

Dialog State System for Restaurants recommendation

Developing a machine learning-based state transitioning dialog learning

P. Bakker

Master AI, Utrecht University, p.bakker4@students.uu.nl

J.T HOITING

Master AI, Utrecht University, j.t.hoiting@students.uu.nl

A. LEENDERTSE

Master AI, Utrecht University, a.leendertse@students.uu.nl

Abstract

This report outlines the development of a machine learning-based dialog system designed to recommend restaurants based on user preferences. The system interacts with the user by asking for preferences regarding food type, location, and price range, and offers restaurant recommendations accordingly. The dialog flow is controlled by a state-transition mechanism, which classifies user inputs using K-nearest neighbors and decision tree algorithms trained on dialog act data. The system also incorporates a reasoning component, allowing users to specify additional requirements, such as whether the restaurant should be "romantic" or "child-friendly." A rule-based reasoning engine evaluates these preferences and filters restaurant options accordingly. Additionally, the system is configurable, allowing for variations in response behavior and includes the use of Levenshtein distance for handling user misspellings.

Additional Keywords and Phrases: Dialog system, dialog state, machine learning, restaurants

ACM Reference Format:

First Author's Name, Initials, and Last Name, Second Author's Name, Initials, and Last Name, and Third Author's Name, Initials, and Last Name. 2018. The Title of the Paper: ACM Conference Proceedings Manuscript Submission Template: This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word. In Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages. NOTE: This block will be automatically generated when manuscripts are processed after acceptance.

1. INTRODUCTION

In the year 1996, Dillon and Watson [1] highlighted the importance of Human-Computer Interaction (HCI). Computers should be designed in a way that improves this interaction, and therefore research in mainstream psychology should be applied to the field of HCI. They [1] stated that analyzing systems to improve the HCI could be beneficial for the future.

Years later, in 2011, Microsoft [2] launched an edition of their on-going of research community challenge tasks called the “Dialog State Tracking Challenge” (DSTC). The challenge consists of multiple different tasks, each provided with dialog data that can be utilized to train a system to predict new dialogs. One of the tasks of the DSTC in 2011 used human-computer dialogs in the restaurant information domain and this forms the basis for this report. The aim of this project, therefore, is to implement a dialog system in the restaurant recommendation domain.

The system that is developed during this project is programmed in Python 3. It operates through a text-based interface and allows a user to interact with the program through the terminal. During this human-computer interaction, it classifies the dialog act of a user utterance and extracts their restaurant preferences. The classification of utterances is performed with the help of a machine learning method that is trained on the data generated by the DSTC mentioned above. Utilizing these extracted preferences, it identifies restaurants that match the user’s desires from a database, resulting in a fitting restaurant recommendation. More detailed information of the data is provided in section 2 of this report, and section 3 describes the applied machine learning technique(s).

In order to converse with a user, the restaurant recommendation system employs a dialog management system. It manages to shift through a conversation while extracting preferences by transitioning through different dialog states. The implementation of this dialog management system can be found in section 4.

Finally, both a reasoning component was incorporated in the system, as well as multiple different extra configurability features. Because of the inclusion of the reasoning component, as explained in section 5, the system is able to use simple inference rules to integrate additional user preferences when recommending a restaurant. In section 6, the configurability features are discussed.

2. DATA

The next section describes the data that is employed for this project.

2.1. Dataset description

As is mentioned in section 1, the data that is utilized for this project was obtained by the means of the Microsoft DSTC and is therefore textual data. It consists of a total of 25.501 utterances, labeled as one of fifteen dialog act classes. The individual classes and a description of their roles can be observed in Table 1. The third column of the table, containing example sentences, shows the domain of the data; they are all dialogs in the restaurant domain.

The datafile is formatted as follows: *dialog act [space] utterance content*. After analysis, it becomes clear that the dataset is quite heavily skewed towards some of the dialog acts, especially INFORM and REQUEST, taking up around 40% and 25% of the dataset respectively. This can be seen in the bar graph in Figure 1. This is important, because a skewed dataset can potentially result in overfitting on those classes.

Out of the 25.501 utterances in the dataset, a lot of duplicates are present. After discarding the duplicate instances, 5.359 unique utterances are left. In the utterances, no capital letters or punctuation is present. In addition, the data is generated through automatic speech recognition. This means the data is not always perfect English and grammatically correct. However, because grammar might not be perfect when humans interact with the chatbot that is created during this project, this might not be a problem.

Table 1: 15 Different dialog acts present in dataset

| DIALOG ACT | DESCRIPTION | EXAMPLE SENTENCE |
|------------|--|--|
| ACK | acknowledgment | <i>okay um</i> |
| AFFIRM | positive confirmation | <i>yes right</i> |
| BYE | greeting at the end of the dialog | <i>see you good bye</i> |
| CONFIRM | check if given information confirms to query | <i>is it in the center of town</i> |
| DENY | reject system suggestion | <i>i dont want vietnamese food</i> |
| HELLO | greeting at the start of the dialog | <i>hi i want a restaurant</i> |
| INFORM | state a preference or other information | <i>im looking for a restaurant that serves seafood</i> |
| NEGATE | negation | <i>no in any area</i> |
| NULL | noise or utterance without content | <i>cough</i> |
| REPEAT | ask for repetition | <i>can you repeat that</i> |
| REQALTS | request alternative suggestions | <i>how about korean food</i> |
| REQMORE | request more suggestions | <i>more</i> |
| REQUEST | ask for information | <i>what is the post code</i> |
| RESTART | attempt to restart the dialog | <i>okay start over</i> |
| THANKYOU | express thanks | <i>thank you good bye</i> |

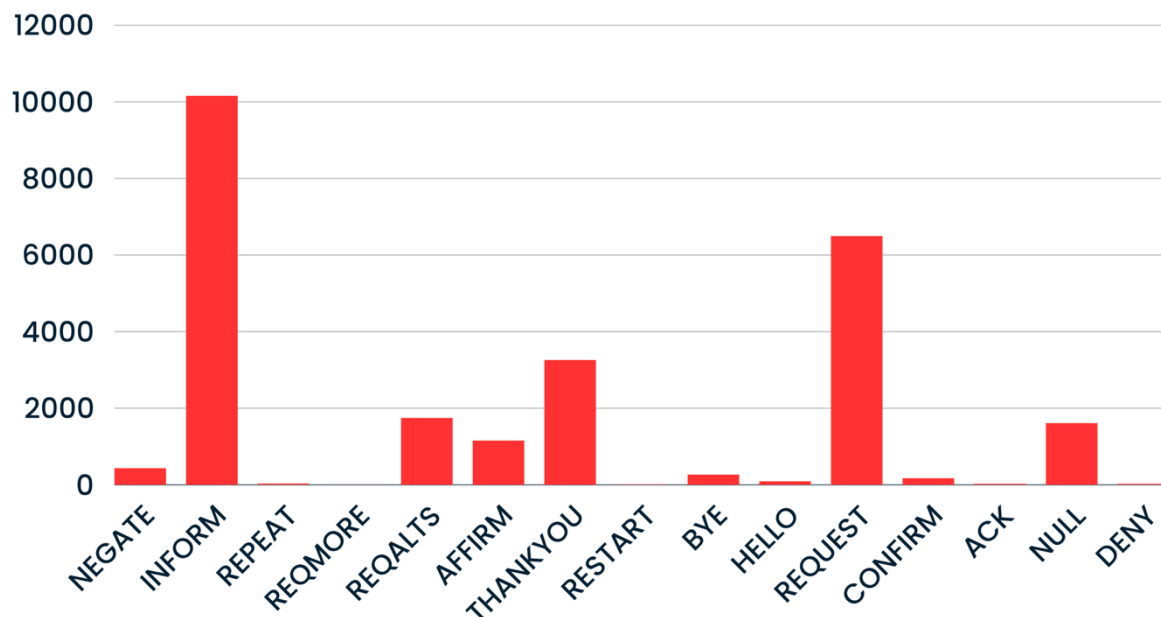
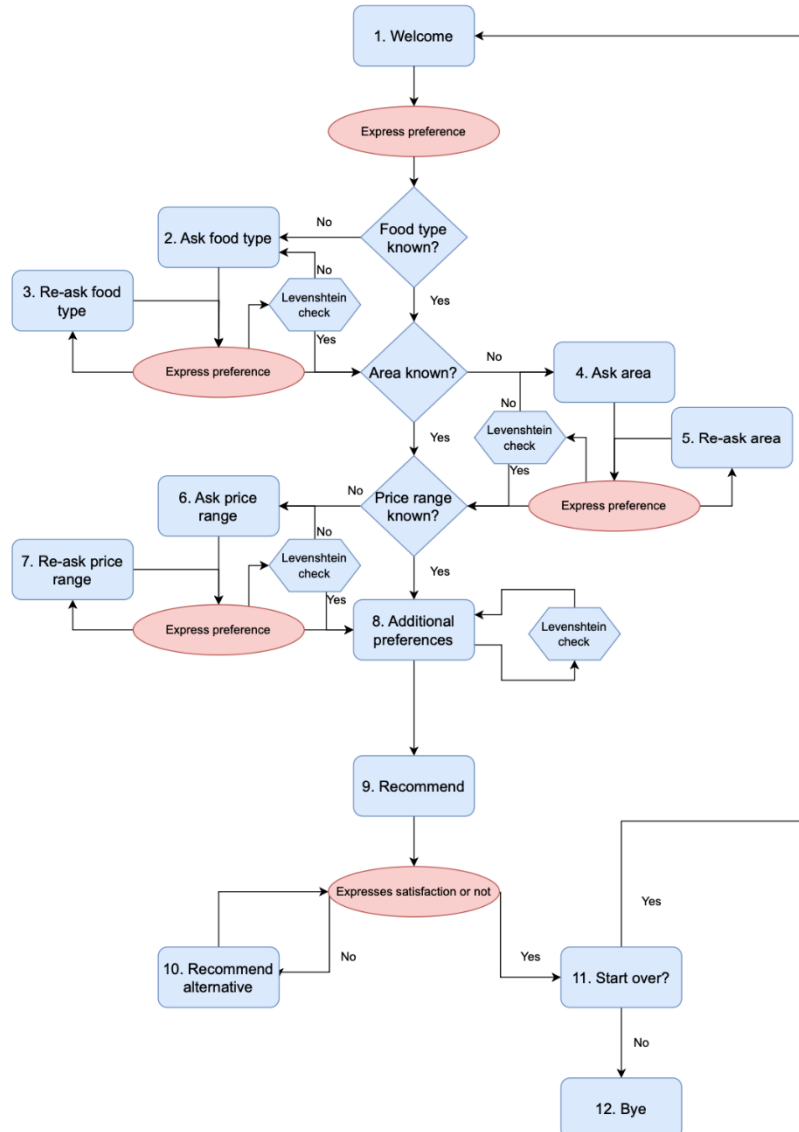


Figure 1: Bar graph of class balance in dataset

2.2. Dialog state transition diagram

The dialog state transition diagram for this project can be observed in Figure 2 of the report. The model consists of 12 different dialog states. The main goal of the model is to extract the three important user preferences, which are the type of food they would like, the area of the city (north, east, south, west and center), and the price range they have in mind (cheap, moderate expensive). It is possible that the user has no preference for one or more of these, in that case, anything in that category is accepted. The model transitions through these preferences in order of importance. It is decided that food type is most important, area thereafter, and lastly the price range.

After these three components are extracted, the system asks for additional preferences ('child friendly', 'touristic', 'assigned seats', 'romantic'). Based on these preferences, simple reasoning is applied to recommend the best fitting restaurant. More on this is discussed in section 5 of this report.



When all preference components are successfully extracted, the system recommends a restaurant that fits the preferences, whereafter its able to ask for an alternative recommendation. Then, when the user is satisfied with the recommendation, the option is given whether to start over or terminate the program.

Figure 2: Dialog state transition diagram

3. MACHINE LEARNING

The next section describes the techniques used to classify new instances based on the dataset

3.1. Data preprocessing

To process the data, the datafile is stored in a Pandas DataFrame [3], a 2-dimensional data structure, resembling a table with rows and columns. Utilizing this data structure, the data can easily be accessed, transformed or adjusted, with the goal of for example dataset analysis. The balance of the label count for example, is shown earlier in this report in Figure 1. Thereafter, it has been determined that the utterances are not long, with an average of 19.3 characters per utterance. Lastly, it is discovered that many duplicates are present in the dataset and discarding those brings the size of the dataset from 25.501 to 5.359 utterances. However, this means that there are almost no instances left for the classes with the least number of instances in the dataset. The label REQMORE for example, has 5 instances in the non-deduplicated dataset and only 1 instance in the deduplicated dataset. Therefore, the choice is made to deduplicate for every label except REQMORE, RESTART, DENY and REPEAT. This leaves 5.390 utterances.

3.2. Baseline systems

Two different baseline systems are employed in this project: a majority baseline system as well as a keyword baseline system

3.2.1. Majority baseline

The majority baseline system works in a simple way. It counts for each label in the dataset the amount instances, thus discovering the label that has the most instances. The baseline system then always classifies a new utterance as the “majority label”. In case of this dataset, the majority label is INFORM, accounting for 40% of all utterances in the non-deduplicated dataset.

3.2.2. Keyword baseline

The keyword baseline system classifies utterances based on a keyword lookup. If a keyword is detected in a utterance that is linked to one of the classes, the utterance is classified accordingly. If no keyword is detected, the majority label INFORM is used to classify the utterance. This keyword baseline system achieves a total accuracy of 72.75%. The keywords for every label can be found below.

```

label_keywords = {
    'inform' : ['any', 'looking'],
    'request' : ['what', 'whats'],
    'thankyou' : ['thank', 'thanks'],
    'reqalts' : ['else', 'about'],
    'null' : ['noise', 'cough'],
    'affirm' : ['yes'],
    'negate' : ['no'],
    'bye' : ['bye', 'goodbye', 'adieu'],
    'confirm' : ['does', 'serve'],
    'repeat' : ['repeat', 'again'],
    'ack' : ['okay', 'um'],
    'hello' : ['hello', 'hi'],
    'deny' : ['wrong'],
    'restart' : ['start', 'again'],
    'reqmore' : ['more']
}

```

3.3. K-nearest neighbors

The K-nearest neighbor approach [4] involves identifying a specified number of training samples that are closest to a new data point based on distance, and using them to predict the label of the new point. When a new utterance needs to be classified, the K-nearest neighbors algorithm searches through the training set and identifies the nearest neighbors of the new utterance and classifies it accordingly. This means that the model requires some computing power for each new classification, as it must search through the entire training set every time it classifies an input. For this project, the scikit-learn library for Python is utilized. The K-nearest neighbors model was tested on both the non-deduplicated dataset as well as the deduplicated dataset. These are the results of both:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| ack | 1.00 | 0.25 | 0.40 | 4 |
| affirm | 1.00 | 0.92 | 0.96 | 156 |
| bye | 0.95 | 0.95 | 0.95 | 42 |
| confirm | 0.94 | 0.70 | 0.80 | 23 |
| deny | 0.50 | 0.25 | 0.33 | 4 |
| hello | 0.92 | 0.73 | 0.81 | 15 |
| inform | 0.97 | 0.98 | 0.98 | 1530 |
| negate | 1.00 | 0.86 | 0.93 | 58 |
| null | 0.89 | 0.95 | 0.92 | 244 |
| repeat | 1.00 | 0.83 | 0.91 | 6 |
| reqalts | 0.95 | 0.92 | 0.93 | 261 |
| reqmore | 1.00 | 1.00 | 1.00 | 2 |
| request | 0.98 | 0.99 | 0.99 | 968 |
| restart | 1.00 | 0.50 | 0.67 | 4 |
| thankyou | 1.00 | 1.00 | 1.00 | 508 |
| accuracy | | | 0.97 | 3825 |
| macro avg | 0.94 | 0.79 | 0.84 | 3825 |
| weighted avg | 0.97 | 0.97 | 0.97 | 3825 |

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| ack | 1.00 | 0.17 | 0.29 | 6 |
| affirm | 1.00 | 0.24 | 0.38 | 21 |
| bye | 0.25 | 0.50 | 0.33 | 4 |
| confirm | 0.93 | 0.78 | 0.85 | 18 |
| deny | 0.00 | 0.00 | 0.00 | 1 |
| hello | 0.00 | 0.00 | 0.00 | 2 |
| inform | 0.91 | 0.93 | 0.92 | 475 |
| negate | 1.00 | 0.60 | 0.75 | 20 |
| null | 0.33 | 0.59 | 0.42 | 29 |
| repeat | 0.88 | 0.78 | 0.82 | 9 |
| reqalts | 0.78 | 0.77 | 0.78 | 75 |
| reqmore | 0.67 | 1.00 | 0.80 | 2 |
| request | 0.90 | 0.92 | 0.91 | 131 |
| restart | 1.00 | 1.00 | 1.00 | 4 |
| thankyou | 0.86 | 0.50 | 0.63 | 12 |
| accuracy | | | 0.85 | 809 |
| macro avg | 0.70 | 0.58 | 0.59 | 809 |
| weighted avg | 0.87 | 0.85 | 0.85 | 809 |

Figure 3: Knn performance on non-deduplicated dataset

Figure 4: Knn performance on deduplicated dataset

The results make it apparent that deleting the duplicates does not seem to have a great influence on the performance of K-nearest neighbors. The F1-score is not increased for any of the dialog act classes. The non-deduplicated dataset achieves a rather high F1-score for each of the labels, only ACK and DENY obtain a F1-score that is lower than 0.5.

3.4. Decision tree

A decision tree is a machine learning model that utilizes a structure that resembles a tree, to classify an input. The model splits the training data in branches and moves down these until it reaches a leaf node, which predicts a class. For this project, the decision tree from scikit-learn library [5] for Python is utilized once again. A key difference between the decision tree and a K-nearest neighbors model is the computing power needed for classification. Once a decision tree is trained, classification only requires traversing through the tree, in contrast to the K-nearest neighbors model. The performance of the decision tree on the non-deduplicated dataset and the deduplicated dataset can be observed in Figures 5 and 6. The performance of the model when trained on the non-deduplicated dataset seems to be better for a lot of classes. However, the model trained on the deduplicated dataset yields a better performance for ACK, DENY, REPEAT and RESTART. This shows especially for restart, hence an utterance that simply states “restart” does not get classified correctly.

Figure 5: Decision tree performance on non-deduplicated dataset

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| ack | 0.50 | 0.25 | 0.33 | 4 |
| affirm | 0.99 | 1.00 | 1.00 | 156 |
| bye | 0.98 | 0.95 | 0.96 | 42 |
| confirm | 0.76 | 0.83 | 0.79 | 23 |
| deny | 1.00 | 0.25 | 0.40 | 4 |
| hello | 1.00 | 1.00 | 1.00 | 15 |
| inform | 0.99 | 0.97 | 0.98 | 1530 |
| negate | 0.98 | 1.00 | 0.99 | 58 |
| null | 0.84 | 0.98 | 0.91 | 244 |
| repeat | 1.00 | 0.83 | 0.91 | 6 |
| reqalts | 0.96 | 0.94 | 0.95 | 261 |
| reqmore | 1.00 | 1.00 | 1.00 | 2 |
| request | 0.99 | 0.99 | 0.99 | 968 |
| restart | 1.00 | 0.50 | 0.67 | 4 |
| thankyou | 1.00 | 1.00 | 1.00 | 508 |
| accuracy | | | 0.98 | 3825 |
| macro avg | 0.93 | 0.83 | 0.86 | 3825 |
| weighted avg | 0.98 | 0.98 | 0.98 | 3825 |

Figure 6: Decision tree performance on deduplicated dataset

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| ack | 1.00 | 1.00 | 1.00 | 1 |
| affirm | 1.00 | 0.88 | 0.94 | 33 |
| bye | 0.86 | 0.75 | 0.80 | 8 |
| confirm | 0.59 | 0.71 | 0.65 | 14 |
| deny | 1.00 | 0.50 | 0.67 | 2 |
| hello | 1.00 | 0.86 | 0.92 | 7 |
| inform | 0.95 | 0.90 | 0.92 | 472 |
| negate | 0.94 | 1.00 | 0.97 | 15 |
| null | 0.38 | 0.79 | 0.51 | 34 |
| repeat | 1.00 | 1.00 | 1.00 | 4 |
| reqalts | 0.87 | 0.73 | 0.79 | 73 |
| reqmore | 0.50 | 1.00 | 0.67 | 1 |
| request | 0.92 | 0.93 | 0.93 | 131 |
| restart | 1.00 | 1.00 | 1.00 | 4 |
| thankyou | 0.79 | 0.92 | 0.85 | 12 |
| accuracy | | | 0.88 | 811 |
| macro avg | 0.85 | 0.86 | 0.84 | 811 |
| weighted avg | 0.91 | 0.88 | 0.89 | 811 |

4. DIALOG MANAGER

4.1. State transition function

In the function we named responder, we made state transitions possible along with the system responses after having extracted the user utterance (called the “extractor” function). The state transitions also come with a response, hence the name for the function. The dialog system operates using different states (i.e. welcome, askfood, askarea) as could be seen from the state transition diagram in figure 2. With the use of the perform classification function the response of the user is classified so the next dialog act can be determined. State 1. is the welcome state where the user is greeted and asked what preferences the user has. Either the user says hello whereafter the system responds with: “Hi, please respond with your preferences” or the state transitions to askfood.

This is state 2 and asks the user specifically what the preference for food is if that is not already known from the response to the welcome state. If the utterance from the user has to use levenshtein, thus is not sure what the user wants for food, the state transitions to state 3 to ask whether the user meant what is a close match (using levenshtein). The system responds with: “Did you mean {x food}”. If the system already gathered which food the user wants, the state immediately transitions to askarea, which is state 4 and asks the user for their preference for area.

State 4 goes to state 5 if levenshtein is used and transitions to ask price (state 6) if the system gathered which area the user wants. The same goes for price range and then the system eventually gets to additional preferences state. This state, state 8, asks if the user has any additional requirements: “Got it! you want {food} food in {area} area with in the {price} price range. Do you have any additional details? (type “any” if not).” The user can respond with an additional requirement or with any (which is how we determine there is no additional requirement) which gets the system to transition to state 9, the recommend state, which recommends a restaurant according to the preferences given by the user or mentions there are no restaurants that fit all requirements.

If the system can recommend a restaurant and there are more restaurants that fit the preferences, the system asks if the user wants an alternative, if so, the system recommends a different restaurant. If the system did not recommend a restaurant the system immediately skips over to the 11th state: start over. This state is also reached when the user expresses satisfaction over the recommended restaurant. This state asks whether the user wants to start over, if so, state 11 returns to the welcome state 1. If the user does not want to start over the system goes into the last state 12: bye, where the user is greeted: “ok, bye then.”

4.2. System utterances templates

Table 2 contains the templates for all sentences and for each template an example. Some sentences are static and do not differ from the template

Table 2. template and example sentences by the chatbot system

| Template | Example |
|---|---|
| "Hi, please respond with your preferences." | "Hi, please respond with your preferences." |

| | |
|--|---|
| "What type of food do you want?" | "What type of food do you want?" |
| "Got it! You want {food} food. In which area do you want to eat (reply with north/east/south)?" | "Got it! You want Italian food. In which area do you want to eat (reply with north/east/south)?" |
| "Got it! You want {food} food in {area} area. What price range do you want?" | "Got it! You want Indian food in north area. What price range do you want?" |
| "Got it! You want {food} food in {area} area within the {price} price range. Do you have any additional details? (type "any" if not)" | "Got it! You want Thai food in east area within the cheap price range. Do you have any additional details? (type "any" if not)" |
| "A restaurant that serves {food} food in {area} part of town and that has {price} price is "{restaurant_name}"." | "A restaurant that serves Indian food in east part of town and that has moderate price is "Rajmahal"." |
| "I'm very sorry, but there are no restaurants that fit your preferences, would you like to start over?" | "I'm very sorry, but there are no restaurants that fit your preferences, would you like to start over?" |
| "An alternative restaurant could be {restaurant_name}. If you would like another, type 'alternative'." | "An alternative restaurant could be "frankie and bennys". If you would like another, type 'alternative'." |
| "Alright we will start over, what are your new preferences?" | "Alright we will start over, what are your new preferences?" |
| "A restaurant that serves {food} food in {area} part of town, that has a {price} price and is {preference} is "{restaurant_name}". It is {preference} because it is not busy and allows for long stays. \n In case you want an alternative, type "alternative", otherwise type "restart" to start over." | "A restaurant that serves British food in centre part of town, that has a moderate price and is romantic is "cotto". It is romantic because it is not busy and allows for long stays. In case you want an alternative, type "alternative", otherwise type "restart" to start over." |

| | |
|--|---|
| A restaurant that serves {food} food in {area} part of town, that has a {price} price and is {preference} is "{restaurant_name}". It is {preference} because the average stay is not long. \n In case you want an alternative, type "alternative", otherwise type "restart" to start over. | A restaurant that serves Indian food in north part of town, that has a cheap price and is child-friendly is "royal spice". It is child-friendly because the average stay is not long. In case you want an alternative, type "alternative", otherwise type "restart" to start over. |
| "A restaurant that serves {food} food in {area} part of town, that has a {price} price and has {preference} is "{restaurant_name}". It has {preference} because it is busy. In case you want an alternative, type "alternative", otherwise type "restart" to start over." | A restaurant that serves African food in centre part of town, that has a expensive price and has assigned seats is "bedouin". It has assigned seats because it is busy. In case you want an alternative, type "alternative", otherwise type "\restart\" to start over. |
| 'A restaurant that serves {food} food in {area} part of town, that has a {price} price and is {preference} is "{restaurant_name}". It is {preference} because it is cheap, has good food and is not Romanian. \n In case you want an alternative, type "alternative", otherwise type "restart" to start over.' | 'A restaurant that serves traditional food in north part of town, that has a cheap price and is touristic is "da vinci pizzeria". It is touristic because it is cheap, has good food and is not Romanian. In case you want an alternative, type "alternative", otherwise type "restart" to start over.' |
| "ok, bye then." | "ok, bye then." |

4.3. Example dialog

This is an example dialog, going through all of the states:

A full example dialog must be provided (possibly in the appendix) that traverses the diagram from the start state to the end state

system: Hello, welcome to the greatest MAIR restaurant system! You can ask for restaurants by area, price range or food type. How may I help you?

system: Hello, welcome to the greatest MAIR restaurant system! You can ask for restaurants by area, price range or food type. How may I help you?

user: hi

system: Hi, please respond with your preferences.

user: I want bri'ish food

system: Did you mean british?

user: yes

system: Got it! You want british food. In which area do you want to eat (reply with north/east/south/west/centre)?

user: certre

system: Did you mean centre?

user: yes

system: Got it! you want british food in centre area. What price range do you want?

user: moderate

system: Did you mean moderate?

user: yes

system: Got it! you want british food in centre area with in the moderate price range. Do you have any additional details (type "any" if not)?

user: it has to be romantic

system: A restaurant that serves british food in centre part of town, that has a moderate price and is romantic is "cotto".

It is romantic because it is not busy and allows for long stays.

In case you want an alternative, type "alternative", otherwise type "restart" to start over.

user: alternative

system: An alternative restaurant could be the oak bistro. If you would you like an alternative, type "alternative"?

user: no

system: Ok, bye then

5. REASONING

5.1. The reasoning component

A description of the reasoning component, including example dialog snippets

In the chatbot, the user gets asked, after giving the preference for the food, area, and price range, whether there are any additional requirements a restaurant should fulfill. This can either be romantic, child-friendly, touristic, or assigned seats. These additional requirements are assigned to a restaurant through rules. Every restaurant in the .csv file had three additional attributes: food quality (good or ok), crowdedness (busy or idle), and length of stay (long or short). Along with those three attribute, the price range was also used to determine whether a restaurant fit a certain requirement

- Touristic: A restaurant was touristic when the price is cheap and the food is good, as tourists often seek affordable but high quality dining options. The restaurant loses the ability to be touristic if the food was Romanian, this is a very unknown cuisine and less likely something tourists are familiar with.
- Assigned seats: if the restaurant is busy, it is assumed that there is no choice in the seating, the waiter assigns your seat.
- Child-friendly: Restaurants with shorter lengths of stay are considered child-friendly, based on the assumption that shorter dining times are more suitable for families with children.
- Romantic: Romantic restaurants are defined by two attributes, they should not be crowded and should allow for a longer stay. A less busy atmosphere combined with longer time for dining creates a more intimate setting

After evaluating each of the remaining potential restaurants, the system builds a new refined list that meets the earlier mentioned criteria and the rule-based/additional criteria.

5.2. Handling contradictions

There are two contradictions that are handled within the code, namely, the romantic and the touristic requirement. For the romantic requirement both conditions: uncrowded and long stay must be met for a restaurant to be considered romantic. For touristic restaurants, cuisine type takes precedence over price and quality.

6. CONFIGURABILITY

For this project, we wanted the possibility to change certain global variables from outside of the dialog class so these could be set and saved throughout multiple conversations depending on the preferences of the end user. This flexibility allows for a more personalised and adaptable user experience.

The configuration options are as follows:

All Caps Output: The possibility to set all output from the responder to all caps. This means that the system prints everything in capital letters. By default, this option is turned off, but it can be enabled if the user prefers to have all responses in uppercase for better visibility or emphasis.

Levenshtein Cutoff Variability: We wanted to provide the user with some variability for the Levenshtein cutoff. The Levenshtein system uses this cutoff variable to determine which ratio to use for providing a suggestion when the user misspells a preference for food, area, price, or any additional preferences they might have. By default, this ratio is set to 0.8 for food type, 0.65 for area, and 0.75 for both price range and additional preferences. This allows the system to be more or less strict in its suggestions based on user needs.

Response Delay: In our system, a delay was built in to make interactions feel more realistic. We didn't want this value to be too large, so by default, it is set to half a second. This delay can be adjusted according to the end user's needs, providing a balance between responsiveness and realism.

Dialog Restart: Lastly, we have the possibility to restart the dialog after the recommendation state. This can be useful when no recommendations are found for a certain set of preferences, for example. By default, we allow the restart, but this can be set to False if needed, giving the user control over the flow of the conversation.

6.1. High-Level Description of How the Options Are Set by the User and Implemented in Code

As for how these aforementioned options can be set by the user, this is done within the config.ini file. When starting the Python script, a check is performed to see whether the config file exists. If it does not, the script will create a new one with the default values. The file is then read, and the corresponding values are passed to the dialog class as input to be stored.

When a user changes certain global variables in the config file, the Python script must be fully restarted for these changes to take effect. This ensures that all new settings are properly loaded and applied. If a conversation is restarted within the dialog manager, the values are set according to the values stored from the config.ini file, meaning it will not reread the file during the same session. This approach ensures consistency and stability in the user experience, while saving computation.

7. CONCLUSION

7.1. Discussion of the approach for the dialog system in general

7.1.1. General overview of the project

We implemented a text-based dialog system aimed at recommending restaurants based on user preferences, which are gathered through a series of questions in a structured conversation. The system makes use of a machine-learning-based classification technique to determine dialog acts and guide the flow of conversation through different dialog states. The primary task of the system is to gather preferences regarding food type, area, and price range, and to provide restaurant recommendations from a pre-existing dataset. It also incorporates a reasoning component, allowing for additional

preferences, such as "romantic" or "child-friendly," to refine recommendations. Furthermore, a configurability feature allows for customisation, such as setting response delay or allowing dialog restarts.

The dataset used for training is heavily skewed towards certain dialog acts, which may lead to overfitting and reduced performance for less frequent classes. Additionally, the system's reliance on text-based input limits its usability for users who prefer voice interactions. The reasoning component is rule-based, which may not capture the full complexity of user preferences and restaurant attributes.

Possible improvements could include incorporating a more balanced dataset or using data augmentation techniques to address class imbalance. Extending the system to support voice-based interactions would enhance accessibility and user experience. Moreover, enhancing the reasoning component with more sophisticated algorithms, such as machine learning-based recommendation systems, could better capture complex user preferences.

Alternative approaches we could take include using deep learning models or transformers for dialog act classification and state management. Implementing a hybrid approach that combines rule-based reasoning with machine learning could improve recommendation accuracy and flexibility.

7.2. Discussion of the approach for dialog act classification

7.2.1. General overview of the experimental methodology

The dialog act classification was carried out using two machine learning models: K-nearest neighbours (KNN) and decision trees. Both models were trained on a dataset consisting of 25,501 labeled user utterances. Data preprocessing involved deduplication of utterances and keyword-based methods as a baseline for performance comparison. The system attempted to classify user input into one of 15 predefined dialog acts to determine how to transition between different dialog states. The system extracted dialog acts such as "INFORM," "REQUEST," or "AFFIRM" and used them to drive the conversation.

The deduplication process reduced the dataset size significantly, which may affect the model's ability to generalise. Additionally, the keyword-based baseline system, while simple, may not capture the nuances of natural language, leading to misclassifications.

Possible improvements include using more advanced preprocessing techniques, such as data augmentation or synthetic data generation, to increase the dataset size and diversity. Implementing more sophisticated feature extraction methods, such as word embeddings, could also improve the model's understanding of natural language.

The choice of KNN and decision trees was based on their simplicity and interpretability. However, these models may not perform as well as more complex algorithms on large and diverse datasets. Alternative approaches could include using deep learning models, such as convolutional neural networks (CNNs) or transformers, which have shown superior performance in natural language processing tasks. Exploring ensemble methods, such as random forests or gradient boosting, could also improve classification accuracy by combining the strengths of multiple models.

By addressing these limitations and exploring alternative approaches, the dialog system can be further enhanced to provide more accurate and user-friendly restaurant recommendations.

REFERENCES

1. Dillon, Andrew, and Ralph Gabbard. "Hypermedia as an educational technology: A review of the quantitative research literature on learner comprehension, control, and style." *Review of educational research* 68.3 (1998): 322-349. <https://doi.org/10.1006/ijhc.1996.0071>
2. Microsoft. (n.d.). *2011 Dialog System Technology Challenge (DSTC)*. Retrieved September 14, 2024, from <https://www.microsoft.com/en-us/research/event/dialog-state-tracking-challenge/>
3. Pandas. (n.d.). *Pandas Dataframe*. Retrieved September 14, 2024, from <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>
4. scikit learn (n.d.). *Nearest neighbors*. scikit-learn.org. Retrieved September 14, 2024, from <https://scikit-learn.org/stable/modules/neighbors.html>
5. scikit-learn (n.d.). *Decision Trees*. (n.d.). scikit-learn.org. Retrieved September 14, 2024, from <https://scikit-learn.org/stable/modules/tree.html>
- 6.
- 7.
- 8.

| | Jesse | Peter | Anne |
|--|-------|-------|-------|
| Baseline systems | 4hr | 0hr | 2hr |
| Data preprocessing and analysis | 2hr | 2hr | 3hr |
| Responder method (state transition system) | 5hr | 3hr | 5hr |
| Report | 3hr | 7hr | 5hr |
| First reasoning function | 1hr | 3hr | 1.5hr |
| Extractor method | 3hr | 1hr | 1.5hr |
| Configurability | 2hr | 0.5hr | 1.5hr |