 and a semantic decoder based on Y-clustering is described in section 3. Section 4 then presents experimental results using the ATIS corpus. Finally, section 5 presents our conclusions.

2. Y-Clustering

2.1. A General Schema for Sentence Clustering

Given a set of N training sentences $\{S_1, \ldots, S_N\}$, the Y-clustering algorithm seeks to group the sentences into Y classes and then select one sentence from each class to serve as an exemplar. Each such exemplar is called a *template*. The basic process is similar to the familiar K-means clustering algorithm which in the context of sentences can be described as follows:

1. (Initialisation): Randomly select Y different sentences as the initial templates $\{T_1, T_2, \cdots, T_Y\}$.
2. (Clustering): Assign each training sentence $\{S_i\}$ to the class m^* with the most similar template, i.e.

   $$m^* = \underset{m}{\text{argmax}} \; \{d(S_i, T_m)\} \quad \text{for } m = 1, \cdots, Y. \quad (1)$$

   where $d(S, T)$ is a distance measure between sentence S and template T.
3. (Template regeneration): For each class, select the sentence S^T which yields the highest within-class similarity $\mathcal{D}$ as the template T for that class:

   $$S^T = \underset{S_i}{\text{argmax}} \; \{\mathcal{D}(S_i)\} \quad \text{for} \quad i = 1, \cdots, H. \quad (2)$$

   where $\mathcal{D}(S)$ is the total similarity between sentence S and all members of the class, i.e.

   $$\mathcal{D}(S) = \sum_{h=1}^{H} d(S, S_h) \quad (3)$$

   and where H is the number of sentences assigned to that class.

4. Return to Step 2 until the termination criteria is satisfied. For example, similar to K-means, the process can be terminated when the newly regenerated templates are identical to those generated in the previous iteration.

The two key issues which arise in converting the above to a practical algorithm are the choice of distance metric and the method of controlling the generation of classes. These are dealt with next.

2.2. Similarity and Saliency

The similarity between two sentences will clearly depend on the words within them. To ensure that key informational bearing words such as content words are weighted more heavily than the less relevant words such as function words, every word is assigned a *saliency*. The saliency of a word will then represent how important the word is in distinguishing the current class from the other classes.

Assume that there are H sentences in a class, and α sentences in that class contain the word w. Additionally, assume that there are M classes, and among them β classes contain the word w. Then the saliency of w in that class is defined as

$$\mathcal{I}(w) = \sqrt{\frac{\alpha}{H} \times (1 - \frac{\beta}{M})}. \qquad (4)$$

This is a form of mutual information between the within class frequency and inter-class frequency, and it ranges from 0 to 1. A word with high saliency in a class will occur frequently in that class and it will be rare in all of the other classes.

Given a definition for the saliency of a word, a distance metric between a sentence S and a template T can be defined as follows. Firstly, a DTW algorithm is used to align S against T. When two identical words are aligned they receive a score equal to their saliency, and in all other cases the score is zero. Thus, no penalty is applied for mismatched words or for insertions and deletions and the overall alignment is chosen so as to maximise the total score. Given this DTW alignment, assume that there are L words in the sentence S, K words in the template T with saliency $\mathcal{I}_i$ for $i = 1, \cdots, K$, and J words, $w_1, w_2, \cdots, w_J$ of S were aligned to the template, then the similarity between the sentence S and the template T is given by:

$$d(S,T) = \sqrt{\frac{J}{L} \times \frac{\sum_{j=1}^{J} \mathcal{I}(w_j)}{\sum_{i=1}^{K} \mathcal{I}_i}}. \qquad (5)$$

Again, this is a mutual information style of definition which balances the degree of similarity as viewed from the perspective of the sentence (the first term in equation (5)) and the template (the second term in equation (5)). Its use avoids assigning high similarity to cases of unbalanced matching, e.g. where a long sentence matches a short template or vice versa, and it ensures that similarities always fall in the range 0 to 1.

2.3. Controlling the generation of classes

The Y-clustering algorithm is based on the K-means style clustering outlined in section 2.1 and the distance metric defined above in equation (5). However, conventional K-means clustering does not necessarily lead to the best sentence classes in terms of sentence similarity. In fact, if the number of classes is fixed in advance, then two undesirable consequences result. Firstly, sentences will be assigned to the nearest class during training regardless of their within class similarity i.e. the within class similarity is not controllable. Secondly, similar classes can be generated, i.e. the inter-class similarity is not controllable. This often happens when the number of classes to train is larger than the number of underlying classes. Hence it is necessary to perform sentence clustering in a more controlled way.

The problem can be simplified by viewing every sentence as a point in two dimensional space. The problem now becomes how to use a number of circles with the same radius p to cover all these points so that every point must be inside a circle. Additionally, two conditions must be met: 1) circles can only be located with the centre at an existing point; 2) the number of circles should be minimised to reduce overlapping. Returning now to the actual problem of sentence clustering, the circles are the sentence classes, the centre of the circle is the template, and the radius of the circle p is the maximal allowed distance within a class, i.e. the similarity threshold. The Y-clustering algorithm attempts to satisfy the above requirements by generating sentence classes in stages. The number of classes that can be generated at each stage is controlled by a similarity threshold p. In each stage, the algorithm evaluates the number of out-of-class (OOC) sentences/points to check that the coverage of the generated classes/circles is increasing, and it continues iteratively searching and merging similar classes until coverage starts decreasing. In the following stage, the process is repeated but just using the OOC sentences/points remaining from the preceding stage. This continues until all of the training data have been allocated to classes within the threshold p.

Since there are often some out-of-domain sentences in the training set, some of the generated classes will have very few members (very likely just one sentence in a class). Hence on completion of the Y-clustering algorithm, all generated templates are collected to form a single set and all of the training sentences are re-assigned to the nearest template. The number of sentences in each class is then used to rank the templates. Finally, the desired number of templates Y can be selected as the top Y entries in the ranked list, thereby pruning all of the out-of-domain classes at the bottom of the list.

2.4. The Y-clustering Algorithm

The Y-clustering algorithm is as follows:

1. Assign a value to a similarity threshold p ranging from 0 to 1 (e.g. $p = 0.7$), and set the maximum number of templates to generate in each pass M (e.g. $M = 500$).

2. Tag all training sentences as OOC and select the first M distinct sentences in the corpus to form the initial set of templates. Set all word saliencies to 1.

3. For each training sentence S, select each template T in turn and compute the similarity $d(S,T)$ using equation (5). If the similarity is higher than p, assign S to template class T and go to the next sentence; otherwise if the similarity is less than p for all templates, assign the sentence to the class with highest similarity score and mark the sentence as OOC.

4. After assigning all sentences, sort the templates into order based on the number of sentences in each template class, then delete the empty classes. This improves efficiency in subsequent iterations and encourages merging of similar classes by ensuring that templates with the most members are always examined first.

5. If the number of OOC sentences is fewer than in the previous iteration, then update the word saliencies using equation (4), regenerate the templates as in step 3 of section 2.1, and return to Step 3.

6. Store the set of templates generated in the previous steps, and set aside all of the sentences which are not marked as OOC. Then repeat from step 2 until all of the data has been used.

7. Finally, collect together all of the generated templates and assign each training sentence to the nearest template. Rank the templates according to the number of assigned sentences. If there are more than the required Y templates in the ranked list, discard all but the top Y entries.

This algorithm guarantees that the sentence similarities in every classes are above the similarity threshold. The number of final classes in the ranked list depends on the threshold p and the distribution in the training corpus.

3. Semantic decoding using Y-clustering

The target application area for the type of semantic decoding being described in this paper is limited domain spoken dialog systems. These systems are typically concerned with so-called slot-filling dialogues where the application is characterised by a set of lexical classes (i.e. slots) such as `city_name`, `day`, `ticket_class`, etc. and extracting the semantics of an utterance primarily involves identifying the slot names and their values[1]. Thus, for example, the sentence

```
    Show me flights from London to Paris
```

would have semantics defined by two slot value pairs

```
Slots/Values: FROMLOC.CITY = London
              TOLOC.CITY = Paris
```

To use the Y-clustering approach for semantic decoding, the following steps are performed. Firstly before clustering, each sentence in the corpus is pre-processed such that any lexical class members are replaced by their class names. The sentences are then clustered to generate a set of templates. Each template then has the associated slot/values attached where the values reference the position of the corresponding word in the utterance. Thus, if the above sentence was a template, it would be stored as:

```
Show me flights from city_name to city_name
Slots/Values: FROMLOC.CITY = T(5)
              TOLOC.CITY = T(7)
```

Decoding a sentence then consists of matching it against every template using the same DTW alignment procedure used in training. For every template whose similarity lies above a threshold, the corresponding slot/value pairs are instantiated along with the similarity score itself. Thus, more than one template can contribute to the set of slot/value pairs extracted from an utterance. This allows new forms of sentence to be processed by partially matching against different templates. Where there is competition, the rule used is that each word in the sentence can only appear in one slot/value pair and if a word matches more than one slot, then the match with the highest similarity score is selected. However, the same slot can be used multiple times when it is bound to different words.

[1]The intended dialog act also needs to be determined but this is not considered here.

4. Experiments

4.1. Experimental Setup

This section describes the experimental evaluation of the Y-clustering algorithm using the ATIS-3 NOV93 and DEC94 test sets. The training set has 4978 sentences in total which are selected from the Class A training data in the ATIS-2 and ATIS-3 corpora. The same experimental setup as in [7] was applied so that the Y-clustering approach can be compared fairly with the FST and HVS models presented there.

As described in the previous section, the training sentences were first preprocessed to replace lexical class members by their corresponding lexical class names. There are a total of 30 such domain specific lexical classes defined for the ATIS domain. Then after Y-clustering, all the resulting templates were annotated with the corresponding slot value pairs. These were derived from the same semantic frame structures used in [7] [2]. There were in total 47 distinct semantic slots.

To parse a test sentence, it was first preprocessed to apply the same lexical class substitution as used in training, then aligned to each of the templates to compute a similarity score.

The following example shows the matching of a typical ATIS sentence.

```
O:show flights from burbank to milwaukee for today
S:show flights from city_name to city_name for time
T:show all flight_stop flights from city_name to city_name
A:S(1) S(0)  S(0)       S(2)   S(3)  S(4)    S(5) S(6)
Similarity: 0.778
Template Slots:    FLIGHT_STOP = T(3)
                   FROMLOC.CITY_NAME = T(6)
                   TOLOC.CITY_NAME =T(8)
Nominated Slots/Values:
 FLIGHT_STOP=T(3)=S(0)=NULL (Discarded)
 FROMLOC.CITY=T(6)=S(4)=O(4)=burbank (Confidence 0.778)
 TOLOC.CITY=T(8)=S(6)=O(6)=milwaukee (Confidence 0.778)
```

where O is the original test sentence, S is the test sentence after lexical class substitution, T is the matched template, T(n), S(n) and O(n) means the n-th word in the template, the preprocessed sentence and the original sentence respectively, and A denotes the word alignment. For example, in the above case, the first word of the template was aligned to the first word of the test sentence as denoted by S(1), and the second word of the template has no counterpart in the test sentence as denoted by S(0).

Finally, the slots/values for all matches scoring more than 0.1 were pooled and ranked as described in the previous section to give the final output.

4.2. Results

The Y-clustering approach was tested using the NOV93 and DEC94 ATIS-3 test sets which contain 893 sentences in total. Each sentence was annotated using the same frame structure as that applied to the training set. This annotation was used as the reference. The extracted semantics of the test sentences were represented by the slot/value pairs output by the Y-clustering algorithm and these were compared with the slot/value pairs from the reference. An F-measure computed from the precision (P) and recall (R) of slot/value pairs was used to evaluate the overall decoding performance. It should be noted that a strict comparison was adopted in the experiments where the match of slot/value pairs between the reference and the decoder output is only marked correct if both the slot name and slot value are matched. This is the same strict criterion as used in [7].

[2]These semantic frame structures were automatically derived from the SQL database queries supplied with the ATIS corpus.

Table 1: *Performance comparison of Y-clustering, FST and HVS models on ATIS.*

	Y-clustering				FST	HVS
p	0.6	0.7	0.8	0.9	/	/
#templates	657	1105	1827	2821	/	/
Recall	93.86%	94.46%	95.39%	95.32%	86.71%	89.82%
Precision	87.61%	87.36%	88.22%	87.15%	84.84%	88.75%
F-measure	90.63%	90.76%	91.66%	91.05%	85.77%	89.28%

Table 1 shows the F-measure scores obtained by the Y-clustering decoder for various values of similarity threshold ‘p’ when the full list of generated templates is used (i.e. Y is the maximum possible in each case). For comparison, the performance of the FST and HVS models reported in [7] are also shown. It can be observed that the performance of the Y-clustering approach is competitive whilst requiring only a relatively small set of templates to be annotated (e.g. 657 for the case of $p = 0.6$). The FST and HVS models on the other hand required the full training set to be annotated i.e. nearly 5000 sentences. Note that increasing the similarity threshold resulted in more templates, however, the corresponding increase in performance is modest. This suggests that many of the additional templates generated are either redundant or correspond to out of domain sentences.

As explained in section 2.3, an alternative way to control the number of templates generated is to prune the final ranked list. This in principle should remove outlier templates and focus the decoding more on the in-domain templates. Table 2 presents the results obtained for the $p = 0.8$ case when the generated template list is pruned back to various Y values. As can be seen, this pruning results in a significant increase in performance. When the top 600 templates were selected, the recall and precision values were balanced and the F-measure reached a maximum at 93.42%. Comparing this result with the case for $p = 0.6$ in Table 1, it can be seen that for a similar number of templates, setting a higher threshold and then pruning works much better. Fig 2 also shows that even when as few as 400 templates were used, the 92.89% F-measure was still significantly better than the results obtained by the statistical parsers even though less than 10% of the training data had to be semantically annotated. Finally, consistent with Table 1, Table 2 shows that generating more templates does not necessarily lead to better performance, since the introduction of redundant templates leads to poorer precision.

Table 2: *Performance of template selection. (p=0.8)*

	Y-clustering (p=0.8)					
#templates	200	400	600	800	1000	1827
Recall	88.47%	93.12%	93.75%	94.50%	94.85%	95.39%
Precision	92.06%	92.66%	93.09%	90.06%	90.33%	88.22%
F-measure	90.23%	92.89%	93.42%	92.23%	92.54%	91.66%

The above experimental results show that the Y-clustering algorithm can produce competitive performance at reduced cost compared to existing statistical semantic decoders. Furthermore, for real applications, Y-clustering is extremely flexible. For example, the templates can be manually defined for a new application and only the word saliencies updated using Y-clustering. This might be an effective way of bootstrapping a system, when initially little training data is available.

Another advantage of Y-clustering is that disfluent and ill-formed sentences as well as speech recognition errors do not significantly degrade performance, as long as the key words are intact. For instance, in the following example, the user first wanted to ask for “the earliest” flights then changed her mind to ask for the “cheapest” flights. The incomplete word then resulted in recognition errors in both O and S. However, since the key words were correctly recognised, the decoder was still able to fully recover the semantics without error.

```
U:show the earl- cheapest flights from london to boston
  can you
O:should me flights from london to boston can you
S:should me flights from city_name to city_name can you
T:flights from city_name to city_name
Similarity: 0.745
```

Finally, note that in the experiments reported here, the Y-clustering algorithm was only used for semantic decoding. However, there seems to be no reason why templates could not also be tagged with the appropriate dialog act to allow dialog act detection to be performed simultaneously.

5. Conclusion

This paper has presented the Y-clustering algorithm for semantic decoding. Unlike conventional semantic parsers which either use hand-crafted rules or statistical models trained from fully annotated data, the proposed approach uses an unsupervised sentence clustering technique to automatically select a set of exemplar sentences from a training corpus. These exemplars are combined with simple sentence-level semantic annotations to form templates which are then used for semantic decoding. The performance of this approach was evaluated in the travel domain using the ATIS corpus, and compared with two previously reported statistical parsers. The results obtained from Y-clustering in terms of F-measure are significantly better and furthermore, they are achieved with only 10% of the training sentences requiring annotation.

These results suggest that the Y-clustering approach has considerable potential. The DTW-based matching process is intrinsically robust to variations in the input such as those caused by recognition errors, disfluencies and ill-formed sentences. Furthermore it is simple and flexible. For example, it should be straightforward to extend the method to provide dialogue act decoding. The method is now being incorporated into a working spoken dialogue system and evaluation of its efficiency and robustness in other applications will be reported in future work.

6. References

[1] Seneff, S., “Robust parsing for spoken language systems”, ICASSP 1992.

[2] Ward, W., Issar, S., “Recent improvements in the CMU spoken language understanding system”, In Proc. of the ARPA Human Language Technology Workshop 1996.

[3] Dowding, J., Moore, R., Andry, F., Moran, D., “Interleaving syntax and semantics in an efficient bottom-up parser”, In Proc. of the 32nd Annual Meeting of the Association for Computational Linguistics 1994.

[4] Pieraccini, R., Tzoukermann, E., Gorelov, Z., Levin, E., Lee, C., Gauvain, J., “Progress report on the CHRONUS system: ATIS benchmark results”, In Proc. of the DARPA Speech and Natural Language Workshop, 1992.

[5] Schwartz, R., Miller, S., Stallard, D., Makhoul, J., “Hidden understanding models for statistical sentence understanding”, In Proc. of ICASSP 1997.

[6] Fine, S., Singer, Y., Tishby, N., “The hierarchical hidden markov model: Analysis and applications”, Machine Learning 1998.

[7] He, Y., Young, S., “Semantic processing using the hidden vector state model”, Computer Speech and Language 19(1): 85-106, 2005.