A Conversational Agent for Cooking Recipes

Matteo Destro (221222)
University of Trento
Via Sommarive, 9, 38123 Povo, Trento TN
matteo.destro@studenti.unitn.it

Abstract

This work presents a conversational agent for helping users in searching and preparing cooking recipes. The system was developed entirely with the Rasa framework and the source code and dataset are available in the public repository[1].

1 Introduction

This work presents a conversational agent to help users in cooking food recipes, developed with the *Rasa* framework. The main idea behind this assistant is to provide an easy way to search and prepare recipes in an interactive manner.

The proposed agent is *task-based*, where the final goal is to guide the user through a recipe, providing additional information and question-answering during the process. In particular, the agent is able to search for recipes in a database, allows the users to specify additional constraints (e.g. dietary preferences), provides additional information about the recipes and ingredients, guides the users during the preparation step-by-step, and assists them in personalizing the recipe (e.g. suggesting alternative ingredients or adapting the quantities for a specific number of people).

In designing the agent no assumption was made about the characteristics of the final users, making it usable by anybody without any prior knowledge about its features. To do so, its design is centered around a *mixed initiative* approach, so that the functionalities can be discovered progressively by interacting with it.

In the following sections the details of the system are presented. In Section 2 the conversation design is discussed, while Section 3 gives an overview of the data used as knowledge base and for training the model. Section 4 describes the general pipeline of the agent and finally Section 5 presents the evaluation process and the final performances.

[1] https://github.com/materight/cooking-assistant

2 Conversation Design

The type of dialogues the system can handle can be characterized by three main consecutive stages:

1. **Recipe search:** in the first stage the user can search for a specific recipe. The agent can handle different types of search criteria: by *recipe name*, *ingredients*, *cuisine* or *tag* (i.e. dietary constraints like "gluten-free" or recipe properties like "quick"). Any number and combination of criteria can be specified by the user. In case multiple alternatives are found, the agent can ask the users additional questions to further narrow down the list. Alternatively, the user may ask explicitly for an alternative recipe to the one the system proposes.

2. **General information retrieval:** after a recipe has been selected, the user can ask additional questions to obtain more details, like the required ingredients, the expected preparation and cooking times, or possible substitutes for any missing ingredient. More details can be also given to the agent so that the recipe can be adapted accordingly, e.g. changing the ingredients quantities based on the total number of people eating.

3. **Step-by-step preparation:** once the user has all the information needed and everything ready, the actual cooking can start. The agent will read the recipe step-by-step and wait for the user response before continuing. In this phase the user can also ask more questions about the previous step, like the amount needed for a particular ingredient for a certain amount of people, the expected time required, or search for a possible substitute to an ingredient. Moreover a timer can be set in case it is needed for a recipe, and the agent will give an alert when the time is up.

As stated previously, the agent was designed around mixed initiative dialogues, so that it is able to both

guide the user through a recipe and answer questions about it. For instance, the user may specify some dietary requirements when searching for something to cook, but the agent is also able to ask for any preference in case multiple results are found in the database.

The agent implements an *implicit confirmation* strategy in most of its responses since the task is non-critical and this approach reduces the number of turns needed to reach the goal w.r.t an explicit confirmation. A set of *conversational markers* are also present in most of the agent's responses, both as explicit acknowledgment signals and as timeline markers (e.g. when reading the steps of a recipe). Moreover, a multimodal component is present, since the user is also provided with an image of the recipe along its description.

In case the user provides an unknown or unexpected intent, the agent implements a *two-stage fallback* and can ask the users to confirm their intentions. If the agent is still not able to handle the input, a simple fallback response is returned informing the user that his request was not understood. No human hand-off or other last-fallback policies were implemented since they would not fit the main use case. More details about the fallback and error recovery policies can be found in Section 4.

3 Data Description and Analysis

The *conversation data* was initially written manually to account for the most common happy and unhappy interaction paths, but it was progressively expanded using the interactions with real users. The data went through a total of three revisions, each taking into account the conversations of four different users on average. This process allowed to discover additional unhappy paths that were not included in the original stories.

As explained in Section 2, each story can be subdivided in three main stages: *recipe search*, *information retrieval* and *step-by-step preparation*. Therefore, the stories for these steps were written separately and recombined at training time using the *checkpoint* functionality of Rasa. This allowed to increase exponentially the total number of stories available for training. An average of 7 stories per stage were used to train the core model, with 16 turns per story on average.

The *language data* used to train the model was instead generated with *Chatette*[2], a library for NLU data generation. Even if programmatically generated, the data went through multiple revisions like for the conversation data, where the interactions of the users were taken into account to expand and improve the intent templates used for data generation. In particular, the users utterances that were wrongly classified during the interactions were included in the training set. Moreover, they were also integrated in the data generation pipeline, so that multiple augmented instances of these utterances could be easily generated to include common synonyms and variations. An average of 100 examples were produced for each of the 21 supported intents, with more examples generated for the more complex intents (i.e. with more entities). Table 1 provides an overview of the supported entities and the data used to train the classifier. This approach to data generation resulted in 1385 intent samples, 782 entity samples and a domain vocabulary of 396 unique tokens.

Table 1: An overview of the supported entities and data used during training. Note that the DIET classier was trained also over the "lookup table" and "SpaCy" entities to increase the model's robustness.

entity	**example**	**count**	**classifier**
recipe	Pizza	120	DIET
ingredient	flour	147	DIET
tag	gluten-free	7	lookup table
cuisine	italian	10	lookup table
CARDINAL	three	-	SpaCy
TIME	45 minutes	-	SpaCy

A total of 7 slots were defined as memory layer, for example to store the total number of people to cook for. A frame was used to handle the *step-by-step cooking* stage, since the user can ask repeatedly for the next step until the recipe is done. This phase also supports a certain degree of *contextual digression*, which allows the user to ask for more information about the last cooking step.

During hyperparameter search different held-out fractions were tested to understand how having more or less data would influence performances. More details on this can be found in Section 5.

The domain data, i.e. the actual recipes, was provided by *justthedarnrecipe.com*[3], a free recipe website offering a simple interface. The data was scraped and then parsed to a standard format. An example of a recipe entry is given in Table 2. Additional post-processing was done to uniformize the ingredients and their amounts, add additional informative tags (e.g. dietary constraints), pair each ingredient with a potential substitute and add explicative images. A total of 65 recipes were parsed, with 147 unique ingredients, 7

[2] https://github.com/SimGus/Chatette

[3] https://justthedarnrecipe.com

dietary constraints and 10 cuisine types. Details about the preparation and cooking times are also available for each recipe. The resulting recipes have an average of 7.5 ingredients and 8.3 steps.

Table 2: An example of a recipe with its attributes.

attribute	example
title:	Pizza Margherita
image:	i.imgur.com/OaqXSMf.jpg
tags:	[vegetarian]
cuisine:	italian
prep_time:	25 (minutes)
cook_time:	10 (minutes)
servings:	2
ingredients:	- flour, 300g
	- water, 200ml
steps:	- "Mix the flour with..."
	- "Pour in the water..."

4 Conversation Model

As stated previously, the model is built using the Rasa framework. The trained pipeline for the NLU part comprises the following main components, which selected after an extensive hyperparameter search (see Section 5):

- **SpacyTokenizer:** uses a pretrained SpaCy language model to split the text into tokens. This pretrained model was trained over a large corpus of data from the web (Ralph Weischedel, 2013; Miller, 1994; Pennington et al., 2014), and can be used to perform different tasks like named entity recognition or token vectorization.

- **Featurizers:** the agent uses a *SpaCy featurizer* (a pretrained model to produce word vectors), a *regex featurizer*, a *lexical syntactic featurizer* and a *count vectors featurizer* (both at word and char level) to compute features both at token and sentence level, which can be then used by other components (e.g. for entity extraction).

- **SpacyEntityExtractor:** used only for the extraction of the "CARDINAL" (numbers expressed as words) and "TIME" (used to handle the timer intents) entities, therefore no training examples were needed since they are handled by the SpaCy's NER.

- **RegexEntityExtractor:** extract entities using the lookup tables, which were defined for the "tag" and "cuisine" entities, since they can only assume limited and well-defined values.

- **DIETClassifier:** a Dual Intent Entity Transformer (Bunk et al., 2020) model that uses a multi-task architecture to perform both intent classification and entity extraction.

- **FallbackClassifier:** a simple component that classify an intent as ambiguous if the predictions from the previous classifiers have low confidence. This was used to implement a *two-stage fallback* policy to handle the cases where a user provides an unexpected message that cannot be mapped with confidence to any known intent. If such situation occurs, the user is asked to confirm or deny the low-confident intent prediction. In case the user denies, he is asked to rephrase the message. If the new message obtains again a low confidence and the user reject the prediction, an ultimate fallback is triggered and the agent responds that it is not able to help.

Additional details about the configuration and the hyperparameters used can be found in the repository. For determining the next action to take, the agent was trained with the following response policies:

- **MemoizationPolicy:** memorizes the stories in the training data and tries to match them to the current conversation to predict the next action.

- **RulePolicy:** applies a set of predefined rules to handle small conversation patterns.

- **TEDPolicy:** a Transformer Embedding Dialogue policy (Vlasov et al., 2019), uses a multi-task architecture to perform both next action prediction and entity recognition.

- **UnexpecTEDIntentPolicy:** uses the same architecture of *TEDPolicy* to handle interactions with the users that are unlikely and could cause wrong next action predictions. In case such action is detected, the user is asked to rephrase the last message.

5 Evaluation

A total of 100 hyperparameters combinations were trained and evaluated to find the best configuration, both for NLU and dialogue management components.

For NLU, different models were tested against different tokenizers, featurizers, classifiers, embedding sizes, drop rates, n-gram sizes, fallback thresholds, learning rates and number of epochs. The training dataset was first split into a 80% train and 20% validation subsets. For each configuration, a certain amount

Table 3: The configurations for the NLU pipeline with the best performances. For each held-out percentage of the training set the f1-score over the validation set is reported.

tokenizer	featurizer	entity classifier	intent classifier	entity f1-score				intent f1-score			
				0%	50%	75%	test	0%	50%	75%	test
SpaCy (lg)	SpaCy	DIET	DIET	99.87	98.43	92.20	86.27	99.59	98.52	94.14	76.02
SpaCy (lg)	DistilBERT	DIET	DIET	99.14	97.98	91.96	84.83	98.88	97.86	93.95	74.27
SpaCy (md)	SpaCy	DIET	DIET	97.65	96.62	91.79	84.19	98.72	97.63	93.44	73.95
SpaCy (md)	SpaCy	CRF	Sklearn (SVM)	94.22	92.56	87.98	81.56	94.12	92.34	90.46	68.25
SpaCy (md)	DistilBERT	CRF	Sklearn (SVM)	93.46	91.21	86.12	80.01	93.72	91.18	89.84	67.94
white-space	DistilBERT	CRF	Sklearn (SVM)	91.78	89.02	84.58	78.87	92.17	90.96	87.95	66.12
SpaCy (sm)	SpaCy	DIET	DIET	89.40	87.52	83.10	76.26	97.72	95.83	92.73	72.19

Table 4: Ablation study over the policy components used for dialogue management.

using policies			max history	action prediction (f1-score)		successful stories (accuracy)	
Rule	TED	Memoization		train	test	train	test
✓	✓	✓	15	100.00	87.97	98.50	80.00
✓	✓	✓	10	99.58	86.28	96.00	80.00
✓	✓		15	96.32	83.85	92.50	80.00
✓	✓	✓	5	91.82	80.13	86.50	70.00
✓		✓	15	78.38	72.39	68.51	50.00
✓			-	72.56	65.23	64.00	45.00

of data was excluded from the train split to be able to understand how increasing the available training data would affect the model performances. In particular, three held-out fractions were tested for each configuration: 0, 50%, and 75%. A test split comprised of data collected from real users interacting with the agent was also used. Since training is nondeterministic, each experiment was repeated three times to account for possible random factors. In total, 900 training and testing runs were executed. Table 3 reports the results obtained w.r.t different classifiers configurations. The best configuration proved to be a SpaCy NLP model used as tokenizer and featurizer with a DIET classifier for both intent and entity classification, which slightly outperformed configurations with a DistilBERT (Sanh et al., 2019) pre-trained language model, while providing more significant improvements w.r.t. using a CRF and SVM for entity and intent classification respectively.

For the dialogue management component, different policies were tested against different max-history sizes, embedding sizes, fallback thresholds, learning rates and number of epochs, for a total of 200 runs. An ablation study was also conducted on the policy components to understand their effects on the overall performances. The results are shown in Table 4. The ability of each policy to predict correctly the next action in a story was used as metric. The table also reports the overall accuracy in correctly following the whole training and testing stories. As for the NLU evaluation, the test set was comprised of stories collected from real user-agent interactions. The results clearly show that all the three policy components are effective in improving the overall performances, even if only marginally in case of the Memoization component. Moreover, a larger history size proved to be better, meaning that the agent benefits from a larger context when performing action prediction.

Once the intrinsic evaluation was completed and the best model configuration was determined, the model was deployed on *Google Assistant* to conduct a final extrinsic evaluation using also a vocal model. Five users were asked to rate their interactions with the conversational agent on a 1-5 scale, over an average of three conversations per user, using both vocal and textual input channels. In particular, they were asked: *"How likely are you to interact the agent again?"* (3.60 on average); *"Did you feel the system understood you?"* (3.20 on average); *"How appropriate were the agent responses?"* (4.00 on average). In general the users were happy with the interactions and the way the agent responded, but the limitedness of the domain data (i.e. the amount of recipes) proved to be a major downside since the system was often not able to answer the user queries.

6 Conclusions

In this work a conversational agent for preparing cooking recipes was presented. The final model was able to provide a satisfactory user experience, according to different types of evaluation. Further improvements could be done by expanding both the domain data, with more recipes, and the conversation data, to support more features and improve the overall performances.

References

Tanja Bunk, Daksh Varshneya, Vladimir Vlasov, and Alan Nichol. 2020. Diet: Lightweight language understanding for dialogue systems.

George A. Miller. 1994. WordNet: A lexical database for English.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Mitchell Marcus Eduard Hovy Sameer Pradhan Lance Ramshaw Nianwen Xue Ann Taylor Jeff Kaufman Michelle Franchini Mohammed El-Bachouti Robert Belvin Ann Houston Ralph Weischedel, Martha Palmer. 2013. Ontonotes release 5.0.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Vladimir Vlasov, Johannes E. M. Mosig, and Alan Nichol. 2019. Dialogue transformers.