Dialog Data Collection for an Interactive Plotting Agent

Yutong Shao and **Ndapa Nakashole**
Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093
yshao@eng.ucsd.edu, nnakashole@eng.ucsd.edu

Abstract

Plotting is a key tool for making data readable and for exposing useful trends in the data. In this work, we aim to enable data plotting using natural language instructions. As a first step, we are developing a data collection pipeline for user-agent dialogs which will allow us to train a slot filling system that maps from natural language to slots and values supported by a popular plotting library, `matplotlib`.

1 Introduction

Data has become ubiquitous in everyday life. For example, mobile phones and wearable devices generate data about our activities. Businesses collect data and generate massive amounts of data. Cities and countries also provide publicly available data. This plethora of data can be utilized for the benefit of people and businesses. However, in its raw form, data is often unreadable and therefore difficult to derive insights from. Plotting is a simple yet powerful technique for making raw data readable and useful. However, current plotting tools require a certain amount of expertise and knowledge of programming. In this work, we seek to enable data plotting using natural language instructions.

We thus seek to build a *Plotting Agent*, whose goal is to take natural language input, fill the slots and carry out the plotting action using `matplotlib`. We formulate this problem as a *slot-based task-oriented dialog* problem. Example slots include *line color*, *line width*, *marker size*, *marker color*, *marker interval*, *data series name*, etc. Also, when necessary, the *Plotting Agent* may ask clarification questions to improve its understanding of the user instructions. In this way, the *Plotting Agent* and the user engage in a dialog where the user specifies how they want their data to be plotted to the *Plotting Agent*.

We are developing a dialog data collection pipeline which will allow us to train the slot filling dialog system in a supervised way. In this short paper, we describe this new task, especially the dialog data collection process.

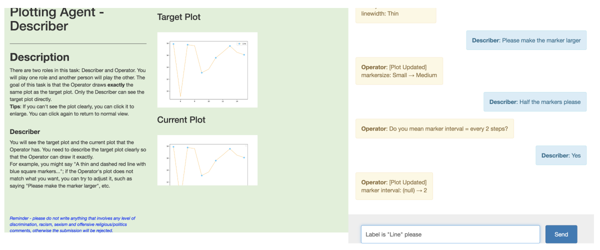
2 Dialog Collection

A widely-used dialog collection scheme is the Wizard-of-Oz (Strauß et al., 2006), in which one worker plays the user and another worker plays the agent. Our data collection pipeline also follows this scheme.

We are collecting dialogs using Amazon Mechanical Turk (MTurk). In particular, we designed our Mturk Human Intelligence Task (HIT) to have a *Describer* worker, who plays the role of the user; and an *Operator* worker, who plays the role of the plotting agent. Several screenshots of the UI for both workers are shown in Figure 1, 2, 3 and 4. The Describer has access to a "target plot" that is to be plotted, while the Operator has access to an "operation panel" which consists of one changeable field for each slot. The Operator can use this panel to send plotting request to the server and the slot values will also be recorded. Both workers have access to a "current plot" which is the latest plot that the Operator has plotted. It is initialized to a placeholder plot with default slot values, only for showing the underlying data series[1].

To start, the Describer sends a message (in natural language) to the Operator to describe the plot style that he/she sees in the target plot. The Operator can respond in natural language to ask clarification questions, or fill out some slots in the operation panel and show the current plot to the Describer. Then the two workers take turns to speak or plot, until the current plot exactly matches the target plot.

[1]This is also to ensure that the workers do not describe the data but merely focus on the "plot style".



Plotting Agent - Describer

Description

There are two roles in this task: Describer and Operator. You will play one role and another person will play the other. The goal of this task is that the Operator draws **exactly** the same plot as the target plot. Only the Describer can see the target plot directly.
Tips: If you can't see the plot clearly, you can click it to enlarge. You can click again to return to normal view.

Describer

You will see the target plot and the current plot that the Operator has. You need to describe the target plot clearly so that the Operator can draw it exactly.
For example, you might say "A thin and dashed red line with blue square markers..."; if the Operator's plot does not match what you want, you can try to adjust it, such as saying "Please make the marker larger", etc.

Reminder - please do not write anything that involves any level of discrimination, racism, sexism and offensive religious/politics comments, otherwise the submission will be rejected.

Target Plot

Current Plot

linewidth: Thin

Describer: Please make the marker larger

Operator: [Plot Updated]
markersize: Small → Medium

Describer: Half the markers please

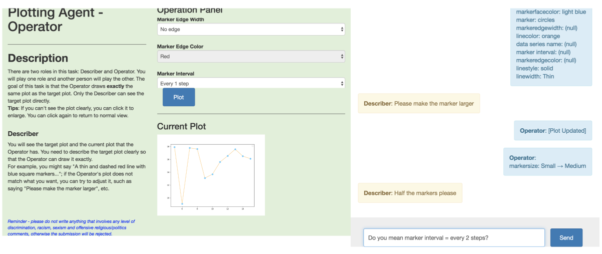
Operator: Do you mean marker interval = every 2 steps?

Describer: Yes

Operator: [Plot Updated]
marker interval: (null) → 2

Label is "Line" please

Figure 1: Describer UI



Plotting Agent - Operator

Description

There are two roles in this task: Describer and Operator. You will play one role and another person will play the other. The goal of this task is that the Operator draws **exactly** the same plot as the target plot. Only the Describer can see the target plot directly.
Tips: If you can't see the plot clearly, you can click it to enlarge. You can click again to return to normal view.

Describer

You will see the target plot and the current plot that the Operator has. You need to describe the target plot clearly so that the Operator can draw it exactly.
For example, you might say "A thin and dashed red line with blue square markers..."; if the Operator's plot does not match what you want, you can try to adjust it, such as saying "Please make the marker larger", etc.

Reminder - please do not write anything that involves any level of discrimination, racism, sexism and offensive religious/politics comments, otherwise the submission will be rejected.

Operation Panel

Marker Edge Width: No edge

Marker Edge Color: Red

Marker Interval: Every 1 step

Current Plot

markerfacecolor: light blue
marker: circles
markeredgewidth: (null)
linecolor: orange
data series name: (null)
marker interval: (null)
markeredgecolor: (null)
linestyle: solid
linewidth: Thin

Describer: Please make the marker larger

Operator: [Plot Updated]

Operator:
markersize: Small → Medium

Describer: Half the markers please

Do you mean marker interval = every 2 steps?

Figure 2: Operator UI

One concern about this pipeline is the distracting latency of response from the other worker (Wen et al., 2017), which is a general problem of Wizard-of-Oz pipelines. Also, workers need to be paired to start this HIT, which means the first worker to accept the HIT has to wait until another worker comes. If the waiting time is long, workers will lose patience and leave. To address these problems, many previous works ask each participating worker to contribute only one turn to the dialog (Wen et al., 2017; Eric and Manning, 2017). However, in our task setting, this approach may cause other problems. Firstly, as the rewards no longer depend on task success, Describers would minimize the descriptions they write, e.g. referring to only one slot in a brief way, such as only saying "yellow line" or "thick line"; if rewards are based on task success, workers would be more willing to write better descriptions. Also, Operators can always ask clarification questions like "what do you mean" instead of trying to understand the instruction and update the plot. In all, one-turn contribution can mitigate the problem of response latency and pairing latency, but it loses the advantage of paying workers upon success, which encourages workers to make real constructive actions. This is an important design choice to consider.

3 HIT Interface

We now describe the UI in more detail using screenshots.

Figure 1 shows the main UI for the Describer. In the leftmost column, we show the descriptions for this task, including the task goal, action instructions and other tips and notice. In the middle column are the target plot and current plot, so that the Describer can easily compare and find the difference. On the right side is the dialog section, which shows the whole dialog history. In the Describer's turn, he/she can type text in the input text field and send to Operator. When waiting for response, the UI will show a "waiting" message and the input text field will be disabled.

Figure 2 shows the main UI for the Operator. Similar to the Describer UI, we also have descriptions on the left and dialog section on the right. In the middle, we have the "operation panel" where the Operator can change the value of different slots, and a "plot" button to submit the plotting request. Below that is the current plot. In the Operator's turn, he/she can choose either to send a plotting request, which will update the current plot and automatically send a formatted message indicating which slots have been changed (see the dialog section in both Figure 1 and 2); or, to send a text message (clarification question) to the Describer. When waiting for response, both the "plot" button and the input text field will be disabled.

When the current plot exactly matches the target plot, the HIT ends automatically and both workers are shown a success message and the "Done with this HIT" button (see Figure 4).

4 Future Work

This is ongoing work and we are in the process of completing our dialog collection pipeline and deploying it to collect a dataset for our task. After data collection, we will implement baselines that include simple rule-based baseline models and state-of-the-art models on other similar tasks, as well as designing a specialized model for this task. Experiments will be conducted to illustrate the difficulty of this task and compare the performance of different models.

Further, if the slot-based Plotting Agent task can be well-addressed, we can extend it into more complex settings, e.g. utilizing a set of API functions in an intelligent way (instead of a single slot-based function call), which will be more similar to Program Synthesis.