Chatbot implementation for restaurant recommendations

GUY EYLON, 2748312
LAMAR KIEL, 6024343
JOEP NOUWENS, 6537618
MARINOS SAVVA, 2076888

ACM Reference Format:
Guy Eylon, Lamar Kiel, Joep Nouwens, and Marinos Savva. 2021. Chatbot implementation for restaurant recommendations. 1, 1 (October 2021), 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Since birth, human beings are confronted with language. From our parents we learn how to understand and speak our first words. Eventually, we get familiar with semantics and word variations. By the time we speak fluently, we are able to alter between alternative word variations, grammar differences, moods and contexts. A set of words can result in an infinite set of sentences. Our model tries to replicate this human interaction through dialog. It does this by responding to the user's input and providing an appropriate answer with the goal of eventually coming up with a recommendation. It does this in the domain of a restaurant system, and will gather the user's preferences to come up with the ideal restaurant according to their wishes. This report covers four sections, namely: the data, machine learning, the dialog manager and the reasoning. Finally, the shortcomings of this model will be discussed, what could be improved now and what should be improved in the future. This introduction provides a quick glance over the upcoming sections.

Firstly, we discuss the data that is used for our model. Utterances are divided by different labels, so we need a good understanding of what these labels consist of. We give examples of the dialogs in our data and come up with a dialog state diagram that is used for our state transition function.

The second aspect is the machine learning algorithm. At first, we developed two baselines: a majority baseline and a rule-based baseline. After that we created a decision tree algorithm and a logistic regression algorithm, and trained both on the previously discussed dataset. We also consider some difficult examples for our algorithm.

We follow up with the dialog manager and explain our state transition function with examples from the dialogs. Furthermore, we elaborate on our reasoning component with additional dialog snippets.

Finally, we will discuss our findings, such as ways we could achieve a better model and what we should pay attention to in the future. Each model has its own limitations, and we encountered these as well. Some recommendations will be made for the improvement of our data preprocessing and algorithm for further research. At the end you will find our group contributions.

Authors' addresses: Guy Eylon2748312, g.eylon@students.uu.nl; Lamar Kiel6024343, l.d.kiel@students.uu.nl; Joep Nouwens6537618, j.w.j.nouwens@students.uu.nl; Marinos Savva2076888, m.savva@students.uu.nl.

Manuscript submitted to ACM

2 DATA

The data that is used for this project consists of the dialogs from the Dialog State Tracking Challenge (DSTC 2). This dataset contains 3235 dialogs focused only on the restaurant domain. The interaction is between a user and the system. The user is given a task to obtain a restaurant recommendation. Through the preferences of the user the system should suggest a restaurant. First it asks which kind of food the user wants (type), then it asks in which part of town the user want to eat (area), at last it wants to know if they want to have cheap, moderate or expensive prices. An example of this is:

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: cheap
speech act: inform(pricerange=cheap)
turn index: 1
system: What part of town do you have in mind?
user: any
speech act: inform(area=dontcare)
turn index: 2
system: What kind of food would you like?
user: mediterranean
speech act: inform(food=mediterranean)
turn index: 3
system: the gardenia serves mediterranean food in the cheap price range

Here you see the system extracting the preferences and suggesting a restaurant. Another option could be that there are no recommendations available for the specified preferences. The system will then say that there are no restaurants matching the preferences. If the system provides a suggestion, then the user can ask for extra information about the restaurant, such as the address, phone number or postal code.

system: loch fyne serves seafood food
user: can i get the address and phone number
speech act: request(addr,phone)
turn index: 4
system: The phone number of loch fyne is 01223 362433 and it is on The Little Rose 37 Trumpington Street.

When the user tells the system a variation of 'thank you', the system closes off with: 'Goodbye, hope to see you again.' As you can see the dialogs are in English and are collected using automatic speech recognition. This can have the consequence that certain words are picked up differently than intended and this could influence the trained model. Our dialog system is able to solve some of these spelling mistakes, by using the Levenshtein edit distance. More details on the Levenshtein distance will appear in the following section. The formality that is used in the dialog is normally short and concrete, and this makes it more difficult to extract context for a specific utterance. In the following part we discuss the dialog state transition diagram

2.1 Dialog State Transition Diagram (DSTD)

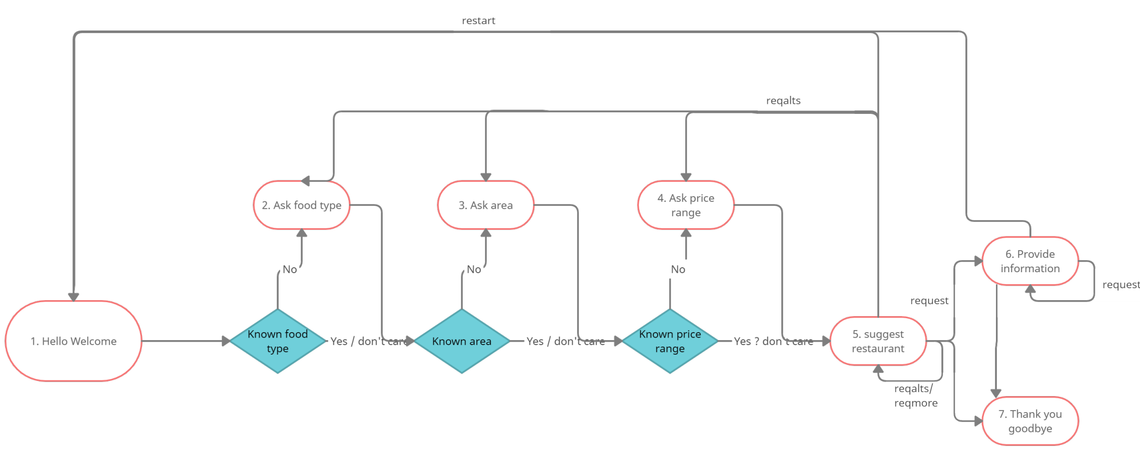
The dialog in the data can be represented in a dialog state transition diagram. Our model can be found as Figure 1. The first state of the model is the `welcome` state; this is how each conversation starts. After that it is time to extract the preferences. It will start with the food type, if it does not know the food type, it will ask what food type the user has in mind. If it already knows the food type because the user said it right away, it will ask for the area of the restaurant, which is the third state. The fourth state is asking the price range. If any of these states are directly stated by the user, it will be remembered by the model and it will ask what it still needs to know. One thing that needs to be mentioned, when the user states that one of the preferences is of no importance, the system should not consider this variable.

When all the preferences are collected we arrive at the fifth state which is the state that recommends a restaurant given the user's answers. After this the user has several different options. The first one is the so called `reqalts` option, after which the system needs to go back to collect preferences and eventually recommend a different restaurant. The other option is the `reqmore` option, when there is more than one option matching the given preferences, the system can suggest an alternative restaurant to the one suggested initially.

If the user is satisfied with the recommendation there are two options for them. They can decide that they have enough information (e.g., they know all the information of this restaurant) and get to state number seven - the `goodbye` state. Here the system will say farewell and wishes that it will see you soon. The system stops after this. Alternatively, the user can go to state six, which is the `give information` state. Here the user can request the restaurant's telephone number, address or postal code. After this, the user can go to the seventh state and say goodbye.

A last option, which is always available, is the `restart` option. From here the whole conversation starts from the beginning. In the next subsection we will look at an example of the DTSD diagram

Fig. 1. dialog state transition diagram



2.2 DSTD example

When we take a look at the dialog example beneath. We see that the user immediately states two of the required preferences. The system then asks for the third, and the user responds that it does not matter for them. The system can now suggest a restaurant. After that the user opts for the action `reqalts` and the system outputs a second option. The user asks for some extra information, namely the address and phone number and afterwards says goodbye. Now the system terminates.

`**Task 10443**: You want to find a cheap restaurant and it should be in the west part of town. Don't go for the first venue the system offers you, ask if there is anything else. You want to know the address and phone number of the venue.`
`turn index: 0`
`**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**: id like a cheap restaurant in the west part of town`
`speech act: inform(pricerange=cheap,type=restaurant,area=west)`
`turn index: 1`
`**system**: There are 2 restaurants in the cheap price range and the west of town . What type of food would you like?`
`**user**: i dont care`
`speech act: inform(=dontcare)`
`turn index: 2`
`**system**: thanh binh is a nice place in the west of town and the prices are cheap **user**: anything else`
`speech act: reqalts()`
`turn index: 3`
`**system**: la margherita is a nice place in the west of town and the prices are cheap`
`**user**: could i have the address and phone number please`
`speech act: request(addr,phone)`
`turn index: 4`
`**system**: The phone number of la margherita is 01223 315232 and it is on 15 Magdalene Street City Centre .`
`**user**: thank you good bye`
`speech act: thankyou()|bye()`

3 MACHINE LEARNING

3.1 Baseline models

For this research we needed to implement two baseline systems. Baselines are mostly used to compare the machine learning algorithms with simple algorithms. The two baselines are:

- Most frequent baseline

- Rule-based baseline

The most frequent baseline always assigns the class that is most common in the data. The data used for this is from the data section as discussed but then only the user utterances, labeled with different classes. This will be elaborated on later in this section. The most common label is the label `inform`, and the algorithm has an accuracy of 0.40, as expected, because it only predicts the inform label for every input. Since 40 percent of the training set came with the label `inform`, this is the correct percentage for the most frequent baseline.

For the rule-based baseline 14 different labels are used. The labels are: ‘thankyou’, ‘request’, ‘null’, ‘reqalts’, ‘confirm’, ‘deny’, ‘affirm’, ‘ack’, ‘negate’, ‘bye’, ‘repeat’, ‘hello’, ‘reqmore’ and ‘restart’. In this list ‘inform’ is not included, because the system will return ‘inform’ as the default label when it cannot match the user’s input with any of the keywords used in the baseline rules. Since the ’inform’ label is the most common as stated earlier on, this gives the highest chance of a correct label prediction in this case of no match. Thus, instead of only recognizing one label as in the case of the other baseline, it can recognize 14 different labels. Hereby, the accuracy should be a lot higher, since we actually use rules to determine which label corresponds with the input. In fact, an accuracy of 0.90 is achieved for the rule-based baseline. This means it is more than twice as accurate as the "most frequent" baseline.

To give an idea of how the rule-based baseline works, we show a couple of examples. If one of the keywords of a set appears in the input, the system automatically assigns the corresponding label to that input. For the label ‘hello’, the following set of keywords is used: “hi”, “hey”, “hello”, “halo”. For ’confirm’, the keywords "is there", "is it", "does it", "is that" and "do they" are used.

3.2 Machine Learning algorithms

In addition to these baselines, two classifiers were added with the intention to get an even better accuracy on the input. The two algorithms that were used are:

- Decision Tree
- Logistic Regression

The first machine learning algorithm is the decision tree algorithm. The algorithm is imported from the Python library `Sklearn`. Decision trees are a type of supervised learning algorithm, which means that we tell the machine what labels the input has and that it learns these values. The algorithm tries to find the best splits for the nodes and the leaves. First, we need to vectorize the sentences to create a bag of words. After this preprocessing of the data and training the model with the method `DecisionTreeClassifier.fit()`, an accuracy of 0.97 was achieved, which is much better than both of our baselines.

The second machine learning algorithm is the logistic regression algorithm. This algorithm was also imported from the `Sklearn` library as well. In the `sklearn.linear_model` package the method `LogisticRegression.fit()` can be found. This was used to fit our model on the training data. The algorithm uses the Sigmoid activation function to assist it in making better predictions. The model output probabilities and this can be used to predict classes. After fitting and training the model an accuracy of 0.98 was achieved, which is similar to the the accuracy of the decision tree model, and again much better than both of our baselines.

As previously discussed, the baseline and machine learning classifiers were evaluated by calculating the accuracy on the test data set. The accuracy of baseline 1 (most frequent baseline) is 0.40, and the rule-based baseline came with an accuracy of 0.90. Both machine learning classifiers scored significantly better, producing an accuracy of 0.97 for the decision tree classifier and 0.98 for the logistic regression classifier. This accuracy was calculated by dividing the correctly classified samples by the total number of samples.

Since none of the classifiers achieved an accuracy of 1.00, we can conclude that the algorithms made some mistakes on the test set. However, this is not a problem since it is almost impossible to get a classifier that makes zero mistakes. For baseline 2 and both classifiers, some input sentences caused some issues. A list of difficulties is described below.

3.2.1 Baseline 2. For baseline 2 a lot of times a person responded “good”. In the set with actual labels, this was seen as a ‘bye’-sentence instead of ‘ack’. This is not specifically a mistake by our baseline, but this keyword came with different classifications. Since utterances could only have 1 label, this made it difficult to predict the corresponding label. In addition, some ‘request’ sentences were seen as ‘confirm’ sentences. Keywords in our ‘confirm’ set that led to this mistake were, for example, ‘do they’ or ‘is it’: “what type of food do they serve” was classified as a ’confirm’ sentence, which was incorrect. This mistake was made because the keyword-combination “do they” appears earlier in the list of rules than “what”. That is why this sentence received the incorrect classification.

3.2.2 Logistic regression classifier. A printed table for the precision and recall for every label showed that the precision percentage was the lowest for the ‘ack’ label, with a score of 0.00. This means that every instance of the ‘ack’ classification was predicted incorrectly by the classifier. However, this label only occurred four times in the whole test set. The second lowest precision score was 0.95 for the ‘null’ label. This can be explained by the fact that some null sentences still contained some of the most common words of other classifications.

Label	count
inform	10160
request	6484
thankyou	3259
reqalts	1747
null	1612
affirm	1156
negate	435
bye	366
confirm	172
hello	93
repeat	33
ack	28
deny	27
restart	14
reqmore	5

Table 1. Labelcounts

3.2.3 Decision Tree classifier. The decision tree showed a significantly higher precision for the ‘ack’ label, with a precision of 0.75. However, the ‘hello’ label had a precision of only 0.26, which is quite strange because these sentences should be among the easiest to recognize. The ‘repeat’ sentences were also only classified correctly in sixty percent of all instances. An explanation for these low percentages compared to others could be the fact that only a few ‘repeat’, ‘ack’ and ‘hello’ sentences appeared in the training set, which made it difficult for the classifier to recognize. All other instances that appeared higher than 100 times had an average precision score of 0.97.

3.3 Error analysis and difficult instances

To sum up all complications of the previous three paragraphs, two different types of utterances were more difficult to classify. The first example is the ’ack’ label, mostly because of the word "good". This caused some wrong classifications for every classifier, but especially for baseline 2. The second example was the ’null’ label together with the ’inform’ label. These were sometimes mixed up, especially in the logistic regression classifier. The decision tree worked well on cases of this last example, but baseline 2 and the logistic regression classifier had some problems with identifying ’null’ or ’inform’ labels.

3.4 Data preprocessing

The data that was used came from a .dat file. This file contained 25501 utterances with a corresponding label, it could be the case that the same utterances have different labels due to just assigning one label. All utterances were lower cased to avoid mistakes in the process and all white spaces from beginning and end were removed. Afterwards, we conducted some exploratory analysis. As discussed earlier there were 15 unique labels, with 'inform' being the most frequent (10159 times). The remaining numbers can be found in table 1. There are altogether 5359 unique sentences. A lot of the sentences are short and specific, and this is why some sentences occur more often (e.g., "a cheap restaurant" is being used 9 times in the set). The average length of the utterances is 3.72 words. Another proof that most of the times very short answers are given by the user.

3.5 Quantitative Evaluation

To measure the accurateness of both classifiers, three different metrics will be used. These three metrics are accuracy, precision and recall. These metrics can be calculated as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

TP = true positive, FP = false positive, TN = true negative, FN = false negative

Logistic Regression Classifier:
$accuracy = 0.98, precision = 0.98, recall = 0.98$

Decision Tree Classifier:
$accuracy = 0.97, precision = 0.98, recall = 0.97$

This shows that the logistic regression classifier works just a little better than the decision tree classifier.

4 DIALOG MANAGER

4.1 State transition function

The state transition function is responsible for giving an adequate response to the user. Ideally it takes the user from "Hello! Welcome to the UU restaurant system. How may I help you?" to "Goodbye! And thank you for using the UU restaurant system" with hopefully a nice restaurant to go to. It functions on the `previous_state`, `next_state`, `predicted_label` and `preferences`. The system start with the welcome message and asks how it may help the user. Then the user articulates one of their preferences. From the chat input the system predicts for instance that `predicted_label == 'inform'`. This calls the function `get_preference` and attempts to pick up on the user's preference for food, area or price if the user articulated any of those. The system does this for each one of the possible preferences and tries to fill these up. After it has collected all of the preferences, it will query the user to check if they have additional requirements, specifically checking for a match with one of the extra restaurant properties that are derived based on a set of inference rules. These rules are discussed in the next section, but in short they function for narrowing down the preferences based on additional requirements. Finally, the user can ask for information about the restaurant's phone number, postal code, address, price, food and area and the system should match this request for information with the recommended restaurant. For example, when the user agrees with the restaurant Saigon City and they ask for a phone number, the system should answer with "The telephone number of Saigon City is 01223 356555".

If the recommendations are fulfilled or there are other `predicted_labels` then the state transition function goes to the other so-called `elif` parts of the function. The first part being if the user denies, negates or asks for other recommendations and they are not there, the system's next state will be `alt_preferences`, the system will say that there are no recommendations available and it prompts the user to alter their preferences. When the predicted label is repeat, the next state will be the same state it is currently at. The obvious state transitions being for goodbye, hello and restart will give a state transition to `goodbye, welcome` and `restart_convo` respectively. When the system is unable to understand what the user actually said, the predicted label will be `null` and it goes to the clarify state, where it tries to check whether the user made a spelling mistake, and it suggests the word it thinks the user intended to write.

Next state	System utterances
Welcome	Hello! Welcome to the UU restaurant system. How may I help you?
Preferences	**food_type**: What kind of food do you have in mind?
	area: In which area do you please to eat?
	prince_range: Do you wish to eat in the cheap, moderate or expensive price range?
Restaurant_suggestion	%s is a fantastic %s restaurant in the %s of town.
Provide_info	**address**: 'The address of %s is %s'
	tel_nr: The telephone number of %s is %s
	postal_code: The postal code of %s is %s
	food: %s serves %s food
	area: %s is in the %s of town
	price: The price range of %s is %s
Restart_convo	Welcome again! How can I help you this time?
Alternative	Would you like to eat at %s in the %s of town, their prices are %s
Alt_preferences	There are no restaurants available with your preferences, would you like to change them?
Goodbye	Goodbye! And thank you for using the UU restaurant system
Clarify	I did not quite understand that, did you mean %s?
Nomatch	Unfortunately we did not find a restaurant matching your description.
Nopref	You must first choose your preferred restaurant.
Additional	Do you have any additional requirements?

Table 2. Utterances per state

4.2 Dialog manager function

The `state_transition` function takes as input the `predicted_label` as determined by our active classifier as well as the user input chat_input in order to decide which state we should transition to. This function call is closely followed by our `dialog_manager` function, which in a given state will handle all input/output operations for out chatbot. The dialog manager is responsible for finding restaurants and printing the utterances that can be found in table 2, thus providing restaurant suggestion and information about those restaurants. In figure 2 you can see an example dialog utilizing the state transition function as seen in the diagram in figure 1. In the next part we discuss the reasoning part of the dialog system.

```
>hi
input: hi -> ['hello']
state: None -> welcome
Hello! Welcome to the UU restaurant system. How may I help you?
preference: {'restaurantname': None, 'food': None, 'area': None, 'pricerange': None, 'additional': None}
>I am looking for an indian restaurant
input: i am looking for an indian restaurant -> ['inform']
state: welcome -> request_info
In which area do you please to eat?
preference: {'restaurantname': None, 'food': 'indian', 'area': None, 'pricerange': None, 'additional': None}
>In the centre
input: in the centre -> ['inform']
state: request_info -> request_info
Do you wish to eat in the cheap, moderate of expensive price range?
preference: {'restaurantname': None, 'food': 'indian', 'area': 'centre', 'pricerange': None, 'additional': None}
>cheap
input: cheap -> ['inform']
state: request_info -> request_additional
Do you have any additional requirements?
preference: {'restaurantname': None, 'food': 'indian', 'area': 'centre', 'pricerange': 'cheap', 'additional': None}
>Not busy
input: not busy -> ['inform']
state: request_additional -> restaurant_suggestion
the gandhi is a fantastic indian restaurant in the centre of town.
preference: {'restaurantname': 'the gandhi', 'food': 'indian', 'area': 'centre', 'pricerange': 'cheap', 'additional': {'property': 'busy', 'value': False}}
>and how about an italian one?
input: and how about an italian one? -> ['reqalts']
state: restaurant_suggestion -> restaurant_suggestion
pizza hut city centre is a fantastic italian restaurant in the centre of town.
preference: {'restaurantname': 'pizza hut city centre', 'food': 'italian', 'area': 'centre', 'pricerange': 'cheap', 'additional': {'property': 'busy', 'value': False}}
>
```

Fig. 2. Example dialog

5 REASONING

The reasoning component of the program is meant to narrow the recommendations down to a smaller, more relevant, set of recommendations, based on extra requirements obtained from the user. As we discussed earlier, the system first collects the user's preferences based on food, price and area. After these preferences have been filled, the system asks whether there are any additional requirements. The extra restaurant properties recognized by the system for the filtering purpose are crowdedness (busy/not busy), length of stay (short/long stays), romantic atmosphere and suitability for children. Say for instance, you want a restaurant that is romantic, the system may infer that from spending a long time in a restaurant (perhaps with your significant other) it is romantic, and therefore recommend a restaurant where people normally spend a long time. .The system gets these additional restaurant from the same restaurant .csv file that was used for the original recommendations. What is changed is that different values were randomly assigned to the .csv file for several categories used as antecedents in the inference rules' reasoning logic. These properties are inferred on the basis of a set of rules as listed in table 3. The inference logic is applied for each restaurant and added to its in-memory data object when the system boots up, in order to save processing time during the actual conversation. In the following example you see the user looks for a British, moderately priced restaurant in the west. Following that, the system asks for any other preferences, after you tell it not suitable for children, it will set the property of children to `False` and infers that you need to go to the saint johns chop house.

```
>moderate input: moderate -> ['inform']
state: request_info -> request_additional
Do you have any additional requirements?
preference: 'restaurantname': None, 'food': 'british', 'area': 'west', 'pricerange': 'moderate',
'additional': None
>not suitable children
```

ID	Antecedent	Consequent	T/F	Description
1	pricerange: cheap	busy	True	a cheap restaurant with good food is busy
2	food: Spanish	long stay	True	Spanish restaurants serve extensive dinners that take a long time to finish
3	Busy: True	long stay	True	you spend a long time in a busy restaurant (waiting for food)
4	long stay: True	children	False	spending a long time is not advised when taking children
5	long stay: True	romantic	True	spending a long time in a restaurant is romantic

Table 3. Inference Rules

```
input: not suitable children -> ['inform']
state: request_additional -> restaurant_suggestion
saint johns chop house is a fantastic british restaurant in the west of town. it is not suitable
for children because spending a long time is not advised when taking children.
preference: 'restaurantname': 'saint johns chop house', 'food': 'british', 'area': 'west', 'pricerange':
'moderate',  'additional': 'property': 'children', 'value': False
```

In the next example the user wants to go to a romantic restaurant. The preferences are set, and after you tell it to search for a romantic restaurant it infers that you will need to go to the saint johns chop house. This shows that a restaurant can have multiple characteristics and then the same restaurant can be suggested for different inferred properties.

```
>west
input: west -> ['inform']
state: request_info -> request_additional
Do you have any additional requirements?
preference: 'restaurantname': None, 'food': 'british', 'area': 'west', 'pricerange': 'moderate',
'additional': None
>romantic
input: romantic -> ['inform']
state: request_additional -> restaurant_suggestion
saint johns chop house is a fantastic british restaurant in the west of town. it is romantic
because spending a long time in a restaurant is romantic.
preference: 'restaurantname': 'saint johns chop house', 'food': 'british', 'area': 'west', 'pricerange':
'moderate',
'additional': 'property': 'romantic', 'value': True
```

Apart from the reasoning implementation is it also possible to switch some configurabilty options. The chatbot will recognize inputs starting with "/" as commands and not as input text. These commands are dissected into two parts, one for the command and one for the command value. In some cases a command value is not needed. A list of commands can be seen in table 4.

Command	Executes
/e	Exit program
/h	Print a list of commands
/t [b1][b2][c1][c2]	Test the performance of a classifier
/s [b1][b2][c1][c2]	Switch the classifier being used for label prediction
/d [on|off]	Enable/disable debug mode, showing more/less descriptive text
/l [value]	Changes the maximum Levenshtein distance to [value]
/delay [value]	Adds a delay to the system response of [value] seconds
/o [on|off]	outputs system messages in all caps when enabled

Table 4. configuration options

6 DISCUSSION

The discussion will be divided into three different elements. The first element discusses the text classification, then the dialog management and finally the reasoning and configurability.

6.1 Text classification

The text classification was done using two different baselines and two classifications. Baseline 1 worked exactly how it should work, coming with an accuracy score of 0.40. Baseline 2 could have been improved a little, by looking even more at the incorrect classifications or adding more rules to get even more correct classifications. Since these rules have to be created manually, we could add as many keywords as we want. However, an accuracy of 0.90 is quite high for a rule-based baseline system, and we focused more on the classifiers.

The logistic regression classifier achieved an accuracy of 0.98 on average, which is quite high. Since this classifier was implemented using the `Sklearn` package and it is a relatively straightforward model, not much could be improved with the code responsible for training the model. However, there may be room for improvement with the pre-processing done on the dataset (for instance, the smart removal of stop words). The same holds for the decision tree, which could also be implemented from the `Sklearn` package. We were satisfied with the accuracy of 0.97, which meant that only three percent of all cases was classified incorrectly. We attempted to improve this percentage, but we did not achieve even higher results.

A technical optimization which could be added to the classifiers module, is allowing the system to load a previously trained model, instead of running the training code during its start-up. However, since it takes mere seconds to perform the training operation, we have not yet implemented this feature.

6.2 Dialog management

The dialog manager works well within an expected interaction space, however, when diverting from the expected interactions we can observe some unplanned behaviour. Since the state transition function and dialog manager function closely together, more options and transitions would need to be introduced to adapt the system to be more flexible in its transitions. Furthermore, correction of spelling mistakes will only occur for a few specific pattern matches, and more patterns can be added to recognize a variety of user input.

6.3 Reasoning and configurability

The system is able to correctly infer the extra properties mentioned in the inference table. It is also designed to cope with extra rules being added to the table, including a situation where an inference made from a rule towards the end of the table will result in one of the previous rules becoming true, and then inferring based on that early rule as well. Current limitations of the reasoning module is the lack of a proper error message when a wrong input value is entered for the additional requirements prompt, as well as only searching for exact keyword matches for one of the possible properties, or the word "not" (for negation). It is possible to improve on this, by using similar a pattern matching logic with Levenshtein distance, similar to the one employed in extracting the basic user preferences (food type, etc...), as well as smarter matching of negative cases (for example recognizing 'short stay' as matching 'not long stay') Additionally, the dialog system currently remembers the "additional property" required by the user even after they switch other preferences (for example for a different area), and this can only be reset with a "restart" label.

7 TEAM CONTRIBUTION

The contributions of our team can be found in table 5 in the appendix. It is divided in tasks per week, how many hours were spent and who spent those hours. For this project we were divided in groups with different skill sets. Some of us took lead in programming, others took lead in writing the report. Although, in hours you can probably see who had done more on, we all learned some new skills through this project. We met a couple of times on teams to discuss what our plan was and to discuss how to implement everything. Apart from the weekly meetings on campus we discussed all the assignments on chat. Sometimes we were a little late and missed the review parts, but we worked hard in the end to make this a successful project.

8 APPENDIX

Task	Team member	Time (hours)
Data preprocessing	Joep, Guy, Marinos	3
Classifiers	All	6
Error analysis	Joep	4
Levenshtein	Guy	2
Code testing, examples	All	3
Debugging	Marinos, Guy	5
Preference Extraction	Guy, Marinos	4
Restaurant lookup function	Guy	1
Inference Rules	Guy	5
Configurability	Marinos	2
Weekly meetups	All	8
State transition	Lamar, Marinos, Guy	6
Utterances	Lamar, Marinos, Guy	2
Diagram	Lamar	1.5
Data analysis	Lamar, Joep	3
Report		
Abstract	Lamar	0.5
Introduction	Lamar	1
Data	Lamar, Joep	3
Machine learning	Lamar, Joep	4
Dialog manager	Lamar, Marinos	6
Reasoning	Lamar, Guy	3
Discussion	All	3
Revision	All	9

Table 5. contributions