Chat Disentanglement: Identifying Semantic Reply Relationships with Random Forests and Recurrent Neural Networks

Shikib Mehri
Department of Computer Science
University of British Columbia
Vancouver, Canada
mehrishikib@gmail.com

Giuseppe Carenini
Department of Computer Science
University of British Columbia
Vancouver, Canada
carenini@cs.ubc.ca

Abstract

Thread disentanglement is a precursor to any high-level analysis of multiparticipant chats. Existing research approaches the problem by calculating the likelihood of two messages belonging in the same thread. Our approach leverages a newly annotated dataset to identify reply relationships. Furthermore, we explore the usage of an RNN, along with large quantities of unlabeled data, to learn semantic relationships between messages. Our proposed pipeline, which utilizes a reply classifier and an RNN to generate a set of disentangled threads, is novel and performs well against previous work.

1 Introduction

The problem of thread disentanglement is a precursor to high-level analysis of multiparticipant chats (Carenini et al., 2011). A typical chat consists of multiple simultaneous and distinct conversations, with Elsner and Charniak (2010) observing an average of 2.75 simultaneous threads of dialogue. Since a conversation does not necessarily entail a contiguous sequence of messages, the interwoven threads must be identified and segmented prior to any high-level analysis of the chat.

To further illustrate the need for thread disentanglement, consider the chat log in *Figure 1*. It should be clear to a reader that there are two independent threads of dialogue occurring within this sequence of messages, the first between John, Jack and Brian and the second between Jenny and Katie. Humans are adept at mentally disentangling conversations and even go as far as to adjust their behavior in order to ease the process of disentanglement which O'Neill and Martin (2003) observed in the form of name mentioning.

John:	i need a new tv show to watch
Jack:	psych/house of cards/breaking bad sound like things you might enjoy
Brian:	oh I should probably renew my Netflix
Katie:	I forgot my laprop at home D:
Jenny:	Katie, that sucks...
Jenny:	Are you going to go back home to get it?
Katie:	laptop*
Brian:	try Black Mirror

Figure 1. An example of a multiparticipant chat with two threads of dialogue.

Our work is novel because it approaches the problem of thread disentanglement by attempting to predict immediate reply relationships between messages. The potential benefits of this idea were discussed by Elsner and Charniak (2010) and Uthus and Aha (2013), but the idea has not yet been explored. Furthermore, Elsner and Charniak (2010) suggest that this approach "might yield more reliable annotations".

Additionally, we explore the usage of unlabeled data for the purpose of identifying semantic relationships between messages. Previous attempts at semantic modeling by Elsner and Charniak (2010) and Adams and Martel (2010) have not been very effective, however recent accomplishments in next utterance prediction (Lowe et al., 2015) can be leveraged for the purpose of thread disentanglement.

The main contributions of this paper are as follows.

1. We create an annotated dataset[1] which labels direct reply relationships between pairs

[1] The dataset is publicly available and can be found at http://shikib.com/td_annotations.

Proceedings of the The 8th International Joint Conference on Natural Language Processing, pages 615–623,
Taipei, Taiwan, November 27 – December 1, 2017

of messages in a transcript.

2. We create and open-source a tool[2] for the efficient reply annotation of a dataset.

3. We propose a pipeline for the task of thread disentanglement, consisting of:

   (a) A classifier, trained on the aforementioned dataset, that predicts reply relationships,

   (b) A recurrent neural network that models semantic relationships between messages,

   (c) A thread partitioning algorithm that utilizes a variety of features, including the previous stages of the pipeline, to ultimately partition a transcript into threads.

4. We evaluate our algorithm against all comparable previous approaches and explore potential improvements to the pipeline.

As a preview of the paper, *Section 2* further describes related work. *Section 3* introduces the proposed pipeline, with the subsections detailing the distinct stages of the pipeline. *Section 4* presents the metrics used for evaluating the agreement between a pair of disentanglements. *Section 5* discusses the datasets used by the pipeline, with specific attention to our newly annotated dataset. *Section 6* describes our experiments and presents the results. Finally, *Section 7* discusses our results and suggests potential future improvements upon our work.

2 Related Work

There have been a number of approaches to thread disentanglement, the majority of which contain a clustering/partitioning algorithm using a measure of message relatedness to segment a chat transcript into distinct threads.

Shen et al. (2006) introduce the problem of thread disentanglement, and approach it by using the cosine-similarity of messages to compute the distance between a message and a thread.

Elsner and Charniak (2010) present comprehensive metrics for evaluation along with a corpus to aid with disentanglement. They train a classifier on their corpus, to predict whether two messages belong in the same thread.

[2] The open-sourced annotation interface can be found at https://github.com/Shikib/react-chat-reply-annotation.

Wang and Oard (2003) construct expanded messages using temporal, author and conversational context. By expanding messages using this contextual information, they have more signal to use when assigning a message to a thread.

We build on this work by using our newly annotated dataset to train a classifier which predicts immediate reply relationships. This is a supervised alternative to heuristics used by previous research. Additionally, it is an improvement over the classifier trained by Elsner and Charniak (2010) since the nature of the annotation leads to stronger relationships between message pairs in the training data.

Previous work has explored modeling semantic relationships between messages using a predefined list of technical words (Elsner and Charniak, 2010) and applying Latent Dirichlet Allocation (Adams and Martel, 2010). In contrast, we apply the research done by Lowe et al. (2015) by utilizing a Recurrent Neural Network to predict the probability of a message occurring in a given thread.

3 Proposed Pipeline

We propose a novel pipeline to approach the problem of thread disentanglement. This pipeline consists of four stages, as visualized in *Figure 2*.

The first stage, as described in *Section 3.1*, is a classifier to detect reply relationships between pairs of messages.

The second stage is a classifier that predicts whether two messages belong in the same thread. This classifier, described in *Section 3.2*, is similar to the one trained by Elsner and Charniak (2010).

The third stage, described in *Section 3.3*, is a recurrent neural network that uses the content of the messages to predict the probability of a message following a sequence of messages.

The fourth and final stage is a thread partitioning algorithm that uses the information outputted from the previous stages to generate threads. This stage is described in *Section 3.4*.

3.1 Reply Classifier

Given two input messages, the reply classifier outputs the likelihood of the first message being a reply to the second. Given a child and a parent message, a feature vector is generated in order to describe the relationship between the two messages. The features utilizes are described in *Table 1*.

Table 1: Description of features utilized for the reply classifier.

Time	The time difference in seconds.
Mention Parent	Whether the child message mentions the author of the parent message.
Mention Child	Whether the parent message mentions the author of the child message.
Same author	Whether the author of the two messages is the same.
Distance	The number of messages separating the two messages.
RNN Output	The probability outputted by the RNN.

These feature vectors are then used to train a random forest (Breiman, 2001) classifier with 250 trees. We performed comparisons to other classifiers, namely a logistic regression classifier, a support vector classifier and a multilayer perceptron classifier, however cross-validation proved random forests to have the highest accuracy. The models were trained and evaluated using scikit-learn (Pedregosa et al., 2011).

The pairs of input messages in the training data were specifically annotated as consisting of a reply relationship. This suggests that each pair of messages is directly related, leading to the classifier learning to identify strong, immediate relationships between messages.

3.2 Same-Thread Classification

Elsner and Charniak (2010) trained a classifier that predicted whether two input messages belonged to the same thread. We implemented a similar classifier through utilization of the features described in *Section 3.1* and a dataset that they provided.

Given a pair of messages, the same-thread classifier outputs the probability of the messages belonging to the same thread. The classifier trained was a random forest (Breiman, 2001) with 250 trees, using scikit-learn (Pedregosa et al., 2011).

Unlike the reply classifier, the same-thread classifier was trained on data with weak relationships. Instead of specifically annotated relationships, the pairs of messages used to train the same-thread classifier were labeled as belonging to the same thread. Because the same-thread pairs are a superset of the reply pairs, this leads to the classifier learning to identify broader relationships between messages, rather than strictly immediate reply relationships.

As a result of the same-thread classifier being trained on a different set of data than the reply classifier, the learned relationships are different in nature, which suggests that the two classifiers can strongly complement each other.

3.3 RNN for Next Utterance Classification

The third stage of the pipeline attempts to leverage large amounts of unlabeled data for the purpose of modeling semantic relationships between messages. We train a recurrent neural network with LSTM (Hochreiter and Schmidhuber, 1997) hidden units to predict the probability of a message following a sequence of messages (Lowe et al., 2015).

The reasoning behind utilizing an LSTM is for the purposes of identifying dependencies between non-adjacent messages. LSTM units are best able to capture long-term dependencies through the use of a series of gates which control whether an input is remembered, forgotten or used as output. Formally, at every time step an LSTM unit updates the internal state C_t as a function of the observed variable x_t and the previous internal state C_{t-1} and h_{t-1}.

Both the context (the previous sequence of messages) and the message are passed through the LSTM units one word at a time, in the form of learned word embeddings. Let us use c and r to denote the final hidden state representations of the context and reply respectively. We can use these hidden states, along with a learned matrix M, to compute the probability of a reply, as:

$$P(reply \mid context) = \sigma(c^T M r) \qquad (1)$$

The model, implemented in PyTorch (A. Paszke, 2017), was trained using hyperparameters recommended by Lowe et al. (2015).

This RNN-based next utterance classification is useful as it supplements the aforementioned classifiers by leveraging unlabeled data to semantically model message relationships.

Additionally, the output of this classifier is added as a feature to the previous two classifiers as well. This allows the reply classifier and the same-thread classifier to use the semantic relatedness of the input messages during classification.

Table 2: Description of the features utilized by the in-thread classifier.

Same-Thread Mean	Mean output of the same-thread classifier.
Same-Thread STD	Standard deviation of the output of the same-thread classifier.
Reply Mean	Mean output of the reply classifier.
Reply STD	Standard deviation of the output of the reply classifier.
Thread Length	Current length of the thread.
Author count	Number of author's messages already in the thread.
Author total	Number of author's messages in the chat.
In-thread proportion	Author count divided by Author total.
Author proportion	Author count divided by Thread Length.
Author mentions	Number of times the author's name was mentioned in the thread.
Time	The time difference between the message and the last message in the thread.
RNN Prediction	The prediction outputted by the RNN.

3.4 Thread Partitioning

Our ultimate goal is to generate a set of disentangled threads, suggesting the usage of a thread partitioning algorithm as the next stage in the pipeline. Using the previously described classifiers, we must identify an optimal segmentation of an input transcript.

3.4.1 In-Thread Classifier

We construct a classifier, referred to as the in-thread classifier, which leverages the output of all previous stages of the pipeline to predict the probability of a message belonging to a thread.

Given a message and a thread, we generate a feature vector using the features described in *Table 2*.

The model used for classification, built in scikit-learn (Pedregosa et al., 2011), is a random forest classifier with 300 trees.

3.4.2 Thread Partitioning

Given the in-thread classifier, we can compute the probability that a message belongs in a given thread of conversation. This is used by our thread partitioning algorithm to generate threads.

This algorithm processes the messages in chronological order. For every message, m_i, the algorithm considers every existing thread, t_j, and passed m_i and t_j into the in-thread classifier.

Ultimately m_i is assigned to the thread which maximizes the probability outputted by the classifier, provided that the best probability output is above a threshold. If the best probability is below the threshold, a new thread is created for the message. This threshold was fine-tuned on the validation dataset.

After sequentially iterating over all of the messages in the transcript, the thread partitioning algorithm will output the disentangled threads.

3.5 Pipeline Overview

Figure 2 provides an overview of the proposed pipeline. The thread partitioning algorithm generates a feature vector to represent the relationship between a thread and a message. The features, as listed in *Table 2*, include the values outputted by the previous stages of the pipeline.

The in-thread classifier, trained on the pilot dataset provided by Elsner and Charniak (2010), uses this generated feature vector to predict the probability of the message belonging to the given thread. Using the outputted probabilities, the thread partitioning algorithm either assigns the message to the most probable existing thread or generates a new thread.

4 Metrics

A number of metrics are used throughout this paper to evaluate the agreement of two disentanglements. It is a non-trivial task to compare two disentanglements which have a different number of threads.

To measure the global similarity between annotations, we utilize **one-to-one accuracy**. This is computed by using optimal bipartite matching to pair up threads between the two annotations and computing the overlap between every pair of matched threads. This metric measures "how well

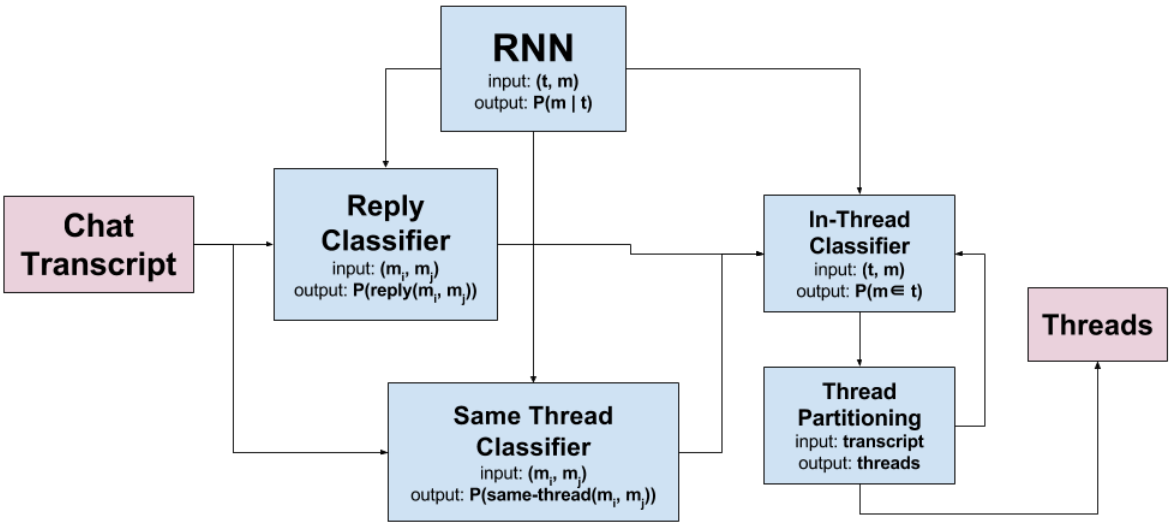

Figure 2. The proposed pipeline for disentangling a chat transcript into threads. Throughout the diagram, t represents a thread and m denotes a message.

we extract whole conversations intact" (Wang and Oard, 2009).

We also use the loc_k score to measure **local agreement**. For a particular message, the previous k messages are either in the **same** or a **different** thread. The loc_k score is computed by considering, for each message, the previous k messages and counting the matches in the same/different thread assignments across annotations.

The third and final metric used to score annotation agreement is the **Shen-F score**, as defined by Shen et al. (2006) which measures how well related messages are grouped. The Shen-F score is defined as:

$$F = \sum_i \frac{n_i}{n} max_j(F(i,j)) \tag{2}$$

where i is a ground truth thread with a length of n_i, n is the total length of the transcript and $F(i,j)$ is the harmonic mean of the precision and the recall. If the thread overlap is n_{ij}, the length of the gold-standard thread is n_i and the length of the proposed thread is n_j, then $F(i,j)$ is defined as follows:

$$P = \frac{n_{ij}}{n_j} \quad R = \frac{n_{ij}}{n_i} \quad F(i,j) = \frac{2PR}{P+R} \tag{3}$$

The max_j operation is taken over all detected threads. Since the matching is multiway (i.e., the same j can be chosen for different values of i), the score is not symmetric. When comparing human annotations, this lack of symmetry is addressed by treating the annotation with higher entropy as the gold standard.

Given a transcript of length n, with thread i having a size of n_i, the entropy of the annotation can be computed as:

$$H(c) = \sum_i \frac{n_i}{n} log_2 \frac{n}{n_i} \tag{4}$$

These metrics are utilized by both Elsner and Charniak (2010) and Wang and Oard (2009).

To account for differences in annotation specificity, Elsner and Charniak (2010) introduce **many-to-one accuracy**. This metric maps each of the threads of the source annotation to the single thread in the target with which it has the greatest overlap and counts the total percentage of overlap. Similarly to **Shen-F**, the higher entropy annotation is mapped to the lower one.

5 Datasets

We utilize a number of datasets to train various algorithms in our thread disentanglement pipeline. Two of these datasets are external datasets provided by previous research. The reply dataset[1] was annotated as part of this study as described in *Section 5.1*.

Since our classifier aims to learn immediate reply relationships between pairs of messages, there is a need for a newly annotated dataset. This section details the process of data acquisition and provides some preliminary analysis of the data.

Table 3: Single annotation statistics describing three annotations of a 524 message transcript. All of these metrics describe a single annotation. Thread density refers to the number of active threads at any given time.

	Mean	Max	Min
Threads	55.33	62	49
Avg. Thread Length	9.6	10.7	8.5
Avg. Thread Density	1.79	1.82	1.73
Entropy	3.99	4.42	3.64

5.1 Data Acquisition

For the acquisition of our annotated reply dataset a subset of the #linux IRC log data provided by Elsner and Charniak (2010) was used.

An interface was built and open-sourced to allow volunteers to manually annotate the data for the purpose of identifying direct reply relationships between messages. Users were instructed to proceed through the messages in the chat and select the immediate parents for every message.

Three volunteers, familiar with Linux terminology, independently annotated a set of 524 messages from the development dataset provided by Elsner and Charniak (2010). For every message in the dataset, annotators identified the potential immediate parents of the message, where $parent(m)$ is identified as the messages to which m is a direct reply. It is possible for a message to have no parents (e.g., starting a new thread of conversation) and multiple parents (e.g., following up on a multi-participant conversation).

5.2 Data Analysis

On average, we find that a message has 1.22 direct parents and 1.70 direct children. The relatively high number of children suggests that typically a message receives more than one reply. On the other hand, the lower number of direct parents suggests that a message is typically replying to a single parent message.

We can use the reply annotations to retrieve a thread annotation which resembles the structure of the annotated data by Elsner and Charniak (2010). This is done by identifying a disjoint set of messages such that no two messages in different sets share a reply relationship. This allows us to apply the read-based metrics described in *Section 4* to evaluate the annotation quality.

As is demonstrated in *Table 4*, our annotations have high inter-annotator agreement. While our agreement is not directly comparable to that of Elsner and Charniak (2010) due to our annotations being done on a subset of the data, our agreement is evidently much greater. This suggests that a reply-based annotation approach removes ambiguity and by extension removes noise from the data.

Table 4: Pair annotation statistics describing three annotations of a 524 message transcript. These metrics are all computed on a pair of threads and therefore describe inter-annotator agreement.

	Mean	Max	Min
one-to-one	81.49	87.79	74.81
loc_3	90.36	91.81	88.61
many-to-one	97.39	98.28	95.80
Shen F	87.70	100.0	75.00

5.3 External Datasets

In addition to the aforementioned annotated dataset, we utilized the corpus provided by Elsner and Charniak (2010) to train the same-thread classifier described in *Section 3.2*. This corpus consists of a pilot set, a development set and a testing set.

We also utilize the Ubuntu Dialogue Corpus (Lowe et al., 2015) to train a recurrent neural network to output the probability of a message occurring after a sequence of messages, as described in *Section 3.3*.

It is reasonable to utilize both of these datasets, as both are IRC chat logs from channels concerned with Linux related content. The corpora provided by Elsner and Charniak (2010) is from the ##LINUX channel, whereas the Ubuntu Dialogue Corpus (Lowe et al., 2015) consists of data from the ##UBUNTU channel.

6 Experiments and Evaluation Results

6.1 Reply Classifier

We use our annotated data, described in *Section 5.1*, to generate a set of positive examples (i.e., labeled replies) and a set of negative examples (i.e., the complement of the annotated replies). This data is used to train and validate the reply classifier described in *Section 3.1*.

The data is very imbalanced, with most pairs being non-replies which naturally leads to a large number of false positives (i.e., non-replies classified as replies). This issue was addressed by ad-

justing the class weights in order to stronger penalize false positives when training the random forest model. Specifically, the class weight of the negative class was set to be $\frac{num\_non\_replies}{num\_replies}$ while the class of the positive class was 1.

To further address the problem of imbalanced data, we employ a strategy used by Elsner and Charniak (2010) by only having our classifier consider messages within 129 seconds of each other. This brings the ratio of the two classes much closer and is useful for speeding up inference as well.

Using 10-fold cross-validation, we obtain an average precision of 0.91 and average recall of 0.92. Most message pairs are non-replies and as such these scores are non-representative. For the reply class, we have a precision of 0.83 and a recall of 0.49.

Using Gini feature importance (Breiman, 2001), we find the most important feature to be the probability outputted by the RNN, followed by the time difference and the message distance. This confirms that semantic relatedness is vital for detecting reply relationships. The time difference and message distance are both temporal measurements which represent the fact that new messages are typically a reply to recently sent messages.

6.2 Same-Thread Classifier

The same-thread classifier, described in *Section 3.2*, was trained on the development set provided by Elsner and Charniak (2010) which was described in *Section 5.3*. This classifier obtains a precision and recall of 0.7, which is similar to the result obtained by Elsner and Charniak (2010).

The Gini feature importances (Breiman, 2001) indicate that for the same-thread classifier, the most important features are the output of the RNN, the time difference and the 'same author' feature. The relatively high importance of the 'same author' feature can be explained by the fact that an author typically responds numerous times within a conversation. This reaffirms the idea that the two classifiers learn to identify different relationships between messages, and therefore complement each other.

6.3 RNN

The recurrent neural network was trained with the hyperparameters described by Lowe et al. (2015) on the Ubuntu Dialogue corpus, described in *Section 5.3*. The final model performs within a few percentage points of the result they reported.

6.4 In-Thread Classifier

This classifier was trained and tuned on the pilot dataset provided by Elsner and Charniak (2010) as described in *Section 5.3*.

To handle the imbalance of the data, we adjust the class weight when training and only consider thread-message pairs within 129 seconds of each other.

With 10-fold cross-validation, we obtain an average precision of 0.91 and an average recall of 0.92. For the positive class, we have a precision of 0.89 and a recall of 0.69.

Using the Gini feature importances (Breiman, 2001), we find that the most important features for the in-thread classifier are the time difference, the proportion of the author's previous messages which belong to the thread, the mean output of the reply classifier and the mean output of the same-thread classifier.

The in-thread proportion of the author's messages is likely a strong feature because high activity within a particular conversation indicates a high likelihood of continued activity.

It is plausible that the output of the RNN is not a strongly used feature because it is already present in the output of the reply and same-thread classifiers.

6.5 Disentanglement Evaluation

We compare the performance of our thread partitioning pipeline to the results reported by Elsner and Charniak (2010) and Wang and Oard (2009). Both of these papers evaluated their disentanglement models on the same dataset, consisting of six annotations of the same transcript of 1000 messages, provided by Elsner and Charniak (2010). The comparison is shown in *Table 5*, where our approach is shown to outperform the other methods with respect to loc_3 and outperforms Elsner and Charniak (2010) in all metrics.

However, we suspect that the results reported by Wang and Oard (2009) were boosted by the inclusion of *system messages* when computing all of their metrics. In contrast, Elsner and Charniak (2010) only include them for the computation of loc_3.

System messages are classified into thread -1 by all of the annotators and are extremely easy to classify as they always appear in the form *"X joined the room"* or *"X left the room"*. We compare the performance of our pipeline, provided

Table 5: The results obtained by our pipeline compared to annotators, Elsner and Charniak (2010) and Wang and Oard (2009).

	Annotators	Elsner & Charniak	Wang & Oard	Proposed Pipeline
Mean one-to-one	52.98	41.23	47.00	44.02
Max one-to-one	63.50	52.12	-	52.75
Min one-to-one	35.63	31.62	-	36.75
Mean loc_3	81.09	72.94	75.13	78.64
Max loc_3	86.53	74.70	-	80.27
Min loc_3	74.75	70.77	-	77.23
Mean Shen F	53.87	43.47	52.79	45.86
Max Shen F	66.08	57.57	-	52.22
Min Shen F	33.43	32.97	-	36.75

Table 6: Our results compared to Wang and Oard (2009), provided we include the system messages when computing the metrics.

	Wang & Oard	Pipeline
Mean one-to-one	47.00	55.22
Mean loc_3	75.13	78.64
Mean Shen F	52.79	56.62

Table 7: Demonstration of the significant quality drop observed when removing the RNN.

	Pipeline	Without RNN
Mean one-to-one	44.02	41.06
Mean loc_3	78.64	76.52
Mean Shen F	45.86	43.01

that we include all system messages, to the results obtained by Wang and Oard (2009) in *Table 6* where our approach outperforms in all metrics.

6.6 Pipeline Discussion

Our pipeline strongly outperforms previous research, likely due to the fact that we leverage different models and data sources to capture a variety of relationships between messages.

The reply classifier is trained on specifically annotated reply relationships, leading it to excel at identifying strong, local agreements between messages.

The same-thread classifier is trained on pairs of messages that belong to the same thread, resulting in a model which captures broad relationships between messages in the same thread. This is observed in the relatively high importance the model places on the 'same author' feature.

The RNN, trained on a large corpus of unlabeled data, is adept at identifying the strength of semantic relationships between messages. As a result, it is utilized as a feature in every classifier of the pipeline.

Removing the RNN entirely from the pipeline results in a significant quality drop, as shown in *Table 7*. This drop reaffirms that the pipeline's quality stems from its ability to model semantic relationships between messages.

The in-thread classifier is the final model that trains and evaluates on top of the output of the previous models. This model decides how to best utilize the previous models and combines their outputs to ultimately decide whether a message belongs to a thread.

7 Conclusions and Future Work

This work approaches the problem of thread disentanglement from the perspective of identifying reply relationships between messages. A corpus of annotated data for this task is provided, which should allow future research to expand on the work presented in this paper. From our high inter-annotator agreement, it is evident that detecting reply relationships has relatively little noise. We incorporate a combination of features in our disentanglement pipeline, using both meta-data and semantic relationships between messages. We further show that unlabeled data in combination with neural based approaches are effective in aiding with thread disentanglement. The model that we present outperforms all previous work when system messages are included.

In the near future, we plan to expand our reply

annotated corpus, using our open-sourced annotation software, particularly for the purpose of evaluation. We have shown that our reply-based annotation has higher inter-annotator agreement relative to the annotations provided by Elsner and Charniak (2010), therefore supplementing the reply corpus and using it for evaluation may lead to less noisy evaluations.

While we have shown that using an RNN is effective for detecting replies, it may prove useful to perform further experiments to determine if it can be applied more effectively. For example, adding an attention mechanism could be beneficial for the purposes of disentanglement. Additionally, in order to reduce noise in the data and therefore lead the model to converge faster, it is worth experimenting with training the RNN on a disentangled corpus. Other directions for future research could involve exploring these ideas.

References

S. Gross S. Chintala A. Paszke. 2017. Pytorch. github.com/pytorch/pytorch.

Paige Adams and Craig Martel. 2010. Conversational thread extraction and topic detection in text-based chat. *Semantic Computing* pages 87–113.

Leo Breiman. 2001. Random forests. *Machine learning* 45(1):5–32.

Giuseppe Carenini, Gabriel Murray, and Raymond Ng. 2011. Methods for mining and summarizing text conversations. *Synthesis Lectures on Data Management* 3(3):1–130.

Micha Elsner and Eugene Charniak. 2010. Disentangling chat. *Computational Linguistics* 36(3):389–409.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *arXiv preprint arXiv:1506.08909* .

Jacki O'Neill and David Martin. 2003. Text chat in action. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*. ACM, pages 40–49.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Dou Shen, Qiang Yang, Jian-Tao Sun, and Zheng Chen. 2006. Thread detection in dynamic text message streams. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pages 35–42.

David C Uthus and David W Aha. 2013. Multiparticipant chat analysis: A survey. *Artificial Intelligence* 199:106–121.

Lidan Wang and Douglas W Oard. 2009. Context-based message expansion for disentanglement of interleaved text conversations. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*. Association for Computational Linguistics, pages 200–208.