Comparing Different Clustering Methods for Conversation Disentanglement

Atom Roese, Jici Huang
Kutztown University of Pennsylvania
aroes474@live.kutztown.edu

ABSTRACT

There is an increasing number of technical questions were discussed via text messages using platforms such as Discord. With 150 million monthly active users according to data from statistics, Discord has been producing a large quantity of data crossing various channels. Mining the messaging data has been significantly beneficial for business and technology purposes. For example, there are portions of conversations from users who are mainly working as a developer that are closely relevant to technical issues and concerns. Mining Discord chat messages can potentially add more knowledge to the domain of the technology as well as clarify developers' concerns. The limitation from the data, however, is tangled together because the channel can have many users responding concurrently.

To better analyze the dataset originating from Discord, the disentanglement efforts need to be paid to categorize the chat messages. In this paper, we introduce two clustering methods. The two clustering methods are implemented to process the dataset collected from the Discord chat application. One is the Greedy clustering method and the other one is the hierarchical clustering. We applied both methods on the same dataset to compare the accuracy and efficiency of the two methods. Furthermore, Random Forest algorithm was implemented to prepare the dataset for the previous two methods to proceed successfully. We compared the results generated from Greedy clustering and hierarchical clustering.

KEYWORDS

Greedy clustering, hierarchical clustering, messaging data, Random Forest algorithm, disentanglement.

1 Introduction

Given the convenience of the mobile apps, users tend to communicate through text-messaging rather than verbally. Such users can be technical workers or professionals. The poplar chat apps for technical usage are Discord, Slack, Internet Relay Chat (IRC), Microsoft Teams. With the apps that operates on multiple platforms, problems can be resolved more efficiently. Often times, the solution of those problems can be recorded through chat messages. Tremendous messages can be sent and received every second. Potentially, all the messages have gone through the network as well as through the server. These Discord messages can be potentially captured and analyzed for commercial use or research purposes, often non-profit purposes.

There are a few existing techniques on analyzing messaging data. For example, Sentiment Analysis (SA) has been used mainly to extract opinions from the text data. Objectively, SA aims at capturing the sentences with subjective opinions and effective statements that support the subjective opinions [1]. Latent Dirichlet Allocation (LDA) for topic modeling has also been widely used in text-based data analysis [2]. The challenge appeared when the text messages are tangled and conversations are mixed together. More algorithms are turning their attention to group up messages into each individual conversation to better analyze the topics and attract meanings of the messaging data. This process specifically is called disentanglement [3]. A corpus and algorithm were designed to run manually on a data sourced from IRC [3]. The manual work can cost vast amount of time for the data nowadays. The error rate from manual work can be hard to estimate. A software was developed to replace the manual work and was tested on the data sourced from Slack [4]. Along the similar approach and the data pipeline, create DISCO dataset from executing disentanglement techniques and demonstrate their results based on the manual validation [5]. The pioneer work on Discord data applied a disentanglement technique on the one-year public data from Discord chat conversations of several software development communities [5]. The dataset is named as DISCO [5]. More detailed information about the DISCO dataset is provided in the next section, Section 2.

Follow along the DISCO dataset, we experimented with two clustering methods on the dataset collected from Discord application. One is the Greedy clustering method [3][6, p. 157]. The other method is called the hierarchical clustering method. Essentially, we use Greedy clustering method to group up the text messages that are belong to the same person. We perform this task repeatedly until the cluster is formed, which is a group of messages.

The paper is organized as follows. Section 2 describes the dataset we experimented with. Section 3 provides methods we are using to conduct the experiments. Section 4 charted

Ivor : It's not really a big deal but it's kind of a hassle to run video and audio through WSL. That's the only issue I've had with it.
Yahaya : isn't it a lot easier with WSL2?
Yahaya : I haven't played around iwth it much
Yahaya : on account of me having Covid19 when it released
Ivor : I didn't find an easy way to do it anyway.
Tsering : Why does vscode shows linter installation is not on PATHconsider ......
Tsering : [emojis]
Jahid : because it is not on Path
Syere : Ye
Tsering : Something error with python

Figure 1: Example of an entangled conversation from the DISCO dataset. (Pythongeneral Aug2020) This excerpt contains two intuitively distinct conversations about WSL and an issue with VSCode configuration in the same stream of messages

out the results from running the methods. Section 5 conducts the discussion on the different clustering methods used.

2 Datasets

The DISCO dataset is a collection of $1,508,093$ messages from channels of four different programming-related Discord servers posted from November 2019 to October 2020 [5]. In Discord channels, multiple distinct conversations can occur simultaneously in the same channel without any clear discriminator other than that of human intuition,as seen in Figure 1. Such data can be rich with information and is readily available, which makes difficult to apply existing natural language processing techniques, especially when a message could potentially be unrelated to some of the surrounding messages.

The process of breaking subsets of messages up into its own individual conversation is known as disentanglement. The disentanglement process can be reduced into a clustering problem. The high-level components of this process are the Grouping Classifier (abbreviated "GC") and the clustering method. The GC takes two messages and decides whether they belong to the same conversation, and to what degree of certainty. Permuted comparisons by this GC are then used as input to a clustering algorithm to break the messages into subsets of messages. The messages within the same subset have high GC outputs, and messages in different subsets have low GC outputs. This translates to messages that belong to the same group being together and messages that do not belong to the same group being kept apart. The formal definition is given in section 3.

3 Methods

The disentanglement process involves two major components: The Grouping Classifier and the clustering method. The GC is a function that takes two messages and determines whether they belong together, while the clustering method uses the outputs from the GC to break the messages up into subsets. Several clustering methods require their inputs to be points in a Euclidean space. Thus, all messages are embedded in a single unique N-dimensional Euclidean space where N is the number of messages being clustered. Each message gets an axis in this space on which all other messages are plotted based on their GC values.

The formal definition for this clustering space is as follows: Let $\mathbb{M}$ be the set of all messages being clustered, and let N be the cardinality of $\mathbb{M}$. Let there be a grouping classifier function $gc : \mathbb{M} \times \mathbb{M} \rightarrow [-1, 1]$ which returns the similarity between two messages and satisfies $gc(a, b) = gc(b, a)$ and $gc(a, a) = 1$. It is not guaranteed that gc is transitive. The space $\mathbb{R}^N_{\mathbb{M}}$ is defined such that $\forall\ m \in \mathbb{M}$, $\exists\ s \in \mathbb{R}^N_{\mathbb{M}} \mid s = (gc(\mathbb{M}_1, m), gc(\mathbb{M}_2, m), ..., gc(m, m), ..., gc(\mathbb{M}_N, m))$. The distance metric between two points in $\mathbb{R}^N_{\mathbb{M}}$ is the Euclidean distance. The messages are then passed to the clustering method as vectors in $\mathbb{R}^N_{\mathbb{M}}$ so that the Euclidean distance between them is always meaningful. The construction of this space is intended to preserve a composite of direct similarity (similarity between any two messages) and mutual similarity/dissimilarity. That is, if two messages are similar or dissimilar to the same messages, then they are all closer to each other within this space. Similarly, mutual dissimilarity causes messages to appear closer to one another on the relevant axis.

Messages cannot be passed to most candidate GC functions as plain text or strings. Thus, each message is converted to feature vectors whose components represent different conventionally measurable aspects of the text such as message length in words, time sent, author of the message, whether the message mentions another user, and if the message contains specific technical words. The specific elements of message feature vectors is the same as in the Subash et. al. [5].

In this paper, a Random Forest classifier is used as the GC. Various clustering methods from the SciPy [7] hierarchical clustering module are then compared to the ground-truth human disentanglement provided by the DISCO dataset [5]. The result of these clustering methods is not one particular clustering of the messages, but a dendrogram of message relatedness. This dendrogram is then "flattened" by cutting off clusters at a pre-specified threshold and considering messages in the resulting sub-trees as clusters. This "cutoff" threshold can be decided based on several factors, such as messages per cluster or the total number of resulting clusters. Here, the cutoff threshold targets at the total number of clusters. We attempt to keep other disentanglement components identical or comparable to the methods used in Subash et. al. [5] save for the clustering mechanism. As such, the choice for the GC remains a Random Forest classifier with 100 estimators using the same feature vectors.

We can describe Elsner et. Charniak's Greedy clustering algorithm recursively as follows [3]. Let $\mathbb{M}$ be the set of all messages being clustered. $\forall\ m \in \mathbb{M}$, let $\mathbb{P}_m = \{n \in \mathbb{M} \mid n$ was sent within five minutes before $m\}$. The message thread containing m is the same as the message $h \in \mathbb{P}_m$ satisfying $\forall\ n \in \mathbb{P}_m, gc(h, m) \geq gc(n, m) \geq 0$. That is, the chosen message h maximizes $gc(m, h)$ where $h \in \mathbb{P}_m$ and $gc(m, h) > 0$. If no such h exists, then m is in a new thread. This algorithm has a strong bias towards consecutive, nonoverlapping threads. The reason for and consequences of this bias are in Section 5.

The Greedy clustering method described above does not allow for choosing the number of resulting clusters. Therefore, two sensible cutoffs are chosen for the hierarchical methods to allow for a fair comparison. The first cutoff is set to target the actual number of different conversations that occurred in the transcript. This information may not be present at the time of disentanglement (infeasible to obtain/estimate) and may give the hierarchical methods an unfair advantage as a result, so a second cutoff is chosen to target the number of threads determined by the Greedy disentanglement which relies on no outside information. The results of the comparison between these clustering methods can be found in Section 4.

4 Results

The metrics table compares the accuracy of disentanglement and shows how many messages a particular method disentangled accurately. The statistics table provides information on the threads resulting from the disentanglement, regardless of accuracy. Table 1 and Table 2 list data for disentanglements made with the knowledge that 254 different conversations occur in the conversation being disentangled. Table 3 and Table 4 list data for disentanglements restricted to the thread count approximated by the Greedy algorithm.

The thread density statistic describes the average number of conversations occurring at any one time. For any given message in a transcript, the thread density at that point is equal to the number of threads that contain both a message sent before and a message sent after the given message. The entropy column refers to the entropy of the lengths of threads in the resulting disentanglement. The adjusted rand score is provided by mapping the thread index of each message into a list, which is sequentially numbered starting at 0. The index lists for the reference disentanglement and proposed disentanglement are then passed into the sklearn.metrics.adjusted_rand_score function [8]. An identical process is done with the sklearn.metric.adjusted_mutual_info_score metric [8].

The Greedy method is the simple Greedy clustering described in the original paper [3]. Others are various hierarchical clustering methods exposed by the SciPy linkage function [7]. In almost all metrics, hierarchical clustering performs poorly compared to the simpler approach. The "Human" method in the statistics tables refers to the ground-truth human disentanglement being used for comparison.

The pairs metric counts the number of pairs in a given disentanglement that are grouped incorrectly. The score is counted based on whether any two messages are in the same group and other factors such as which group are ignored. The metric is represented by this count divided by the total number of pairs (maximum possible number incorrect).

Table 1: Disentanglement Metrics, True Thread Count

Method	**Pairs Metric Norm.**	**Adj. Rand.**	**Adj. Mutual Info**
Greedy	0.0017	0.8945	0.9643
Ward	0.0206	0.2670	0.7780
Complete	0.0401	0.1511	0.6529
Weighted	0.1035	0.0658	0.6308
Average	0.1287	0.0534	0.6019
Median	0.2739	0.0235	0.4651
Single	0.3210	0.0193	0.4245
Centroid	0.3469	0.0160	0.3938

Table 2: Disentanglement Statistics, True Thread Count

Method	**Avg. Len.**	**Density**	**# of Threads**	**Entropy**
Complete	13.9528	3.9252	254	6.4142
Ward	13.9528	2.0982	254	7.3312
Weighted	13.9528	1.8584	254	5.8760
Average	13.9528	1.8047	254	5.5688
Median	14.1195	1.5762	251	4.1720
Human	14.0079	1.5274	253	7.2405
Single	13.9528	1.5093	254	3.8377
Centroid	14.2329	1.4989	249	3.6432
Greedy	11.6579	1.1002	304	7.4559

Table 3: Disentanglement Metrics, Greedy Thread Count

Method	**Pairs Metric Norm.**	**Adj Rand.**	**Adj. Mutual Info**
Greedy	0.0017	0.8945	0.9643
Ward	0.0139	0.3287	0.7842
Complete	0.0308	0.1801	0.6727
Weighted	0.0798	0.0829	0.6543
Average	0.0929	0.0704	0.6318
Median	0.2062	0.0329	0.5175
Single	0.2757	0.0235	0.4536
Centroid	0.2889	0.0213	0.4360

Table 4: Disentanglement Statistics, Greedy Thread Count

Method	**Avg Len.**	**Density**	**# of Threads**	**Entropy**
Complete	11.6579	3.9673	304	6.8240
Ward	11.6579	2.2097	304	7.7071
Weighted	11.6579	1.9235	304	6.2836
Average	11.6579	1.8922	304	6.1005
Median	11.6964	1.6538	303	4.8309
Single	11.6579	1.5672	304	4.2542
Centroid	11.6579	1.5466	304	4.1369
Human	14.0079	1.5274	253	7.2405
Greedy	11.6579	1.1002	304	7.4559

5 Discussion

According to Table 1 and Table 3, the simple Greedy clustering algorithm results in a much more accurate disentanglement than more complex methods on all metrics with wide variability. Disentanglements using the Ward and Complete methods retain accuracy comparable to the Greedy method, but others result in accuracy losses of approximately $10-34\%$. This indicates that certain clustering mechanisms fundamentally model the flow of natural conversation better than others. The Greedy algorithm is an extreme case that depends heavily on consecutive messages more than the other methods and thus provides more sensible disentanglements. If most messages in the same thread come one after another and the Greedy algorithm fundamentally tends towards these groupings, then more natural disentanglements will be produced.

Under this Greedy algorithm, a new thread is created only when no message has been sent within less than five minutes before a message or all messages sent five minutes before a message have a grouping classifier rating less than zero. This difficulty in making a new thread becomes apparent in the resulting thread density produced by the algorithm: From the thread density column in Table 2 and Table 4, we see an average thread density of 1.53 in the manually disentangled transcript in the DISCO dataset. The Greedy clustering results in an average thread density of 1.1 for the same transcript. From this, we can see that the Greedy algorithm is reluctant to declare that more than one conversation is occurring at any given time even at times when this is true; one conversation must end before another can begin.

Ignoring utterance accuracy, several of the hierarchical clustering methods give a thread density closer to the actual than the Greedy clustering. This effect is pronounced in Table 2 when they are given special information about the number of threads present, and all hierarchical methods result in closer thread densities except for the Complete and Ward methods. This tendency persists even in Table 4 when they are forced to match the thread count that the Greedy clustering detects (an over-estimate). Here, all hierarchical thread densities increase, but all hierarchical methods, except for the Complete and Ward methods, remain closer to the human disentanglement than the Greedy method.

Although most pronounced in the Greedy algorithm, the trade-off between accurately grouping utterances and accurately modeling the distribution of threads is present in the hierarchical clustering methods. Those methods that have lower pairs metric (higher accuracy) have thread densities further from the human disentanglement, and those with higher pairs metrics (lower accuracy) have thread densities closer to the human disentanglement. To accurately model the distribution of conversations within a transcript, the accurate groupings of individual utterances must be sacrificed. Depending on the prospective application of disentangled conversations, clustering methods with very poor accuracy may be appealing due to their superior modeling of the transcript as a whole.

The methods presented here are still open to improvement in several ways which may eliminate the necessary trade-off between fine-grained and general accuracy. For example, the message representation given to the GC function remains simplistic and poorly preserves the actual information available within the utterances. Changing this representation to a more modern text embedding could significantly aid the GC in ranking messages as related or unrelated, and thus make any subsequent clustering more reliable. Additionally, the clustering space $\mathbb{R}^N_{\mathbb{M}}$ could be modified to include the time at which an utterance occurred to preserve that information past the GC in the same way the Greedy algorithm does. Such a modification may force the hierarchical clustering methods to give extra weight to the utterance time, and thus result in clusters that group messages more naturally.

6 Acknowledgement

We are very thankful for the support for graduate student from Example University through graduate assistant scholarship.

References

[1] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, R. Oliveto, Sentiment analysis for software engineering: How far can we go?, in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 94–104 (2018), doi:10.1145/3180155.3180195.

[2] *The Pushshift Telegram Dataset* (Zenodo, 2020), doi:10.5281/zenodo.3607497.

[3] M. Elsner, E. Charniak, You talking to me? a corpus and algorithm for conversation disentanglement, in J. D. Moore, S. Teufel, J. Allan, S. Furui, editors, *Proceedings of ACL-08: HLT*, pages 834–842 (Association for Computational Linguistics, Columbus, Ohio, 2008).

[4] P. Chatterjee, K. Damevski, N. A. Kraft, L. Pollock, Software-related slack chats with disentangled conversations, in *Proceedings of the 17th international conference on mining software repositories*, pages 588–592 (2020).

[5] K. Muthu Subash, L. Prasanna Kumar, S. L. Vadlamani, P. Chatterjee, O. Baysal, DISCO: A Dataset of Discord Chat Conversations for Software Engineering Research, 2022, doi: 10.5281/zenodo.5909202.

[6] J. Kleinberg, E. Tardos, *Algorithm Design* (Addison Wesley, 2006).

[7] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific

Computing in Python, *Nature Methods*, *17*:(2020), 261–272, doi:10.1038/s41592-019-0686-2.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, *Journal of machine learning research*, *12*(Oct):(2011), 2825–2830.