README.md

Lowes_Campus_Hackathon

Solution to Problem Theme #2.

Build a conversational solution that enables customers to discover and order products

This is a developmental project that acts as a solution to problem 2 of this competition.

Tools & Services Used

The various tools and services used include

- AWS Elastic Beanstalk
- Google Dialogflow
- Kommunicate

Steps of Development

Step 1 : Data Scraping

Approximately 100 rows of Product Data was scraped using a python script that made use of Selenium.

This is the csv file that was obtained after scraping the data.

This is the python script that handled data scraping.

These lines of data concentrate on only the following goods that Lowe's Sells on its website

- Appliances
  - Refrigerators
  - Washing Machines
  - Microwaves
  - Dishwashers
  - Oven / Ranges
  - Driers
- Home-Decor
  - Area Mats
  - Bathroom Mats
  - Rugs

NOTE: The values of Prices (Discounted and Current) are dummy values. This had to be done due to inconsistencies.

Step 2 : Create a SQLite3 Database and Insert Scraped Data

The motivation to choose SQLite3 is due to the fact that the list of products always increases, and we needed a mechanism to enable that products get added. Along with this, we felt the need for a querying mechanism. This is help us answer more advanced questions from the user.

Additionally, it seemed to match the requirements for an Open Source RDBMS as in the Rules, and had a vast community and good support.

This is the python scipt that enabled creation of the SQLite3 database.

The **initial_data_dump_in_db()** was the responsible method.

Good database practices were followed while CREATEing and INSERTing values into the database, such as making it SQL-injection proof.

Step 3 : Perform String Parsing on Individual Product Descriptions

Each product description is parsed and is normalized by using regular expressions.

This is done to ensure that natural language processing happens with utmost ease.

For example:

A product that has a description

```
"Samsung 24.52-cu ft Side-by-Side Refrigerator with Ice Maker (Stainless Steel)"
```

will become

```
22 23 24 25 26 Samsung cubic feet Side by Side Refrigerator with Ice Maker Stainless Steel
```

This python script handles the string parsing on the product descriptions.

The **parse_string()** function was the responsible method.

Step 4 : Generating Synsets for each stemmed / lemmatized string

Each string in a product description is stemmed and then fed to a NLP based synset generator.

A synset is a group of words that are similar to a given word. This is not contextualized.

This synset can be used for a hash-table data-structure for fast retrieval of data.

This python script handles the both the stemming and synset generation of individual words in the parsed product descriptions.

The **all_syno()** method was the responsible method.

Step 5 : Create a hash-table data-structure

For the efficient retrieval and verification of query strings, a hash-table is an ideal data structure.

A garbage value that is guarenteed to have no adverse affects on the hash-table, is initially pushed into it.

Soon after this, the hash-table is pickled (saved), as `product_synonym_hash.pkl` .

This ensures that it can be used further on.

This python script handles the creation of the hash-table.

The **create_hash_map()** method was the responsible method.

Step 6 : Update hash-table with all synsets of all processed product descriptions

The pickled hash-table is loaded into the memory.

Synsets of each string in the processed product description, are found and updated in the hash-table.

Soon after this, the hash-table is pickled (saved), as `product_synonym_hash.pkl` .

This ensures that it can be used further on.

This python script handles the updation of the hash-table.

The **update_hash_map()** method was the responsible method.

Step 7 : Create mechanism to update db and update hash-table simultaneously

A flask application **app.py**, was created, and a template inde.html is linked along with the database table **inventory.db**.

A trigger is set, for updating hash-table with inputs, as soon as inputs are sent into db for insertion.

This python script is the Flask Web Application.

The **update_db()** method was the responsible method.

This python script contains the code responsible for triggering the updating of the hash table.

The **dynamic_data_dump()** method was the responsible method.

Step 8 : Create API around accessing hash-table and finding most similar products

A REST-API is created around code that accepts a string or customer query and returns JSON for 2 of the most similar products.

The 2 most similar products are found, and their corresponding asset_links, ratings, dummy prices, website links, and descriptions are generated in JSON.

This python script contains the code responsible for creating the API around processing the input and delivers JSON.

The **general_search()** method was the responsible method.

Step 9 : Ensure that Dialogflow bot has above API method present in Fulfillment, and data is recieved and displayed on iframe, or window.

Prerequisites

This package assumes you use Python 3.x.

Expected package dependencies are listed in the "requirements.txt" file for PIP, you need to run the following command to get dependencies:

```
pip install -r requirements.txt
```

Installation

clone this repo

To update database, run app.py in the Scripts folder, and insert values into template.

To just use chatbot, open trial_indian_web.html

Contributing

1. Fork this project.
2. Commit your changes.
3. Create a new Pull Request and link an issue with it.

Meta

Follow 23

Name	Github	LinkedIn	E-mail	Phone
Viswalahiri Swamy Hejeebu	@viswalahiri	@viswalahiri	E-mail	(+91) 630-152-9655
Kurapati Abhilash	@abhilash	@abhilash	E-mail	(+91) 850-072-8944

Distributed under the MIT Liscence. See `LICENSE` for more information.