Layered Dialog and Word Clustering

Tynan Smith

1. The Problem

The Restaurant Game is part of a project to develop an AI system that can play a video game with a human or another AI just by using annotated recordings of humans playing the game as examples. The Restaurant Game is a simple two-player restaurant simulation in which character are instructed to act out a typical interaction between a customer and a waitress. We have collected about 10,000 recordings of humans playing the games, which consist of the typed dialog and events they perform within the game.

The goal is for the AI to interpret the game it is currently experiencing in terms of the recorded, annotated games and choose actions appropriately. In order to accomplish this, we are seeking to annotate the game logs with additional information. We want to cluster

and annotate words, phrases and dialog lines. Furthermore, we want to group events and dialog (that may have other events interleaved) into sequences that correspond to higher-level actions. For example, the waitress asking the bartender for beer and bringing it to the customer and putting it down may all be grouped into the sequence “waitress serves drink.”

As part of my Master’s thesis I am exploring ways to automatically find and cluster sequences, but one of the challenges is dealing with the variety of dialog encountered. Therefore, I also want to cluster dialog lines, words and phrases in order to decrease the variation to make find and clustering sequences more tractable. This project is an exploration of one possible method of clustering dialog lines and words (and hopefully later extend to sequences as well) simultaneously in interacting layers.

2. Proposed Solution

a. Simultaneous Layered Clustering

Since the goals is to cluster words, dialog and sequences and all three are related because the contain or are contained by one another I proposed to cluster all three sets simultaneously, using the current clustering of each layer to influences the future clustering of other layers at each iteration. The motivation being that if two dialog lines mean the same thing, but one uses several synonyms in place of the words contained in the other then it can be hard to automatically determine that they are similar, unless they appear in similar contexts. But by clustering the words they contain, and comparing them in terms of those clusters, it is more apparent. Similarly, if two similar words are used in different phrases that mean the same thing, clustering those phrases can help to make the similarity of the words more apparent. Each layer’s similarity is influenced by the similarities of the layers above and below it, which can be represented by the clustering of those layers.

i. Layers

For this exploration I sought only to focus on the clustering of words and dialog lines and used the following layer: letters, words, dialog lines and sequences. I kept the letters and dialog lines fixed, and only clustered on the middle two layers, but the outer layers’ fixed clustering was factored into similarity measurements.

ii. Similarity Measurements

Within each layer I computed the similarity between each pair of elements in the layer as a weighted sum of the following three similarity components, normalized to be between 0 and 1. This algorithm is designed to work with soft cluster assignments so the current similarity of elements in all layers is estimated as the current cluster overlap of those elements, that is the dot-product of their normalized cluster assignments.

1. **Sequence Similarity (base on lower layer)**

Sequence similarity is a measure of how similar the sequences of elements from the layer below that compose these two elements are. It is computed as the levenshtein distance between the two sequences (the number of insertions, deletes and substitutions needed to get from one to the other), where substitution is weighted by the cluster overlap of the two lower-layer elements being swapped.

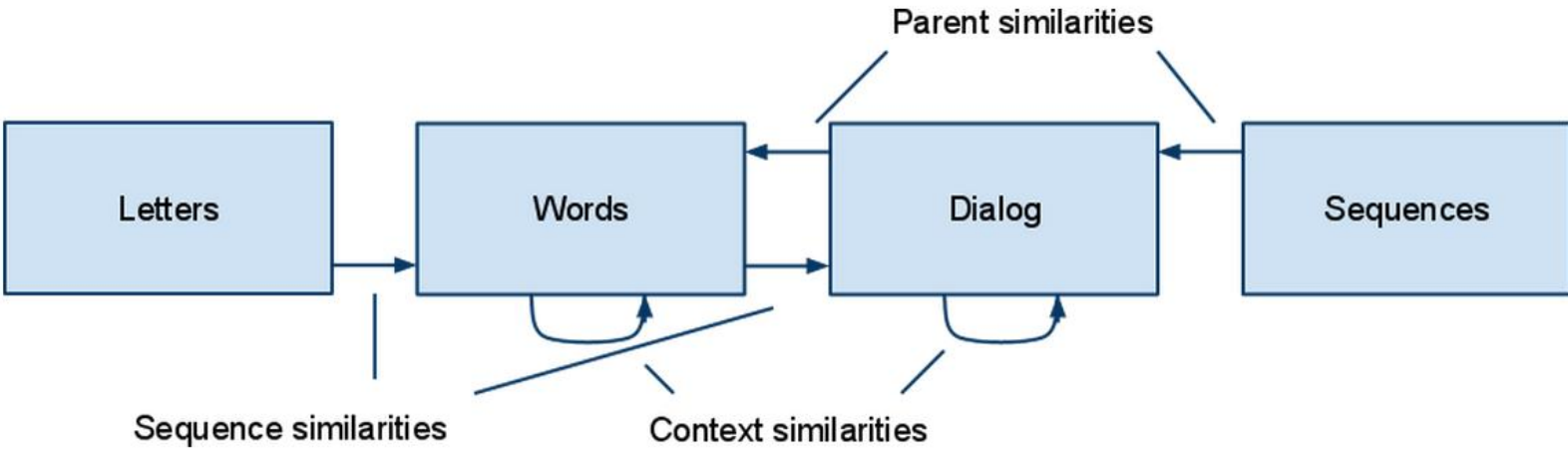
2. **Context Similarity (based on same layer)**

Context similarity is a measure of how similar the context the elements appear in is. It is based on the cluster overlap of siblings, elements that appear nearby a given element in elements of the higher layer. For example, in the line “there are two dogs,” if we are looking at neighbors only one space away, “two” and “there” are both neighbors of “are.” First n-gram distributions are computed for each element by looking at all the siblings that occur within a given range of the element in any of its parents and combing their clusters according to where they occur in relation to it. This gives a vector of cluster distributions corresponding to positions near the elements, these are dotted together after being normalized to produce context similarity.

3. **Parent Similarity (based on higher layer)**

Parent similarity is a measure of how similar the elements containing the particular elements are. Once again we add and normalize the cluster distributions and then compute the overlap, but we only compute using the parents that don’t contain both of the elements because that would be more a measure of how much the words co-occurr rather than how similar their contexts are.

All these similarities are combined into a single similarity score and after the similarities have been computed for all pairs they are fed into a clustering algorithm that can use pair-wise similarities instead of a feature space and then the cluster distributions are updated and the process is repeated.



3. Implementation

a. Feature Extraction

The data set I used was 100 game logs annotated with sequence information and dialog information. I used the sequence labels to create the sequence elements and as fixed values for the cluster distributions in the sequence layer. The dialog labels were used as the gold standard for dialog lines. I extracted the sequences and gold standard clusters using a Perl script. This resulted in about 5142 sequences (2106 unique) in 28 clusters, 4280 dialog lines (2944 unique) in 309 clusters, and 15935 words (1862 unique). For most of the time I was working on tuning the algorithm's parameters so I worked with a smaller set of just 10 games which resulted in 519 sequences (242 unique) in 27 clusters, 597 dialog lines (452 unique) in 115 clusters and 2328 words (543 unique).

b. Algorithm

The extracted data was fed my C++ algorithm and run with varying parameters the pseudo-code is as follows:

```
layers <- LoadData

for i from 1 to NUM_ITERS
     similiarities <- layers.computeSims()
     cluster_distributions <- cluster(similarities)
     layers.cds <- prune(cluster_distributions)

layers.output_final_clusters()
```

The algorithm was very computationally intensive and I spent a lot of time optimizing it. One thing I experimented with was pruning the similarities that were very small so that the clustering algorithm wouldn't consider those pairs. Another thing I experimented with and

left in was pruning the cluster distributions which I left in as seen above so that for each element I only included the clusters that had an above average distribution. This greatly improved computation time since each of the similarities required dot producting cluster distributions. I had originally wanted to test multiple internal clustering algorithms, but had issues with spectral clustering and k-modes but would still like to do it in the future. Instead the clustering algorithm I used was affinity propagation, which does not require the number of clusters to be specified ahead of time and which only needs pairs of similarities.

c. Parameters

There were many parameters in my algorithm that I spent a lot of time tuning described in more detail below, though I would like to evaluate them further on larger data-sets. I would like to explore all these more, especially tuning each one separately for each layer.

i. **N-gram size**

The distance in each direction around each element to look for siblings when computing context similarity. Best value so far has been 2.

ii. **Cluster threshold**

The percentage of the average cluster distribution a cluster must exceed to be kept in an elements cluster distribution. Best value 1.0.

iii. **Lambda**

The damping factor used in affinity propagation, controls how quickly it converges. Best value 0.9.

iv. **Number of iterations**

The number of times to perform the similarity computation - clustering cycle. Best value 3. (I would like to try updating similarities and clustering simultaneously in the future)

v. **Preference**

The self-similarity of the elements, a parameter for affinity propagation that influences how likely the element is to be a cluster exemplar. I use the average of all the other similarities.

vi. **Similarity weights**

The relative weights of each of the similarity components, I did not experiment wiht this nearly as much as I would have liked. Best so far: 0.2 Sequence similarity, 0.6 Context similarity, 0.2 Parent similarity.

4. Experimentation, Testing and Evaluation

a. Evaluation of Performance

¡. Naive quantitative score

I did not have time to finish coming up with gold standard clusters for words so I started by only evaluating the dialog clusters.

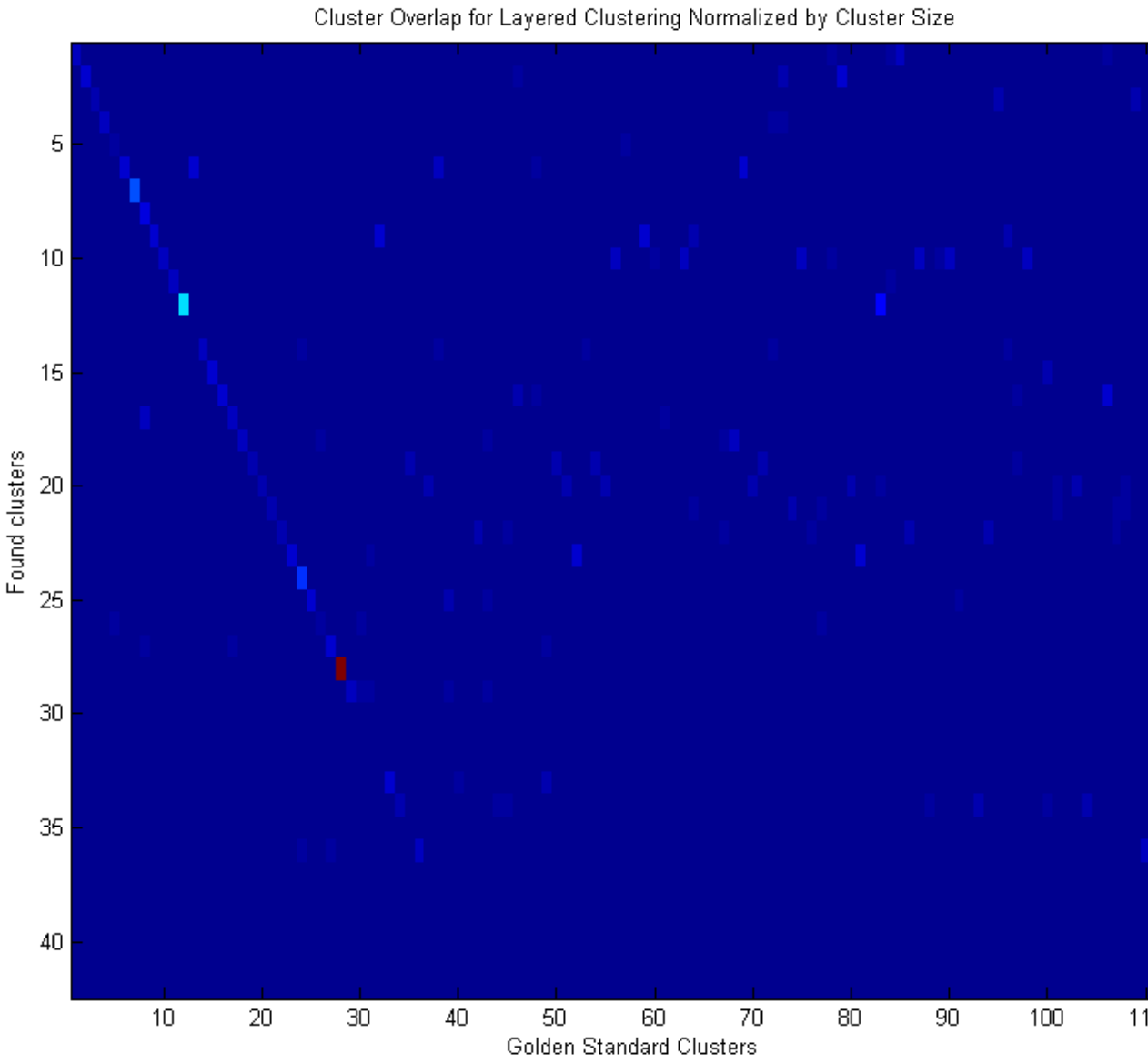
I wanted to try to quantitatively evaluate the clustering method and compare it to using just affinity propagation by itself. But the difference in the number of clusters and the sparsity of the data made it difficult to evaluate. The naive method I came up with was to sum up over all the pairs of elements the pairs that were int the same cluster in both clusterings and those that were in different clusterings in both (the one being tested and the gold standard) and divide by the total number of pairs. I found however, that although my method scored slightly better than pure affinity propagation, putting each element in its own cluster so I discarded this method of evaluation.

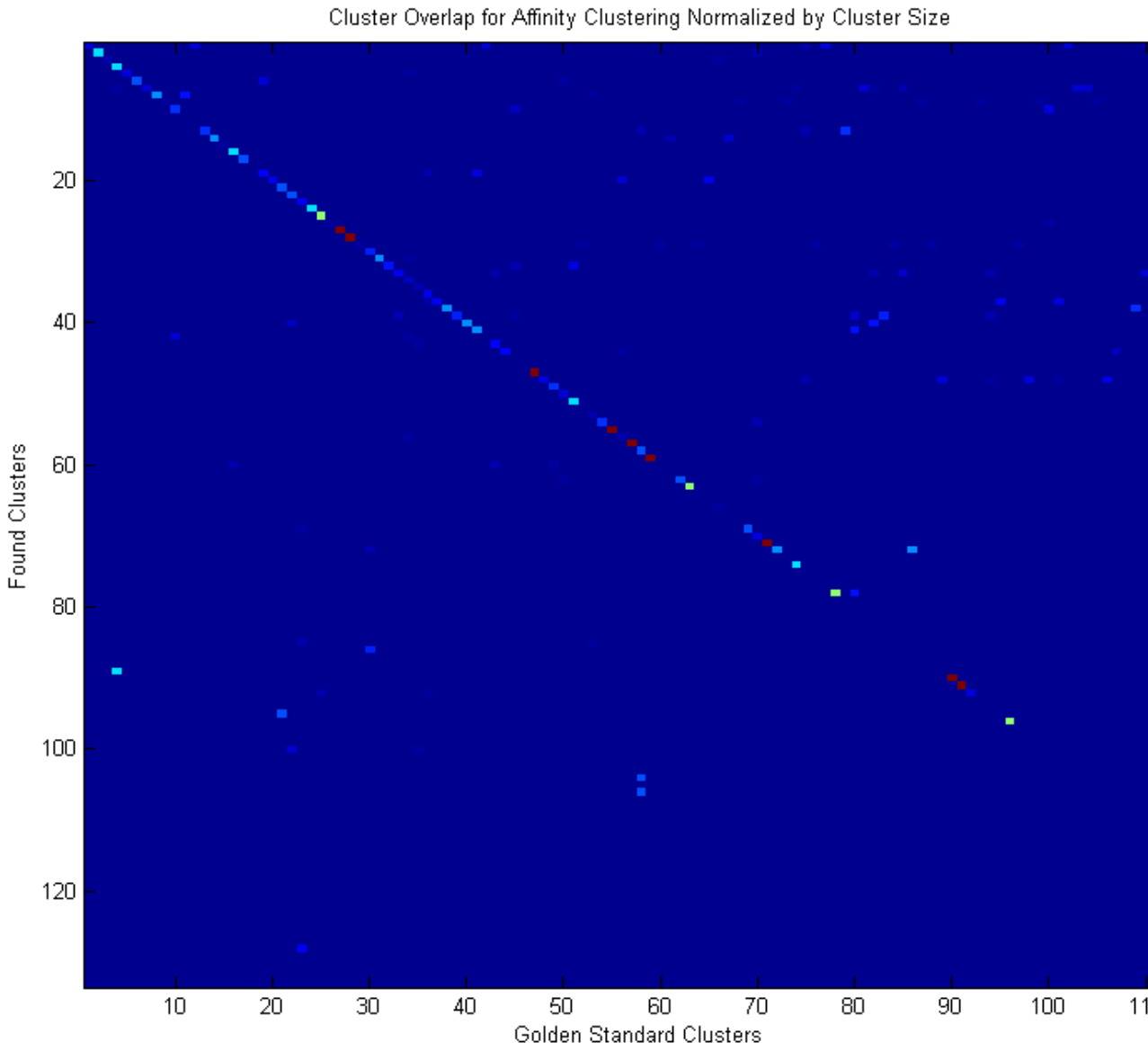
¡¡. Qualitative evaluation

Instead of a quantitative measurement I came up with a qualitative way of evaluating the clusterings. I computed the cluster overlap matrices between the found clusters and the gold standard clusters, normalized by the size of the clusters and used a matlab script to sort the rows and columns to maximize the weight along the diagonal and plotted it as a colormap. The more weight along the diagonal the better the found clusters because it means there is more correspondence between the found clusters and the gold standard clusters.

5. Results

I would have liked to evaluate my algorithm on the full dataset, but as of yet it has not completed because it is very computationally intense. However, below are the qualitative results for pure affinity propagation and my layered clustering algorithm.


Cluster Overlap for Layered Clustering Normalized by Cluster Size
Found clusters
5
10
15
20
25
30
35
40
10
20
30
40
50
60
70
80
90
100
Golden Standard Clusters



6. Conclusions

So far it is still too early to decide for sure whether this method is worthwhile or not, because I have not yet found a good method for evaluating and it needs to be run on larger datasets. I may try to find another data-set to use as a proof of concept that would be larger, but less sparse and less computationally intensive.

7. Future Work

I would like to evaluate the method further, using more data, and better evaluation methods. I would like to experiment with the parameters more and try tuning them on a per-

level basis. I would like to experiment with adding another level. I would like to try updating the similarities in real-time while the clustering algorithm is running so the two influence each other simultaneously instead of iteratively.