Assignment 2: Chatbot Implementation Using Watson Conversation

Daanish Ahmed

DATA 650 9041

Spring 2018

dsahmed2334@yahoo.com

Professor Elena Gortcheva

UMUC

February 26, 2018

Introduction

Cognitive computing is a major development in the field of computer science. It involves developing computer systems that can process the human language and imitate human problem solving in ways that are impossible for traditional computing approaches (Gortcheva, 2018). IBM is one of the leading developers of cognitive computing technology due to the creation of their Watson computer system. Instead of simply analyzing the frequencies of certain key words, Watson can understand the context of such terms. It achieves this by extracting features from the input text, examining the massive number of stored texts to generate possible responses, comparing the responses using numerous algorithms, and choosing the response with the highest confidence level (High, 2012). IBM has developed numerous services using Watson's cognitive technology, and one of its most useful services is Watson Conversation. This tool allows developers to design a chatbot, which is a program that can interact with end users through natural language conversations. The chatbot can perform certain tasks as requested by the user, which makes it valuable for many applications in business, health, and other fields.

My goal is to use Watson Conversation to create a chatbot that can operate a fictional ice cream parlor named "Ahmed's Ice Cream and Frozen Yogurt." The purpose of the chatbot is to assist customers in performing one of two tasks: ordering a specific product online, or locating the nearest store to visit. The bot will warmly welcome each customer and ask for their name to provide a more friendly and human-like experience. From there, it will ask about the user's intent. If they wish to visit a nearby store, then the bot will ask for the state that they live in. It will provide either a list of stores in that state and their addresses, or it will tell the user that there are currently no stores open in that state. If the user wants to shop online, then the bot will ask questions about which product the user wishes to purchase. Customers are only permitted to purchase one item at a time due to high customer demand. They are allowed to purchase either ice cream or frozen yogurt, and they can choose from a variety of sizes and flavors. The prices are determined by the size of the item and whether they select ice cream or frozen yogurt (with frozen yogurt being $0.25 more expensive). Users are asked to confirm their order at the end, and they may choose to cancel the order to make necessary changes. At any point in the dialog, the user can exit the conversation by saying goodbye to the chatbot. The bot is also trained to handle additional user queries, such as asking about store hours or prices. All conversation scenarios can be seen in the dialog flow diagram (see Figure 1).

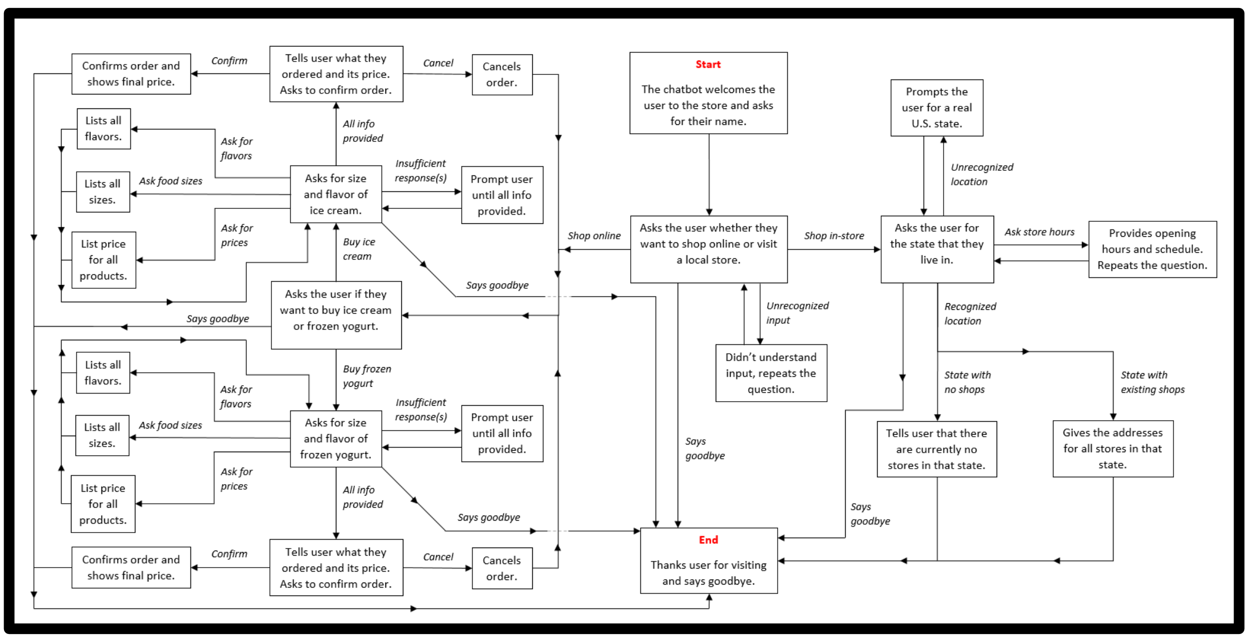

Figure 1. Dialog Flow Diagram of Ice Cream Parlor Chatbot.

There are several possible scenarios that the bot can handle. If the user wishes to visit a nearby store, then they will be asked to provide the state in which they live. They are required to give a valid location, otherwise the bot will continue to prompt them for one. Our company has stores in 15 states, including Maryland, Virginia, New York, Pennsylvania, Florida, California, and Arizona, in addition to Washington, D.C. If the user lives in one of these locations, then the bot will list the exact addresses for every shop in that state. However, the user can also ask about store hours, and the bot will mention the days and hours that our stores are open. If the user enters a location that is not among these 16 locations, then the bot will tell them that there are no stores currently operating in their state. After discussing the store locations, the bot will provide a goodbye message thanking the user for using this service.

If the customer wants to shop online, they will be taken to the company's online store where they can buy either ice cream or frozen yogurt. Frozen yogurt will cost $0.25 more than ice cream. Users will then be asked about the size and flavor of their purchase. There are 19 available flavors in four sizes: small, medium, large, and extra-large. The user is required to enter the size and flavor, otherwise they will be prompted to do so. During this stage, the bot can answer questions about the available flavors, sizes, and pricing. Products are priced according their size, not their flavor. After the user enters the order, the bot will ask for confirmation while displaying the product details and pricing. If the user confirms the order, the bot will show a confirmation message with the price the user owes. The dialog will then display the goodbye message. But if the user cancels the order, they will be taken back to the beginning of the online store. All order information and pricing will be cleaned, allowing them to make a new order from scratch. Finally, the user can cancel the transaction at any point of the dialog by saying goodbye. Overall, this chatbot is designed to provide the best possible customer experience for users.

Implementation

In this section, I will describe the methods used to implement the chatbot and its features. I used the following intents: #confirm_order, #decline_order, #food_size, #goodbye, #hours,

#prices, #shop_in_store, #shop_online, and #what_flavors. #Confirm_order and #decline_order are used by customers to either confirm or cancel their current order when shopping online. #Food_size, #what_flavors, and #prices are used when shopping online and inquiring about the available flavors, sizes, and corresponding prices. #Goodbye can be used at any point in the dialog to exit the conversation and display the goodbye message. #Hours can be called when the user is looking for a nearby store and wants to know our operating hours and weekly schedule. #Shop_online and #shop_in_store are used to indicate that the user wants to visit our online store or look for the nearest in-store location.

I also incorporated several system entities built by IBM to improve the performance of my chatbot. Although I enabled all system entities except for @sys-currency and @sys-percentage, I primarily used @sys-location and @sys-person. @Sys-location was used to search for stores within the customer's state of residence. It was used primarily to validate whether the user entered the name of a valid location. @Sys-person is needed to recognize the user's name and store their name in a context variable to be used throughout the conversation. If the user were to type "my name is Charlie," then the bot would call them "Charlie" instead of calling them "my name is Charlie." In addition, I created four custom entities: @flavors, @food_type, @locations, and @sizes. @Flavors contains all 19 available flavors and their synonyms. @Food_type contains two values: ice cream and frozen yogurt. It is used to indicate which of the two products the customer wishes to order. @Sizes contains the four available sizes: small, medium, large, and extra-large. Finally, @locations consists of the states in which the company has existing stores. Since the @sys-location entity is case-sensitive, I created another entity that makes it easier to recognize certain locations. I used these two entities together to help the chatbot recognize whether the input is a valid location and whether we have a store there.

After creating intents and entities, I can now describe the dialog itself. The first node is a welcome node that greets the user with one of five random greetings and asks for their name. If the user calls the #goodbye intent, then the dialog will move to a child node that jumps to the goodbye node—which displays one of six possible goodbye messages in random order. The goodbye message will be displayed at any point in the dialog when the user calls the #goodbye intent. However, if the user responds with their name (or any other input), then the dialog will progress to the “ask for name” node. Here, the bot will try to identify the name using the @sys-person entity. If it recognizes the name, then it will be stored as a variable called $username. If it does not recognize the name, then it will store a null string (or empty string) as the user’s name.

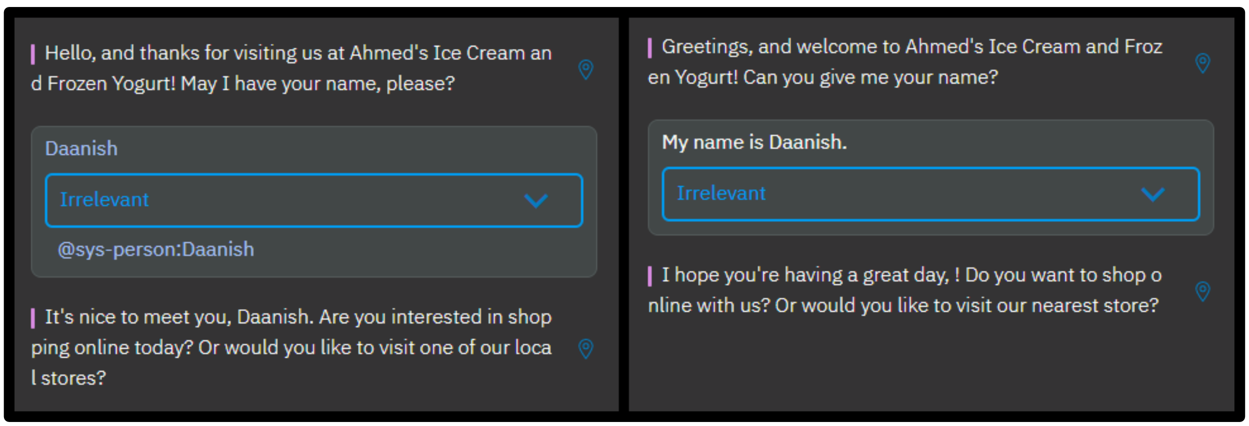

Figure 2. Issues With @Sys-Person Name Identification.

One of the bot’s limitations is that the @sys-person entity is in Beta development, meaning that it has issues identifying names in certain sentences. Some names can be recognized when written in all lower case, while other names cannot. Furthermore, certain names (such as my name, “Daanish”) are correctly identified when written alone, but are not identified when used in a sentence such as “my name is Daanish (see Figure 2)." One possible solution would be to create a custom context variable that stores the user input as the name without using the @sys-person entity. This would prevent the dialog from displaying an empty string as the user's name. But the issue with using this approach is that the entire user input would be stored as the customer's name. If the user typed "My name is John," then the bot would call him "My name is John" instead of "John." Another solution I attempted was to use both the @sys-person entity and a separate context variable $name. If the bot recognized the name, then it would use the $username context variable. Otherwise, it would store the input into the $name variable. This approach did not work, but I believe further experimentation could produce the desired results.

After asking for the user's name, the bot will ask whether the user wants to shop online or visit an in-store location. If the user calls the #shop_in_store intent, the dialog will go to the "shop in-store" node. The bot will then ask the user to enter the state that they live in so that it can find the stores located in that state. Once the user responds, the dialog will progress to the "locations" node. This node uses a slot that checks whether the user entered a valid location. If the user did not, then the bot will prompt the user to do so. Otherwise, it will store the provided location in a context variable named $location. Furthermore, it uses handlers to deal with specific user requests. If the user calls the #goodbye intent, then the program will skip to the response and display the goodbye message. But if they ask for store hours, the bot will give details about the weekly schedule. The user will still have to provide their location afterwards. Examples of handling unexpected input can be seen in the "try it out" panel (see Figure 3).

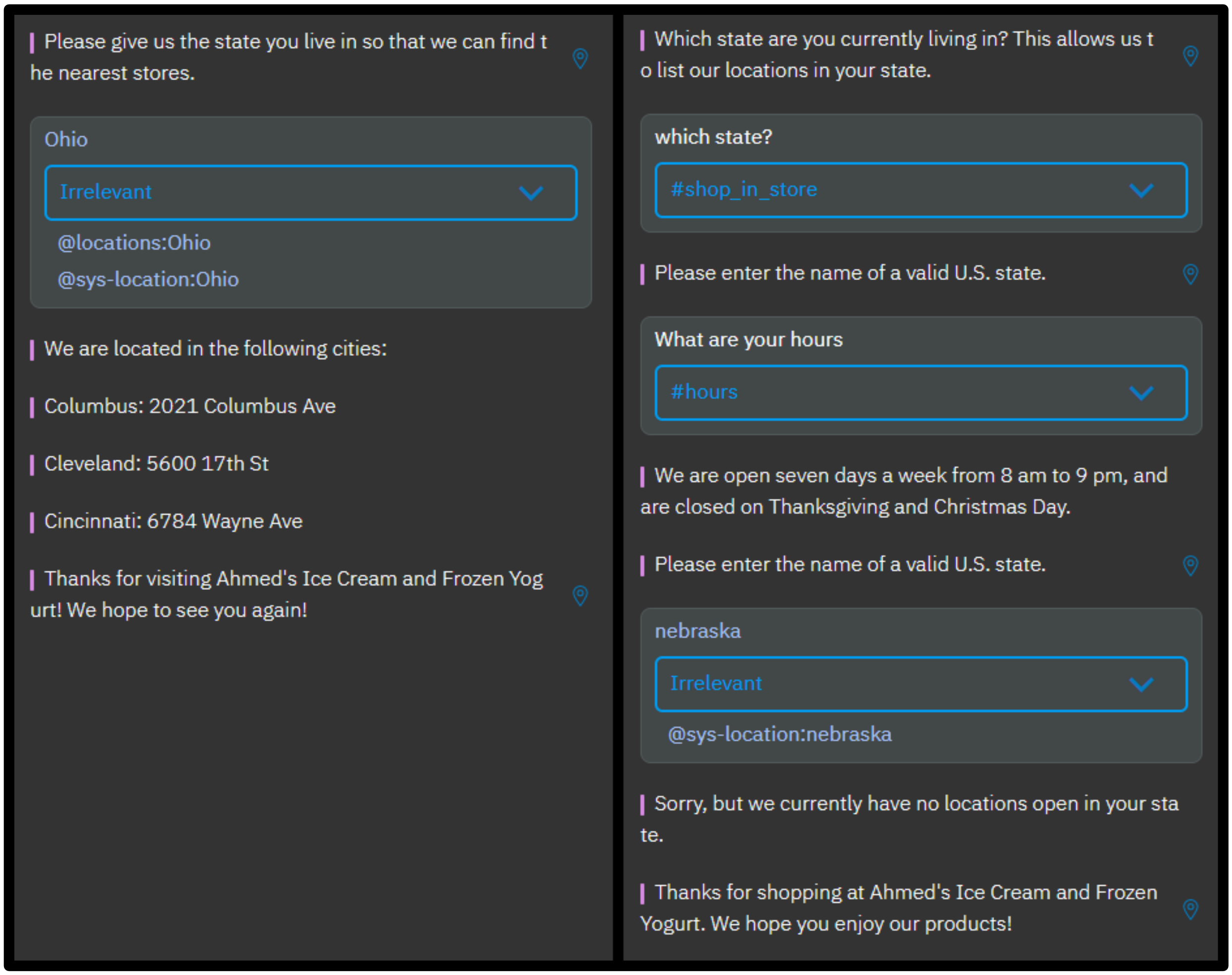

Figure 3. Handling of User Location Input and Demonstration of Possible Outputs.

Once the customer provides a valid location, the bot will use conditional responses to handle each possible input. There are 15 states (plus Washington, D.C.) in which the company has active stores. If one of these locations is recognized in the user input, the chatbot will show a list of cities in that state where we have a store, as well as the address for each store (see Figure 3). If the user had called the #goodbye intent, then no additional message will be displayed, and the dialog will jump to the goodbye node. And if the user had entered a valid location in which there are no open stores, then the bot will display a message telling them that there are no stores currently in their state (see Figure 3). In all cases, the dialog will jump to the goodbye node.

I will now look at the possible interactions if the user decides to shop online. Once the user calls the #shop_online intent, they will be taken to the "shop online" node where they are asked whether they want to buy ice cream or frozen yogurt. One of five random responses will be shown, each telling the user that frozen yogurt costs $0.25 more than ice cream and that they are only permitted to make one purchase at a time due to high customer demand. Ideally, I would like to allow customers to purchase multiple products in a single order. I found difficulties implementing this feature, but I would like to add it at some point during future analysis. At this point, the user can ask for either ice cream or frozen yogurt—bringing the dialog to the "buy ice cream" and "buy frozen yogurt" nodes respectively. Otherwise, the user can say "goodbye" to end the transaction and display the goodbye node.

The "buy ice cream" and "buy frozen yogurt" nodes function almost identically and contain similar child nodes, slots, and handlers. The only difference is that the price of frozen yogurt items are $0.25 more expensive than their ice cream counterparts. Once the user specifies the @food_type entity of their choice, they will be taken to a corresponding child node. Regardless of their choice, the dialog will involve two slots that check for the flavor and size of the order—storing each in the context variables $flavor and $size respectively. The user will be prompted for one or both of these entities, depending on the information already provided. If the user specified all information in the parent node (for instance, they asked for a "small chocolate frozen yogurt"), then the bot will recognize all three pieces of information and display the product's name and price (see Figure 4). Otherwise, the user will be prompted until all required information is given or until the user ends the dialog by saying goodbye.

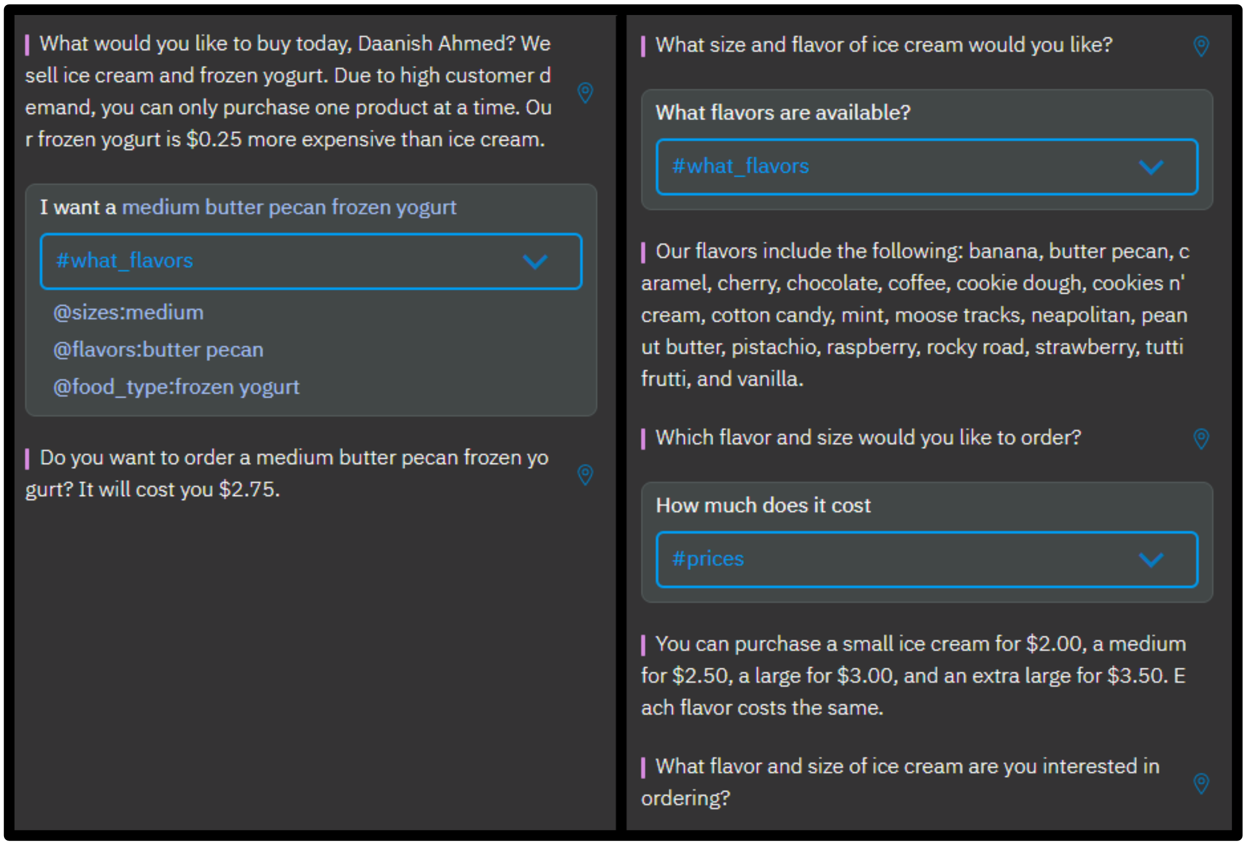

Figure 4. Handling of Online Customer Purchase Queries.

During this stage, the user can ask questions about the available flavors, sizes, and their corresponding prices. There are four handlers to deal with each possible user input. If the bot recognizes the #what_flavors intent, then it will show a list of all 19 available flavors (see Figure 4). If it recognizes the #food_size intent, then it will list the four available sizes and their prices. If the user calls the #prices intent, then the bot will explain the pricing of each product—namely that the cost increases based on the size of the item rather than the choice of flavor (see Figure 4). In all three cases, the user will still be prompted about the details of their order. But if the user calls the #goodbye intent, then the transaction is cancelled and the goodbye node is shown.

Once the bot gathers all required information, it will display the product name and price and it will ask the user to confirm their order. If the customer confirms the order, then the dialog moves to the "confirm" node. They will receive one of five random messages telling them that their order was confirmed as well as the price that they owe us (see Figure 5). Since this is a fictional company, I do not ask them for their credit card or any payment information. The dialog will then jump to the goodbye node, ending the conversation. However, if the user chooses not to confirm the order, then the dialog will move to the "decline" node. Here, they are told that their order was cancelled and the dialogue jumps back to the "shop online" node (see Figure 5). But in addition, the "decline" node will also set the $size, $price, and $flavor context variables to null—allowing users to start fresh with another order ("Watson Conversation," 2018). The system will not remember their previous order after it is cancelled, and thus the user can buy a different product and will not be charged with the previous order's price.

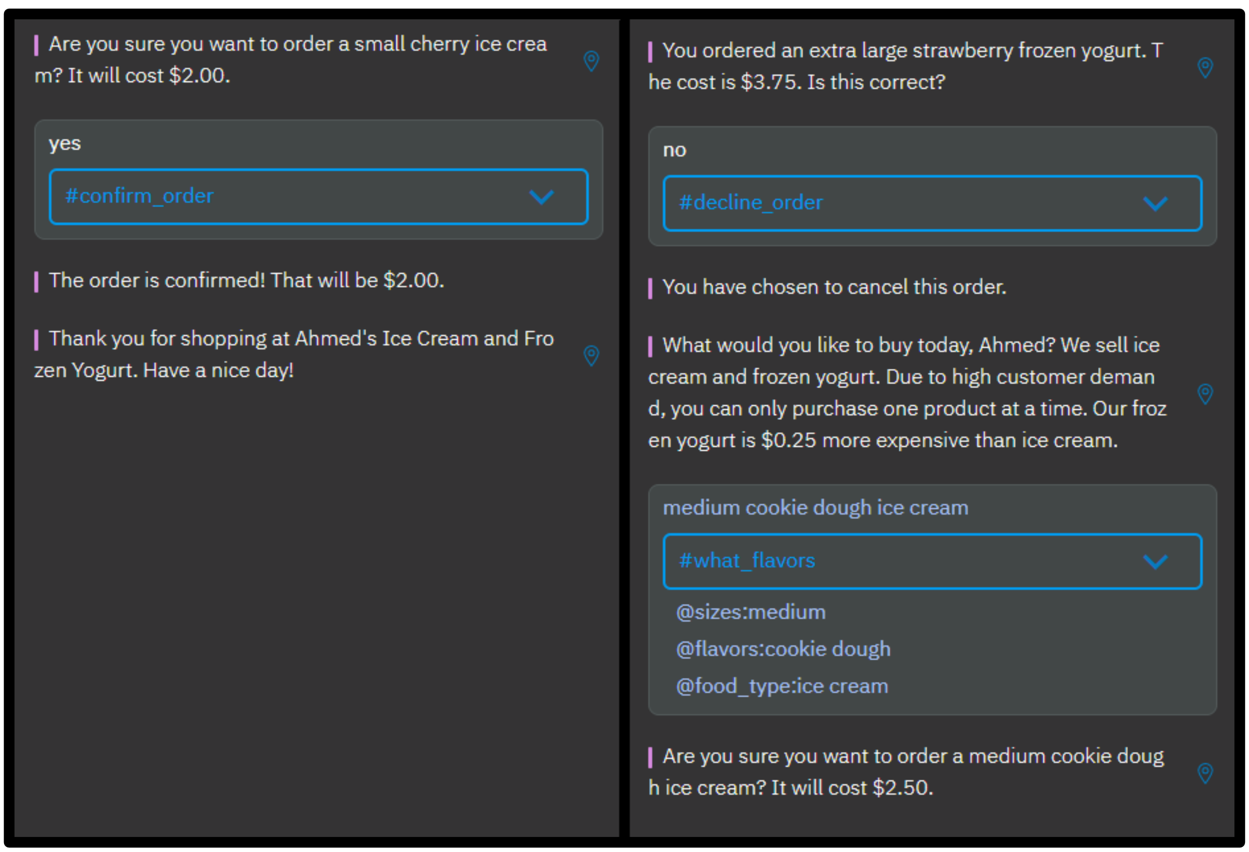

Figure 5. Handling of Online Order Confirmation Dialog.

As I mentioned throughout this section, there were several methods that I used to handle unexpected input. I used slots that require users to enter certain values, and I implemented handlers that allow them to ask questions about products or store hours. Furthermore, there were several actions that I performed to train the chatbot. For each intent, I included as many user examples as possible to allow for multiple variations of the same query. I tested the bot using the "try it out" panel—if a user's response does not provide the expected answer, then I would change the intent to match the expected intent. For example, if a customer types "when can I visit," then I will set the intent to #hours so that the bot will describe the store's schedule (see Figure 6). Likewise, for each entity value I included as many synonyms as possible to account for a user's language variation. I also enabled fuzzy matching for each entity to allow the bot to recognize misspelled words. These actions allow the bot to process user input more effectively.

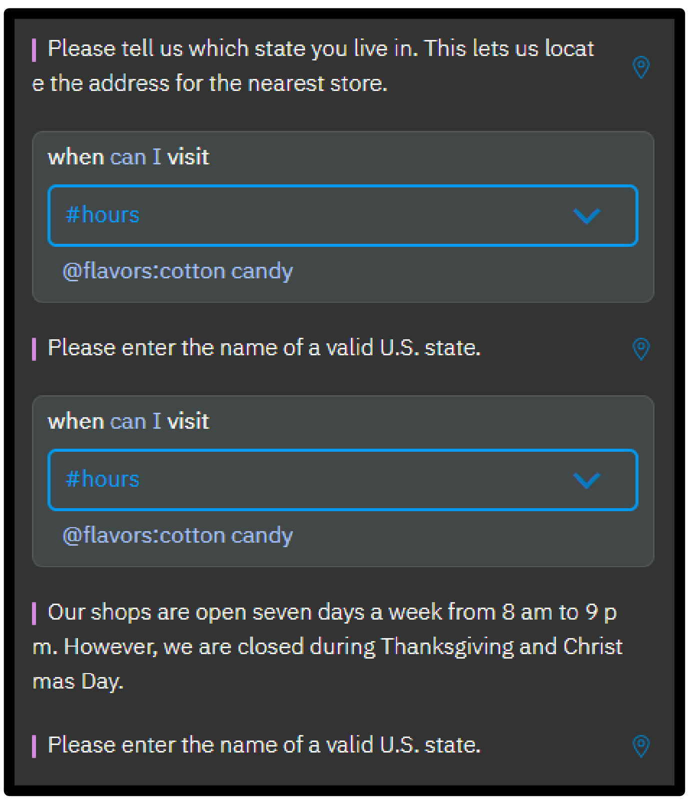

Figure 6. Example of Training the Chatbot by Changing the User Response's Intent.

Analysis and Enhancement

In this section of the paper, I will analyze the abilities of my chatbot and look for ways to improve it during a future analysis. Overall, this bot was designed to offer a more personalized experience to each customer—as if they were interacting with another human. Although the bot is not perfect, there were several measures I used to simulate human-like behavior. For instance, I used context variables to store information such as the user's name and the details of their order (such as size and flavor). Identifying context is one of the greatest challenges of Natural Language Processing, and it is one of the fields in which Watson has made great progress. Storing the customer's name and calling them by their name makes each conversation friendlier and more human-like. Additionally, remembering the context of a customer's order prevents the bot from asking a question after already obtaining the answer. Traditional computing systems often collect data only when prompted, but a human can remember information provided at any point of the conversation. This also allows the bot to collect multiple pieces of information at once, instead of prompting the user for every piece of input one at a time.

Another method used to enhance the bot's performance is to provide random variations of each possible response. Instead of stating the same sentence over and over, the bot will randomly select a variation of that response within the dialog node. This helps to make interactions appear less robotic. Additionally, the bot can answer unexpected questions regarding the prices, store hours, and available products. It does this by parsing the question and extracting major features such as the intent name, which enables it to provide a relevant response (High, 2012). This ability gives the chatbot an advantage over traditional computing systems, which usually are only able to handle certain responses at a designated time. Furthermore, it fulfills one of the goals of Natural Language Understanding (NLU), which is to answer specific questions about the text's contents

(Liu, n.d.). Finally, the repeated training provided to the chatbot helps it to deliver a more natural level of interaction. The bot learns from its experience interacting with humans, which is one of the key components of cognitive computing systems (Gortcheva, 2018). The bot is not an example of artificial intelligence, since it still requires human interference to learn. However, the training process allows it to respond more effectively to specific user requests and handle variations in word usage or sentence structure.

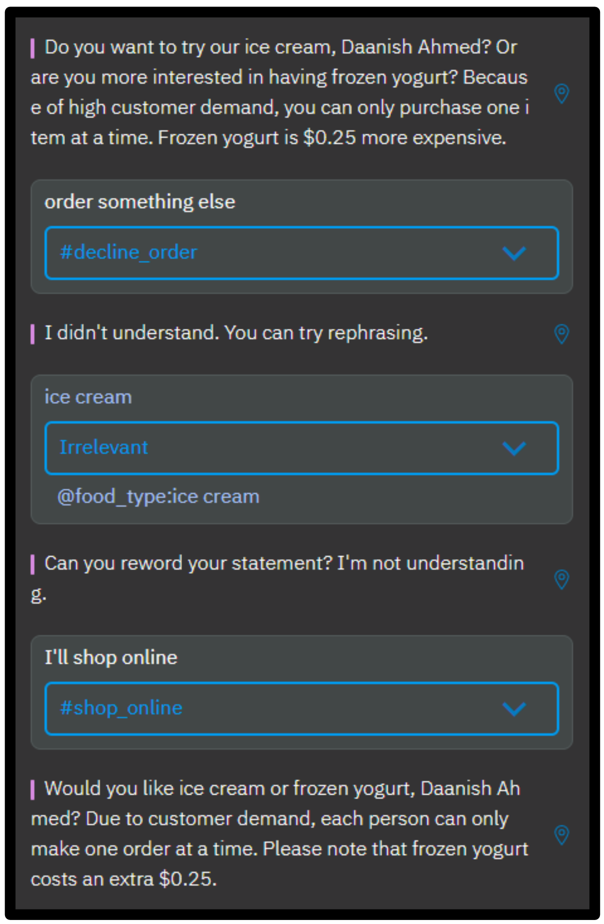

Figure 7. Issues with the "Anything Else" Node Within Child Nodes.

But during the chatbot implementation, I noticed several problems that were difficult to resolve. One of these issues involved the "anything else" node, which shows a message to the user whenever the bot does not understand the input. When a user enters an unrecognized query within certain child nodes, the dialog will exit the child node and go to the “anything else” node at the base of the dialog tree. This means that conversations within the child node will be cancelled, and the user will have to work their way back up the dialog tree to get back to that node. For example, if the user chooses to shop online and asks for anything other than ice cream or frozen yogurt, the bot will not understand. However, the user cannot immediately ask for ice cream or frozen yogurt again, since the bot is no longer in the “shop online” node. Therefore, the user will have to call the #shop_online intent again to answer the initial question (see Figure 7). I resolved this issue for the order confirmation nodes by confirming the order if the user says “yes,” and cancelling it if they say anything else. A similar approach can also be applied to the “ice cream” or “frozen yogurt” issue, but it may not be the most effective solution. Slots can also be used to solve this issue, but that will make it harder to implement the “goodbye” feature.

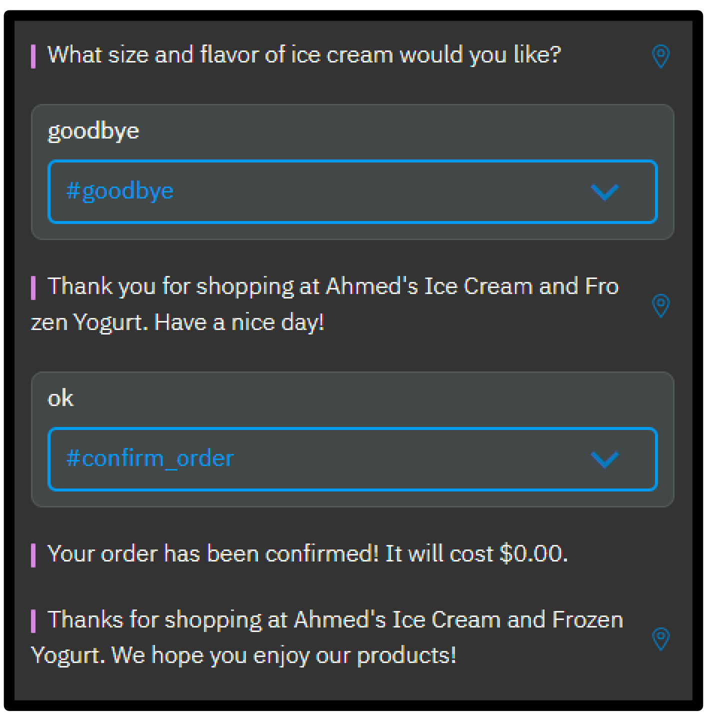

Figure 8. Early Testing Issue with the #Goodbye Intent Within Child Nodes.

With regards to the #goodbye intent, I also found challenges incorporating this feature into certain child nodes. This issue was most prevalent within nodes that use multiple slots—namely "buy ice cream" and "buy frozen yogurt." I implemented the "goodbye" feature as a handler that skips to the response if the #goodbye intent is found. It does not use the "skip current slot" option because the nodes contain more than one slot. The problem with my initial approach was that these two nodes contain their own child nodes and they would not immediately jump to the main "goodbye" node. Although the bot would display a goodbye message, the dialog stays within the child node—meaning that the user can continue the dialog and confirm their order even if they never made an order (see Figure 8). This may not seem like a major issue, especially since the user was always able to continue the dialog after visiting the real "goodbye" node. However, it is better to prevent customers from making unintentional purchases, even if the cost is $0.00.

To solve this issue, I set the "buy ice cream" and "buy frozen yogurt" nodes to skip user input after obtaining the information required by the slots. Next, I created additional child nodes that would jump to the main "goodbye" node if the #goodbye intent was used at certain points in the dialog. After this node, I would have a node that prompts users to confirm their order. This node will always show up unless the user called the #goodbye intent earlier. From this node, there will be three child nodes: one that confirms the order, one that declines the order, and one that jumps to the "goodbye" node. These revisions successfully allowed the conversation to move to the main "goodbye" node regardless of when the #goodbye intent is used.

To make my bot even more effective, I wish to integrate several Watson services in addition to Watson Conversation. For instance, I would incorporate Watson Text to Speech and Speech to Text. These services would allow the user and chatbot to communicate both verbally and through text. Customers can communicate their orders through speech, and the bot will understand and process their orders. Furthermore, the bot will be able to speak as well, instead of only displaying text to the user. This will ideally make the customer experience more immersive and give users more communication options. However, the current state of Text to Speech produces very robotic voices, which makes it clear to customers that they are still interacting with a computer program. Also, Speech to Text is likely to make mistakes when handling unusual terms and when listening to customers with uncommon accents or speech patterns. I believe that as Watson's technology improves, these services will become valuable tools to implement with my chatbot.

Additionally, I would like to use Watson Tone Analyzer to enhance my chatbot's ability to deal with customers. This service analyzes a document to determine the possible emotions expressed in the text. It assigns a percentage score to measure each emotion detected in the entire text and across individual sentences. It can be extremely useful for evaluating customer satisfaction with our services, and it can be used to improve the quality of those services as well. For instance, if a customer is found to be unhappy with their experience, then the bot can find a way to appeal to that customer—such as offering refunds or discounts if applicable. This service will require further development of my chatbot so that it can handle more advanced transactions. Tone Analyzer is also likely to have issues in its current state, since it can potentially miss or overestimate certain emotions. However, this feature may prove to be the next step towards developing a human-like chatbot.

Once the bot has been developed sufficiently, the next step is to deploy it using a messaging platform. One platform I would consider is Slack, which is commonly used for communication at work. The reason I choose Slack is because it is easy to use for customer support purposes, and it provides extensive support for many types of bots (Brisson, 2016). One of my goals is to improve my current bot's customer support abilities, such as evaluating customers' experiences and appealing to their needs when necessary. Thus, Slack may serve as a good choice for deployment. But although many companies have incorporated bots using Slack, it is still primarily a workplace communications platform and is not always ideal for customer-based interactions (Brisson, 2016). One alternative platform is Skype, which has expanded beyond its video conversation format to include text and image communications. Skype has also implemented bot support (Brisson, 2016), which makes it a viable choice for deployment. However, its focus is still primarily on video communications, and thus I am more likely to choose Slack as my platform.

Conclusion

In this analysis, I used Watson's cognitive computing technology to create a chatbot that interacts with customers at my fictional ice cream parlor. The bot was designed to communicate with customers in a more realistic and human-like manner. This was achieved by using multiple techniques, such as storing useful information in context variables, providing some answers to specific user questions, and displaying multiple variations of every response. The bot was created to assist users with two tasks: ordering products online and finding addresses for nearby stores. Users who choose the latter task will be asked to provide the state that they live in. Depending on their location, they will either be shown the list of stores and their addresses, or they will be told that no stores currently exist in their state. If the customer wants to shop online, they will be allowed to make a single order of either ice cream or frozen yogurt. After specifying the size and flavor, they will be asked for confirmation—for which they can either confirm the order or cancel it and choose a different product. Finally, users can end the dialog at any point by calling the #goodbye intent. Overall, the bot produced a certain degree of human-like behavior by interacting with users in a flexible manner and offering a wide variety of relevant responses.

However, there were several challenges that I experienced during the implementation of my chatbot. One of these challenges was an issue with name recognition using the @sys-person entity—in which the bot would fail to recognize certain names if they are written in lower case or are used in a sentence. I tried to resolve this issue by using an additional context variable, but I was unable to fix it. Another challenge involved navigating the dialog tree when the "anything else" node is visited in a child node. This can cancel the conversation within that child node, forcing the user to navigate their way back up the dialog tree to get to their original position. I solved this issue for some of the affected nodes, such as the confirm and decline order nodes. This was done by confirming the order if the user says "yes" and declining it if they say anything else. However, this problem still needs to be resolved for the "shop online" node. The third challenge involved using the #goodbye intent within certain child nodes and ensuring that these cases link back to the main "goodbye" node. I solved this issue by reorganizing the structure of some of the child nodes and creating nodes that jump directly to the "goodbye" node. For the issues that currently remain unresolved, I intend to study them further in a future analysis.

One recommendation I have for improving my chatbot is to allow the bot to collect multiple orders rather than only allowing a single order at a time. In my experiment, I only allowed users to order one item at a time—telling them that this was due to high customer demand. However, the real reason is that the chatbot was not configured to collect multiple orders and calculate the price of the combined order. On real company websites, customers will almost never be limited to making a single order when purchasing products online. Therefore, this feature is a major consideration for future development of the chatbot. Another recommendation for making a smart chatbot is to incorporate additional Watson services such as Speech to Text, Text to Speech, and Tone Analyzer. As mentioned earlier, these features will help the bot communicate more effectively with customers using speech as well as text, and they will allow the bot to appeal to certain customer emotions. However, these technologies are still developing and may need further improvement before they can display true human-like behavior. One final recommendation is to incorporate date and time-of-day information using the @sys-currency and @sys-time entities. One of my goals is to allow my chatbot to greet users appropriately depending on the time of day (morning, afternoon, and evening). Furthermore, I would also like to incorporate discounts on specific flavors during certain days of the week, as well as having major discounts during holidays. If the bot can recognize these entities and provide appropriate responses, then it will make the chatbot service much more appealing to our customers.

There are several additional take-aways from this experiment. Firstly, the approach used in this analysis offers a friendlier and more welcoming environment for users. The bot can remember context such as a user's name and the details of their order, and it can answer numerous questions and offer varied responses. These features allow it to provide relevant information to customers while treating them with respect and providing quality service. As such, I expect that users will enjoy interacting with my chatbot and are more likely to become frequent customers. Secondly, Watson's usage of cognitive computing technology allows for more flexible interactions between users and the chatbot. The bot can collect multiple pieces of information at once, even when it is not prompting for that information. This allows it to avoid the rigid, sequential approach of traditional computing systems. Lastly, the technology—while impressive—is still a long way off from portraying true human-like behavior. As mentioned, there were numerous challenges that the bot had difficulty overcoming. They are not only prone to errors, but they are also unable to learn themselves and require constant training by humans. However, their capabilities will certainly improve as cognitive computing technology matures.

References

Brisson, K. (2016, August 15). 11 best messaging platforms compared. Retrieved March 2, 2018, from https://venturebeat.com/2016/08/13/11-best-messaging-platforms-compared/

Gortcheva, E. (2018, February 13). *DATA 650 NLP & Cognitive*. Retrieved February 14, 2018, from https://www.youtube.com/watch?time_continue=474&v=xvBg4ZCJIS8

High, R. (2012, December 12). The Era of Cognitive Systems: An Inside Look at IBM Watson and How it Works. Retrieved February 19, 2018, from http://www.redbooks.ibm.com/redpapers/pdfs/redp4955.pdf

Liu, X. (n.d.). Natural Language Processing. Retrieved February 14, 2018, from https://datajobs.com/data-science-repo/NLP-Background-%5bSU%5d.pdf

Watson Conversation - Advanced dialog tutorial. (2018, January 22). Retrieved February 25, 2018, from https://www.youtube.com/watch?v=Z_vmzC0tu60&feature=youtu.be