University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

A Conversational Movie Recommender System

Master's Thesis in Computer Science
by

Javeria Habib

Supervisor

Krisztian Balog

Spring 2020

"It's not about where your starting point is, but your end goal and the journey that will get you there."

"Success isn't about being the best. It's about always getting better."

Abstract

The purpose of a Conversational Recommender System is to help the users achieve their recommendation specific goals using a multi-turn dialogue. In recent years, numerous studies are conducted on improving the quality attributes of a conversational recommender system. Multiple conversational movie recommender systems are proposed. However, there is a need for a conversational system for a movie recommendation, which can be used for research purposes.

The main goal of this thesis is to create Jarvis, an open-source, rule-based conversational movie recommendation system focusing on understanding the users' goals and adapting to their changing requirements. In order to understand the users' goals, a database is created, which contains the attributes with higher coverage of possible users' goals. A multi-model chat interface is designed for Jarvis. This interface introduces the components for better user interaction and providing users a guide during the conversation.

The success of a conversational system is measured in terms of the quality of the conversation and the satisfaction of the users. To guarantee the success of Jarvis, the conversation of the system with different users is recorded. Moreover, the users are requested to rate their conversation and give feedback about the system. The behavior of the system during the conversation and user feedback is studied to improve Jarvis.

The results have shown that conversational data and users' feedback plays an essential role in improving the performance of Jarvis. The users' satisfaction has improved, and the system adapts better to the previously unknown scenarios in the conversation. However, to make the system more adjustable and user-friendly, more users are required to test the system.

Acknowledgements

I would like to thanks my thesis supervisor **Prof. Krisztian Balog** for his exceptional supervision, valuable guidelines, and continuous support at every stage of the thesis. I am thankful for his trust in my work and abilities. He has helped me learn the true essence of research and its setbacks.

My deepest gratitude for my **parents** for their love and support. They have raised me as a strong, independent, and hardworking woman who is able to endure challenges. Thank you for giving me the strength to aim for the stars and embracing my failures.

This thesis could not have been possible without the support of my **husband**, who had supported me through my rough times and motivated me. Thank you for your endless efforts.

Finally, I am thankful to **Allah** for his blessings and always answering my prayers. Thanks to Him for making this attempt a fruitful one.

This thesis is the beginning of my journey, and I am proud of the outcome.

Contents

Abstract **vi**

Acknowledgements **viii**

1 Introduction **1**
1.1 Approach and Contributions . . . 3
1.2 Outline . . . 4

2 Overview of Dialogue Systems and Conversational Recommendation Systems (CRS) **5**
2.1 Dialogue Systems . . . 5
2.1.1 Goal-Driven Dialogue Systems . . . 7
2.1.2 Non-Goal-Driven Dialogue Systems (Chatbots) . . . 8
2.1.3 Dialogue Systems vs. Conversational AI . . . 11
2.2 The Dialogue-System Architecture . . . 12
2.2.1 Natural Language Understanding (NLU) . . . 12
2.2.2 Natural Language Generation (NLG) . . . 13
2.2.3 Dialogue Manager (DM) . . . 13
2.3 Conversational Recommender Systems . . . 14
2.3.1 High-Level requirements for Conversational Recommender Systems 14
2.3.2 Examples of Conversational Movie Recommender Systems . . . 16
2.3.3 Evaluation of Conversation Recommender Systems . . . 18

3 Conversational Recommender System for Movie Recommendation **21**
3.1 Main Components of Jarvis . . . 21
3.1.1 User and Agent Intents designed for Jarvis . . . 22
3.2 Jarvis Version 1 . . . 26
3.2.1 External Components in Jarvis . . . 26
3.2.2 Dialogue Manager . . . 27
3.2.3 Natural Language Understanding (NLU) . . . 35
3.2.4 Natural Language Generation (NLG) . . . 44
3.2.5 Multi-modal Chat Interface . . . 45
3.3 Jarvis Version 2 . . . 46
3.3.1 External Components in Jarvis . . . 47
3.3.2 Dialogue Manager . . . 49

3.3.3 Natural Language Understanding (NLU) . . . 50
3.3.4 Natural Language Generation (NLG) . . . 54

4 Evaluation **55**
4.1 Experimental Setup . . . 55
4.1.1 User satisfaction survey . . . 58
4.2 User satisfaction survey . . . 59
4.2.1 Feedback . . . 59
4.2.2 Analysis . . . 59
4.3 Movie Recommendation - Quality of the conversation . . . 60
4.3.1 Experimental Results . . . 60
4.3.2 An example of user-case in Jarvis . . . 69
4.3.3 Analysis . . . 72

5 Conclusion and Future Directions **73**
5.1 Conclusion . . . 73
5.2 Future directions . . . 75

Bibliography **77**

Chapter 1

Introduction

Recent technological advancements in Natural Language Processing (NLP) and Dialogue Systems lead to state-of-the-art conversational human assistants like Microsoft Cortana, Google Assistant, and Apple Siri. A dialogue system, also known as a conversational agent, is a machine that interacts with the users in their language for either a regular conversation or help the users reach their goals. Therefore, a dialogue system can be categorized as goal-driven (task-oriented conversational agents) and non-goal-driven (chatbots) systems. The goal-driven dialogue systems help the users accomplish a specific goal (for example, finding a movie, booking tickets, etc.). Their efficiency is measured in terms of how many conversational turns the system takes to reach a user-defined goal. The non-goal-driven systems are created as human companions that converse with a human on regular-life topics. Their efficiency is measured in terms of user engagement, i.e., how long the users stay in the conversation. ELIZA [1] and PARRY [2] are famous non-goal-driven rule-based dialogue systems.

The conversational recommender systems (CRS) are a type of goal-driven systems that make recommendations based on the users' goals. CRS focus on guiding the users through a natural conversation to collect their preferences [3] instead of asking them to list all at once. This process can be done via text-based conversation and speech, or both. Many end-to-end open-source conversational agents are developed for research purposes. PyDial [4] and OpenDial [5] are multi-domain frameworks for both written and spoken dialogue systems. Google Dialogflow is an online platform providing limited free functionalities. Although Dialogflow is widely used for dialogue systems, it does not provide a room for customization of the agent. It can only be used for designing and training the intents for the agent. ParlAI [6] is an open-source framework by Facebook AI Research for only speech-based dialogue systems. Plato [7] is also a multi-agent open-source framework providing a basic structure of a task-based dialogue system for developers. Vote GOAT [8] is a speech-based conversational movie recommender system built using Google Dialogflow. Vote GOAT comprises of movie recommendation, a movie rating system, and gamification for the users to maximize the ratings. However, the architecture of Vote GOAT comprises of multiple frameworks connected to perform the dialogue. However, there is a need for a low-latency, open-source, conversational movie recommender system that does not need any initial conversational data for the system's training.

The main goal of this project is to create Jarvis, an open-source, rule-based, single-domain movie recommender system. This system must be developed using an open-source platform and provide a scaleable structure for a more straight forward implementation of future amendments. As the system will be rule-based, no initial conversational data is required to train the component. However, the data collected using this system can be used to train the components for dialogue systems in the movie domain. The component of a dialogue system that converses with the users is referred to as an *agent*. Therefore, the terms agent and system have the same meaning, i.e., a system that interacts with a human. The system must have the following functionalities:

- *Entity Recognition:* Identifies an entity based on user's responses gathered over a sequential process. This functionality involves the annotation of the user's response to find a value that matches an entity or a part of it in the database. The term entity does not only refer to the title of the movie. It also includes other attributes that can be a part of the user's preferences. For example, in a response "I need a romantic comedy from the 90s", the entities recognized are "romantic", "comedy", and "90s".
- *Sentiment Analysis:* Detects the user's sentiments for the annotated entity. The system must be able to detect if the users have a positive or negative sentiment towards the entity. Moreover, if the system recommends a movie, the user's sentiment for that recommendation should also be recognized.
- *Intent Detection:* Detects the user's intents. At any stage in the conversation, the system must be able to detect the intents based on the following factors:
  - If any entities are recognized in the user's response.
  - If the agent should look for a specific intent based on the state of the conversation.
- *History Analysis:* Generates new recommendations based on the history of user's feedback on the recommendations made in the dialogue. The system must make sure not to recommend a movie twice in the same conversation. It should also try to find similar movies if the user likes a recommendation.

- *Feedback Collection:* Gathers user-feedback for further improvement. The development of Jarvis must be spanned over multiple phases. For each phase, the system must be tested for real-time users, and their feedback on the system must be used for future improvements.

1.1 Approach and Contributions

In this study, Jarvis is designed and improved in two phases. For each phase, a dialogue example is created. In the dialogue example, the agent and the user's responses are tagged with the corresponding machine-understandable translations. These examples define the goals that the system must achieve during the phase. Moreover, a dialogue-flow is designed to define the possible actions Jarvis can take at each stage in the conversation.

For the first phase, Jarvis and its components are designed from scratch. This phase includes designing a list of intents for both agent and user in a multi-turn dialogue. Moreover, for better user interaction, a multi-model chat interface is designed for Jarvis. This interface is implemented using a messaging app named Telegram[1]. Also, customized buttons at every stage are provided to guide the users during the conversation. To analyze its performance, Jarvis is used by multiple human-users. The users' conversational data with the results of each component in Jarvis is collected. This data helps in understanding the problem areas, which can be improved in later phases.

Moreover, the users are requested to rate the system for its quality attributes and overall performance. This feedback is used to target the areas which can be improved in Jarvis for the second phase. The implementation of any conversational system is an ongoing learning process. Therefore, the same evaluation process is implemented for the second version of Jarvis and will be further used for improvements in the later versions.

The following research questions (RQ) are addressed in this thesis:

RQ1 Is it possible to implement a complex dialogue-flow using only a rule-based conversational agent?

RQ2 Can a rule-based Natural Language Understander (NLU) be created to understand users' preferences for a movie?

RQ3 Is it possible to create a database that contains a high coverage of possible users' preferences?

[1] `https://telegram.org/`

RQ4 Can Jarvis meets the expectations of the users and helps them reach their goals easily?

Considering the RQs, the main contribution of this thesis are:

1. A rule-based conversation movie recommender system is created, which adapts to the user's changing requirements.
2. Using the available database, a rule-based NLU is implemented. The performance of the NLU is improved in both versions of Jarvis.
3. A database is explicitly created for recommending movies based on user's preferences. The attribute-values in the database match a higher percentage of the user's requirements.
4. The feedback of the users is improved for the second version of Jarvis. Therefore, the overall performance of the system is being improved for every next version.

1.2 Outline

The content in this thesis is organized as follows: In Chapter 2, an overview of dialogue systems and conversational AI is provided. Types of dialogue systems are explained with examples. The dialogue system architecture is also presented. Moreover, the higher-level requirements of the CRS and the evaluation criteria of CRS are explained. In Chapter 3, the main components of the proposed Jarvis movie recommender system are presented. Moreover, the developments of both versions of Jarvis is explained with algorithms and examples if required. In Chapter 4, the designed experimental setup and its features are explained. The evaluation of the system with a detailed analysis is also presented. The conclusion of the thesis with the goals for future work is elaborated in Chapter 5.

Chapter 2

Overview of Dialogue Systems and Conversational Recommendation Systems (CRS)

A dialogue system is a machine based conversational agent that interacts with the users using natural language. A dialogue system is expected to address three major problems in a real-world conversation [9]: (1) answering questions, (2) completing a task/goal, and (3) general conversation (chit-chat). A dialogue addressing all these tasks can be accomplished by using multiple conversational agents (bots) designed to fulfill particular responsibilities. In this chapter, we present different approaches to implement a dialogue system. A dialogue system is also called a conversational agent. Therefore, the terms system and agent are used interchangeably.

In Section 2.1, different types of dialogue systems are presented with the main architecture of a goal-driven dialogue system in Section 2.2. Section 2.3 lists the properties of a CRS and examples of available systems.

2.1 Dialogue Systems

A variety of end-to-end conversational AI platforms (dialogue systems) are available over the internet. Most of these are made for commercial purposes and cannot be customized for research. However, there are some platforms with open-source APIs available for further configuration and tweaking. Some of the APIs available are as follows:

- **PyDial** [4]: An Python-based platform developed and used by the Dialogue Systems Group at the University of Cambridge. It is an open-source dialogue system providing a multi-domain functionality and domain-independent implementation of both written and spoken dialogue. The Conversational Entity Dialogue Model (CEDM) in PyDial allows the users to model entities.

- **OpenDial** [5]: A Java-based platform, developed by the Language Technology Group[1] of the University of Oslo (Norway). It is also an open-source and domain-independent framework for only spoken dialogue systems.
- **Dialogflow**: A conversational platform powered by Google. It is an online platform providing a domain-independent implementation for both speech and text-based dialog. Dialogflow provides many free functionalities[2] like intent training and recognition and follow-up intents. However, some of its features, like sentiment analysis, are only available in the Enterprise edition[3].
- **ParlAI** [6]: A Python-based platform developed by Facebook AI Research (FAIR)[4] for speech based dialogues. Being an open-source framework, ParlAI provides a shared repository for researchers to add new tasks and algorithms. Moreover, integration with Amazon Mechanical Turk [5] improves the training and evolution of the system.
- **ConvLab** [10]: A Python-based conversational platform designed by the Conversational Systems Group at Microsoft. ConvLab is also an open-source, end-to-end, multi-domain and multi-intent dialogue system. Moreover, it also supports integration with Mechanical Turk for human evaluation.
- **Plato** [7]: A Python-based agent build by Uber Engineering for both beginners and experienced researchers in the field of conversational AI. It is an open-source, multi-agent platform allowing users to define their architectures.

The purpose of dialogue systems or conversational agents is to communicate with the users using natural language. This conversation can either be speech, text, or both. Dialogue systems can be further classified into goal-driven (task-oriented conversational agents) and non-goal-driven (chatbots) agents [11, 12]. Difference between both system will be further discussed in Section 2.1.1 and Section 2.1.2. Section 2.1.3 gives an overview of the AI approaches implemented in the dialogue systems.

[1] `https://www.mn.uio.no/ifi/english/research/groups/ltg/`
[2] `https://cloud.google.com/dialogflow/docs/basics`
[3] `https://cloud.google.com/dialogflow/docs/editions`
[4] `https://ai.facebook.com/`
[5] `https://www.mturk.com/`

2.1.1 Goal-Driven Dialogue Systems

Task-oriented dialogue systems converse with the users and help them to complete a specific task/goal or find an answer, e.g., checking the schedule, booking a flight/restaurant, ordering an item, etc. Digital assistants like Alexa, Siri, Cortana, etc. provide excellent examples of such dialogue agents. Goal-driven dialogue systems are mostly implemented by adding a database module to the system where it can access the data to fulfill a task [9, 13]. Initially, the systems were based on hand-crafted rules [14] (rule-based systems), which were later improved using machine learning algorithms to understand users' intents [15]. Goal-driven conversational agents are widely used in industries where 24/7 assistance is required to guide the customers. Moreover, a large dialogue corpus helped researchers build more efficient agents.

Proposed in 1977 by Bobrow et al., the GUS [16] architecture is a simple yet popular frame-based dialogue system. It is an underlying framework for many modern conversational assistants. The task-based dialogue systems (either GUS or any modern architectures) are based on a structure called **frames**, which defines a set of intents and information the system can infer from the user. Frames comprise of **slots**, where each slot has a set of possible values. This set of possible values for each slot is defined in the **domain ontology**.

2.1.1.I Main structure of a frame-based dialogue system

The information slots required by the system specify the agent's query requirements to fulfill the users' goals. The flow of the dialogue systems is based on the slots of the frame. The system fills these slots by inferring values from the user's responses and performs an action (recommend a movie, book a flight, or find a restaurant). The goals can be voluntarily provided by the users, or the system can ask specific template based questions to fill the slots.

The first and most crucial component in frame-based systems is using Natural Language Understanding (NLU) to identify the domain of the goal and intents of the users. For example, if the users want to book a flight or search for a movie. This domain selection will help the agent to decide the slots and values for the conversation. Although this step would not be required for domain-specific agents, it is a vital step for multi-domain systems like Alexa, Siri, and Cortana, etc. Rule-based NLUs use a hand-crafted semantic grammar to understand the user intents and slot-filling. However, most commercial conversational agents do intent-recognition and slot-filling using machine learning algorithms.

2.1.2 Non-Goal-Driven Dialogue Systems (Chatbots)

Non-goal-driven systems are created as AI companions to humans and are not directed towards completing a particular task. These systems mimic human behavior and are designed to have a prolonged conversation with users. For a user's query/dialogue, such systems generate a response based on statistical models, data-driven learning models [17], or both. To imitate a style of natural conversation, the response-generation process is created by training deep neural network models on large datasets comprising a human-human conversation [18–20].

Non-goal-driven conversational agents can be integrated with task-oriented systems to make them appear more natural (human-like). Such systems are mostly used for users' entertainment and also have some practical application. Some famous examples of entertainment based conversational systems are ELIZA [1, 21], Cleverbot [22] and XiaoIce [23] by Microsoft. ELIZA is designed to simulate the Regorian psychology and is used for counseling. Cleverbot is an IR-based chatbot that learns how humans converse from the internet. XiaoIce is characterized as an AI companion designed to understand human feelings and interact with humans to provide emotional support. XiaoIce uses Markov Decision Processes to optimize the decision-making process during the conversation. Its performance is measured in terms of the conversation turns per session (CPS), i.e., the length of the human-machine interaction. The average CPS of XiaoIce is 23, which is higher than a normal human-human conversation.

Non-goal-driven conversational agents are generally divided into two major categories, rule-based and corpus-based systems [11]. In this section, different types of rule-based systems are discussed in detail. Moreover, the techniques to generate a response in corpus-based systems are also discussed.

2.1.2.I Rule-based systems

As apparent from the name, rule-based systems use hand-crafted rules to geenrate a response. These rules can be based on the last user utterance or the history of the dialogue.

a) ELIZA [1, 21]: ELIZA is a very influential chatbot in the field of non-goal-driven dialogue systems. It was designed to reproduce the methods of a Rogerian psychologist where you guide the patient by reflecting back a patient's declaration. Weizenbaum states that, in Rygorian psycology, you can "assume the pose of knowing almost nothing of the real world". For example, if patient says "You don't argue with me", the psychiatrist will reply "What made you think I don't argue with you". The rules ELIZA works by are *pattern/transform* i.e., based on a pattern inferred from user's statement, ELIZA generates a response which transforms the statement to agent response. For example,

```
pattern: (* YOU * ME)
transform: (WHAT MAKES YOU THINK I 3 YOU)
here 3 refers to the position of second * in the pattern.
```

This pattern/transform rule in ELIZA applies to the words linked to the keywords in the user's response. Keywords are ranked with more common words being ranked lower than the rare ones. For example, for a user's statement,

```
I know people always think they are better.
```

The algorithm can match to a rule with keyword "I"

```
Pattern: (I *)
Transform: (YOU SAY YOU *)
```

generating a response:

```
You say you know people always think they are better.
```

However, "I" is a more common word as compared to the word "people", which can generate a more productive response leading to a conversation goal user has in mind. Therefore ELIZA would generate a response by ranking the word "people" higher than "I" in the lexicon and generate a response:

```
Who in particular are you referring to?
```

If ELIZA is not able to match a keyword form the statement, it chooses a response to carry on the conversation like "Please go on" or "I see". ELIZA can also use its memory when it is not able to generate a response. Whenever the word "my" has the highest rank in the user's statement, it saves a transform in the memory. Some examples of such transforms are:

```
(* MY *) -> (YOU SAID YOUR 3)
(* MY *) -> (DO YOU KNOW WHY YOUR 3)
(* MY *) -> (DOES THIS LINK WITH THE FACT THAT YOUR 3)
```

Later, where no keywords are found, ELIZA will respond using the latest transformed response stored in the memory.

b) PARRY [2]: Another chatbot focused on clinical psychology was proposed by Colby et al. [2] to study schizophrenia. As the next step to ELIZA, PARRY also added another parameter that accounts for its mental condition, which can affect the agent's level of fear or anger. PARRY was successfully able to imitate real paranoids. Therefore, the psychiatrists could not differentiate between the conversations with PARRY and the real patients [24].

2.1.2.II Corpus-based systems

CCorpus-based chatbots use corpus data to learn human behavior from human-human or human-machine conversations. Such systems need a large amount of conversational data to train the models [25]. Many large datasets have been used for both text and spoken language. In order to collect spoken dialogue data, telephonic conversations [26] and also movie dialogues [27, 28] are used to generate large amount of data. For text-based corpora, sites like Twitter [29] or databases of crowd-sourced conversations [30] have been used. For topical chatbots, corpora created are limited to the relevant topics. Moreover, further data can be collected once the chatbot is implemented and can be used to improve the trained model further.

The two main techniques used to generate responses in corpus-based chatbots are information retrieval (IR) and machine-learned sequence generators. Corpus-based chatbots do not model the dialogue state and generate a response based on the last user's response.

a) IR-based chatbots: Information retrieval based chatbots generate a response to a user's response by selecting a suitable entry from the corpus containing a natural-language conversation. For this purpose, the IR-based agents use a retrieval method. Some examples [11], of retrieval methods, are listed as follows:

- Return the most similar turn: For a user utterance q, return the turn t from corpus C with the most similarity to q.

$$r = \underset{t \in C}{\operatorname{argmax}} \frac{q^T t}{\parallel q \parallel \parallel t \parallel}$$

- Return the response to the most similar turn [31, 32]: For a user utterance q, find the turn t_{-1} from corpus C that has the most similarity to q and return the following turn t_0.

$$r = response(\underset{t \in C}{\operatorname{argmax}} \frac{q^T t}{\parallel q \parallel \parallel t \parallel})$$

b) Machine-learned response generators: These response generators are a machine learning version of ELIZA as they transform the user's utterance into a system's utterance. The system uses the corpus to learn a translated response to the user [11].

As the primary goal of this thesis is to create a goal-driven domain-specific conversational agent, the main focus of the remaining sections would exclude non-goal-driven systems.

2.1.3 Dialogue Systems vs. Conversational AI

The terms "Conversational AI" and "Dialogue Systems" often mean the same thing in the literature. However, conversational AI can be regarded as a sub-component of dialogue systems. The latter also refers to rule-based conversational agents, and the former only covers the topic of AI-based agents.

The different types of conversational agents mentioned in Section 2.1.1 and Section 2.1.2 can be combined to create a unified conversational agent. This method involves a *decision-making* process comprising of a higher-level action of selecting the relevant agents (task-oriented or casual chat) and agent-specific lower-level actions. Such decision-making processes can be created by implementing Markov Decision Processes (MDP) [33] using a mathematical framework of possible actions/decisions. The agent in an MDP framework can learn actions using Reinforcement Learning (RL). At each turn, the agent interacts with the user and generates an action based on the current state and the dialogue policy. These actions have a reward associated with them, and RL is used to learn the policies in order to maximize the net reward of the conversation. The higher net reward for a goal-driven system indicates that it took lesser Conversation-turns Per Session (CPS) for task-completion. On the contrary, the success of non-goal-driven systems is measured in terms of user engagements, i.e., more CPS.

2.1.3.I Machine Learning in Dialogue Systems

Although RL provides a complete Machine Learning (ML) framework to generate actions, it required more user interaction for the agent to train [9]. This approach can take more time (needs more conversations with the users) until RL generates appropriate dialogue acts. Therefore, another approach is to use different ML methods, like supervised learning (SL), and using data from human-human or human side of human-machine conversations. This approach can be further extended to RL once the agent is implemented. However, SL also needs a large corpus to train the models.

2.1.3.II Deep Leaning in Dialogue Systems

Deep Learning is the training of neural networks that initially consisted of a single layer [34] and advanced towards using multiple hidden layers between learning the output from the input [35, 36]. This architecture of multiple layers is called a deep neural network (DNN). The difference between the traditional (feature-based) ML methods and DNNs is that the later does not require a set of features extracted from the input [9].

2.2 The Dialogue-System Architecture

In this section, the main architecture of frame-based dialogue systems is presented. The three main components in a dialogue system architecture (Figure 2.1) are Natural Language Understander (NLU), Natural Language Generator (NLG), and the Dialogue Manager (DM).

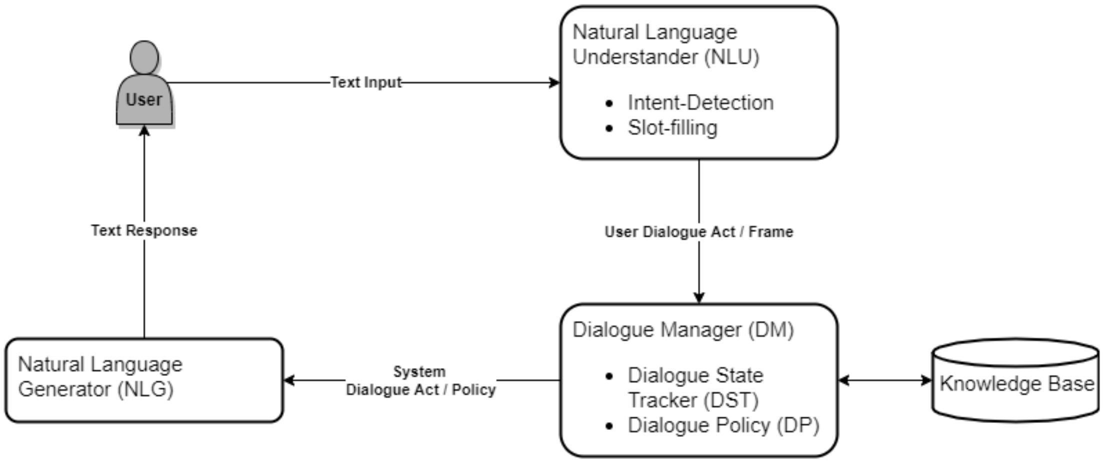


Figure 2.1: Dialogue State Architecture of Goal-Driven dialogue systems

2.2.1 Natural Language Understanding (NLU)

Two major tasks of NLU in dialogue systems are domain recognition and intent classification from the user statement. The goal of domain recognition is to identify the significant scope of the goals users want to achieve. As mentioned in Section 2.1.1.I, this task is only required for multi-domain systems. For a specific domain, intent classification defines the intent of the users. This task includes filling any slots in the frame-based systems if the users want to reveal any requirements.

Rule-based NLU works on the principle of hand-crafted patterns and annotators to detects intents and slot-values from the user's response. For an ML-based NLU, Haffner et al. [37] proposed support vector machines (SVM) and maximum entropy was proposed for NLU by Chelba et al. [38]. Later hidden Markov model (HMM) and conditional random fields (CRF) were used for slot filling [39–41].

Neural models are used for intent and domain recognition tasks [42–44], while RNN was proposed for intent classification in 2015 [45]. For the task of slot-filling, deep learning is used for generating features as slots from the utterance [46], and later RNNs are used to detect slot-values pairs from the sequence of tokens in the utterance [47, 48].

2.2.2 Natural Language Generation (NLG)

The task of NLG is to present the agent's response in a human-understandable manner. The NLG receives the agent's intents as input and generates a natural language sentence as an output. Generating a natural language using ML models trained on a corpus is better than the hand-crafted templates because it benefits from the natural language in the data, and more effort is not required in creating templates for unknown agent actions [49].

2.2.3 Dialogue Manager (DM)

Conversations are mainly viewed as a decision-making process [50], and this process is accomplished using the DM in the conversational agents. DM comprises of two essential components:

- **Dialogue State Tracker (DST):** It keeps a record of the current state of the dialogue. This record comprises the current users' and agent's responses, the users' current preferences, and the results presented to them.
- **Dialogue Policy:** This component controls the decision-making role of the dialogue manager. The performance of the dialogue policy is dependant on the current state of the dialogue.

For dialogue management, RL was applied [51, 52] with an assumption that the NLU detects the intents accurately. However, in real-life, the NLU can not be perfect considering the randomness in users' utterances. For this uncertainty created by noisy utterances, a partially observable Markov decision process (POMDP) has shown promising results [53, 54]. The dialogue policy generated by POMDP is trained further using RL [33], giving a feedback score of the policy's performance.

2.3 Conversational Recommender Systems

A Conversational Recommender System (CRS) is a task-oriented system that supports its users in accomplishing recommendation-related goals through a multi-turn conversational interaction [55]. Although research in conversational recommender systems results in many state-of-the-art techniques, it is not always recommended in all search situations [56]. For example, in a typical search task of booking tickets: the agent will sequentially ask users questions. These questions are simple and not based on the context of the conversation. However, from a user's perspective, it would be easier to input the information on a screen where all slots are mentioned in a parallel fashion. Therefore, it would not be efficient to use a dialogue-based recommender system for such a simple case.

However, CRSs are proved to be more effective for more complex recommendations [57] with information overload. For example, planning a trip where multiple agents with different goals are required, or recommend a book/movie where the agent queries are mostly relevant to the system's current context.

2.3.1 High-Level requirements for Conversational Recommender Systems

A primary goal of the conversational agents is to make the communication between a system and human as natural as possible. Liu et al. [56] listed some challenges a recommendation system can encounter while conversing with human users:

1. *Determine the keywords in the dialogue*: This includes filtering out the unnecessary words in the user's utterance.

2. *Determine an appropriate agent response*: This depends on the agent's decision-making policy and should depend on the context of the conversation with the users. An agent response can include *results* or a *query* to understand users requirements.

3. *Answer aggregation*: Generating a list of results can be overwhelming for the users. Therefore, an agent must have the ability to summarize the options in a presentable manner.

4. *Conversation management*: Includes keeping track of the user's preferences, analyzing the context/history, and updating the state of the conversation. This also helps the agent in the decision-making process.

   At some points in the conversation, it can happen that the user's response is not at all relevant to the search goals. Therefore, it is a critical decision not to trigger any search task for such utterances.

5. *Knowledge/Information*: Having knowledge about the domain and possible aspects of it. One of the most significant challenges a conversation agent faces is keeping track of the domain concept and external world linked to it.

6. *Human Nature*: Expected to respond/interact with a human, in the same manner, as humans interact with each other. Moreover, an agent is expected to cater to the emotional needs of the users whenever possible.

Radlinski and Craswell [58] studied human-human conversations and derived a set of characteristics for a conversation IR system. They provided a theoretical framework listing five properties that make a CRS.

- *User revealment*: Users can disclose their requirements to the agent
- *Agent revealment*: Agent can reveal what it understands and further actions users can take at a specific point in the conversation
- *Mixed initiative*: During the conversation, both user and agent can direct the route of the conversation
- *Memory*: Agent keeps track of the context of the conversation and stores the user's requirements
- *Set retreival*: Agent must be able to present and manipulate multiple options considering the current user's requirements

Trippas et al. [59] listed three stages of search process *Query Formulation*, *Search Result Exploration*, and *Query Reformulation*. They mentioned that at each stage, the agent needs to inquire the users about their requirements, gather feedback from the users on the results presented and reformulate the query based on that feedback.

The purpose of a CRS is to find the users' preferences via a natural language interaction/-dialogue with the users [58]. An important action for such dialogue systems is *preference elicitation* [60], where an agent learns about the users' preferences by a series of questions (elicitation). During elicitation, an agent can also ask the users for *clarification* [60] when required. Moreover, for the users to easily understand the search space, an agent must be able to list or summarize the *recommendation* [58, 61, 62]. The second step is of *information presentation* to the users. It comprises informing the users about the details of the recommendation(s) made. It is a significant factor in predicting the duration of the dialogue [63]. Therefore, a more extended conversation can have a negative impact on the users' experience [64]. As a third element, a conversational process must have a *memory* [58], i.e., able to remember the previous conversation. This aspect will allow the agent to ask questions sequentially and store the user's preferences whenever revealed. Also, it will help the users refer back to their previous statements to indicate what is important and what must not be remembered by the agent for future recommendations.

Considering the main goal of a conversational system, Azzopardi et al. [65] proposed a framework of intents for both the user and the agent. This framework explains how these actions enable the agent to help users with the completion of a task by understanding their requirements. The list of intents with their description and examples are shown in Table 2.1 and Table 2.2 for both user and agent respectively.

2.3.2 Examples of Conversational Movie Recommender Systems

Vote GOAT [8] is a speech-based movie recommender system based on Google Dialogflow. Voat GOAT recommends movies based on the users' ratings and uses gamification as an incentive to receive votes for movies from the users. Each component in Voat Goat architecture is developed using a different framework. The intent-detection is performed using the Google Assistant Server (Dialogflow). A firebase cloud function is used to connect Dialogflow to back-end computation, which uses the HUG REST API for computation.

F. Narducci et al. [66] proposed a domain-independent movie recommender system[6] co-located with RecSys in 2018. It is a recommender system with multiple interaction modules, such as buttons and natural language responses. The general conversation comprises asking the users to rate a few movies. Based on the users' ratings, it can recommend a movie to the user with some details like IMDb rating, director, actors, and genres.

Another famous commercial movie recommender system is **And chill**[7], a Facebook messenger bot. It asks the user its favorite movie and the reason. Based on the user answers, it recommends three most-relevant movies and also can schedule a reminder for the users to watch the movies later.

Papangelis et al. [67] proposed a multi-agent dialogue model training using RL. The purpose of the framework was to model an agent-agent conversation to train multiple agents in parallel. Its basic modeling is based on training a restaurant recommender agent using reinforcement learning.

The proposed system, Jarvis, is an open-source, single-agent, rule-based system developed using Python, an open-source platform. The dialogues in Jarvis are text-based. It

[6] @MovieRecSysBot on Telegram
[7] http://www.andchill.io/

Table 2.1: An Overview of User Actions by Azzopardi et al. [65]
Examples are copied from [65]

Intent	**Description with examples in [65]**
	Reveal
Disclose	Users will disclose details regarding their preferences. "*I would like to arrange a holiday to Italy on 4th of May*"
Non-Disclose	Users chooses not to disclose the information asked by the agent. "*I prefer not to say.*"
Revise	Change the information need already disclosed. "*Actually, we need to go on the 3rd of May.*"
Refine	Refine the preference by adding more details. "*We need to go in the evening.*"
Expand	Opposite of refine. Expand the search space by removing a criteria. "*Can you also check to see what kinds of holidays are available in Spain*"
	Inquire
List	List of different options "*Tell me about all the different things you can do in Tuscany?*"
Summarize	Summary of different options "*Can you give me an overview of the things to do there?*"
Compare	Comparison between options "*What are the main differences between Tuscany and Galicia?*"
Subset	Selection of different options "*What is the best thing to do in Tuscany?*"
Similar	Option similar to the current available options "*What other regions in Europe are like that?*"
	Navigate
Repeat	Revisit the options revealed
Back	Go back to the previous recommendation (this refers to the latest one)
More	Learn about more options in the list
Note	Save an option to revisit it later "*Save that hotel for later.*"
	Interrogate
Understand	Checks if agent understands the users' preferences "*What do you think I am looking for?*"
Explain	Ask for an explanation of why a particular suggestion is made. "*Why are you showing me this?*"

recommends a movie based on the users' requirements rather than their ratings for other movies. Jarvis also uses a multi-model interface where the users can interact via natural language as well as buttons whenever required.

Table 2.2: An Overview of Agent Actions by Azzopardi et al. [65]
Examples are copied from [65]

Intent	Description with examples in [65]
Inquire	
Elicit	Ask the users regarding various constraints to narrow down the list of options
Extract	Extract the users' preferences during the conversation
Clarify	Agent want to check the criteria it extracted or specify use preference "*What do you mean by cold? Less than 20 degrees Celsius?*"
Reveal	
List	Show all possible options in the search space "*I've found a number of possible tours around the wine yards. One leaves at 8.30am for 100 pounds, another is at 1.30pm for 75 pounds, and the last one is at 4pm for 139 pounds.*"
Summarize	Give users an overview/summary of the possible recommendations "*Tours range from 75 to 139 pounds, and leave in the morning, afternoon and early evening.*"
Compare	Comparison of various options in the list "*The cheapest tour is for 75 pounds and leaves at 1.30pm, while the evening tour leaves at 4pm and includes a three course dinner, but is more expensive at 139 pounds.*"
Subset	Present some of the recommendation from the list of options.
Similar	Find similar option given the current list of objects agent has in the search space.
Traverse	
Repeat	Revisit the options revealed
Back	Go back to the previous recommendation (this refers to the latest one)
More	Learn about more options in the list
Record	Save an option if users show interest and it can be used later
Suggest	
Recommend	Agent can be asked or will be able to make recommendation based on received users' preferences
Hypothesize	Based on current information, agent can generate other possible preferences "*What if the user wants to go to a different country.*"
Explain	
Report	Report its understanding of the users' needs
Reason	Give reason for why it made specific recommendations

2.3.3 Evaluation of Conversation Recommender Systems

Radziwill et al. [68] review the quality assessment approach to evaluate the conversational agents' performance. They grouped the quality attributes of conversational agents from multiple studies. They concluded that these attributes align with the ISO 9241 concept of usability: *"The **effectiveness**, **efficiency** and **satisfaction** with which specified users achieve specified goals in particular environments."* [69].

Effectiveness defines how the agent understands the users' goals, executes the task, and responds to the users understandably. The *efficiency* describes the performance of the system. This includes how the system adapts to the changes and responds to an unexpected occurrence in the conversation. This indicates how the system has used its components to help the users achieve the goal quickly. The element *satisfaction* is measured in terms of how at-ease the users feel in the conversation. This quality depends on how the agent greets the users, responds to the users' sentiments, and gives the users cues to continue the conversation.

Chapter 3

Conversational Recommender System for Movie Recommendation

In this chapter, the main approach of implementing Jarvis - a conversational movie recommender system is presented. The implementation of Jarvis is spanned over two phases having a version of their own. The first phase comprises of the development of the basic components from scratch. In this phase, Jarvis handles a straight forward flow of the conversation with the main focus on understanding users' intents *Reveal* and *Inquire* (Table 2.1). The system is then used by human users to record the following components:

- The performance of the system. For every user's utterance, the dialogue manager updates the dialogue state and generates a response using the dialogue policy.
- The feedback of the users. Users are asked to leave some comments about how this system can be improved.

These components are crucial for better user experience and are observed further to make improvements in the second phase. In this section, the main components for Jarvis with the detailed implementation of each component are explained. The improvements in Jarvis with each phase are also elaborated.

3.1 Main Components of Jarvis

The main elements responsible for the multi-turn conversational recommendation in Jarvis are shown in Figure 3.1.

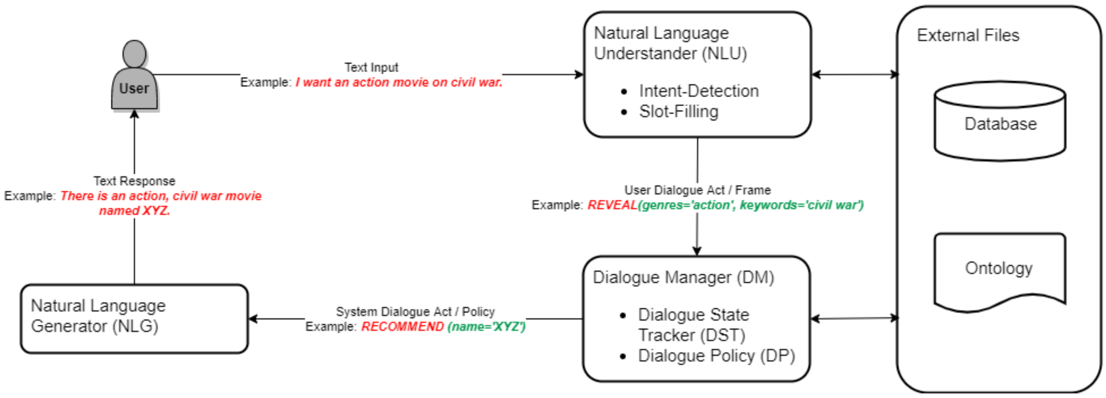


Figure 3.1: Dialogue System Architecture of Jarvis Movie Recommender System

The conversation is initiated by the user, and the component responsible for converting the natural language utterance from the user's dialogue to a machine-understandable dialogue is **Natural Language Understander (NLU)**. The NLU generates a ***Dialogue Act*** which comprises of the **user's *intent*** and and its ***parameters***. The parameters depend on the value of the intent detected and will be described further in this section.

The second component in the process of generating agent's response is **Dialogue Manager (DM)**. DM has two components 1) **Dialogue State Tracker (DST)** and 2) the **Dialogue Policy (DP)**. The DST is responsible for tracking and updating the current ***Dialogue State (DS)*** of the conversation. Based of the updated DS, the DP generates an agent's response in the form of a *Dialogue Act* comprising of the **agent's *intent*** and its ***parameters***.

As mentioned in the Section 2.1.1, task-oriented systems use an external **Knowledge Base** to complete a task. This knowledgebase can consist of a ***database*** or an external ***web-hook*** to Wikipedia, DBpedia, etc. For Jarvis, the external data used for movie recommendation is a *MYSQL database*. Another external file is the Ontology, which defines how each attribute must be accessed by the NLU and the DM. The main attributes of the database table and the ontology will be explained separately for both phases.

The Dialogue Acts generated by the DM are then converted to a human-understandable natural-language response by the **Natural Language Generator (NLG)**.

3.1.1 User and Agent Intents designed for Jarvis

The design of intents in Jarvis is inspired by Azzopardi et al. [65]. As mentioned in the examples of intents in Table 2.1 and Table 2.2, the framework explained is implemented based on the task of planning a holiday. This task has complex dependencies between the recommendable items, which is therefore very different from the task of recommending a movie. For example,

- The user's intents under the category *Inquire* are based on further exploration of the things/activities one can relate to the item selected. While planning a holiday, a user can further ask for sub-components like restaurants, hotels, and attractions near each other. Therefore, these intents can lead to a different domain. However, for movie recommendation in Jarvis, the result exploration is limited to asking further details about the items presented by the agent.
- During the phase of *user revealment*, the slots in Jarvis can be multi-valued, the goal of intents revise, refine, and expand can be fulfilled by a single *disclose* intent. A user's intent also comprises of the parameters extracted from the user's utterance. Therefore, an extra intent of extract (Table 2.2) will not be necessary.

Although the user and agent actions in Jarvis would be inspired by Azzopardi et al. [65], a different set of intents at some stages would be required in the conversation to incorporate the basic structure of movie recommendation and further explanation of the recommended items in Jarvis. The list of intents for both user and agent with their description and examples are mentioned in Table 3.1 and 3.2 respectively.

3.1.1.1 Examples of User Intents in Jarvis and their categorization

For the implementation of Jarvis, a list of examples of possible users' queries was generated. These examples are further divided into five categories.

1. Users *reveal the main keywords* the agent must look for in the database.
   - Recommend me some good Netflix movies.
   - I want a thriller from the 80s.
   - I want a comedy movie from 2019.
   - Can you recommend some horror movies?
   - Can you suggest an action movie.
2. Users *inquire* about the movie recommended.
   - Tell me more about this movie.
   - Who directed this movie?
   - What's the storyline of this movie?

Table 3.1: An Overview of User Actions in Jarvis
The bold text defines the intents which can be further split into a list of sub-intents (below the dotted line)

Intent	**Description**
User Revealment [58]/**Query Formulation** [59]/**Query Reformulation** [59]	
Reveal	Whenever the users want to reveal any information need "*Do you have any sports movies?* or *Can you also show me any recent fairy tale movie after year.*"
Remove Preference	The users want to undo any requirements they have revealed previously "*I won't prefer any sports movies anymore.?* or *Don't want it to be after year 2000.*"
System Revealment [58]/**Result Exploration** [59]	
Inquire	Once the agent has revealed a result, the users can ask further details about the movie. "*Please tell me more about this one.* or *When was this movie released?*"
Accept/Reject	Users decide if they like the movie recommendation or not
Accept	The users can accept the recommendation. This will determine the success of the system as finding a relevant recommendation. "*I like this recommendation.*"
Reject	Based on the agent recommendation, users may have either watched the movie or does not like it. "*I have already seen this one.* or *Recommend me something else please.*"
Continue Recommendation	If the users hve liked the recommendation, they can either restart, quit or continue the process and get another similar movie. "*I would like a similar recommendation.*"
Miscellaneous Intents	
Hi	While users initiate the conversation, they usually start with a formal *hi/hello*.
Acknowledge	Acknowledge the agent question where required.
Deny	Negate the agent question where required.
Bye	End the conversation by saying a bye message or exit command.

- What's the rating?
- When was it released?

3. Users *initiate* for the next recommendation.

- I like this recommendation.
- I would like something similar to this one.
- I have already watched this movie.
- I don't like this one. Do you have something else?
- I don't want a comedy movie anymore.

Table 3.2: An Overview of Agent Actions in Jarvis
The bold text defines the intents which can be further split into a list of sub-intents (below the dotted line)

Intent	**Description**
User Revealment [58]/Query Formulation [59]	
Elicit	Ask the user to describe thiei information needs. "*Which genres do you prefer?*" or "*Can you be more specific to help me narrow down the space? Give me some key words.*"
System Revealment [58]/Result Exploration [59]	
Reveal	Reveal results of a count of the results in possible search space to the user
Count Results	If the list of movies matching user requirement is longer than a maximum limit. This will be followed by an Elicit intent. "*There are almost 1100 action movies.*"
Recommend	Based on information needs revealed by the user, make a recommendation. "*I would like to recommend a fairy tale film named Shrek.*"
No Results	For particular query formulated during elicitation, there can be a chance that the query results in no movies from the database. "*Sorry. I couldn't find any romance Korean movies.*"
Inform	Once the agent has revealed a result, it should be able to provide further information about the movie. "*The director of the recommended movie is XYZ.*" or "*It was released in 1992.*"
Miscellaneous Intents	
Welcome	Start the conversation by giving a short introduction
Acknowledge	Acknowledge the user query where required.
Cant Help	If the agent does not understand user query or is not able to respond properly based on its state in the dialogue.
Bye	End the conversation.

- I want to restart this search.

4. Users also *reveal their dislikes* while searching for a movie.

   - I want a comedy movie but not by Jim Carry.
   - I don't like horror movies.
   - I don't want old movies.
   - Give me an action movie but not romantic.
   - I want an animated film but it shouldn't be a fairy tale.

5. Users wants to *navigate* between the current recommendations. *(Some of these examples are for the scenario where agent recommends multiple options)*

   - Go back to the previous movie.

- Tell me about the first recommendation.
- Who directed the second movie?

3.2 Jarvis Version 1

For the first phase of Jarvis development, the basic structure and its components are developed from scratch. Therefore, in this section, the outline of the algorithm and its motivation will be elaborated.

The Jarvis conversational agent starts the conversation if initiated by the users. The *controller* in Jarvis gathers the users' input, controls the conversational agents, and shows agent output to the users. The structure of the controller is shown in Algorithm 3.1.

Algorithm 3.1 Main *Controller* of Jarvis

Load *paths* and *configurations* from external configuration file
Initialize the conversational agent and its components
Wait for user's input
if user says something **then**
  agent starts dialogue
  while user does not exit **do**
    agent continues the conversation
  end while
  agent ends the conversation
end if

3.2.1 External Components in Jarvis

The category "external components" is assigned to the files in Jarvis, which are not implemented as part of the algorithm but have a vital role in decision-making and intent-derivation.

3.2.1.I Database

MYSQL Database is used an an external knowledgebase to fetch movies based on the agent preferences. For creating the database in version 1, a metadata of IMDb movies is generated form an existing GitHub project[1].The main attributes of database used for this version are *movie title*, *genres*, *plot keywords*, *director*, *actors* (three name), *movie duration*, *release year*, *IMDb rating* and *IMDb link*. For a single movie, multiple genres, plot keywords and actors were available in this pre-existing data.

[1] https://github.com/nitishghosal/IMDB-Data-Analysis/blob/master/movie_metadata.csv

3.2.1.II Ontology

The ontology of the system defines how the database attributes (slots) can be accessed during a conversation. These parameters help with the intent-recognition, and development and updating of dialogue-state for every turn in the dialogue. The ontology comprises of the following components:

- **agent's requestable**: The slots agent must fill before formulating a query to make the recommendation. The agent must *elicit* the users about these intents.
- **user's requestable**: The attributes agent should *inform* the users about. These are the slots which should be detected for the user's intent *inquire*.
- **slots not required by NLU**: A user can voluntarily *reveal* their preferences. While annotating user's utterances, an agent must not check for the slots mentioned in this category.
- **multiple values slots**: While storing the user's information needs, every slots is designated to have one value. This will be the latest value revealed by the user for this slot. However, the slots mentioned in this category can have multiple values. This agent will formulate a query by search for all the values in the list and will remove a value only if user want to *remove a preference*.

  For example, for the *genres* slot, a user can search for *action*, *comedy* and *romance*. The agent will try to find a movie that fits these three genres. If the user wants to remove the preference of an action movie, the agent will then search for a comedy and romance film.

The above-mentioned components in the ontology are assigned the following slots:

- **agent's requestable**: *genres*, *plot keywords*
- **user's requestable**: *genres*, *director*, *actors*, *movie duration*, *release year*, *IMDb rating*
- **slots not required by NLU**: *movie duration*, *IMDb rating*, *IMDb link*
- **multiple values slots**: *genres*

3.2.2 Dialogue Manager

As described in Section 3.1, the Dialogue Manager receives an input from the NLU and generates an output for the NLG. Therefore, DM and its components define the flow of the conversation, i.e., what steps an agent must take at every stage. Before further developing the conversational agent and its components, a dialogue-flow chart is designed, as presented in Figure 3.2.

For the implementation of Jarvis in this thesis, the agent is expected to give one recommendation at a time (no set retrieval). The route of the dialogue is decided based on the intents extracted by the NLU, and the agent intents decided based on the updated DST.

3.2.2.I Dialogue Act

The input received by the DM and the output generated are of the same format. This format is named the **Dialogue Act**. The Dialogue Act has two component: 1) **intent** and 2) **parameters**. The intent is described in detail in the Section 3.1.1. The parameters for a dialogue act comprise of a list of slot-value pairs. Dialogue Act can be defined symbolically as $\textbf{intent}(slot_1 : value_1, slot_2 : value_2, ..., slot_n : value_n)$ where n is the number of parameters. Each slot is related to its corresponding value by a mathematical **operator**. The operator used for the current version are $=, != , <, >, \leq$ *and* $\geq$.

3.2.2.II Dialogue State Tracker

The DST updates the Dialogue State (DS) and Dialogue Context (DC). The DS comprises of the items which can help the NLU and the DP design the Dialogue Acts for the corresponding turns. Some of the essential components are:

- **Information Needs**: Current Information Needs (CIN) store the current user's preference.

  The CIN store values for each slot that can be annotated by the NLU. However, for slots that can have multiple values, CIN should accommodate all. For example, if the user wants a *romance and comedy movie starring Anne Hathaway*, the CIN will be represented as follows:

  ```
  genres = romance, comedy
  actors = Anne Hathaway
  ```

  Whenever the user's preferences are updated, the previous information needs (PIN) are stored in a separate element.

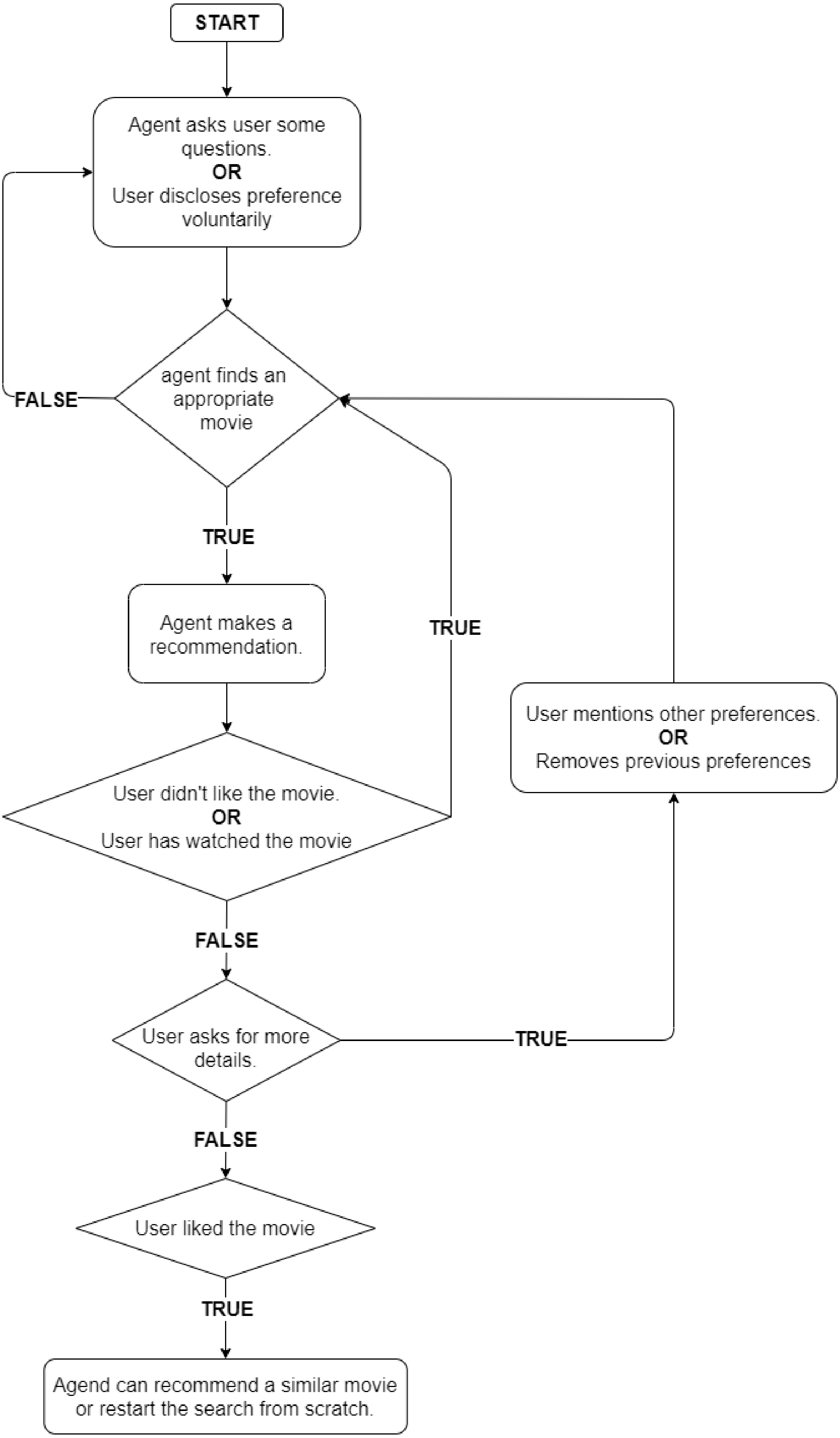


Figure 3.2: Dialogue-Flow of Jarvis Version 1

- **Dialogue Acts**: The recent agent's and user's dialogue acts are stored. These components will be referred in later algorithms as $agent_{dacts}$ and $user_{dacts}$ respectively.
- **Requestables**: A copy of user's and agent's requestables is loaded in the ontology. The user's requestables are used to guide the users on what they can inquire about. If the users inquire about a parameter, it is removed from this copy. The user's requestables are re-loaded when a new recommendation is made.
- **Item in focus**: This element defines the movie recommended by the agent. This element contains all the attributes of the movie under consideration and will be changes as soon as the agent jumps to another recommendation.
- **Database Results**: At any stage, the DM formulates a query and accesses the database, the database results are updated in this item.
- **Agent Offering State**: The items describing the agent state are crucial to decide the next action to be taken by the DP. This items are Boolean flags and are updated based on the User's Dialogue Acts. For the last four items, if one is set to be True, the rest will be automatically set to False.
  - **Agent's Requirements Filled**: If the necessary information mentioned in the Agent's Requestables are filled.
  - **Agent can Lookup**: There are scenarios where the agent requirements are not filled but still agent can search the database to update the users if there are any relevant movies.
  - **At terminal state**: The user wants to exit. This will terminate the conversation.
  - **Agent made *Partial Offer***: If a database has more than **100** movies after querying for user's CIN, the agent will inform the user about the result but will have to ask more questions to narrow down the search space.
  - **Agent should make an Offer**: This item indicates that the agent will be making a recommendation in its next turn. It can be under two scenarios:
    1. There are less than a hundred database items for the recently formulated query.
    2. The agent can leave a maximum of two out of five CIN slots empty. If there are three or less empty slots, agent should not ask more questions and recommend a movie.
  - **Agent made Offer**: This element indicates that the agent has made a recommendation. This flag indicates if agent should answer to user's **Inquire**

intent. If the users **remove a preference** or **reveal** anything more, the agent state can go back to any of the previously mentioned states.

  - **Agent offers no results**: This state indicates that for the given user's CIN, there are no matching database results.

The DC keeps track of the movies recommended to the users with their feedback based on acceptance or rejection of the movie. The feedback for every movie is stored in the form of keywords ("accepted", "don't like", "watched" and "inquired"). This context is used to ensure that the agent does not recommend a movie twice in the same session. If a user quits or restarts the dialogue, the context will be initialized again.

The update of both DS an DC is performed at three stages

- DM receives the Dialogue Acts from the NLU (Algortihm 3.2).
- A list of recommendation(s) is generated from the database (Algortihm 3.3).
- DP generates Dialogue Acts as output of the DM (Algortihm 3.4).

3.2.2.III Database Query Formulation

The agent can access the database if the agent can lookup or its requestables are answered by the user. The query formulation (or reformulation) for this step is kept simple for this phase. A movie is only selected if it satisfies all the conditions saved in the current CIN. Therefore, the conditions are joined by the operand ***and*** for the MYSQL database query.

3.2.2.IV Dialogue Policy

The Dialogue Policy (DP) for Jarvis is a rule-based decision-making process that generates the next action based on the updated DS and DC. This action is in the form of a Dialogue Act. The Dialogue Act always has the intent, but it is not always necessary to have parameters. In this section, the decision-making process of generating intents mentioned in Table 3.2 is explained.

a) Elicit

The intent Elicit is decided under the following conditions.

- If the **agent's requirements are not yet filled**.

- The agent's requirements are filled but the agent is making a **partial offer**.

Algorithm 3.2 *Receive input from the NLU and Update DS* in Jarvis DST

Input: $user_{dacts}$
Output: Updated Dialogue State (DS) and Dialogue Context (DC)
Returns: Nothing

$last\_user_{dacts}$ in DS $\leftarrow$ $user_{dacts}$
for $user_{dact}$ in $user_{dacts}$ **do**
  if *intent* of $user_{dact}$ is REVEAL **or** REMOVE PREFERENCE **then**
    PIN of DS $\leftarrow$ CIN of DS
    agent made offer $\leftarrow$ $False$
    if *intent* of $user_{dact}$ is REVEAL **then**
      Add the *parameters* to the CIN of DS
    else if *intent* of $user_{dact}$ is REMOVE PREFERENCE **then**
      Remove the *parameters* form the CIN of DS
    end if
  end if
  if *intent* of $user_{dact}$ is ACCEPT **then**
    update item in focus of DS to DC
  end if
  if *intent* of $user_{dact}$ is REJECT **then**
    agent made offer $\leftarrow$ $False$
    agent should make an offer $\leftarrow$ $True$
    update item in focus of DS to DC
  end if
  if *intent* of $user_{dact}$ is INQUIRE **then**
    update item in focus of DS to DC
    remove the *parameters* from *user requestables*
  end if
  if *intent* of $user_{dact}$ is CONTINUE RECOMMENDATION **then**
    agent made offer $\leftarrow$ $False$
    agent should make an offer $\leftarrow$ $True$
  end if
  if *intent* of $user_{dact}$ is BYE **then**
    agent at terminal state $\leftarrow$ $True$
  end if
end for
if all(*slots* in **agent requestables**) are filled in CIN **then**
  agent requirements filled $\leftarrow$ $True$
else
  agent requirements filled $\leftarrow$ $False$
end if
if any(*slot* is filled in CIN) **then**
  agent can lookup $\leftarrow$ $True$
end if

The intent's parameters are based on the agent's requestables or other slots for which the agent can ask the users. Algorithm 3.5 shows how the slots are select for the intent elicit. The slots assigned to CIN are the ones that can be annotated by the NLU. Therefore, it also includes the slots in agent's requestables from the ontology. As agent's requestables

Algorithm 3.3 *Updates the DS based on the database results* in Jarvis DST

Input: database results
Output: Updated Dialogue State (DS)
Returns: Nothing

if (count(database results) > 100) **and** (agent can ask more questions) **then**
  Agent made partial offer ← $True$
  return
end if
shuffle the database results
for $result$ in database results **do**
  if $result$ not in DC **then**
    item in focus ← $result$
    agent should make offer ← $True$
    break
  end if
end for
if no $result$ picked **then**
  agent offers no results ← $True$
end if

Algorithm 3.4 *Receive output from DP and Update DS* in Jarvis DST

Input: $agent_{dacts}$
Output: Updated Dialogue State (DS)
Returns: Nothing

$last\_agent_{dacts}$ in DS ← $agent_{dacts}$
for $agent_{dact}$ in $agent_{dacts}$ **do**
  if $intent$ of $agent_{dact}$ is RECOMMEND **then**
    agent made offer ← $tRUE$
    agent should make an offer ← $False$
    add item in focus of DS to DC
  end if
end for

are of higher priority, the algorithm checks for them first.

b) Reveal

The agent intents in the category Reveal are related to showing information or a summary of the information.

- **Count Results**: In Algorithm 3.5, if the agent makes a partial offer, two Dialogue Acts are added to the output of the DP. First, the dialogue act has intent Count Results with parameters defining the number of database results saved in the DS. This is followed by Elicit intent.
- **Recommend**: This intent is selected if the agent should make an offer. Alternatively, if the agent should make a partial offer but only two slots in CIN are left

Algorithm 3.5 Decison making for *elicit* intent in Jarvis DP

Input: Dialogue State (DS)
Output: agent dialogue act containing intent and parameters

```
if not agent requirements filled then
    intnet of agent_dact ← ELICIT
    for slot in CIN do
        if CIN does not have a value for slot then
            parameters of agent_dact ← (slot)
        end if
    end for
else if agent makes partial offer then
    intnet of agent_dact ← ELICIT
    shuffle the slots of CIN
    for slot in CIN do
        if CIN does not have a value for slot then
            parameters of agent_dact ← (slot)
        end if
    end for
end if
```

empty. The agent has already asked enough questions in this case and now should make a recommendation. The parameter of this intent includes the title of the movie it is going to recommend.

- **No Results**: This intent has no parameters as it is an indication that the agent could not find any results for user's CINs.
- **Inform**: As this intent is a response to the user's *Inquire* intent, its parameters include the slots from the user's intent with the value from the item in focus.

c) Miscellaneous Intents

- **Welcome**: This is a response to the user's intent Hi; the agent welcomes the user to the conversation.
- **Bye**: If the agent is terminated, this intent is generated.
- **Acknowledge**: This intent is not yet applied. No such scenarios are created where an agent must agree or disagree with the user.
- **Can't Help**: If the NLU could not calculate any intent or the DP could not reach any decision based on the current DS, this intent is generated.

3.2.3 Natural Language Understanding (NLU)

The NLU in Jarvis considers the following constraints while processing a user's utterance. It is assumed that the users can *reveal* their preference at any stage in the conversation. The NLU must try to find the intent relevant to the agent's last response. However, if not intent detected, it must check if the users want to reveal a new preference.

3.2.3.I Loading Data for NLU

The NLU is initialized to load/create specific components that will help with the intent-detection and slot-filling while detection.

- **Create the slot-values pairs** for each slot that should be annotated by the NLU. This includes a list of values in the database corresponding to each slot. This slot-values is different from the slot-value pairs in the Dialogue Act parameters. The later defines the components of a dialogue act with one-to-one slot to value correspondence while the former has 1:N slot-value**s** correspondence.
- **Load the tags-words for the slots**. This indicates the alternative words a user can write while inquiring about a movie.

  For example, if the users want to ask about the rating of a movie, their utterances can be like

  – What is its **IMDb score**?
  – What is its **rating**?

  An extra slot named **more info** is added to this file to check if the users want general information about the movie. There is a chance that the users may use these tag-words while inquiring about the movie attributes. However, this is left for the scope of phase 2.

3.2.3.II Slot Annotation (Slot-Filling)

The slot annotator in NLU is specifically designed for *slot-filling* for the intent **reveal**. The slots that must be annotated for this intent are mentioned in the ontology (Section 3.2.1.II). The annotation for each slot is an important aspect of NLU, and the specific techniques are described in this section.

For this purpose, the input received from the users is **lemmatized** for a better value matching. For lemmatization, NLTK Wordnet Lemmatizer[2] is used. An example of how lemmatization can help with detecting slots is that the word *star wars* will be lemmatized to *star **war***. This can help with finding star wars movies if the user types "I want a star war movie from the 90s.".

For most annotators, **n-grams are generated** for every input sentence. An n-gram is a sequence of n tokens (words) in a text. For example, for a sentence "I would like to watch the movie on hunting.", the n-grams for $n = 2$ are ["I would", "would like", "like to", "to watch", "watch the", "the movie", "movie on", "on hunting."]. The maximum n-gram size is defined based on the type of slot. The annotator tries to find a match while decreasing the n-gram size. For example, if the maximum n-gram size defined for a slot is 4, the annotator will try to find a match for n-grams of size decreasing from 4 to 1, and if it finds a match for an n-gram of size 3, it will not try to repeat the steps for size 2 and 1.

A list of **stop-words** is also created for slot-annotation. This contains the English stop-words by NLTK[3]. Besides this, another list is generated specifically for movie recommendation. This list contains the common words a user can use in their utterance and can also have a high occurrence in the database. Some examples are "*watch*", "*good*", etc.

a) Movie title

This annotator is used to check if the users have revealed a movie title (for example, *Inception*) or a part of the title (for example, *Star Wars*). The utterance and slot-values are lemmatized. The maximum n-grams size for this slot is 8 considering the user can write a long name to be precise. For each n-gram extracted from the lemmatized utterance (Algorithm 3.6),

1. If that n-gram is in the list of lemmatized movie titles, it will be selected as the output (annotated value for title)
2. Else, extra conditions are applied to the n-gram.
    - It should not be a part of the list of stop-words specified and should not have any number. This is done to avoid n-grams like "*I want to*" or "*90s*" being selected as the output value in this algorithm.

[2] `https://www.nltk.org/_modules/nltk/stem/wordnet.html`
[3] `http://www.nltk.org/nltk_data/`

- It should have more than one token (word). This step is performed to avoid matching of single words containing no significant context as a title. For example, word "*love*" in utterance "*I would love to watch a good movie.*".

3. If the n-gram qualifies the conditions specified in the second point. For a particular n-gram size, the count of lemmatized title-values having this n-gram as a sub-string is calculated.

4. The n-gram with the least count is selected. For example in an utterance "*I would like to watch a star war movie.*". An n-gram of size can yield the occurrence count as (*"would like" = 30, "star war" = 8*). Therefore, the algorithm will select the least common occurrence, and hence there will be a better chance of recommending a more relevant movie.

As the single words are filtered out in the first step. However, this would not affect if the user mentions movies with one token only like "*Inception*" because such instance can be detected in step two, and no further processing is required.

Algorithm 3.6 *Movie Title Annotator* in Jarvis NLU

Input: utterance
Output: the annotated word(s) resembling a title in the database

$slot = Title$
$lemmatized\_utterance \leftarrow lemmatize(utterance)$
$lemmatized\_values \leftarrow list(lemmatize(value)$ for value in $slot\_values[slot])$
for $ngram\_size \leftarrow 8$ to 1 **do**
  $ngrams \leftarrow$ ngrams of $lemmatized\_utterance$ of size $ngram\_size$
  for $gram$ in $ngrams$ **do**
    if $gram$ in $list(lemmatized\_values)$ **then**
      return $gram$
    end if
    if $gram$ has no $stopwords$ **and** $gram$ has no numbers **then**
      if $ngram\_size > 1$ **then**
        $gram\_occurence \leftarrow$ count of the number of $lemmatized\_values$ in which $gram$ occurs as a sub-string
      end if
    end if
    if any $gram\_occurence$ found **then**
      return $gram$ with the lowest $gram\_occurence$
    end if
  end for
end for

b) Genres

The annotation of the genres is the simplest method. However, there is a probability that the users may not give the exact genre present in the database. However, they can mention a word having the same meaning as one of the genres. Therefore a list of genres with possible alternative words is created. The annotator will return the detected genre or its alternative (Algorithm 3.7). Table 3.3 shows a list of alternatives created with examples of annotations.

Table 3.3: Alternatives of Genres in Jarvis

Genres	Alternative	Example	Annotation
romance	romantic	I want a romantic comedy	romance, comedy
crime	criminal	I want criminal movies	crime
drama	dramatic	Give me something with a dramatic end	drama
sport	sports	What about sports?	sport
comedy	funny	I want something funny but not boring.	comedy
history	historical	Do you have a documentary on historical events?	documentary, history
animation	animated	I want a good animated movie for my 3 years old.	animation

Algorithm 3.7 *Movie Genres Annotator* in Jarvis NLU

Input: utterance
Output: the annotated word(s) resembling a genres in the database or its alternatives

$slot = Genres$
for $value$ in $list(slot\_values[slot])$ **do**
  if $value$ in $utterance$ **then**
    return $value$
  end if
  if synonym of $value$ in $utterance$ **then**
    return synonym of $value$
  end if
end for

c) Keywords

The keywords annotator has some similarities with the movie title annotator. The only differences are (Algorithm 3.8)

- while checking if the n-gram is in the list of lemmatized keywords, it is made sure that the n-gram is not in the list of stop-words. This will avoid the single tokens like "*of*", "*I*" or "*please*" if present in the slot-values list for keywords.
- The count of occurrences for a n-gram is not calculated as this step is not as critical as title annotation.

Algorithm 3.8 *Movie Keywords Annotator* in Jarvis NLU

Input: utterance
Output: the annotated word(s) resembling a plot keyword in the database

```
slot = Plot_Keywords
lemmatized_utterance ← lemmatize(utterance)
lemmatized_values ← list(lemmatize(value) for value in slot_values[slot])
for ngram_size ← 8 to 1 do
  ngrams ← ngrams of lemmatized_utterance of size ngram_size
  for gram in ngrams do
    if gram in list(lemmatized_values) and gram is not in the list of stopwords
    then
      return gram
    end if
    if gram has no stopwords and gram has no numbers then
      for value in lemmatized_values do
        if (ngram_size > 1 and gram is sub-string of value) or (ngram_size = 1
        and gram = value) then
          return gram
        end if
      end for
    end if
  end for
end for
```

d) Person names

To annotate the person names, no lemmatization is required. The maximum n-gram size is set to 3, considering this can be the highest number of words a user can use for any longer name (Algorithm 3.9). For every n-gram, the following conditions apply

- the n-gram is not in the list of stop-words. This will filter out single tokens considering some person names in database have tokens like "*hi*".

if an n-gram is a sub-string of any value in the slot-values of slots actors or directors, it will be returned as the output.

e) Year

For this last slot indicating the release year of the movie, the annotation does not account for the slot-values extracted from the database. There is a very rare occurrence when the users write an exact year for the movie. The input can be like

- "I want a movie from the **90s**."
- "Give me hits from the **19th** century."

Algorithm 3.9 *Person Names Annotator* in Jarvis NLU

Input: utterance
Output: the annotated word(s) resembling actor or director names in the database

$slot = Actors$ **or** $slot = Director$
for $ngram\_size \leftarrow 3$ to 1 **do**
  $ngrams \leftarrow$ ngrams of $utterance$ of size $ngram\_size$
  for $gram$ in $ngrams$ **do**
    if ($gram$ is a sub-string of $value$ in $list(values)$) **and** ($gram$ is not in the list of $stopwords$) **then**
      return $gram$
    end if
  end for
end for

- "Do you have something from **1920s**?"
- *wrong sample:* "I want a *1* and a *2* from **1995**."

The algorithm (Algorithm 3.10) filters out the numbers from the input utterance. For the examples above it will be 1) **90**, 2) **19**, 3) **1920** and 4) **list(1, 2, 1995)**.

- The algorithm first checks if any instances like 90s or 1920s are mentioned. For 90s, the NLU will indicate that the agent must search for movies from 1990 to 2000. For 1920, it will indicate the range 1920 to 1930.
- For the 2nd example, the 19th will be converted to range 1900 to 2000.
- For the last example, we can see that there are numbers like 1 and 2 detected by the annotator. This value would not pass the conditions mentioned in the annotator and it will return the value 1995 indication agent must find a movie from 1995.

3.2.3.III Intent-Detection

This task in the NLU detects the user intents based on the current Dialogue State and generates a Dialogue Act. The Dialogue Act contains both intent and the slots. However, some intents do not need to have a slot. For this task, the intents mentioned in Table 3.1 will be detected by the NLU.

a) Reveal

For the detection of this intent (algorithm 3.11), the utterance is checked for slot annotation when the last agent intent is *Elicit*. For example, if the agent elicits about *what genres the user will prefer*, the intent detector should annotate the utterance for

Algorithm 3.10 *Release Year Annotator* in Jarvis NLU

Input: utterance
Output: year or range of year

```
slot = Release_Year
possible_years = list(number in words from the utterance that have) a number
for value in possible_years do
  if value appears as (value + s) in the utterance then
    if value has 4 digits then
      if value is divisible by 10 then
        return range from value to value + 10
      else
        return year
      end if
    else if value has 2 digits then
      if value ≤ 20 then
        append 20 in the beginning of the value
      else
        append 19 in the beginning of the value
      end if
      if value is divisible by 10 then
        return range from value to value + 10
      else
        return year
      end if
    end if
  else if value appears as (value + th) in the utterance then
    if value has 2 digits then
      append 00 in the end of the value
      return range from value to value + 100
    end if
  else if value has 2 digits then
    return value
  end if
end for
```

the slot named *genres*. If the user input does not have a specified genre, it is assumed that the user may have voluntarily revealed a movie title or another keyword to give hints to the agent. This assumption is also made when the NLU is not able to detect any other intent.

It should be noted that the users may not want to answer the agent's question. Users may mention that *anything* can be selected for this slot. Therefore, if the agent does not find an answer, it also checks if the *users do not care to give any preference*. A particular value for this scenario is defined, and the returning Dialogue Act with the intent Reveal is assigned the value in its parameters. During the database search in Section 3.2.2.III, such values are ignored.

Algorithm 3.11 Intent Detection for *reveal* intent

Input: utterance

Output: user dialogue act containing intent and parameters

1: $agent_{dact} \leftarrow$ dialogue act of the recent agent response
2: **if** *intnet* of $agent_{dact}$ is **ELICIT then**
3:     $annotations \leftarrow$ **annotate** the utterance for *slots* in the $agent_{dact}$ *parameters*
4:     **if** *annotations* found **then**
5:         *intnet* of $user_{dact} \leftarrow REVEAL$
6:         *parameters* of $user_{dact} \leftarrow (slot = annotations)$
7:     **else**
            parameters **for** *slot* in list(slots that must be annotated) **do**
8:             $annotations \leftarrow$ **annotate** the utterance for the slot
10:             **if** *annotations* found **then**
11:                 *intnet* of $user_{dact} \leftarrow REVEAL$
12:                 *parameters* of $user_{dact} \leftarrow (slot = annotations)$
13:             **end if**
14:         **end for**
15:         **if** more than one slot annotation found **then**
16:             Filter the annotations based on priority.
17:         **end if**
18:     **end if**
19:     **if** $user_{dact}$ found **then**
20:         **return** $user_{dact}$
21:     **end if**
22: **end if**

If the utterance is checked for a voluntary reveal, the same n-gram can be returned for multiple slots. For example, the word "*war*" can be a movie title and also a genre. Moreover, the value of one slot can be a sub-string of the value of another slot. For example, in an utterance "I want movies on civil war.", the genres annotator will return ***"war"*** and a keyword annotator will detect ***"civil war"***. Therefore, a further step of filtering parameters (line 16 in Algorithm 3.11) is required.

While filtering the parameter, it is checked that the annotations detected do not match the patterns defined for other intents in Jarvis. The *patters* defined in NLU contain a list of one or more words which, if occur in a sentence, can help with the intent detection. After filtering such parameters, the filtering is performed to remove sub-strings like "*war*" in the above example. If two parameters have the same values, it is resolved based on the priority assigned to every slot. The slot with the lower priority is removed.

b) Remove Preference

This intent is detected if any user's preference is saved in the agent's Dialogue State. For example, if the agent has saved *genres* = *Action* as user's CIN, an utterance "I don't want action movies" will remove this preference. For this intent, Telegram keyboard options

(Section 3.2.5) are generated. Therefore a proper intent-detection is not implemented in this phase of Jarvis.

c) Inquire

NLU checks for the user's intent Inquire only if the agent has made any recommendation. If the agent is still in the process of asking questions, this intent would not be checked. As mentioned in Section 3.2.3.I, a list of pattern words for each slot is created. The user's input is checked for these words, and if any exists, the resulting Dialogue Act has intent Inquire with parameter equals to the respective slot (Algorithm 3.12).

Algorithm 3.12 Intent Detection for *inquire* intent

Input: utterance
Output: user dialogue act containing intent and parameters

$agent_{dact} \leftarrow$ dialogue act of the recent agent response
if agent has made a recommendation **then**
    intnet of $user_{dact} \leftarrow INQUIRE$
    for *slot* in *tag-words* **do**
        if any *tag-words* for *slot* in *utterance* **then**
            $parameters$ of $user_{dact} \leftarrow (slot = annotations)$
        end if
    end for
    if more than one $parameters$ found **then**
        return $user_{dact}$
    end if
end if

d) Accept/Reject

As the Accept and Reject intents can only be recognized once the user has made a recommendation. Therefore these intents are also checked in parallel with the intent Inquire. For these intents, patterns are designed to match an utterance to.

- For the intent **Accept**, the intent can be detected before or after the recommendation is made.
- For the intent **Reject**, the parameter defines the reason for the rejection of a movie. It can either be **watched** defining that the users have watched a movie or **don't like** indicating that the users do not like the recommendation before or after inquiring further details about it.

e) Miscellaneous Intents

To detect the intents Hi, Bye, Acknowledge, or Deny, specific patterns are designed. If the utterance have these patterns, there is a probability that the user has a matching intent under the following conditions

- **Hi:** If the conversation is not started yet and the user wants just to say Hi.
- **Bye:** The users can exit anytime. Therefore, irrespective of the last agent response, the user response is first checked for this intent.
- **Acknowledge:** For the current version, the agent is not asking the user any questions which need an acknowledgment. However, this intent is checked when the agent welcomes the users. The agent asks if it should start with the search process. The users can either acknowledge or reveal any preference.
- **Deny:** This intent is not yet required for the current version.

3.2.4 Natural Language Generation (NLG)

For each agent intent generated by Dialogue Policy in Section 3.2.2, the NLG generates a human-understandable output. For every intent, multiple responses are designed for a random selection. This will generate a more natural dialogue rather than the agent generating the same sentence for an intent every time.

- **Elicit**: For every parameter slot, multiple questioning statements are created, and the NLG selects the response randomly. For example, if the parameter has slot “keywords”, the NLG will randomly select from: 1) “Can you give me a few keywords?” and 2) “What are you looking for in a movie? Some keywords would be good.”
- **Reveal**: For intents under this category, the NLG also **clarifies the user’s information needs**. This clarification includes a summary of what the agent has stored as CINs and helps the users understand the process.
  - **Count Results:** The NLG will round the count in the parameter to the nearest 100s. This count will be mentioned while the agent clarifies the CIN. As this intent is followed by an Elicit intent, a connecting statement is added. For example, “Please answer a few more questions to help me find a good movie.” This statement is also selected randomly from a list of options created.
  - **Recommend**: While recommending the movie, a follow-up statement is added, asking if the users have watched this movie.
  - **No Result:** This declaration follows up with explicitly mentioning if the agent wants to remove a preference.

– **Inform:** Rather than just writing the values fetched from the item in focus, the values are changed for a more natural response.

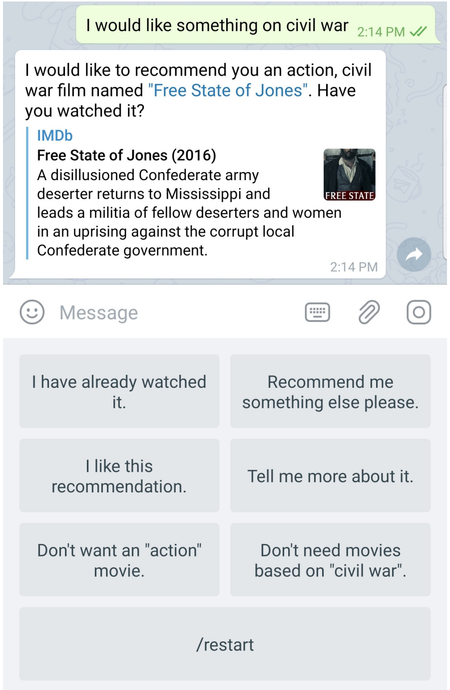
3.2.5 Multi-modal Chat Interface

To converse with the users and provide them with a more interactive environment, the Jarvis movie recommender system is launched on a messaging app named Telegram [4]. To integrate a Telegram Bot in Jarvis, python-telegram-bot [5] API is used. A significant benefit of using Telegram is that while an agent sends a response, it can also show some buttons for the users to choose from. For version 1, keyboard buttons are used. If the users select a button, the text mentioned on the button will be the next user's response. However, the users can always choose not to use these buttons and type another response they want. The buttons are generated by the NLG, and the list of buttons depends on the current agent intents. However, the intent for the buttons is generated with a user's intent attached to them. This makes it easier for the NLU to process the utterance.

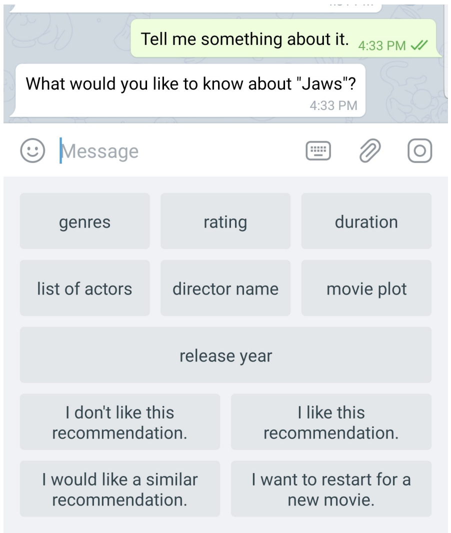
- **Remove Preference:** While the user *clarifies the information needs*, it also generates buttons for the user to remove their preferences (Figure 3.3 (a)).
- **Accept/Reject**: The options to Accept and Reject a movie are presented when the agent intent is *Recommend* a movie or *Inform* about it (Figure 3.3 (a)).
- The button to **Continue Recommendation** is presented when the agent *Informs* the user about the movie (Figure 3.3 (b)) or the user has *Accepted* the recommendation and agent gives them options to Continue (find a similar recommendation), restart or exit the conversation (Figure 3.3 (c)).
- **Inquire:** When the agent *Recommends* a movie, another button is provided with a linked intent Inquire and parameter "**more info**". Once this button is clicked, the agent will list all the attributes a user can ask about. While the agent clicks on these buttons, the respective buttons will disappear from the keyboard buttons and the agent will Inform the user about the selected attribute (Figure 3.3 (b)).

Moreover, while executing Jarvis on telegram, multiple users can access the system at one instance. Therefore, the controller must be able to differentiate between users. For this purpose, separate instances of the agent are initialized for each user. For memory optimization, the instance is removed if the user exits the conversation. The telegram controller also keeps track of its users using their identification numbers. If a new user

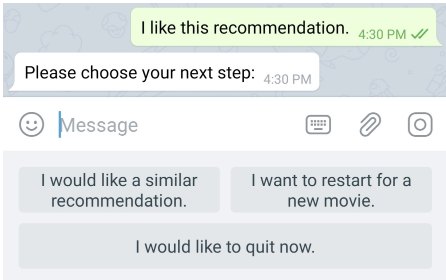
[4] `https://telegram.org/`
[5] `https://github.com/python-telegram-bot/python-telegram-bot`



(a) Accept/Reject and Remove Preference



(b) Attributes user can ask about



(c) Continue Recommendation

Figure 3.3: Telegram keyboard buttons

interacts with the system, a brief guide is presented to the user. This helps the user to understand the shortcuts it can use during the conversation. The figures for the user instruction are mentioned in Chapter 4.

3.3 Jarvis Version 2

The comments received for version 1 (see Section 4.2) are used to designed a list of features that can be implemented in the next version. The selected drawbacks with the proposed implementation is as follows:

1. *Database and Dialogue Policy*: The database is not updated and it mostly recommends old movies with a low rating. This problem can be solved by updating the

database. Moreover, during query formulation, it should be made sure that the agent does not select the movies with low rating. While making a recommendation, the agent randomly selects a movie from the list of results. This can result in picking a low-rated movie. Therefore, the random selection from the results will be removed for version 2.

2. *NLU, Database Query and Dialogue Policy*: When the user ask for a movie *similar* to the current recommendation, the users perceived that movies were not really similar. This is because the similar recommendation involved getting a movie from the database using the same user's CIN. The CIN does not ensure a similar movie. Therefore, a new method is implemented which will be discussed in the description of NLU and DP.

3. *NLU and NLG*: The Telegram app showed many buttons at some stages in the conversation. The users like some options but it is noticed that some buttons were redundant and unnecessary.

4. *Dialogue Policy*: The agent should not ask more questions. Buttons are reduced wherever possible. To reduce the buttons, addition checks are implemented in the NLU and NLG.

5. *NLU, Database Query*: The response from the agent should not take longer. The reason behind this can be the slot-filling in the NLU. The latency will be improved in this version.

6. *NLU* The agent detects the wrong keywords from the user statement. This issue is a learning curve and will every test, the slot-annotation is being improved.

3.3.1 External Components in Jarvis

3.3.1.I Database

As the main focus of version 1 was on intent detection and dialogue policy design, not much effort was made to create a good database. A major draw-back in the database were:

- The data is not sufficient. Only forty-seven hundred movies were available.
- The database is not updated. It had movies till the year 2016.

In order to address these issue, a new database is created. To get a list of movies for this purpose, MovieLens 25M Dataset[6] is used. MovieLens is an online available dataset by GroupLens[7], a research lab in the Department of Computer Science at the University of Minnesota, Twin Cities. This dataset included 62,000 movies with 25 million rating and 1 million tags assigned by 162,999 users. For the list of movies in MovieLens dataset, a python package name IMDbPY[8] is used to retrieve the details of movies from IMDb[9]. The parameters of movies fetched from IMDb are genres, plot keywords, list of actors, directors, movie duration, movie plotline (summary), release year, movie rating with number of votes and links to the movie page and its cover image on IMDb.

As the data was raw, significant efforts are made to improve the quality of data for smooth search process and user experience. One of the important attributes for query formulation in Jarvis is *plot keywords or keywords*. During the initial implementation of database creation, the movie tags from the MovieLens dataset were selected as keywords. However,only seventeen thousand movies were assigned tags by the users. If a tag is assigned to a movie by minimum three users, it will be added to the database as keywords for the movie. The tags were further filtered for common terms like “IMDb top 250” or person (actor/director/writer) names. After the initial processing of movie tags, only forty-seven hundred movies were left in the database. Therefore, IMDbPy is used to fetch movie keywords from IMDb. More than enough keywords are available at IMDb and this abundance can effect the performance of NLU. Algorithm 3.13 shows a step-by-step process for finalizing the keyword parameter for every movie in the database.

Algorithm 3.13 Keywords extraction for movies in the Database

if $length(keywords)$ 5 **then**
    select the first five keywords
end if
add keywords which are also in the plot-outline/summary of the movie
Remove keywords having person names
if $length(keywords) \leq 1$ **then**
    include tags from MovieLens which are not included as genres or person names for the movie
end if

Another attribute that can be compared with MovieLens data is the *movie rating*. For each movie, number of votes from both IMDb and MovieLens are compared. Only a hundred movies have more votes in MovieLens than the ones in IMDb. Therefore, to keep the consistency, only movie ratings from IMDb are used for the final database.

[6] https://grouplens.org/datasets/movielens/25m/
[7] https://grouplens.org/
[8] https://imdbpy.github.io/
[9] https://www.imdb.com/

The movie *cast (actors)* fetched from IMDb also contains a long list. In order to optimize the conversational process, only first six actors per movie are added to the database. It has been observed that the list of actors fetched is ordered by the importance of the actor's role in the movie. Moreover, the movies with *genres =short* are also not added to the database. This is because short movies are not very common, have length less than forty minutes and therefore can effect the quality of the recommendations made.

Except for the attributes containing the url to the cover image and the movie duration, it is required that the movie should have a value saved for other attributes. The system needs the genres and plot keywords to formulate a good query and a user can further inquire about values like plot, actors, ratings, etc. Therefore, if any movie has no values for the important attributes, it must be omitted from the database. It is made sure that filtering such movies is not omitting the top-rated or famous movies from the database. After this final step, the database has almost forty-thousand movies for the version 2.

3.3.1.II Ontology

The definition of ontology parameters designed for version 2 are kept similar to those in version 1. The only difference is that a new component defined as **"slots to annotate"** is added to the ontology. This parameter explicitly defines the slot values NLU must check for in the user utterance.

3.3.2 Dialogue Manager

In the DM, major changes are made in the DST. As every configuration change in one component of the agent leads to the corresponding changes in the other components. The changes is DST are meant to improve the latency and adapt to the changes made in the NLU. However, the changes in DP are not significant. The algorithm is only tweaked to improve the speed of the agent response. Therefore, DP would not be discussed in detail in this section.

3.3.2.I Dialogue State Tracker

For version 2, the following parameters are added to the Dialogue State.

- **agent must clarify** and **slots to clarify**: If the slot-annotator in NLU detects the same value for two slots, it must ask the user to clarify. For this purpose, the flag *agent must clarify* was set to true.

For example, if the detected values are *actors = Tom Hanks* and *director = Tom Hanks*, and the user has not mentioned if they need the person as an actor or director. These values are added to the parameter *slots to clarify*. The agent will continue the conversation and show keyboard buttons for the user to choose one option.

- **agent should recommend similar**: This flag indicates the query formulation for the database. If a user has asked for titles *similar* to a specific title, this flag is set to True. This flag will be set to False if the user CIN is updated or it did not accept any of the movies being recommended. For example, for CIN x and movie item in focus y, the agent has n movies similar to y. The agent will recommend one movie at a time. The user has option to reject the movie. At a specific turn, if agent has recommended all the n movies, the flag will set to false and the agent will now recommend movies for the CIN x.

For the intent Reveal, if the slot-value pair indicates that the user does not prefer this value for the slot, the value is added to the CIN with a specific tag. This tag helps with database query formulation and agent utterance in NLG. Moreover, if the slot-value pair already exists in the CIN, it will be removed (similar to remove preference) and if it does not exists, it will be added with the negative tag.

3.3.2.II Database Query Formulation

The database query is different for the state where *agent must recommend similar*, the list of similar movies is fetched from the database with condition *title = <movie-name>* and the conditions are joined by an **OR**. This means all the relevant movies are fetched at once and stored in the database.

It was noted that the database query was another element which was causing latency in the conversation. The query was being processed every time the agent must make a recommendation. For a large database, this can be a time consuming process. Therefore, in version 2, the database is only accessed if the user's CIN change or similar movies are required to be fetched. This step has visibly improved the time agent takes to respond.

3.3.3 Natural Language Understanding (NLU)

3.3.3.I Loading Data for NLU

As the database gets larger, there is a need to load more parameters that will save computational complexity later in the slot-annotation process. The step that takes the most time in slot-filling is the lemmatization of the values in the slot-values. Therefore, for each value, a lemmatized results is saved. To save more computation, the slot-values with lemmatized value are saved as a JSON file. It can be loaded whenever the chatbot initializes the NLU.

Considering the additional slots in the new database and ontology, new tag-words are added.

3.3.3.II Slot Annotation (Slot-Filling)

For the task of slot-filling in NLU, main focus was on improving the computational speed of the annotators. As the database size is increased, the annotation can take longer time. For the first step, focus is on improving the annotation results. For this purpose, multiple new experiments were employed. These techniques, include a Wikipedia search using the Python Wikipedia API[10]. Algorithm 3.15 shows the steps for this technique. For version 1, the Genres and Release Year annotation gives 100% results. Therefore, this technique is only applied to the movie Title, Keywords and Person Names.

Algorithm 3.14 Annotation of utterance using Wikipedia

Input: utterance, slot_values
Output: the annotation for title, keywords and person names

```
results ← Wikipedia search result for utterance
filter results that contain any word from utterance
lemmatize_results ← lemmatize(results)
for r in lemmatize_results do
    for v in lemmatized_values do
        if r is substring of v then
            add non-lemmatized value of r to annotations
            break
        else if v is sub-string of r then
            add non-lemmatized value of v to annotations
        end if
    end for
end for
if annotations have any word similar to utterance then
    return matching annotations
end if
```

This technique is tested for the conversations recorded in Version 1. Although it is a generalized technique and it gives good results. However, the performance of this technique is not better than the annotators in version 1. It is observed that for 50% instances, annotation in V1 is better than this current algorithm and vice versa. Moreover,

[10] https://pypi.org/project/wikipedia/

the execution time is observed for annotation in both V1 and the Wikipedia annotator. The annotators in version 1 are 2-10 times faster than the Wikipedia annotator. This speed depends on the slot under consideration. The total time of annotating all slots using the Wikipedia Annotator takes 2-3 times more time than in V1. Considering the Genres and Release Year annotators were kept unchanged, the Wikipedia Annotator is not a good approach for improving the latency of the conversation. Besides the algorithm in Algorithm 3.15, multiple other techniques based on Wikipedia and IMDb search and term-frequency ranking were employed. However, nothing helped with the improvement of rule-based annotation. Therefore, the algorithms in Section 3.2.3.II are further improved for better annotation. The techniques are tweaked to minimize any redundant computations. For example, as it is obvious that lemmatization of the values fetched from the database is required, this step is performed for all the values during the phase of loading the data from the database (Section 3.2.3.I).

For improving the performance of the annotation results

- The list of stop-words is expanded. This is done by checking the previous usage data and experimenting with new sentences.
- Release year annotation checks for terms like *new movie* etc. are added.
- Person name annotators is improved for
  - Detecting a name for both slots actor and director simultaneously. This will resolve the wrong detection of a name in slot x if annotation in slot y is better. For example, if the user writes *Brad Pitt*, the annotator used to detect the actor *Brad Pitt* but also the director *Brad*.
  - If an annotated name is present in the slot-values for both actors and director, it should check the utterance (or a relevant part of it) for the tag-word of both slots. If it finds a relevant word, the annotation is assigned to the respective slot. Else, it is left up-to the user to decide by giving options using Telegram buttons.
- As the annotation is performed on a lemmatized sentence, the relevant full-form word is selected form the user utterance. This will help with a better clarification in the NLG.
- It is possible that the user is specifying what it does not like rather than what it would prefer. To detect the user's preference for each annotation in the sentence, the part of the sentence between the corresponding instance and the previous detected annotation is checked for a pattern that identifies if the user dislikes this annotation.

> For example, in a sentence *I want action movies but not on civil war.* This sentence annotates the words *action* and *civil war* for which the relevant parts of the sentence are *I want* and *but not on*. This indicates that the user needs an action movie but any movie on civil war should not be recommended.

When the user needs similar recommendations to the current movie in focus, the NLU extracts a list of movies using Wikipedia search. The Wikipedia query to generate results is "*I need films similar to <movie-in-focus>*". Maximum twenty results are fetched which can contain titles of the movies. To extract movie titles form the results, the list of movie titles (slot-values for slot titles) fetched from the database is checked. If a value in the results is in the database, it is selected as a similar movie. These filtered results will be used in the database query formulation and DP to continue the recommendation.

Algorithm 3.15 Selection of similar movies using Wikipedia

Input: movie-in-focus
Output: titles of similar movies

query ← "I need films similar to movie-in-focus"
results ← Wikipedia search result for *query*
for value in *results* **do**
    if value **not** in slot-values for title **then**
        remove value from *results*
    end if
end for
return *results*

3.3.3.III Intent Detection

a) Remove Preference

In version 1, this intent was not recognized by the NLU and specific buttons were created to remove preference. However, to reduce the amount of buttons, this intent is merged with the **Reveal** intent. The negative preference of the user is detected during slot-annotation (Section 3.3.3.II).

b) Deny

There is a stage in the conversation where the agent asks the user if it has watched the movie. It was observed that the user may deny in response rather than selecting the available buttons. For such instance, the Deny intent is being checked for.

3.3.4 Natural Language Generation (NLG)

NLG responses are updated for clarifying the user's information needs. If a user has expressed a negative sentiment towards a CIN element, it must be reflected in the NLG response and the buttons (options) it creates for Telegram.

To reduce the frequency of Telegram buttons, the options to remove a preference are only integrated

- If the agent could not find any more movies for the CIN.
- Multiple slots in the CIN have the same value and the user must be able to select one.

Chapter 4

Evaluation

In this chapter, the evaluation of the Jarvis conversational recommendation system is presented. The system is analyzed based on the the following:

- The feedback received from the real-time users of Jarvis. This determines how well the users will like this movie recommendation system.
- The quality of the conversation. This is defined in terms of how well the agent detects the user intents and generates a suitable response.

4.1 Experimental Setup

To evaluate the performance of the system, the Jarvis movie recommendation system on Telegram is shared with the participants. The users are given an overview of the objectives Jarvis can achieve and are also provided with instruction to use the Telegram interface. Figure 4.1 shows the instructions for the version 1 of Jarvis. For version 2, written instructions were sent to users (Figure 4.2). Besides these instructions, a welcome screen is created for users (Figure 4.3 (a)) and if a new user starts a conversations, brief instructions are displayed for them (Figure 4.3 (b)).

Jarvis

Welcome to Jarvis. Jarvis is a chatbot created to help you with finding a good movie recommendation. You can access the system at https://t.me/Jarvis_MRSv1_bot

Instructions

- To start the conversation, you can enter the following option:
  - /start
  - **Hi** or **Hello**
  - Ask any questions like *I want some romance movies from the 90s*
- The chatbot will ask you a few questions to find you a suitable movie. During the the conversation, the chatbot will provide you with buttons. For example,
  - If the user says **Give me some comedy movies**, the chatbot will display the buttons to remove the criteria of comedy movies.
  - If the user says **I want star war movies**, he/she can always press the button saying **I don't want "star war" movies**.
  - When the chatbot recommends a movie, you may have **watched it** or **don't like it**. Or you can ask **more about the movie**.
  - Even later, you can **ask for a similar movie** or **restart**.

a) Part 1

Note: On the bottom of the keyboard, if the icon ▪ appears, you have the option to select from the buttons.

- If the agents asks for your preference, you have the option of not answering.
  - For example, if the agent asks "Do you have any favorite director?", you can answer "It doesn't matter", "I don't care", "Anything", "nobody", "none" etc.
- Type **/** or click on the icon ▪ enter a command. The commands are:
  - /start : start a conversation
  - /restart : restart from scratch. The agent will forget whatever you said previously.
  - /help : get a list of instructions
  - /exit : exit the recommendation. You can also say BYE in a humanly manner :smile:
- **Important:** Kindly provide us with your valuable feedback at this link: https://forms.gle/WZpDcqKRxLZRSV6G8

Jarvis is maintained by javeriahabib09
This page was generated by GitHub Pages.

b) Part 2

Figure 4.1: Instruction for Jarvis version 1

1. For this version, you can specify what you do not like.

   For example, I want action movies but I don't like Brad Pitt.

   Or you can say I want old movies or new movies.

2. The movies recommended have the highest ratings and no movie rated below 5 will be shown.

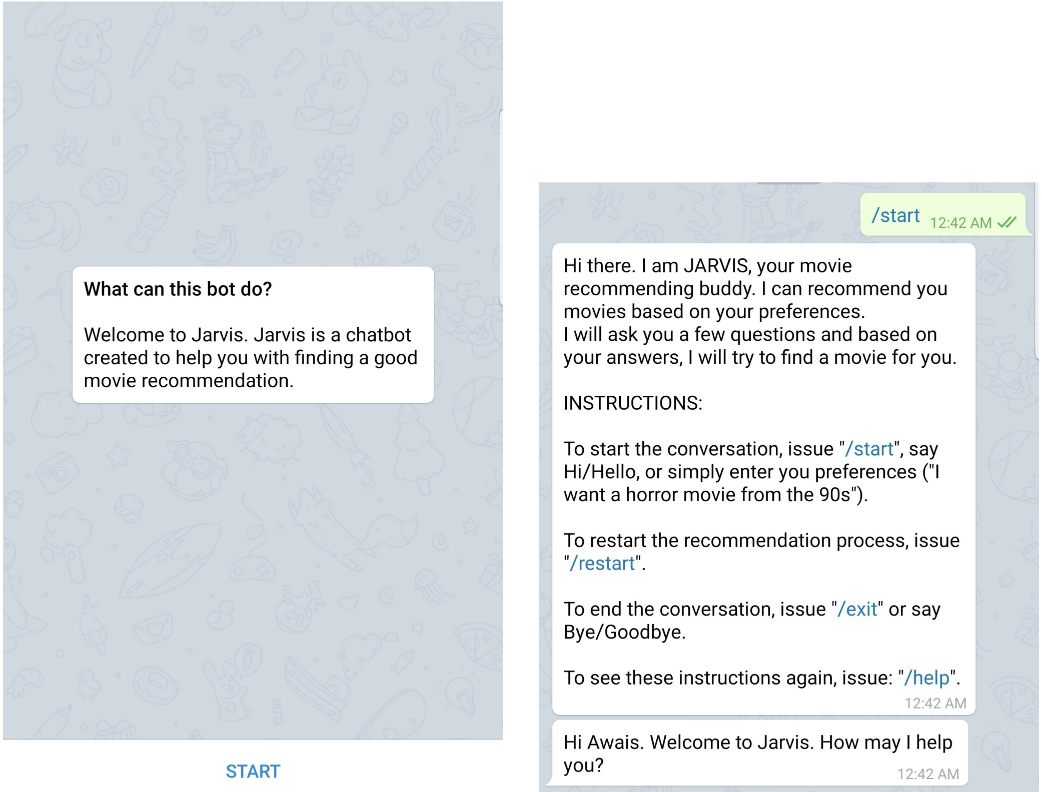
3. Lesser buttons than before. You will only see buttons to change preferences if the chatbot gets confused somehow.

4. Similar movies recommended to the current movie are carefully selected.

5. It better understands the requirements.

6. The feedback needs to be filled again.

Figure 4.2: Instructions for Jarvis version 2



(a) Welcome screen (b) Instructions for new users

Figure 4.3: Instruction for Jarvis version 1

To facilitate the user, Telegram commands are used which can help the user to start, exit, or get help during the conversation.

The conversations of the participants are recorded with their consent. For each dialogue turn in the conversation, the following parameters are stored to further analyze the system:

- *User's Input:* What user says for the specific turn
- *Previous Information Needs:* This is stored to analyze how and if the user information needs change with every turn.
- *User's Dialogue Acts:* The dialogue acts calculated by the NLU. This item is critical in deciding the performance of the NLU.
- *Current Information Needs:* Based on the dialogue acts, the CINs are store to check if the changes are being reflected properly.
- *Agent's offer State:* To evaluate if the DST is updating the current state of the agent efficiently. Based on the CIN and User Dialogue Acts, it is checked if the agent is adapting to the user's changing requirements.

- *Agent Dialogue Acts:* This parameter store the agent dialogue state reflecting the agents offer state. Therefore, it defines the performance of the DP.
- *Agent's Response:* This parameter is stored to analyze the NLG and to check if the NLG is exhibiting correct results for the unknown scenarios.
- *Telegram Keyboard Option:* Parameter exhibiting the keyboard options NLG creates and the user's dialogue acts associated with each option.
- *Dialogue Context:* DC gives an overview of how user's feedback is being stored for each movie.

4.1.1 User satisfaction survey

For the evaluation of Jarvis, a feedback form [1] using Google Forms was created to gather the reviews. This form gathered the basic information about the users (age, gender, education etc). The users are asked to rate the system based on the quality assessment statements in Table 4.1. The rating is done using 5-point Likert scale to analyze how much they agree or disagree with a giver statement. The questions in the Table 4.1 are generated using the criteria described by Radziwill et a. [68]. Moreover, users were also asked to list what they liked and disliked most about the system. This feedback guided a better route while implementing further improvements in the version 2 of Jarvis.

Table 4.1: Questions created for User Satisfaction Survey

Evaluation Category [68]	**Question**
Efficiency	Q1. The system is effective in recommending movies.
Effectiveness	Q2. The system is able to understand my requirements.
	Q3. The system is able to adapt to changes in my requirements.
	Q4. I understand what my options are during the conversation.
	Q5. The conversation with the system feels like a normal human-human conversation (i.e. it is not robotic at all)
Satisfaction	Q6. My experience with the system was overall enjoyable.
	Q7. I would use this system in the future for movie recommendations.

[1] `https://forms.gle/WZpDcqKRxLZRSV6G8`

4.2 User satisfaction survey

4.2.1 Feedback

The evaluation of the version 1 involved fifteen participants who volunteered to give a feedback on Jarvis. Eleven participants (73%) were male and four (27%) were female. These participants were 25-40 years old with a mean age of 32. Only one participant was a native English speaker and 75% of them had a Masters as a highest degree.

Figure 4.4 shows the rating user provided to version 1 based on the questions mentioned in Table 4.1. It can be seen from the chart that the system was not able to understand requirements (Q2) for many users and it lacked the properties of human-human conversation (Q4). For the remaining properties (Table 4.1), the performance of Jarvis is version 1 was quite satisfactory.

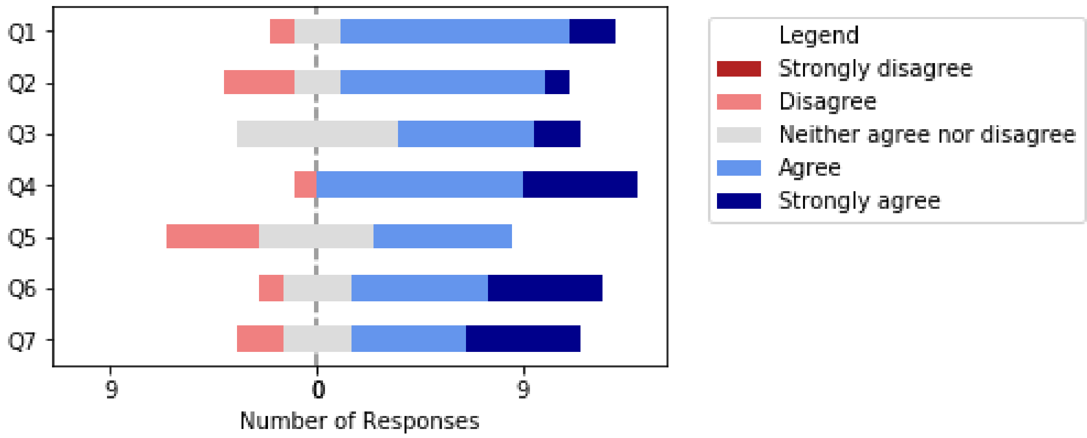


Figure 4.4: User Feedback for Jarvis Version 1

The evaluation of version 2 involved fifteen participants and thirteen participants were the same as for the version 1. Ten participants (66%) were male and five (33%) were female. These participants were 25-40 years old with a mean age of 31.No participants were native English speaker and 73% of them had a Masters as a highest degree.

Figure 4.5 shows the rating of the user for version 2.

4.2.2 Analysis

The user satisfaction is improved form Jarvis Version 1 (Figure 4.4) to Version 2 (Figure 4.5). For version 1, it is shown by the Likert scale plot that the system was not able to understand requirements (Q2) for many users and it lacked the properties of human-human conversation (Q4). For the remaining properties (Table 4.1), the performance of

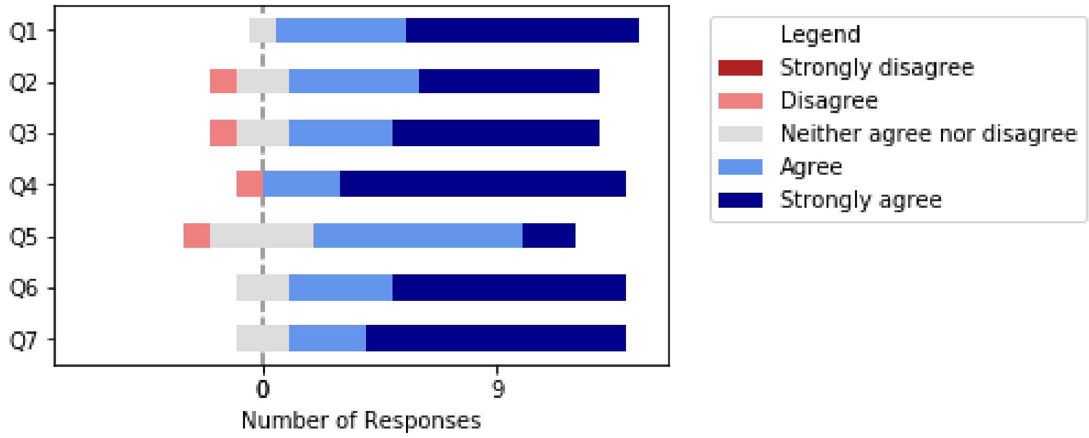


Figure 4.5: User Feedback for Jarvis Version 2

Jarvis is version 1 was quite satisfactory. In version 2, the performance of the system is improved for all the quality scenarios. For most scenarios, the average rating of the system is improved from 4 to 5. This indicates a visible upgrade of Jarvis from version 1 to version 2.

4.3 Movie Recommendation - Quality of the conversation

In this section the resulting conversation for both versions of Jarvis is shown in parallel with a comparison of elapsed time for each conversational turn. It must be noted that for version 2, the size of the database is increase by ten times. Therefore, if the larger database would be used for version 2, some dialogues would have taken even longer time. For the comparison, the user has carried out conversation with the same goal, i.e, the query utterance by the user is kept same to analyze the difference between the performance of both systems.

4.3.1 Experimental Results

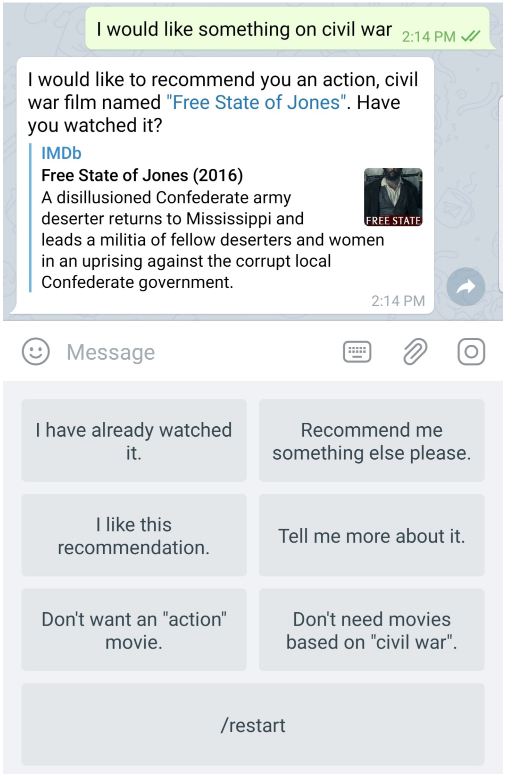
Figure 4.6 to 4.13 show a sample conversation of the user with both versions of Jarvis. Moreover, Figure 4.15 to 4.16 show the improvement in Version 2. For each user utterance in these figures, the Dialogue Acts calculated by the NLU and total time taken by the agent to respond is mentioned in Tables 4.2 and 4.3.

Figure 4.6 shows the simple query where the user starts a conversation and asks for a movie with *genres* action. It can be seen that Version 1 has 1100 action movies while version 2 has 4700. The dialogue acts detected in both versions are the same (Table 4.2). As the agent has not already asked (elicit) a question, the slot-value pairs in the dialogue

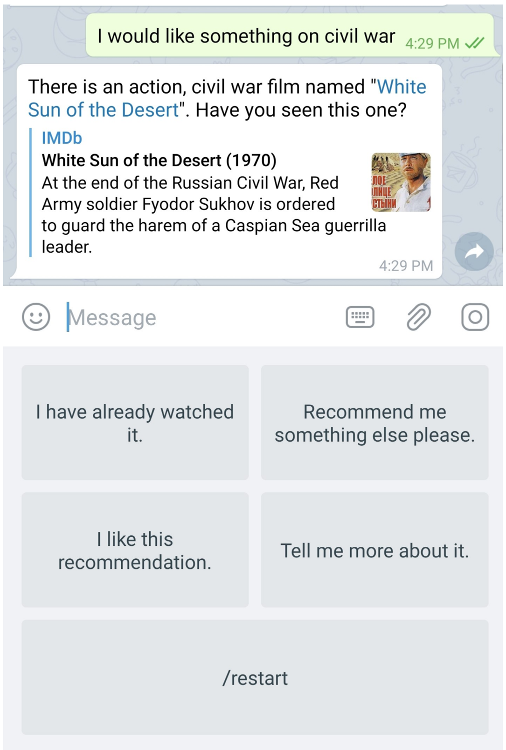
(a) Version 1

(b) Version 2

Figure 4.6: Agent detects the intent reveal and responds with the total count of results and a following question



(a) Version 1



(b) Version 2

Figure 4.7: User answers the agent's question and agent recommends a movie

Table 4.2: User utterance with corresponding Dialogue Acts in version 1 and version 2

	Utterance	Version 1	Version 2
Fig. 4.6	Hi. I want action movies.	reveal(genres = action)	reveal(genres = action)
Fig. 4.7	I would like something on civil war	reveal(keywords = civil war)	reveal(keywords = civil war)
Fig. 4.8	I have already watched it.	reject(reason = watched)	reject(reason = watched)
Fig. 4.8	Recommend me something else please.	reject(reason = dont_like)	reject(reason = dont_like)
Fig. 4.9	I like this recommendation.	accept()	accept()
Fig. 4.10	I would like a similar recommendation.	continue_ recommendation()	continue_ recommendation(titles = titles of similar movies)
Fig. 4.11	Tell me something about it.	inform(more_info = Jaws)	inform(more_info = Jaws)
Fig. 4.15 - V1	I don't want anything by Annabelle Wallis.	reveal(name = annabelle, actors = annabelle wallis)	
Fig. 4.15 - V2	I don't want anything by Ashley Palmer.		reveal(actors != Ashley Palmer)
Fig. 4.16	Hi. I need action movies but not by Angelina Jolie	reveal(genres = action, $director_name = angelinajolie$, $actors = angelinajolie$)	reveal(genres = action, actors != Angelina Jolie, directors != Angelina Jolie)
Fig. 4.17	I want action movies but not by Brad Pitt	reveal(genres = action, actors = brad pitt)	reveal(genres = action, actors != Brad Pitt)
Fig. 4.18	I want a new movie.	reveal(actors = new)	reveal(year = > 2010)
Fig. 4.16	No	Unknown	inform(deny = Avec l'amour)

act are calculated by annotating for all the slots. However, although searching through a larger data, version 2 responds 4.5 times faster than version 1 (Table 4.3).

Figure 4.7 shows the next step in the conversation where the user revealed their second preference. The agent annotated the user's utterance for the slot *keywords*. As the user has found a slot-value in keywords, it did not annotate the utterance for other slots. The user has provided their preference for two CINs, therefore, the agent is now recommending a movie. The outlined process and the dialogue act is same for both versions (Table 4.2).

Table 4.3: User utterance with the time taken (in seconds) by agent to respond in version 1 and version 2

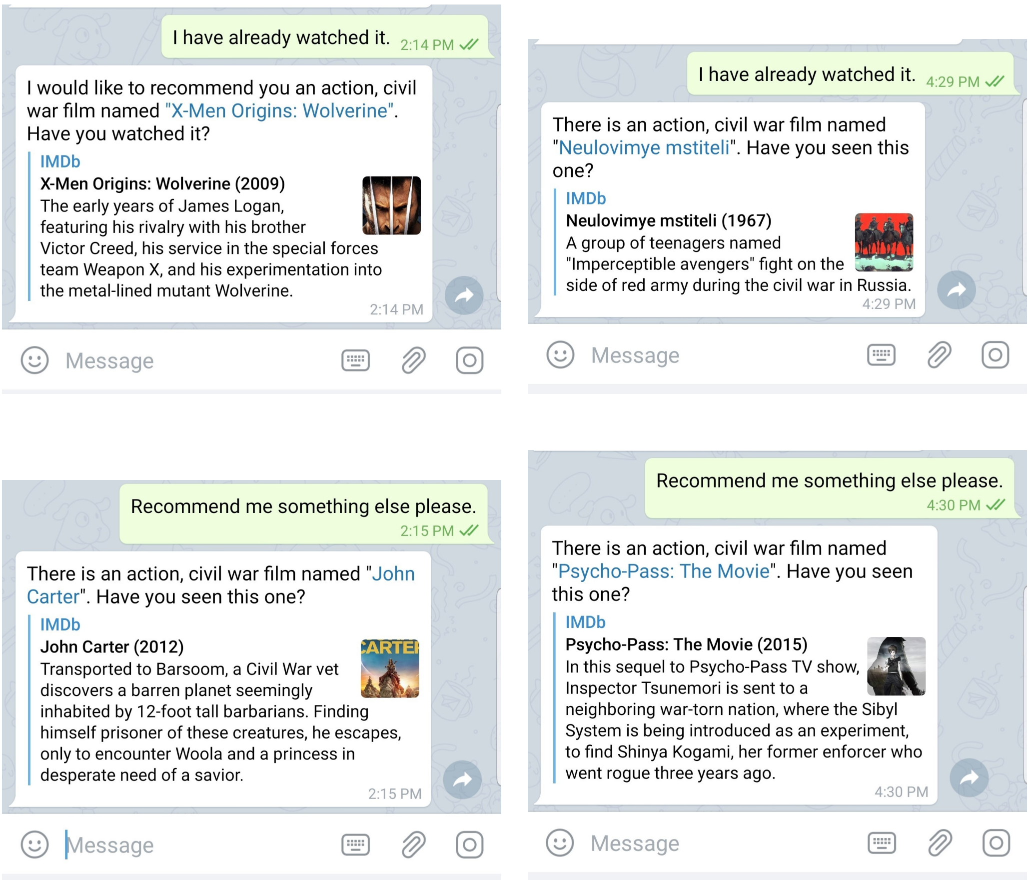
	Utterance	Version 1	Version 2
Fig. 4.6	Hi. I want action movies.	7.82	1.763
Fig. 4.7	I would like something on civil war	5.4	0.098
Fig. 4.8	I have already watched it.	0.011	0.008
Fig. 4.8	Recommend me something else please.	0.011	0.008
Fig. 4.9	I like this recommendation.	0.001	0.001
Fig. 4.10	I would like a similar recommendation.	0.010	0.948
Fig. 4.11	Tell me something about it.	0.001	0.01
Fig. 4.15		10.37	1.23
Fig. 4.16	Hi. I need action movies but not by Angelina Jolie	21.73	2.2
Fig. 4.17	I want action movies but not by Brad Pitt	21.07	0.812
Fig. 4.18	I want a new movie.	17.45	0.75
Fig. 4.16	No	0.011	0.133

However, the agent's response time for Version 2 is exceptionally faster than version 1 (Table 4.3). The amount of buttons is reduced for Version 2 (Figure 4.7) and this helps the user with the key decisions they can take once the movie is recommended.

Figure 4.8 shows the response of the user where they have watched the movie (Table 4.2) or do not like it (Table 4.2). The user are offered a next recommendation in both cases. As these utterances do not require annotation, the agent major task is to search for a new movie in the database (Table 4.3). However, this time is significantly reduced for a larger database in version 2. The keyboard buttons for both versions in Figure 4.8 are the same as those in Figure 4.7 and therefore are not shown again.

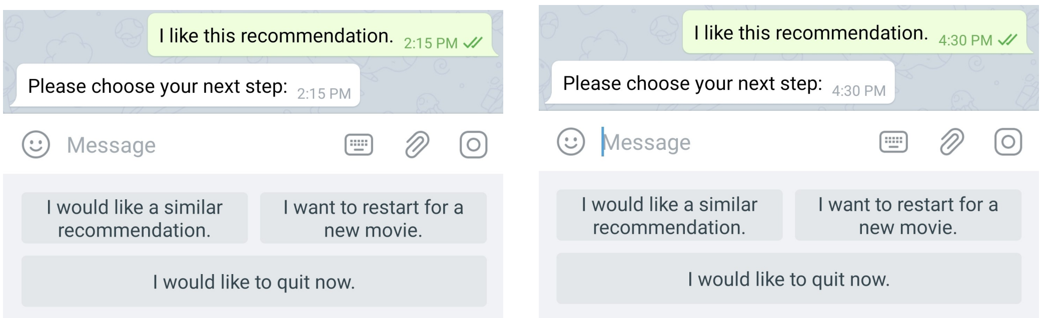
In Figure 4.9, the user accept the recommended movie. The intent is predefined by the keyboard button (Table 4.2). Here the user must be provided with options to continue the conversation. Therefore, the agent is not supposed to perform any complex computation and this step does not take more time to execute (Table 4.3).

If the user wants a similar recommendation (Figure 4.10), the processing of this intent in different for both versions. As the NLU returns a list of similar movies with the dialogue act (Table 4.2), this tasks takes a longer time than in version 1 (Table 4.3). However, this is a one time process and the agent stores this list for the scenario where the user does not accept the first similar title. In Figure 4.10, the agent response is also different for both versions.



(a) Version 1 (b) Version 2

Figure 4.8: Top row: User has watched the recommended movie and gets another recommendation. **Bottom row**: User does not like the movie and gets another recommendation.

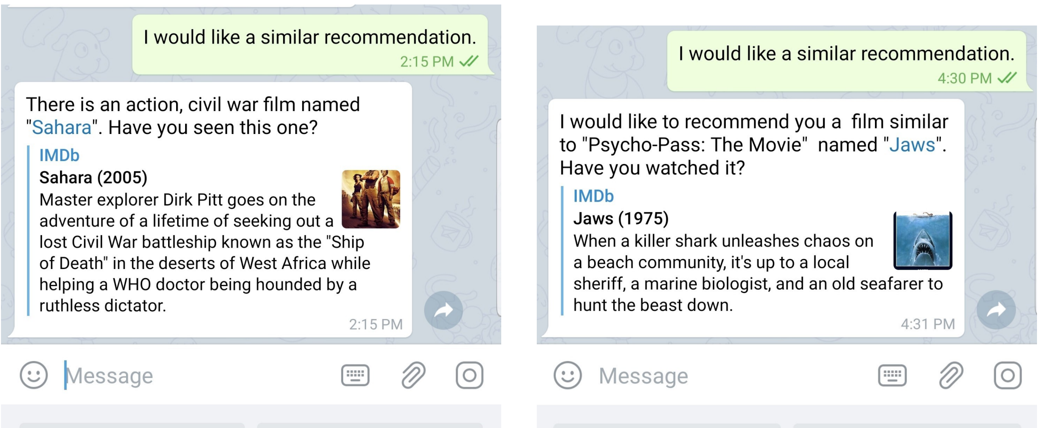


(a) Version 1 (b) Version 2

Figure 4.9: Use likes the recommended movie without inquiring about it.

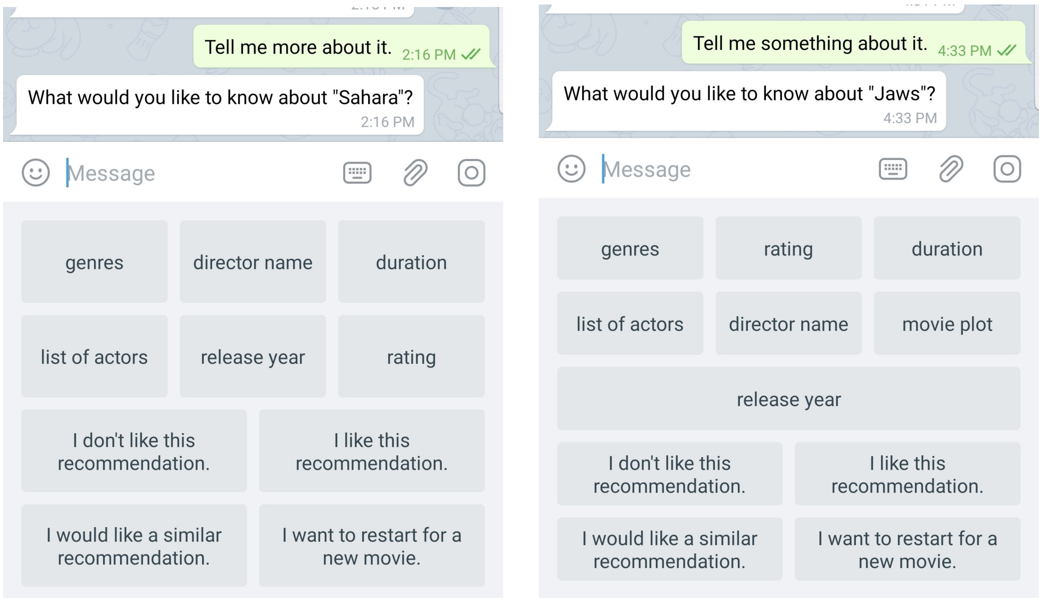
In Figure 4.11 and 4.12, the user inquires about the recommended movie. The dialogue act for one example is show in Table 4.2 and such queries take minimal time as shown in Table 4.3. In Figure 4.12, if the user inquires about a movie attribute, this attribute will be removed from the keyboard buttons until a new movie is recommended.

In Figure 4.7, user can only remove a preference by clicking a keyboard button. This



(a) Version 1 (b) Version 2

Figure 4.10: User would like a similar recommendation.

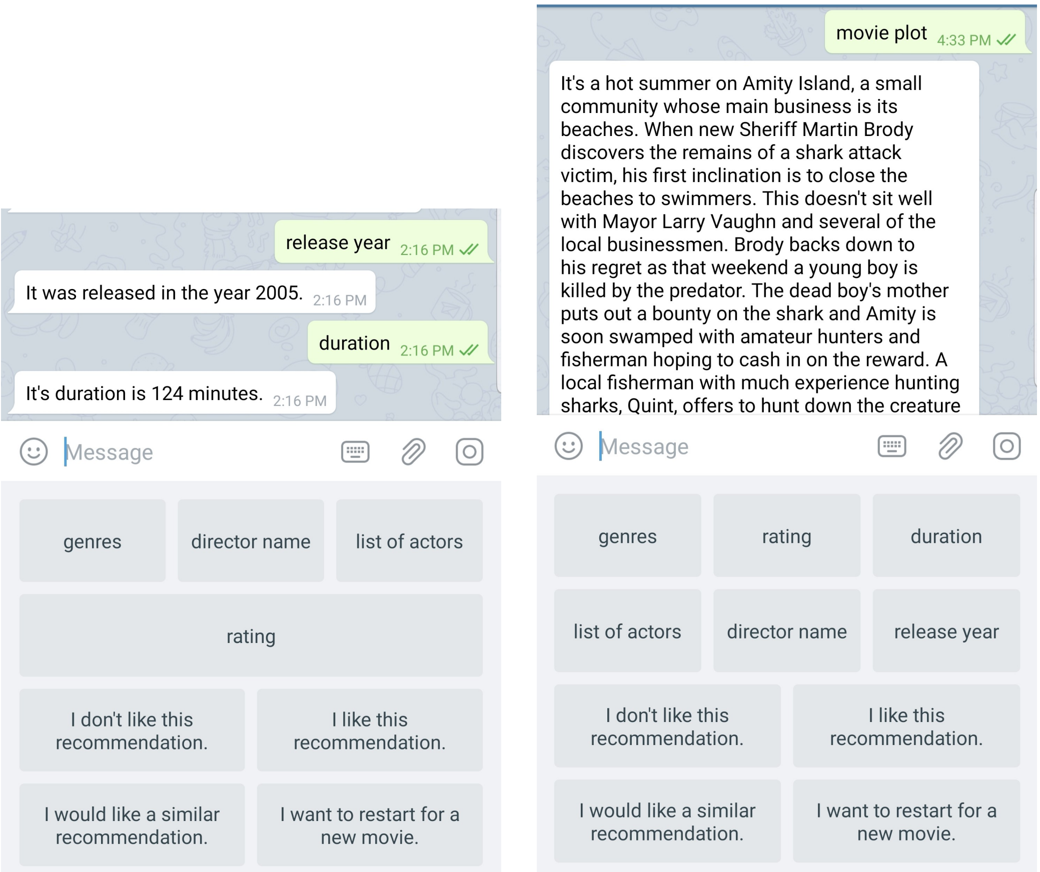


(a) Version 1 (b) Version 2

Figure 4.11: User inquires about the recommendation and is provided with options to ask about.

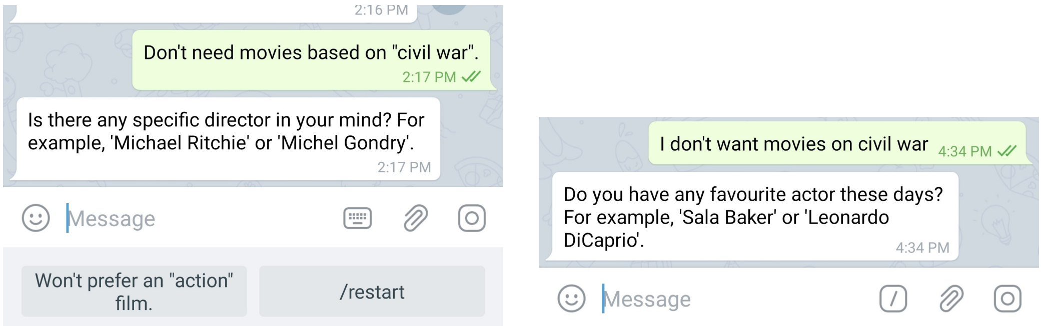
is because, the detection of negative preference is not implemented in the first version. However, in Figure 4.13, it can be seen that the user can remove preference by writing a similar response and does not need keyboard buttons anymore.

Figure 4.14 shows a simple conversation that can be executed in both Version 1 and Version 2. However, there are some improvement in Version 2 where the agent understands the user utterance better. For example, in Figure 4.15, 4.16 and 4.17, the user wants to reveal a negative sentiment for the preference. It can be seen that the agent in Version 1, does not understand such preferences and tries to find the annotates results in the database. In Figure 4.16, if the user had mentioned "directed by Angelina Jolie", the



(a) Version 1 (b) Version 2

Figure 4.12: User has selected some parameters to inquire about.

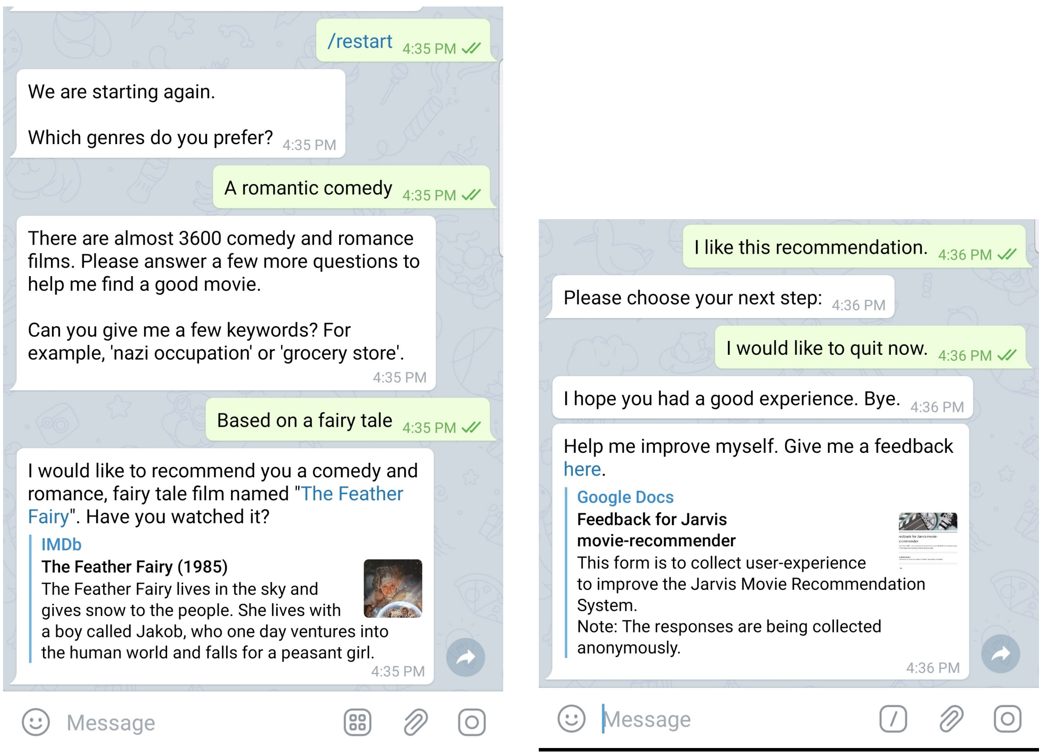


(a) Version 1 (b) Version 2

Figure 4.13: User removes a preference by clicking a button in Version 1 and writing a preference in Version 2.

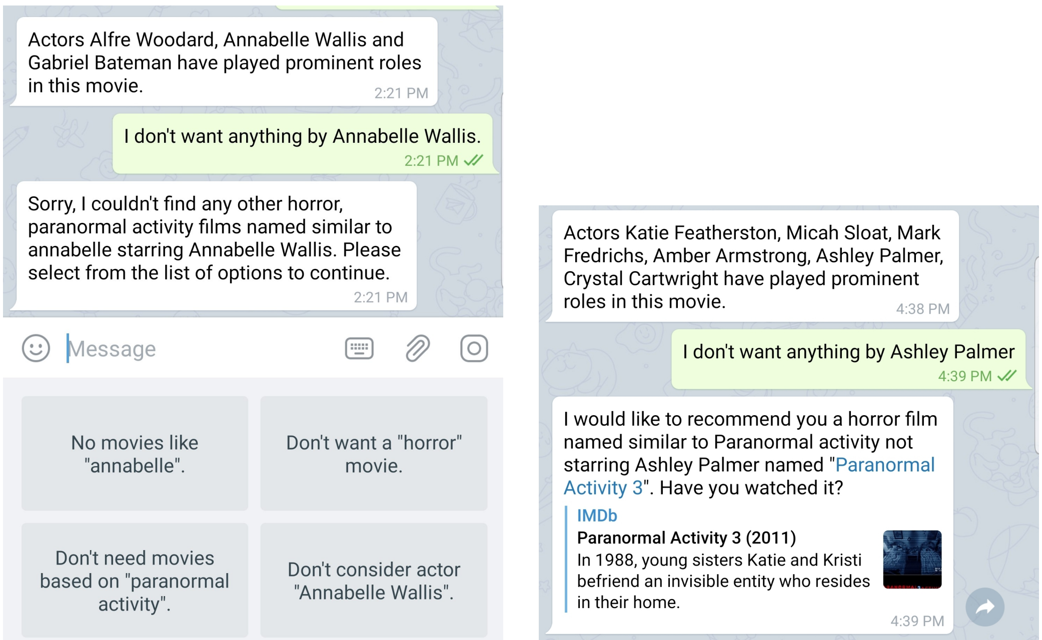
agent would not show keyboard buttons to choose either actor or director. The NLU will calculate that the user does not want the movies *directed* by the person. The dialogue acts and the agent's response time is mentioned in Tables 4.2 and 4.3.

Moreover, in Figure 4.18, the agent in Version 1 does not understand the term "new" or "old" movie. Similarly, if the agent in Version 1 asks the user if they have watched a



(a) Part 1 (b) Part 2

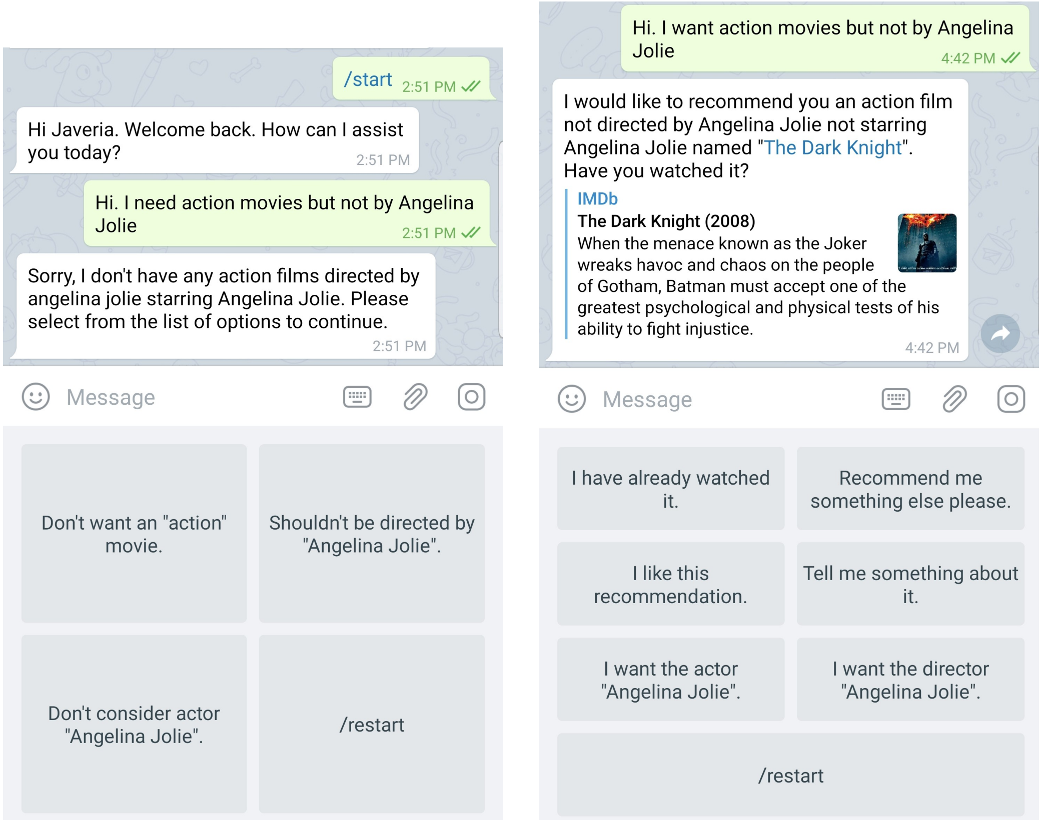
Figure 4.14: Sample conversation in Jarvis



(a) Version 1 (b) Version 2

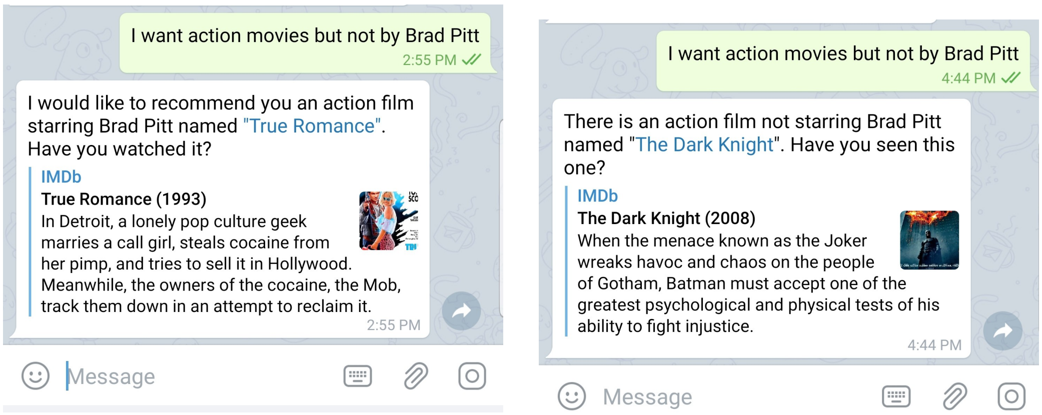
Figure 4.15

movie, it does not understand responses except for the keyboard options shown in Figure 4.7. However, such utterances are being detected for intents in Version 2 (Figure 4.19).



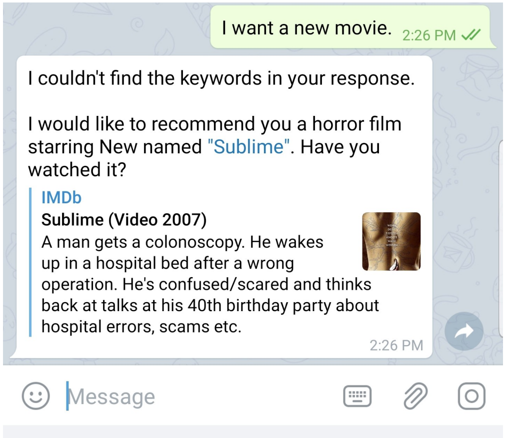
(a) Version 1 (b) Version 2

Figure 4.16

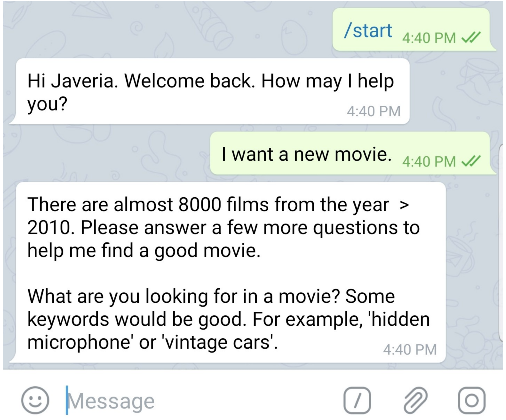


(a) Version 1 (b) Version 2

Figure 4.17



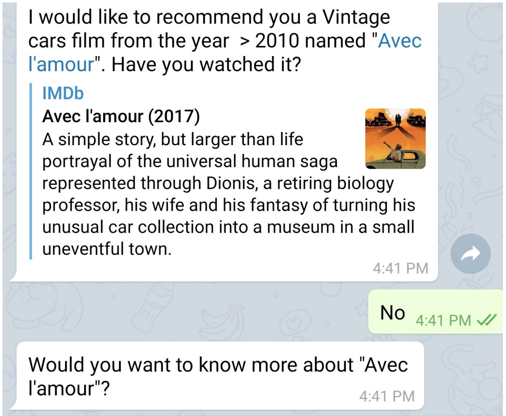
(a) Version 1



(b) Version 2

Figure 4.18

(a) Version 1



(b) Version 2

Figure 4.19

4.3.2 An example of user-case in Jarvis

In this section, an example of a conversation by one of the users is presented. Before adding this conversation, the consent of the user is acquired.

U_1 : Hi

A_1 : Hi Fatima. Welcome back. How can I assist you today?

U_2 : I want suggestions for science fiction movie

A_2 : There is a science fiction film named "**Fantastic Planet**". Have you seen this one?

U_3 : Tell me more about it.

A_3 : What would you like to know about "Fantastic Planet"?

U_4 : movie plot

A_4 : Slaves and masters dominate the narrative of the faraway world of Ygam. Set around the lifespan of Terr, a minute human shaped Om slave, and pet, of the giant blue alien Draags. Escaping into the wilderness and with a device used for intellectual advancement of the Draags, Terr finds refuge and support from fellow Oms and using the learning tool, he finds that knowledge is power and then sets to use the new found knowledge to revolt against the Draag masters...

U_5 : I don't like this recommendation.

A_5 : I would like to recommend you a science fiction film named "**The Fabulous Baron Munchausen**". Have you watched it?

U_6 : Recommend me something else please.

A_6 : There is a science fiction film named "**Adele's Dinner**". Have you seen this one?

U_7 : I would like to watch virus related movies.

A_7 : I would like to recommend you a virus film named "**Blood Brother**". Have you watched it?

U_8 : Tell me something about it.

A_8 : What would you like to know about "Blood Brother"?

U_9 : Plot

A_9 : Blood Brother is an intimate portrait of Rocky Braat, a young man who longed to find a family. He didn't know it, but this desire would lead him to an AIDS hostel in India, a place of unspeakable hardship, where he would find almost more love and need than he could bear.

U_{10} : release year

A_{10} : The movie was released in 2013.

U_{11} : rating

A_{11} : The rating of this movie on IMDb is 8.3.

U_{12} : I would like a similar recommendation.

A_{12} : There is a film similar to "Blood Brother" named "**No Country for Old Men**". Have you seen this one?

U_{13} : Recommend me something else please.

A_{13} : I would like to recommend you a film similar to "Blood Brother" named "**O Brother, Where Art Thou?**". Have you watched it?

U_{14} : I have already watched it.

A_{14} : I would like to recommend you a film similar to "Blood Brother" named "**Superbad**". Have you watched it?

U_{15} : Rating?

A_{15} : Its rating on IMDb is 7.6.

U_{16} : movie plot

A_{16} : Seth and Evan are best friends, inseparable, navigating the last weeks of high school. Usually shunned by the popular kids, Seth and Evan luck into an invitation to a party, and spend a long day, with the help of their nerdy friend Fogell, trying to score enough alcohol to lubricate the party and inebriate two girls, Jules and

Becca, so they can kick-start their sex lives and go off to college with a summer full of experience and new skills. Their quest is complicated by Fogell's falling in with two inept cops who both slow and assist the plan. If they do get the liquor to the party, what then? Is sex the only rite of passage at hand?

U_{17} : I want to restart for a new movie.

A_{17} : Let's restart. Do you have any specific genres in mind?

U_{18} : Emotional

A_{18} : I could not find the genres Emotional. There is a Emotional film named "**Now, Voyager**". Have you seen this one?

U_{19} : Tell me something about it.

A_{19} : What would you like to know about "Now, Voyager"?

U_{20} : movie plot

A_{20} : Boston spinster Charlotte has had her life controlled entirely by her mother; a wealthy mother, Mrs. Henry Vale. Feeling despondent, she's convinced to spend some time in a sanitarium. Soon, Charlotte's transformed into a sophisticated, confident woman. On a cruise to South America, Charlotte meets, and begins a torrid affair with Jerry Durrance, a married architect. 6 months later, she returns home, and confronts her mother with her independence. One day, after a brief argument, her mother has a heart attack and dies. Charlotte inherits the Vale fortune but feels guilty for her mother's death. She returns to the sanitarium, where she befriends a depressed, young adolescent, named Tina. The young girls' depression brought on by having been rejected by her mother; Charlotte's former lover, Jerry's wife. Charlotte takes Tina home to Boston with her

U_{21} : I don't like this recommendation.

A_{21} : I would like to recommend you a Emotional film named "**The Swimmer**". Have you watched it?

U_{22} : Tell me something about it.

A_{22} : What would you like to know about "The Swimmer"?

U_{23} : movie plot

A_{23} : Neddy Merrill has been away for most of the Summer. He reappears at a friend's pool. As they talk, someone notices that there are pools spanning the entire valley. He decided to jog from pool to pool to swim across the whole valley. As he stops in each pool his interactions tell his life story.

U_{24} : rating

A_{24} : Its rating on IMDb is 7.7.

U_{25} : I like this recommendation.

A_{25} : Please choose your next step:

U_{26} : I would like to quit now.

A_{26} : I hope you had a good experience. Bye.

This conversation provides an example of how a user changes their requirements and gets multiple recommendation during the conversation. For example, at the user response U_7, the user changed their preference from a science fiction movie to a virus related film. The agent's adapted to the changing user preferences and recommended a different movie (response A_7).

4.3.3 Analysis

According to the Table 4.2, it is analyzed the quality of the conversation of Jarvis movie recommendation system has improved in the second version. The NLU of the Jarvis is more capable of understanding user's information needs at every stage in the dialogue. The conversation examples show that Jarvis is able to adapt to the complex conversational scenarios. The agent response time in Table 4.3 shows a significant improvement in computational speed of the agent and this can further improve the user's experience with Jarvis.

Chapter 5

Conclusion and Future Directions

In this chapter, a summary and future prospects of the study are presented. In Section 5.1, a summary of the work is presented with an explanation of the RQs in Chapter 1. In Section 5.2, the future direction for the word done are reported.

5.1 Conclusion

Conversation recommender systems (CRS) are widely used by users to find a suitable recommendation. CRSs converse with the users in their language, ask questions to find a recommendation. In this thesis, a conversational movie recommender system named Jarvis is proposed. Jarvis is a python based open-source system and offers a scaleable structure where each component is easily distinguishable for future implementations. The primary goal of Jarvis is to understand users' preferences, adapt to their changing requirements, recommend a movie, and improve its performance based on the feedback from the user. The user feedback gives an overview of how well the system performs and what can be improved further. Moreover, it is essential to make the system user-friendly.

For the development of Jarvis, the objectives of the system and a dialogue-flow is designed. The objectives define the functionalities in Jarvis that must be achieved for a good conversation. These functionalities include entity recognition, sentiment analysis, intent detection, conversational history analysis, and implementation of user feedback for further improvements. The dialogue-flow contains the possible changes in the conversation based on the user's feedback for a recommendation made and how the user wants to continue a conversation. Using the dialogue-flow as a guide, a list of intents for both agent and use is created.

The first version of Jarvis included the development of the component according to the dialogue-flow. The objectives of this version exclude sentiment analysis. To test the performance and user-satisfaction in Jarvis, it is used by fifteen participants. The conversation of the users is recorded to study the performance of Jarvis. To gather a feedback from the participants, quality attributes are designed, and users are requested to rate the system on these attributes. Moreover, users also commented on what they liked and what can be improved in the system.

The second version is an upgrade of Jarvis in version 1. This version also includes sentiment analysis. The feedback received in the previous version is used to improve the component of Jarvis. Moreover, the recorded conversational data is analyzed to improve the flow of the conversation. The significant improvements include a better entity recognition, sentiment analysis, a database with higher coverage of movies, and faster and improved agent response even to unexpected instances. Like the first version, Jarvis version 2 is also used by fifteen users, and their feedback reflects that version two is better than version 1.

The research questions proposed in Chapter 1 are answered as follows:

RQ1 Is it possible to implement a complex dialogue-flow using only a rule-based conversational agent?

Both versions of Jarvis follow a complex dialogue-flow mentioned in Figure 3.2. Although, using a rule-based approach makes it a challenging task to consider all possible scenarios while taking the next action for the agent. However, considering the lack of available conversational data on movie recommendation, it is not possible to train the agent. Therefore, creating a rule-based agent proved to be a successful approach.

RQ2 Can a rule-based Natural Language Understander (NLU) be created to understand users' preferences for a movie?

In Jarvis, the slot-annotation and intent-detection are rule-based, and it is proved that the rule-based NLU can understand the user requirements in an efficient manner (Table 4.2).

RQ3 Is it possible to create a database that contains a high coverage of possible users' preferences?

The feedback and conversation-data in version 2 prove that the database has a higher coverage of user preference. In the database, different approaches to generate keywords for the movies are implemented. The results of the approaches are analyzed to ensure the quality of the database (Section 3.3.1.I).

RQ4 Can Jarvis meets the expectations of the users and helps them reach their goals easily?

It is a learning process in Jarvis to improve and meet the expectations of the users. However, it can be seen that user satisfaction has improved for the second version, and the users can get quality recommendations (Chapter 4).

5.2 Future directions

In this thesis, a conversation movie recommendation system is developed while creating a database that contains specific attributes for movie preferences. The system is being continuously upgraded by analyzing user conversations. In the future, the following approaches can be applied to Jarvis

- Based on the user feedback, some improvements can still be implemented in Jarvis. For example, the agent must list multiple recommendations in a single response. There is a need to design intents and change the dialogue-flow for this feature.
- The list of users for the second version has almost the same users in the first. New users must be selected for testing version 2 to include variance in the feedback.
- The agent can be trained for the intent-detection and decision-making process. For this purpose, a large corpus is required.

Bibliography

[1] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1): 36–45, 1966.

[2] Kenneth Mark Colby, Sylvia Weber, and Franklin Dennis Hilf. Artificial paranoia. *Artificial Intelligence*, 2(1):1–25, 1971.

[3] Michael Jugovac and Dietmar Jannach. Interacting with recommenders - overview and research directions. *TiiS*, 7(3):10:1–10:46, 2017. doi: 10.1145/3001837. URL `https://doi.org/10.1145/3001837`.

[4] Stefan Ultes, Lina Maria Rojas-Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Pawel Budzianowski, Nikola Mrksic, Tsung-Hsien Wen, Milica Gasic, and Steve J. Young. Pydial: A multi-domain statistical dialogue system toolkit. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, System Demonstrations*, pages 73–78, 2017. doi: 10.18653/v1/P17-4013. URL `https://doi.org/10.18653/v1/P17-4013`.

[5] Pierre Lison and Casey Kennington. Opendial: A toolkit for developing spoken dialogue systems with probabilistic rules. In *Proceedings of ACL-2016 System Demonstrations, Berlin, Germany, August 7-12, 2016*, pages 67–72, 2016. doi: 10.18653/v1/P16-4012. URL `https://doi.org/10.18653/v1/P16-4012`.

[6] Alexander H. Miller, Will Feng, Dhruv Batra, Antoine Bordes, Adam Fisch, Jiasen Lu, Devi Parikh, and Jason Weston. Parlai: A dialog research software platform. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017 - System Demonstrations*, pages 79–84, 2017. URL `https://www.aclweb.org/anthology/D17-2014/`.

[7] Alexandros Papangelis, Yi-Chia Wang, Piero Molino, and Gokhan Tur. Collaborative multi-agent dialogue model training via reinforcement learning. *arXiv preprint arXiv:1907.05507*, 2019.

[8] Jeffrey Dalton, Victor Ajayi, and Richard Main. Vote goat: Conversational movie recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 1285–1288, 2018. doi: 10.1145/3209978.3210168. URL `https://doi.org/10.1145/3209978.3210168`.

[9] Jianfeng Gao, Michel Galley, and Lihong Li. Neural approaches to conversational AI. *CoRR*, abs/1809.08267, 2018. URL `http://arxiv.org/abs/1809.08267`.

[10] Sungjin Lee, Qi Zhu, Ryuichi Takanobu, Zheng Zhang, Yaoqin Zhang, Xiang Li, Jinchao Li, Baolin Peng, Xiujun Li, Minlie Huang, and Jianfeng Gao. Convlab: Multi-domain end-to-end dialog system platform. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28 - August 2, 2019, Volume 3: System Demonstrations*, pages 64–69, 2019. URL `https://www.aclweb.org/anthology/P19-3011/`.

[11] Daniel Jurafsky and James H Martin. Dialog systems and chatbots. *Speech and language processing*, 3, 2017.

[12] Iulian Vlad Serban, Ryan Lowe, Peter Henderson, Laurent Charlin, and Joelle Pineau. A survey of available corpora for building data-driven dialogue systems. *CoRR*, abs/1512.05742, 2015. URL `http://arxiv.org/abs/1512.05742`.

[13] S. Young, M. Gašić, B. Thomson, and J. D. Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

[14] Harald Aust, Martin Oerder, Frank Seide, and Volker Steinbiss. The philips automatic train timetable information system. *Speech Communication*, 17(3-4):249–262, 1995. doi: 10.1016/0167-6393(95)00028-M. URL `https://doi.org/10.1016/0167-6393(95)00028-M`.

[15] Allen L. Gorin, Giuseppe Riccardi, and Jeremy H. Wright. How may I help you? *Speech Communication*, 23(1-2):113–127, 1997. doi: 10.1016/S0167-6393(97)00040-X. URL `https://doi.org/10.1016/S0167-6393(97)00040-X`.

[16] Daniel G Bobrow, Ronald M Kaplan, Martin Kay, Donald A Norman, Henry Thompson, and Terry Winograd. Gus, a frame-driven dialog system. *Artificial intelligence*, 8(2):155–173, 1977.

[17] Jason L. Hutchens and Michael D. Alder. Introducing megahal. In *Proceedings of the Joint Conference on New Methods in Language Processing and Computational Natural Language Learning, NeMLaP/CoNLL 1998, Macquarie University, Sydney, NSW, Australia, January 11-17, 1998*, pages 271–274, 1998. URL `https://www.aclweb.org/anthology/W98-1233/`.

[18] Alan Ritter, Colin Cherry, and William B Dolan. Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*, pages 583–593. Association for Computational Linguistics, 2011.

[19] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*, 2015.

[20] Lifeng Shang, Zhengdong Lu, and Hang Li. Neural responding machine for short-text conversation. *arXiv preprint arXiv:1503.02364*, 2015.

[21] Joseph Weizenbaum. ELIZA - A computer program for the study of natural language communication between man and machine (reprint). *Commun. ACM*, 26(1):23–28, 1983. doi: 10.1145/357980.357991. URL `https://doi.org/10.1145/357980.357991`.

[22] Rollo Carpenter. Cleverbot, 2012, 2018.

[23] Li Zhou, Jianfeng Gao, Di Li, and Heung-Yeung Shum. The design and implementation of xiaoice, an empathetic social chatbot. *CoRR*, abs/1812.08989, 2018. URL `http://arxiv.org/abs/1812.08989`.

[24] Kenneth Mark Colby, Franklin Dennis Hilf, Sylvia Weber, and Helena C Kraemer. Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes. *Artificial Intelligence*, 3:199–221, 1972.

[25] Iulian Vlad Serban, Ryan Lowe, Peter Henderson, Laurent Charlin, and Joelle Pineau. A survey of available corpora for building data-driven dialogue systems: The journal version. *Dialogue & Discourse*, 9(1):1–49, 2018.

[26] John J Godfrey, Edward C Holliman, and Jane McDaniel. Switchboard: Telephone speech corpus for research and development. In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 517–520. IEEE, 1992.

[27] Pierfranca Forchini. Using movie corpora to explore spoken american english. *Variation and Change in Spoken and Written Discourse: Perspectives from corpus linguistics*, 21:123, 2013.

[28] Pierre Lison and Jörg Tiedemann. Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles. 2016.

[29] Alan Ritter, Colin Cherry, and Bill Dolan. Unsupervised modeling of twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 172–180. Association for Computational Linguistics, 2010.

[30] Karthik Gopalakrishnan, Behnam Hedayatnia, Qinlang Chen, Anna Gottardi, Sanjeev Kwatra, Anu Venkatesh, Raefer Gabriel, Dilek Hakkani-Tür, and Amazon Alexa AI. Topical-chat: Towards knowledge-grounded open-domain conversations. *Proc. Interspeech 2019*, pages 1891–1895, 2019.

[31] Sina Jafarpour, Christopher JC Burges, and Alan Ritter. Filter, rank, and transfer the knowledge: Learning to chat. *Advances in Ranking*, 10:2329–9290, 2010.

[32] Anton Leuski and David Traum. Npceditor: Creating virtual human dialogue using information retrieval techniques. *Ai Magazine*, 32(2):42–56, 2011.

[33] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[34] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[35] F Rosenblatt. Principles of neurodynamics (spartan, new york, 1962). *Google Scholar*.

[36] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[37] Patrick Haffner, Gokhan Tur, and Jerry H Wright. Optimizing svms for complex call classification. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 1, pages I–I. IEEE, 2003.

[38] Ciprian Chelba, Milind Mahajan, and Alex Acero. Speech utterance classification. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 1, pages I–I. IEEE, 2003.

[39] Roberto Pieraccini, Evelyne Tzoukermann, Zakhar Gorelov, J-L Gauvain, Esther Levin, C-H Lee, and Jay G Wilpon. A speech understanding system based on statistical representation of semantics. In *[Proceedings] ICASSP-92: 1992 IEEE*

International Conference on Acoustics, Speech, and Signal Processing, volume 1, pages 193–196. IEEE, 1992.

[40] Ye-Yi Wang, Li Deng, and Alex Acero. Spoken language understanding. *IEEE Signal Processing Magazine*, 22(5):16–31, 2005.

[41] Christian Raymond and Giuseppe Riccardi. Generative and discriminative algorithms for spoken language understanding. In *Eighth Annual Conference of the International Speech Communication Association*, 2007.

[42] Ruhi Sarikaya, Geoffrey E Hinton, and Bhuvana Ramabhadran. Deep belief nets for natural language call-routing. In *2011 IEEE International conference on acoustics, speech and signal processing (ICASSP)*, pages 5680–5683. IEEE, 2011.

[43] Gokhan Tur, Li Deng, Dilek Hakkani-Tür, and Xiaodong He. Towards deeper understanding: Deep convex networks for semantic utterance classification. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5045–5048. IEEE, 2012.

[44] Ruhi Sarikaya, Geoffrey E Hinton, and Anoop Deoras. Application of deep belief networks for natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(4):778–784, 2014.

[45] Suman Ravuri and Andreas Stolcke. Recurrent neural network and lstm models for lexical utterance classification. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[46] Puyang Xu and Ruhi Sarikaya. Convolutional neural network based triangular crf for joint intent detection and slot filling. In *2013 ieee workshop on automatic speech recognition and understanding*, pages 78–83. IEEE, 2013.

[47] Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. Recurrent neural networks for language understanding. In *Interspeech*, pages 2524–2528, 2013.

[48] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539, 2014.

[49] Yun-Nung Chen, Asli Celikyilmaz, and Dilek Hakkani-Tur. Deep learning for dialogue systems. In *Proceedings of the 27th International Conference on Computational Linguistics: Tutorial Abstracts*, pages 25–31, 2018.

[50] Esther Levin, Roberto Pieraccini, and Wieland Eckert. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on speech and audio processing*, 8(1):11–23, 2000.

[51] Marilyn A Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12:387–416, 2000.

[52] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research*, 16:105–133, 2002.

[53] Jason D Williams and Steve Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422, 2007.

[54] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5): 1160–1179, 2013.

[55] Dietmar Jannach, Ahtsham Manzoor, Wanling Cai, and Li Chen. A survey on conversational recommender systems, 2020.

[56] Jian-Yun Nie Hua Wu Haifeng Wang Zhanyi Liu, Zheng-Yu Niu. Conversation in ir: its role and utility. In *Proceedings of SIGIR 1st International Workshop on Conversational Approaches to Information Retrieval (CAIR'17), CAIR*, volume 17, 2017.

[57] Diane Kelly, Jaime Arguello, and Robert Capra. Nsf workshop on task-based information search systems. In *ACM SIGIR Forum*, volume 47, pages 116–127. ACM New York, NY, USA, 2013.

[58] Filip Radlinski and Nick Craswell. A theoretical framework for conversational search. In *Proceedings of the 2017 conference on conference human information interaction and retrieval*, pages 117–126, 2017.

[59] Johanne R Trippas, Damiano Spina, Lawrence Cavedon, Hideo Joho, and Mark Sanderson. Informing the design of spoken conversational search: Perspective paper. In *Proceedings of the 2018 Conference on Human Information Interaction & Retrieval*, pages 32–41, 2018.

[60] Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. Towards conversational recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 815–824, 2016.

[61] Andi K Winterboer, Martin I Tietze, Maria K Wolters, and Johanna D Moore. The user model-based summarize and refine approach improves information presentation in spoken dialog systems. *Computer Speech & Language*, 25(2):175–191, 2011.

[62] Vera Demberg, Andi Winterboer, and Johanna D Moore. A strategy for information presentation in spoken dialog systems. *Computational Linguistics*, 37(3):489–539, 2011.

[63] Vera Demberg and Johanna D Moore. Information presentation in spoken dialogue systems. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

[64] Marilyn A Walker, Rebecca Passonneau, and Julie E Boland. Quantitative and qualitative evaluation of darpa communicator spoken dialogue systems. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 515–522. Association for Computational Linguistics, 2001.

[65] Leif Azzopardi, Mateusz Dubiel, Martin Halvey, and Jeffery Dalton. Conceptualizing agent-human interactions during the conversational search process. In *The Second International Workshop on Conversational Approaches to Information Retrieval*, 2018.

[66] Fedelucio Narducci, Pierpaolo Basile, Andrea Iovine, Marco de Gemmis, Pasquale Lops, and Giovanni Semeraro. A domain-independent framework for building conversational recommender systems. In *Proceedings of the Workshop on Knowledge-aware and Conversational Recommender Systems 2018 co-located with 12th ACM Conference on Recommender Systems, KaRS@RecSys 2018, Vancouver, Canada, October 7, 2018*, pages 29–34, 2018. URL `http://ceur-ws.org/Vol-2290/kars2018_paper6.pdf`.

[67] Alexandros Papangelis, Yi-Chia Wang, Piero Molino, and Gökhan Tür. Collaborative multi-agent dialogue model training via reinforcement learning. *CoRR*, abs/1907.05507, 2019. URL `http://arxiv.org/abs/1907.05507`.

[68] Nicole M. Radziwill and Morgan C. Benton. Evaluating quality of chatbots and intelligent conversational agents. *CoRR*, abs/1704.04579, 2017. URL `http://arxiv.org/abs/1704.04579`.

[69] Alain Abran Witold, Witold Suryn, Adel Khelifi, Juergen Rilling, Ahmed Seffah, and Franois Robert. Consolidating the iso usability models. In *In submitted to the 11th International Software Quality Management Conference and the 8th Annual INSPIRE Conference*. Citeseer, 2003.