

Clustering-based Summarization of Transactional Chatbot Logs

Luxun Xu
Computer Science & Engineering
University of California, Riverside
Email: lxu051@ucr.edu

Vagelis Hristidis
Computer Science & Engineering
University of California, Riverside
Email: vagelis@cs.ucr.edu

Nhat X.T. Le
Computer Science & Engineering
University of California, Riverside
Email: nle020@ucr.edu

Abstract—Transactional chatbots have become popular today, as they can automate repetitive transactions such as making an appointment or buying a ticket. As users interact with a chatbot, rich chat logs are generated to evaluate and improve the effectiveness of the chatbot, which is the ratio of chats that lead to a successful state such as buying a ticket. A fundamental operation to achieve such analyses is the clustering of the chats in the chat log, which requires effective distance functions between a pair of chat sessions.

In this paper, we propose and compare various distance measures for individual messages as well as whole sessions. We evaluate these measures using user studies on Mechanical Turk, where we first ask users to use our chatbots, and then ask them to judge the similarity of messages and sessions. Finally, we provide anecdotal results showing that our distance functions are effective in clustering messages and sessions.

I. INTRODUCTION

In the last years, chatbots (or conversational bots) have gained popularity thanks to advances in deep learning and natural language understanding. There has been much work – especially recently, using deep learning techniques – on how to train chatbots, using a training set of conversations (see Section VI). Most of these systems’ goal is to generate a bot message that is meaningful given the chat history. These systems are effective and appropriate for several applications such as providing basic information (e.g., a store’s opening hours), or never-ending chitchat conversations. Recently, there is also an increased interest in *transactional chatbots*, which aim to complete a specific transaction instead of just keeping a reasonable conversation going. Typical transactions include making a medical transportation reservation, scheduling a meeting, or connecting a patient to a suitable care provider. Several real-world examples of such chatbots are Kayak, Dom (from Domino’s) and Babylon Health (symptoms checker).

Note that open-ended informational bots, such as IBM Watson, are outside the scope of this paper. Such bots typically input a large knowledge base or even text documents, and offer question-answering capabilities, e.g., “what disease is associated with finger redness?” In contrast, we focus on multi-step transactional scenarios, where a user is trying to achieve a goal, such as booking a visit to a doctor office or making a food delivery order.

A limitation of current transactional chatbots is that there are no effective methods to utilize the chat logs to continuously improve future users’ experience.

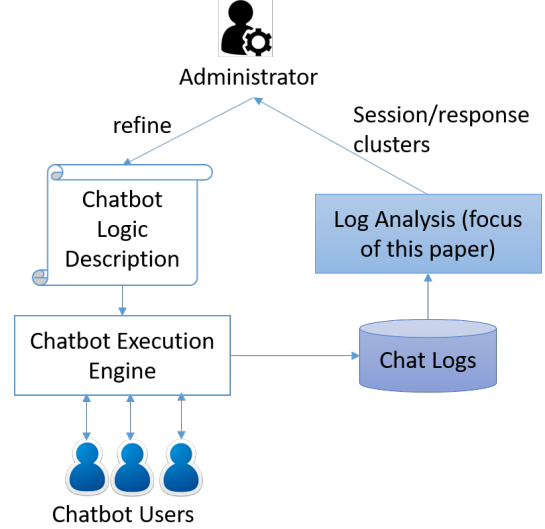


Fig. 1: Workflow of a transactional chatbot

We consider a comprehensive framework of transactional chatbots, as illustrated in Figure 1. In this framework, an administrator designs and deploys the chatbot. Once a fair amount of users have used the chatbot, a log analysis engine will analyze their chat logs to provide feedback to the administrator. Based on the feedback, the administrator can refine the chatbot logic to serve users better. In this paper, we focus on the log analysis engine, which summarizes the chat logs into clusters of similar chat sessions, in order to help the administrator on how to improve the chatbots. For instance, a cluster of similar erroneous user responses on the bot question of delivery address helps the administrator re-formulate the bot question into a clearer form.

The log analysis engine clusters chat logs at two levels: user responses and whole chat sessions. To achieve good clustering performance we propose novel distance functions, customized to chat log data. Our distance functions for user responses utilize a combination of textual, information extraction and conversational features, well-tuned by Information Theoretic Metric Learning technique. The chat session distance functions use the aforementioned response distance function as a module and employ sequence alignment techniques to consider the

conversation order.

We conduct user studies on Amazon Mechanical Turk (MTurk), where we first ask workers to use our chatbots to collect chat logs, and then ask them to compare pairs of responses or sessions with respect to their distance. We compare our distance functions against several baselines. For responses, our distance function achieves 0.82 overall accuracy over baseline’s 0.73. For sessions, our similarity function also achieves an overall accuracy of 0.85, higher than other measures.

This paper has the following contributions.

- We propose a novel distance function for chat responses to a bot question using textual, Information Extraction (IE), sentiment, and user interaction history (response time and attempts) features.
- We extend the response distance function to work for whole chat sessions.
- We present experiments on the analysis of chat logs we collected through MTurk, which shows that our distance functions outperform baselines.
- We publish the chat logs that we collected through MTurk, as well as the pairwise labels of responses and chat sessions ¹.

The rest of the paper is structured as follows. In Section II, we discuss the architecture of the chatbot platform and the problem definition. We propose our distance functions for user responses and whole sessions in Section III and IV respectively. In Section V, we present our experimental results and followed by related work in Section VI and conclusions in Section VII.

II. CHATBOT PLATFORM AND PROBLEM DEFINITION

A. Transactional Chatbot Abstraction

A transactional chatbot consists of (a) a logical flow description that defines the flow of the chatbot, and (b) an Information Extraction (IE) engine that extracts entities from the user input at each step. The extracted entities and entity values (as shown in Figure 2) are collected and subsequently determine the flow of the bot. For example, in Figure 3, the IE engine tries to extract the entity “location” from user’s response to the first (root) chatbot message. If such an entity value is successfully extracted, the bot moves on to the next message. Otherwise, it repeats the same message until the expected entities and values are extracted. Moreover, the extracted entity values can also be used as branching conditions. For example, in Figure 5, depending on the user’s response to Q₈, the bot can develop through different branches based on the entity and value extracted.

Note that slot-filling bots, like Amazon Lex, where the bot consists of a list of questions to fill slots – e.g., “what type of flowers?” or “when will you pick them up?” – can also be viewed as an abstraction of transactional chatbots, where a sequence of questions (called “prompt”) defines the bot logic, similar to one in Figure 3, and “slot type” defines the entity.

However, such bots generally do not support easy branching capabilities.

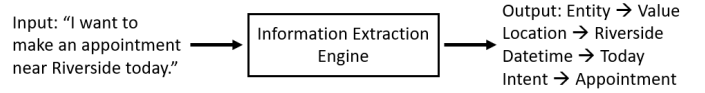


Fig. 2: Information Extraction engine

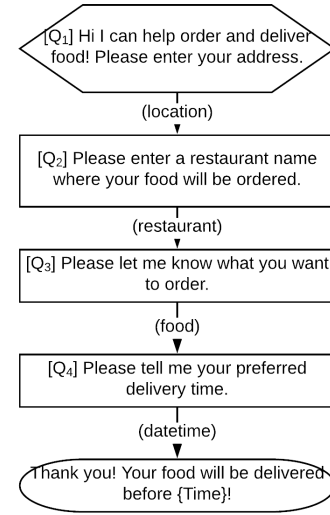


Fig. 3: Food ordering bot flowchart

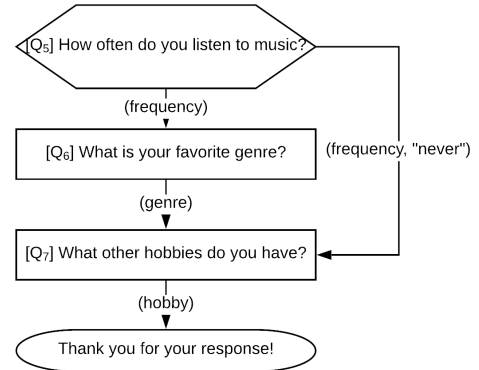


Fig. 4: Hobby bot flowchart

B. Chat Log Clustering Problem

To effectively cluster chat logs, we need an effective distance function, which is the focus of this paper. We divide this into two steps: first we compute the distance between two individual responses, and then use this function to compute the distance between two chat sessions.

Compute the distance between two user responses: A user response is a user message in response to a bot message. We

¹Data available at <https://github.com/LuxunXu/ChatbotDataset>

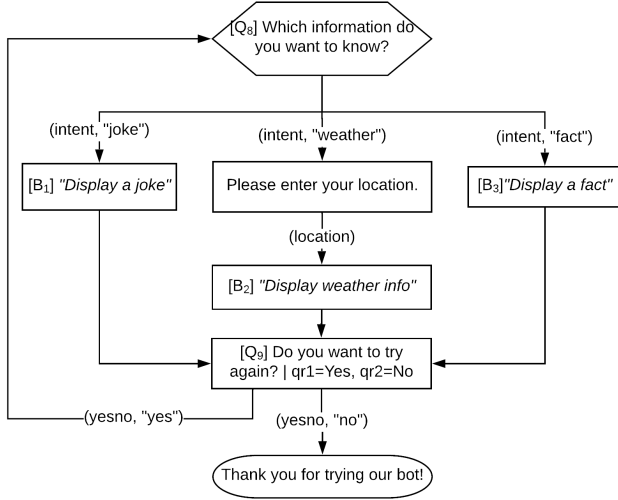


Fig. 5: Branching bot flowchart

view two responses as similar if they can be treated in the same way by the administrator for the purpose of improving the chatbot. Take a bot message “What is your date of birth?” as an example. User responses can be considered similar in the following cases (not an exhaustive list):

- If they are both “valid”, that is, if the IE engine correctly processes one (such as successfully extracting entity or entity values), it will likely process the other correctly. For example, “10/12/1983” and “1/12/2001” are both “valid” responses (so that the admin does not need to do anything about them).
- If they are both “invalid” in the same fashion. For example, (i) both responses are completely irrelevant to the bot message, e.g., “I have a headache”. (ii) If both responses may be incorrectly parsed by the IE engine due to its own limitation. For instance, response “05231990” resembles some characteristics of a birth date but may be extracted as a phone number by the IE engine. In this case, the admin will have to train a more robust IE engine. (iii) If both responses reveal similar confusion by the user due to possible imperfect designs of bot. For example, “Jan 02” and “March 04” both miss the “year”. In this case, the admin can split the bot message into simpler questions, or modify it to be more specific.

Compute the distance between two chat sessions: We define a session as a sequence of bot messages and user messages. Similar to user response distance, we define two sessions similar, if they can be treated together by the admin. Intuitively, two sessions are similar if they have similar sequences of bot messages and user messages.

III. USER RESPONSE DISTANCE FUNCTION

Our distance function is based on a combination of carefully chosen features, and Information Theoretic Metric Learn-

ing [1] to automatically learn appropriate weights and dependencies among these features.

A. Response Features

We model user responses by vectors of features (Table I) and use their Euclidean distance as response distance function. Statistical features capture basic structural information of the response. Information extraction (IE) features capture how confident or successful the IE engine extracts the entity. Conversational features embeds the user’s effectiveness and sentiment in responding to the bot. We later show that they are useful in session clustering. For example, “attempt” feature indicates the number of attempts of the current response to the bot message due to repetition (when bot is unable to understand the response). A high attempt number indicates poor response quality and implies that the bot message is poorly formulated. Lastly, we include a word embedding feature to capture semantic information of a response. We use fastText’s pre-trained English vectors [2] to construct a 300-dimensional vector for each response.

B. Mahalanobis Distance and Metric Learning

Euclidean distance is often not a good metric in high dimensions due to its lack of considering variable correlations. This issue can be solved by using Mahalanobis distance. Given a set of n points $x_1, \dots, x_n$ in $\mathbb{R}^d$, the (squared) Mahalanobis distance is given in Equation 1 [1].

$$d_A(x_i, x_j) = (\mathbf{x}_i - \mathbf{x}_j)^T A (\mathbf{x}_i - \mathbf{x}_j), \quad (1)$$

Practically, matrix A is re-weighting the features. By choosing a particular A matrix, we can boost or reduce certain features’ importance in the clustering.

We use the Information Theoretic Metric Learning (ITML) algorithm [1] to learn matrix A that reflects the similarity opinions of experts (we use pairwise similarity preferences in our experiments). It is fast and scalable, thus applicable to huge datasets. ITML assumes prior (expert) knowledge regarding inter-point distances, which serves as constraints in learning matrix A . Such constraints correspond to the aforementioned similarity opinions of experts. In practice, we input an $d \times n$ matrix (where d is the dimension of the response vector and n is the number of responses to a particular bot message) and similarity constraints, ITML learns matrix A efficiently and matrix A is used in Equation 1 to calculate the Mahalanobis distance between two responses. Moreover, the input of ITML also requires a slack variable γ and an initial matrix A_0 . We use $\gamma = 1$ and identity matrix for A_0 . We denote distance between two responses as $Dist(R_i, R_j)$.

IV. SESSION DISTANCE FUNCTION

We define a chat session as a sequence of conversational pairs consisting of a bot message Q and a user response R , as shown in Table II. Formally, we define *Session A* : $A_1, A_2, \dots, A_n$, where $A_i = (Q_i^A, R_i^A)$ and $len(A) = n$.

Feature Type	Feature	Value
Statistical Feature	No. of Words/Tokens	3
	No. of Digits	6
	No. of Letters	3
	Total Length	12
IE Feature (Confidence)	Food	0
	Location	0
	Datetime	0.965
	Restaurant	0
Conversational Feature	Time to Respond (sec)	10
	Attempt	1
	Sentiment level	0.0
Word Embedding Feature	fastText Vector	1.1321
		-0.2313
		...
		0.9494

TABLE I: Features and raw values for sample response “May 29, 1992” in the food ordering bot shown in Figure 3

We first extend the distance between responses of the same bot message to the similarity of any user responses in Equation 2

$$Sim_R(A_i, B_j) = \begin{cases} \frac{D_{Max} - Dist(R_i^A, R_j^B)}{D_{Max}} & \text{if } Q_i^A = Q_j^B, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

where D_{Max} is the maximum distance between two responses to the same bot message. Thus the similarity between two responses will be between 0 and 1. Moreover, the similarity between responses to different bot messages will be 0.

We define a matching M between session A and session B as a way to match every conversational pair in A to a correspondent in B . A conversational pair in session A can be matched to either a single pair (with the same bot message) in session B or *null* and vice versa. Formally, we define M as following: $M(A_i) = B_j$ or *null*.

We define the similarity between two sessions A and B as the maximum similarity over all matchings between them.

$$Sim(A, B) = \max_M \sum_i^n Sim_R(A_i, M(A_i)) \quad (3)$$

where n is the length of A . The matching that maximizes the session similarity is called the optimal matching. Note that, there are pairs in B without matched pairs in A , but they do not change the above similarity formula (without loss of generality). The final session similarity is normalized by the greater length between Session A and Session B as below.

$$SimNorm(A, B) = \frac{Sim(A, B)}{\max(len(A), len(B))} \quad (4)$$

We consider two variants of matching, one where crossing is allowed and one when not. A matching is non-crossing if, given two matched pairs $M(A_{i_1}) = B_{j_1}$ and $M(A_{i_2}) = B_{j_2}$, without loss of generality, if A_{i_1} precedes A_{i_2} , i.e. $i_1 < i_2$, then B_{j_1} must precede B_{j_2} , i.e. $j_1 < j_2$.

The matching, where crossing is not allowed (we call it *ordered*), is illustrated in Figure 6a whereas matching with crossing is illustrated in Figure 6b. When crossing is allowed (we call it *unordered*), the matching may produce a higher similarity between two sessions (because more matchings between conversational pairs are possible) but the structure of the sessions (flow of the chat) is not strictly considered. When crossing is allowed, we can use Hungarian algorithm to find the matching that maximize the similarity between two sessions.

By	Message	Variable
Bot	Hi, I can help you order food from popular restaurant chains.	–
Bot	Please enter your address.	Q_1
User	5 Buckingham ct, Rockledge, FL, 32955	R_1 A_1
Bot	Please enter the restaurant name you prefer (within 10 miles).	Q_2
User	Chili’s	R_2 A_2
Bot	Chili’s is set as your restaurant. Now please let me know what you want to order.	Q_3
User	Honey Chipotle Chicken Crispers	R_3 A_3
Bot	Honey Chipotle Chicken Crispers will be ordered. When do you want it to be delivered?	Q_4
User	1:00 pm	R_4 A_4
Bot	Thanks! Your order will be delivered on 2019-01-11 13:00:00!	–

TABLE II: a session example from a food ordering bot

A. Sequence Alignment for Ordered Matching

We compute similarity between two sessions using sequence alignment technique [3] for ordered matching because the alignment produced naturally corresponds to matching without crossing. Dynamic programming is used with the recurrence relation defined in Equation 5.

$$Sim_P(A_i, B_j) = \max \begin{cases} Sim_P(A_{i-1}, B_j) \\ Sim_P(A_i, B_{j-1}) \\ Sim_P(A_{i-1}, B_{j-1}) + Sim_R(A_i, B_j) \end{cases} \quad (5)$$

where $Sim_P(A_i, B_j)$ is the similarity between sessions $A_1, \dots, A_i$ and $B_1, \dots, B_j$, that is, the prefixes of the sessions that end in A_i and B_j . $Sim(A, B) = Sim(A_n, B_m)$, where n and m are the lengths of A and B respectively. Typically, sequence alignment penalizes for unmatched pairs. Instead, we indirectly account for this by dividing by the maximum length as shown in Equation 4.

Figure 7 illustrates a typical process of finding the optimal alignment by dynamic programming. In this example, the normalized similarity is $SimNorm(A, B) = \frac{1.3}{3} = 0.433$. Moreover, the optimal matching can be obtained by performing standard back-tracing. The time complexity for finding the maximum similarity is $O(mn)$, where m and n are the length of two sessions respectively.

B. Other Similarity Functions

In this section, we propose two similarity functions that only consider one of the two main properties of a session, the structure or the content, which we include in our experimental evaluation to study which of the two properties is more important in defining a distance function, and if combining them is a good idea.

Structure-only: Structure-only similarity function only considers the structure of chat sessions, which is defined as the sequence of bot questions (on the bot flow diagram) triggered through the user's responses. In particular, we use Boolean values (shown in Equation 6) to indicate if there is a match between two conversational pairs.

$$Sim_R(A_i, B_j) = \begin{cases} 1 & \text{if } Q_i^A = Q_j^B, \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

Document-based: We can also consider chat sessions as documents, where each document is a collection of all user responses from the chat session. We use a vector space model (with term frequency) to represent each document and use cosine similarity to measure similarity between two sessions.

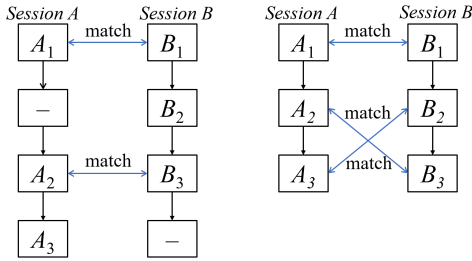


Fig. 6: (a) Left - Ordered matching, (b) Right - Unordered matching

	–	A_1	A_2	A_3
–	0.0	0.0	0.0	0.0
B_1	0.0	0.8(M)	0.8	0.8
B_2	0.0	0.8(–)	1.1	1.1
B_3	0.0	0.8	1.3(M)	1.3(–)

Fig. 7: Dynamic programming calculation of optimal matching for the example of Figure 6a. Arrows illustrates back-tracing process, where “–” indicates a matching to *null*, and “M” indicates a matching between A_i and B_j .

V. EXPERIMENTS

We conduct experiments to evaluate our response distance and session similarity functions. We first asked Mechanical Turk users to use our chatbots to collect chat logs, and then performed another experiment asking users to evaluate the similarity between pairs of responses or sessions.

We have built a chatbot platform, SmartBot360, which allows designing and deploying chatbots on Web pages,

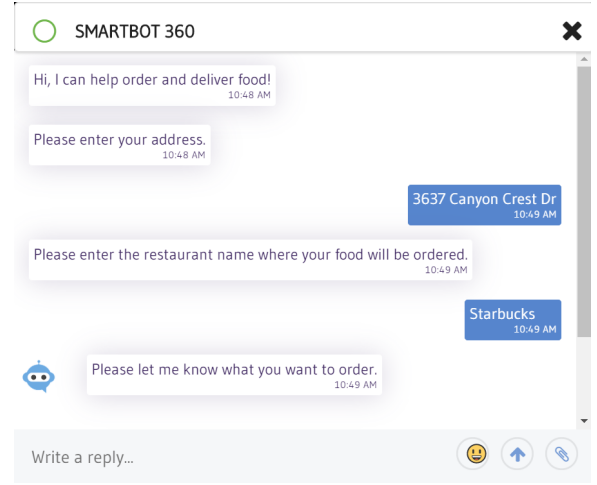


Fig. 8: SmartBot360 web interface

Facebook Messenger and on SMS. It follows the chatbot abstraction described in Section II-A, where the logic of the chatbot is defined by a flow diagram and an IE engine (we use wit.ai [4] in our experiments) is used to extract entities and entity values. The flow diagram is executed by our *Chatbot Execution Engine* (Figure 1). Figure 8 shows the web interface of SmartBot360 for the food ordering bot in action.

A. Response Similarity

To evaluate the effectiveness of response distance function, we collect human labels judging similarity of two pairs of responses. If one pair of responses is more similar than the other, the distance between them should be smaller. We measure the accuracy of the distance function agreeing with the labels. Our baseline is using fastText vector only. Other competitors include using non-fastText features and combining them. We also include comparisons for features with or without metric learning. Therefore, without metric learning, Euclidean distance is used whereas with metric learning, Mahalanobis distance is used. Furthermore, we find out that non-fastText features tend to have wide ranges. Hence, we perform min-max normalization on non-fastText features as an extra pre-processing step.

We start by deploy the food-ordering bot (Figure 3) on SmartBot360 platform and use Amazon Mechanical Turk (MTurk) to get people to use our chatbot. Next, we give MTurk workers a simple labeling task as illustrated in Figure 9. Specifically, each labeling task has two pairs of responses (to the same bot message), where the first response of each pair is fixed. MTurk workers need to label which pair of responses is more similar than the other both syntactically and semantically. For example, “Pair ONE” is the desired label in Figure 9. We also give a third “neutral” option if two pairs are really close.

For each bot message, we randomly generated 300 pairs under the condition that the two pairs generated must differ by at least 20% according to our baseline. This condition minimizes pairs that are too close to label because we want workers to choose a positive label instead of a neutral one.

Which pair of two reponses is more similar?

Pair ONE:
Response 1: As soon as possible.
Response 2: The sooner the better.

Pair TWO:
Response 1: As soon as possible.
Response 2: Monday.

Select an option

Pair ONE is more similar	1
Pair TWO is more similar	2
It is REALLY hard to tell	3

Fig. 9: Mechanical Turk task example

For all 300 labeling tasks, each one of them was assigned to three different workers. Only the labeling tasks that receive unanimous labels (either Pair ONE or TWO) are selected as valid labels as shown in Table III.

We also collect similarity labels for pairs of responses for ITML using MTurk. We randomly generate 200 pairs of responses to each bot message and ask MTurk workers to label them as similar or dissimilar. Each labeling task is assigned to three workers and the majority vote becomes the label. In practice, this labeling can be done by experts such as the administrator.

As shown in Table III, our proposed distance function combining all features with metric learning yields the highest accuracy for all four bot messages. It is the only one that achieves over 80% overall accuracy. We also theorize that IE features play an important role in the distance function. That is, a higher quality IE engine tends to produce better result. This can be backed by the fact that the accuracy for Q_1 and Q_4 is higher than that of Q_2 and Q_3 , because Q_1 and Q_4 use Wit.ai to extract entities and fine-grained values for IE features, whereas Q_2 and Q_3 use nutritionix API [5] that is only able to extract Boolean values for IE features.

B. Session Similarity

We use human evaluation to measure the accuracy that human labels of session pairs agree with the calculated session similarity score. We design two new chatbots to study session similarity to achieve high inter-coder agreement when performing similarity-labeling tasks. We differentiate them in terms of structures and we name them “hobby” (Figure 4) and “branching” bot (Figure 5). Note that in Figure 5, node B_1 , B_2 and B_3 use external APIs to generate the corresponding bot messages, thus are not of interest in calculating session similarity.

Again we use MTurk to collect chat logs and similarity-labeling data. We show two pairs of sessions using a difference viewer shown in Figure 10a and Figure 10b and only give two options (either Pair ONE is more similar or Pair TWO). Unfortunately, we are unable to get reliable labels from MTurk workers. We believe that this is due to the difficulty of the session labeling. Moreover, MTurk users vary greatly in terms of understanding the set criteria. Instead, we asked 10 students from UC Riverside to label sessions similarities. In the instruction, we state that by similar we mean both syntactically and semantically so that the worker must take both the flow of the chat and content of the responses into

account. Each similarity labeling task is assigned to three different students. Since we only allow two options for each label, the one with majority vote will be the final label. Finally, we compare the accuracy that labels of pairs of sessions that agree with the calculated session similarity score using different similarity functions. We break tie in similarity score comparisons arbitrarily.

The experimental results on the two bots are shown in Tables IV and V. For hobby bot, our ordered-matching and unordered-matching functions perform the best. This is expected because the hobby bot is designed to be linear, where the flow of the bot is somewhat “fixed”, thus there is no crossing in any matching. Moreover, the structure-only function performs bad because it is unable to differentiate between sessions with the same flows but different contents. The document-based function is the worst since it omits the session flow completely.

For branching bot, document-based function performs better compared to its performance on hobby bot. We believe that this is due to the increased size of sessions. Since each session for branching bot is larger, the structure of each session is less important comparing to the content. This can be backed by the fact that structure-only function performs the worst for branching bot. Still, our ordered-matching approach outperforms the rest. This shows that our method is effective in capturing both structure and content characteristics in complex chat sessions.

We now show a case study of session similarity. In Figures 10a and 10b, two sessions have the same structure where the sequence of the bot messages and user messages are the same, but Pair One is more similar because of the content of responses. Therefore, structure-only method is unable to tell the difference between two sessions, returning an equal similarity score of 1.0 for both pairs. However, the two matching methods as well as the document-based method, which consider content are able to capture the difference between two sessions, therefore agree with the label as shown in Table VI.

C. Anecdotal Results from Clustering Responses and Sessions

We further show that our distance functions can be used to generate meaningful clusters for both responses and sessions. These clusters provide high-level ideas of how the bot performs.

For response clustering, we use K-means clustering algorithm. The number of clusters is determined by the elbow method. In Table VII, we see that the majority of the users are able to provide valid addresses. Invalid responses are clustered in Cluster 1. In Cluster 5, it is shown that many users only enter a state instead of an actual, detailed address. This information suggests to the administrator to fine-tune the question or to add certain checks so that the bot will not progress if users only enter a state.

For session clustering, we use K-medoids clustering algorithm, specifically PAM) [6], because we do not have a vector representation of sessions. Similarly to responses, we

Features used	Q ₁ accuracy (<i>n</i> = 185)	Q ₂ accuracy (<i>n</i> = 136)	Q ₃ accuracy (<i>n</i> = 139)	Q ₄ accuracy (<i>n</i> = 156)	Overall accuracy
fastText only (baseline)	0.84	0.51	0.76	0.78	0.73
Non-fastText features only (normalized)	0.86	0.46	0.65	0.69	0.68
Non-fastText features only (normalized & metric learning)	0.81	0.54	0.61	0.73	0.68
All features (with non-fastText features normalized)	0.90	0.49	0.75	0.78	0.74
All features (with non-fastText features normalized & metric learning)	0.92	0.72	0.81	0.82	0.82

TABLE III: Accuracies of the agreement of various distance function to the human opinion for the responses to four bot messages. *n* denotes the number of response pairs with valid labels

1 Log 3713	1 Log 3179
2 Bot: Please answer the following question.	2 Bot: Please answer the following question.
3 Bot: How often do you listen to music?	3 Bot: How often do you listen to music?
4 Human: Almost everyday	4 Human: All the time
5 Bot: What is your favorite genre?	5 Bot: What is your favorite genre?
6 Human: 80's	6 Human: I don't have a favorite. i have some least favorites
7 Bot: What other hobbies do you have?	7 Bot: What other hobbies do you have?
8 Human: Swimming, arts & crafts	8 Human: Anytime related to work , family, or home. if you want to call those hobbies.
9 Bot: Thanks!	9 Bot: Thanks!

Fig. 10a: Pair ONE: Session 3713 and Session 3179

1 Log 3713	1 Log 3563
2 Bot: Please answer the following question.	2 Bot: Please answer the following question.
3 Bot: How often do you listen to music?	3 Bot: How often do you listen to music?
4 Human: Almost everyday	4 Human: Everyday
5 Bot: What is your favorite genre?	5 Bot: What is your favorite genre?
6 Human: 80's	6 Human: Posthardcore
7 Bot: What other hobbies do you have?	7 Bot: What other hobbies do you have?
8 Human: Swimming, arts & crafts	8 Human: Gaming, working out, rugby
9 Bot: Thanks!	9 Bot: Thanks!

Fig. 10b: Pair TWO: Session 3713 and Session 3563

	Correct	Wrong	Accuracy
Ordered-matching	94	6	0.94
Structure-only	70	30	0.70
Unordered-matching	94	6	0.94
Document-based	56	44	0.56

TABLE IV: Hobby Bot Experiment Result

	Correct	Wrong	Accuracy
Ordered-matching	75	25	0.75
Structure-only	58	42	0.58
Unordered-matching	66	34	0.66
Document-based	71	29	0.71

TABLE V: Branching Bot Experiment Result

use the elbow method to get the optimal number of clusters. In Table VIII, Log 3163 and Log 3461 are two cluster medoids. An intuitive interpretation from this clustering is that Log 3163 consists of only one round of user response, while Log 3461 consists of two rounds.

VI. RELATED WORK

Many studies on chatbots focus on response generation. However, studies on chat logs are scarce, not to mention using such studies to improve current chatbots. Therefore, we extend

	Pair ONE	Pair TWO	Agrees w/ label
Ordered-matching	0.470	0.735	Yes
Structure-only	1.0	1.0	No
Unordered-matching	0.470	0.735	Yes
Document-based	0.073	0.304	Yes

TABLE VI: Session similarity case study

Cluster	Sample Responses	Explanation
1	[hi, 02546, 13060632, what time is now?,...]	Invalid
2	[1250 rodeo drive, los angeles, 3653 losh lane,...]	Valid address 1
3	[104 market st, 1200 e 35th erie pa,...]	Valid address 2
4	[south avenue florida, amherst ma,...]	Valid address 3
5	[florida, new york, california, dallas,...]	State/city only
6	[2121 wisconsin ave, nw, washington dc 20007, 1 bridge street, chicago,...]	Valid address 4

TABLE VII: Response Cluster for Q₁

relevant works to other forms of documents and study their works on clustering and summarization.

Forum threads similarity Online forums resemble chat logs because discussions in forums between users resemble chat sessions in our context. There any many works on

Log	3163	Log	3461
Bot:	What would you like to know?	Bot:	What would you like to know?
Human:	Weather	Human:	Weather
Bot:	Please enter the location.	Bot:	Please enter the location.
Human:	Miami	Human:	Westlake village, ca
Bot:	The weather in miami is 84.02 degrees fahrenheit or 28.90 degrees celsius with broken clouds.	Bot:	The weather in westlake village is 79.03 degrees fahrenheit or 26.13 degrees celsius with haze.
Bot:	Would you like to try again?	Bot:	Would you like to try again?
Human:	No	Human:	Yes
Bot:	Thanks!	Bot:	What would you like to know?
		Human:	Fact
		Bot:	166,875,000,000 pieces of mail are delivered each year in the us
		Human:	Would you like to try again?
		Bot:	No
		Human:	Thanks!

TABLE VIII: Medoids of two clusters of branching bot.

clustering forum topics or threads. Singh et al. [7] devise a similarity measure between forum threads by decomposing threads into weighted overlapping components. Specifically, they divide threads into individual posts and post-reply pairs and reduce the problem into finding a maximum weight independent set. Said et al. [8] propose using asymmetric pairwise distance between posts to cluster posts in online forums. A key difference we have from thread similarity works is that the flow diagram in transactional chatbots imposes a more structured sequence of message pairs, which is not available in free-form threads. We leverage this structure in our distance measures.

XML document similarity The structural similarity is related to the problem of XML documents similarity. This is because XML schema inherently supports tree structures, which we can fit chat sessions (sequences of conversations) into. Tekli et al. [9] feature comprehensive studies on XML similarities which include tag similarity, edge similarity, path similarity and etc. Path similarity can be a good metric for chat sessions because chats successfully completing transactions tend to follow similar paths. Nierman et al. [10] propose a tree edit distance based similarity measure for XML documents. However, XML document similarity measures do not naturally distinguish the two roles (bot and user) in a chat session, nor do they adequately model loops.

Document summarization Ultimately we aim to provide chat log summaries, which is naturally relating to the multi-document summarization problem. Mani et al. [11] provide a method for summarizing similarities and differences of pairs of related documents using a graph representation for text. Radev et al. [12] propose centroid-based summarization of multiple documents. Goldstein et al. [13] propose multi-document summarization using sentence extraction. Summaries can be in a form of extraction or abstraction [14]. Unlike these techniques, which generally focus on the document’s textual content only, our work also leverages the structure, which is important in chat log data. Moreover, many of these papers conduct experiments on DUC datasets, thus their effectiveness on chat session data remains unknown.

VII. CONCLUSION AND FUTURE WORK

In this paper we proposed distance functions for chatbot responses and sessions that outperform baselines. We also showed that these functions can be used to generate meaningful clusters of responses and sessions. These clusters can then be used to help the administrators to improve the bots. In the future, we will focus on automatically generating high-level cluster summaries as well as automatic improvement suggestions. Ideally, such suggestions should be incorporated into the system with little or no human intervention.

ACKNOWLEDGMENT

This work was partially supported by NSF grants IIS-1838222, IIS-1619463, IIS-1901379 and IIS-1447826

REFERENCES

- [1] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon, “Information-theoretic metric learning,” in *ICML*, 2007.
- [2] T. Mikolov, E. Grave, P. Bojanowski, C. Puhresch, and A. Joulin, “Advances in pre-training distributed word representations,” in *LREC*, 2018.
- [3] W. Wang and O. R. Zaiane, “Clustering web sessions by sequence alignment,” in *Proc. International Workshop on DEXA’02*, 2002.
- [4] “wit.ai,” <http://wit.ai>, 2019.
- [5] “nutritionix api,” <https://www.nutritionix.com/business/api>, 2019.
- [6] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.
- [7] A. Singh, D. P. and D. Raghu, “Retrieving similar discussion forum threads: A structure based approach,” in *Proc. SIGIR’12*, 2012.
- [8] D. Said and N. Wanas, “Clustering posts in online discussion forum threads,” *IJCSIT*, 2011.
- [9] J. Tekli, R. Chbeir, and K. Yetongnon, “An overview on xml similarity: Background, current trends and future directions,” *Computer science review*, 2009.
- [10] A. Nierman and H. Jagadish, “Evaluating structural similarity in xml documents,” in *webdb*, 2002.
- [11] I. Mani and E. Bloedorn, “Multi-document summarization by graph search and matching,” *arXiv preprint cmp-lg/9712004*, 1997.
- [12] D. R. Radev, H. Jing, and M. Budzikowska, “Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies,” in *NAACL-ANLP*, 2000.
- [13] J. Goldstein, V. Mittal, J. Carbonell, and M. Kantrowitz, “Multi-document summarization by sentence extraction,” in *NAACL-ANLP*, 2000.
- [14] H. Lin and J. Bilmes, “Multi-document summarization via budgeted maximization of submodular functions,” in *NAACL HLT’10*, 2010.