# IAI MovieBot: A Conversational Movie Recommender System

Javeria Habib
University of Stavanger
Stavanger, Norway
javeriahabib09@gmail.com

Shuo Zhang
Bloomberg
London, United Kingdom
szhang611@bloomberg.net

Krisztian Balog
University of Stavanger
Stavanger, Norway
krisztian.balog@uis.no

## ABSTRACT

Conversational recommender systems support users in accomplishing recommendation-related goals via multi-turn conversations. To better model dynamically changing user preferences and provide the community with a reusable development framework, we introduce IAI MovieBot, a conversational recommender system for movies. It features a task-specific dialogue flow, a multi-modal chat interface, and an effective way to deal with dynamically changing user preferences. The system is made available open source and is operated as a channel on Telegram.

## CCS CONCEPTS

• **Information systems → Users and interactive retrieval**; **Recommender systems**.

## KEYWORDS

Dialogue systems; conversational recommender systems, conversational information access, user preference modeling

**ACM Reference Format:**
Javeria Habib, Shuo Zhang, and Krisztian Balog. 2020. IAI MovieBot: A Conversational Movie Recommender System. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3417433

## 1 INTRODUCTION

Conversational information access is a rapidly growing field that has been gaining attention over the past years [1, 3]. A *conversational recommender system* is a task-orientated system that supports its users in accomplishing recommendation-related goals through a multi-turn conversational interaction [4].

There exist a broad range of dialogue systems, from ELIZA of the 60s [11] to modern-day social chatbots like Microsoft's XiaoIce [12]. The task of conversational recommendation, however, is characterized by a set of unique needs. The agent needs to elicit the user's preferences, before it can make recommendations. Further, these preferences may evolve or change during the course of interactions, as the user learns about the set of available items in the collection. This desired behavior is currently not well supported in conversational information access systems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*CIKM '20, October 19–23, 2020, Virtual Event, Ireland*
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3417433

While there are a number of open-domain dialogue frameworks to aid development, these are not suitable for the task at hand. OpenDial [5] is designed to facilitate the development of agents for single-turn Q&A-style dialogues. Plato [6] and PyDial [10] attempt to model user preferences, but do not track the preference evolution over conversations. Further, most of the available domain-specific (movie) recommender systems are closed-source commercial products, such as the Facebook messenger bot And chill.[1] Vote GOAT [2] is open-sourced, but its architecture comprises commercial components like Google Dialogflow, which require a paid plan.

In this work, we aim to address the above issues by developing an open-source conversational movie recommender system, which models users' preferences dynamically and supports multi-turn recommendations. Our system, termed IAI MovieBot, is based on an extensible architecture, comprising typical components, such as natural language understanding, dialogue manager, and natural language generation. IAI MovieBot features several innovative elements:

- A task-specific *dialogue flow*, along with a set of associated *intents*, to support the effective elicitation of user preferences and to provide suggestions from a large collection of items.
- A *multi-Modal chat interface* to offer multiple modes of interactions and increase the ease of use of the system by allowing users to utilize on-screen buttons.
- An effective way of dealing with dynamically changing *user preferences*, which involves dialogue intents for revealing and modifying preferences, a machine-understandable structured representation of them, and an user interface and experience design where the agent makes the understood preferences transparent as well as scrutable, with the help of dedicated buttons.

As the name suggests, we operate in the movies domain. It should be noted, however, that the framework itself (i.e., the main components and dialogue intents) is domain independent and the specific system components can reasonably easily be adapted to other domains.

To summarize, the main contribution of this work is an open-source conversational movie recommender system featuring (1) a task-specific dialogue flow, (2) a multi-modal chat interface, and (3) an effective way to deal with dynamically changing user preferences. The source code can be found at https://github.com/iai-group/moviebot and the system can be tried on the Telegram channel @IAI_MovieBot.[2] These resources are intended both for users acquiring movie recommendations and for researchers working on conversational recommender systems.

[1] http://www.andchill.io/
[2] https://t.me/IAI_MovieBot

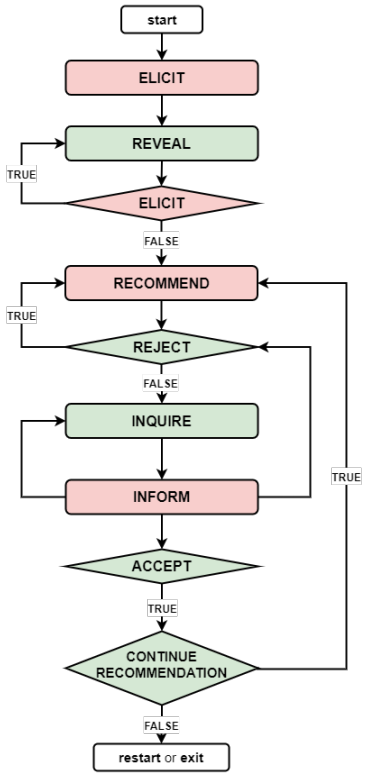


**Figure 1: Dialogue flow in IAI MovieBot. The colors *red* and *green* indicate agent's and users' intents respectively.**

## 2 MODELING AND ARCHITECTURE

In this section, we present the overview of our framework for the task of conversational item recommendation. Specifically, we discuss how we model dialogue and user preferences (Sect. 2.1) and describe the main architectural components (Sect. 2.2). This framework is domain independent and is meant to be portable to other item recommendation scenarios beyond movies.

### 2.1 Dialogue Modeling

We assume that recommendations happen via a multi-turn conversation with an agent, which is initiated and terminated by the user. The agent keeps eliciting user preferences until (a) the result set is sufficiently small or (b) it has reached the maximum amount of questions it is allowed to ask (to avoid fatiguing the user). Then, the agent makes recommendations for specific items and elicits feedback on them, until the user finds an item to their liking (or terminates the process).

User and agent utterances are modeled as *dialogue acts*, which are machine-understandable representations of the natural language text. A dialogue act is mathematically represented as

$$intent\Big((slot_1, op_1, value_1), \ldots, (slot_n, op_n, value_n)\Big)\,,$$

where operators ($op \in \{=, \neq, <, >, \leq, \geq\}$) specify the relationship between for each slot and its corresponding value. We create and define the set of possible intents based on [7] and [9]; these are listed along with examples in Table 1. Fig. 1 shows the dialogue flow in our system using these intents.

**Table 1: Overview of intents. Boldfaced intents can be further split into sub-intents (listed below the dotted line).**

| | Intent | Description |
|---|---|---|
| | **User Revealment [7] / Query Formulation [9] / Query Reformulation [9]** | |
| User | Reveal | The user wants to reveal a preference. *"Do you have any sports movies?"* |
| | Remove preference | The user wants to remove any previously stated preference. *"I don't want to see any sports movies anymore."* |
| Agent | Elicit | Ask the user to describe their preferences. *"Which genres do you prefer?"* |
| | **System Revealment [7] / Result Exploration [9]** | |
| User | Inquire | Once the agent has recommended an item, the user can ask further details about it. *"Please tell me more about this movie."* |
| | **Accept/Reject** | The user can decide if they like the recommendation or not. |
| | Accept | The user accepts (likes) the recommendation. This will determine the success of the system as being able to find a recommendation the user liked. *"I like this recommendation."* |
| | Reject | The user either has already seen/consumed the recommended item or does not like it. *"I have already seen this one."* |
| | Continue recommendation | If the user likes a recommendation, they can either restart, quit or continue the process to get a similar recommendation. *"I would like a similar recommendation."* |
| Agent | **Reveal** | Reveal the results or the number of matching results to the user. |
| | Too many results | The number of items matching the user's preferences is larger than a maximum limit. This will be followed by an *elicit* intent. *"There are almost 1100 action movies."* |
| | Recommend | Based on the user's preferences, make a recommendation. *"I would like to recommend a fairy tale film, named Shrek."* |
| | No Results | The database does not contain any items matching the user's preferences. *"Sorry. I couldn't find any romantic Korean movies."* |
| | Inform | If the user *inquires* about the recommended item, the agent provides the relevant information. *"The director of this movie is XYZ."* |
| | **Miscellaneous Intents** | |
| User | Hi | When the user initiates the conversation, they start with a formal *hi/hello* or *reveal* preferences. |
| | Acknowledge | Acknowledge the agent's question where required. |
| | Deny | Negate the agent's question where required. |
| | Bye | End the conversation by sending a bye message or an exit command. |
| Agent | Welcome | Start the conversation by giving a short introduction. |
| | Acknowledge | Acknowledge the user's query where required. |
| | Cant Help | The agent does not understand the user's query or is not able to respond properly based on the current dialogue state. |
| | Bye | End the conversation. |

The user's preferences are represented as an *information need* (IN). The user can reveal their preferences at any stage in the conversation, which will trigger an update to the IN. Information needs are represented as slot-value pairs, and get their values assigned based on *reveal* intents. For example, if a user wants a "*romance and comedy movie, starring Meryl Streep from the 90s*," the IN will be modeled as *[genres = romance, comedy; actors = Meryl Streep; release year ≥ 1990 & < 2000]*. Note that some slots can be multi-valued (this is defined by a domain-specific ontology). Further, it may be that the system attempts to elicit preference for a slot that the user does not care about. Those responses are also registered, but they will not narrow the set of matching items.

If the number of items matching the information need exceed a predefined threshold, the agent will attempt to elicit additional preferences (i.e., slot values for the IN). For example, if the user states a preference for *action* movies, the agent will follow this up with the following request: "*There are almost 4700 action films. Please answer a few more questions to help me find a good movie...*" As shown in Fig. 1, once the agent has *recommended* an item, the user has options to either *reject* or *accept* it, or *inquire* further details about the item. These responses are recorded as *dialogue context*. The dialogue context is a dictionary structure that stores all feedback associated with specific items, e.g., { *'Inception': [watched],*

*'Amsterdamned': [dont_like],'The Mountain II': [inquire, accept], ...*}. The user can receive more recommendations if they *reject* an item and are offered similar recommendations if they *accept*. They can also *restart* or *exit* a conversation. Restarting a conversation will erase the current IN as well as the history of recommended items.

## 2.2 System Architecture

The main architecture is shown in Fig. 2, illustrating the core process for each dialogue turn. The *natural language understander* (NLU) converts the natural language response from the user into a dialogue act. This process, comprising of *intent detection* and *slot filling*, is performed based on the current dialogue state. The *dialogue state tracker* (DST) in the *dialogue manager* (DM) updates the *dialogue state* (DS) and *dialogue context* (DC) based on the dialogue acts by both the agent and the user. The dialogue state includes the recent dialogue acts for both the user and the agent, the information need, the matching results with respect to the IN, the current recommendation by the agent, and the agent's state that defines its next step. The dialogue context keeps track of the items recommended to the user with their feedback (where possible values include "accepted," "rejected/don't like," and "inquire"). The dialogue policy (DP) generates a dialogue act by the agent based on the current dialogue state. It defines the flow of the conversation, i.e., what steps an agent must take at every stage. For example, the intent *elicit* is generated if the IN contains less than two user preferences. If the IN has at least one value, the intent *too many results* is generated. The parameters of the dialogue act represent what the agent must elicit, recommend, or inform. At any stage, a *reveal* or *remove preference* intent by the user will lead to *recommend* or *elicit*. For the intent *recommend*, the dialogue policy triggers the generation of an item recommendation from the collection. The output of the DP is converted to a natural language response by the *natural language generator* (NLG). The NLG also summarizes the IN back to the user, to help them keep track of their stated preferences. Further, the NLG helps the user to explore the item space by providing options.

# 3 IMPLEMENTATION

IAI MovieBot is implemented in Python as a client-server application. This section highlight some of the domain-specific aspects; for specific details, we refer to the GitHub repository.

## 3.1 Item Collection

We base our item collection on the publicly available MovieLens 25M Dataset.[3] This dataset contains 62k movies with 25M ratings and 1M tags assigned by 163k users. We use IMDbPY[4] to retrieve movie details from IMDb, including *genres*, *movie keywords*, list of *actors*, *directors*, movie *duration*, *plot (summary)*, *release year*, IMDb *rating* with *number of votes* and *links* to the movie page and its cover image on IMDb. After filtering movies that are missing essential attributes, we end up with around 40k movies, which we store in a MySQL database.

[3]https://grouplens.org/datasets/movielens/25m/
[4]https://imdbpy.github.io/

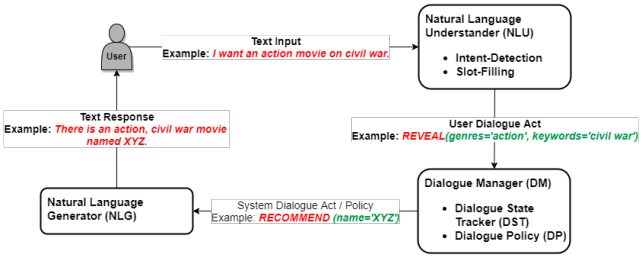


**Figure 2: IAI MovieBot system architecture.**

## 3.2 Natural Language Understanding

*3.2.1 Intent detection.* We use a pattern-based method for intent detection, where a matched intent with a high probability is considered as the output. For example, *inquire* happens mostly when the agent has recommended a movie. To check if the user is inquiring or revealing a *director name*, the response should have one of the patters *"directed," "director,"* or *"directors."*

*3.2.2 Slot filling.* A list of possible values is created for each slot using the underlying database, then matching is performed on lemmatized word forms. However, for recognizing *genres*, synonyms are also considered. For *movie keywords* and *title*, we operate on n-grams ($n = 8...1$). For each n-gram, the substring is lemmatized, and it is matched to the lemmatized slot-values. The detection of *release years* employs a set of specific patters to be able to deal with values like *"90s," "1950s," "1995,"* or *"20th century."* Annotations are further filtered to ensure that the same text span is not annotated for multiple slots. For example, given the user utterance *"I want movies on the civil war,"* the initial slot filling will yield *genres=war* and *keywords=civil war*. Therefore, the filtered results will exclude the *genres* slot as the word *war* is also present in *keywords*. Further, slot filling also detects preference statements. This is done by pattern matching for the set of words preceding the detected value. For example, *"I want action movies but not directed by Brad Pitt,"* the resulting values are *genres=action* and *director* $\neq$ *Brad Pitt*. For detecting a person name, which can be both a director or an actor in the collection, we use pattern matching to check if a preference is mentioned, e.g., *directed by* or *starring*. It is also considered that the user may prefer a movie *before* or *after* a specified time period.

## 3.3 Natural Language Generation

To avoid robotic responses, multiple templates are designed for each intent and its parameters to select a response randomly. For example, for a dialogue act *elicit(keywords)*, the response will be generated randomly from (i) *"Can you give me a few keywords?"* and (ii) *"What are you looking for in a movie? Some keywords would be good."* Moreover, for dialogue act *inform(director = Jennifer Lee)*, the response can be either (i) *"The director of this movie is Jennifer Lee."* or (ii) *"Its directed by Jennifer Lee."*

## 3.4 Dialogue Manager

The dialogue state tracker updates the dialogue state and dialogue context in three stages: (i) the DM receives the user's dialogue acts from the NLU; (ii) candidate recommendations that match the user's preferences as represented in the IN are generated from the item collection; (iii) the DP generates the agent's dialogue acts as output.

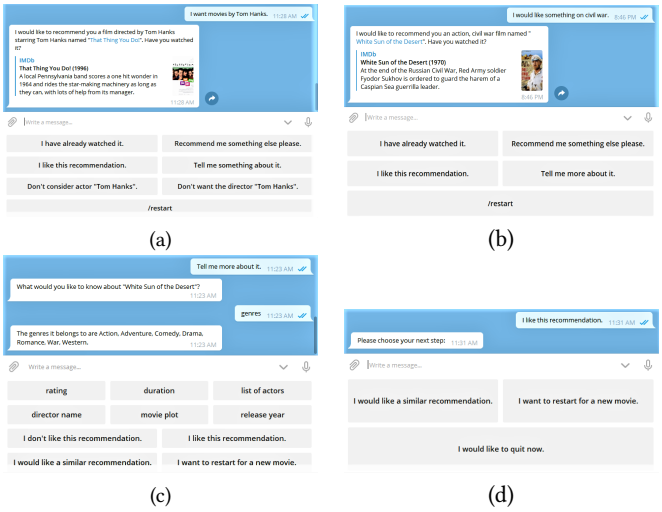


**Figure 3: Telegram keyboard buttons (a) the information need has the same value (Tom Hanks) for two slots (actors and directors), (b) a recommendation is made, and the options are accepted/reject and inquire, (c) the user has inquired about the movie genres, and remaining attributes are presented as buttons, (d) continue recommendation.**

The dialogue policy takes the conversation history into account and filters out movies that have already been recommended (i.e., those that are stored in the dialogue context).

## 3.5 Multi-modal Chat Interface

The chat interface is implemented on Telegram.[5] For the Telegram integration, the python-telegram-bot API is used.[6] The user's options, generated by the NLG, are shown as keyboard buttons in the Telegram app. The text of each button corresponds to a possible response and is linked to a specific dialogue act. The following intents are shown as buttons:

- *Remove Preference*: If two slots in the IN have the same value, it is assumed that the user intends to assign the value to one of these slots. Therefore, the buttons are shown for the user to ease the removal one of their preferences, if needed (Fig 3 (a)). This ambiguity typically occurs if a person name is detected in the utterance.
- *Accept/Reject*: The options to *accept* and *reject* a movie are presented when the agent has made a recommendation (Fig 3 (a, b)) or is informing the user about it (Fig 3 (c)). The button to *continue recommendation* is presented while the agent is informing the user about the movie (Fig 3 (c)) or the user likes the recommendation and may want to find similar items (Fig 3 (d)).
- *Inquire*: When the agent recommends a movie, the user may want to inquire further about it. The buttons for inquiring about a movie are split into two steps: (1) One button stating that the user wants to know more about the movie (Fig 3 (a, b)). (2) Buttons representing the movie attributes that can be inquired about. Each button gets be removed once the user has asked about that attribute (Fig 3 (c)).

[5]https://telegram.org/
[6]https://github.com/python-telegram-bot/python-telegram-bot

Moreover, commands to get *help*, and *start*, *restart*, and *exit* the conversation are added to the interface.

## 3.6 Feedback Collection

To help improve the system, a feedback form is created.[7] Users fill out this form anonymously, by providing only basic demographic information (age, gender, and education). Survey respondants are asked to rate the system based on quality attributes: *effectiveness*, *efficiency*, and *satisfaction* [8]. They are further invited to provide free text feedback on what they liked and disliked most about the system. The feedback we solicited during various stages of development helped us to shape and improve the system's functionality.

# 4 CONCLUSION

We have presented IAI MovieBot, an open-source conversational recommender system for movies. The system follows a task-specific dialogue flow, in which user preferences are elicited until the set of matching items is sufficiently small to make effective recommendations. The user experience has been designed to cater for dynamically changing preferences. IAI MovieBot offers a multi-model chat interface and is made available as a channel on Telegram.

According to the feedback we have received from users, IAI MovieBot has proved to be successful in understanding their preferences, helping them to grasp their options during various stages of the conversation, and ultimately recommending a good movie. Nevertheless, the current system is seen as a starting point and we envisage expanding it further. The NLU and NLG components employ simple rule/template-based solutions, which we aim to replace with more advanced (neural) methods in the future. We also wish to extend the system to other domains.

# REFERENCES

[1] Avishek Anand, Lawrence Cavedon, Hideo Joho, Mark Sanderson, and Benno Stein. 2020. Conversational Search (Dagstuhl Seminar 19461). *Dagstuhl Reports* 9, 11 (2020), 34–83.
[2] Jeffrey Dalton, Victor Ajayi, and Richard Main. 2018. Vote Goat: Conversational Movie Recommendation. In *Proc. of SIGIR '18*. 1285–1288.
[3] Jianfeng Gao, Michel Galley, and Lihong Li. 2019. Neural Approaches to Conversational AI. *Found. Trends Inf. Retr.* 13, 2-3 (2019), 127–298.
[4] Dietmar Jannach, Ahtsham Manzoor, Wanling Cai, and Li Chen. 2020. A Survey on Conversational Recommender Systems. *CoRR* abs/2004.00646 (2020).
[5] Pierre Lison and Casey Kennington. 2016. OpenDial: A Toolkit for Developing Spoken Dialogue Systems with Probabilistic Rules. In *Proc. of ACL '16*. 67–72.
[6] Alexandros Papangelis, Yi-Chia Wang, Piero Molino, and Gökhan Tür. 2019. Collaborative Multi-Agent Dialogue Model Training via Reinforcement Learning. *CoRR* abs/1907.05507 (2019).
[7] Filip Radlinski and Nick Craswell. 2017. A theoretical framework for conversational search. In *Proc. of CHIIR '17*. 117–126.
[8] Nicole M. Radziwill and Morgan C. Benton. 2017. Evaluating Quality of Chatbots and Intelligent Conversational Agents. *CoRR* abs/1704.04579 (2017).
[9] Johanne R Trippas, Damiano Spina, Lawrence Cavedon, Hideo Joho, and Mark Sanderson. 2018. Informing the design of spoken conversational search: Perspective paper. In *Proc. of CHIIR '18*. 32–41.
[10] Stefan Ultes, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gašić, and Steve Young. 2017. PyDial: A Multi-domain Statistical Dialogue System Toolkit. In *Proc. of ACL '17*. 73–78.
[11] Joseph Weizenbaum. 1966. ELIZA–A computer program for the study of natural language communication between man and machine. *Commun. ACM* 9, 1 (1966), 36–45.
[12] Li Zhou, Jianfeng Gao, Di Li, and Heung-Yeung Shum. 2018. The Design and Implementation of XiaoIce, an Empathetic Social Chatbot. *CoRR* abs/1812.08989 (2018).

[7]https://forms.gle/q4m5fwaWMZigVsaP7