Conversational Used Car Price Predictor

CS702 - Computing Lab

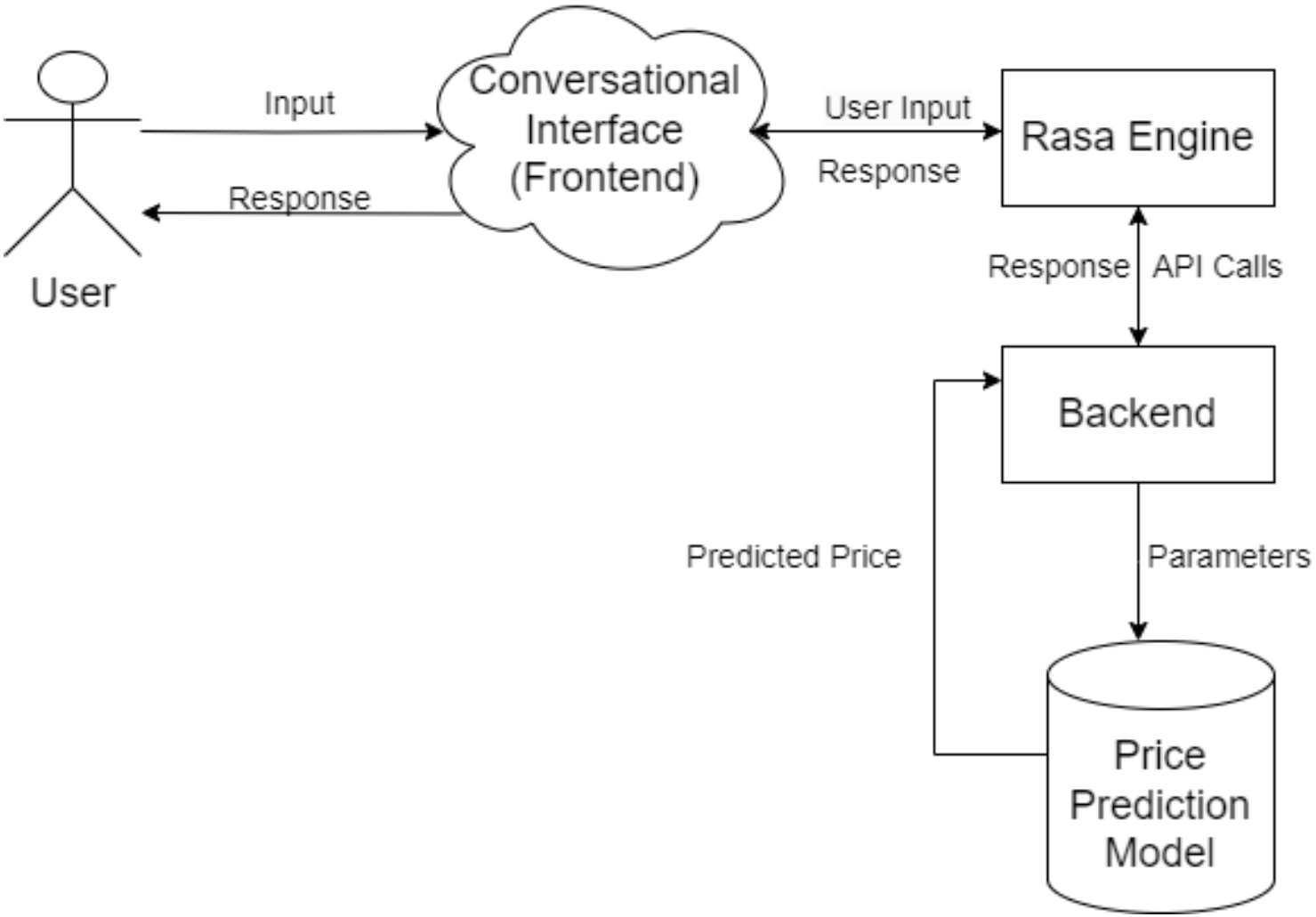
ABHIJITH C ANAND M K

Department of Computer Science and Engineering
NITK Surathkal

Introduction

- This project focuses on developing a **Conversational Used Car Price Predictor**, integrating a chatbot interface with a machine learning model.
- **Goal:** To allow users to interact through natural conversation rather than filling out traditional forms to predict used car prices.
- The chatbot will collect necessary car details (brand, model, year, mileage, etc.) step by step through an intuitive and engaging interface.
- A machine learning model will use the gathered data to predict the price of the used car, ensuring accurate and reliable predictions.
- The chatbot also handles additional queries, such as explaining how the price was calculated or what factors affect the car's value.

System Architecture



,

Project Overview: Progress and Next Steps

- **Milestones Achieved:**
  - Data collection and preprocessing for the car price prediction model.
  - Model development using **Random Forest** and SHAP calculation for feature contribution analysis.
  - Integration of the backend with Rasa.
  - Development of a sample chatbot to collect car parameters from the user.
  - Validation of user input parameters in the Rasa chatbot.
- **Upcoming Work:**
  - Improve the Rasa chatbot by adding more test data.
  - Handle general questions in Rasa.

Backend API Creation

- **Framework Used:** Flask
  - A lightweight web framework for Python, used to create web applications.
- **Endpoints:**
  - `/predict_price`:
    - Accepts car attributes as query parameters.
    - Returns the predicted selling price of the car.
  - `/max_contribution`:
    - Accepts the same car attributes as query parameters.
    - Returns the highest contributing feature to the predicted price and its percentage contribution.
- **Input Handling:**
  - Retrieves input data via query parameters (GET API).
- **Response Format:**
  - Outputs predictions and contributions as JSON, facilitating easy integration with frontend applications.

Intents in NLU

- **greet** - Intent for greetings
- **goodbye** - Intent for farewells
- **affirm** - Intent for confirming or agreeing
- **deny** - Intent for negating or disagreeing
- **bot_challenge** - Intent for asking about the nature of the bot
- **inform** - Intent for providing car details like brand, model, mileage, etc.
- **stop** - Intent to stop the conversation
- **ask_shap** - Intent to ask about factors impacting price

Entities in NLU

- **brand** - represents the car brand (e.g., Toyota, Ford, etc.)
- **model** - represents the car model (e.g., Fortuner, Swift, etc.)
- **mileage** - represents the car's mileage (e.g., 15 km/l)
- **km_driven** - represents the distance the car has been driven (e.g., 10000 km)
- **fuel_type** - represents the type of fuel the car uses (e.g., petrol, diesel)
- **transmission_type** - represents the transmission type of the car (e.g., manual, automatic)
- **engine** - represents the engine capacity of the car (e.g., 1500 cc)
- **max_power** - represents the car's maximum power (e.g., 150 bhp)
- **seats** - represents the seating capacity of the car (e.g., 5 seats)
- **year_of_manufacture** - represents the car's year of manufacture (e.g., 2020)

Entity Extraction using Regex and Lookup Tables

- **Regex Patterns for Entity Extraction:**
  - **km_driven**
  - **mileage**
  - **engine**
  - **max_power**
  - **seats**
  - **year_of_manufacture**
  - `Example:\b{1,2}\b(?=\s(kmpl|km\l|km per liter)\b)`
    15 kmpl, 18 km/l
- **Lookup Tables for Entity Extraction:**
  - **brand**: Maruti, Hyundai, Ford, Renault, Mini, Mercedes-Benz, etc.
  - **model**: Alto, Grand, i20, Ecosport, Wagon R, i10, Venue, Swift, etc.
  - **transmission_type**: manual, automatic
  - **fuel_type**: petrol, diesel, electric

Form: car_details_form

- The form defines set of required slots that need to be filled.
- When form is active, the system will ask the user for details (one by one), and as the user provides answers, the values will populate the respective slots.
- Form deactivates after all slots are filled.
- **Required Slots**:
  - `brand`, `model`, `km_driven`, `mileage`, `fuel_type`,
  - `transmission_type`, `engine`, `max_power`, `seats`, `year_of_manufacture`
- **Slot Mappings**:
  - Each slot corresponds to an entity, e.g., `brand` is mapped to the `brand` entity.
  - These mappings allow automatic extraction from user input.

Actions

- **utter_slots_values**: Confirms the filled slot values.
- **utter_greet**: Greets the user and starts the conversation.
- **utter_goodbye**: Ends the conversation.
- **utter_iamabot**: Informs the user that the assistant is a virtual bot.
- **utter_ask_brand**, **utter_ask_model**, . . . : Asks the user for specific slot values like brand, model, mileage, etc.
- **action_clear_slots**: Clears the filled slots.
- **action_predict_car_price**: Triggers the car price prediction.
- **utter_please_rephrase**: Asks the user to rephrase if the input is unclear.
- **action_max_contribution**: Returns the feature with the highest contribution to the price prediction.
- **validate_car_details_form**: Validates filled slots to ensure correct information.

ValidateCarDetailsForm - Slot Validation

- **Brand & Model**: Fuzzy matching with predefined lists of car brands/models.
- **Numeric Slots**: Checks ranges for km driven, mileage, engine capacity, max power, seats, and year of manufacture.
- **Fuel Type & Transmission**: Validates against allowed values like `petrol`, `diesel`, `automatic`, etc.

Telegram Integration

- Rasa provides an API endpoint (default: `http://localhost:5005/webhooks/telegram/webhook`) to handle incoming messages.
- The endpoint allows Rasa to receive and send messages to Telegram through webhooks.
- Telegram bot integration:
  - Create a bot using `BotFather` on Telegram and obtain a token.
  - Set the Telegram bot token in Rasa configuration `credentials.yml`.
- Rasa listens to messages from users on Telegram, processes them, and sends back responses via the webhook.

Upcoming Work

- Improve the Rasa chatbot by adding more test data to enhance its understanding and responsiveness.
- Handle general questions within the chatbot to improve user engagement and satisfaction.

References

- Rasa Technologies, "Rasa Documentation," *Rasa*, 2023. [Online]. Available: https://rasa.com/docs/
- Armin Ronacher, "Flask Documentation," *Flask*, 2023. [Online]. Available: https://flask.palletsprojects.com/
- SHAP Documentation, "SHAP (SHapley Additive exPlanations)," 2023. [Online]. Available: https://shap.readthedocs.io/en/latest/
- Leo Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, 2001, pp. 5-32. [Online]. Available: https://link.springer.com/article/10.1023/A:1010933404324
- Scikit-learn, "Random Forests," 2023. [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Thank You!