Restaurant Recommendation Dialog System Implementation

Group 17 - Part 1: System Implementation*

JULIUS BIJKERK, Utrecht University, Student Number: 1234567, The Netherlands
CHRISTOS IOANNIDIS, Utrecht University, Student Number: 2250098, The Netherlands
JORT KOEDIJK, Utrecht University, Student Number: 7072775, The Netherlands
LUC MINNEE, Utrecht University, Student Number: 8678901, The Netherlands

1 ABSTRACT

Provide a short summary of the goal of the project, the implementation approach, the machine learning results, the functionality of the dialog system, and possible improvements (250 words maximum).

This paper is about the implementation of a restaurant recommendation dialog system that works on an SVM classifier. The system can classify user utterances, and extract their preferences about features such as the food type, the price range and the area. The system is based on a state transition function that guides it through all the possible scenarios of user input, until the preferences have been collected and a suggestion has been providede. The design choices are all entailed in the report, which also presents the performance of selected classifiers that we experimented with, along with different components of the system.

2 INTRODUCTION

In recent years, the field of Artificial Intelligence (AI) has experienced enormous advancements. Many innovative applications have significantly pushed the boundaries of technology. From AlphaGo (AI system developed by Google DeepMind) that defeated the world's top Go player to the rise of self-driving vehicles and, lastly the generation of highly sophisticated content in the form of images, video, and text. These few examples already show that the potential of AI continues to expand. One of the most prominent areas of AI that has recently gained widespread public attention is generative AI, particularly in the domain of language. Large Language Models (LLMs) are now accessible to anyone with an internet connection, through systems like ChatGPT [1], Gemini [2] (formerly known as Bard), Claude [3], and Perplexity [4] (just to name a few). These systems have become renowned for their ability to interpret, analyze, and generate (super)human-like responses to user prompts. This shows the incredible strength of Natural Language Processing (NLP).

Despite these groundbreaking advancements, it is crucial not to overlook the importance of foundational NLP techniques. While LLMs have demonstrated extraordinary performance, they are not a universally applicable solution. In many NLP applications, traditional machine learning (ML) models are still essential. Therefore, this research focuses on examining the fundamentals of NLP through the development, evaluation, and comparison of several machine learning models.

*Methods in AI Research (MAIR), Utrecht University, 2023-2024
[1] www.chatgpt.com
[2] www.https://gemini.google.com/app
[3] www.https://claude.ai/
[4] www.perplexity.ai/

Authors' addresses: Julius Bijkerk, Utrecht University, Student Number: 1234567, The Netherlands, Utrecht; Christos Ioannidis, Utrecht University, Student Number: 2250098, The Netherlands, Utrecht; Jort Koedijk, Utrecht University, Student Number: 7072775, The Netherlands, Utrecht; Luc Minnee, Utrecht University, Student Number: 8678901, The Netherlands, Utrecht.

In this study, we develop a so-called chatbot. A chatbot is a system designed to simulate a normal human conversation, normally with a specific goal. This chatbot will be aimed at recommending restaurants based on specific preferences that are given by the user. The development of the dialog system will be done step-by-step. The overall idea is that the system can interpret user requests, keep track of states of conversation, and provide relevant restaurant recommendations from a predefined dataset. The interaction is based on specific preferences such as cuisine, price range, and location.

To fulfill this, we will use two key datasets. First, the Dialog State Tracking Challenge (DSTC 2) dataset [reference 2], which consists of 3,235 dialogues in the restaurant domain. Each dialog represents an interaction between a user and a recommendation system. The user has the goal to obtain a restaurant recommendation from the system, based on their preferences. The second dataset is a CSV file containing detailed information on various restaurants, including attributes like name, price range, area, cuisine type, and location.

This study focuses on the process of creating and chatbot and comparing several machine-learning models for user utterance classification. We will implement and compare three distinct models: Support Vector Machine (SVM), Decision Tree (DT), and Logistic Regression (LR). Each model will be assessed against two baseline methods: a majority class classifier and a keyword-matching classifier. Through this approach, we aim to gain deeper insights into the strengths and limitations of traditional ML models in the context of dialog and recommendation systems.

The system's overall design looks as follows:

```
restaurant recommendation project/
├── src/
│   ├── main.py
│   ├── load_data.py
│   ├── baselines.py
│   ├── classifier.py .3 dialog_states.py
│   ├── keyword_mapping.py
│   ├── reasoning.py
│   └── models/
│       ├── logreg.joblib
│       ├── svm.joblib
│       └── tree.joblib
├── notebooks/
│   └── text_classification.ipynb
├── data/
│   ├── all_dialogs.txt
│   ├── dialog_acts.dat
│   └── restaurant_info.csv
├── README.md
└── requirements.txt
```

Fig. 1. Structure of the Restaurant Recommendation Dialog System

Each component in the project is responsible for a specific task:

- **src/** - This directory contains the core Python modules for data preprocessing, baseline models, machine learning models, evaluation metrics, and configuration settings.
  - **main.py** - The entry point for the dialog system, integrating all components for running the system and interacting with users, providing the ability to customize the configuration of the system
  - `load_data.py` - Handles the cleaning and preparation of dialog data, such as converting to lowercase and deduplication.
  - `baselines.py` - Implements the majority class baseline and keyword matching baseline.
  - `ml_models.py` - Contains machine learning model definitions (SVM, DT, LR) and the corresponding training code.
  - `evaluation.py` - Includes functions for evaluating model performance using metrics such as accuracy and F1-score.
  - `config.py` - Manages configuration options for experiments, such as hyperparameters and file paths.
- **notebooks/** - This folder contains a Jupyter notebook (`text_classification.ipynb`) for exploratory data analysis and visualization of experiment results.
- **data/** - This directory holds the original datasets.

To align with the recent trends of open-sourcing AI projects, we have published our work publicly available on GitHub [5].

3 DATA

3.1 Datasets explained

There are three datasets that are used for different purposes. We will start with explaining the dataset of dialog acts, that is used for the machine learning classifier.
The first dataset 'dialog_acts.dat' is focused on dialogs in the restaurant recommendation domain. Each line is a interaction between a user and a recommendation system. The dataset has 25501 dialogs, because there are 25501 individual utterances. Each line in the data consist of two parts:

- Dialog act: The function or intent behind the utterance (is the first words in a dialog)
- Utterance: The spoken words of the user to the system (the rest of the dialog)

There are 5359 unique utterances, so there are $25501 - 5359 = 20142$ duplicate utterances. The English language is used, but because it is based of spoken language it has mainly informal utterances. Also because of the automatic speech recognition the utterances often lack formal sentence structure. The grammar is simplified and sometimes incomplete, because of noise in the speech for example.

The dialog acts are the following:

[5] https://github.com/JungCesar/INFOMAIR

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

Table 1. Dialog Acts and Example Sentences

The second dataset that we used is the 'all_dialogs.txt' file. The previous dataset is a subset of this dataset. So this dataset is also focused on dialogs in the restaurant recommendation domain. There are 3235 dialog acts. The user input often begins with a query for a restaurant's preferences in food type, price range, and location. The system provides information or asks questions to the user. There are also speech acts like: 'inform(type=restaurant, pricerange=moderate)' (the user is informing the system) or 'request(phone)' (the user wants to know more information, such as a phone number). There are 604 unique duplicate utterances in the file, but that is over the whole file. The dataset contains a total of 2,086 duplicate utterances across individual dialogues. The language spoken is also English and has also mainly informal utterances. There are instances where the system doesn't understand the user or misinterprets the user's request. This can be the case due to the informal user input. It sometimes struggles to handle incomplete or ambiguous utterances, as seen when users give short or vague responses.

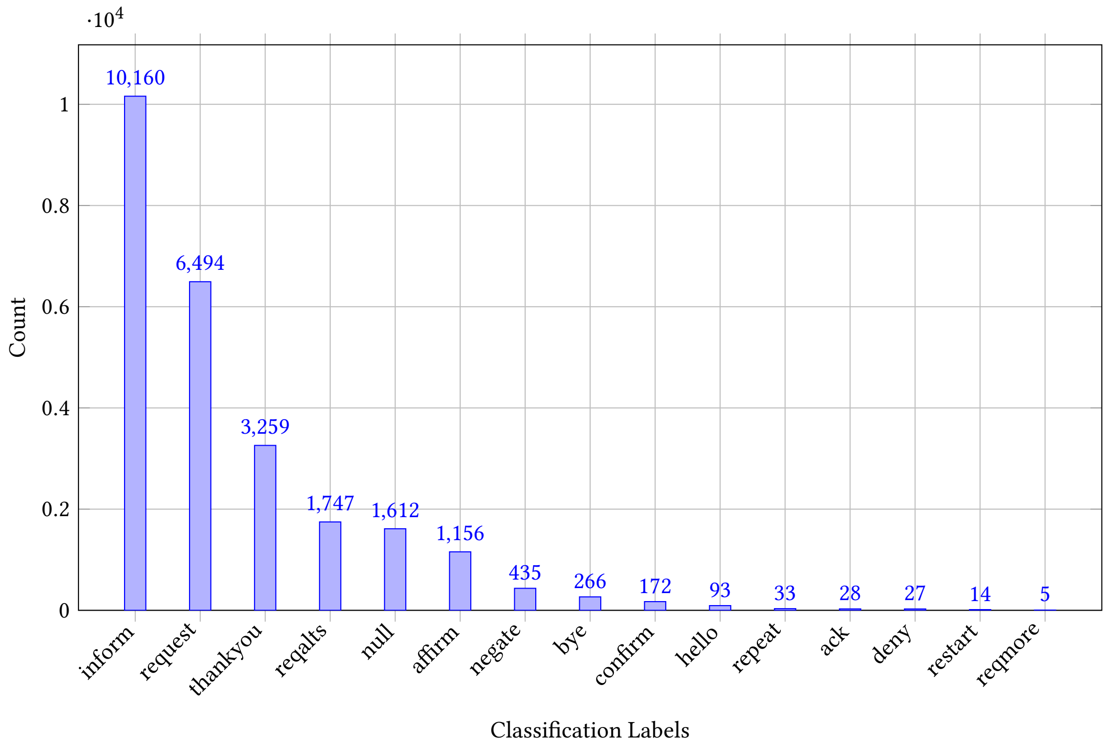


Fig. 2. User Utterance Classification Label Counts

3.2 Dialog diagram

When first inspecting the dialog acts, we were looking to copy the exact same 'moves' the system makes. We started on the dialog diagram that way, but we stumbled across some problems. When the user says that he is 'looking for' a restaurant gave other outputs then when the user says the exact same but instead of 'looking for', he 'wants' a kind of restaurant. So the system can give other answers when the user is using different words, which makes the system a bit inconsistent. Therefore, we used a simplified version by simply asking all the preferences step by step, and then the system looks for a restaurant. The diagram is shown in the appendix 8. We made the type of diagram where the user utterances are made explicit with a single node that represents all possible user utterances.
Here is an example snippit:

The dialog here starts in state 1 and goes on to state 3. It skips state 2, because there is already an area expressed. After state 3 it passes through to state 5.

4 MACHINE LEARNING

Machine learning is a more explicitly defined sub-field of artificial intelligence. There are four types of ML, namely: supervised, semi-supervised, unsupervised, and reinforcement. For this project, there is a focus on supervised ML, since we are using a labeled dataset to improve the model's performance. Before creating the ML models, two baseline models are established. These baseline models are handcrafted and less complicated to program, therefore they provide a sufficient introduction to this project's goal of creating ML models.

4.1 Preproccessing

Before the data was fed into the models we did the following preprocessing: standardize it to all lower characters, remove special characters and lemmatizing the words. We also tried to remove short words (1-2 characters) and removing stopwords. These would actually decrease the performance.

4.2 Baseline Systems

We have created two different baseline systems to classify new user utterances into one of the 15 dialog acts. The baseline systems are very straightforward classification implementations. Both are used to compare against our, more complex, machine learning models.

4.2.1 Baseline 1: Majority. The first one of the two baseline systems we have created to compare our machine learning classifiers against is called the "majority class" model. This is because this baseline model always assigns the majority class label to all of the new incoming user utterances. To create this model, the majority class needed to be found. Therefore we researched the distribution of the classes and found that the "inform" class was over-represented and occurred in 40% of the data. By creating this model, we constructed a very straightforward implementation that already obtains an accuracy of 40%.

4.2.2 Baseline 2: Keyword Matching. The second baseline system that we created has a significantly higher accuracy compared to the one based on the majority class. This implementation was also more complex to create, it verged some research into the dataset to gain an understanding of which words occur often in combination with which dialog acts, i.e. classification labels. To construct this model, a "keywords dictionary" was created, which contained the 15 categories as keys and their corresponding keywords as values. This allowed the system to look up new incoming user utterances, and check if one of the words in the utterance happens to be a value in the keywords dictionary. If so, this keyword is linked to a key in the dictionary, which is the same as the output classification label, which will then be assigned. If not, we again assign the incoming utterance to the most occurring dialog act (classification label), just as in the "majority" baseline model described in section 4.2.1.

4.3 Machine Learning 1: Support Vector Machine (SVM)

For our first machine learning model, we have chosen for the Support-Vector Machine (SVM). According to IBM, the SVM model: "classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space" [3]. It works by finding the optimal hyperplane that best separates different classes in the dataset A crucial issue in the functionality of SVM's are the kernels, which determine how the input data is transformed into a higher-dimensional space to make it separable. In the development of the current implementation, a grid search was performed across different kernels:

- Linear Kernel: The linear kernel computes the inner product between the input vectors, which means it fits a straight hyperplane to separate the classes.
- Polynomial Kernel: The polynomial kernel allows for more complex decision boundaries by computing the similarity between data points in a polynomial feature space.

- Radial Basis Function Kernel: The RBF Kernel measures the similarity between points based on the distance between them, and its importance lies in its ability to map data into an infinite-dimensional space

and with different values for the other hyperparameters, namely the regularization parameters C and gamma. The above influences the performance of the algorithm importantly, and the optimal parameters collected include a linear kernel with a regularization parameter C equal to 1,

4.4 Machine Learning 2: Decision Tree (DT)

A decision tree is a (upside-down) tree-like structure. It contains nodes and branches. Each node questions a specific attribute, each branch corresponds to that attribute value and each leaf node represents the final decision or prediction [1]. The goal is then to create the model such that it predicts the value of a target variable by learning simple decision rules inferred from the data features [2].

In the context of our restaurant recommendation system, the following are important points of consideration concerning decision trees:

- Advantage: Decision trees are relatively easy to understand and it is possible to visualize the tree to observe how the system classifies user utterances.
- Disadvantage: Decision trees can become too complex, also known as over-fitting.This means that it aligns too much with the training data, instead of generalizing beyond that. This could potentially lead to a higher misclassification rate in unseen user utterances.

4.5 Machine Learning 3: Logistic Regression (or logit regression)

There are three types [6] of logistic regression [4], also called logit regression, namely:

- Binomial: In binomial Logistic regression, there can be only two possible types of dependent variables, such as 0 or 1, Pass or Fail, etc.
- Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as “cat”, “dogs”, or “sheep”
- Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as “low”, “Medium”, or “High”.

Since we are working with a dataset that contains 15 different classes, we will implement a multinomial logistic regression model.

4.6 Results

The evaluation of the different machine learning models was conducted using accuracy and other classification metrics such as precision, recall, and F1-score. Figure 2 displays these results which are then discussed in their separate sections results for the Support Vector Machine (SVM) in section 4.6.1, Decision Tree in section 4.6.2, Logistic Regression in section 4.6.3, the results on the stratified dataset in section 4.6.4 and a conclusion based on these results in section 4.6.5.

4.6.1 Support Vector Machine (SVM). The SVM model achieved the highest accuracy of all models, namely an accuracy of 0.982 on the test set. This shows its strong overall performance. The weighted average F1-score was also high: 0.98 This indicates that the model handles the so-called “majority” classes well.

[6] https://www.geeksforgeeks.org/understanding-logistic-regression/

Model	Accuracy	Precision (avg)	Recall (avg)	F1-score (avg)	Support
SVM	**0.982**	0.86	**0.91**	**0.88**	5101
Decision Tree	0.976	**0.89**	0.90	**0.88**	5101
Logistic Regression	0.979	0.84	0.79	0.81	5101
SVM (stratification)	0.909	0.74	0.76	0.74	1072
Decision Tree (stratification)	0.869	0.74	0.76	0.74	1072
Logistic Regression (stratification)	0.899	0.74	0.67	0.69	1072

Table 2. Performance comparison of models on both large and small datasets.

4.6.2 Decision Tree (DT). The Decision Tree model showed an accuracy of 0.976, therefore performing slightly less than the SVM. Notably, class 2 (reqalts) achieved a precision of 1.00 and a recall of 0.92. This highlights the model's specific ability to capture distinctions in that category.

4.6.3 Logistic Regression (or logit regression). The Logistic Regression model showed an accuracy of 0.979, between the SVM and Decision Tree. However, the model faced similar challenges in handling minority classes.

4.6.4 Stratification Dataset Results. For a smaller dataset, containing only 14 classes, SVM, Decision Tree, and Logistic Regression were re-evaluated. In this context, SVM achieved an accuracy of 0.909, performing well for the majority of classes, but still struggling with some minority classes, such as class 0, with a precision of 0.20 and recall of 0.25. The Decision Tree model had an accuracy of 0.869 and suffered from low precision and recall for minority classes, such as class 8. Logistic Regression achieved 0.899 accuracy but showed significant challenges with undefined metrics due to zero predicted samples for certain classes.

4.6.5 Conclusion. Across all models, the SVM showed the best performance in terms of accuracy and F1-score, but all models struggled with minority classes. Imbalanced class distributions significantly impacted precision and recall for smaller classes, suggesting a need for more advanced techniques like oversampling, undersampling, or class-weight adjustments to handle these cases more effectively.

5 DIALOG MANAGER

In the beginning, the system welcomes the user, as in node 1 in the State Transition Diagram in Appendix C, with a system utterance: *"Hello, welcome to restaurant recommender!"*

The user utterance is then classified. What the system expects is the preferences, more precisely at first the expression of the food type. The dialog manager then initially consists of two different branches: one where the food has been expressed, and one where it has not. If it has not, the system will then first ask for the food.
The system has a template for asking for a preference. The core part that always exists is *"What type of [preference category] would you like? Please type 'any' if you have no particular preference."* [template 1]. If this question has been asked and no answer for the particular category has been given, either as a particular value for it or as 'any', the question is replaced by *"Please, specify the type of [preference category] that you would prefer."* [template 2]. For node 2. the aforementioned templates will be uttered with "food" as the preference category. For example, an indicative such dialog start is:

System: *Hello, welcome to the restaurant recommender!*
User: Hi!
System: *What type of food would you like? Please type 'any' if you have no particular preference.*
User: I am trying to think about what type of food I want.
System: *Please, specify the type of food that you would prefer.*

On the other hand, and branch, if the food category has been expressed, the system will follow the template *"You specified that you want the restaurant to be."* [template 3]. This is also the utterance after the response for the food category is given in the other branch.
The above message is concatenated with another one to complete the next system utterance. The next part depends on whether the price has also been defined, which seperates the dialog manager into two branches. If it has not, the system reaches node 3., and second part of the utterance is *"I would also like to know if you want the restaurant to be cheap, moderate, or expensive."*, and the third part of it will be the template 1. If, on the other branch, the food category has already been expressed before, then the system will move on to a question about the area.
The part about the area follows the same logic about before. An example utterance for asking the area and its own template after the food type and price range have been expressed is the following: *"You specified that you want the restaurant to be cheap. Would you like the restaurant to be in the north, south, center, east or west? What type of area would you like? Please type 'any' if you have no particular preference."*.
FUrthermore, the dialog manager recaps the preferences of the user, and asks them about possible changes that they want to make. Next, as in node 5, the system asks for additional requirements, which have their own template, as is evident in the example:
system: *"You have selected: -italian food. -cheap prices. -location in the south. Would you like to change any of them? Please state them now, or we will move on."*
user: *"I want the food to be chinese"*
system: *"You specified that you want the restaurant to be chinese. Do you have any additional requirements, such as assigned seating, a romantic atomsphere, a touristic place or a children friendly environment?"*

If the user replies with 'any', the system will utter: *"You indicated that you have no additional preferences."* before moving on to the querying the database for all the restaurants that match the user's preferences. In the event that the user has provided an unrecognized option for the additional requirements, the template for asking again is: *"No valid preferences were found. Please specify the additional requirements. Or type 'any' to move on."*. After they have been specified, the system reaches seperates in two branches: One where a restaurant match has been found and one where it has not. If it has, the template follows:
"I would like to propose [rice house]. It has [chinese] food, [cheap] prices and is located in the [south]. It is [romantic because it is ideal for long stay and it is not busy.] Are you okay with this suggestion?"
The inferred reasons for this template are the following:
It is romantic because it is ideal for long stay and it is not busy. It is ideal for children because it is ideal for short stay. It has assigned seats as it is busy. It is touristic because it has good quality food, the food is cheap and the food is not romanian. Additionally, when the system offers a suggestion, as written in the above example, it asks for confirmation. If the answer is yes, then the system utters *I hope you enjoy your time in [rice house]*. If not, the converstation resets. If,

lastly, no valid match has been found, even in the occasions of contradiction, the additional preferences are overlooked to look at only the basic ones. e.g. *"No restaurants match the given additional preferences We will attempt to relax the additional preferences and suggest a restaurant without them."*

An example of a full conversation that traverses the diagram from the start state to the end state can be found in the Appendix A.

6 REASONING

After defining the basic preferences of the user that have to do with the food category, the price range, and the area of the restaurant, the next part of the dialog regards the specification of some extra requirements from the user. Namely, the user has to mention if they wish for a touristic restaurant, a romantic atmosphere, assigned seats, or a children-friendly restaurant.

When the user defines these, the system has to apply some inference rules that apply the three basic features that were mentioned before, as well as some added features (crowdedness, length of stay, food quality). In other words, from the additional preferences, the system implies some additional filters for the recommendation process. Namely, the rules are:

- If the user specifies that they prefer a touristic restaurant, then the system creates a filter for the restaurant database that the food quality is good (food quality), the prices are cheap (pricerange), and the restaurant is not Romanian (food).
- If the user expresses their wish for a romantic atmosphere, the system will then conclude that it should only include restaurants that allow long stay (length of stay) and also disregard the ones that are busy (crowdedness).
- If the user prefers a children-friendly environment, the system should imply that the restaurant does not work with long stay (length of stay).
- If the user wants to opt for a restaurant that provides assigned seating, then it is inferred that the restaurant is busy (crowdedness).

Therefore, after the first filters are collected in the parts of the dialog where the system asks for the food, price range and area preferences, the system will then ask for the additional requirements and convert them into the specified filters, while notifying the user for the reasoning behind each choice. It is worth noting that in this part there is a possibility of contradictions. For example, if the user expresses that they prefer touristic place despite having indicated that they prefer an expensive restaurant, the resulting filter would be that the restaurant is both cheap and expensive at the same time. The design choice that has been implemented in this point is that the system will then disregard the additional requirements, and focus on the previously specified filters. The reasoning system is communicated in the utterance templates: *It is romantic because it is ideal for long stay and it is not busy. It is ideal for children because it is ideal for short stay. It has assigned seats as it is busy. It is touristic because it has good quality food, the food is cheap and the food is not romanian.* And the contradictions are handled by the following template: *No restaurants match the given additional preferences. We will attempt to relax the additional preferences and suggest a restaurant without them.*

7 CONFIGURABILITY

One of the core strengths of the restaurant recommendation system lies in its configurability, and in this section, the provided possibilities will be analyzed. The configuration options that our system supports are listed below, and then a high-level description of them is provided:

- The system can provide suggestions either after each user preference declaration, or after all of them are expressed.
- The system's keyword matching algorithm can work either with exact matches between the user's uttered preference and the possible restaurant features in the database or with a loosened criterion, where the Levenshtein distance can be computed between the aforementioned to account for spelling mistakes or variant spellings.
- The system can either allow changes in preferences while communicating with the user, or adhere to the value that is expressed for each feature initially.
- The system supports text-to-speech for system utterances, provided that this configuration option is chosen.

7.1 Description of configurability options

7.1.1 Early suggestions. Firstly, it is reminded that the system asks for the user's preferences for the food type, the price range, the location, and the additional preferences sequentially. Therefore, the scenarios are that either the system will expect all the parameters for the search to be filled before querying the database for matching restaurants, or it can proceed to query the database after every single user preference is acquired. With the latter, it is possible at any point when a suggestion is made by the system that the user can accept the suggestion, thus bringing the dialog to an end. It is worth mentioning that based on the current implementation, when the option to provide suggestions after every preference is stated is enabled, every restaurant that is rejected by the user will not be presented again in the following suggestions. The reason for this design choice is to avoid repetitions, especially in cases where the restaurant matches more than one of the user's preferences, but they have already dismissed it.

7.1.2 Levenshtein distance. The Levenshtein distance has been implemented in the system to account for spelling mistakes by the user (e.g. "moderate" instead of "moderate") or, as mentioned, variant spellings (e.g. “center” is the standard spelling, while in British English, “centre” is commonly used). However, it can introduce misunderstandings between the user's preferences and the matched value for a certain feature. For example, if the threshold to match two words was a Levenshtein distance of 2, the user could be searching for an "inexpensive" restaurant, whereas the system would match the particular user utterance to the value "expensive". Whereas in the current implementation, the Levenshtein distance threshold is set to 1 to avoid such confusion, the system provides the configurability option to completely disable matching while accounting for this distance and presents a more rigid implementation.

7.1.3 Ability to change preferences. The initial way that the system was designed allowed a user to express their preference for different features even if the question asked regarded a particular one. For example, the system would keep track of the user's declaration that they wish for a cheap restaurant, despite the question regarding the food option. Moreover, if at any point a preference would be expressed about a feature that had already been defined before, the system would take into account only the most recent preference. Lastly, after the user has provided their basic preferences (food, price range, area), the system would print out the acquired preferences and ask for any modifications that the user could ask for. With this configuration option disabled, the system can once again become more rigid, and:

- only keep track of expressed preferences if the particular feature has not been defined earlier
- not ask for modifications before moving on to the part of the additional requirements

7.1.4 System utterances with text-to-speech. Another feature that enhances the system's usability and accessibility is its ability to provide text-to-speech for system utterances. This feature makes the user interface more evolved when enabled. Every time the system outputs an utterance in the terminal, the audio of this utterance will be echoed by the system. This particular functionality is implemented through a dedicated speech engine, allowing the system to convert the textual content of its prompts and responses into spoken language.

8 CONCLUSION

In this project a chat bot was made to give users a recommendation for a restaurant they want to eat at. Classifiers and keyword matching algorithms were used to achieve this. The classifiers were used to determine the intention of the user like 'inform' and 'hello'. However, this does not always work as expected. Although the models used had around 98% accuracy on testdata, it did not always predict the right outcome. For example, "Hello, I would like Italian food" is classified as 'hello' instead of 'inform'. A possible improvement on this is to fine-tune the classifier or to use a different way to measure the users intention. This limited the way we could create the transition function.

The keyword matching also had some limitations. The algorithm looks for (almost) exact matches, but not for synonyms. If the user asks for an "inexpensive Italian restaurant" the system will not recognise the 'cheap' feature, because it does not look for synonyms. A way to solve this problem is to include synonyms into the matching list as well.

The final models used for the classifiers have a high accuracy in their testdata. Trying more hyper parameters or finding out a new model would be a waste of time.

ACKNOWLEDGMENTS

We would like to thank our course instructors, Dr. D. Doder and Dr. D.P. Nguyen, for the interesting and interactive lectures. We also would like to acknowledge the teaching assistants, especially C. Zhu, MSc, for their guidance and support during the workshops.

REFERENCES

[1] Breiman, L. *Classification and regression trees.* Routledge, 2017.
[2] Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. *The elements of statistical learning: data mining, inference, and prediction*, vol. 2. Springer, 2009.
[3] IBM. What are svms? https://www.ibm.com/topics/support-vector-machine#:~:text=A%20support%20vector%20machine%20(SVM,the%201990s%20by%20Vladimir%20N., Dec. 2023.
[4] Yu, H.-F., Huang, F.-L., and Lin, C.-J. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning 85* (2011), 41–75.

OVERVIEW OF CONTRIBUTIONS OF INDIVIDUAL GROUP MEMBERS

Overview of the contributions of individual group members to the project as a table containing tasks, the person that performed the task, and the amount of time spent on this task. Include a brief reflection on the planning of the project and the individual contributions. Add the total amount of hours per person. Some tasks are likely to be performed by everybody, such as reading dialogs at the start of the project to understand the data. However, try to limit the number of entries assigned to everyone to a minimum, and instead split the task to reflect what each person has actually done.

Task	**Julius**	**Christos**	**Jort**	**Luc**
Part 1a: Dialog Act Classification				
Majority baseline	1hr	–	2hr	–
Keyword baseline	2hr	–	2hr	–
ML1: SVM	1hr	4hr	–	–
ML2: Decision Tree	1hr	–	–	–
ML3: Logistic Regression	1hr	–	–	–
Evaluation	1hr	2hr	–	–
Deduplication	1hr	1hr	–	–
Overall program structure	2hr	3hr	–	–
Part 1b: Dialog Management				
State transition diagram	–	3hr	–	1hr
Dialog manager implementation	–	4hr	3hr	2hr
Preference extraction algorithm	–	4hr	3hr	–
Levenshtein distance integration	–	1hr	2hr	–
Restaurant lookup function	–	3hr	–	3hr
User interface development	6hr	4hr	3hr	3hr
Part 1c: Reasoning and Configurability				
Inference rules implementation	–	3hr	–	–
Configurability features	2hr	2hr	2hr	2hr
Contradiction handling	–	2hr	1hr	–
System integration	1hr	1hr	3hr	1hr
Updated state transition diagram	–	2hr	–	1hr
Report Writing				
Abstract and Introduction	3hr	–	2hr	–
Data	2hr	2hr	1hr	2hr
Machine Learning	3hr	1hr	2hr	1hr
Dialog manager	1hr	4hr	2hr	1hr
Reasoning	–	3hr	2hr	2hr
Configurability	–	2hr	1hr	1hr
Conclusion	–	–	1hr	1hr
Proofreading and Formatting	8hr	1hr	4hr	1hr
Total Hours	**43hr**	**52hr**	**39hr**	**16hr**

Table 3. Detailed work distribution across tasks and team members for Parts 1a, 1b, and 1c.

APPENDIX

A FULL EXAMPLE CONVERSATION

An example of a complete conversation from start to finish is provided below:

System: *Hello, welcome to the restaurant recommender!*

User: Hi!

System: *What type of food would you like? Please type 'any' if you have no particular preference.*

User: I would like some Italian food.

System: *You specified that you want the restaurant to be Italian. I would also like to know if you want the restaurant to be cheap, moderate, or expensive. What type of price range would you like? Please type 'any' if you have no particular preference.*

User: I want it to be cheap.

System: *You specified that you want the restaurant to be cheap. Would you like the restaurant to be in the north, south, center, east, or west? What type of area would you like? Please type 'any' if you have no particular preference.*

User: I would prefer if it was in the south.

System: *You have selected:*

- Italian food.

- Cheap prices.

- Location in the south.

Would you like to change any of them? Please state them now, or we will move on.

User: No!

System: *Do you have any additional requirements, such as assigned seating, a romantic atmosphere, a touristic place, or a child-friendly environment?*

User: A romantic atmosphere would be nice!

System: *No restaurants match the given additional preferences. We will attempt to relax the additional preferences and suggest a restaurant without them.*

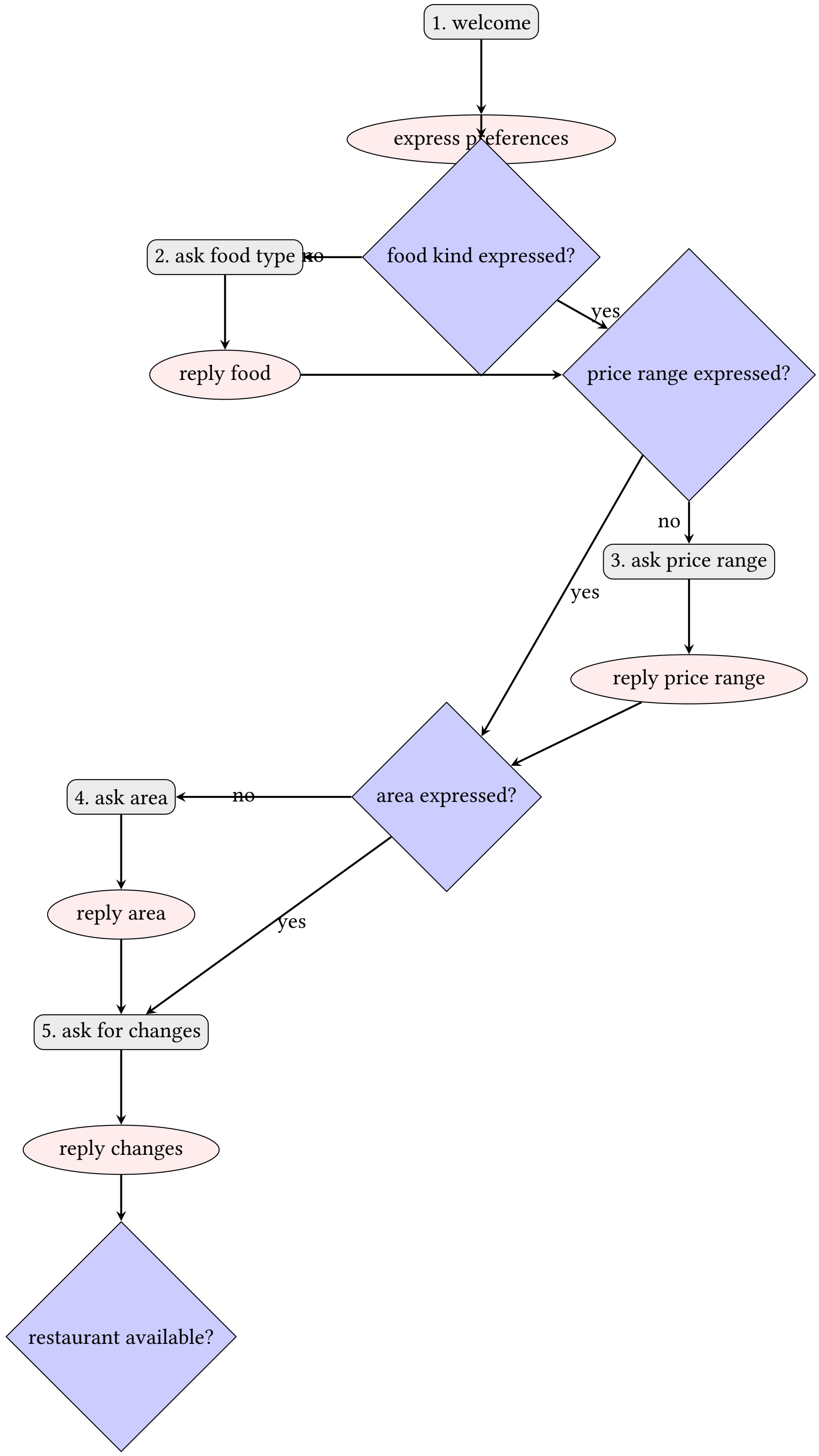
System: *I would like to propose La Margherita. It has Italian food, cheap prices, and is located in the south.*

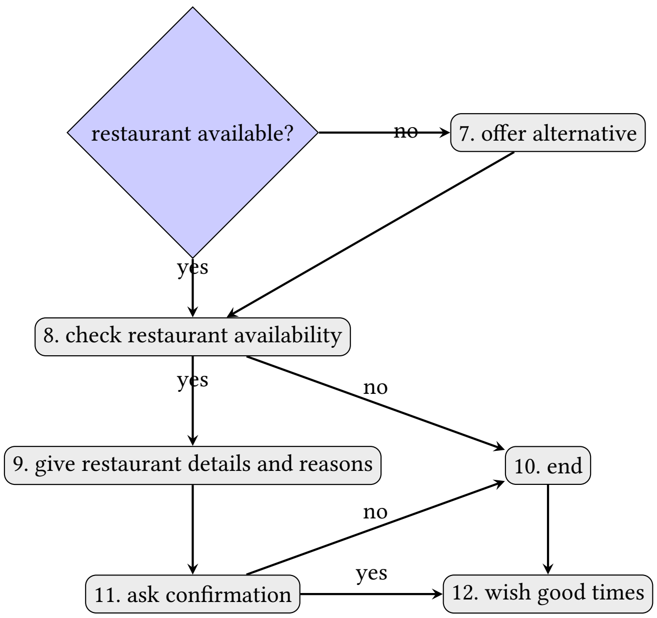
System: *Are you okay with the aforementioned suggestion?*

User: Yes!

System: *I hope you enjoy your time at La Margherita.*

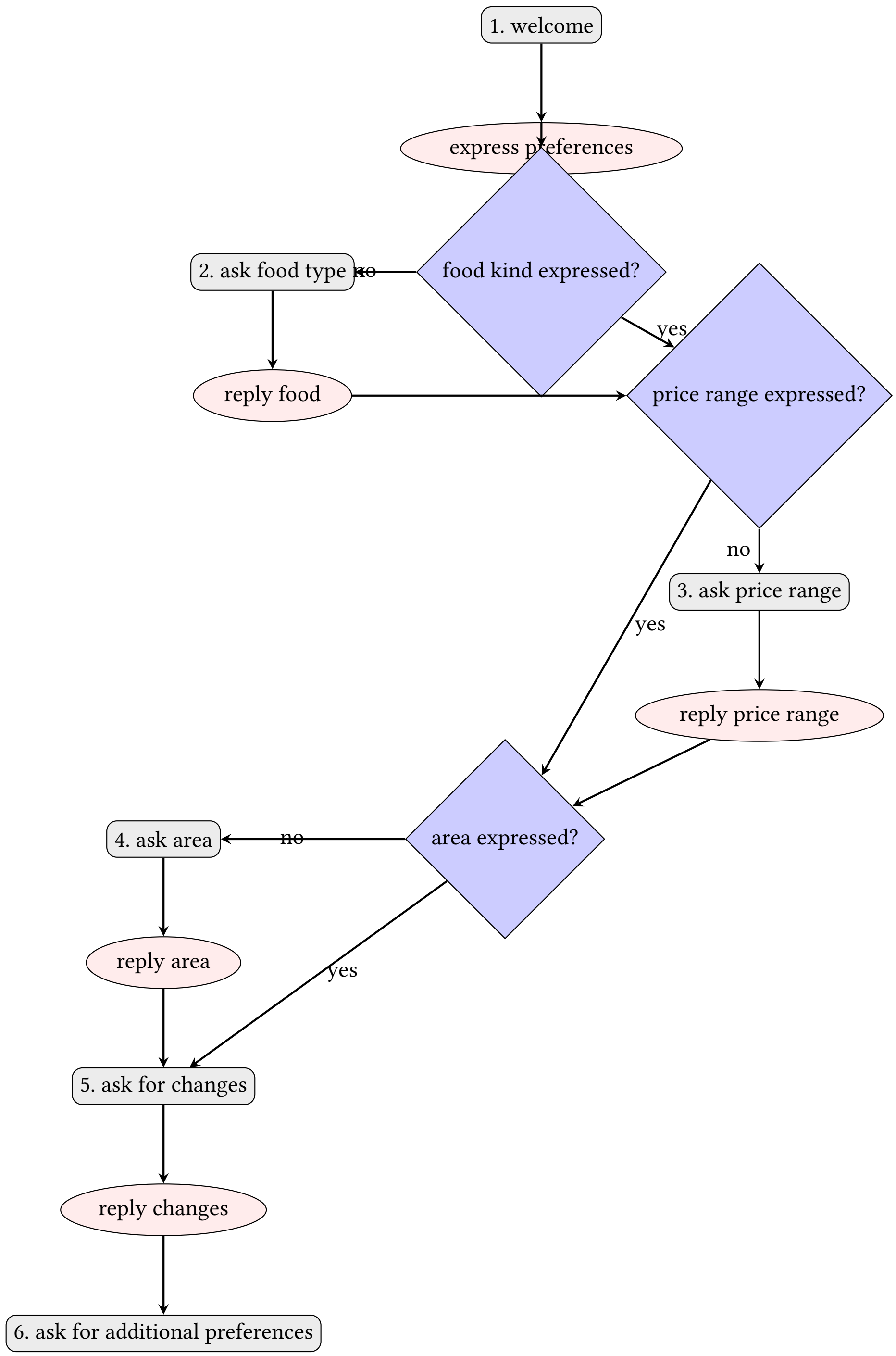
B THE STATE TRANSITION DIAGRAM 1B

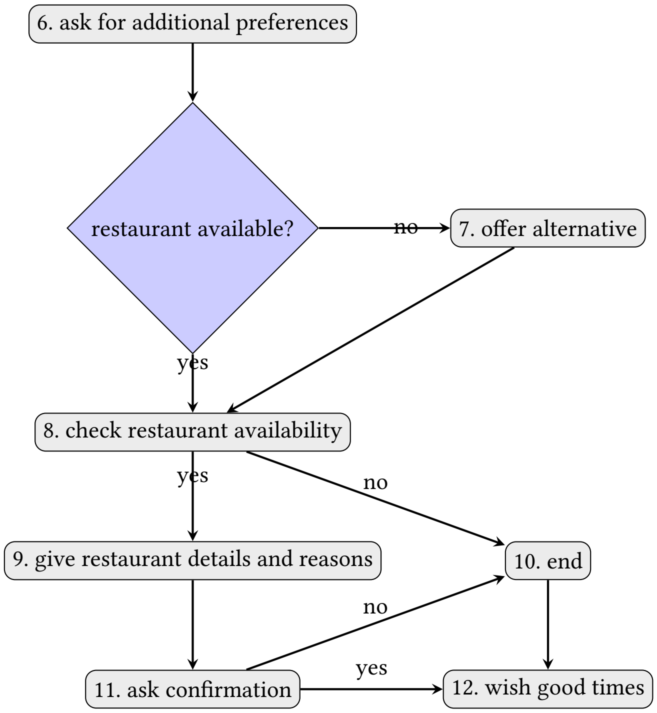




C THE STATE TRANSITION DIAGRAM 1C

The figure below shows the dialog transition diagram for our implementation of a restaurant recommendation system.


1. welcome
express preferences
food kind expressed?
no
2. ask food type
yes
reply food
price range expressed?
no
3. ask price range
yes
reply price range
area expressed?
no
4. ask area
reply area
yes
5. ask for changes
reply changes
6. ask for additional preferences


6. ask for additional preferences
restaurant available?
no
7. offer alternative
yes
8. check restaurant availability
yes
no
9. give restaurant details and reasons
10. end
no
yes
11. ask confirmation
12. wish good times