

Creating a dialog system to recommend restaurants in the Cambridge area

Riccardo Bassani

6866840

r.bassani@students.uu.nl

Samuel Meyer

5648122

s.j.meyer2@students.uu.nl

Geanne Barkmeijer

6967280

a.g.barkmeijer@students.uu.nl

Wiebe de Vries

5713595

w.r.vries@students.uu.nl

ABSTRACT

The goal of this project is to build a text-based dialog system, which gives a restaurant recommendation based on the preferences given by the user. The dialog system is managed by a state transition function, a frame-based architecture which has the current dialog state and the user utterance as input and the next dialog state with associated system utterance as output. For the classification of the dialog acts we first implemented two baselines: a rule-based system based on keyword matching and a distribution based system, which classifies the utterances based on the proportions of dialog acts in a training data set. We then implemented a machine learning classifier, namely a decision tree. To measure the performance of the classifiers we used the F1-metric. The results show that the decision tree system performed best, having a macro average f1-score of 93,0%. The distribution based system performed worst, having a macro average f1-score of 6,6 %. The dialog system appears to be robust and able to handle a broad range of user utterances. However, limitations were found in classifying less frequent dialog acts, negations and “null” utterances. Possible improvements would be to use a larger data set, such that the system is able to recognize a higher number of user utterances. Also, the dialog system could be improved by handling negation and ‘null’ utterances in the rule-based system. Furthermore, the performance of the decision tree algorithm could be improved by tuning the hyperparameters of the decision tree or by using a random forest.

KEYWORDS

dialog system, dialog act, rule-based system, decision tree, machine learning, supervised learning

1 INTRODUCTION

Conversations and dialogs are fundamental to natural language. Nowadays, natural language processing applications, like conversational agents or dialog systems, are widely used for commercial purposes, like Siri, Cortana, Alexa and Google Now/Home. Since a dialog system communicates with a user

in natural language, it brings major components of artificial intelligence together, such as knowledge, reasoning, natural language understanding and learning. In this paper we discuss the design and implementation of a goal-based dialog system, the implementation and evaluation of a machine learning classifier for dialog acts and the implementation of different interaction features. The scope of this project is to build a dialog system which helps to achieve the goal of the user: finding a restaurant in the Cambridge area.

The type of system which is discussed in this paper is a task-based dialog system. This system is designed to recommend a restaurant, based on the preferences from the user. This system has typical short conversations to extract information from the user and is terminal/command line based: the user communicates with the dialog system on a screen via text. The architecture of this system is mainly designed around a system-initiative frame: in general the system controls the conversations by asking the user’s preferences, like the preferred food type, area and price range. Nevertheless, the architecture also allows speaker-initiative: the user is free to express a preference that the system did not ask for or to ask for more details of the restaurant, like phone number, zipcode or address.

Since the dialog system interacts with users, general human-computer principles apply to this dialog system [2]. First, we studied the users and tasks to understand the potential users and the nature of the tasks. Therefore, we investigated a similar system: a dataset of dialogs from the second Dialog State Tracking Challenge (DSTC 2). More details can be found in section 2. First we extracted all the couple utterances dialog acts to understand the type of utterances. Based on our findings, we then specified a domain ontology. This knowledge structure consists of a set of frames of information which should be acquired from the user. In this case, the system should acquire from the user his or her preferences regarding food type, price range and area.

This dialog system must be able to understand the user’s utterances. Each utterance of the user can be seen as a kind of action being performed by the speaker [5]. For example, the user can inform, acknowledge, affirm, request, etc. To be able to keep track of these actions, a dialog-state architecture, also called a believe or information state-architecture, must be implemented [3]. For the dialog-state architecture we implemented the state transition function. This state transition function keeps track of the user’s dialog acts and includes a policy for generating dialog acts of the system in return, like asking questions, making a proposal or rejecting a request (more information can be found in section 4).

For the implementation of the classifiers we chose a rule-based approach and a machine learning approach. The rule-based system was responsible to look for matching keywords in the user’s utterances. We chose for this approach, because a rule-based approach is good for narrow, scriptable applications, like restaurant recommendations (more details can be found in section 3.1). We therefore developed a relatively complex rule-based baseline to have a good touchstone against which we could measure the quality of the machine learning algorithm. The approach we chose for the machine learning algorithm is supervised machine learning for classification: decision trees. Supervised learning is used, since a model is learnt while using labelled instances, like dialog act classifications. Each user utterance was converted in a sparse array of binary values, where each feature represented the presence/absence of a word in the utterance. These features and the associated dialog act were then used as target to fit the decision tree. The training allowed the decision tree to generalize to new input of the user (more information can be found in section 3.3).

Furthermore, several features are implemented to improve the performance of the system. The typos in the user’s utterances are detected by using the Levenshtein edit distance and the user is asked to confirm or reject the corrections (more details in section 4.2). Also, the system asks confirmation after extracting one or more preferences from the user’s utterances. For this, the user demonstrates a part of what the system has understood the user to mean. Demonstrating whether or not the hearer understood the speaker is a strong method of trying to establish common ground [1]. Common ground is a set of things that are mutually believed by both the user and dialog system [4] and fundamental for any conversation or dialog.

These and other features can be turned on or off to observe the quality of the dialog system in various configurations. A detailed list of all the available features can be found in section 4.2.

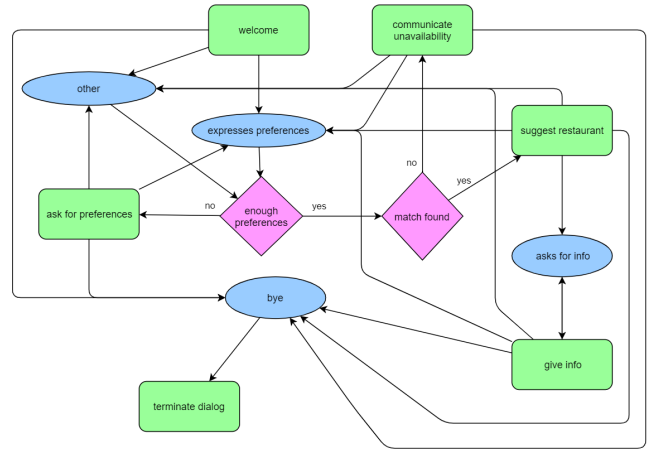


Figure 1: Dialog state transition diagram

2 DATA

For the design of our experiment and for the training of the machine learning classifier we used a data set made available to us, consisting of dialogs from the second Dialog State Tracking Challenge (DSTC 2). This data set is composed of 3235 dialogs as JSON files. In these dialogs a user is interacting with a restaurant recommendation system. Each dialog is split in two, one part describes the user input the system acquires, the other describes the output of the system. The relevant information (i.e. utterances and related dialog acts) is saved in easily accessible JSON attributes.

The data has not been fully cleaned, so noise will be prevalent. The way in which the noise presents itself is mostly in the form of actual noise, but sometimes also as a misrecognition of the users voice. This is because the text recognition software used by the creator of this dataset was not able to understand everything the user said. Every dialog is in english, there are some grammatical errors. These mostly stem from the system contracting words that should not be contracted, such as "kitalian" which is probably a contraction of like and italian. Some words are also misspelled since the system does not recognize nor use apostrophes. These grammar errors only affect the machine learning part of the model as the other ones do not take into account the grammar errors in the training set.

In Figure 1 the dialog state transition diagram is shown. We have captured the system’s behaviour in it, which we will illustrate below by giving a few example dialogs with their relevant path through the diagram. The diagram parts are mentioned in *italics* below.

S: Hello , welcome to the Cambridge restaurant system? You

can ask for restaurants by area , price range or food type . How may I help you?

U: im looking for an expensive restaurant that serves

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

U: cantonese food

S: Sorry there is no cantonese restaurant in the expensive price range

U: okay what about north american food

S: gourmet burger kitchen is a great restaurant serving north american food and it is in the expensive price range

U: can i have the phone number

S: The phone number of gourmet burger kitchen is 01223 312598.

U: can i have the post code

S: The post code of gourmet burger kitchen is C.B 2, 1 A.B

U: thank you good bye

In the dialog above the system starts out by being too quick to react. First it *welcomes* the user, then it thinks it only gets one preference. Therefore it *asks for another preference*, after that it gets a second preference and then it has *found no match*. This is *communicated* to the user. The user then expresses a *new preference*, the system updates the preference and a *match has been found*. The system *suggests a restaurant* to the user, the user then asks for *more info* twice. Then the dialog is terminated.

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

U: mexican food

S: chiquito restaurant bar serves mexican food

U: cheap

S: I'm sorry but there is no restaurant serving cheap mexican food

U: vietnamese

S: thanh binh serves vietnamese food in the cheap price range

U: phone number

S: thanh binh is a great restaurant

U: phone number

S: The phone number of thanh binh is 01223 362456 .

U: thank you good bye

In this dialog the user starts out *giving preference* for a type of food. The system *finds a match* and *suggests a restaurant*. The user expresses *another preference* on which the systems response is that this combination of preference is *not available*. The user then *respecifies* a preference, the restaurant *finds a match* and *suggests the restaurant*. After some noise the user *asks for more information* which the system provides, followed by this the *dialog terminates*.

3 MACHINE LEARNING

In order to identify how to respond to a user utterance, a machine learning classifier was implemented. The purpose of this classifier is to predict what kind of dialog act a user is trying to perform based on the words in the user utterance, so the system can respond appropriately. An utterance can be classified as one of fifteen different acts.

<i>dialog act</i>	<i>description</i>
ack	acknowledgement
bye	greeting at the end of dialog
confirm	check if given information confirms to query
deny	reject system suggestion
hello	greeting at the start of the dialog
inform	state a preference or other information
negate	negation
null	noise or utterance without context
repeat	ask for repetition
reqalts	request alternate suggestions
reqmore	request more suggestions
request	ask for information
restart	attempt to restart the dialog
thankyou	express thanks

Whenever a user enters an utterance, this utterance is transformed to lowercase, as is any utterance taken from the given data sets. This increases the accuracy of the classifiers. The act-utterance pairs in the data were extracted and split into two sets. 85% of the pairs became the training set, and the remaining 15% became the test set. In order to ensure that there was enough training data for every act, and to ensure that for every act the test data could provide proper performance measurements, a function was implemented to keep the same proportions of act occurrences in both test and training set as there was in the original set. This function repeatedly divides act-utterance pairs into the two test sets until both sets are within a set tolerance of having the desired act proportions. To be able to assess the performance of the machine learning classifier, two baseline systems were first implemented.

3.1 Rule-Based System

A rule-based system based on keyword matching was implemented. This system takes a user utterance as its parameter. First, a set of words and phrases were identified that, when part of an utterance, should directly classify it as a specific act. This is useful in cases where utterances of acts tend to be similar, except for a specific word. An example of this is in user utterances for an alternate request, which were very similar to those for a normal request. To ensure alternate requests, the trend that alternate requests seem to always

contain the keyword "more" was used to directly identify an utterance as an alternate request.

If an utterance contains none of this first set of words or phrases, an additional set of keywords typical to each user act is used. Every time a word in the user utterance matches a keyword for one of the dialog acts, the occurrence gets stored in a list. The dialog act with the most keyword occurrences for a user utterance is then chosen.

In the case that none of the words in the user utterance match any of the keywords, the assumption is made that the user is attempting to inform the system of some preference. This is because in the given data user utterances often contained instances where only preferences were given without the use of additional words. Given that the keyword list does not contain possible preferences, it is therefore likely that when no words match, it is one of these instances.

3.2 Distribution-Based System

Additionally, a distribution-based system was implemented. This system looked at the proportions of dialog acts of the classified user statements in the training data set. Regardless of what utterance a user enters, this system classifies it based on the found proportions. For example, if 40% of the user utterances in the training set were labeled as "inform" acts, the system has a 40% chance of classifying any user utterance as an "inform" act.

3.3 Decision Tree System

The assumption was made that, due to the limited number of ways specific acts are expressed in user utterances, a decision tree would be an adequate choice for the machine learning classifier. Because the aim was to use a bag of words approach, the features of the decision tree were set to boolean values for all words occurring in the test set. Each user utterance was transformed to a representation of these features, setting words contained in the utterance to the value of True. These representations were then used with their associated dialog act as target to fit the decision tree.

While there are several parameters to consider when fitting a decision tree, the standard parameters of the method used were kept, as these showed to perform quite well. Most importantly this meant that the quality of a possible split in the decision tree was measured through the Gini index, and the maximum depth of the decision tree was not limited.

3.4 Quantitative evaluation

In order to evaluate the different classifier systems we chose to compare their macro averaged f1-scores. We chose this metric because the classification distribution is skewed, and

Table 1: A sample of macro averaged F1-scores for the three classifier systems

Rule-based	Distribution-based	Decision tree
0.791	0.067	0.948
0.732	0.065	0.920
0.754	0.064	0.938
0.754	0.065	0.933
0.758	0.071	0.910
Average	Average	Average
0,793	0,066	0,930

this metric assigns all classes the same weight. The *Inform* acts happen a lot more frequently than the other acts. If we would use a metric such as accuracy the result could be an incorrect predictor of effectiveness. Because the system could be really good at predicting *Inform* acts and not so good at predicting other acts. The f1-scores of all three systems can be acquired by using the "0" command when the system boots. The results will vary slightly between separate runs because the training and testsets are built randomly from the data. In Table 1 five test macro averaged f1-scores were calculated from five separate runs. The best performer was the decision tree, with an average f1-score of over 90%, the worst performer was the distribution-based system. The Averages shown are taken calculated the f1-scores above. As to be expected both classifier systems perform much better than the distribution-based system. The decision tree also performs 17% percent better than the rule based classifier. Even if we tried to encapture more behaviour in our rule-based system it would be unlikely that we would get better results than the decision tree.

3.5 Error analysis and difficult instances

The overall performance of the decision tree classifier is good, with all the frequent dialog acts (i.e. support > 50) having a f1-score consistently above 90% and most of them often approaching or surpassing 99%.

The performance related to less frequent dialog acts are more difficult to assess due to the narrow support (several acts can appear 5 times or less in the test data) For instance, the high f1-score that can be obtained when analysing the recognition of "deny" acts by the decision tree is misleading, since the support for this particular dialog act is very narrow. In some cases, the dialog act decision tree appears to overcome the limitations of the rule-based baseline in the recognition of "deny" acts, as shown in the example below.

utterance: i dont want chinese
baseline act: inform (exp = deny)
dt act: deny (exp = deny)

Nevertheless, there are cases in which the decision tree still fails.

utterance: i dont want a spanish restaurant
baseline act: inform (exp = deny)
dt act: inform (exp = deny)

utterance: i am not looking for a chinese restaurant
baseline act: negate (exp = deny)
dt act: inform (exp = deny)

The impact of these errors can be reduced by activating the option that makes the system ask for confirmation after each expressed preference. Moreover, a "deny" utterance is usually pronounced after a wrong suggestion. Since the quality of the dialog system should make this scenario improbable, the user should be able to have a conversation without recurring too often to "deny" utterances.

Another case of difficult instances is represented by the "null" utterances. The f1-score for this act is not particularly low (around 90%), but there are cases in which both the baseline and the decision tree misinterpret "null" utterances and classify them as "inform" utterances.

utterance: ffjndjsfn
baseline act: inform (exp = null)
dt act: inform (exp = null)

utterance: empty_string
baseline act: inform (exp = null)
dt act: inform (exp = null)

In this case, the consequences are marginal: if a "null" act is classified as a "inform" act, the system response will be "*Can you express your preferences in a different way?*" instead of "*Could you please be more clear?*" and the dialog will continue without further complications.

The most serious issue is the inability of the classifier to identify some acknowledgment expressions. Below a brief extract of a conversation is reported; the acts wrongly identified are shown between brackets.

S: Ok, royal spice is a nice restaurant in the cheap range in the north of the town
U: ok (inform)
S: Can you express your preferences in a different way?
U: okay (null)

S: Could you please be more clear?

A possible solution to this problem is proposed in the discussion section.

Another case which has been analysed consists in the presence of apostrophes in the utterance. This caused the system not to be able to recognize utterances that would have been correctly classified in the absence of the apostrophes. This is due to the absence of apostrophes in the data the decision tree has been trained on.

utterance: i don't care
baseline act: inform (exp = inform)
dt act: null (exp = inform)

utterance: i don't want chinese
baseline act: inform (exp = deny)
dt act: null (exp = deny)

This problem was solved by removing the apostrophes in a pre-processing phase.

4 DIALOG MANAGER

4.1 The state transition function

The dialog is handled by a `dialog_manager()` function, which handles multiple turns by calling a `state_transition()` function until the end state is reached and the dialog terminates.

The state transition function takes as parameters the user input opportunely modified (after the lower-case conversion and some optional pre-processing operations) and a set of other parameters containing information about the state of the dialog.

Namely, among these parameters we find:

- `state`: an actual state. An overview of the four states is given below:
 - (1) `ASK_PREFERENCE_STATE`: the starting state, in which the system remains until it has collected enough preferences to suggest a restaurant; The system will go back to this state if the user desires to change his/her preference
 - (2) `SUGGEST_STATE`: the state in which the system enters once he has collected enough preferences to suggest one or multiple restaurants;
 - (3) `ASK_CONFIRMATION_STATE`: the state in which the system enter while handling a request of confirmation;
 - (4) `END_STATE`: the end state. Once reached, the dialog terminates;
- `old_system_utt`: the last utterance printed by the system;

- `expressed_pref`: a dictionary containing the current preferences;
- `reset_pref`: a dictionary containing the second to last preferences update;
- `suggestions`: a list containing the restaurant to suggest to the user.

At the beginning of the dialogue, the system is in the `ASK_PREFERENCE_STATE` and prints:

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

In the dialog state diagram `WELCOME_STATE` and `ASK_PREFERENCE_STATE` are displayed as distinct states to help the viewer understand the diagram flow, but in a computational context the `WELCOME_STATE` and `ASK_PREFERENCE_STATE` coincide and the welcome phase is implemented simply by setting the first system utterances as shown above.

Some acts, namely “bye”, “repeat”, “start” and “null”, are state-independent. Their meaning and their consequences are very understandable, then they have been omitted in the diagram for readability reasons (except from “bye”, which helps understanding the dialog flow). When an utterance act is identified as one of them, the state remains unchanged, the system performs the correct action and the `state_transition()` function returns. Therefore, the following discussion does not concern these acts. The only exception is the `ASK_CONFIRMATION_CASE`, that has the precedence over the “repeat” and the “null” acts.

The reaction to the other acts, on the other hand, depends on the state of the dialog.

In the `ASK_PREFERENCE_STATE`, if the first user utterance is classified as “inform” or “reqalts”, the system registers the preferences and suggests a restaurant or asks for another preference, depending on whether the minimum number of preference has been reached or not.

This decision is actually taken by the `manage_lookup()` function, which decides whether to call the `lookup()` function and returns to the `state_transition()` function the next state, the answer and the updated list of suggestions. This function is crucial and it is also called in other states.

The “reqalts” utterances are different in the extent that, if effective (i.e. a slot is filled), all the previously saved preferences are deleted. Below an example with two “inform” acts and `MIN_PREF = 2`.

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

U: I want Spanish food

S: Ok, what part of town do you have in mind?

U: North

S: Ok, sorry, I could not find any Spanish restaurant in the north of the town

If the user does not express a preference (i.e. the dialog act is not “inform” or “reqalts”) the system asks him for one, building an appropriate sentence depending on the user utterance. Some examples below.

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

U: I want more

S: Please tell me what you’re looking for before asking for more. What part of town do you have in mind?

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

U: thank you

S: You’re welcome, would you like something in the cheap, moderate or expensive price range?

As the dialog state diagram shows, this process continues until the minimum number of required preferences is reached. Once that happens, the system enters in a `SUGGEST_STATE`. Here if the user utterance is classified as “inform” or “reqalts” the `manage_lookup()` function is called and the next state is set consequently (i.e. depending on whether there are enough preferences).

A `COMMUNICATE_UNAVAILABILITY_STATE` (present in the diagram) is not necessary in the function since it is handled like a special case of suggestion.

Below an example with `MIN_PREF = 2`, where an extra preference modifies the suggested restaurant.

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

U: italian restaurant in the south

S: Ok, la margherita is a nice italian restaurant in the south of the town

U: moderate

S: Ok, pizza hut cherry hinton is a nice italian restaurant in the moderate range in the south of the town

The continuation of the dialog shows how the system reacts to different types of dialog acts. For complex acts like “reqmore”, “confirm” or “request” a specific function is called, while for easier acts like “thankyou” or “deny” the system simply replies with an ad hoc utterance.

A GIVE_INFO_STATE (present in the dialog) is not necessary, since the request is handles simply by calling the give_info() function.

U: more
S: There are no other restaurants meeting your criteria
U: ok, what is the address then?
S: You can find the restaurant at g4 cambridge leisure park clifton way cherry hinton
U: thank you
S: You're welcome, do you need anything else?
U: no
S: Oh, how can I help you then?
U: is it an italian restaurant right?
S: yes

If at least one of the properties ASK_CONFIRMATION and ASK_LEVENSHTEIN_CORRECTNESS is set to True, the system must be able to handle confirmation requests. For this purpose, a new state has been added: the ASK_CONFIRMATION_STATE.

The system does not exit this state until it receives an explicit confirmation/negation from the user. After this, the manage_lookup() function is called and the next state and the answer are set consequently. In the case of a negation the answer is set to a predefined string, as shown below.

S: Hello, welcome to the Cambridge restaurant system! You can ask for restaurants by area, price range or food type. How may I help you?
U: I'd like chinese food
S: Mmm... chinese restaurant, is it what you're looking for?
U: I love chinese
S: Mmm... chinese restaurant, is it what you're looking for?
U: yes
S: Ok, what part of town do you have in mind?
U: north
S: Let me see... chinese restaurant in the north of the town, am I correct?
U: I want something expensive
S: Let me see... chinese restaurant in the north of the town, am I correct?
U: no
S: Ok, what can I do then?

4.2 Dialog configuration

The dialog can be configured by modifying the value of some global variables:

- ASK_CONFIRMATION: whether confirmation is asked after every preference expressed;
- MIN_PREF: the minimum number of preferences to express before a restaurant is suggested;
- RANDOM_ORDER: whether the preferences can be expressed in a random order;
- CHANGE_POSSIBLE: whether it is possible to modify the preferences;
- RE_ASK_INEFFECTIVE_PREFERENCES: whether the user is asked to repeat when an expressed preference has no effects;
- ASK_LEVENSHTEIN_CORRECTNESS: whether confirmation is asked after levenshtein is used to predict a mistake;
- LEVENSHTEIN_LIMIT: the maximum levensthein distance accepted;
- MAX_LEVENSTHEIN_RATIO: the maximum ratio distance/length accepted;
- LEVENSHTEIN_PREPROCESSING: whether the system checks for typos as soon as the input is received;
- LIMIT_UTTERANCES: whether the dialog terminates once the max number of utterances is reached;
- MAX_UTTERANCES_NUMBER: the maximum length of the dialog;
- BASELINE_CLASSIFIER: whether the rule-based baseline is used instead of the machine learning classifier;
- RESTART_ALLOWED: whether the user is given the possibility to restart the dialog;
- ALL_CAPS: whether the system output is capitalized;
- RANDOM_SUGG: whether the suggestions are given in a random order;
- SINGLE_PREF: whether a single preference per utterance is accepted;
- INPUT_TO_LOWER_CASE: whether the input from the user is converted to lowercase;
- LOWER_CASE_ACT_IDENTIFICATION: whether the input is converted to lowercase during the identification phase. Necessary to test the upppercase CSV lookup without affecting the classification performance;
- TEXT_TO_SPEECH: whether text to speech is active for system utterances;
- PRINT_RESPONSE: whether system utterances are printed to console.

The Levenshtein edit distance is defined as the number of edit operations necessary to convert a string into another string of the closest domain term.

When an utterance is identified as an "inform" or a "re-qalts", for every word in the user utterance the dialog system compares it to the words in a JSON ontology containing correct terms for type of food, price ranges and town areas. To operate the comparison the system calculates the ratio of the Levenshtein distance and the word length and if it does not exceed an established threshold it edits the user's utterance and asks the user for confirmation by showing the edited string.

In the preprocessing phase the system operates in a similar way, but the following words are used as domain: "phone", "number", "address", "postcode", "restaurant", "location", "more", "about" and "repeat". This is essential not to cause typos to affect the classifier's performance, then no confirmation is asked in this phase.

4.3 Utterance templates

The dialog manager interacts with the user through four types of utterances:

- Ad hoc simple utterances:
brief utterances that do not need to be composed with variables, like:
S: You're welcome, do you need something else?
S: Ok, what can I do then?
- Utterances composed with ask_next_question() function:
the ask_next_question() function computes which one is the next preference to extract (randomly or following a precise order) and returns a string containing the question. This string is then optionally concatenated to a specific string depending on the state and the user dialog act.

S: Ok, what part of town do you have in mind?
S: Please cooperate. Would you like something in the cheap, moderate or expensive price range?
- Utterances composed with manage_lookup() function:
the manage_lookup() exploits the suggest() or the notify_absence() function in order to return the correct string.
These two both take advantage of a build_sentence() function, which builds a string starting from the preferences.

The string returned by manage_lookup() is then optionally concatenated to a specific string depending on the state and the user dialog act.

S: Ok, pizza express is a nice spanish restaurant in the cheap range

S: Ok, sorry, I could not find any spanish restaurant in the expensive range in the north of the town

- Utterances returned directly from a function:
The functions check() and give_info() return a string that is printed to console as is.
S: Yes
S: You can reach the restaurant by calling 01223 247877

5 DISCUSSION

The developed dialog system appears robust and able to handle a broad range of user utterances. The dialog act approach, indeed, allows the system to reply properly interpreting the user's intentions. It also permits it to answer to unexpected utterances by using expressions like "Could you please be more clear?", that allow the continuation of the dialog even when the user does not show a standard behaviour.

However, due to the relatively small size of the training data, this could happen too often leading to the unsatisfaction of the user.

A possible improvement in this context would be using a larger data set, which would obviously make the system able to recognize a higher number of user utterances and would then improve the user experience.

Other betterments could regard the dialog management itself: when no restaurants are found with some specific features, the system could be programmed to suggest to the user restaurants that almost match their preference, taking into account that some preferences are more important than others or maybe even asking the user which preference she cares more about.

The decision tree for the dialog acts classification works well enough to allow a fluent dialog flow. Nevertheless, there are some problems linked to the dataset the classifier is trained on, where the identification of the dialog acts occasionally appears debatable.

Namely, the classification of utterances like "thank you good bye" as "thankyou" dialog acts is misleading, since the utterance's dialog act should probably be identified as a "bye". Since the decision tree learned from the data set, the answer to such an utterance is now "you're welcome, do you need something else?", while the dialog should understand the

user’s intention (i.e. end the conversation) and terminate the dialog.

Also, the distinction between “deny” and “negate” is not always clear and is not well exploited in this system, so it is up to further studies to decide whether the dataset can be improved and whether both the dialog acts mentioned above are really necessary for this scenario and, in the case, how to best exploit them.

A possible improvement in the dialog act classification could be made by complementing it with a rule-based classifier, maybe adding a pre-processing phase in which the utterances that can be easily identified by a rule-based system are classified. The remaining utterances would then be classified by the machine learning algorithm and the combination of the two approaches would probably lead to better results in term of accuracy.

For what concern the specific machine learning algorithm, it could possibly be improved by tuning the hyperparameters of the decision tree or by using a random forest.

Also, other algorithms as logistic regression or neural networks could be alternatively used.

However, decision trees need very little data preparation during pre-processing and the data contains many words like “thank” or “bye” that almost uniquely determine the dialog act. Therefore, the nature of the classification problem itself suggested to select such an algorithm, but this does not mean other machine learning algorithms would not be suitable and only by testing different classifiers it would be possible to know for sure which one is the best for this particular context .

6 TEAM MEMBER CONTRIBUTION

We think that this project went well, mostly this is because we communicated very clearly with each other. We also made sure to take the time for every step so that we would not have to backtrack. We also took the non-mandatory deadlines very seriously which really helped us to keep pace. The result of this was that for the final report deadline we only had to create the report and do some bug fixing.

Due to the relatively little dimension of the project (around 1000 lines in python code) and in order to guarantee a coherent and organized implementation, the system was designed, implemented and debugged mainly by two people.

The other members of the group actively participated in the process, both discussing implementation details and writing specific functions necessary for the functioning of the whole system.

They also designed the state diagram, so that everybody knew the details of the dialog policy.

The writing was equally divided among the members and everyone checked every section, to guarantee a higher overall quality and clarity.

Table 2: Memembr contributions

Task	Performer(s)	h
abstract(report)	a.g.barkmeijer	1
introduction(report)	a.g.barkmeijer	5
data(report)	w.r.vries	3
State diagram	w.r.vries, a.g.barkmeijer	2, 2
machine learning(report)	s.j.meyer	2
evaluation(report)	w.r.vries	2
error analysis(report)	r.bassani	1
dialog manager(report)	r.bassani	2
discussion(report)	r.bassani	1
code design, partial implementation and debugging	r.bassani, s.j.meyer	25, 20
lookup function	a.g.barkmeijer	10
give info function	a.g.barkmeijer	2
Evaluation functions	w.r.vries	10
LaTeX style	w.r.vries	13

REFERENCES

- [1] Herbert H. Clark and Edward F. Schaefer. 1989. *Contributing to discourse*. Vol. 13.
- [2] John D. Gould and Clayton Lewis. 1985. *Designing for usability: key principles and what designers think*. Vol. 28.
- [3] Dan Jurafsky and James H. Martin. 2019. *Speech and Language Processing* (3rd ed. draft). To be published.
- [4] Robert C. Stalnaker. 1978. *Assertion*. In *Cole, P. (Ed.)*. Vol. 9.
- [5] Ludwig Wittgenstein. 1953. *Philosophical Investigations*.