Problem Context:

Every statement user speaks that could be each utterance. For example : “Do you know chatbot and show me recent growth statistics of chatbot” the entire sentence is the utterance. An intent is the user intention. For example if the user says “Show me recent growth statistics” the user intent is to retrieve a list of statistical headlines.

An entity modifies an intent. For example if user types “show me recent growth statistics of chatbot” the entities are “recent” and “statistics” and entities are sometimes referred to as slots.

Do you know chatbot and show me recent growth statistics of chatbot-> utterance.

- Show me ->Intent - something needs to be shown.
- Growth statistics -> Entity that modifies the intent.

Problem statement:

Not every entity serves to identify an intent and ability to identify which one is valid or which one is not valid. This major problem can be solved using chatbots. Hence this task is to create the API’s which a chatbot could easily refer while identifying the entities.

We can assume that the validation criteria to be already provided some other service. Our task is to validate the entities based on this criteria. Here we need to validate the entire finite set and validate a slot with numeric values.

Intention validator:

This project intendes to validates intention composed of entities behind the utterance. It's a basic Django app which has three POST Apis

- validate_finite_entity : https://localhost:8000/validateFiniteEntity
- validate_num_entity: https://localhost:8000/validateNumEntity

Assumptions:

- If pick_first is present, it will simply overide the behaviour of support_multiple key and first value will be picked regardless the entity value is valid or not
- Constraint is considered to be boolean or bad request response is thrown.
- If users enter garbage values or irrelevant information then our API's will throw the exceptions either it will show it as a payload or bad request or data types are incorrect it may be null values or required information is missing.

Run Automated Tests

You can run the already pre-built tests for the API's to test sanity functions as follows: (It takes around 0.030s to 0.045s)

- **./manage.py test**
- **python manage.py test**

The Future Scenarios and it’s solving ideas:

- Slot validation : The problem may occur regarding the slot validation. We can create another API’s about the slots filled and its details. For such situation the following link can be used to provide the solution: https://github.com/Pavan-GL/IntentValidationApp/tree/main/Solution-Slot

- Data file may be large: Instead of creating the API’s we can train the data by using machine learning concepts it will reduce the time and provide fast response.

- If the user is deaf and dumb: Once the bot is ready the end users will get the result based on the keystroke entered by the user he needs to get the responses. This can be done by implementing the NLP concepts such as:

    1) Bag of Words: Whenever we apply any algorithm in NLP, it works on numbers. We cannot directly feed our text into that algorithm. Hence, Bag of Words model is used to preprocess the text by converting it into a bag of words, which keeps a count of the total occurrences of most frequently used words.

       Step 1 : We will first preprocess the data, in order to:
       Step 2 : Obtaining most frequent words in our text.
       Step 3 : Building the Bag of Words model.

2) Tokenization: Tokenization is breaking a text chunk in smaller parts. Whether it is breaking Paragraph in sentences, sentence into words or word in characters. Hence, tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization.

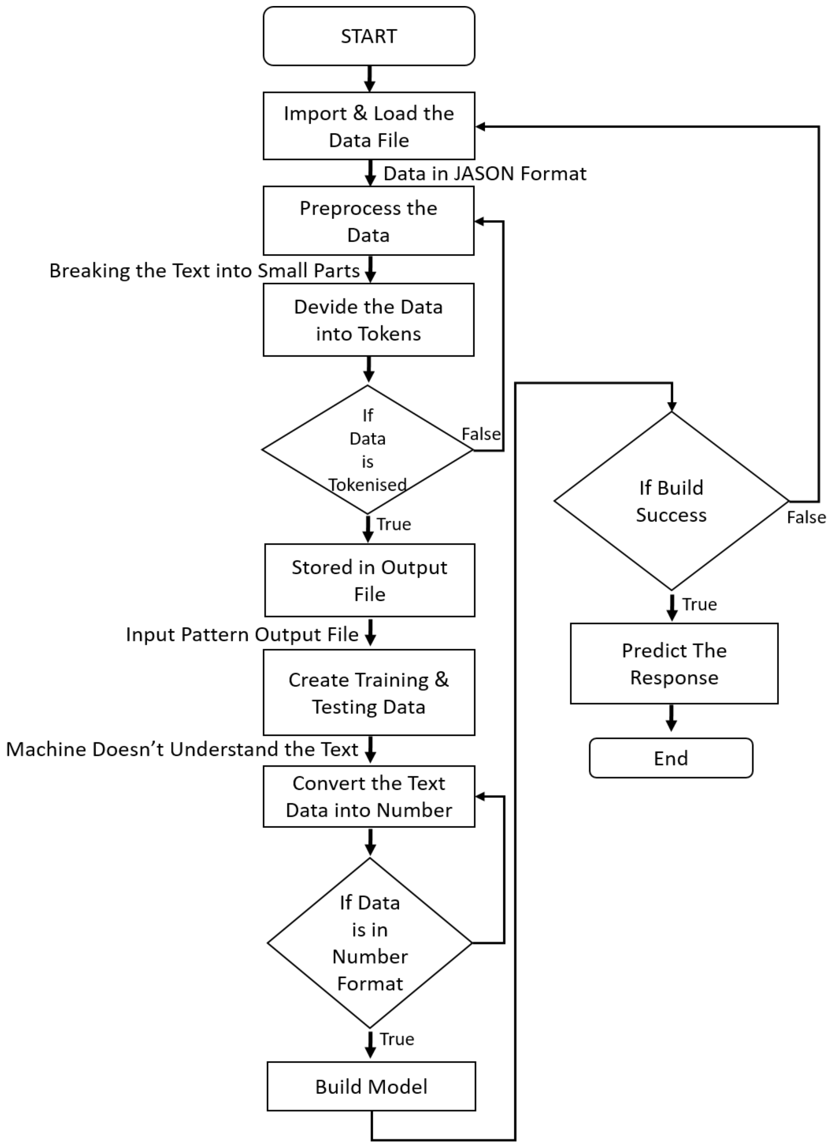
   Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. In Advanced Deep Learning-based NLP architectures, vocabulary is used to create the tokenized input sentences. Finally, the tokens of these sentences are passed as inputs to the model.

3) Stemming : Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in natural language understanding (NLU) and natural language processing (NLP).Stemming is also a part of queries and Internet search engines.

   A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”. Stemming is an important part of the pipelining process in Natural language processing.

   https://github.com/Pavan-GL/IntentValidationApp/blob/main/Solution-Slot/nltksolution.py

Sample Flow chart of a Bot:


START
Import & Load the Data File
Data in JASON Format
Preprocess the Data
Breaking the Text into Small Parts
Devide the Data into Tokens
If Data is Tokenised
False
True
Stored in Output File
Input Pattern Output File
Create Training & Testing Data
Machine Doesn't Understand the Text
Convert the Text Data into Number
If Data is in Number Format
True
Build Model
If Build Success
False
True
Predict The Response
End

For references : https://github.com/Pavan-GL/AI-Chatbot-Using-Pytorch