Restaurant Recommendations Dialog System

YU-LAN DIBBITS[*], ALEXIA NTANTOURI[†], KAJA PHILIPPENS[‡], and PEPIJN STOOP[§]

Abstract

This paper constitutes the implementation of a dialog system for restaurant recommendations in the Cambridge area. The goal of this project is twofold. Namely, to build a machine learning model to classify the user preferences with dialog acts and create a text-based dialog system based on a dialog state transition diagram. The text-based dialog system is able to handle input from the user as well as generate system responses based on which dialog acts the user input is classified as. The dialog system is also provided with a reasoning component as well as a degree of configurability. For the machine learning models we created four models in total, from which the user can choose before starting the dialog due to the added degree of configurability. Results suggest that the Logistic Regression model has the best performance based on various evaluation metrics. Overall, the dialog system functions well, as it is able to handle a conversation with the user and work around things that are not explicitly mentioned in the dialogs of the dataset. There are, however, some possible improvements such as extending the scope of the project so that we could model the complexities of the dialog better. Other possible improvements could be re-evaluating the Levenshtein distance as well as looking at the balancing the data as well as the train-test-split for the machine learning models.

1 INTRODUCTION

Conversational agents, such as Siri, Alexa and Google Assistant, have been integrated in the daily lives of users for quite a while now, with the most recent development being ChatGPT [6] [3]. Moreover, these agents bring major components of Artificial Intelligence (AI) together such as reasoning, knowledge, learning and language understanding, This makes them both an increasingly important as well as interesting topic of research.

There are generally two kinds of conversational agents (i.e. dialog systems): chatbots and task-oriented dialog systems. Chatbots are designed to keep the user more engaged by mimicking informal human interaction and are usually either focused on providing information or social chat. Task-oriented dialog systems on the other hand, help the user achieve a specific goal such as booking a flight or finding a restaurant. These systems use spoken or text-based dialog to understand user requirements, by asking questions about relevant attributes. These questions are usually drawn from a small set of fixed attributes in a close domain, such as the price or cuisine of a restaurant [3, 10].

This paper focuses specifically on a task-oriented dialog system that is designed and built to recommend a restaurant in Cambridge based on the price range, cuisine and part of town that has been specified by the user. Our system consists of three components: a classifier for the dialog acts (Section 3), a dialog manager (Section 4) and a reasoning component (Section 5).

For the classifier, we constructed two baseline models as well as two classifiers to compare their performances. Then we modeled and constructed a dialog manager that chooses an action based on both the current state and the input of the user. We did this by implementing the state transition diagram as a state transition function

[*] Student number: 6679730
[†] Student number: 2958481
[‡] Student number: 6790526
[§] Student number: 9291164

Authors' address: Yu-Lan Dibbits, y.y.s.h.dibbits@students.uu.nl; Alexia Ntantouri, a.ntantouri@students.uu.nl; Kaja Philippens, k.m. philippens@students.uu.nl; Pepijn Stoop, p.j.c.stoop@students.uu.nl.

and using the dialog acts from the classifier to determine what course of action the program has to take. In general, the system takes initiative by asking the user about their preferences concerning restaurants. However, the program also allows the user to express a preference that the system did not ask for or does not exist. The user is also allowed to ask for more details about the restaurant.

Thirdly, we built a reasoning component that uses inference rules to determine additional restaurant properties based on the initial properties that were extracted by the dialog manager.

And finally, we implemented a degree of configurability in our dialog manager (Section 6). Specifically, we let the user choose from multiple classifiers, introduce a delay in the system's response, set the Levenshtein edit distance for preference extraction, allow users to change their preferences or not and allow preferences to be stated in a single utterance only, or in multiple utterances with one preference per utterance only.

2 DATA

2.1 Dataset

In order to train our classifier we used a dataset containing dialog acts and utterances relating to dialogs for restaurant recommendation. The dataset for dialog act classification consists of 25501 utterances, of which 20142 are duplicates, and is divided in two columns. The first column describes the type of dialog act and there are 15 distinct dialog acts as labels (ack, affirm, bye, confirm, deny, hello, inform, negate, null, repeat, reqalts, reqmore, request, restart, thankyou). The second column consists of user utterances.

The utterances were derived from a dataset that was obtained through the Dialog State Tracking Challenge (DSTC 2) [1]. This dataset consists of 3235 dialogs, all of which were collected using automatic speech recognition. Each dialog in the DSTC 2 dataset represents an interaction between a user and a system, where the user is given a task to obtain a recommendation for a restaurant according to a number of preferences. The exchanges of utterances between the system and the user were used to build the dialog state transition diagram.

There are some limitations to the dataset. First of all, the utterances in the DSTC dataset were collected in a spoken dialog system while our system is a text-based dialog system. This might have some consequences for the way the system performs, due to multiple reasons. One of them being that spoken language differs from chat dialog [5, 8]. Moreover, transcript errors might also have had an effect on system performance. This has for example led to some spelling errors in the data such as “dont” instead of “don't”.

Furthermore, there is a lot of literal noise in the dataset such as “unintelligible”, “noise” and “cough”. Although these utterances are labeled as “null”, this reduces the amount of usable data. Another limitation could be that it is only possible to assign one label, as there are some utterances that could be a combination of categories such as “thank you, bye” or “no, thank you”. Usually you can solve this by giving preference to certain words. For example, if a sentence contains both “thank you” as well as “bye”, “bye” is more relevant for classifying the dialog act (in this case as “bye”). However, this is specifically hard with negations, as “I do not care” is an inform statement, but “No I don't want thai food” is a negate statement.

Lastly, there are a lot of duplicates in the dialog acts dataset. This makes it that there is not a lot of unique utterances, so there are not a lot of different utterances to train the model on. This also causes some overlap between the train and test set when developing the classifier, which might also be problematic.

2.2 Dialog State Transition Diagram

2.2.1 Finite State Machine. The dialog state transition diagram was designed based on the general principles of a Finite State Machine (FSM) [11]. A FSM consists of a finite set of events Σ, a finite set of states Q, including one start state q_0 and one of more final states Q_m, and a set of transition functions between these states.

$$FSM = (\Sigma, Q, \delta, q_0, Q_m), \text{ where}$$

$$\begin{aligned} \Sigma&\text{: is the (finite) event set,} \\ Q&\text{: the (finite) state set,} \\ \delta : \Sigma \times Q &\longrightarrow Q \text{ the transition function,} \\ q_0&\text{: the initial state,} \\ Q_m&\text{: the marked (or final) states.} \end{aligned}$$

The final diagram for part 1B (Figure 3 in the Appendix) consists of 10 states (Q) of which the 'WELCOME (1)' state is the start state q0 and 'END (10)' is the end state Qm, while the final diagram for part 1C (Figure 4 in the Appendix) has one more state, the ADDITIONAL PREFERENCES (11) state. The set of events, denoted by strings above the arrows, are triggered by a combination of both user input and the response of the system, and consists of two types. The first type is a boolean ('yes', 'no') that denotes whether or not correct user input was obtained to transition to the next state. The second type consists of dialog act utterances such as 'deny' or 'else', which refers to the detection of other classes than those required to transition to other states, which denote that the user input should be classified as a specific dialog act utterance to proceed to a specific state.

2.2.2 Diagram Development. As stated before, Figure 3 and Figure 4 show the final diagrams for part 1B and part 1C, respectively, in the Appendix. The data for the design of the diagrams was manually explored. In the first part of the dialogs, the program tries to extract the preferences from the user in order to find a restaurant. This is achieved by asking questions until all of the preferences are extracted. The next step is trying to find a restaurant. If no restaurant is found, the user can try to find something else and the program starts to extract preferences again, or they can end the conversation. If there is a matching restaurant, the program prints the restaurant. Now the user can either ask for more recommendations or ask for details about the restaurant already provided by the program. If there are no more restaurants despite the user asking for more, they can either start again or end the conversation. For the initial restaurant the user can request more details or end the conversation.

For the development of the diagrams, multiple design choices were made. First of all the diagrams are implicit as they do not contain nodes representing user utterances. This design choice was made to maintain the diagrams' simplicity by adding specific user responses as events to the arrows instead of creating additional nodes.

Second of all, loops were incorporated between the third (ASK FOOD), fourth (ASK PRICE), and fifth (ASK AREA) states and their respective system utterances to facilitate input validation, as the dialog system (see Section Dialog Manager) requires valid preferences before proceeding with the search process.

Thirdly, after the seventh (NO MORE RECOMMENDATIONS) state, a loop was introduced to return to the first (WELCOME) state, allowing for the initiation of a new search process if no restaurants were found and the user indicated a desire to search again. And finally, no loop was incorporated between the eleventh (ADDITIONAL PREFERENCES) state and its respective system utterance because valid additional preferences are not required for the system to proceed to another state. When the user gives invalid additional preferences, it is prompted again for correct input, but it can always choose to proceed to a new state by not giving new input.

Example dialog snippets corresponding to each state are given in the Dialog Manager Section.

3 MACHINE LEARNING

We trained two classifiers for predicting the dialog act of a user utterance. In order to be able to compare these classifiers, two baseline models were implemented. In total we created four systems, namely a Majority Label baseline model, a Rule-Based baseline model, a Decision Tree model and a Logistic Regression model.

3.1 Baseline Models

For the Majority Label system, we built a function that counts the occurrences of the unique labels in the 'label' column, selects the most frequent label from that list and returns that label. We then built another function that takes a list of test utterances from a train-test-split and assigns that most frequent label as the 'predicted label' to each test utterance.

For the Rule-Based system, we manually selected characteristic (sets of) keywords and phrases for each label by looking at patterns in the utterances. We collected these keywords and phrases in a dictionary with the corresponding labels as the keys. We then build a function to iterate through the given user input and to detect whether any of the selected keywords or phrases is in the input string. If that is the case, the function returns the associated label. If none is detected, the majority label 'inform' is returned. We kept on adding new keywords and phrases until an accuracy rate of 0.8 was achieved in the classification process of the whole restaurant csv dataset.

3.2 Machine Learning Models

For dialog act classification, two machine learning models were trained: a Decision Tree model and a Multinomial Logistic Regression model.

The Decision Tree model was chosen due to its robustness in handling highly skewed data, making it well-suited for the imbalanced dialog act dataset (see Section Data Preprocessing). Additionally, it is relatively simple to implement and interpret compared to more complex models like feedforward networks [9]. The standard hyperparameter settings of the model were used since those showed sufficient performance during the training phase: the quality of the splits was measured by the Gini index and a maximum depth limit of the tree was not set.

The Logistic Regression model was selected for its ease of implementation and interpretation, as well as its resistance to overfitting, which made it a suitable choice for the relatively small dataset. The maximum amount of iterations for the model was set to 400 to ensure convergence during training. The limited-memory BFGS algorithm was applied for solving the optimization problem because of its limited hardware needs.

The design of both the Decision Tree and the Logistic Regression models consist of a trained Bag-Of-Words (BoW) vectorizer model and the classifier itself. For training both the vectorizer model and the ML-models, we used the scikit-learn Python library.

The BoW vectorizer model creates a feature representation of the dialog act data by creating vector representations based on the word frequency in the utterances [4]. Since BoW models do not include word order in these representations, the word order in the utterances of the dataset was not preserved during vectorization.

3.3 Data Preprocessing

The dataset for dialog act classification consists of 25501 dialog instances (or utterances), of which 20142 are duplicates (check Data Section). Figure 1 shows the distribution of each label in the dataset. The distribution shows that the “inform” label is the majority labels with around 11000 instances, which is around 44% of all the dialog instances, followed by the ’request’ label.

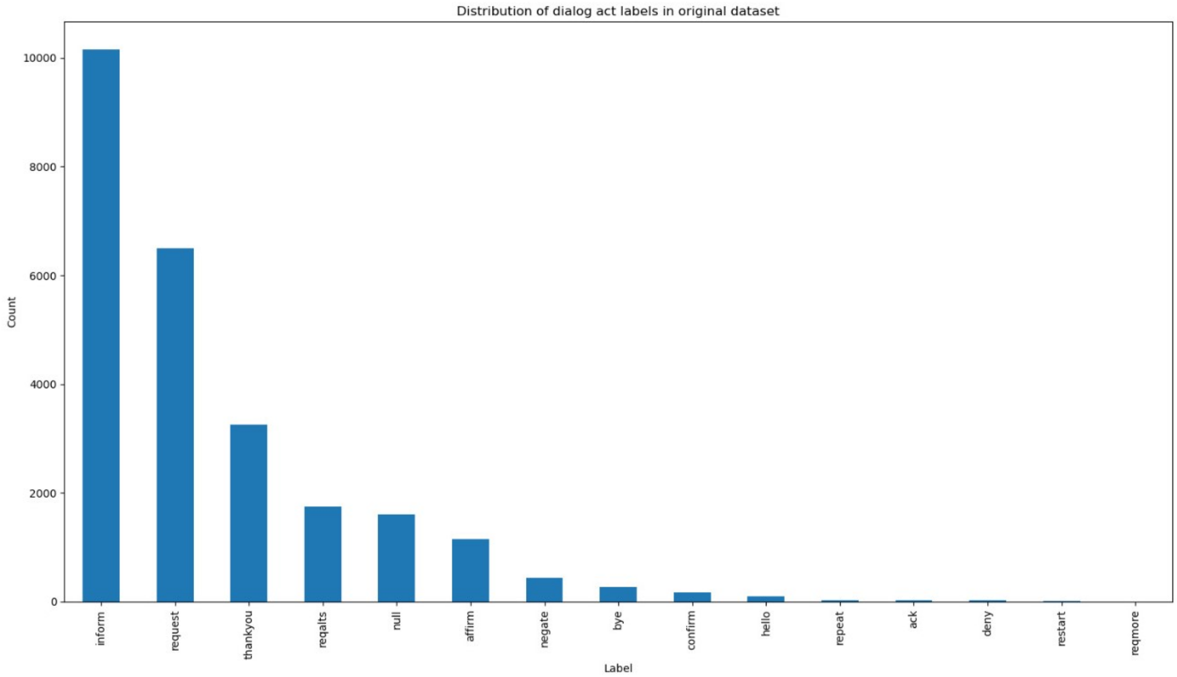

Fig. 1. Dialog act labels distribution

Figure 2 shows the distribution of the utterance length, the number of words per utterance, of the dataset. The distribution is skewed heavily towards utterances with less than five words. Most of the utterances (more than 6000 utterance instances) consist of only one word, followed by 4-word utterances, which matches the observed distribution of utterance length used by humans in chatbot-interaction in previous research [2].

The original data consisted of duplicates of utterances. As stated before, this could affect the ML performance since the same utterance could end up in both the train and test data set after splitting. Therefore, a second dataframe was created without duplicates. To achieve this, duplicates of the same utterance in the first dataframe were dropped and only the first occurrence of each of those utterances was kept. The resulting dataframe was used as the second dataframe or ‘no duplications’ dataframe. Both dataframes were split into train and test sets in a 85/15 ratio, for which we used the sklearn *train_test_split* package.

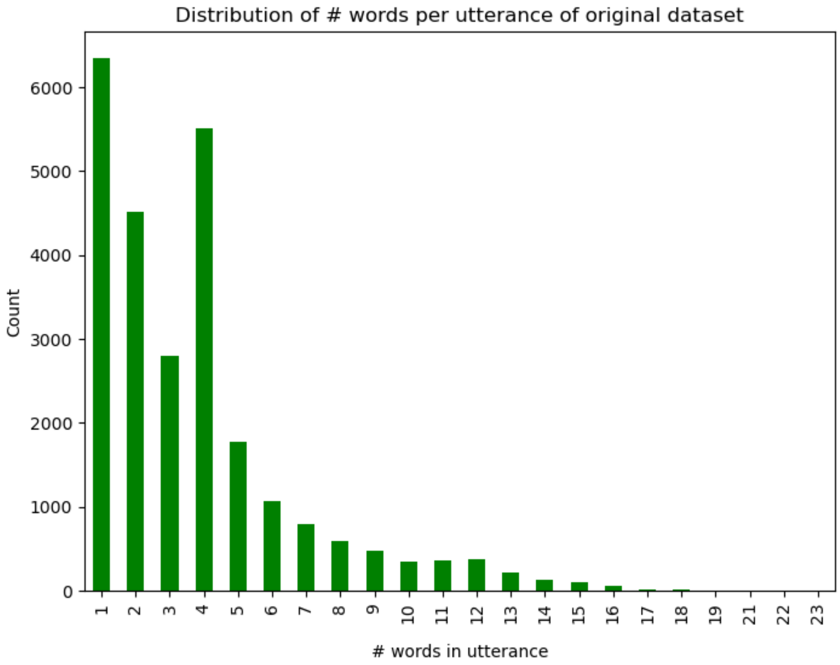

Fig. 2. Distribution of number of words per utterance

3.4 Quantitative Evaluation

To acquire insight into the performance of these systems, we evaluated each using accuracy-based evaluation metrics, of which we chose accuracy, precision, recall, and the F1-score, as seen in Table 1, below.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Majority Label Model	40.04	16.03	40.04	22.89
Rule-Based Model	83.56	85.72	83.56	85.93
Decision Tree with duplicates	98.07	**98.14**	98.07	98.03
Decision Tree without duplicates	87.31	88.08	87.31	87.57
Logistic Regression with duplicates	**98.27**	98.12	**98.27**	**98.18**
Logistic Regression without duplicates	90.17	88.82	90.17	88.97

Table 1. Performance metrics of different models.

Accuracy is a useful metric to quickly assess how well the system is performing, but there is a risk of smaller classes being misclassified. To mitigate this risk, we include the other metrics. Precision shows what percentage of the utterances being labeled as a certain dialog act is actually correct. Recall shows how many utterances of a certain dialog act have been correctly labeled as that dialog act. Lastly, the F1-score is the harmonic mean of precision and recall. Together, these metrics provide a more complete image of the performance of the systems than accuracy alone.

For a clear and more elaborate overview of the performance of each system, confusion matrices for each are included in the Appendix of this report.

3.5 Error Analysis and Difficult Instances

By analyzing the performance metrics (Table 1), the confusion matrices (see Appendix) and the plots (Figure 1 and Figure 2), we can identify significant trends in model performance.

Notably, there is a substantial class imbalance, particularly with the “inform” class dominating the dataset, which can lead to biased predictions favoring this majority class. Moreover, the utterance-length distribution is unbalanced as well, which could lead to prediction errors with longer utterances that are less represented in the data.

The performance also varies widely across different classes. Models consistently perform well in classifying “inform” and “request,” while they struggle with classes such as “ack,” “affirm,” “bye,” “confirm,” “deny,” and “hello.”

Additionally, there is frequent confusion among similar classes, notably between pairs like “ack” and “affirm,” “hello” and “inform,” and “request” and “reqmore,” suggesting semantic overlaps that complicate accurate classification.

The most difficult dialog act to classify appeared to be “acknowledge”. The Logistic Regression model had the most difficulty, with an F1-score of 0.00 for this class. An example of an utterance from that class from the dataset is “okay and what about the address”. It is labeled as an acknowledgement, but is not classified as such by the Logistic Regression model. The simplification of the dataset could be to blame for this. Obviously the utterance is more than an acknowledgement, it could also be classified as a request for example. However, the dataset only allows one label per utterance, which limits the possibilities and leads to such perceived errors.

One difficult instance is when there are conflicting keywords in the utterance. This is illustrated by the examples “is the food there good” and “is the food there any good”. The first utterance is labeled as a confirm act by both machine learning models, but the Decision Tree model predicts the label for the second example to be reqalts, while the Logistic Regression model predicts it to be inform. For a human speaker, the two sentences have the exact same meaning, but the conflicting keyword ‘any’ appears to create confusion for the systems.

Another example of a difficult instance is the use of colloquial language. For example, “I’m good” generally means “no” in colloquial English, but the systems do not classify it as such. The Decision Tree model expects it to be an acknowledgement, while the Logistic Regression model considers it to be an inform act. This might be resolved by using a larger and more varied dataset.

3.6 Models Comparison

Generally, the machine learning models performed better than the baselines. Out of the baselines, the Rule-Based model is better. The machine learning models also had higher accuracy on the dataset with duplicates. This is frankly unsurprising. The dataset without duplicates was roughly 20% the size of the original, and if 80% of the data is already seen then the performance is better. As a result, using a machine learning model with the full dataset, including duplicates, may provide an overly optimistic view of performance. Out of these models, Logistic Regression was somewhat better than the Decision Tree model. Therefore, Logistic Regression on a dataset with duplicates would be a logical choice for our dialog system.

4 DIALOG MANAGER

The structure of the dialog manager system is based on the Dialog State Transition Diagram (Figure 3 for part 1b and Figure 4 for part 1C, in Appendix). This structure contains two types of functions involved: a state-transition function and handler function.

4.1 State transition function

The state-transition function *′handle_transition′* handles the flow of the conversation by facilitating the transition process between different conversational states. The function achieves this by recursively calling handler functions based on current state in the conversational flow, received from a handler functions after the WELCOME (1)' state, and updating the current state according to these calls until the end state Qm (10) is reached in the process, when a goodbye message ("I hope I was helpful, goodbye!") is printed by the function.

4.2 Handler function

The handler functions facilitate the process of managing user input and system actions for a specific state. These functions can be recognized by the *_handler* suffix in their function names. Handler functions call other types of functions, except for the state-transition function, to obtain, extract and classify user input, print information for the user during the conversational flow and to make recommendations.

To show the general workings of both these types of functions, let us consider the following parts of example dialogs. A full example dialog is provided in the Appendix (Listing 1).

> System: Hello, welcome to the restaurant recommendations dialog system! You can ask for restaurants by area, price range, or food type.

The system starts by calling *handle_transition*(), the state-transition function, with the '1. WELCOME' state. The *welcome_state_handler*() is called immediately. This handler first invokes a delay preference (see Section Configurability) before printing the 'Hello, welcome...' welcome message. Then it asks the user for input before returning back to the first function.

> »west indian

After collecting the user input, the state-transition function invokes a transition to the '2. EXTRACT PREFERENCES' state. Then it calls the *ask_initial_preferences_handler*(), which calls functions that try to extract preferences (food, price and area), including 'don't care' preferences, from the user input. The handler checks if all preferences were extracted. Since the user gave no price range, it returns the '4. ASK PRICE REFERENCE' state to the state-transition function.

> System: What price range do you want?
> »any

The state-transition function invokes a transition to the *ask_price_handler*(), which collects user input and calls a function that tries to extract a price preference with that input. The handler function keeps asking the user for a valid preference before it invokes a return to the state-transition function.

System: I found 1 restaurant based on your preferences. Do you have additional requirements?
»yes

The system is in the '6. SEARCH FOR RESTAURANT' state, handled by the '*search_handler*' that prompts the user for additional requirements. The user answers affirmative, so the handler returns the '11. ADDITIONAL_PREFERENCES' state back to the state-transition function.

System: What is your additional preference?
»romantic
System: Ok, searching with romantic as additional preference... I found no restaurant based on your preferences. Do you want something else?
»no

The state-transition function receives the state and invokes the *additional_preferences_handler*, which asks the user for additional preferences, keeps prompting the user for a valid preference and searches for a new match once that is given. After no restaurant is found, the handler asks the user if they want something else, collects their answer and returns the '7. NO MORE RECOMMENDATIONS' state back to the state-transition function.

System: I hope I was helpful, goodbye!

The state-transition function invokes the *no_more_recommendations_handler* after receiving the state, which calls a function for dialog act prediction on the user input from the previous handler. The user input is classified as 'negate', so the handler returns the '10. END' state to the state-transition function. That function receives that state and responds by ending the conversation by printing the goodbye message.

4.3 System utterances

The dialog manager system uses informative, requesting and process-related utterances. A list of all utterances templates with examples is provided in the Appendix.

Informative utterances are dynamic phrases that provide information or responses based on user input and search results. For example,

"I have the following details for [RestaurantName] restaurant."

Requesting utterances are static phrases that include both error messages for invalid inputs, as well as responses when no matches or details are found, and prompts for user input, such as preferences or affirmative/negative statements. Examples of such utterances would be:

"What kind of food would you like?"
or
"I found no restaurant based on your preferences. Do you want something else?".

Lastly, process-related utterances inform the user about the system's internal processes and actions, like searching preferences or ending the conversation. Examples of these utterances include:

> "Hello, welcome to the restaurant recommendations dialog system! You can ask for restaurants by area, price range, or food type."
> and
> "I hope I was helpful, goodbye!".

5 REASONING

Furthermore, we have added a reasoning component to our program. When a user is selecting an appropriate restaurant, they need more information than just the price, area, cuisine, and contact information of the restaurant. For example, they might need to know whether the restaurant is romantic, touristic, or suitable for children.

These features can be inferred from other information. If a restaurant is busy, it is unlikely to be romantic, and if it is cheap and good, it is likely to be touristic. With the available information, as much as possible can be inferred about the restaurant, and the most fitting recommendations can be offered to the user.

For the sake of efficiency, first the primary preferences are obtained from the user. The system makes a filtered dataframe of all restaurants that suit the user's preferences and then asks if the user has any additional preferences. If that is the case, all inferences are made about the suitable restaurants, and the preference is applied to that selection.

The inference goes as follows, if the restaurant is cheap and good, its 'touristic' quality is set to 1, which denotes 'true' in our binary representation. Next, if it is Romanian, 'touristic' is set to 0, because Romanian food is less well known. We generally assume restaurants are suitable for children, but if the restaurant tends toward long stays, then it is unlikely to be suitable for children, so the 'children' quality is set to 0. However, it is likely to be romantic, so the 'romantic' quality is set to 1. Lastly, if the restaurant is busy, it is likely to use assigned seats, so the 'assigned seats' quality is set to 1, and it is unlikely to be romantic, so the 'romantic' quality is set to 0.

The order of these inferences matters, as the last inference has the last word. If a restaurant has long stays but is busy, then it is not romantic, for example. Thanks to this feature, contradictions are handled in an intuitive way.

The reasoning component is integrated in the dialog. The system does not elaborate on the inferences it has made, it reasons internally and recommends restaurants that fulfill the preferences based on this reasoning. An example dialog snippet is provided below.

> Do you have additional requirements?
> »Yes, I want it to be romantic.
> Ok, searching with romantic as additional preference...
> I found 3 restaurants based on your additional preferences.
> saint johns chop house is a nice restaurant serving british food.

In this case, the system knows that Saint John's Chop House is s suitable for long stays, so it infers that this is a romantic restaurant.

6 CONFIGURABILITY

We have implemented the following configuration options, some of which can also be found in the state diagram for Part 1C (Figure 4 in the Appendix).

(1) Allow the user to use one of the baselines for dialog act recognition instead of the machine learning classifier.
(2) Set the Levenshtein edit distance for preference extraction.
(3) Introduce a delay before showing system responses.
(4) Allow users to change their preferences or not.
(5) Allow preferences to be stated in a single utterance only, or in multiple utterances with one preference per utterance only.

When the system is run, it first asks the user which model to use for dialog act classification. The user can type '1' for the Linear Regression Model, '2' for the Decision Tree Model, or '3' for the Rule Based Model. Then the system asks which Levenshtein distance threshold the user wants to implement, and the user can choose '0', '1', '2', or '3'. Next the system asks if the user wants a small delay in the system responses. Such a delay can make the conversation feel more natural and trustworthy. The user can enter '0' for no delay, or '1' for a delay of one second. After the user has set these three configurations, the conversation starts.

These user options are passed to the restaurant recommendation system, to be saved as variables that can be used during the conversation. The user's chosen model is imported or trained, their chosen Levenshtein edit distance is used in the code to set the actual Levenshtein distance, and their chosen delay is implemented any time the system replies to the user.

Finally, after a restaurant has been recommended, the user has the option to change their preferences to get a new recommendation. Also, if all restaurants that fulfill the preferences have been recommended, the system asks the user if they want to change their preferences. The optionality of this feature covers configurability 4. If the user decides to change their preferences, the system returns to the 'EXTRACT PREFERENCES (2)' state.

Another feature we added was to allow preferences to be stated in a single utterance only. Specifically, after the system's welcome message, the user can give several preferences in the same utterance, and after that, any missing preferences are obtained through specific queries by the system. Therefore, both sides of this feature are implemented: if the user wants to give all preferences at once, they can do so at the start. If they fail to do so, they can give their preferences one at a time.

7 CONCLUSION

The goal of this paper was to implement a dialog system for restaurant recommendations. This was done by using a dialog state transition diagram for generating system responses and handling input from the user. Machine learning models were trained to classify the user utterances, which contained the user preferences, with dialog acts. These dialog acts were then used to update the state of the system.

The dialog system was also provided with a reasoning component so that the system can recommend a more specific restaurant based on the extra preferences stated by the user. Additionally, some degree of configurability was added. Specifically, users are allowed to use one of the baselines for dialog act recognition instead of the machine learning classifier, set the Levenshtein edit distance for preference extraction, introduce a delay before showing system responses, change their preferences (or not) later on, state preferences in a single utterance only, or in multiple utterances with one preference per utterance only.

Overall, the system can handle a conversation to recommend restaurants. It is able to extract the necessary preferences as well as handle unexpected utterances. For example, if a user expresses something that is not handled by the system it states: “Please give a valid [food, price or area] preference”.

Nonetheless, the possibilities in conversation are limited. This is mainly due to the fact that the program is based on a finite state diagram. It would, for example, be desirable if the system could suggest restaurants that are very close to the user’s preferences instead of going back to asking preferences. This would, however, require more machine learning not only for labeling the dialog acts, but generating the dialog as well. Therefore, solving this would lie beyond the scope of this project.

Moreover, there seems to be a bit of a limitation with the implementation of the Levenshtein distance. For example, if the user types “want” or “yes” the word is matched to “west” when the Levenshtein threshold was set to equal or larger than 2. Although the system handles both “want” and “yes” as special cases, there could be other words that are not matched correctly. A possible improvement could be to test more intensively if there are other such errors with our distance metric. Furthermore, an alternative approach could be to use a different metric such as *pg_trgm* or the word mover’s distance. The latter for example, incorporates the implied meaning of words [7].

In order to label the dialog acts, we constructed two baseline models as well as two machine learning models. For the baseline models we created a rule based model and a majority label model and for the machine learning models we used a decision tree and logistic regression. There are, however, some limitations related to the classifiers. Specifically, regarding all four systems, there seem to be errors in classification which then hinders the flow of the conversation as the program is not able to follow the state transition diagram. One of the things that could contribute to classification errors are for example utterances that only have one label, when they could have multiple labels. Another limitation that could contribute to classification errors is the system having difficulties with handling negations. Nevertheless, accuracy of the models could be improved.

Improving the accuracy of the models can be achieved in multiple ways. First of all, adding the ability to have multiple labels or combined labels per dialog act. Although this would increase the complexity of the system. Secondly, we could improve the accuracy of the models by using data that is written dialog instead of speech-to-text transcriptions, as there would be less noise due to both duplicates as well as literal noise due to transcription errors. Finally, due to the duplicates, the dataset is unbalanced as there is quite an overlap between training and test set. This could be fixed with stratification, a technique that ensures each class or label is proportionally represented in both the training and test sets, preventing bias towards overrepresented classes. An alternative approach could be to use a different model, such as a multi-layer perceptron would be able to capture the more complex relationships that occur in dialog.

In conclusion, the development of the restaurant recommendation dialog system successfully demonstrates the integration of machine learning models with a finite state-based dialog manager to deliver effective and dynamic user interactions, with room for improvement in the system’s complexity.

8 OVERVIEW OF CONTRIBUTIONS

Table 2 shows how many hours each group member contributed to the project.

	Alexia	**Kaja**	**Pepijn**	**Yu-Lan**
Majority Label Baseline Model	3		0.5	
Rule Based Baseline Model	4		1	0.5
Decision Tree Classifier			4	
Logistic Regression Classifier			4	
Miscellaneous Tasks		4		0.5
Report Part 1A	1	2	3	
Design State Diagram	1	4.5	0.5	1.5
Implement State Transition Function	6			
Extract Preferences	2			
Lookup Function	2			
Implement States	7			
Add Block Comments	1.5	0.5	2.5	
Error messages			1	
Modify State Diagram	1	0.5	0.5	
Implement Reasoning	3			
Implement Configurabities	4	1		
Data Preprocessing			3.5	
Communicate with TA			2	
Final Report	4	6	7.5	20.5
Meetings and Tutorials	8	8,5	8	8
Total	**47.5**	**27hr**	**38hr**	**31hr**

Table 2. Overview of contributions

REFERENCES

[1] Matthew Henderson, Blaise Thomson, and Jason D. Williams. 2014. Dialog State Tracking Challenge 2. *Proceedings of the SIGDIAL 2014 Conference* (2014).

[2] Jennifer Hill, W. Randolph Ford, and Ingrid G. Farreras. 2015. Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations. *Computers in Human Behavior* 49 (2015), 245–250. https://doi.org/10.1016/j.chb.2015.02.026

[3] Daniel Jurafsky and James H. Martin. 2024. Chatbots Dialogue Systems. *Speech and Language Processing* (2024).

[4] Wisam Qader, Musa M. Ameen, and Bilal Ahmed. 2019. An Overview of Bag of Words;Importance, Implementation, Applications, and Challenges. 200–204. https://doi.org/10.1109/IEC47844.2019.8950616

[5] Gisela Redekker. 1984. On differences between spoken and written language. *Discourse processes* (1984).

[6] Kevin Reschke, Adam Vogel, and Dan Jurafski. 2013. Generating Recommendation Dialogs by Extracting Information from User Reviews. *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (2013).

[7] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. 2022. Re-evaluating Word Mover's Distance. *Proceedings of the 39th International Conference on Machine Learning* (2022).

[8] John C. Shafer. 1981. The Linguistics Analysis of Spoken and Written Texts. *Exploring speaking-writing relationships: Connections and contrasts* (1981).

[9] Yan-Yan Song and LU Ying. 2015. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry* 27, 2 (2015), 130.

[10] Yueming Sun and Yi Zhang. 2018. Conversational Recommender System. *SIGIR '18: The 41st International ACM SIGIR Conference on Research Development in Information Retrieval'* (2018).

[11] Junhui Zhao, Yi-Liang Chen, Zhong Chen, Feng Lin, Caisheng Wang, and Hongwei Zhang. 2012. Modeling and control of discrete event systems using finite state machines with variables and their applications in power grids. *Systems & Control Letters* 61, 1 (2012), 212–222.

A APPENDIX

A.1 State Diagrams

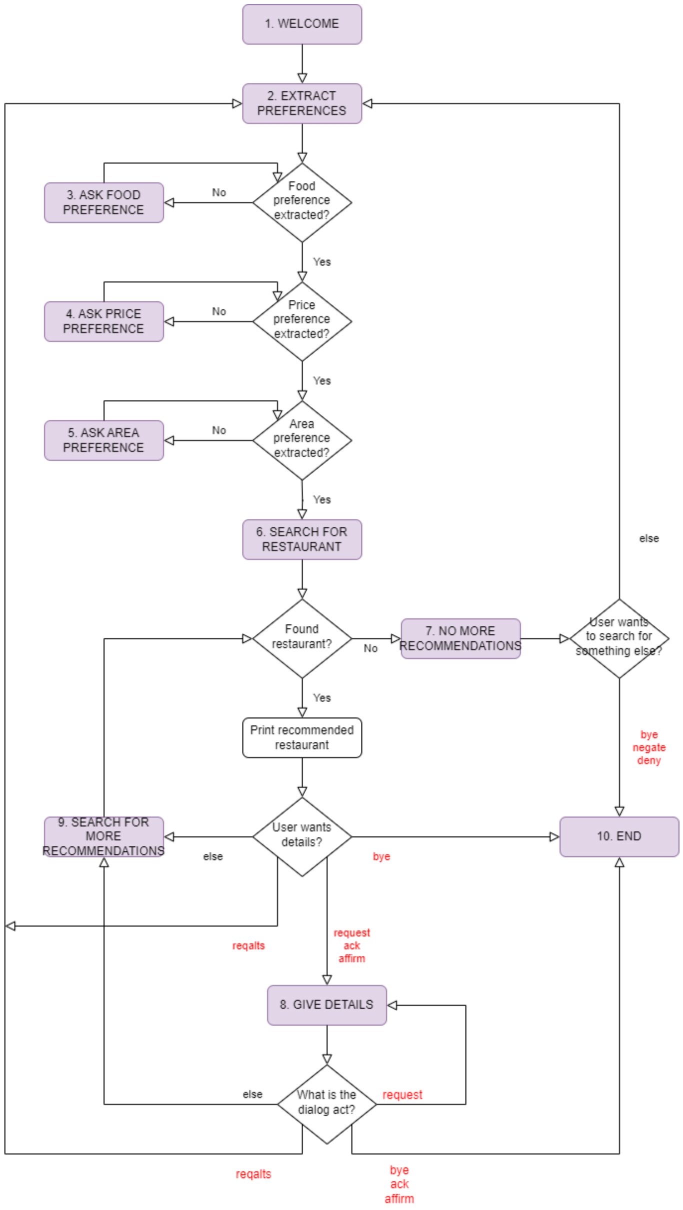

Fig. 3. State diagram for part 1B

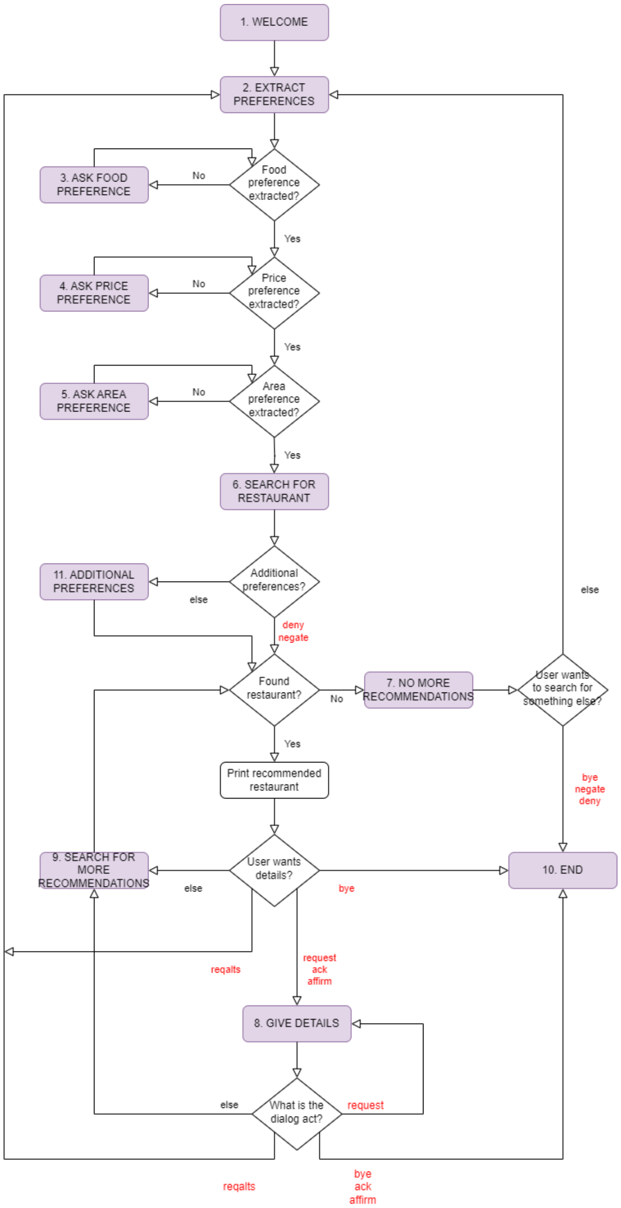

Fig. 4. State diagram for part 1C

A.2 Example Dialog

```
Options:
1. Linear Regression Model
2. Decision Tree Model
3. Rule Based Model

Enter your model choice (1/2/3).
>>3

Set Levenshtein distance threshold for preference extraction.
(Enter '1', '2', '3' or '0' to ignore Levenshtein distance).
>>2

Do you want a short delay before showing system responses?
('1' for Yes, '0' for No)
>>0

Starting conversation...

Hello, welcome to the restaurant recommendations dialogue system!
You can ask for restaurants by area, price range, or food type.
>>Hello, I want a birtish restaurant at any area.
What price range do you want?
>>Whatever price.
Ok, I am searching for a restaurant based on the following preferences:
british restaurant at any price in any area...
I found  10  restaurants based on your preferences.
Do you have additional requirements?
>>Yes, I want it to be romantic.
Ok, searching with romantic as additional preference...
I found  2  restaurants based on your additional preferences.
fitzbillies restaurant is a nice restaurant serving british food.
Do you want details for this restaurant?
>>Yes.
What details do you want?
>>I want the address.
The address is 51 trumpington street city centre.
>>Can you also give me the phone number?
The phone number is 01223 352500.
>>Do you have any other recommendations?
the cambridge chop house is another nice restaurant serving british food.
Do you want details for this restaurant?
>>No, thanks.
There are no more restaurants to recommend. Do you want something else?
>>No, bye.
I hope I was helpful, goodbye!
```

Listing 1. Example dialog

A.3 System utterance templates and examples

We provide all the system utterance templates and corresponding examples.

A.3.1 Details State (9).

Example output: I have the following details for saint johns chop house restaurant:

```
print(f"I have the following details for {self.possible_restaurants[0]['restaurantname']} restaurant:")
```

Listing 2. Print restaurant details

Example output: The phone number is 01223 353110.

```
print(f"The phone number is {self.possible_restaurants[0]['phone']}.")
```

Listing 3. Print restaurant phone number

Example output: The address is 21 - 24 northampton street.

```
print(f"The address is {self.possible_restaurants[0]['addr']}.")
```

Listing 4. Print restaurant address

Example output: The post code is c.b 3.

```
print(f"The post code is {self.possible_restaurants[0]['postcode']}.")
```

Listing 5. Print restaurant post code

A.3.2 Search State (6) / Additional Preferences State (11).

Example output: Ok, I am searching for a restaurant based on the following preferences: british restaurant at any price in any area...

```
print(f"Ok, I am searching for a restaurant based on the following preferences: {self.food_preference} restaurant
      at {self.price_preference} price in {self.area_preference} area...")
```

Listing 6. Search for restaurant based on preferences

Example output: Ok, searching with romantic as additional preference...

```
print(f"Ok, searching with {self.additional_preference} as additional preference...")
```

Listing 7. Search with additional preferences

Example output: I found 1 restaurant based on your preferences.

```
print("I found ", len(self.possible_restaurants), " restaurant based on your preferences.")
```

Listing 8. Print number of restaurants found

Example output: I found 10 restaurants based on your preferences.

```
print("I found ", len(self.possible_restaurants), " restaurants based on your preferences.")
```

Listing 9. Print number of restaurants found (multiple)

Example output: saint johns chop house is a nice restaurant serving british food.

```python
print(f"{self.possible_restaurants[0]['restaurantname']} is a nice restaurant serving {self.possible_restaurants
    [0]['food']} food.")
```

Listing 10. Print restaurant recommendation

A.3.3 Search More State (9).

Example output: the oak bistro is another nice restaurant serving british food.

```python
print(f"{self.possible_restaurants[0]['restaurantname']} is another nice restaurant serving {self.
    possible_restaurants[0]['food']} food.")
```

Listing 11. Print another restaurant recommendation

A.4 Confusion Matrices

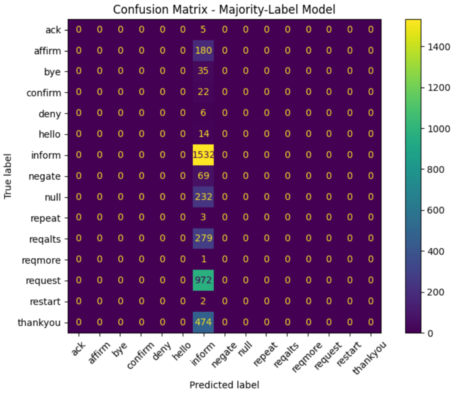

Fig. 5. Majority Label Model

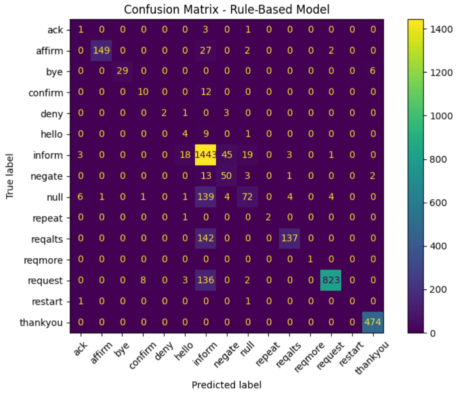

Fig. 6. Rule-Based Model

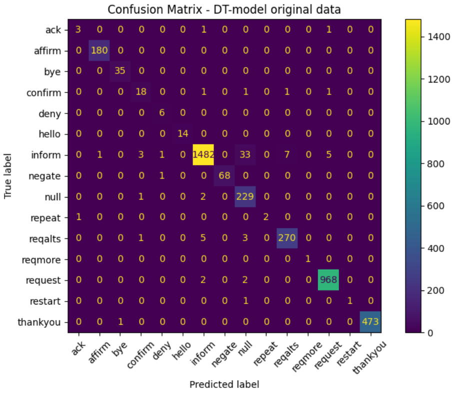

Fig. 7. Decision Tree (Original data)

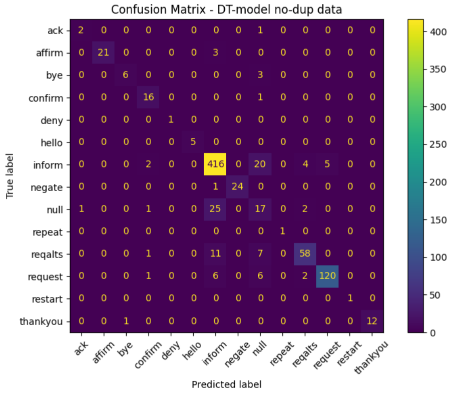

Fig. 8. Decision Tree (No-duplicate data)

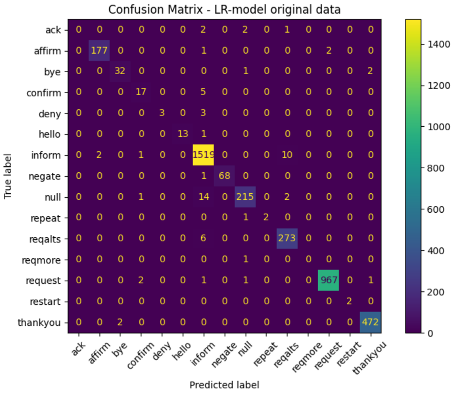

Fig. 9. Logistic Regression (Original data)

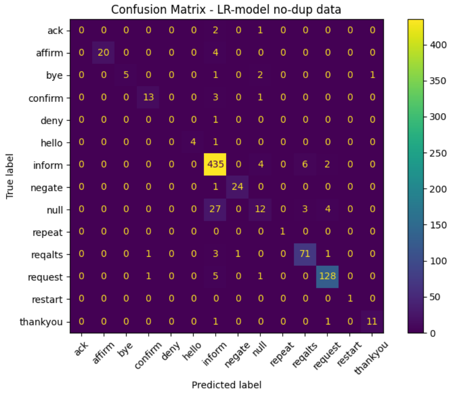

Fig. 10. Logistic Regression (No-duplicate data)