Documentation

Chatbot

Objective: -

To cater to the customer query/FAQs on a website/forums that are asked frequently.

Proposal: -

To develop a chatbot which would be able to answer the queries of the users visiting the website.

Technologies Used: -

Flask – For front end of the web application

Python Scripts – For backend, to act as the processing unit of the chatbot.

AWS /Azure – For deploying the chatbot online.

Libraries Used: -

NLTK, flask, Numpy, Sklearn, IO, Random, string and warnings.

Theory: -

For the chatbot to understand what the meaning of the input is, Natural Language Processing is important (NLP). NLP is the ability of a computer to understand the meaning of the human language as it is spoken. It is a component of Artificial Intelligence. It can be thought of the translator which translates the human language to one which computers can understand. For Python, Natural Language Tool Kit (NLTK) is the most widely used library to work on human language data. In other words, NLTK is a library used for NLP in Python. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum. It also takes care of the intricacies of the NLP. As already discussed NLTK already comes with corpuses loaded into which can be used as the “training data” for the bot to learn the kind of answers to give to user’s input.

The main idea behind the working of the chatbot is to find cosine similarity between the user's input and training corpus. The training corpus that is going to be used as the training data.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The formula for cosine similarity is

$$A \cdot B = \|A\|\|B\| \cos\theta$$

$$\cos\theta = \frac{A \cdot B}{\|A\|\|B\|}$$

Since the cosine similarity formula makes use of vectors so we need to convert the input into the form of vectors. So, the input is converted is converted into vectors and then the cosine similarity is calculated with the words in corpus, the sentence which has the highest similarity is returned as the response.

Algorithm: -

To setup NLTK for future use, we need to download packages that are necessary for basic functionality of the library. So, the command

nltk.download('popular', quiet=True)

does the work.

The algorithm works in the following way : -

- If the script is being run for the first time, then the training corpus and word tokenizer has to be downloaded. For reading corpuses "WordNet" has been used and "punkt" has been used as the tokenizer
- Since the chatbot is going to specifically answer to questions addressed to "bot" or "chatbot" we need a corpus specific for the purpose. Thus, a text document is used as the training corpus
- The training corpus is read and converted into lower case
- The input is first tokenized into sentences and then those sentences are further tokenized into words
- As part of the data pre-processing we "lemmatize" the words. Lemmatization takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma.

Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.
For example, runs, running, ran are all forms of the word run, therefore run is the lemma of all these words. Because lemmatization returns an actual word of the language, it is used where it is necessary to get valid words.

- Some general inputs and responses are defined to include inputs and responses which may have been skipped by chance in the training corpus or the training data.
- Now, for the heart of the algorithm, the response function.
    - The input tokens are tokenized using Lemmatizer and then fed to a TF-IDF Vectorizer.
        - TF IDF stands for Term Frequency - Inverse Document Frequency. TF IDF is a crude but effective way of distilling documents down to a reasonable and manageable abstraction.
        - TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:
          TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).
        - IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones, by computing the following:
          IDF(t) = log_e(Total number of documents / Number of documents with term t in it).
          For detailed explanation refer http://www.tfidf.com/

- The next step is to learn vocabulary and the IDF and the returned matrix is saved in a vector.
- Once we have everything prepared we find the cosine similarity between the input and the learnt vocabulary
- The returned value is then flattened since the return type of cosine similarity is a kernel matrix and then sorted out.
- Since the sorting arranges the similarity in an increasing order the last element is fetched and given as the response.

Web application: -

The web application has been developed in flask using the above-mentioned algorithm in another script as the backend. The flask makes use of the HTML form to fetch a text and when submitted the text in field provided calls a function which displays the user's input and bot's response onto the HTML page if it is a valid text.

Deployment: -

Currently the code has been deployed on an Amazon EC2 Linux free tier eligible server. The IP to access the chatbot is http://13.127.58.101:8080 and the service is up 24x7. The server can be accessed in different ways mentioned in the link https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html

Future Work: -

Currently the deployment is on AWS services, for future expansion, the same code could be published onto Azure App Service. The steps differ depending upon the Operating System the developer is using; Windows and Linux help guide link have been provided below.

https://docs.microsoft.com/en-us/visualstudio/python/publish-to-app-service-windows?view=vs-2019

https://docs.microsoft.com/en-us/visualstudio/python/publishing-python-web-applications-to-azure-from-visual-studio?view=vs-2019

For a consolidated guide, refer the following git guide

https://github.com/Cojacfar/FlaskWeb

Also, right now the NLTK library is being used as NLP guide for the chatbot, so the working algorithm could be changed to a generative-based RNN chatbot to see whether the performance change based on the change.

To get a clear idea about RNN chatbots and what a generative based chatbot is refer below link.

https://medium.com/@BhashkarKunal/conversational-ai-chatbot-using-deep-learning-how-bi-directional-lstm-machine-reading-38dc5cf5a5a3

https://medium.com/botsupply/generative-model-chatbots-e422ab08461e

If there is any feedback, please contact the undersigned

Anindya Vedant

Developer

anindya.vedant@sourcefuse.com

SourceFuse LLC, Noida