

# Hungry for Dialogue: A Restaurant Recommendation System

MICHAEL KONSTANTINOU (2784602) LOÏS DONA (2313715) FREDERIK VOM LEHN (6892299)  
MARCELLO EIERMANN (4940539)

The current project dealt with the implementation of a dialogue system that assists a user in finding a suitable restaurant based on personal preferences, such as price range or type of food. The dialogue system is capable of understanding user input and providing an appropriate response. A state transition function was used to control the flow of the conversation, i.e., the system classifies the user's utterances and determines how to proceed. The underlying machine learning model used a logistic regression classifier in addition with a keyword search approach. Furthermore, the system has a reasoning component, using logical expressions to provide more personalised results. Despite the limitations, it is a promising base for further improvements regarding recommendation systems in the restaurant domain.

Additional Key Words and Phrases: logistic regression, recommendation system, dialogue system, state based

## ACM Reference Format:

Michael Konstantinou (2784602), Loïs Dona (2313715), Frederik vom Lehn (6892299), and, Marcello Eiermann (4940539). 2021. Hungry for Dialogue: A Restaurant Recommendation System. 1, 1 (October 2021), 12 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

Automation is becoming more in demand due to the growth concerning human workload. Chatbots or dialogue systems are an example of automation, with which human workload is relieved by transferring the task of holding a conversation to a machine. Relieving human workload can be beneficial, because the human can focus on tasks that require the most attention and finesse. In this way, the quality of service that for example companies can provide will improve.

This project involved the implementation of a dialogue system that is able to recommend restaurants based on the preferences of a user. This application of automation is beneficial, because the user is able to find a restaurant quickly and human workload is relieved because the conversation is handled by a machine. The system is able to assess the status of a conversation using a state based approach, which is discussed in detail in sections 2.2 and 4.1. Transitions between states depend on the dialogue act of the user, which is defined by Bunt (1999) as [1]:

"The functional units used by the speaker to change the context"

Thus, a dialogue act can be for example a question or a command. A set of states and transitions have been defined using a data set containing example dialogues, in order to be able to match as many possible dialogue scenarios as possible. This data set and the construction of state transition diagram has been discussed in section 2.2. The implementation of this diagram as a state transition function for the system has been outlined in section 4.1.

---

Author's address: Michael Konstantinou (2784602); Loïs Dona (2313715); Frederik vom Lehn (6892299);  
Marcello Eiermann (4940539), {m.konstantinou1,l.m.dona,f.a.vomlehn,r.m.eiermann}@students.uu.nl.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2021 Association for Computing Machinery.

Manuscript submitted to ACM

Manuscript submitted to ACM

The encoding of the user’s utterances has been achieved by combining machine learning, keyword search and reasoning approaches, which are outlined in sections 3, 4 and 5 respectively. In short, a machine learning algorithm has been used to determine the dialogue act of the user, which is necessary for the state transition function. The data set that has been used for the training of the algorithm, is discussed in section 2.1. Additionally, the keyword search has been used to extract restaurant preferences from the user’s responses, even if they contain typos, and the implementation of a reasoning component allows the system to process additional requirements based on predefined features of restaurants in the database in combination with implemented inference rules. In addition to the general implementation of the system, seven configurable parameters were also implemented regarding the preference extraction, machine learning model, system responses and state transitions. What these options entail exactly has been explained throughout all sections and further details on the implementation are provided in section 6. Finally, the limitations of the system and how they could be improved are discussed in section 7.

## 2 DIALOGUE DATA

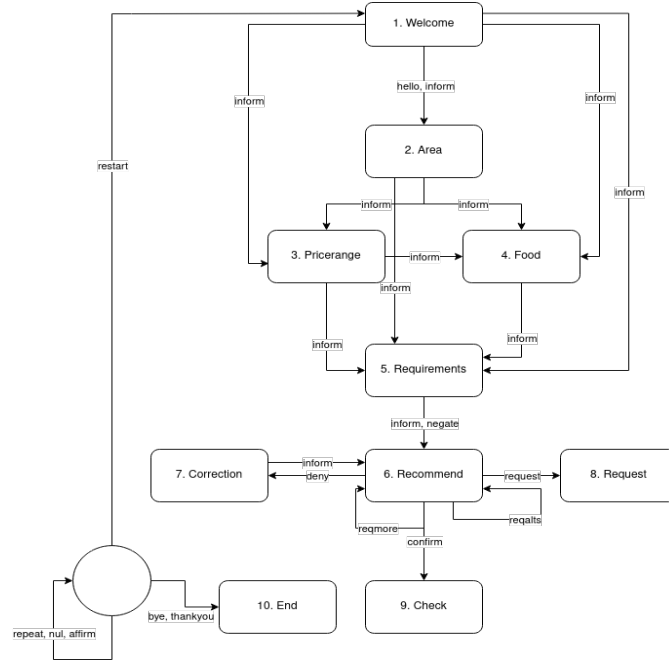


Fig. 1. The state transition diagram with the labels on the arrow corresponding to dialogue acts of the user. The circle state refers to the fact that the transition applies to all states.

A state transition diagram has been designed in order to graphically represent the flow of a conversation, which will be useful for the implementation of the dialogue system. This diagram has been designed by exploring a data set containing 3235 dialogues. By analysing a subset of these dialogues, the diagram has been constructed so that it correctly represents the dialogue examples from the data. States were moved, added or deleted by iteratively traversing the diagram using this subset of dialogues, making the diagram increasingly more applicable to the wide range of dialogues. It has been found that the most efficient way to determine transitions is based on mainly the users’ dialogue acts and sometimes

also the informational content of an utterance. Since the data revealed users using some dialogue acts, namely "restart", "repeat", "affirm", "bye", "thankyou" and "nul" at any time, an additional state was defined that represented all states at once.

Some things about the data were striking. Many of the dialogues represent user utterances that are relatively concise and ad rem. However, it could be that a user would talk to a chatbot in the same way as they would talk to a person, using complicated sentences containing irrelevant information and terms that the system might not recognize. That could make it harder for the system to filter out relevant information. Furthermore, the utterances mostly display the same grammatical structure and vocabulary, which possibly poses a problem when another user has a different style of grammar and vocabulary. The system might then have a harder time recognizing the user's intentions.

The resulting state transition diagram is shown in figure 1. As can be seen, the transition from state to state depends on the dialogue act of the user. And, according to the state, the system delivers suitable message. The state transition diagram in figure 1 can be illustrated further using the example dialogue below. At the start of every dialogue, the system is in state 1. The system transitions to a next state depending on the dialogue act of the user's response and possibly the informational content. In this case, the system first transfers to state 2, because the dialogue act is inform and the desired area of the user is not yet specified in the user's initial message. Next, it transitions to state 3, because the dialogue act corresponding to the response "any" is inform and the desired price range has also not been mentioned by the user yet. Then, because the user already specified their preference regarding food type after the welcome message, the system transitions to state 5, skipping state 4. After handling the user's response in state 5, the system transfers to state 6 and gives a recommendation if possible. Here, the user has an additional request, meaning the system transfers to state 8 in figure 1. After the system responds, the user ends the conversation and the system transfers to state 10 and provides a goodbye message. At any point in the conversation the user can restart the dialogue using a "restart" dialogue act, making the system return to state 1. In section 4, the implementation of this state transition diagram will be discussed in more detail.

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?

USER: im looking for a restaurant that serves seafood

SYSTEM: What part of town do you have in mind?

USER: any

SYSTEM: What kind of food would you like?

USER: seafood

SYSTEM: Do you have any additional requirements?

USER: no

SYSTEM: loch fyne serves seafood food

USER: can i get the address and phone number

SYSTEM: The phone number of loch fyne is 01223 362433 and it is on The Little Rose 37 Trumpington Street .

USER: thank you goodbye

SYSTEM: I hope you found a place to eat, bye!

State 7 and 9 did not occur in this example dialogue, so below other examples are shown to illustrate those states. State 7 is accessed when the user signifies that the recommendation of the system is not in line with their preferences, which is exemplified below.

USER: no I don't want seafood

State 9 is accessed when the user wants to confirm that the recommendation is indeed in line with the preferences. The dialogue snippet below shows an example.

USER: loch fyne serves seafood right?

### 3 MACHINE LEARNING

#### 3.1 Data Preprocessing and Analysis

Dialogue act	Description	Example sentence
ack	acknowledgment	okay um
affirm	positive confirmation	yes right
bye	greeting at the end of the dialog	see you good bye
confirm	check if given information confirms to query	is it in the center of town
deny	reject system suggestion	i dont want vietnamese food
hello	greeting at the start of the dialog	hi i want a restaurant
inform	state a preference or other information	im looking for a restaurant that serves seafood
negate	negation	no in any area
null	noise or utterance without content	cough
repeat	ask for repetition	can you repeat that
reqalts	request alternative suggestions	how about korean food
reqmore	request more suggestions	more
request	ask for information	what is the post code
restart	attempt to restart the dialog	okay start over
thankyou	express thanks	thank you good bye

Table 1. Example utterances belonging to labels in the data set

A data set containing a set of utterances with corresponding dialogue act labels has been used for training the machine learning classifier. It has been preprocessed first and then analysed. For reading and processing the data, the library Pandas has been used. The initial preprocessing step involved converting the data to a Pandas data frame object containing two columns. The data was split prior to each time the models were trained in 70% training and 30% test set using the Scikit-Learn library. For the implementation of the machine learning models, it was also necessary to convert the sentences in the data to bag of words vectors. This has been done by fitting a CountVectorizer to the training data in order to define the vocabulary and using it to transform the training and test data to vectors, also available from Scikit-Learn. Each item in a bag of words vector represents a unique word in from the data it was fitted on, also referred to as the vocabulary. The bag of words representation of a new sentence is made by setting each item in the vocabulary vector to the number of times it occurred in that sentence. Thus, words that are not part of the training vocabulary were completely ignored. The bag of words representations contained both unigrams and bigrams, in order to not only consider single words as features, but also possibly relevant word pairs.

The data set contained 25501 instances in total and an overview of the different labels and example utterances is provided in table 1. Figure 2 illustrates the class distribution. The data is heavily biased towards the inform label, which

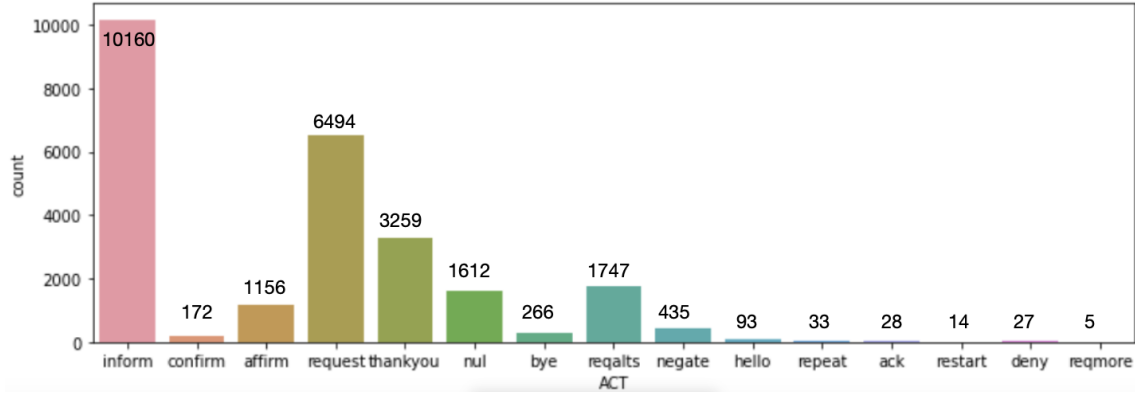


Fig. 2. Overview of the label distribution in the data set

occurs approximately 10,000 times, while the average occurrence of all labels is 1700. The average utterance length of a dialog sentence was 3.7 words with the longest sentence containing 23 words and the shortest only one word.

### 3.2 Implementation

A baseline system is a crucial component in assessing a model's performance. The machine learning model needs to perform significantly better than the baseline to be considered for use. Two different baseline models were implemented and used to evaluate the performance of two machine learning models. Our baseline model 1 always returns the most common class label in the training data. In our case, the "inform" class occurred the most, as can be seen in table 2. On the other hand, the second baseline system takes the utterance into account by finding the overlapping words between the class keywords and the utterance. The class keywords are lists of words relevant for each class. In the case of equal overlap between classes, the baseline model chooses the class which is most common in the training data.

Two different machine learning models were implemented using Scikit-Learn, of which one will be used in the dialogue system. A decision tree and a logistic regression classifier were trained using the training data and tested using the test data. The decision tree as well as the logistic regression model were parameterized using `class_weights = 'balanced'` to minimize the negative impact of the class imbalance. Furthermore, the maximum depth of the tree was set to fifty to prevent overfitting. The logistic regression model was chosen to be regularized using 'L2' regularization, also to prevent overfitting. The logistic regression model was also set to be suitable for multi-class classification. The performance of all four models has been measured by training and testing them ten times on different splits of the data, measuring accuracy and F1 score using Scikit-Learn. The F1 score was chosen to be included, because both precision and recall are equally important for the task of dialogue act classification.

### 3.3 Error Analysis

The mean performance measures of both the machine learning as well as the baseline models is documented in table 2. The table shows that performance is significantly better for the machine learning models compared to the baseline models, meaning that both models have sufficient performance. The comparable minimal differences between training and testing accuracy and F1 score show that one model does not overfit more than the other and that they both generalize well to unseen utterances. However, the decision tree model seems to be perform slightly less well than the

logistic regression model, when looking at the testing accuracy metrics. The logistic regression model has been used for the dialog act classification due to this superior performance. Apart from using logistic regression, the system has the configurable option to use baseline classifier 2 as a classifier instead.

Model	Training accuracy	Testing accuracy	Training F1 score	Testing F1 score
Baseline 1	N/A	0.40	N/A	0.04
Baseline 2	N/A	0.84	N/A	0.67
Decision Tree	0.97	0.96	0.98	0.87
Logistic Regression	0.99	0.98	0.96	0.87

Table 2. Overview of the models' performance

The analysis of the data and the use of the classification model did reveal some other issues. As described earlier, the distribution of the 15 classes in our specific training data set is imbalanced. The fast majority (76%) is made up of the top three following classes, "inform" with 39.8%, "request" (25%) and 12.7% "thankyou". The remaining 12 classes only occur in 24% of the cases. This imbalance leads to a over-classification of the inform, request and thank you classes. In contrast, the smaller classes are more often misclassified, because the model does not encounter enough examples to construct a good representation. In particular, the classification of repeat, acknowledge, restart, deny, and request more are highly susceptible to misclassification.

In addition to the imbalanced data, some characteristics of utterances seem to have a negative effect on classification. For example, utterances containing "something" or single word utterances are very often classified as "null", whereas the user could use it in a sentence that conveys preferences or requirements, which will unfortunately be looked over by the system. Moreover, when an utterance starts with a greeting, but also contains preferences, it is classified as the dialogue act "hello", causing no preferences to be extracted. Similarly, the informal "thanks" is not recognized by our system to be dialogue act "thankyou", while it is obvious it is. This causes the system to not ext when it actually should. All these consequences impair the system's user friendliness.

## 4 DIALOG MANAGER

### 4.1 State Transition Function

Every time a user interacts with the system, a state transition function is called which determines the next state based on the current state and the last dialogue act of the user. In addition, apart from the user's current response, a significant role is played by the user's current known preferences. As has been discussed already in section 2.2, states 2, 3, and 4 can be skipped if those preference were already made known during state 1.

At the beginning of a conversation the system is always in state 1 and the user can submit their first input. The user's input is decoded and based on the content the user's preferences are updated. It is clear to see from figure 1, that the next step is to check whether the area preference is known. If this is not the case then the system switches to state 2, where the system asks what area the user prefers.

The same principles apply for the other preferences; price range and food type. In these situations, the system switches to the states 3 and 4 respectively. For instance, if the user initiates the conversation with *"I'm looking for a restaurant that serves seafood"* then the system already knows the type of food should be "seafood". Following through the diagram, the system checks first whether the area preference is known. Since in this case it is not mentioned yet, the system will transition to the state 2 and ask the user's area preference. The same goes for the price range. However,

since the desired food type is already known, the system skips state 4. Preferences are extracted from the sentence searching for relevant keywords for each preference category with the configurable option of including a Levenshtein edit distance measure for handling any minor typos made by the user. Levenshtein distance is defined here as the number of deletions, insertions and substitutions need to be made in order to convert a typo to a keyword. If the user makes a typo, the keyword that is closest to the typo with a configurable maximum Levenshtein distance, which is set to three by default, is selected as a preference. This maximum distance After all these preferences are known to the system, the system asks if the user has any additional requirements, which can be processed through the reasoning component that is discussed in detail in section 5. It can make a recommendation by eliminating all restaurants from the database that do not fit the preferences and requirements. From the restaurants that are left, the system chooses a random one. If there are no restaurants left, the system will let the user know.

During state 6, the system might switch to the state 8 or visit state 6 again depending on the user's input. As the figure 1 depicts, the system will transition to state 8 if the user asks for more information such as the address, the postcode or the telephone number of the restaurant. In this case, the system will answer the request with the appropriate information, by looking it up in the database. For instance, let's assume that the system is in state 8. The system was able to understand what the user requested using a keyword search approach. For example, if the user asks "Can you give me the postcode?" then the system will extract the word "postcode" and look it up in the database. Similarly, the system can provide information regarding the address or the telephone number.

Moreover, the user can also request alternative or more suggestions after a recommendation has been made. For the first they can do so by saying "What about Italian restaurants?" after a recommendation for a Greek restaurant has been made for example. The system will adjust the preferences of the user using preference extraction and go back to state 6. For the latter, the user can ask for more suggestions and the system will select a different suitable recommendation than the one that has been given previously.

It can also happen that the user might try to confirm that the recommendation does indeed match their reported preferences. If this is the case, the system goes to state 9, checking if the extracted preferences match the preference that the user wants to check. Either way, the system lets the user know. Finally, when the user thanks the system or says good bye, the system goes to state 10, prints a goodbye message and exits.

It was already mentioned in section 2 that the user can restart the conversation at any point, making the system return to state 1 and reset by forgetting all the preferences and requirements the user had given. Though, this feature can be turned off, which makes it impossible for the user to restart a conversation.

## 4.2 System Utterances

The system uses a dedicated service class which is responsible for the system's output. The main components of this class are the:

- Private print function  
Overrides the default print function by handling configurable options such as displaying the system's utterances in all caps and using text to speech for the system's utterances. The text to speech component has been implemented using the pyttsx3 library.
- Print loading bar function

Its main purpose is to handle the configurable option that initiates a delay before the system responses. The function is responsible for showing a loading bar for a given amount of milliseconds, shown in figure 3. To clarify, the function is called only when the specific option is set to true and only if the system is in the states 2, 3, 4 or 6.

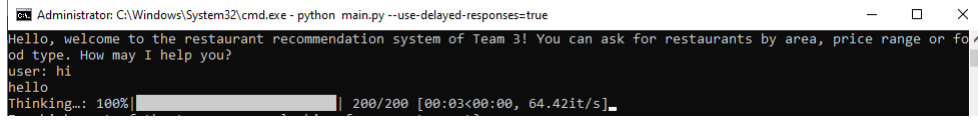


Fig. 3. Loading bar when user is about to extract user's preferences, or ask about specific preferences like Area, Price-range or Food

- Generate system utterances function

Based on the current state, as explained in section 4.1, the system generates an utterance. The function is responsible for handling the current state and invoking the function that will generate the correct system utterance. For each state, there is a dedicated function that is responsible for generating an utterance. For instance, when the system is in state 4, a dedicated function "ask\_food" is called that is responsible for asking for the user's preference regarding type of food. Each of these functions contain a list of possible outputs. Randomly, the system will output one of the items in the list. For instance, for displaying a goodbye message (end state), the following three sentences are included in the list of possible outputs:

- Have a nice day, bye!
- I hope you found a place to eat, bye!
- Bye!

Moreover, there exist two types of utterances, namely fixed and flexible. The flexible ones have a template that can be filled in with values, whereas the fixed utterances are predefined. The previously stated goodbye messages are fixed utterances, whereas the example below is a flexible one.

- The [request type] is [answer] →The *postcode* is *LL63 5YN*

To sum up the steps required to handle a turn in the dialogue, the system will handle the state transition first. Second, the system will call the respective function for the current state within the "generate system utterances" function, where after the system will randomly choose one of the available sentences of the state. Finally, the system will call the two print functions mentioned before that are responsible for any possible manipulation of the system output.

## 5 REASONING

Inference rules were used for the reasoning component of the system, deriving additional requirements using the existing properties. The following three properties were added to the database:

- Length of stay: short, medium, long
- Food quality: bad, ok, good
- Crowdedness: empty, moderate, busy, overcrowded

The following rules were implemented, where  $x$  represents a restaurant. The database was altered to include column referring to the antecedents of these rules containing boolean values. In the case of crowdedness or length of stay, the values were retrieved from the existing columns in the database.

- $Cheap(x) \wedge GoodFood(x) \rightarrow Busy(x)$
- $Spanish(x) \rightarrow LongStay(x)$



- $Busy(x) \vee Overcrowded(x) \rightarrow LongStay(x)$
- $LongStay(x) \rightarrow \neg ForChildren(x)$
- $LongStay(x) \wedge \neg Busy(x) \rightarrow Romantic(x)$
- $LongStay(x) \wedge GoodFood(x) \rightarrow Fancy(x)$
- $(Busy(x) \vee Overcrowded(x)) \wedge GoodFood(x) \rightarrow Healthy(x)$
- $(Empty(x) \vee ModeratelyCrowded(x)) \wedge LongStay(x) \rightarrow ForGroups(x)$

The reasoning component is used in state 5, the example dialogue snippet below shows how that proceeds. The user specifies that they want to eat thai food in any area and at any price. The list of possible restaurants that is left from those preferences is reduced further when the user specifies that they also want a romantic restaurant. The extraction of requirements is implemented similar to the preference extraction using a keyword search approach.

USER: I want thai food

SYSTEM: What part of town do you have in mind?

USER: Any

SYSTEM: Should I look for cheap, moderate or expensive restaurants?

USER: I don't care

SYSTEM: Do you have any additional requirements?

USER: The restaurant should be romantic

SYSTEM: What do you think of bangkok city

It is possible for a user to specify multiple additional requirements at once, but this could result in contradictions. For example, the rules implicitly specify that there exist no restaurants where you can stay for a long time and that are suitable for children. When a user specifies such contradictory requirements, the system will respond with a message notifying the user that there are no restaurants that fit their requirements and that they should alter their preferences in order to find a restaurant.

## 6 CONFIGURABILITY

The possible configurable options have already been explained throughout all sections. This section aims to give an overview of the options, how to access them and the implementation. The following configurable options could be activated by setting them to True using the command line as follows:

"python3 main.py -option=True, -another-option=True"

An exception is the option for adjusting the Levenshtein distance threshold, which should be set using an integer. This has been achieved by using the argparse library to retrieve the desired settings from the command line and set the parameters of the model accordingly.

- output-all-caps  $\rightarrow$  the system's responses are displayed in capital letters
- use-levenshtein  $\rightarrow$  enable the use of Levenshtein distance for preference extraction
- levenshtein-threshold  $\rightarrow$  set the threshold for a Levenshtein distance match to a given integer
- use-delayed-responses  $\rightarrow$  set a delay accompanied with a loading screen prior to giving a recommendation
- text-to-speech  $\rightarrow$  generate speech for the system's responses in addition to display on the screen
- baseline-2  $\rightarrow$  use baseline model 2 for the classification of dialogue acts
- no-restarts  $\rightarrow$  do not allow the user to restart the dialogue

## 7 DISCUSSION

This section elaborates on the limitations of the dialogue system, starting with the dialogue system in general, followed by the machine learning classification component. Possible improvements regarding these limitations are proposed, as well as alternative approaches that might also be successful.

### 7.1 Dialogue System

This system does not allow the use of anything else than preferences for the area, price range and type of food. Although some minor improvement regarding additional requirements was added, the truth is that the system lacks the following aspects:

- (1) Processing preferences that do not fall into the categories area, price range and type of food
- (2) Option of having multiple preferences in the same category
- (3) Processing preferences that are not available in the database

Regarding the second aspect, a user could prefer a restaurant which is located in the center or the east part of the city because they are both easily accessible for the user. However, this is not an option in the current system. The user would have to make two conversations with the system in order to find out possible restaurants that are located in both the center and east part of the city. This could be fixed by extending the extraction of preferences to involve lists of preferences for each category and retrieving restaurants from the data base accordingly.

The system can recognise whether the user talks about food, area or price range preference. In order to achieve this, the system searches for a wide range of keywords. However, the system cannot identify whether the user has some specific preference which just happens to be missing from the database, which brings us to the third aspect. For instance, if the database does not contain any "Kenyan" restaurants but the user asks for one, then the system will not be able to identify the user's preference. A solution for the first and third lacking aspect is to broaden the database to include more features. However, it would be more preferable to design a system which has a dynamic database, in which features mentioned by the user can be automatically retrieved and added to the data base using other machine learning and webscraping techniques.

There are some aspects of this system that could be approached differently. In the current system, the restaurants are stored in a csv file that we are importing and storing into heap memory. A perhaps more efficient solution would be the use of external API's that contain a list of restaurants, with up to date information. Such API's could be implemented from scratch using web technologies, or used from external services such as Google Maps or Trip Advisor. This approach would have the advantage that the system will always be up to date. If any changes will happen to a restaurant, or if a restaurant will be closed or created, then all we have to do is update the API. On the other hand, this would require a good connection to the internet and support of the API.

Traditionally, reasoning and inferences by the system could be implemented using agents and ontologies. Therefore, it would be possible (and in some cases preferable) to follow a different approach and create a knowledge based agent that can make inferences on the type of restaurant that would be more suitable for the user. Hence, instead of hard-coding the inference rules for identifying additional requirements such as "romantic" or "suitable for children", and instead of searching the database for restaurants with the same preferences, the system could have an ontology that holds this information. Moreover, using the reasoner and/or description logic, the system could make complex decisions based on its knowledge of the environment. This would require the design of an ontology that would capture the relevant information of our environment.

## 7.2 Dialogue Act Classification

There are many machine learning algorithms out there that one could choose to work with. Since the data were already labeled, it only makes sense to use a supervised machine learning algorithm. In addition, the purpose of this model was to **classify** the user's utterances into dialogue acts. Therefore, the options narrowed down to classification algorithms. The approach was to investigate two classification algorithms to find out which of the two has the highest performance.

A decision tree was the first algorithm that was chosen, because it is explainable. Explainability is a very important characteristic of systems that interact with humans, because users have right to an explanation when given information derived from a machine learning algorithm. On the contrary, algorithms such as deep neural networks do not provide this, as they work by the so called "black box" principle. Furthermore, decision trees are tolerant regarding data preprocessing, because they do not require any normalization or scaling of the data. Logistic regression was the second algorithm of choice, because the training time is often far less and it is also more explainable than other complex algorithms such as deep neural networks. Additionally, it is less prone to overfitting, meaning that our model had higher chances of achieving generalization.

Nonetheless, this is not the only possible approach. There are many popular algorithms such as support vector machines or neural networks that might have good results, although neural networks are not explainable. Moreover, extension of the single decision tree algorithm using random forest could also be beneficial, because they provide higher accuracy than single decision trees most of the time and are far less prone to overfitting than single decision trees. Thus, the current performance of the logistic regression model could possibly be topped by random forests, without overfitting.

Aside from machine learning implementation components, there are other aspects of classification that could be improved. A possible improvement regarding the data imbalance discussed in section 3.3 would be to skew the data so the occurrence of the dialogue acts would be approximately equal. As a result, the better the chances are for more accurate predictions for all dialogue acts. Moreover, the words and characteristics of sentences discussed in section 3.3 that can cause incorrect classification could be the cause of too little linguistic variation in the training data. Adding more varied vocabulary and use of grammar to the training data provides the model with the opportunity to learn a more broad spectrum of utterances corresponding to different dialogue acts, improving the generalization of the model.

The error analysis of the dialogue act classification has already been discussed in section 3.3, but a more general limitation of the classification will be discussed in the current section. A major limitation of this dialogue act classification strategy is that it is assumed each utterance only contains one dialogue act, whereas it is not uncommon to express multiple dialogue acts in one turn in a conversation. For example, a user could ask for more recommendations but at the same time also ask for the address of the current recommendation. The utterance could then be classified as either "request" or "more", resulting in the system only acting on one of the two dialogue acts, leaving the user unsatisfied. There is a possibility to work with confidence scores for each class when classifying dialogue acts, which would help with this limitation. For instance, when two labels have a very high confidence score compared to others, the system could infer that the sentence probably contains two dialogue acts and act accordingly.

Furthermore, it should be noted that the data were collected by human-computer interaction. Hence, the fact that the users were communicating with an automatic speech recognition program could influence their linguistic behaviour and style, which could be improved by collecting data from conversations between humans.

## 8 GROUP MEMBER CONTRIBUTIONS

Table 3 shows the tasks involved in the implementation of the dialogue system. Since this is a group project where multiple members work on the same repository, the git version control system was used. The planning of doing these tasks proceeded as follows for each part of the project. the group met in person at the beginning of each week to discuss the objectives for that week and make a plan of action. During the same week, each member of the group worked on their assigned task, working together where necessary. At the end of each week, there was another meeting to discuss the progress, work on remaining tasks and combine all the tasks together to create a functioning deliverable.

Task	Loïs	Marcello	Frederik	Michael
Data preprocessing (csv conversion, splitting, bag of words)	1	0	1	0
Baseline model 1	0.25	0.25	0.5	0
Baseline model 2	1	0.25	1	0
Decision tree model	0.5	0	0	0.5
Logistic regression model	0.5	0	0.5	0.5
Evaluation and error analysis of ML models	1	1	2	0
State transition diagram	2	2	2	2
State transition function	6	2	0.5	4
Preference extraction function	3	1.5	1	3
Integration of Levenshtein distance	1	0	0	3
Question answering component of requests from user	2	1	1	0
System utterance generation	2	1	2	4
Dialogue loop	2	2	2	2
Configurability	0.5	0	1	5
Reasoning rules	0	0.5	0.5	0
Application of reasoning rules to data base	0	3	0.5	0
Integration of reasoning in the system	1	0.5	0	0

Table 3. Overview of the contributions of the team members in hours

## REFERENCES

- [1] Harry Bunt. 1999. Dynamic interpretation and dialogue theory. *The structure of multimodal dialogue 2* (1999), 139–166.