Dining Out Made Easy: Creating a Restaurant Recommender Dialog System

Group 6

EFRAIM DAHL, 1695568

JUUL PETIT, 6446981

MATTHEW SCHEERES, 4545966

CHENGHONG(PATRICK) YANG, 2274035

RUBEN VAN DER ZEIJDEN, 7081111

Abstract

Our aim is to build up a dialogue system including a classifier and a dialogue manager. We first built up several machine learning systems using the Natural Language Processing technique to understand the meaning of users' dialog. Including Decision Tree, KNN, Ridge regression, MLP, and baseline models trained on dialog files. We made model comparisons and error analyses based on model output. We achieved the best F1 score of 0.99 on the MLP model. We built up a dialog manager based on the python-statemachine package. We drew a state diagram to simulate different dialog circumstances, including *Ask qualifier* and *Preference reasoning* parts. Moreover, we added a reasoning component to the system by using manual reasoning rules. Furthermore, we added 6 configurabilities into our system to make the system more intelligent.

Keywords: Dialogue system, Natural Language Processing, Machine Learning, Text Classification, Artificial Intelligence

Authors' addresses: Efraim Dahl, 1695568; Juul Petit, 6446981; Matthew Scheeres, 4545966; ChengHong(Patrick) Yang, 2274035; Ruben van der Zeijden, 7081111.

1 INTRODUCTION

1.1 Scope of the project

The goal of this project is to design a task-based dialogue system in the domain of restaurant recommendation. In order to achieve this goal, we built a natural language processing machine learning algorithm that can classify user utterances in terms of their actions. Furthermore, we modelled the dataset by designing a dialog transition diagram. Together, the machine learning classifier and dialog transition diagram will be used to implement a dialog manager. This dialog manager is able to have actual dialogues with it's users. Additionally, we added a reasoning part in which the user can provide the system with any further preferences. Next to this, we implemented some configurability options that can possibly ease the dialogue with the user.

1.2 Dialog system

In order to achieve the best working dialog system, we reviewed the following models: Majority class baseline model, Keywords baseline model, Decision Tree model, K-nearest neighbors model and the Ridge regression model. We implemented all models on both the complete dataset and the deduplicated dataset. More information and results of the quantitative analysis of the models will be discussed in section 3. After comparing the models, the Decision Tree model was evaluated as best performing, so we chose to continue using this model to build our dialog manager.

1.3 User interaction

The type of user interaction of our model is initially text-based. Meaning, that the user can interact with the system by typing text. The system, on it's turn, will respond with text-based replies. Nevertheless, we implemented some configurability tasks one of which realises text-to-speech for system utterances. Thus, the system's responses will be read out by our algorithm. Further explanation on this configurability task can be found in section 6.

1.4 Components of our system

Our final system is made up out of different components. First, the natural language processing (section 2), second, the text classification system (section 3) and third, the dialog transition diagram (section 4). These three components enabled us to design our dialog manager (section 4). Furthermore, we implemented the reasoning component (section 5) and the configurability tasks (section 6).

2 DATA

The dialog acts dataset was used to train the classifiers analyzed in assignment 1a, of which one was later applied to classify dialog acts for the final chatbot used in 1b and 1c. It contains 25500 entries, taken from Microsoft's Dialog State Tracking Challenge (DSTC2) dataset (Henderson, Thomson, and Williams 2014). The dialog act labels and corresponding dialogs are stored as tab-separated values in a .dat-file format. The dataset contains a lot of duplicates, which might be representative of some sentences occurring more often in natural language but may also introduce bias for the same reason. If a user utterance was classified with multiple dialog acts in the original data, only the first dialog act was included for the data used in this assignment. In **??**, the constructed state transition diagram can be seen.

Example dialog:

- Chatbot: Hello , welcome to the UU restaurant system? You can ask for restaurants by area , price range or food type . How may I help you? (State: 1. 'Welcome')

- User: I'm looking for a nice restaurant in the south of town. (Dialog act: Inform, area: south)
- Chatbot:: What kind of food would you like? (State: 5. 'Ask foodtype')
- User: Cheap thai food please (Dialog act: Inform, pricerange: cheap, foodType: thai)
- Chatbot:: I'm sorry but there is no restaurant serving thai food in the south that has cheap prices. (State: 12. 'No restaurant found')
- User: How about in the center and for any price? (Dialog act: inform, area: centre, price: dontcare)
- Chatbot:: bangkok city is a nice place in the centre part of town serving thai food and the prices are expensive (State: 8. 'Return restaurant')
- User: Sounds good! What are their address and telephone number? (Dialog act: request)
- Chatbot:: The phone number of bangkok city is 01223 354382, their address is 24 green street city centre. (State: 9. 'Give information')
- User: Thanks, bye! (Dialog act: bye)
- Chatbot:: Thank you for using the UU restaurant system. Goodbye! (State: 13. 'Completed')

3 MACHINE LEARNING

In this section we train a number of different machine learning models for predicting the dialog act (or label) of a user utterance. Specifically, we train a Decision Tree model, K-nearest neighbor model, Ridge regression model and MLP model. We then evaluate the systems based on a quantitative analysis and an error analysis, discuss difficult cases and compare the systems to each other as well as to two baseline models (Majority class and Keyword).

3.1 Model selection

To classify the dialog acts of user utterances, we trained four machine learning models: a Decision Tree model, K-nearest neighbor model, Ridge regression model and MLP model. These models were chosen as they represent a broad range of contemporary machine learning approaches for classification tasks. For all of these models we used the implementations in the Keras library. Furthermore, we made use of two models are not machine learning models but are used as a baseline performance test for the machine learning models. This includes the Majority class model, which simply predicts the most frequent label in the training set as the dialog act for every sentence, and the Keyword Matching model, which employs a manually designed keyword matching algorithm to predict the label. The Keyword Matching model was created by exploring which words or phrases are used in which context and adding these phrases to the model with the associated labels. The order in which these keywords are checked in the code was chosen carefully to ensure that more common situations are checked first. For example, if a sentence contains both "thank you" as well as "goodbye" it is usually classified with the label "thankyou" in the data, so by checking for "thank you" before checking for "goodbye" we ensure that the sentence is classified correctly. This is necessary, since the dataset used in this assignment only includes the first dialog act from the original data if multiple are possible. Lastly, when no keyword match is found, the sentence is classified as "inform" since this is by far the most common label.

Category	inform	request	thankyou	reqalts	null	affirm	negate	bye	confirm	hello	repeat	ack	deny	restart	reqmore
Number	10160	6494	3259	1747	1612	1156	435	266	172	93	33	28	27	14	5

Table 1. Numbers of instances in each category.

3.2 Data preprocessing

First, let's make a rough observation of the dataset. We can see all categories distributed unevenly in table **??**.

Training models required vectorized data, so we used one-hot coding to vectorize the whole training file. Other than the training text, we also vectorized labels for the models' input and restored the labels for analytic purposes. This is for improving the compatibility of data during training on different models.

We also compare the differences between models trained with or without stopwords. The reason is stopwords are common words in a language (e.g., "the," "and," "in") that are often removed from text data during the preprocessing step when training natural language processing (NLP) models. These words are considered to be of low value or significance in many NLP tasks, and their removal can help improve the efficiency and effectiveness of the models. However, after the removal, performance dropped by 0.10 on average. So we just use all words on the training file to vectorize.

3.3 Quantitative Analysis

To perform the quantitative analysis, the following metrics were chosen: precision, recall, f1 score and accuracy. The weighted means of these metrics for all our models are presented in this report.

First, we will explain why we used these metrics. Precision is the ratio of true positive predictions compared to the total number of positive predictions. It is a useful metric for our evaluation to analyze the amount of false positives. Recall is the ratio of true positive predictions compared to the total number of real positives. This metric is important to examine whether our models miss out on true positives. The combination of precision and recall is the F1-score. This valuable metric captures false positives as well as false negatives. The balance between precision and recall is of great importance in the evaluation of our models. Last, we used the metric accuracy, which measures the ratio of correct predictions compared to all predictions made. It assesses the overall performance of our models and is thus an indispensable metric in our evaluation.

The formulas of the means of the metrics explained above are defined as follows (Deisenroth, Faisal, and Ong 2020):

$$\begin{aligned}
Precision &= \frac{1}{15}\sum_i \frac{TP_i}{TP_i + FP_i}, \\
Recall &= \frac{1}{15}\sum_i \frac{TP_i}{TP_i + FN_i}, \\
F1\ score &= \frac{1}{15}\sum_i \frac{2TP_i}{2TP_i + FP_i + FN_i}, \\
Accuracy &= \frac{1}{15}\sum_i \frac{TP_i + TN_i}{TP_i + FP_i + FN_i + TN_i}.
\end{aligned}$$

In the above formulas, TP, FP, TN and FN respectively represent true positives, false positives, true negatives and false negatives. $i = 1, 2, ..., 15$ represent the ith label of our system.

The models were evaluated both on the complete training data set and on the deduplicated training data set.

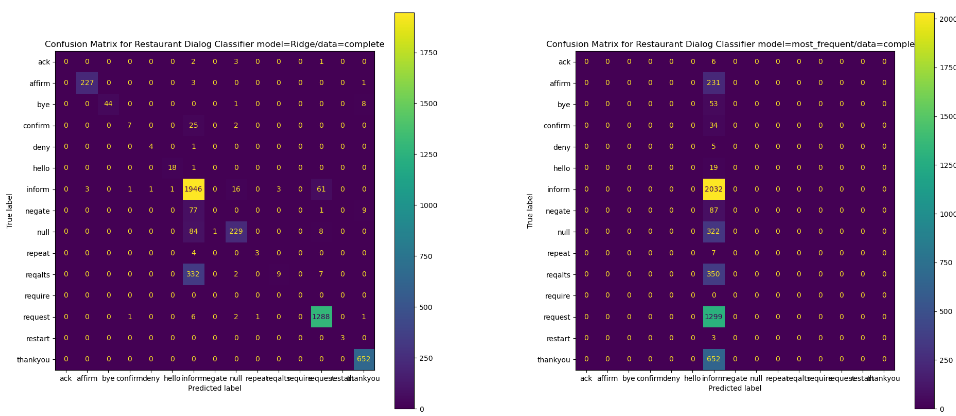
In subsection 3.6 the results of our models will be discussed and the models will be compared.

3.4 Error Analysis

For the error analysis, we will discuss two dialog acts that are more difficult to classify and why. Additionally, we will review whether there are particular utterances that are hard to classify for all models and why.

Figure 1a and 1b show the confusion matrices on the complete dataset for the Ridge regression model and the Majority class baseline model, respectively.

Figure 1a shows that 336 utterances that should be labeled as 'reqalts' are labeled as 'inform'. In the confusion matrices of the Decision Tree model, K-nearest neighbors model and the MLP model (figure 5c, 5d and 5f in appendix D) the same defect arises. This error occurs, among others, at the following utterances: 'no id rather find a moderately priced restaurant' or 'no is there anything else'. Both are labeled with dialog act 'inform' while they should actually be labeled with dialog act 'reqalts'. The error also occurs the other way around for the Decision Tree, the K-nearest neighbors and the MLP model. We can see in figure 5c, 5d and 5f that there over a hundred utterances labeled with 'reqalts' while they should be labeled with dialog act 'inform'. In the confusion matrices for the deduplicated dataset, which are left out of this report for the purpose of clarity, the same inaccuracy is observed for the Decision Tree, K-nearest neighbors, Ridge regression and MLP model.



(a) Ridge regression model

(b) Majority class baseline model

Fig. 1. Confusion matrices for the Ridge regression model and the majority class baseline model trained on the complete dataset

In the confusion matrices 1a and 1b, we can also see that utterances are almost never classified as 'negate'. Often, dialog acts that should be labeled 'negate', are labeled as 'inform'. Utterances as 'no i want spanish food' or 'no moderately priced' are labeled as dialog act 'inform' while they should be labeled 'negate'. Thus, this dialog act is difficult to classify.

Furthermore, there are some labels that have few samples in the training data. The dialog acts 'repeat', 'ack', 'restart', 'deny' and 'reqmore' for example, do not have more than seven samples in the training data. This automatically complicates classifying these dialog acts correctly.

3.5 Difficult Instances

Concerning the difficult cases, five sentences were identified:

(1) **Utterance:** ‘no i dont want vietnamese food i want italian food’
Predicted label: deny/inform
(2) **Utterance:** ‘luck the missing sock’
Predicted label: inform
(3) **Utterance:** ‘thank you and bye’
Predicted label: : thankyou
(4) **Utterance:** ‘street’
Predicted label: inform
(5) **Utterance:** ‘greec’
Predicted label: inform

We will evaluate the difficult cases that are listed above one by one.

First, there is the case where there is a negation followed by an informative utterance. Some of our models classify this utterance as the dialog act ‘deny’, since it would make more sense to classify it as ‘inform’. The keyword baseline model classifies this utterance as ‘deny’ on both the complete dataset as the deduplicated dataset. The Decision Tree model and the K-nearest neighbors model classifies this assertion as ‘deny’ only on the deduplicated set. The other models classify it ‘correctly’ as dialog act ‘inform’.

Second, our systems need to handle speech recognition issues such as the assertion: ‘luck the missing sock’. Obviously, the keyword baseline model classified this utterance as ‘null’ since it did not recognize any keywords. Remarkable is that the Decision Tree model trained on the deduplicated dataset also returns ‘null’ as label and de Decision Tree model trained on the complete dataset returns ‘reqalts’. All other models return the dialog act ‘inform’ on this assertion.

Third, there is the case in which a sentence could be labeled as two different dialog acts. For example, the utterance ‘thank you and bye’, could be labeled as both ‘thankyou’ and ‘bye’. All our models, excepting the majority class baseline which labels it as ‘inform’, classified this assertion with ‘thankyou’. In the original data it was possible for sentences to be classified with multiple labels, which would be appropriate in this case, but we are using a simplified dataset which only includes the first label for each sentence, which causes information to be lost in sentences like these.

Fourth, some assertions are simply words instead of full sentences. When the model is asked to classify the utterance ‘street’, most of the models return the label ‘inform’, while it is quite obvious for a human being that the user is making a request. This is probably the case because the word ‘street’ does not once occur in the training dataset. If we, for example, feed the model the utterance ‘address’, it does classify it correctly as ‘request’. Still, a perfect model should recognize this assertion and classify it as ‘request’.

Last, the test data could consist of spelling mistakes. There are some spelling mistakes contained in the training dataset, such as ‘venesian’ instead of ‘venetian’ and ‘tha’ instead of ‘thai’. The model should be able to recognize the mistake and label the assertion correctly. The utterance ‘greec’ was labeled by our model as follows: ‘null’ by the keywords baseline model trained on both datasets, ‘reqalts’ by the Decision Tree model trained on the complete dataset, and ‘inform’ by all the other models. Thus, our model is recognizing this spelling mistake pretty well.

3.6 System Comparison

The experimental results of the quantitative evaluation will be discussed in this section. Furthermore, it will compare the models we used.

Table 2 shows the results of the quantitative evaluation on the complete dataset. Table 3 shows the results of the quantitative evaluation on the deduplicated data set.

Metric	Majority class baseline	Keywords baseline	Decision Tree	KNN	Ridge	MLP
Precision	0.16	0.86	0.97	0.97	0.96	0.98
Recall	0.40	0.84	0.98	0.97	0.95	0.98
F1 score	0.23	0.84	0.97	0.97	0.96	0.98
Accuracy	0.40	0.84	0.98	0.97	0.95	0.98

Table 2. Results of quantitative evaluation on the complete dataset for the baseline models, Decision Tree model, K-nearest neighbors model, Ridge regression model, and MLP model.

Metric	Majority class baseline	Keywords baseline	Decision Tree	KNN	Ridge	MLP
Precision	0.30	0.75	0.95	0.90	0.91	0.96
Recall	0.55	0.74	0.95	0.89	0.90	0.96
F1 score	0.39	0.73	0.95	0.90	0.91	0.96
Accuracy	0.55	0.74	0.95	0.89	0.89	0.96

Table 3. Results of quantitative evaluation on the deduplicated dataset for the baseline models, Decision Tree model, K-nearest neighbors model, Ridge regression model, and MLP model.

We can see in both tables that overall, the MLP model scores highest on all four metrics trained on both the complete as the deduplicated dataset. We can see that the baseline models perform better on the deduplicated dataset and the other models perform better on the complete dataset. Both datasets have advantages and disadvantages when used for different purposes. The complete dataset is preferable for training our models. It contains all data that is available and thus, is able to understand the underlying relationships in the data. Furthermore, the complete dataset is closest to real-world data and can therefore give a more realistic assessment of the performance of our models when applied in the real world. In contrast, the deduplicated dataset reduces the impact of noise and overfitting in the data, resulting in more reliable and efficient predictions when the model is used in a new dialogue.

When comparing the models' confusion matrices visually, it is rather clear that the chosen models perform much better than the baselines. After deduplication, this is still the case. Still, it is immediately visible that the amount of samples in each category reduces drastically, with the total amount of samples dropping from 20400 and 4534 to 5101 and 1589 in the stratified training and test sets, respectively.

When comparing accuracy scores between the models, it can be seen that the MLP and the Decision Tree classifier outperform the other models on both the complete and deduplicated test sets. Logically, it would make sense to use the MLP or Decision Tree classifier for further development of the dialog system. We chose to implement the Decision Tree model in our system.

Additionally, it might also be interesting to explore similar methods, like a bagging or random forest classifier.

4 DIALOG MANAGER

This section contains information about the state model we used for the implementation of our Dialog Manager (figure 3, appendix B). First, the development of our diagram is explained. Next, we will explain the final version of our diagram. Additionally, we will review an example dialog and reflect on it using our state diagram. Lastly, the system utterance templates will be described using example sentences.

4.1 State Model and Transitions function

Let us consider the following example dialog to explain the dialog system. See figure 3 in appendix B for a simplified state diagram

System: Hello, welcome to the UU restaurant system, how can I help you?
Explanation: We start with the initial *Hello* state, the transition *start_processing* is invoked automatically. We are now in the *State preferences* State, where we can receive user input.
User: I am looking for a french restaurant
Explanation We remain in the *State preferences* state until the user inputs something. Once a message is received, this message is forwarded by the *input_received* transition to the *(*Process preferences) state. Here the parsed input is processed to update matching system variables. In this case, the parser will return "foodType=french" and the foodType variable will be set. In our base implementation, our parser and the *(*Process preferences) state can handle multiple preferences being stated. **System**: What area of town did you have in mind
Explanation: By default the system will step through the *Ask Area*, *Ask FoodType*, *Ask PriceRange* states in that order, skipping states if the corresponding variable is already known.
User: any
Explanation The parser is context-aware (see more in) and in this case, the context of the state being *ask area* is known. This means that the generic answer "any" is recognized as applying to the area. The parser returns "area=dontcare" and the state machine is updated accordingly. The variable "foodType=french" is already set from the first user utterance, so the *Ask foodtyp* State is skipped and we reach the *Ask pricerange* State.
System: Would you like something in the cheap , moderate , or expensive price range?
User: I would like a cheap place
Explanation: The input is passed, so we take the **receive_input** transition from the *Ask pricerange* to the *Process preferences* state. Now that all our variables area, foodType and priceRange are known we automatically transition to the *Return restaurant* state. Here the system performs a CSV lookup for restaurants that match the preferences. When we implemented the reasoning component, this was where the reasoning component was inserted. See section 5 for a detailed explanation of the reasoning component. In this case, no restaurant is found, which triggers a transition to the *No restaurant found* state.
System: I'm sorry there is no restaurant serving french food at cheap prices.
User: How about an Italian restaurant?
Explanation: We are in the *No restaurant found* state, and the system recognizes a change in preferences: "foodtype=italian". The state-machine variables are updated, and a transition back to the *return restaurant* state is triggered. A CSV lookup with new preferences (area="dontcare", price="cheap",foodtype="italian") is performed.
System: la margharita is a nice italian place in the area and the prices are cheap

User: What are the address and postal code of the restaurant

Explanation: The system recognizes the user input as a request, and the requested values as address and postal code. The recommended restaurant from the *Return restaurant* state is saved. We take the *receive_input* transition to the *Give information state*. In the *Give information state* the requested information is retrieved from the saved restaurant.

System: The address of la margherita is 15 magdalene street city centre, their postcode is c.b 3.

User: I would like an expensive place instead

Explanation: The user input is recognized as a preference change, so we transition to the *Process alternative* state, where the preferences "price=expensive" are updated. We transition back to the *Return restaurant* state.

System: frankie and bennys is a nice place in the south part of town serving italian food and the prices are expensive

User: another please

Explanation: The user input is recognized as a request alternative without a preference update. Internally the system keeps track of how many times requests have been made with the same preference configuration. This gets cleared if preferences are changed. We proceed down the list of restaurants matching the preferences.

System: caffe uno is a nice place in the centre part of town serving italian food and the prices are expensive

User: What is their phone number

System: The phone number of caffe uno is 01223 448620.

User: What is their address

System: The address of caffe uno is 32 bridge street city centre

Explanation: While in the *Give information* state, we can continue asking more information about the restaurant.

User: thank you goodbye

Explanation: The input is recognized as a conversation end, a transition to the *Complete* state is initiated.

System: Thank you for using the UU - Restaurant System, Goodbye.

4.2 Additional Remarks on the State Model and choices made

In the preference collection stage, multiple preferences can be collected from a user utterance, even if the state machine is in the *Ask area*,*Ask foodtype*, *Ask pricerange* states. Multiple preferences can also be updated at the same time when we are in the *Return restaurant*, *No restaurant found* or *Give information states*. Generally, the newest preferences are the ones searched for. It is possible to override previously collected preferences, while in the preference collection state. The conversation can be ended from the *Give information*, *No restaurant found*, *Return restaurant*, and in the *State preferences* state. If a user utterance is unintelligible, there are either dedicated error handling states such as the *Cant give information* state, that provide useful error information, or self-to-self redirects that simply re-prompt the user without updating any internal settings. We chose to only search for restaurants if all three preferences were stated. A possible choice would have been to keep track of how many restaurants are left after each preference update and start returning restaurants once the selection is small enough. This is, however, also not the case with the Cambridge System. See the following two sections for changes to the state machine for configurability and preference reasoning.

4.3 Implementation Details

State machine: The state machine is built using the statemachine package, which gives us many simplified syntactical options to define conditional transitions and transition order, functions to be executed on arrival or exit of a given state. The extraction of information and classification is handled by the parser.

User input parsing and classification: The basic classification is done using a trained classifier, by default it is a Decision Tree classifier, but this can be changed in the main function of statemachine.py. Depending on the classification, the parser then looks for different keywords in the sentence. If an utterance is classified as inform, reqalts, confirm, negate, or request words are matched to known possible food types, price ranges, or areas, as well as looking for indicators that specify ambivalence such as "any area". Here levenshtein edit distance is applied to look for similar words that may be the result of a misspelling. Preference is given to correctly spelled words. If several preferences of the same category are given in the same utterance i.e "I am looking for a restaurant in the north in the west", preference is given to the latter.

If an utterance is classified as a request, and the parser is given the context of the state-machine being in the Return restaurant or Give Information stages, then we look for keywords that indicate what information is looked for such as address, phone number, etc. To come up with these lists of keywords, we used the supplementary/dialog_explore.py file, to match whether the keywords covered all, or at least most example cases.

4.4 System Utterances

We will differentiate between informing utterances, user-prompting, and error messages.

For user prompting, as well as greetings and goodbye, we stayed very close to the Cambridge System, all of these are static: 1: "Hello, welcome to the UU restaurant system. You can ask for restaurants by area, price range, or food type. How may I help you?"

2: "What part of town do you have in mind?"

3: "What kind of food would you like?"

4: "Would you like something in the cheap , moderate , or expensive price range?"

5 : "Thank you for using the UU restaurant system. Goodbye!"

For returning restaurants and restaurant information the output is more dynamic.

1: "*restaurantname* is a nice place in the *area* part of town serving *food* food and the prices are *pricerange*"

2: For information about the restaurant, only the information requested is provided. I.e

"The address of caffe uno is 32 bridge street city centre" additional requested fields are appended: "The address of caffe uno is 32 bridge street city centre, their phone number is 01223 448620."

3: If no restaurant is found, only fields that are specified are included. The statement, "I'm sorry but there is no restaurant serving french food in the south that has cheap prices" turns into "I'm sorry but there is no restaurant serving french food that has cheap prices" if "area=dontcare" is specified.

Additionally, we have error messages if input is not understood, as well as pointers to correct them. 1: in the *give information* stage: "Can you provide the specific information you are looking for such as phone number, area or address?"

2: If the user tries making requests for information if no restaurant is provided: "Sorry, but there are no such restaurants, maybe try changing the location, area or foodtype?"

5 REASONING

In this section we will describe our reasoning component. Additionally, we will explain our reasoning component based on example dialog snippets of our system.

5.1 Reasoning Component

We started our reasoning component by adding the parameters for food quality, crowdedness and length of stay randomly to our database with restaurants. After this, we expanded our classifier to recognize the key utterances ‘children’, ‘assigned’, ‘romantic’ and ‘touristic’. Note that our system only recognizes these utterances in their positive form. A phrase like ‘I want a restaurant that is not romantic’ will therefore not be recognized properly. We chose to use the consequent ‘assigned’ generally to allow for more utterances than just ‘assigned seats’ such as ‘assigned tables’ or ‘assigned by waiter’. Furthermore, while multiple properties are possible at the same time, we are only extracting one property at a time.

We prepared our reasoning, by creating a file, where the inference rules, the relevant field names and values, as well as the corresponding descriptions are stored in an easily processed dictionary format.

5.2 Dialog examples explaining the Reasoning Component

Our state machine was expanded by two states. *Ask qualifier* and *Preference reasoning*. Let us consider the following dialog examples:

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

User: I am looking for an italian restaurant in the city centre

System: Would you like something in the cheap , moderate , or expensive price range?

User: any

Explanation: Now all preferences are known (foodType=Italian, area = centre, priceRange = any). First our conditional transition *evaluate_input* checks if there are restaurants left, after applying the preferences. If there are restaurants left, we transition into the *Ask preference* state.

System: Do you have additional requirements?

User: I would like a romantic place

Explanation: Our parser recognizes the keyword "romantic" and our system is updated accordingly. Since qualifier=romantic is given we transition into the *Preference reasoning* state. It is in this state where the reasoning takes place. Using the inference rules defined in the data/reasoning_rules.json the system searches for applicable restaurants. In the case of romantic we have following inference rules, romantic-true if length-of-stay=long, romantic-false if crowdedness=crowded. A restaurant being crowded does not necessarily exclude the possibility of a long stay. Both conditions can be true at the same time. To handle theese contradictions, the following ranking is applied. Restaurants where the positive qualifier is true AND the negative qualifier is false, i.e a restaurant that is not crowded and allows for long stays are ranked first. If no suitable restaurant is found, we proceed to recommend the restaurants that are non-negative. I.e a restaurant that is not crowded. Restaurants where both qualifiers are true, are ranked third, i.e crowded restaurants with long stays. With a list of suitable restaurants in descending order and corresponding explanation now chosen, we proceed to the *Return restaurant* state. If no qualifying keyword is recognized, we proceed with returning restaurants in the regular fashion

System: pizza hut city centre is a nice place in the centre part of town serving italian food and the prices are cheap

System: pizza hut city centre was chosen because spending a long time in a restaurant is romantic

User: Another

Explanation: We work our way through the list of suitable restaurants set by the *Preference reasoning* state.

System: I'm sorry but there is no other restaurant serving italian food in the centre ,that is romantic
Explanation: No other restaurants matches the romantic criteria. We explain this to the user.
Explanation: No other restaurant matches the romantic criteria. We explain this to the user.
User: How about any area
Explanation: The system detects a preference update, and we clear the romantic qualifier. There are several restaurant that match this criteria so we proceed to the *Ask qualifier state*
System: Do you have additional requirements?
User: I want to bring my children
System:la margherita is a nice place in the south part of town serving italian food and the prices are cheap
System:la margherita was chosen because a cheap place might be more child-friendly than an expensive one
User: Thank you goodbye
System: Thank you for using the UU restaurant system. Goodbye!

6 CONFIGURABILITY

We added some degree of configurability to our dialog system. The user is able to switch the different configurability features on and off before the dialog starts. As mentioned in the introduction of this section, the configurability choices will improve the adaptability, customization and compliance of our model to best serve the user's needs. Configurability allows our model to adapt to the needs of the user which we implemented as preparation for user experiments. By adding these configurability options, users can fine-tune the models behavior to best suit their specific requirements which will enable us to improve the user experience by retaining some features and removing others in the future.

In order to get to the best performing dialog manager, we considered a lot of configurability features as options to implement to our system. Eventually, we decided to implement the following:

(1) Confirm with user the correctness of match for Levenshtein edit distance results
(2) Allow preferences to be stated in random order or not
(3) Allow preferences to be stated in a single utterance only or in multiple utterances with one preference per utterance only, or without restrictions
(4) Introduce a delay before showing system responses
(5) Output in capslock or not
(6) Use text-to-speech for system utterances

For each of the features mentioned above, we will explain why we chose the specific feature and how we implemented it in our code.

Firstly, we want our system to confirm with the user whether it correctly found the right match when choosing a word using the Levenshtein edit distance. Say, for example, that the system is receiving the following input: 'i'm looking for a restaurant in the soth part of town'. As humans, we can easily conclude that the user probably means 'south' instead of 'soth'. In our code, we implemented the Levenshtein edit distance (explained in [SECTION REF]) to recognize unintelligible utterances. The system uses the word with the smallest Levenshtein edit distance from the utterance as the proposed correct word. In the example above, the Levenshtein edit distance of 'south' is only one compared to 'soth'. The system will therefore consider this to be the intentional utterance of the user. Yet, this would still be an assumption rather than a fact. Therefore, we wanted to implement the speech act of affirming when our system identifies an unintelligible word using the Levenshtein edit distance. Accordingly, when the Levenshtein edit distance is two or less, the system will recognize the word and ask the user whether it correctly identified the user's utterance.

Secondly, we want our system to be able to allow preferences to be stated in random order or not. To enforce the order, the properties are ranked in the following order; area, food type and price range. We only allow the second variable to be set if the first variable is already known in our statemachine. Likewise, we only allow the third variable to be set if the first and second variable are already known. Otherwise, the preference will be ignored. If several preferences are stated in a single utterance, only the one with the highest rank will be extracted.

Thirdly, we wanted to implement a configurability option that would remove restrictions in user utterances.

Fourthly, we aspire to create a system that is most similar to a human being. Humans need some reaction time before responding appropriately. Therefore, we want to introduce a delay before showing system responses. Accordingly, the system will show 'Thinking...' representing a human being's reaction time. Afterwards, the system will show the applicable response.

Fifthly, the configurability option of returning the output of our system in capslock is added. This could be a helpful feature for visually impaired users. It may also improve clarity and consistency, which could be favorable for some users.

Lastly, there is another option that could help the visually impaired. Specifically, we added text-to-speech for system utterances. This is often a valuable accessibility feature that would make our model more inclusive. Users have different preferences for how they consume content and with this option, they can choose their own preferred mode of interaction.

7 CONCLUSION

7.1 Dialog system

In this assignment, we implemented a dialogue system for restaurant recommendations through the use of a state machine model for handling user input and generating system responses. This was done by training machine learning models to classify user utterances with dialog acts and updating the state of the system dynamically based on the dialog act and the preferences expressed by the user. This turned out to be an effective way of keeping track of what information was known to the system and what had to still be learned in order to recommend a restaurant. The dialogue system was also adaptive by allowing users to request alternative recommendations or change their preferences after a restaurant was recommended, and users were able to configure different aspects of the system. Additionally, some basic reasoning capabilities were added to the system in order to recommend restaurants based on inferred properties. However, our implementation of the dialog system is limited by a few factors. For one, when extracting user preferences, the dialog system is not able to deal with negations. The utterance "I am looking for a place that is not expensive" is treated the same as "I am looking for a place that is expensive", which can lead to incorrect recommendations being provided. Secondly, in most example cases we did not find the affirmative or confirming system utterances to be particularly helpful as they were seemingly randomly returned by the system. Therefore, we only implemented them for the configurable Levenshtein confirmation option in part 6.1. Another issue is that the dataset was based on speech recognition, which causes many bad responses by the system, such as asking for confirmation five times in a row since a user utterance was not understood. The dialog system could likely be improved by using a different dataset which is based on written data. In the future, it would be a good idea to test alternative approaches than state diagrams since these are not very generalize to different data than we used here.

7.2 Dialog Act Classification

To classify user utterance with a dialog act, we trained four machine learning models. Of these, the MLP model was the most accurate, with an accuracy of 98% and an F1 score of 98% (on the dataset with duplicates not removed). This was quite an improvement over the baseline of 84% accuracy for the keyword matching baseline and 40% for the majority label model and the other machine learning models (KNN, Ridge and Decision Tree models). However, one limitation of the dialog act classification is that it only predicts one label for each user utterance, which is also a limitation of the data used. This meant that sentences such as "Thank you goodbye" were classified as "thankyou" while they should also be classified as "bye". Another issue with the data is that the sentences are all very similar to each other, which means it is easy to get a well performing model, but puts into question the wider applicability of the trained models to other data with more natural sentences. The participants used in creating the data set were given specific instructions about what they should order, which is not the case for an actual user of the system, so more natural user input might differ. The applicability of the models would likely be improved by using more natural data.

8 INDIVIDUAL CONTRIBUTIONS

Task	Efraim	Juul	Patrick	Ruben	Matt
1a setup data preparation/training evaluation files	3.5h	1h	2.5h	1h	3.5h
1a model implementation			4h		
1a baseline models				4h	2h
1a refactoring/handling feedback			2h	4h	1h
1b State Machine	4h		1h	7h	4h
1b Text Parser	4h				
1b Levenshtein			2h		
1b State Machine Overhauls	5.5h				
1b State Transition Diagram	5.5h	4h			0.5h
1b Refactoring / Handling Feedback					3.5h
1c Dataset Preparation					1h
1c Configuration Levenshtein	1.5h		3h		
1c Configuration TTS+Utterance Limit	0.5h		3h		
1c Configuration Enforce Order	1.5h				
1c Reasoning	5h	2h			
1c Configurable Visual Chatbot Features					2h
Misc. - Code documentation					3h
Final Report - General	1h			2h	3h
Final Report - Abstract			2h		
Final Report - Introduction		3h			
Final Report - Data		3h			3h
Final Report - Machine Learning		5h	3h	4h	
Final Report - Dialog Manager	2h	5h			
Final Report - Reasoning	2h	4h			
Final Report - Configurability		5h			
Final Report - Conclusion				2h	
Total	30.5h	29h	22.5h	24h	26h

BIBLIOGRAPHY

Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong (2020). *Mathematics for Machine Learning*. Cambridge University Press.

Henderson, Matthew, Blaise Thomson, and Jason D. Williams (June 2014). “The Second Dialog State Tracking Challenge”. In: *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. Philadelphia, PA, U.S.A.: Association for Computational Linguistics, pp. 263–272. DOI: 10.3115/v1/W14-4337. URL: https://aclanthology.org/W14-4337.

A APPENDIX A: FIRST STATE DIAGRAM

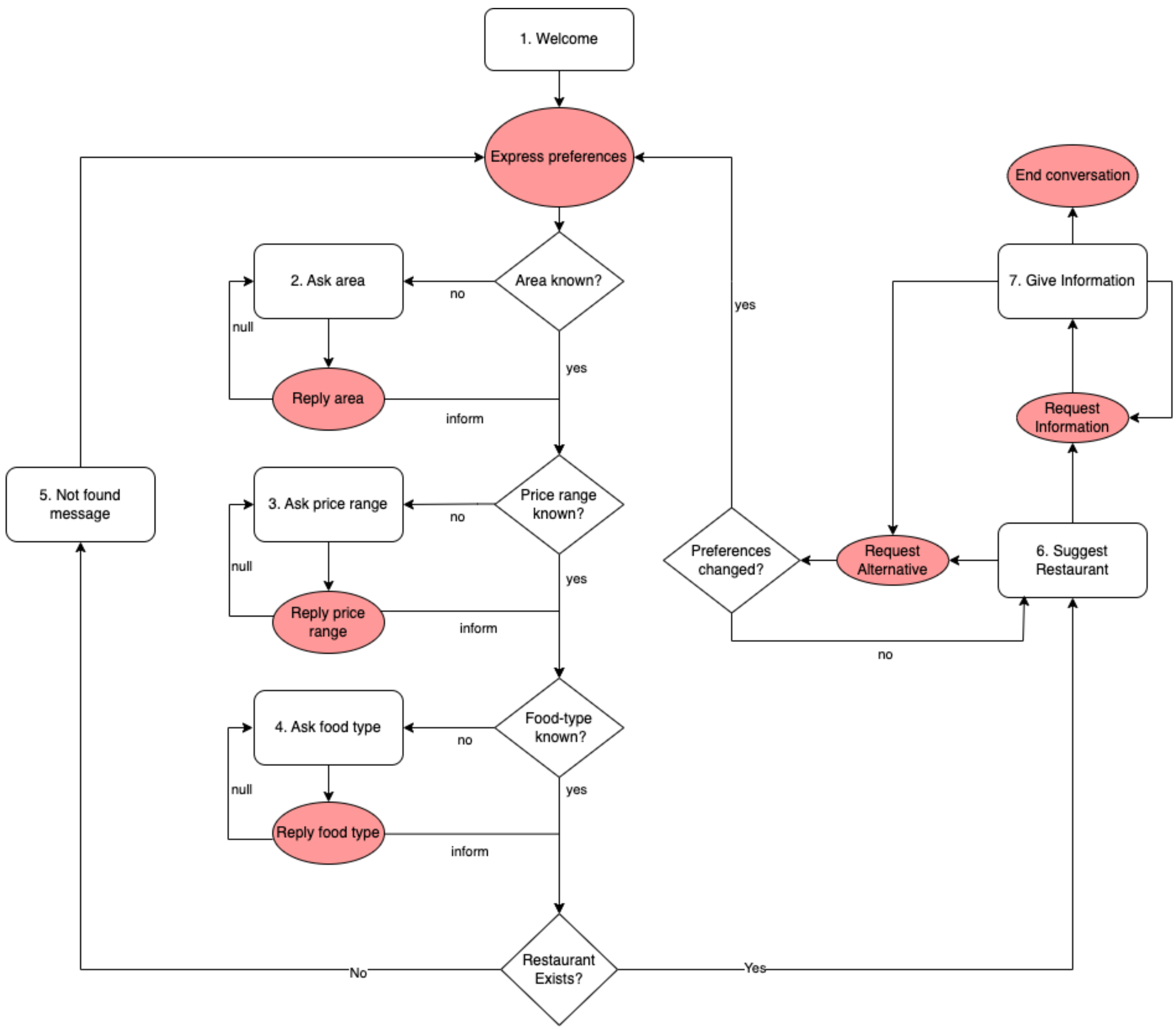


Fig. 2. First State Diagram

B APPENDIX B: FINAL STATE DIAGRAM

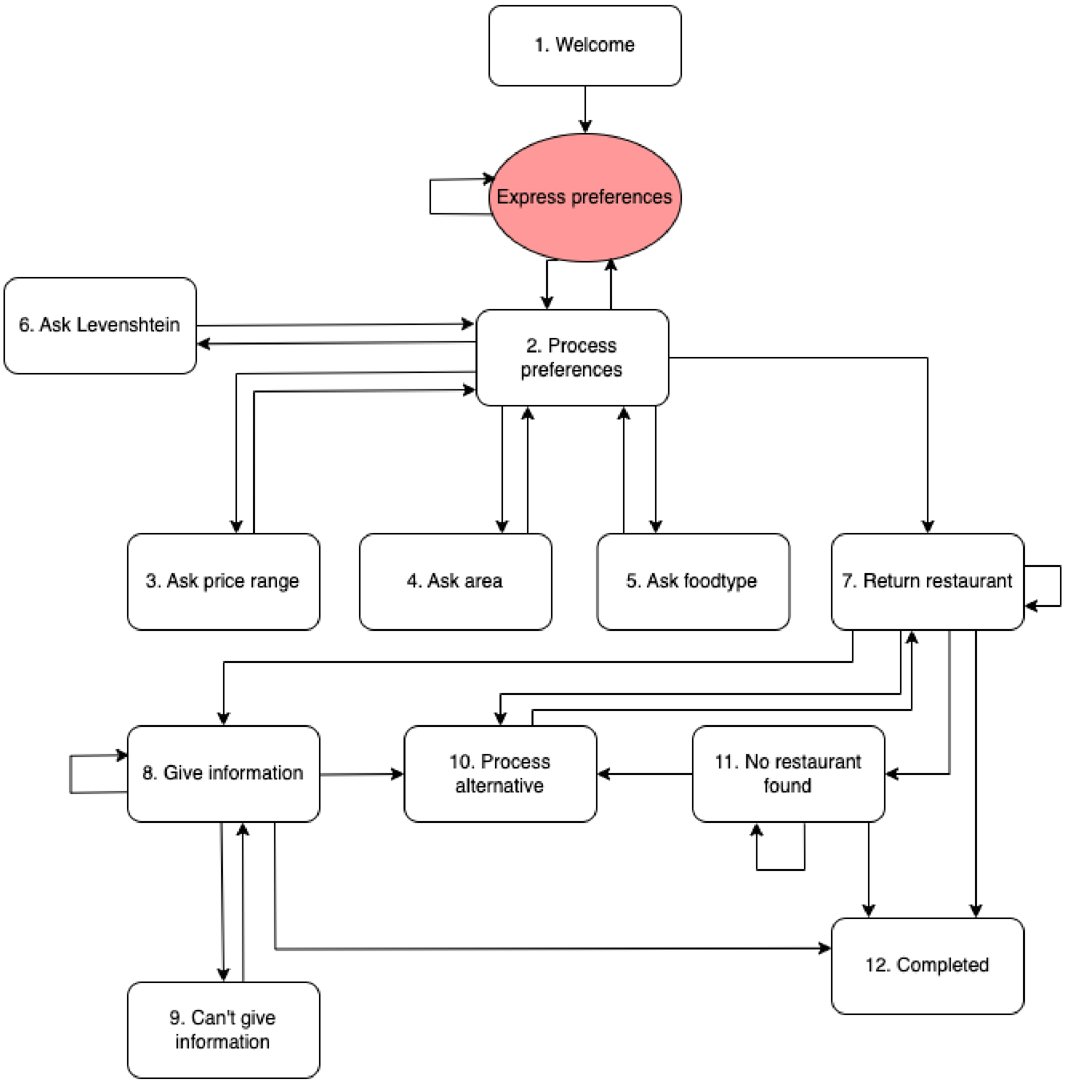


Fig. 3. Final State Diagram

C APPENDIX C: FINAL STATE DIAGRAM WITH REASONING COMPONENT

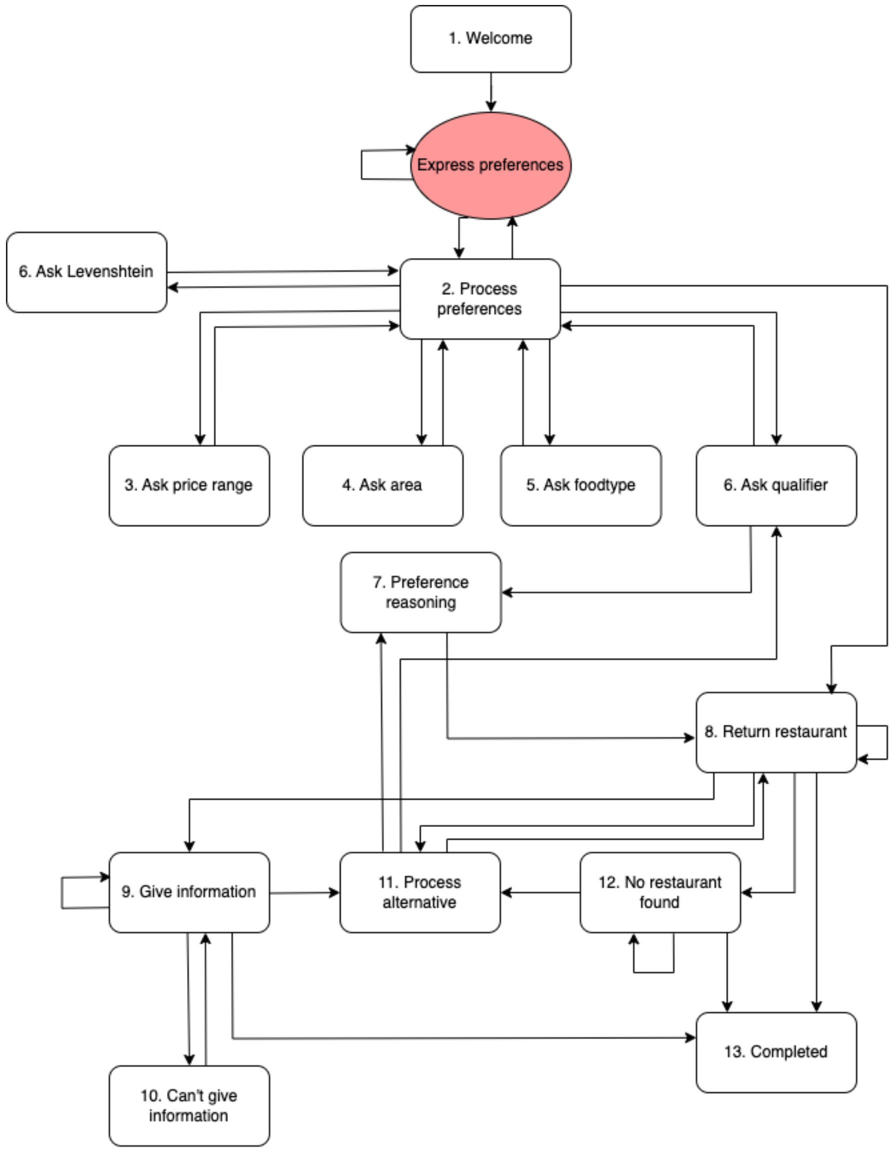
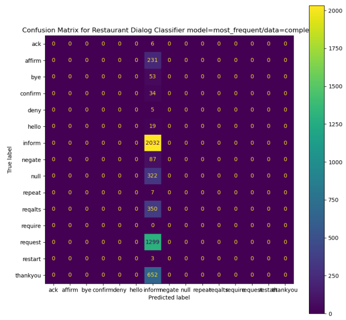
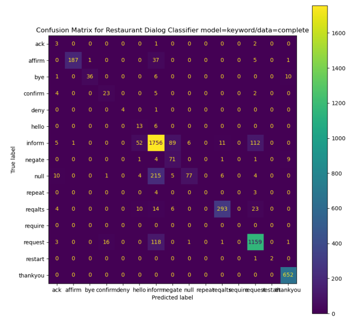


Fig. 4. Final State Diagram with Reasoning Component

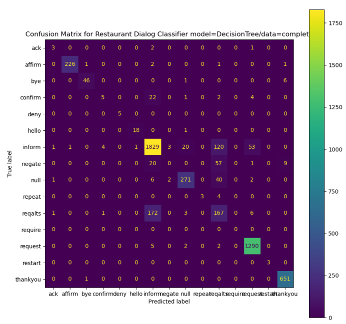
D APPENDIX D: CONFUSION MATRICES FOR ALL MODELS ON THE COMPLETE DATASET



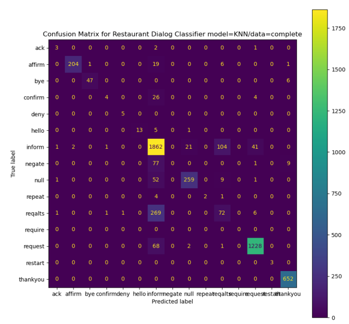
(a) Majority baseline model



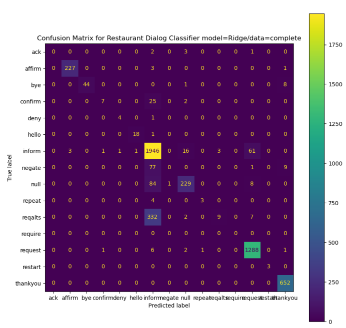
(b) Keyword baseline model



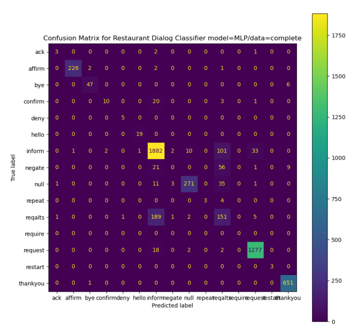
(c) Decision Tree model



(d) K-nearest neighbors model



(e) Ridge regression model



(f) MLP model

Fig. 5. Confusion matrices of all models on the complete dataset