Situation-Based Multiparticipant Chat Summarization: a Concept, an Exploration-Annotation Tool and an Example Collection

Anna Smirnova, Evgeniy Slobodkin, George Chernishev
Saint Petersburg State University
{anna.en.smirnova, eugene.slobodkin, chernishev}@gmail.com

Abstract

Currently, text chatting is one of the primary means of communication. However, modern text chat still in general does not offer any navigation or even full-featured search, although the high volumes of messages demand it. In order to mitigate these inconveniences, we formulate the problem of situation-based summarization and propose a special data annotation tool intended for developing training and gold-standard data.

A situation is a subset of messages revolving around a single event in both temporal and contextual senses: e.g, a group of friends arranging a meeting in chat, agreeing on date, time, and place. Situations can be extracted via information retrieval, natural language processing, and machine learning techniques. Since the task is novel, neither training nor gold-standard datasets for it have been created yet.

In this paper, we present the formulation of the situation-based summarization problem. Next, we describe Chat Corpora Annotator (CCA): the first annotation system designed specifically for exploring and annotating chat log data. We also introduce a custom query language for semi-automatic situation extraction. Finally, we present the first gold-standard dataset for situation-based summarization. The software source code and the dataset are publicly available[1,2].

1 Introduction

In the recent years, the attitude to multiparticipant chat has changed: what was regarded as a distraction is now used as primary means of communication in both professional and personal environments. However, its evident problems, such as the inability to quickly and efficiently navigate a large body of skipped messages, are yet to be addressed. One of the ways of addressing this is summarization. However, due to the specifics of text chat data, such as noise and length, no widely accepted model for this task has been created yet. Nevertheless, there have been notable works in the field. One of them is Collabot (Tepper et al., 2018): a fully-fledged chat summarizer, which, however, never went public. Additionally, there is a considerable body of work on email summarization, such as Ulrich et al. (2008), Loza et al. (2014), Joty et al. (2011), which present both annotated data and a summarization approach. While these works are an indispensable basis for the research in the area, we believe that chat data possesses enough specific qualities (such as extremely short message length, presence of specific slang and emoticons, and largely informal grammar and spelling) to warrant new annotation procedures and summarization methods.

To the best of our knowledge, publicly available annotated data for this task is both rare and, additionally, highly specific. Most of the aforementioned works have created their own specific annotation procedures and applied them to small volumes of data. Annotated data is hard to obtain in and of itself, and creating a gold-standard dataset from noisy raw data may take a lot of effort and time. Therefore, we have focused on creating a full-fledged annotation system for chat data.

In the current paper, we propose novel annotation guidelines for multiparticipant chat data. In our vision, it would be most practical to summarize such datasets by specific *situations*. We define a situation as a subset of messages revolving around a single event in both temporal and contextual senses. The set of situation tags would be specific for each particular dataset, and devising a standardized tagset currently does not seem

[1]https://github.com/mechanicpanic/Chat-Corpora-Annotator
[2]https://github.com/mechanicpanic/Situation_Dataset

Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop, pages 127–137
August 5–6, 2021. ©2021 Association for Computational Linguistics

possible. Each tagset would be devised by a human analyst and will be specifically suited for the needs of each user. This approach takes its roots in the ideas of open-domain event extraction, such as in (Ritter et al., 2012), but differs from them on several points. First, we are interested in groups of documents. Second, we do not explicitly extract event keywords. Instead, we offer the user to decide what situations revolve around which events and how they are represented in the data.

Furthermore, the quality of the training data has to be very high. In our understanding, creating a gold-standard dataset requires full attention of a human annotator, and relying on automatic recommenders would yield inferior results. Nevertheless, a recommender could be helpful for deep dataset exploration and for annotation assistance. Such assistance may come in a form of generating candidates for manual cross-checks when the annotator had finished their job or for rapid dataset prototyping. Since our task formulation is novel, there is no specifically trained machine learning (ML) model for it yet. To address the need for a recommender, we have designed a lightweight query language for rule-based detection of situations in chat datasets.

Next, we introduce Chat Corpora Annotator, a standalone desktop application for exploring and annotating multiparticipant chat datasets. To the best of our knowledge, this is the first tool that addresses both these tasks simultaneously. Additionally, we describe the annotation guidelines and the workflow for the summarization task.

Finally, we present an example collection that can be used to train machine learning models or serve as a gold-standard to assess summarization algorithms.

The main contributions of the paper are:

- An introduction of the situation-based summarization problem.
- A lightweight and easy-to-use annotation tool specifically designed for data exploration in multiparticipant chat logs.
- A special query language that can be used to generate annotation recommendations and run ad-hoc exploration queries.
- A workflow for CCA that is aimed at creating a dataset for the task of situation-based summarization.
- An example collection created using CCA.

2 Situations

Our inspiration for the proposed approach is based on cases such as a user taking a break from an important multi-participant chat for a significant amount of time. For example, it could be an employee taking a vacation. Having returned, they would have to catch up with the rest of their colleagues, which would include browsing chat discussions that happened during their absence. Therefore, they would be forced to navigate a large body of skipped messages which may be distracting and unproductive, as well as require a lot of time.

Basically, they would have to quickly look through all of the messages that were sent while they were away, since they would have no means to “prune” irrelevant discussions. The main issue here is the fact that they would not know whether a particular subset of messages is useful until they read at least some of them.

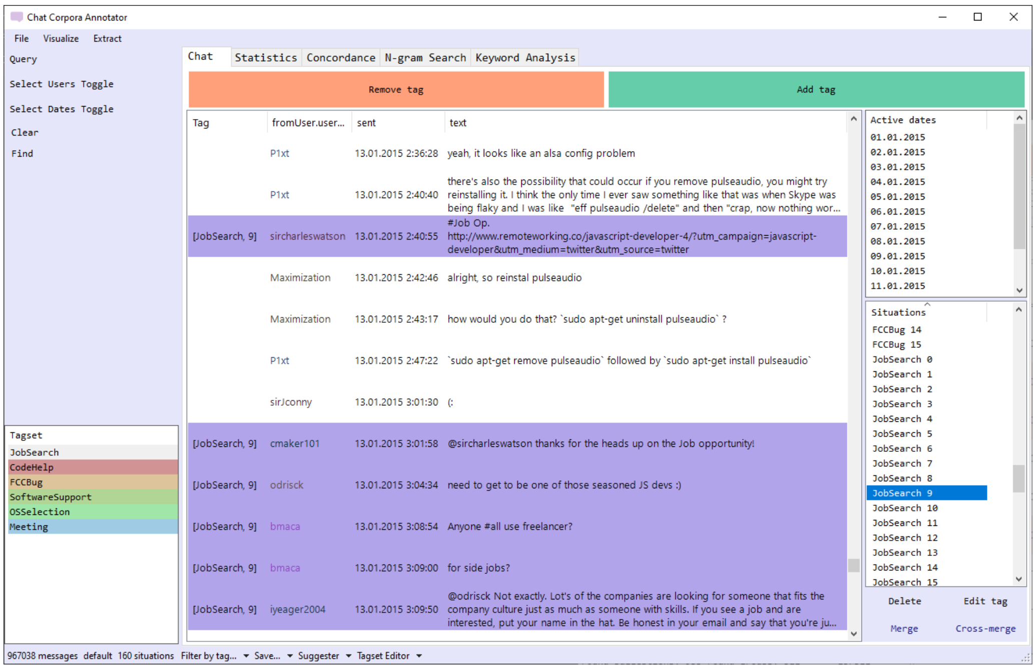
Another frequent scenario is a user searching for a particular conversation that is hard to find. Usually, in this case user issues search queries trying different keywords. In general, chats offer unsophisticated search capabilities, limiting them to simplified textual search, thus hindering efficient retrieval. For example, if the user needs to recall the details of a meeting (for example, the name of the place their friends have agreed to go out to), they would to issue “bar”, “pub”, “restaurant” until they obtain the desired result.

We propose to address such use-cases with chat log summarization that is based on the concept of *situation*. We define a *situation* as a subset of messages revolving around a single event in both temporal and contextual senses. We can propose many various examples: participants are arranging a meeting, selecting a product to be used in their project, solving a code issue and so on.

An example of a *situation* that has been found and tagged with the use of our tool is presented in Fig. 1. In this figure, the user is shown a *situation* in which chat participants find a job offer and discuss the process of applying to it.

Our final goal that exceeds the scope of this paper is to build a system that would automatically detect such *situations* and present them to the user.

The idea is to integrate the hypothetical tool into the interface of multiparticipant chat applications to provide the user with the means to take a *situation*-based perspective on chat history, instead of plain-text browsing as it has to be done currently.



Tag	fromUser.user...	sent	text
	P1xt	13.01.2015 2:36:28	yeah, it looks like an alsa config problem
	P1xt	13.01.2015 2:40:40	there's also the possibility that could occur if you remove pulseaudio, you might try reinstalling it. I think the only time I ever saw something like that was when Skype was being flaky and I was like "eff pulseaudio /delete" and then "crap, now nothing wor...
[JobSearch, 9]	sircharleswatson	13.01.2015 2:40:55	#Job Op. http://www.remoteworking.co/javascript-developer-4/?utm_campaign=javascript-developer&utm_medium=twitter&utm_source=twitter
	Maximization	13.01.2015 2:42:46	alright, so reinstal pulseaudio
	Maximization	13.01.2015 2:43:17	how would you do that? `sudo apt-get uninstall pulseaudio` ?
	P1xt	13.01.2015 2:47:22	`sudo apt-get remove pulseaudio` followed by `sudo apt-get install pulseaudio`
	sirJconny	13.01.2015 3:01:30	(:
[JobSearch, 9]	cmaker101	13.01.2015 3:01:58	@sircharleswatson thanks for the heads up on the Job opportunity!
[JobSearch, 9]	odrisck	13.01.2015 3:04:34	need to get to be one of those seasoned JS devs :)
[JobSearch, 9]	bmaca	13.01.2015 3:08:54	Anyone #all use freelancer?
[JobSearch, 9]	bmaca	13.01.2015 3:09:00	for side jobs?
[JobSearch, 9]	iyeager2004	13.01.2015 3:09:50	@odrisck Not exactly. Lot's of the companies are looking for someone that fits the company culture just as much as someone with skills. If you see a job and are interested, put your name in the hat. Be honest in your email and say that you're ju...

Figure 1: CCA's main window with a tagged situation visible.

In its first version, we plan to highlight situation locations in the history, and then, in the future, present a generated summary.

Automatic extraction of *situations* can be performed using information retrieval, natural language processing (Murray et al., 2018), and machine learning (Carenini and Murray, 2012) techniques. However, currently, there are no corpora to train models for this problem and no gold-standard datasets to run experiments on. Therefore, our first step is to create such a corpus and for this a special tool that assists tagging is needed.

3 Chat Corpora Annotator: System Overview

Following the aforementioned considerations, we have created the Chat Corpora Annotator (CCA) — the first exploration-annotation tool designed specifically for multiparticipant chat log data. Its main use case is creating a dataset for the proposed summarization task.

Furthermore, the provided functionality can help gain clear and immediate insights into raw data. CCA implements all common statistics and exploration tools and does not require any coding skills to use them. Additionally, CCA's CSV viewer is more comfortable to use than, for example, the representation of a dataset that can be created with `pandas`[3] in a Jupyter Notebook. The user can resize and swap columns in the window without affecting the data. CCA can be used for any textual data that contains a date, a username, and a text field, for example, email threads, Twitter logs, etc. Finally, all performance-heavy functionality is implemented separately from the main module and simply searching through a dataset does not require the user to load the CoreNLP models.

3.1 Features and User Interface

CCA's feature set has been inspired by linguistic-oriented tools, which were traditionally intended for a single researcher reading through the data and manually creating a linguistic corpus (Weisser, 2016). However, we have also taken into account the recent developments in the field, such as the simplicity and usability of modern annotators.

The main screen of CCA can be seen in Fig. 1. The user can upload CSV files and read through them, jump through available dates, and use the Lucene full-text search capabilities, as well as the analysis tools:

- **Statistics.** This menu item contains simple corpus statistics and visualizing functionality. Currently available: the number of messages, unique usernames, tokens, noun phrases, as well as the average length of a message, average messages per day, and average token length.

[3]`https://pandas.pydata.org/`

- **N-gram search.** This is a simple tool inspired by Google Ngram Viewer. On its first run it builds a B^+-tree disk-backed index for shingles (Manning et al., 2008) of length from 2 to 5. This tool allows the user to query the index with a single term efficiently and see the frequency of each shingle that contains it.
- **Concordancer.** This is a simple concordancer akin to `nltk`'s (Bird et al., 2009) `concordance()`: the search is constrained to a single term, which is then displayed with its immediate context. The user can select the number of characters that surround the term.

All of these tools are intended for the same purpose: they provide different angles on the topics of discussion in the dataset. For example, searching for the word "help" in the Ubuntu Chat Dataset (Uthus and Aha, 2013) reveals that, indeed, it is tech support chat data.

Additionally, simple visualizations options are provided. At the moment, there are two: a chart for message counts by date and a heatmap for message density by date.

3.2 Query Language

3.2.1 Idea

In this section, we will discuss Matcher, our custom SQL-like query language created for annotation recommendations and rich data exploration.

In modern systems such as (Cejuela et al., 2014), annotation recommendations are usually provided by machine learning models. There are no such models for situation extraction yet, and this has motivated us to adopt a different approach: we provide the users with complex querying functionality. Our approach was inspired by rule-based information extraction systems (Chiticariu et al., 2010, 2013).

Our idea was to allow the user to query the corpus for occurrences of special entities while defining their surroundings. In essence, the approach we have taken is rule-based pattern-matching. It is inspired by the Boolean retrieval model (Manning et al., 2008).

Running such queries in an ad-hoc manner is a powerful and versatile way of dataset exploration. A user can pose a query to check their annotation work, browse the results, refine the query by adding or removing conditions and run it again, effectively fine-tuning their work.

Designing Matcher, we aimed to create an intuitive, simple language that would be easier to learn for non-programmers. SQL seemed to us a suitable choice: so, we have created Matcher as an SQL-like language. Our query editor provides two modes of entering queries: free-text and a visual query builder (as seen on top of Fig. 2), which highlights the operators that would be appropriate to use next.

Matcher is implemented with the ANTLR parser generator[4].

3.2.2 Formalization

The general syntax of a Matcher query is as follows:

SELECT $cond_1^1, \ldots, cond_{n_1}^1$ INWIN $wsize_1$; $cond_1^2, \ldots, cond_{n_2}^2$ INWIN $wsize_2$; $\ldots, cond_1^m, \ldots, cond_{n_m}^m$ INWIN $wsize_m$.

We call each of $cond_1^i, \ldots, cond_{n_i}^i$, $i \in [1 \ldots m]$ a *matching group* and an individual $cond_j^i$ a *matcher*.

A *matcher* is a template that is matched against a single reply in chat history. It consists of a boolean expression which is sequentially (i.e., in a chronological order) checked against each message. If it evaluates to true, then this line is considered to be a part of the answer.

Each *matcher* that follows some $cond_j^i$ searches for the next line that satisfies its corresponding condition $cond_{j+1}^i$. This message does not necessarily have to immediately follow the previous one.

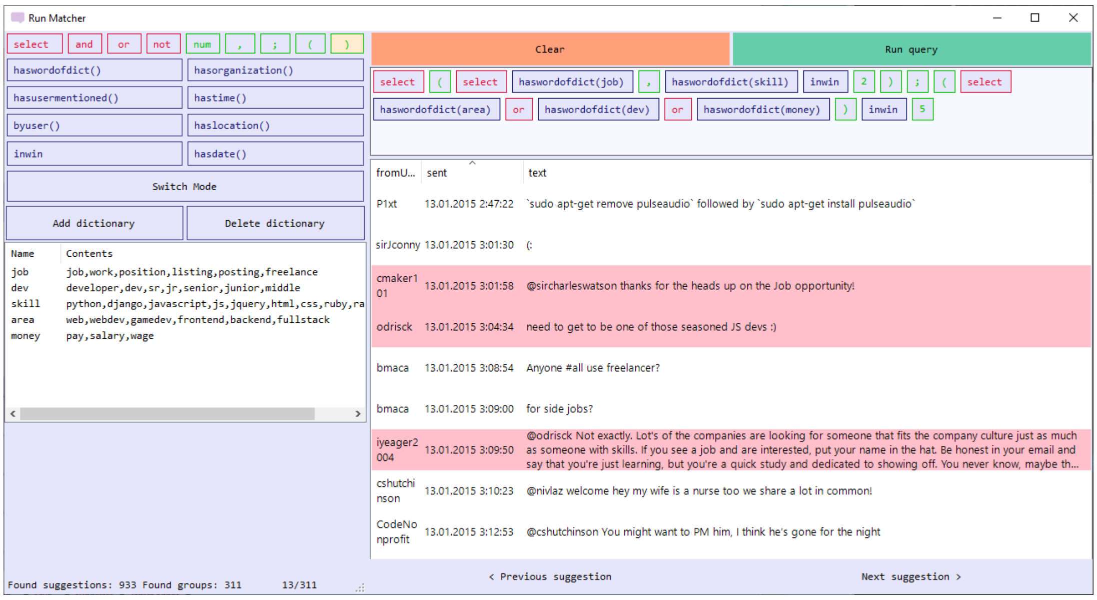
A single $cond_j^i$ consists of a set of atomic predicates joined by Boolean operators. Atomic predicates check the message for simple conditions, such as either the presence of any word from a user-defined word list (`haswordofdict()`, see Fig. 1) or an extracted NER tag (`hastime()`, `haslocation()`, etc). In order to obtain the NER data, we have implemented the CoreNLP pipeline within our tool. The full list of atomic predicates and other operators can be found in the Github README.

For example, consider the following query:

```
SELECT (haswordofdict(meetings)
AND hastime()),
haswordofdict(agreements)
```

It returns all conversations which start with a message containing any word from a user-defined "meetings" dictionary (meeting-related words) and contains a time marker. The conversation has to end with any message that has a word from the

[4]https://www.antlr.org/



Name	Contents
job	job,work,position,listing,posting,freelance
dev	developer,dev,sr,jr,senior,junior,middle
skill	python,django,javascript,js,jquery,html,css,ruby,ra
area	web,webdev,gamedev,frontend,backend,fullstack
money	pay,salary,wage

```
select ( select haswordofdict(job) , haswordofdict(skill) inwin 2 ) ; ( select
haswordofdict(area) or haswordofdict(dev) or haswordofdict(money) ) inwin 5
```

fromU...	sent	text
P1xt	13.01.2015 2:47:22	\`sudo apt-get remove pulseaudio\` followed by \`sudo apt-get install pulseaudio\`
sirJconny	13.01.2015 3:01:30	(:
cmaker101	13.01.2015 3:01:58	@sircharleswatson thanks for the heads up on the Job opportunity!
odrisck	13.01.2015 3:04:34	need to get to be one of those seasoned JS devs :)
bmaca	13.01.2015 3:08:54	Anyone #all use freelancer?
bmaca	13.01.2015 3:09:00	for side jobs?
iyeager2004	13.01.2015 3:09:50	@odrisck Not exactly. Lot's of the companies are looking for someone that fits the company culture just as much as someone with skills. If you see a job and are interested, put your name in the hat. Be honest in your email and say that you're just learning, but you're a quick study and dedicated to showing off. You never know, maybe th...
cshutchinson	13.01.2015 3:10:23	@nivlaz welcome hey my wife is a nurse too we share a lot in common!
CodeNonprofit	13.01.2015 3:12:53	@cshutchinson You might want to PM him, I think he's gone for the night

Found suggestions: 933 Found groups: 311 13/311

Figure 2: A dedicated visual interface for the Matcher query language. The depicted query is intended for retrieval of JobSearch situations.

"agreements" dictionary (words used to give consent). Thus, this query tries to extract all situations of participants scheduling a meeting.

The problem with this query is evident: while it will return the required conversation, it will also return a lot of unrelated messages since it does not restrict the position of the last message. To address this, we have introduced an optional clause INWIN $wsize$, where $wsize$ is a positive integer. It requires that all *matchers* from the *matching group* affected by the INWIN clause fit in a window of at max $wsize$ messages. Therefore, the proper query looks like this:

```
SELECT (haswordofdict(meetings)
AND hastime()),
haswordofdict(agreements)
INWIN 10
```

The purpose of the INWIN clause is not only to restrict the maximum length of the desired conversation fragment. Using it allows to query for a sequence of messages in which each message immediately follows another, i.e. without allowing other messages in between. This functionality comes naturally, if the length of the window is specified to be equal to the number of *matchers* in a given group. It stems from the rule that each *matcher* should correspond to exactly one message in each of the resulting fragments.

In Matcher, a query may have several *matching groups*. In this case, the action of the next INWIN clause starts from the last message of the previously *matched group*. Finally, we have to note that using the INWIN clause is optional for the last *matching group*.

3.2.3 Real query example

The following query was issued by the annotator during the creation of our corpus. Its purpose is to locate situations where participants discuss the current job market in programming, finding and discussing appropriate job postings for themselves. A fragment of the found results can be seen in Fig 1.

```
SELECT
    (SELECT
        haswordofdict(job),
        haswordofdict(skill)
    INWIN 2);
    (SELECT haswordofdict(area)
        OR haswordofdict(dev)
        OR haswordofdict(money))
INWIN 5
```

This query states that the annotator would like to see two messages which contain words from the "job" and "skill" dictionaries respectively, and the distance between them should be less than 2 messages (first inner query). After that, the second inner query will retrieve a third message which contains any word from either "area","dev" or "money" and is not farther away from the first one by more than 5 messages.

Note that Matcher functionality is intended for assistance only, and it should not be considered as the primary means of annotation. The reasoning is simple: due to the inherently variable and noisy nature of chat logs there are no guarantees that the found situations are valid. The results require manual checking. Moreover, there are no guarantees that all relevant situations contained in the logs would be found by a given query (for example, a user-defined dictionary might not contain a specific word that is used in logs). That is, speaking in terms of information retrieval, there are no guarantees on both precision and recall (Manning et al., 2008). This is why our query processor does not recommend tags outright, it only points out the approximate locations of interest to the user in the data.

Concluding this section, we state that the proposed approach is simpler to use than ML recommenders. Our reasoning is that the results obtained during this process are straightforward, while an ML model can produce results as a "black box": the user would have no understanding as to why certain messages are assigned certain labels.

3.3 Annotation Guidelines and Workflow

The annotation guidelines are currently simple and revolve around situation definition, which was given in Section 1. The annotator either receives instructions before tagging or personally devises a tagset during data exploration. Next, they manually read through the data, extracting and annotating subsets of messages as situations.

Concerning the annotation model, we have created it to be more flexible than just assigning a single tag to a sequential subset of messages. Each message can belong to several differently-typed situations, but cannot belong to two different situations of the same type. We rely on the assumption that chat messages are short and the users generally keep them constrained to one topic. However, as the topics shift quickly in multiparticipant chat, the users can try and catch up by compacting information concerning different topics in one message.

Figure 3 contains the workflow we provide for our tool. As it can be seen, the entirety of the data preparation process is done inside CCA. The user receives a semi-structured data file and loads it into the tool. The user then explores it with the analytic search tools (searches through n-grams, issues simple queries, and so on), as well as utilizes Matcher, either coming up with their own dictionaries and queries or importing them. During this process, they simultaneously annotate the data and amend the tagset if required. Finally, they save the resulting output as an XML file.

4 Gold-Standard Corpus

4.1 Corpus Development and Statistics

In order to test CCA's functionality, we have created an annotated situation corpus for the freeCodeCamp dataset[5]. The fragment of the dataset that we used contains $967,038$ messages spanning over 381 days, sent by 29870 unique users.

The constructed corpus contains 236 tagged situations, comprising 4146 messages in total. On average, our situations are 17 messages long. The average length of a message in the corpus is 78 symbols, in contrast to the dataset average of 66 symbols. The average number of users participating in a situation is 3.

Our tagset comprises 6 tags, as can be seen in the list below. They describe a common situation encountered in this particular dataset: e.g., CodeHelp is a user pasting in a faulty code fragment and receiving help. The tags have been manually devised after dataset exploration, and each of them has yielded the following numbers:

- JobSearch: 24 situations, 4 users and 13 messages on average
- CodeHelp: 95 situations, 3 users and 18 messages on average
- SoftwareSupport: 53 situations, 3 users and 22 messages on average
- OSSelection: 19 situations, 4 users and 16 messages on average
- Meeting: 4 situations, 2 users and 12 messages on average
- FCCBug: 42 situations, 3 users and 14 messages on average

Additionally, we have considered *windows* in situations (i.e., gaps containing untagged unrelated messages inside situations) and *intertwined situations* (two or more situations which intersect). The entire number of windows in our corpus is 820, which makes every situation have around 3 windows on average. The average length of a window

[5]`https://www.kaggle.com/freecodecamp/all-posts-public-main-chatroom`

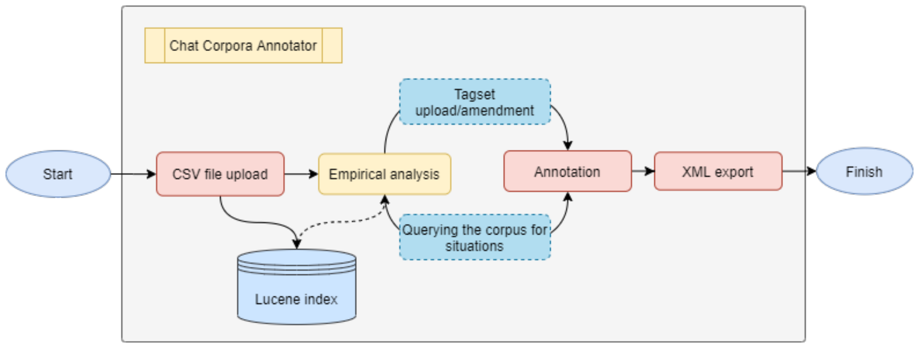


Figure 3: CCA workflow.

is 5 messages. Furthermore, out of 236 total situations, 60 are intertwined.

The creation of the corpus took a single annotator who was acquainted with the task around 20 work hours. They have utilized all available tools, but used Matcher the most. As they have reported, Matcher functionality was very valuable, as reading through a million messages would have been impossible. Additionally, they reported that running even very simple, single-term queries helped navigate the chat log data more efficiently by providing a paginated view of the dataset, coupled with highlighting of relevant messages.

4.2 Inter-Annotator Agreement

Only one annotator was employed during the creation of the corpus, so we did not run into situations that called for conflict resolution.

Going forward, we envision an interface for manual resolution that would allow to compare output files from different annotators either against each other and all at once. The annotator responsible for the comparison should be able to extend or shrink the boundaries of a situation, remove or add single messages, etc. Furthermore, we will implement the computation of various inter-annotator statistics such as Cohen's Kappa (Manning et al., 2008) in order to provide the user with formal means of evaluating the intermediate results.

5 Evaluation

We have conducted two kinds of evaluation tests: a responsiveness study and a usability study.

5.1 Tool Responsiveness

Raw chat log dataset files can be as large as several gigabytes, therefore, we have developed our application taking this into account.

Metric	Results
N-gram indexing	4 minutes
Indexing	1 minute
Heatmap rendering	0.5 s
Jumping dates	less than 0.1s
Opening an indexed file	less than 0.1s
Search query	less than 0.1s
Simple Matcher query	less than 0.1s
Complex Matcher query	around 0.1s

Table 1: Experiments on CCA responsiveness.

Table 1 presents the results we have obtained. We have measured the time it takes CCA to perform crucial operations on a large data file. The setup was as follows: we used CCA on a mid-range home PC running Windows 10 (Intel i5-7600k, 16GB DDR4 RAM, Crucial MX500 500GB SSD), manipulating a 500MB CSV file that contained around 1M chat messages. We adhere to the well-known quote of Jakob Nielsen (Nielsen, 1994): “0.1 second is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result”. As it can be seen, our system is responsive and only takes up a considerable amount of time on tasks that are run once, such as indexing or extracting key phrases, which could also be improved further in the next versions of our system.

5.2 Usability Study

We have run a small usability evaluation with three volunteers. We have explained the annotation task to them, and then asked them to load the tool, index a small CSV file, explore the data and annotate it using our standard tagset. Next, we have conducted a short informal discussion on the tool's interface, responsiveness and feasibility for the task at hand. The users have reported that the task was understandable to them, although it did require a little time to grasp, and the system appeared convenient for reading and searching through large volumes of data. They have proposed the following improvements: developing concise documentation for the system's capabilities, improving the cohesiveness of the UI, and finally, we have asked them to fill out the System Usability Scale questionnaire (Brooke, 1986), which has been slightly modified to fit our system better. Namely, we have modified questions 1 and 9, to "I find the system adequate for the proposed task" and "I could use the system to confidently complete the proposed task" respectively. This has been done since our system is intended for several specific tasks that arise in a research setting, not in daily life. The answers have put CCA at the 50th percentile, which indicates an "OK" level of usability (Sauro, 2018). Going by the responses we have obtained, CCA was easy enough to use, but it lacks better feature integration and perhaps a short tutorial. We consider this an adequate result for a first prototype, however, we will focus our future efforts on improving it.

6 Related Work

In this section, we will review two types of related studies: annotation tools and corpora created from chat log data for various tasks.

6.1 Annotation Tools

As mentioned previously, annotating raw text chat data is a complicated task due to its specifics. In this section, we will go over several well-known annotating tools and frameworks and evaluate their feasibility for the task at hand.

`brat` (Stenetorp et al., 2012) is a flexible all-purpose annotation tool. It supports two modes of annotation: annotating a text span with a label, and connecting these labels with either directed or undirected binary relations. Furthermore, the second mode also includes n-ary relations and attributes of these relations. Finally, brat supports an extensive constraint system for relations and an advanced search system. Due to it being based on a dedicated visualization system, brat was one of the first tools that provided its users with intuitive high-quality annotation visualization. However, as noted by Kummerfeld (2019), it takes considerable effort to set up for any custom task, including ours.

GATE (Bontcheva et al., 2013) and UIMA (Ferrucci and Lally, 2004) are well-known analysis frameworks that have been developed since the early 00's. While they are powerful, customizable and could be extended to suit any task, they are not easy to set up and utilize "on-the-go" — a feature that is essential for many modern tasks. For example, the creators of the Tweebank v1 dataset (Owoputi et al., 2013) admit to creating it in a single day. While it is not claimed to be a gold-standard dataset, the speed is impressive, and the team has used their own dedicated annotation tool. However, with these frameworks, the user would have not only to read through the extensive manuals, but also, most likely, code their own tools in Java[6].

TWIST (Pluss, 2012) and LIDA (Collins et al., 2019) are intended for dialogue annotation, which has been mostly focused on task-oriented dialogue for dialogue systems. Task-oriented dialogues already suppose a predefined topic and predefined roles (e.g., customer support tasks) and little noise. These tools provide their user with functionality such as turn/dialogue segmentation. They also impose constraints on the data, such as requiring only two speakers to be present in the dataset, which already makes them unsuitable for our task. Finally, they do not implement any full-text or constrained search features, which makes data exploration nearly impossible.

TagTog (Cejuela et al., 2014) and LightTag[7] are modern Web-based annotators that advertise flexibility for any task. While they are flexible and require little set up time, they also do not feature any search or exploration functionality in their free versions. Usually, these tools let the user view the data one line at a time, which is simply unfeasible for the task. Although it is possible to set up LightTag to display the "context" of the current message, it is still a constrained view. Further, these tools are oriented at fairly monotonous work such as building a NER dataset with custom tags, and this is why they tightly integrate ML recommenders into their workflow. This is helpful for well-known classification tasks, but it is not a feasible approach for something novel, i.e. that lacks trained models. SLATE by Kummerfeld (2019) is an experimental annotation tool focused on a terminal-based workflow that was released in 2019. Its authors argue that its main advantages are: complete configurability for any task and annotation speed which is not

[6] `https://uima.apache.org/doc-uima-annotator.html`

[7] `https://www.lighttag.io/`

hindered by GUI. Concerning the second point, this tool is controlled via keyboard shortcuts instead of a mouse, and all of its UI is contained within a Linux terminal. It supports annotation of continuous spans of any entities, such as characters, tokens, lines, or documents. Additionally, SLATE supports linking any of these entities. It was specifically designed to create large corpora out of chat and chat-like data in a very short time. This goal has been achieved, however, SLATE does not offer any exploration functionality. Furthermore, its learning curve may be steep for someone who is not used for a keyboard-based workflow.

Finally, we would like to mention Huggingface Dataset Viewer[8], which is a web-based tool for manually looking through NLP datasets from the Huggingface `nlp` library. While it is not an annotator and cannot be directly compared to our or other tools, its existence proves that there is a need to explore a dataset before using it for any task.

As it can be seen, there are no tools that could be readily applied or easily customized for our task. Existing options either lack the desired functionality or require a substantial, often comparable to creating a new tool from scratch, effort in order to make them suitable for the considered task.

6.2 Chat Datasets

Most of the existing annotated chat log datasets are intended for the chat disentanglement task. The first known corpora belongs to Shen et al. (2006), who have drawn their data from an intra-university IRC channel. This dataset was not public. Further on, some of the most well-known work in this area belongs to Elsner and Charniak (2008) who have created a corpora for chat disentanglement based on IRC logs of the `#Linux` channel at `free-node.org`. They have manually annotated around two thousand utterances via a dedicated interface. However, to the best of our knowledge, the data has since ceased to be publicly available. Adams and Martell (2008) developed a disentanglement and topic extraction dataset based on Navy tactical chat which was not released. However, the most well-known dataset belongs to Lowe et al. (2015): they have created the Ubuntu Dialogue Dataset based on IRC data from the `Ubuntu` help channel. It contains around a million of heuristically extracted multi-turn dialogues, and it can be accessed online. A dataset based on the French version of the same channel was presented by Riou et al. (2015), containing 1229 messages. Dulceanu (2016) presents a small dataset of manually collected 884 chat messages which were disentangled and annotated with three speech acts. Finally, Kummerfeld et al. (2019) present the largest disentanglement corpus to date: it contains around 78 thousand manually annotated messages also from the Ubuntu and Linux IRC channels.

Concerning other tasks, we would like to mention Tweebank v2 (3550 tweets) by Liu et al. (2018), which was created for training a full machine learning based NLP pipeline. Its first version by Owoputi et al. (2013) contained 840 tweets tagged for training a part-of-speech tagger.

To the best of our knowledge, very few summarization datasets for chat and chat-like data were made publicly available. The AMI corpus (Carletta et al., 2005) contains transcripts of audio drawn from business meetings, hand-annotated with their abstractive and extractive summaries among many other annotation modes. Further on, Joty et al. (2010) have developed the BC3 corpus that contains email and blog data for summarization. Koto (2016) take the same approach and present a summarization dataset for chats in the Indonesian language, consisting of 300 manually summarized chat segments.

As it can be seen, no attempts on creating annotated corpora from the freeCodeCamp data have been made to date, and our work is the first to attempt that.

7 Conclusion & Future Work

In this paper, we have presented a novel situation-based summarization task, CCA — an annotation-exploration tool for large chat logs, a workflow for creating a situation-based summarization dataset, and an example corpus. Chat Corpora Annotator offers a novel approach to exploration: a query language that allows the user to query a dataset for subsets of messages which could be a situation. To the best of our knowledge, CCA is the only tool designed for these two tasks at once.

Further work on the tool will be focused on improving its usability and efficiency, as well as extending language support. The work on the summarization task will be moving towards implementing the first versions of the summarizer itself.

[8]`https://huggingface.co/nlp/viewer/`

References

Paige H. Adams and Craig H. Martell. 2008. Topic detection and extraction in chat. In *2008 IEEE International Conference on Semantic Computing*, pages 581–588.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc.

Kalina Bontcheva, Hamish Cunningham, Ian Roberts, Angus Roberts, Valentin Tablan, Niraj Aswani, and Genevieve Gorrell. 2013. Gate teamware: a web-based, collaborative text annotation framework. *Language Resources and Evaluation*, 47(4):1007–1029.

John Brooke. 1986. System usability scale (sus): a quick-and-dirty method of system evaluation user information. *Reading, UK: Digital Equipment Co Ltd*, 43.

Giuseppe Carenini and Gabrial Murray. 2012. Methods for mining and summarizing text conversations. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, page 1178–1179, New York, NY, USA. Association for Computing Machinery.

Jean Carletta, Simone Ashby, Sebastien Bourban, Mike Flynn, Mael Guillemot, Thomas Hain, Jaroslav Kadlec, Vasilis Karaiskos, Wessel Kraaij, Melissa Kronenthal, et al. 2005. The ami meeting corpus: A pre-announcement. In *International workshop on machine learning for multimodal interaction*, pages 28–39. Springer.

Juan Miguel Cejuela, Peter McQuilton, Laura Ponting, Steven J Marygold, Raymund Stefancsik, Gillian H Millburn, and Burkhard Rost. 2014. tagtog: interactive and text-mining-assisted annotation of gene mentions in plos full-text articles. *Database*, 2014.

Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. 2010. SystemT: An algebraic approach to declarative information extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 128–137, Uppsala, Sweden. Association for Computational Linguistics.

Laura Chiticariu, Yunyao Li, and Frederick R. Reiss. 2013. Rule-based information extraction is dead! long live rule-based information extraction systems! In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 827–832, Seattle, Washington, USA. Association for Computational Linguistics.

Edward Collins, Nikolai Rozanov, and Bingbing Zhang. 2019. LIDA: Lightweight interactive dialogue annotator. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 121–126, Hong Kong, China. Association for Computational Linguistics.

Andrei Dulceanu. 2016. Recovering implicit thread structure in chat conversations. *Revista Romana de Interactiune Om-Calculator*, 9(3):217–232.

Micha Elsner and Eugene Charniak. 2008. You talking to me? a corpus and algorithm for conversation disentanglement. In *Proceedings of ACL-08: HLT*, pages 834–842, Columbus, Ohio. Association for Computational Linguistics.

David Ferrucci and Adam Lally. 2004. Uima: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10:327–348.

Shafiq Joty, Giuseppe Carenini, Gabriel Murray, and Raymond T. Ng. 2010. Exploiting conversation structure in unsupervised topic segmentation for emails. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 388–398, Cambridge, MA. Association for Computational Linguistics.

Shafiq R. Joty, G. Carenini, Gabriel Murray, and R. Ng. 2011. Supervised topic segmentation of email conversations. In *ICWSM*.

Fajri Koto. 2016. A publicly available Indonesian corpora for automatic abstractive and extractive chat summarization. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 801–805, Portorož, Slovenia. European Language Resources Association (ELRA).

Jonathan K. Kummerfeld. 2019. SLATE: A super-lightweight annotation tool for experts. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 7–12, Florence, Italy. Association for Computational Linguistics.

Jonathan K. Kummerfeld, Sai R. Gouravajhala, Joseph J. Peper, Vignesh Athreya, Chulaka Gunasekara, Jatin Ganhotra, Siva Sankalp Patel, Lazaros C Polymenakos, and Walter Lasecki. 2019. A large-scale corpus for conversation disentanglement. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3846–3856, Florence, Italy. Association for Computational Linguistics.

Yijia Liu, Yi Zhu, Wanxiang Che, Bing Qin, Nathan Schneider, and Noah A. Smith. 2018. Parsing tweets into Universal Dependencies. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 965–975, New Orleans, Louisiana. Association for Computational Linguistics.

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. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 285–294, Prague, Czech Republic. Association for Computational Linguistics.

Vanessa Loza, S. Lahiri, R. Mihalcea, and P. Lai. 2014. Building a dataset for summarization and keyword extraction from emails. In *LREC*.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, USA.

Gabriel Murray, Giuseppe Carenini, and Shafiq Joty. 2018. NLP for conversations: Sentiment, summarization, and group dynamics. In *Proceedings of the 27th International Conference on Computational Linguistics: Tutorial Abstracts*, pages 1–4, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Jakob Nielsen. 1994. *Usability engineering*. Morgan Kaufmann.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. 2013. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–390, Atlanta, Georgia. Association for Computational Linguistics.

Brian Pluss. 2012. TWIST dialogue annotation tool.

Matthieu Riou, Soufian Salim, and Nicolas Hernandez. 2015. Using discursive information to disentangle french language chat. In *2nd Workshop on Natural Language Processing for Computer-Mediated Communication (NLP4CMC 2015)/Social Media at GSCL Conference 2015*, pages 23–27.

Alan Ritter, Oren Etzioni, and Sam Clark. 2012. Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1104–1112.

Jeff Sauro. 2018. 5 ways to interpret a sus score. `https://measuringu.com/interpret-sus-score/`. [Online; accessed 20-November-2020].

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*, SIGIR '06, page 35–42, New York, NY, USA. Association for Computing Machinery.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. brat: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107, Avignon, France. Association for Computational Linguistics.

Naama Tepper, Anat Hashavit, Maya Barnea, Inbal Ronen, and Lior Leiba. 2018. Collabot: Personalized group chat summarization. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 771–774, New York, NY, USA. Association for Computing Machinery.

Jan Ulrich, Gabriel Murray, and Giuseppe Carenini. 2008. A publicly available annotated corpus for supervised email summarization. In *Proceedings of AAAI Email-2008 workshop, Chicago, USA*.

David C Uthus and David W Aha. 2013. The ubuntu chat corpus for multiparticipant chat analysis.

Martin Weisser. 2016. Dart – the dialogue annotation and research tool. *Corpus Linguistics and Linguistic Theory*, 12:355 – 388.