

Implementing a dialog system for restaurant recommendation - Group 21

SACHA VUCINEC, MAUD OVERBEEK, ALEXO CASTRO YÁÑEZ, YFKE SMIT, DOUWE BRINK

Abstract

In this report a dialogue system was implemented for restaurant recommendation. The system is based on the dialogs from Microsoft's second Dialog State Tracking Challenge. From the dialogs, a state transition diagram was modelled. Our implementation follows this model. Machine learning was used for dialog act classification for user utterances. A decision tree and logistic regression model were trained. Following our evaluation and error analysis of the models, we concluded that the decision tree was most suitable for implementation in the final model. The resulting dialog system works for retrieving preferences about area, type of food served and price range of the restaurant. A reasoning component was added to account preferences about whether the restaurant should be touristic, has assigned seats, is suitable for children or is romantic. For the scope of this paper, the implementation was successful. However, we conclude the system has its limitations regarding flexibility. Additionally, a different machine learning architecture could be used for better dialog act classification.

1 INTRODUCTION

Using automated dialogue systems is becoming increasingly popular for communicating efficiently with users in many different domains such as customer service, planning services and online chatting. Questions that users have can be easily answered, without the need for extra human resources. With help of state-of-the-art AI techniques these dialogue systems become more and more advanced. In this report such a dialogue system will be implemented for restaurant recommendation. The implementation will be done in Python 3.x. The system makes use of machine learning model for dialog act classification in combination with a finite state model (FSM) to navigate the interaction. The classifier assigns an intention ("the dialog act") to a user utterance, which is used to give a fitting response to the user and move the system to a (new) state. For this dialog act classification, two machine learning models will be trained and evaluated. The best performing model will be used in the final dialog system. The end goal is to implement a fully working dialog system, that recommends a user one or more restaurants based on expressed preferences.

In Section 2 the data set used for modelling our system and training the machine learning algorithm for dialog act classification is outlined and explored. Section 3 goes over the implementation and design choices of the machine learning model. In Section 4 the dialog system is described. The used state transition function is explained and system utterance templates are given. Section 5 gives an insight into the reasoning component of the dialogue system and, additionally, contains a discussion on handling contradictions within the system. Then, Section 6 presents a list of configurable options. In Section 7 a conclusion is given. Here, the approach for both the dialog system in general and the dialog act classification is discussed. In this discussion limitations, alternative approaches and possible improvements referred to. And lastly, Section 8 contains an overview of the contributions of each of our team members.

2 DATA

The data used in this project consists of the dialogs from the second Dialog State Tracking Challenge (DSTC 2, see <https://www.microsoft.com/en-us/research/event/dialog-state-tracking-challenge/>). It consists of 3235 dialogs in the restaurant domain. In each dialog a user interacts with a system. The user is given a task that specifies a certain set of preferences, with the goal of getting a restaurant recommendation. One thing that should be noted is that the dialogs are collected with automatic speech recognition. This has an influence on the quality of the transcript, as well as the

formality of the user utterances. This might influence system performance, since the data we train on and model after is not always correct. Each dialog in our data set is annotated with additional information. Every dialog has its own unique session-id and a numbered task. The interaction is divided up in turns, which consists of the a system utterance and the following user utterance. The turns within a dialog are indexed. Each user utterance is annotated with a dialog act and information extracted from said utterance.

Figure 3 shows a transition diagram that is modelled after the dialogs in data set. The following two dialog snippets will illustrate how the system states are modelled in the diagram.

Fig. 1. Example snippet from the dialog data set (session id: voip-88f198881b-20130326_032851)

```

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: i need a restaurant in the north
speech act: inform(type=restaurant,area=north)
system: What kind of food would you like?
user: part of town that serves chinese food
speech act: inform(food=chinese)
system: Would you like something in the cheap , moderate , or expensive price range?
user: um
speech act: null()
system: the hotpot is a nice place in the north of town serving tasty chinese food

```

In figure 1, the first system utterance relates to state 1, this is the standard welcome message. The user replies with a preference for an area. If we look in the diagram, we follow the arrow down to "food type expressed?". Since the user did not specify a food type yet, the system moves to state 3 and asks for a preference in food type. The user replies with a food type. In case that there would be one restaurant left to recommend, the system would have given that recommendation. Since this is not the case, the system moves to state 4 and asks for a price range. Now, one should note that state 5 in the diagram is a feature added in our implementation and is not present in the data set dialogs. So in the current dialog, the system state moves to 7 and the system replies with a recommendation. The snippet in Figure 2 is taken from the middle of a dialog. The first system state is 2: asking for an area. The user replies with area and food type. Now we follow the "yes" arrow down from "Food type expressed?" and end up on a choice point. Since there is only one data base option, the system moves all the way to state 7 and immediately and recommends the restaurant.

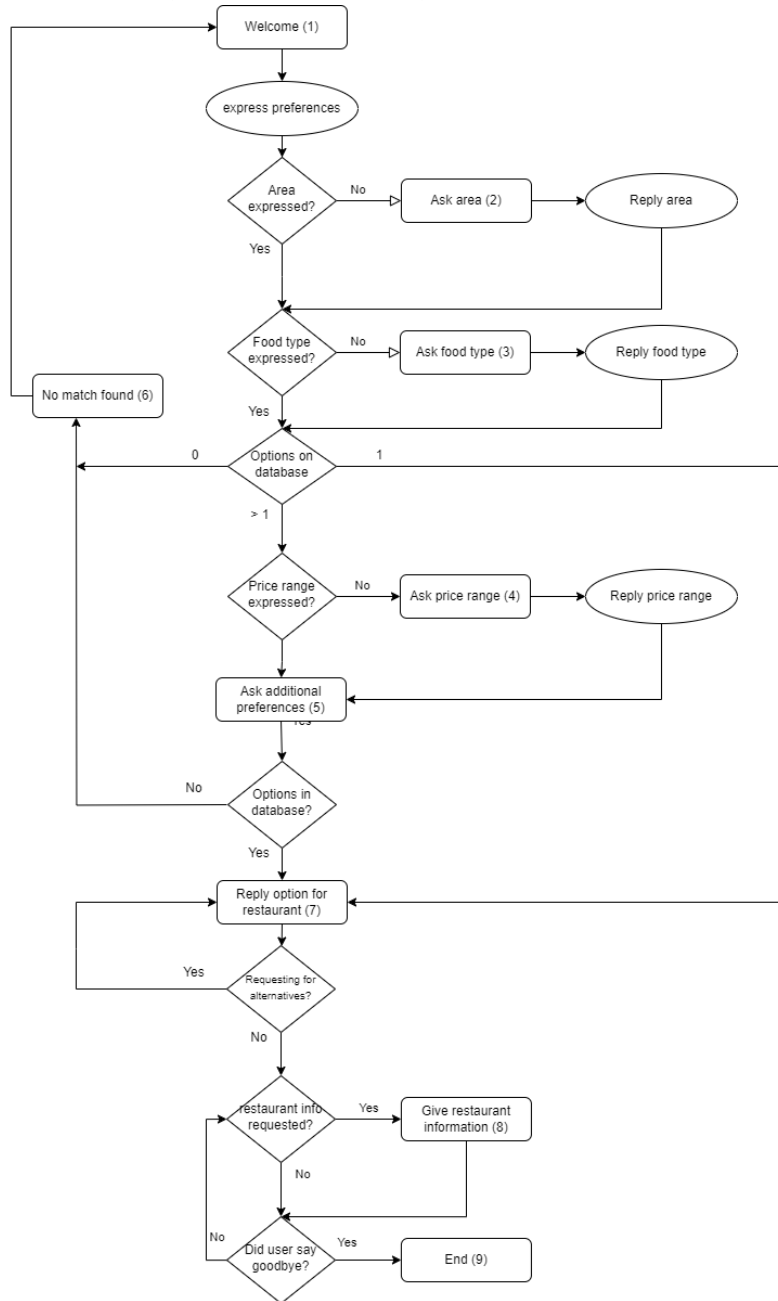
Fig. 2. Example snippet from the dialog data set. (session id: voip-fdf8b50918-20130327_024718)

```

system: What part of town do you have in mind?
user: im looking for a cheap restaurant in the east part of town
speech act: inform(area=east,pricerange=cheap,type=restaurant,task=find)
system: the missing sock is a nice place in the east of town and the prices are cheap

```

Fig. 3. The state transition diagram based on the data set



3 TEXT CLASSIFICATION

We implement some models to classify the user utterances given in the dialog system. That way, the system knows how to handle the input from the user. There are four models developed that can be used in the dialog system. Two baseline models, and two machine learning models.

3.1 Baseline models

One baseline model is trained to return the utterance class that exists the most in the training set. The second baseline model is a rule-based system based on keyword matching. To develop this, keywords were added that exist in the example data at specific utterances, like 'no' for the *negate* class. These keywords were derived after data exploration, like calculating the most common words per class. A ranking is used to make sure some words have higher priority when classifying an utterance. Preprocessing of the data for the baseline models is done by making sure all text is lowercase and splitted into tags and utterances.

3.2 Machine learning models

There are two machine models used: A decision tree, and a logistic regression model.

The logistic regression model is defined by two key hyperparameters: 'C', the inverse of regularization strength (default value of 1), and 'penalty,' set to 'l2' for Euclidean distance regularization. In contrast, the decision tree model comprises four default hyperparameters: 'max depth' (50), 'min_samples_split' (2), 'max_leaf_nodes' (None), and 'ccp_alpha' (0).

Following data preprocessing, both models are trained on the training dataset and subsequently evaluated using the test dataset to assess their predictive performance. This systematic approach ensures the reliability of the models in practical applications.

3.2.1 Preprocessing. Preprocessing is done in order to make sure the input data is clean enough for the model to use. The preprocessing consists of the following items:

Vectorization: The text is converted to a numerical format, using the Bag-of-Words technique.

TF-IDF: The words are weighted by rarity, rare words, are more important.

`min_word_length` (*optional*): Excludes words from and below a certain length, so for example, words of length 1 are not included anymore.

Removing stopwords (*optional*): A list of words can be given that are also excluded from the text.

Stemming (*optional*): Words are reduced to their root or base form.

3.3 Statistics of the data

Without deduplication, the label distribution is not uniformly divided between classes, as seen in figure 4. The mean utterance length per class is also not uniform between classes, as for some classes, like *hello* or *confirm*, the mean utterance length is higher than for classes like *affirm* or *null*, as seen in figure 5. The mean utterance length over all classes is 3.7. So, the data is not divided evenly over all classes in the training data. Stratifying the data is done while splitting the data in training and test data, to make sure all classes are represented in every phase.

Fig. 4. The class label distribution of the non-deduplicated training data

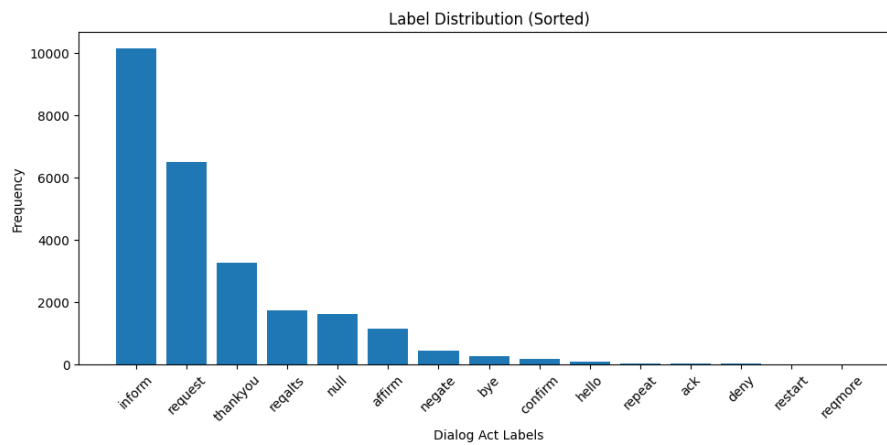
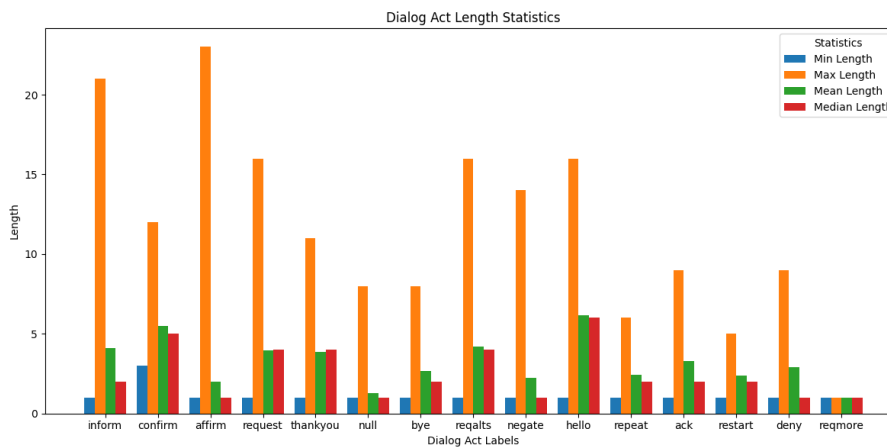


Fig. 5. Statistics on utterance lengths, per utterance label.



3.4 Quantitative evaluation and model selection

We have applied what is probably the most common technique for model evaluation and selection, the k -fold cross validation. We can split it in several steps in order to obtain our possible "best" estimator.

Step 1 Start by splitting the data into a training and test set, as a classic two holdout method. The test set is kept untouched until we perform the final evaluation with our "best" estimator candidates. To solve imbalanced nature of data we perform stratification according to tags, so we make sure that there is at least one sample per class in both in the training and the test set. When dealing with deduplicated data, we also delete classes with only one utterance (reqmore) to keep our stratification.

Step 2 The next step is the hyper-parameter tuning in the training set, in which the cross validation plays its role. Instead of dividing training set into two (three holdout validation), we split it into k disjoint folds and we train k

different models taking each individual training set as the union of $k - 1$ folds and the validation set as the remaining one. Hyper-parameter selection methods for tuning are discussed in Section. The splitting technique will be the stratified k fold due to the same reasons why we stratify the training and test data.

Step 3 We need to select now the hyper-parameter combination with the best score in the k -cross validation procedure.

Step 4 Finally, we can take our selected model from Step 3 and evaluate it in the separated test set of Step 1.

But before selecting models we need to consider which metrics we are going to take into account when evaluating the models.

3.4.1 Evaluation metrics. As we are facing a multiclass classification problem our main score will be the multi class accuracy, which tries to estimate the probability for a sample of being predicted in its right class by the classifier. Despite the binary nature of precision and recall we rather look to average precision within classes than to recall precision. This is because the dialog system will prefer that most utterances classified a certain dialog tag are indeed in that class rather than most of samples in a class being well classified, as once the user gives the utterance to the system it needs to give the right answer following the transition diagram (this will be successful if the utterance is correctly classified). The confusion matrix is probably the best way to see the evaluation of a multiclass model as it allows us to have a general overview of prediction and its interaction within classes and it will be used for error analysis, but it is difficult to store and make simple comparisons due to its array nature.

3.4.2 Hyper-parameter tuning. For the hyper-parameter tuning we have decided to implement a manual adaptive method using the two most common non-iterative parameter space selection procedures: the grid search and the randomized search. This manual iteration will start with a randomized search of hyper-parameters from which we would like to bound some of the parameter ranges or even stop considering some of the possible values of some parameters as result of being "always" worse than others, e.g. having or not LR regularization or tree pruning. After this iteration we will perform a grid search within interesting selected spaces of hyper-parameters to some of the "best" model candidates to be evaluated and selected.

A first general random search suggests us that some of the parameters might not be necessary. So we statistically tested the difference mean accuracy obtained within a random search of two different groups stratified by the performance or not of some model simplification algorithms. For logistic regression will be the regularization and for the tree classifier the cost complexity pruning.

For tree tuning we have significant results to affirm that models with cost complexity pruning outperform models without it as we obtain a p-value for comparison test (Wilcoxon) of order 10^{-34} for 1000 runs without data deduplication, and with deduplication it is even more clear. The non convergence of logistic regression makes difficult to take decisions with respect to regularization.

3.4.3 Selected models. Now we present the evaluation metric for the best models we have found:

Deduplicated	Experiment	Penalty	C	solver	Test accuracy	Test precision
No	-	None	inf	sag	0.962	0.677
Yes	-	None	inf	sag	0.887	0.67
No	Grid Search	L2	100	saga	0.964	0.731
Yes	Grid Search (only L1)	L1	18.8739	saga	0.877	0.555
No	Manual Search	L1	0.01	saga	0.89	0.444
yes	Manual Search	L2	0.01	saga	0.675	0.116

Table 1. Evaluation results for logistic regression models.

Deduplicated	Experiment	Max depth	Min samples split	Max leaf nodes	Pruning α	Test accuracy	Test precision
No	Random Search	68	4	98	2.11234e-05	0.973	0.908
Yes	Random Search	85	8	99	6.18515e-05	0.973	0.912
No	Grid Search	None	2	None	1e-5	0.98	0.937
Yes	Grid Search	None	2	None	1e-5	0.889	0.639
No	Grid Search	None	2	None	1e-5	0.98	0.937
No	Manual search	200	4	None	0	0.982	0.885
Yes	Manual search	200	4	None	0	0.877	0.678

Table 2. Evaluation results for decision tree models.

3.5 Error analysis

The script `error_analysis.py` prints the incorrectly predicted utterances per model, along with their (wrongly) predicted and correct tag. Please note that the examples given below may not always occur when running the analysis script, since each model will be slightly different based on the train and test set.

In the table below are a few example user utterances that have been wrongly classified, along with possible explanations. These utterances are representative of utterances where a wrong prediction is often made. These reasons are roughly the same across the two models, hence they are not analysed separately.

Utterance	True tag	Predicted tag	Possible reason
"care", "free"	null	inform	Bias towards common tags
"uh thank you and good bye"	bye	thankyou	Interchangeable
"next restaurant"	inform	reqalts	Possible mistake in the dataset
"is there a restaurant in the south part of town that serves indian food"	reqalts	inform	Context
"i dont want turkish"	deny	inform	Bias towards common tags
"no there is no such venue how about portuguese type of food"	reqalts	negate	Too complex

Table 3. Examples of wrongly classified utterances.

The reason that occurs most frequently is the bias towards common tags. As shown in Figure 4), the inform tag takes up a relatively large part of the dataset, which will have an effect on how the models are trained. The model will have a bias towards common tags, which makes the model often classify an utterance as ‘inform’ or ‘request’ when its classification is not obvious. This can be seen as both a good or a bad thing. On the one hand, inform utterances will occur most often, so it could be seen as reasonable that the system makes a guess based on that. On the other hand, tags that occur less often in the dataset will only be recognized when the model is really confident.

Another reason that occurs often is ‘context’. The model only looks at the given sentence and it does nothing with the message history. In some cases, important information is left out and because of this, the model makes a wrong prediction.

There are also user utterances that are too complex to be classified by a bag of words implementation. For example the sentence *no there is no such venue how about portuguese type of food* first acknowledges what the system stated in a previous utterance. This is a negative acknowledgement, which makes it extra difficult for the model to recognize. Following that, the user asks a question about alternative restaurants. To classify this correctly, the model would have had to really ‘understand’ the semantics of the sentence, which are lost in the bag of words representation.

We won’t discuss the cases where there is a possible mistake in the dataset or where the true and wrong tags are interchangeable.

4 DIALOG SYSTEM

The conversation flow of the dialog system is managed by a state transition function. The state transition function determines how the system moves from one state to another, given the reply of the user. In our implementation, the function takes the previous system state and the user_utterance as input, along side the recommendation object and the user state. The user state is determined by the classification of the user utterance by a trained decision tree model (see Section 3). The state transition function returns a new system state and a system utterance. The system utterance is the reply that the system gives to the user. It’s dependent on information that the user has given in the dialog and the current state of the system. Figure ??? shows some example templates for the system utterances.

4.1 State transition function

The transition function is modelled after the dialog state diagram (see Figure 3) and is part of the dialogManager class in the *dialog_system.py* file. In state 1, a welcome message is given and the user is asked for their preferences about area, food type and price range. After the user reply, The system saves all given user preferences, if any. From then on the system moves from ask area state (2), to the ask food range state (3), to the ask price range state (4), skipping information if it was already provided by the user. After this stage, the user preferences should be known to the system. At this point, the system will move to state 5. Here the system will ask for any additional requirements (see Section 5 for example dialog snippets). The system will start looking for restaurants that satisfy the user preferences. This is done with a simple database lookup. If there are no matches, the user is asked if they want to restart the recommendation system (state 6). If there are matches, the system will present the recommendation (state 7). From this point on a recommendation.retrieved in the Recommendation object is set to true. If the user requests more alternatives, the system state is navigated back to state 7. If the user asks for information about the restaurant, the system will be in state 8. The system replies with the requested information about the restaurant (phone number, adress, postcode or food type). This is done via a simple keyword matching and database look up. This second part of the transition function (after a recommendation is retrieved) handles the navigation back to state 7 in the case that a user requests more alternatives

and it handles retrieving and providing information about the recommended restaurant(s) (state 8). If at any point the user says bye, the system will move to state 9 and the program will exit.

On a few points our implementation deviates slightly from the transition diagram. First off, in the diagram, the system can only end up in state 9 from state 8 or 7. In our implementation state 9 can be accessed from any state, if the user utterance is classified as 'bye'. What's more, in the implementation, an exception state is added. This state catches any state transitions that do not satisfy moving to any other state. In this state, the user is asked if they want to start another dialog. If the user answers no, the system moves to state 9. In any other case, it moves to state 1.

system state	system utterance
1 (init)	Welcome to the Cambridge restaurant system! You can ask for restaurants by area , price range or food type . How may I help you?
2 (askarea)	What part of town do you have in mind?
3 (askfoodtype)	What kind of food would you like?
4 (askpricerange)	Would you like something in the cheap , moderate , or expensive price range?
5 (askrequirements)	Do you have any additional requirements?
6 (nomatch)	There is no restaurant in the city with the specified preferences. Restarting...
7 (recommending)	[Restaurant Name] is a great restaurant serving [Food Type] in the [Area] part of the city at [Price Range] price.[Reason Why It Satisfies Additional Requirement]
8 (providinginformation)	The [Restaurant Information Asked] of the [Restaurant Name] is [Restaurant Information]
9 (ending)	Exiting the application. Goodbye!
Exception State	Do you want to search again? If no, program will exit

Table 4. System states with their corresponding utterance template

5 REASONING

As mentioned in section 4.1, the user is asked for additional requirements. Here the user can respond with requirements such as romantic, touristic, assigned seats or children friendly. If there are additional requirements the add additional properties function is called. Here a new restaurant info file is created. This file is an extension where three columns have been added 'food quality', 'crowdedness' and 'length of stay'. These columns have been randomly filled with the following values. Food quality: poor, average, good or excellent. Crowdedness: empty, quiet, moderate or busy. Length of stay: short, average or long,

5.1 Inferences

Then the inference rules are applied to the new properties of the restaurants and populate new columns in the restaurant properties, "touristic" default: *False*, "assigned seats" default: *False*, "children" default: *True* and "romantic" default: *False*. Then these values are updated based on the result of the following inferences.

Table 5. Inferences used for handling additional requirements

(1) $\text{cheap} \wedge \text{good food} \implies \text{touristic},$	(2) $\text{romanian} \implies \neg \text{touristic},$
(3) $\text{busy} \implies \text{assigned seats},$	(4) $\text{long stay} \implies \neg \text{children},$
(5) $\text{long stay} \implies \text{romantic},$	(6) $\text{busy} \implies \neg \text{romantic},$

After these operations each restaurant will have the appropriate values for "touristic", "assigned seats", "children" and "romantic". Now restaurants can be filtered based on these properties. See figure 6 for an example dialog interaction with additional requirements.

Fig. 6. Example interaction with the dialog system where additional requirements are asked.

System: Welcome to the Cambridge restaurant system! You can ask for restaurants by area , price range or food type . How may I help you?
 Hello
 System: What part of town do you have in mind?
 any
 System: What kind of food would you like?
 any
 System: Would you like something in the cheap , moderate , or expensive price range?
 any
 System: Do you have any additional requirements?
 romantic
 System: the gardenia is a great restaurant serving romanian in the west part of the city at moderate price.
 The restaurant is romantic because it allows you to stay for a long time

In this dialog example the user is recommended the gardenia restaurant when asking for a romantic restaurant. This restaurant is indeed romantic, as the length of stay is 'long' (see the properties below). Thus satisfying the conditional $\text{long stay} \implies \text{romantic}$.

3. the gardenia, moderate, west, romanian, 01223 356354, 2 rose crescent city centre, c.b 2, average, quiet, long

5.2 Contradictions

This system handles contradictions by following the order of the inferences as found in Table 5. For example, when a restaurant is both busy and long stay, then the last inference is applied and the restaurant is classified as not romantic. See figure 7 for an example dialog. This dialog should return the following restaurant if it was classified as romantic: 6, sitar tandoori, expensive, east, steakhouse, 01223 249955, 43 high street cherry hinton cherry hinton, c.b 1, excellent, busy, long.

But because $\text{busy} \implies \neg \text{romantic}$ is the last inference, this restaurant is instead classified as not romantic. The same applies for other possible contradictions. For example, a restaurant which has good food, is cheap and serves romanian food will be classified as not touristic.

Fig. 7. Example interaction with contradicting preferences and requirements.

System: What part of town do you have in mind?
 east
 System: There are 13 restaurants in the east of the town. What kind of food would you like?
 steakhouse
 System: Do you have any additional requirements?
 romantic
 System: There is no restaurant in the city with the specified preferences. Restarting...

6 CONFIGURABILITY

The users can select their preferences for the dialogue system by applying changes to the "config.json" file. Here the users can set their preferences for the following options by changing the value of the respective preferences:

all_caps This boolean parameter determines if the system's output should be in uppercase. The `give_utterance` function checks its status and prints accordingly.

text_to_speech Another boolean parameter, this one decides whether the system's output is spoken aloud (True) or remains as text (False). The `give_utterance` function uses the `pyttsx3` library when it's set to True.

levenshtein_distance This integer parameter sets the maximum allowed character changes when finding similar words. It's used in the `find_closest_levenshtein` function as the maximum distance. The keyword matching function also makes use of the Levenshtein distance. A more detailed explanation of this is given in section 7.1.1.

allow_restarts a boolean parameter that controls whether users can restart the dialog. If "restart" is detected as tag, it checks `allow_restarts` to reset preferences or continue the current state.

model Users can select a prediction model for the tags, such as "decisionTree" or "logisticRegression." The code initializes the chosen model during the `DialogSystem` class setup.

7 CONCLUSION

In this paper an implementation of a dialog system for recommendations in the restaurant domain is discussed. The implementation was modelled after a state transition diagram. Machine learning was used for dialog act classification for user utterances. With the combination of the two a working dialog system is presented.

7.1 General

7.1.1 Limitations and possible improvements. We believe some features of the dialog system are not satisfactory. The implementation of the Levenshtein distance is one of them. Currently, it is only really used in the 'kind of food' question in the pattern matching function and when recognizing additional requirements. It is used to a lesser degree by the keyword matching function, here it is implemented as regex fuzzy search. The problem we encountered was that some words are very similar to others. For example 'Thai' - 'that', 'south' - 'north' or 'help' - 'cheap' would all be interchangeable with a search distance of two, which was recommended by the teaching assistant. Perhaps this could be improved by using a list of all English words, and only if a word is not in this list, Levenshtein distance is used.

Something else that could be improved in the dialog system is keeping track of the user's additional requirements (eg romantic, touristic). If the user specifies a requirement at the start of the conversation, and later specifies another requirement, the dialog system only saves the last requirement.

Additionally, the system could benefit from a better model. User utterance tags are not equally represented in the dataset. Tags like `reqalts` and `reqmore` that do not occur much are therefore often not recognised by the model. A larger dataset would probably benefit the model. An entirely different model could also be considered: a neural network model would be interesting, for example.

7.1.2 Alternative Approaches. In recent studies, good results on task-oriented conversational systems have been achieved by using more general architectures. These architectures make use of pre-trained language models and task-specific information. For restaurant recommendation this approach seems feasible.

7.2 Dialog act classification

For the dialog act classification, two machine learning algorithms were trained and tested. The two models we chose were a decision tree model and a logistic regression model. We chose these two models, since a logistic regression model handles linear data well and decision trees handle non-linear data well. In our evaluation, the decision tree model performed best in terms of accuracy. Other approaches for handling the dialog act classifications, could be using a random forest classifier or a feed forward neural net.

8 TEAM MEMBER CONTRIBUTIONS

All of our team members contributed their fair share. As a rule everybody attended the labs to the best of their ability. In these labs time was set aside to discuss the work that had been done by each member and what work was expected for the next time. Different task, like writing and programming were not divided entirely equally. Rather, we looked at strengths and preferences of each team member.

Table 6. Time spent on each part per person in hours

Task or meeting	Douwe	Yfke	Maud	Alexo	Sacha
Data: loading and preprocessing				1	
Data: analysis and statistics functions		2		1	
Baselines	3	1		2	
Classifiers: Logistic regression				1.5	3
Classifiers: Tree		1	1	3	
Training experiment			1	3	
Evaluation metrics			2	1	
Grid tuning experiment				3	
Random tuning experiment				6	
Dialog system: Transition diagram				1	
Dialog system: Transition function			2	5	
Recommendation: Database lookup function		2		1.5	
Recommendation: System utterances templates				0.75	
Keyword food,area,price	4			0.5	2.5
Configurability: all caps and text to speech	2			0.25	
Configurability: levenshtein distance	1.5				
Configurability: allow restarts					1
Configurability: model specification				0.5	1
Reasoning implementation		7			
Testing				2	1.5
Code (re)structure and cleaning			1	3.5	
Report: Report structure, Abstract, Introduction, Data, Dialog System, Conclusion, Team Member Contributions			7		
Report: Machine learning		3		3	
Report: Reasoning, configurability	3.5				
Report: conclusion, configurability					2.5
GROUP MEETINGS	9	9	12	12	9
Total	18	26	26	39.5	20.5