**Methods in AI Research – Group 36 Project Report**

Group 36 Project Report

Technical report with a description of the dialog system and results of machine learning experiments.

Amir M. M. Abadi

Utrecht University, [a.a.mehrimahiabadi@students.uu.nl](mailto:a.a.mehrimahiabadi@students.uu.nl), 6551548

Jennifer Batchelor

Utrecht University

Thijme de Bruijn

Utrecht University, [t.m.j.debruijn@students.uu.nl](mailto:t.m.j.debruijn@students.uu.nl), 4707656

Oliver Popović

Utrecht University, [o.popovic@students.uu.nl](mailto:o.popovic@students.uu.nl), 1777947

This project aimed to develop a functional restaurant recommendation system using machine learning principles. The system was implemented through a text-based dialog management interface, supported by an ML algorithm trained on past user interactions and basic restaurant data. Before implementation, various ML algorithms were tested, with the linear regression model being selected based on quantitative evaluation. After refinement, the system offers basic yet effective functionality for restaurant recommendations. Future improvements include expanding location precision, increasing the number of restaurant characteristics considered, and enhancing the conversational capabilities of the dialog manager. Alternative approaches, such as integrating NLP and a voice-based interface, offer potential for future enhancements.

1. Introduction

This document provides a comprehensive overview of the design and development process for a restaurant recommendation system, created as part of the Methods in AI Research course. The system  allows users to specify preferences for a restaurant, and it delivers recommendations using a dialog-based interface. Key components of the system include a machine learning model for labeling dialogs and generating appropriate recommendations and a dialog manager to facilitate interaction between the user and the system.

The system's architecture is composed of several key components, which will be detailed throughout this report. Section 2 discusses the structure of the dialog manager, the core interface through which the user interacts with the recommendation system. The machine learning approaches tested in the project are outlined in Section 3, with a focus on the chosen logistic regression model. This decision was made based on the performance results from experimentation with various approaches. The datasets informing the system, which include the dialog transcripts and restaurant listings, are discussed in Section 4, while Section 5 addresses the reasoning of the dialog manager. Lastly, Section 6 explores the configurability options added to enhance the system’s flexibility.

1. Machine learning

In this project, the classification of dialogue acts was approached through multiple methods. Initially, two baseline models were developed to establish a benchmark accuracy, which we aimed to improve using more advanced machine learning techniques.

* 1. Baseline models

For both baseline models, some preprocessing of the data was necessary. Initially, the dataset was split into two versions: one containing duplicate entries and one without duplicates. After this, we applied a train-test split, using 85% of the data for training and 15% for testing in both versions. This ensured that our baseline models had a consistent structure to evaluate performance. For the machine learning models, it was required to vectorize the data so that it could be appropriately used with machine learning algorithms.

The first baseline model simply classified every utterance as the most frequently occurring dialogue act in the dataset. This model, by always selecting the majority class, achieved a baseline accuracy of 40%, as 40% of the dataset consisted of utterances classified as "inform." Consequently, every utterance was labeled as "inform" by this model. On the cleaned dataset this accuracy rose to 55% and this could be attributed to the fact that more “inform” dialog acts remained after duplicate removal. A reason for this relative increase might be that “inform” dialog acts are more varied than, for example, “bye” dialog acts. Meaning there will be less duplicates in the former and more duplicates to remove in the latter.

Once the first baseline was set, we introduced a second baseline model based on keyword matching. This model demonstrated greater flexibility than the first, resulting in a higher accuracy score. In this model, a predefined set of keywords was associated with specific dialogue acts. For instance, the words "bye" and "thank you" were mapped to the dialogue acts "bye" and "thankyou," respectively. A total of nine rules were applied to match keywords in utterances to corresponding dialogue acts. The keywords used were {"thank", “thanks”, "bye", "hi", “hello”, “helo”, "what", “phone”, “address”, "yes", “right”, “yeah”, "no", “nope”, "looking", “area”, “food”, "else"}, with all other utterances being classified as "inform."

To determine which keywords would optimize the performance of our baseline model, we examined example dialogue data and iteratively refined the model by adding more keywords. This process involved adjusting the keyword set to improve accuracy. Some straightforward cases, such as "hello" or "thanks," were easy to map to specific dialogue acts. However, for more complex cases, we experimented with various words to determine what worked best.

Interestingly, some added rules caused a decline in model performance, as certain words could be used in multiple contexts with different dialogue acts. For example, the word "restaurant" was challenging to assign to a single dialogue act. In the sentence "I want a different restaurant," the appropriate dialogue act would be "reqalts," while in "Do you know the address of the restaurant?" the dialogue act should be "request."

This highlighted the importance of carefully considering the potential implications of adding specific keywords to the keyword-matching model, as their use could vary significantly based on the context.

* 1. Machine learning algorithms

For the machine learning algorithms, Linear Regression and Decision Tree Classifiers were used. Both algorithms required the same vectorized input to work. The design choices for these algorithms were straightforward and to the point. To keep it simple and to save time, no additional hyperparameter tuning was done for either algorithm. This time was better spent on working on other parts of the dialog system.

* 1. Quantitative evaluation

In order to judge the performance of these models an evaluation metric is necessary. For this, a summary was chosen that includes multiple metrics such as precision, recall, accuracy and f1 score.

Both models (Logistic Regression and Decision Tree Classifier) performed similarly in many instances, with each achieving an accuracy of around 87% (figure 1 and 2). However, there were some differences in the other metrics. Specifically, the Decision Tree Classifier demonstrated a higher F1 score. The Logistic Regression model struggled to classify instances of the "repeat" and "restart" dialog acts, whereas the Decision Tree model successfully classified these cases with an F1 score of 1.00. The reason why the Logistic Regression model struggles with these dialog acts is because of their low frequency in the training data. Not much can solve this other than having a better distribution of data. Since Decision Tree Classifiers are highly flexible, this did not seem to be a problem for this algorithm. One dialog act where both algorithms struggled to get consistently accurate classifications is the “null” dialog act. The reason why this happens becomes apparent when looking at the dialog\_acts.dat file. Since the dialog\_acts.dat file consists of many voice recorded utterances, a lot of errors in the speech-to-text functionality caused unusual utterances. Many of these utterances contain nearly fully formed sentences, but were cut off halfway. These uncompleted sentences were then manually labeled as “null”, but they have little in common with each other. For example the following sentence was classified as a “null” dialog act: “looking for a place that serves whole worl”. Sentences like this can be very disruptive for machine learning models, because it contains words like ‘looking’ and ‘place’ that usually lead to other dialog act classifications. So the algorithms will try to match them to those dialog acts where these specific words are often more likely to lead to a successful dialog act classification.

When comparing the models trained on datasets with and without duplicates, all metrics were significantly better for those trained on datasets containing duplicates (figure 3 and 4). This improvement is likely due to the model encountering values in the test set that it had already seen during training, resulting in a higher probability of correct predictions. However, this does not reflect a model that generalizes well. Consequently, we opted not to deploy these models in most of our testing, as their performance was inflated by the duplicate data.

1. Dialog manager

The run function in the dialog manager starts by printing an initial message from the system, which could be a greeting or a prompt for user input. The user responds, and their input is processed to extract information relevant to the trained model, such as restaurant preferences. The function then enters a loop where it continuously processes user input, recognizes the dialog act, and calls the state transition function based on the recognized dialog act and the current state of the system. The state transition function manages what the system should do next, like storing user preferences and determining what the system should say in response. This loop continues until the final state of the dialog is reached, at which point the conversation ends.

The state transition function is responsible for deciding how the system moves from one state to another based on user input and internal logic. It begins by identifying the user's dialog act and the current state of the system. Then it will use that information to see where the user is in the state diagram, and what actions he/she can take. The user always begins in the “welcome” state (state 1), and will continue on to the next state when he/she begins writing their preferences. When this happens, the state will go to the “ask\_preferences” “state”. This isn’t really a state, but rather a main junction point where the system will check 1. What preferences the user has stated, 2. Whether any restaurants exist with the current preferences, 3. If all preferences have been stated, 4. Whether the search area has been defined, 5. Whether the food type has been defined, and 6. Whether the price range has been defined. When the system decides which action to take, it will send a system message to the user, and it will set the next state of the state diagram. It will go back and forth between this “ask\_preferences” “state” and the actual states until all preferences have been correctly defined. When all preferences have been correctly defined, it will send them to the “additional\_requirements” state (state 6). It will then ask the user to choose whether he/she would like to add additional requirements for their suggested restaurants. The user can choose their restaurant to: be touristic, have assigned seats, be suited for children, be romantic or not to choose any preference at all (or choose multiple preferences). Only the restaurants that fulfill all the users preferences will be suggested. If there are no restaurants left that fit all the criteria, the user will be able to restate their preferences. The system will set the state to “suggest\_restaurant” and the system will suggest the first restaurant to the user. At this point, the user has a few options. He/she can 1. ask for details about this restaurant (state 8 “give\_information”), 2. ask for another restaurant suggestion (“ask\_preferences”), or 3. end the dialog manager. For all possible options we’d suggest looking at Appendix Bor to take a look in the code. In the figure below you can see all possible dialog options:

|  |  |
| --- | --- |
| **Formal Template** | **Example (Formal)** |
|  |  |
| print(f"System: I am sorry, there are no restaurants with those preferences: "  f"Area: {self.preferences\_dict['area']}, "  f"Food Type: {self.preferences\_dict['food type']}, "   f"Price Range: {self.preferences\_dict['pricerange']}. "   "Please provide me with different preferences.") | "System: I am sorry, there are no restaurants with those preferences: "  "Area: north, "  "Food Type: british, "  "Price Range: cheap. "  "Please provide me with different preferences." |
| print(f'System: The restaurants have been filtered by the requirements: {requirements\_str}') | 'System: The restaurants have been filtered by the requirements: romantic' |
| print(f"System: The phone number for this restaurant is {restaurant\_info['phone']}") | 'System: The phone number for this restaurant is 01223 566188' |
| print(f"System: The address for this restaurant is {restaurant\_info['addr']}") | "System: The address for this restaurant is 21 burleigh street city centre ' |
| print(f"System: The postcode for this restaurant is {restaurant\_info['postcode']}") | "System: The postcode for this restaurant is c.b 2' |
| print(f"System: I suggest {restaurant\_name}. It serves {restaurant\_food} food in the {restaurant\_area} area and falls within the {restaurant\_pricerange} price range.") | "System: I suggest midsummer house restaurant. It serves british food in the centre area and falls within the expensive price range." |

1. Data

This section discusses the data used by the restaurant recommendation system, specifically the dialog acts dataset and the restaurant information set. It will detail the datasets’ crucial characteristics, the way they inform the dialog state transition diagram, with some accompanying examples.

* 1. The “all\_dialogs.txt” Dataset

**Domain**: restaurant recommendation domain. Consists of dialogues between users and a dialog system designed to recommend restaurants based on user preferences (preferences limited to location, price range, food type).

**Number of Dialogs**: 3235 distinct dialogs.

**Dialog Format**:

* **Session ID**: unique ID for each conversation.
* **Turn number**: Tracks the sequence of interactions between the user and the system.
* **System prompts**: These are the messages generated by the dialog system to guide user input.
* **User responses**: User messages, typically in the form of requests or confirmations. These are annotated with speech acts like "inform", "request", "affirm", and others.

**Duplicate Utterances**: these were initially present in the dataset, however, through data cleaning conducted as part of the beginning stages of the project the duplicates were removed from the set, allowing for a more objective training of the machine learning algorithms.

**Language and Grammar**: Simple language and grammar is employed by users, while a programmatic approach to prompts is adopted by the system. Assumingly for the purposes of the project, the dialogs of the users are simple and straightforward, and all only request restaurants based on the 3 criteria. Some incomplete or ungrammatical sentences. System performance is likely to be affected by the errors in spelling, but this also captures well the nature of actual interactions with an automated system and their challenges, and so was kept in the final, used dataset.

**The “restaurant\_information.cvs” Dataset**

**Domain**: basic information about various restaurants, including names, location, food types, and contact details. This is the essential reference table of the restaurant recommendation system.

**Number of Records**: 109 individual restaurant records.

**Data Columns**:

1. **restaurantname**: name of each restaurant.
2. **pricerange**: cheap, moderate, expensive.
3. **area**: east/west/north/south/center
4. **food**: type of cuisine offered (e.g., Italian, British, Seafood).
5. **phone**: contact number for the restaurant
6. **addr**: The address of the restaurant
7. **postcode**: The postcode for the restaurant

**Dialog State Transition Diagram**

The state transition diagram (see appendix) of the restaurant recommendation system was developed based on characteristics observed in the dialogs in the “all\_dialogs.txt” dataset. Each user interaction, whether an inquiry about food type, price range, or location, was modeled as a distinct state in the system. The dialog acts, such as informing, requesting, confirming, or thanking, were used to define the transitions between states.

Here are a few examples of how the dialog dataset informed the state transition diagram design:

During the welcome state, the system initiates the conversation with a prompt like, *"Hello, welcome to the Cambridge restaurant system. You can ask for restaurants by area, price range, or food type. How may I help you?"* (session id: voip-da10d74c3e-20130328\_141500)​. We use a statement like this to prompt the user to provide us with the necessary information to recommend a restaurant. Based on the user’s input we transition from the welcome state through the ask… states.

When handling user preferences, the system either requires more information or checks for availability. For instance, the user may say, *"Is there a moderately priced restaurant that serves British food?"* (session id: voip-da10d74c3e-20130328\_141500)​. This would correspond to a state where the system verifies or confirms the query, checking the database before responding, which the “recommend restaurant” state in the diagram best exemplifies.

The successful dialogs in the records always resolve themselves with the system providing the user with restaurant details. The user requests final details like, *"Could I get their phone number?"* (session id: voip-0f41c16f2f-20130401\_235748)​ These pertain to either the phone number, address or postcode columns in the restaurant information table. After providing the information the system transitions to the final state where the user says, *"Thank you, goodbye,"* and the interaction ends.

1. reasoning

As an additional feature, in the file containing the 1C assignment, we have added an option to specify an additional feature. After checking whether the user has given input for the three preferences (area, food type and price range), the state transition function reaches the additional requirements state (Appendix C, state six). There are four possible options that can be given as an extra filter, those are stated by the system:

            "Type 'touristic' if you want a touristic restaurant,\n"

            "Type 'assigned seats' if you want the restaurant to have assigned seating,\n"

            "Type 'children' if you want the restaurant to be suitable for children,\n"

            "Type 'romantic' if you want a romantic restaurant.\n"

            "Type 'no additional requirements' if you have no additional requirements."

To achieve being able to filter on those features, we have expanded the restaurant\_info.csv file with the columns ‘food quality options’ (good/not good), ‘crowdedness’ (busy/not busy) and ‘length of stay’ (short/long), in accordance with the assignment of 1C. This is done within the add\_reasoning\_data(df) function.

After the user states their requirement, we update the possible restaurant dataframe that is saved in the State diagram class. The function applies inference rules that are relevant for the extra requirement(s) stated by the user. An example of a dialog is given below:

“Type 'touristic' if you want a touristic restaurant,

Type 'assigned seats' if you want the restaurant to have assigned seating,

Type 'children' if you want the restaurant to be suitable for children,

Type 'romantic' if you want a romantic restaurant.

Type 'no additional requirements' if you have no additional requirements.”

“You: I would like it to be a romantic spot”

“System: The restaurants have been filtered by the requirements: romantic”

“System: I suggest cotto. It serves british food in the centre area and falls within the moderate price range.”

When there is no restaurant findable that meets the extra requirements, the user is asked to make adjustments. This will be done (re-asked) until there is a restaurant that can be recommended to the user in accordance with the users extra requirements.

“Hey, it doesn't look like there's any places that fit what you want. Maybe switch up your demands a bit and check again...” (informal)

“I am sorry, there are no restaurants with those preferences. Please provide me with different preferences.”(formal)

Since we have chosen to explicitly state what the users options are, we think it’s fair to assume that they will be able to include one of them in their answer. But if they don’t, we don’t re-ask about the extra requirements. Therefore we don’t filter further on the possible extra requirements, and the system switches to the next state (7) ‘suggest a restaurant’. This is visible in the next example:

…

“You: I want a restaurant near the river Thames.”

“System: The restaurants have been filtered by the requirements:”

“System: I suggest saint johns chop house. It serves british food in the west area and falls within the moderate price range.”

Note that even though the system prints that the restaurants have been filtered by nothing (an empty string), in the code the system doesn’t call the function that filters on additional requirements. Due to time constraints, we haven’t excluded this statement nor replaced it with for example a statement that explains that the requested extra requirements are not possible to filter on. However, to us the most logical way to interpret this sentence is that the system has not filtered on any requirement. We therefore assume that most users will understand the message that the recommended restaurant doesn’t per se conforms with their request.

For some restaurants it holds that the conclusion of one applied inference rule is contradicting the conclusion of another applied inference rule. For example, a Romanian restaurant that is cheap and serves good food, will be touristic according to one of the inference rules, but not touristic according to the other. We have assumed that such contradictory conclusions imply that it is not clear whether the restaurant is touristic or not (in this example). Therefore, we have chosen not to include restaurants with contradictory conclusions in the updated possible restaurant dataframe. Because of this, it is known in the following example that the restaurant is not busy (according to inference rule 5) and you can stay there long (inference rule 6). A short clarifying statement is given to the user so that this is understood.

“System: Hello, welcome to the Cambridge restaurant system? You can ask for restaurants by area, price range or food type. How may I help you?”

“You: any type of food, any area but moderately priced”

“System: Please enter your preferences from the following options. Note that selecting preferences that contradict each other (e.g., a restaurant cannot be both romantic and not romantic) may result in no suitable recommendations being found.”

“Type 'touristic' if you want a touristic restaurant,”

“Type 'assigned seats' if you want the restaurant to have assigned seating,”

“Type 'children' if you want the restaurant to be suitable for children,”

“Type 'romantic' if you want a romantic restaurant.”

“Type 'no additional requirements' if you have no additional requirements.”

“You: romantic”

“System: The restaurants have been filtered by the requirements: romantic”

“System: I suggest restaurant alimentum. It serves modern european food in the north area and falls within the moderate price range.”

1. configurability

We have implemented four configurability options from the given list: (1) Use one of the baselines for dialog act recognition instead of the machine learning classifier. (2) Levenshtein edit distance for preference extraction. (3) Allow dialog restarts or not. (4) Use formal or informal phrases in system utterances.

The options are set by the system by explicitly asking what the users preferences are, before the start of the dialog. The values are saved and stored in the state diagram when the run method is called.

1. conclusion

In this project we tried to build a functional restaurant recommendation system using machine learning algorithms. At the beginning a set of classification models were tested to find the one with optimal performance. Subsequently, a state transition diagram was designed that served as the basis of the dialog manager’s interaction flow. The system was then refined to add configurability features such as formalized system responses and the use of alternative classification models.

As it stands the system is functional, but there is room for improvement. The dialog system handles structured queries relatively well but may struggle with more ambiguous or complex user inputs. Testing the system further to determine just how well it handles diverse and unexpected user inputs should be the first step in any improvement effort. Expanding the precision of location data, incorporating more restaurant characteristics and improving the conversational capabilities of the dialog manager have also been considered. In terms of alternative approaches, using NLP could enhance the system’s ability to interpret more complex queries. A voice-based interface could also offer a more natural and intuitive interaction for users. Additionally, incorporating real-time restaurant data, such as availability or reviews, could make the system more responsive and relevant.

Dialog act classification approach was done by experimenting with several machine learning algorithms to determine which would be most effective for categorizing user inputs. Linear regression was found to have the highest accuracy score, while decision trees performed better in terms of the F1 score. We chose to use the linear regression, due to its accuracy metric, but an experiment into decision tree performance would definitely be worth conducting as well. Additional improvements to the dialog act classification could include experimenting with other machine learning algorithms.

Overall, this project has delivered a functional basic restaurant recommendation system, with plenty of opportunities for further improvement.

1. overview of contributions by individual group members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| What did I do? | Amir | Jennifer | Thijme | Oliver |
| Initial assignment review |  |  | 1 h |  |
| Created github page and instructed everyone |  |  | 1 h |  |
| Looking at 1a code (made by team) |  |  | 30 m |  |
| Cleaning and fixing 1a code | 1 h |  | 2 h |  |
| Reorganised, deleted, fixed 1a code and put all code into classes |  |  | 1h 20 m |  |
| Add keyword matching and majority class classifier |  | 4 h |  | 2 h |
| Coding part 1A as a team | 4 h | 4 h |  | 3 h |
| Outline 1B |  | 1 h |  |  |
| State diagram B |  | 2 h | 3 h 30 m |  |
| lookup function |  | 1 h |  |  |
| Extract preferences function |  | 6 h |  |  |
| Coding 1b |  |  | 9 h 15 m |  |
| Explain progress to Oliver |  |  | 1 h |  |
| Code reasoning part |  | 4 h |  |  |
| Coding 1c (and 1a, and 1b improvements) |  |  | 18 h |  |
| Paper dialogmanager |  |  | 2 h |  |
| State diagram C |  |  | 1 h |  |
| Programming Functionality |  |  |  | 8 hrs |
| Programming Configuration |  |  |  | 5 hrs |
| Meeting |  |  |  | 2 hrs |
| Report Data |  |  |  | 13 hrs |
| Report Introduction |  |  |  | 3 hrs |
| Report Abstract |  |  |  | 1 hr |
| Report Reasoning and inspect state\_diagram |  | 5 h |  |  |
| Machine Learning Work |  |  |  | 4 hrs |
| Report machine learning | 5 hrs |  |  |  |
| State Diagram Design |  |  |  | 6 hrs |
| Integrating different code and fixing errors before submission | 6.5 hrs |  |  |  |
| Worked with team and brainstormed about 1b | 5.5 hrs | 3 h |  |  |
| Worked on logic for inference rules for 1c | 8 hrs |  |  |  |
| Decision tree classifier coding | 2 hr |  |  |  |
| Meeting and discussing with team | 3 |  |  |  |
| TOTAL | 35 h | 32 h | 40 h 5 m | 37 h |

1. appendices

**Appendix A: State transition diagram 1B**

A diagram of a flowchart

Description automatically generated

**Appendix B: State transition diagram 1C**

A diagram of a flowchart

Description automatically generated

**Appendix: Full dialog example**

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

You: hi, i wanna eat some italian food today

System: In what area would you like to eat?

You: i dont care about the area. any place will do

System: What type of price range are you looking for?

You: uhm, a moderate price range would be best

System: Please enter your preferences from the following options. Note that selecting preferences that contradict each other (e.g., a restaurant cannot be both romantic and not romantic) may result in no suitable recommendations being found.  
Type 'touristic' if you want a touristic restaurant,  
Type 'assigned seats' if you want the restaurant to have assigned seating,  
Type 'children' if you want the restaurant to be suitable for children,  
Type 'romantic' if you want a romantic restaurant.  
Type 'no additional requirements' if you have no additional requirements.

You: i'd like a restaurant thats romantic

System: The restaurants have been filtered by the requirements: romantic

System: I suggest pizza hut cherry hinton. It serves italian food in the south area and falls within the moderate price range.

You: what is their address?

System: The address for this restaurant is g4 cambridge leisure park clifton way cherry hinton

You: thats actually a bit far. do you have any other restaurant recommendations?

System: I suggest saint johns chop house. It serves british food in the west area and falls within the moderate price range.

You: oh i know that one! that one is great. Ill go there. thank you

System: I'm sorry, I didn't understand your request.

You: thats perfect! Thank you

System: You are welcome. Have a nice day!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dialog act** | **precision** | **recall** | **f1-score** | **support** |
| ack | 0.00 | 0.00 | 0.00 | 4 |
| affirm | 1.00 | 0.83 | 0.90 | 23 |
| bye | 1.00 | 0.38 | 0.55 | 8 |
| confirm | 0.82 | 0.69 | 0.75 | 13 |
| deny | 0.00 | 0.00 | 0.00 | 1 |
| hello | 1.00 | 0.71 | 0.83 | 7 |
| inform | 0.86 | 0.98 | 0.91 | 446 |
| negate | 0.96 | 1.00 | 0.98 | 22 |
| null | 0.57 | 0.08 | 0.15 | 48 |
| repeat | 0.00 | 0.00 | 0.00 | 3 |
| reqalts | 0.83 | 0.82 | 0.83 | 79 |
| request | 0.93 | 0.94 | 0.94 | 138 |
| restart | 0.00 | 0.00 | 0.00 | 1 |
| thankyou | 0.82 | 0.82 | 0.82 | 11 |
| **accuracy** |  |  | 0.87 | 804 |
| **macro avg** | 0.63 | 0.52 | 0.55 | 804 |
| **weighted avg** | 0.85 | 0.87 | 0.85 | 804 |

**Figure 1.** Summary of Logistic Regression scores with the cleaned dataset containing no duplicates.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dialog act** | **precision** | **recall** | **f1-score** | **support** |
| ack | 0.00 | 0.00 | 0.00 | 3 |
| affirm | 1.00 | 0.97 | 0.98 | 158 |
| bye | 1.00 | 0.84 | 0.91 | 37 |
| confirm | 0.96 | 0.76 | 0.85 | 29 |
| deny | 1.00 | 0.60 | 0.75 | 5 |
| hello | 1.00 | 0.94 | 0.97 | 18 |
| inform | 0.96 | 0.99 | 0.98 | 1492 |
| negate | 1.00 | 0.94 | 0.97 | 82 |
| null | 0.97 | 0.85 | 0.90 | 240 |
| repeat | 1.00 | 0.75 | 0.86 | 4 |
| reqalts | 0.96 | 0.95 | 0.95 | 277 |
| request | 0.98 | 0.99 | 0.99 | 951 |
| restart | 1.00 | 0.75 | 0.86 | 4 |
| thankyou | 0.99 | 1.00 | 0.99 | 525 |
| **accuracy** |  |  | 0.97 | 3825 |
| **macro avg** | 0.92 | 0.81 | 0.85 | 3825 |
| **weighted avg** | 0.97 | 0.97 | 0.97 | 3825 |

**Figure 2.** Summary of logistic regression scores with the regular dataset containing duplicates.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dialog act** | **precision** | **recall** | **f1-score** | **support** |
| ack | 0.25 | 0.25 | 0.25 | 4 |
| affirm | 1.00 | 0.87 | 0.93 | 23 |
| bye | 1.00 | 0.62 | 0.77 | 8 |
| confirm | 0.60 | 0.69 | 0.64 | 13 |
| deny | 0.00 | 0.00 | 0.00 | 1 |
| hello | 1.00 | 0.86 | 0.92 | 7 |
| inform | 0.88 | 0.94 | 0.91 | 446 |
| negate | 0.96 | 1.00 | 0.98 | 22 |
| null | 0.33 | 0.23 | 0.27 | 48 |
| repeat | 1.00 | 1.00 | 1.00 | 3 |
| reqalts | 0.81 | 0.77 | 0.79 | 79 |
| request | 0.95 | 0.89 | 0.92 | 138 |
| restart | 1.00 | 1.00 | 1.00 | 1 |
| thankyou | 0.85 | 1.00 | 0.92 | 11 |
| **accuracy** |  |  | 0.86 | 804 |
| **macro avg** | 0.76 | 0.72 | 0.74 | 804 |
| **weighted avg** | 0.85 | 0.86 | 0.85 | 804 |

**Figure 3.** Summary of decision tree scores with the cleaned dataset containing no duplicates.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dialog act** | **precision** | **recall** | **f1-score** | **support** |
| ack | 0.25 | 0.33 | 0.29 | 3 |
| affirm | 1.00 | 0.99 | 1.00 | 158 |
| bye | 0.97 | 0.97 | 0.97 | 37 |
| confirm | 1.00 | 0.86 | 0.93 | 29 |
| deny | 0.83 | 1.00 | 0.91 | 5 |
| hello | 0.95 | 1.00 | 0.97 | 18 |
| inform | 0.99 | 0.98 | 0.99 | 1492 |
| negate | 1.00 | 0.96 | 0.98 | 82 |
| null | 0.94 | 0.96 | 0.95 | 240 |
| repeat | 1.00 | 1.00 | 1.00 | 4 |
| reqalts | 0.95 | 0.98 | 0.97 | 277 |
| request | 0.98 | 1.00 | 0.99 | 951 |
| restart | 1.00 | 1.00 | 1.00 | 4 |
| thankyou | 1.00 | 1.00 | 1.00 | 525 |
| **accuracy** |  |  | 0.98 | 3825 |
| **macro avg** | 0.92 | 0.93 | 0.92 | 3825 |
| **weighted avg** | 0.98 | 0.98 | 0.98 | 3825 |

**Figure 4.** Summary of decision tree scores with the regular dataset containing duplicates.