Chatbot Recommendation System

# Introduction :

Recommender systems are one of the most successful and widespread application of machine learning technologies in business. Machine learning algorithms in recommender systems are typically classified into two categories — content based and collaborative filtering methods although modern recommenders combine both approaches. Content based methods are based on similarity of item attributes and collaborative methods calculate similarity from interactions.

# Tools Used :

This project had been built using the Python language. I had used Python’s module ‘Chatterbot’ for creating the chatbot and Mongo DB is used for the storage purpose. Following the tools used in the project :

1. Chatterbot
2. nltk
3. sklearn
4. re
5. pandas
6. Mongo DB
7. logging
8. google search

# Working of the Project :

This project is designed to recommend the user a solution for a particular query which the user faces. For this to happen the training data is passed on to the chatterbot as a csv or a yml file and is read by the python using the Pandas. Once the data is passed to the chatterbot, the chatterbot gets trained through it. As the training goes, we used Mongo DB for storing it at the backend. Once the chatterbot was properly trained, the next part dealt with the Machine learning.

Next, we worked on the Natural Language Processing (NLP), by which our aim was to refine the query asked by the user and find out its contextual meaning by discarding stop words and punctuations from the text. For this to happen, we brought in the NLTK module of Python, which is used for executing the NLP concepts. The foremost part was performing the tokenization of the text, depending on our priorities whether we want it to be word tokenized or sentence tokenized. Tokenization basically divides the text in the list format, in which each of its element is reffered to as token. Using the NLTK, we implemented Stemming and Lemmatization which helped our bot to improve the accuracy by not getting puzzled over the different forms of a word; and by focussing on the morphological meaning of the word rather than being just textual. However, among Stemming and Lemmatization we went with Lemmatization as Stemming sometimes changed the root meaning of the word. So, lemmatization is done now and to make our text be more precise we made the text case sensitive and also removed the punctuations as they are not required for the further processing of the text. Now there are stopwords which don’t hold any importance in this project and if removed could made the text more specific. So, NLTK provides a corpus which hold a list of all the stopwords and using it we made our text free from stopwords.

Uptill here, query is refined now to match it with the data which we are having we first had to made changes to the dataset too. For this again we transformed our dataset using NLTK, and accuracy was increased a lot. We also had to change the format in which Mongo DB stored the data; and for that we created our own customized Mongo DB adapter for the chatterbot. After certain ammendments the responses were quite accurate. This accuracy was also attained by using certain Logic adapters for the chatterbot like, Best match adapter, Specific Response adapter, etc. ChatterBot uses Statement objects to hold information about things that can be said. An important part of how a chat bot selects a response is based on its ability to compare two statements to each other. There are a number of ways to do this, and ChatterBot comes with a handful of methods built in for us to use. Some of them are:

1. JaccardSimilarity
2. Sentiment Comparison
3. levenshtein distance

However, we used levenshtein distance for the similarity because it used 65% of similarity threshold among the statements.

After the comparison is done, we made a feedback system for the chatterbot, in which when a query is asked, the chatterbot answers it to the user and if the answer or solution to the query is inappropriate then the user can feed in the wright solution to the problem which gets stored in the database, so that in future it could response accurately. While doing so, there is an attribute in chattebot named ‘read\_only’, it is assigned ‘False’ argument so that whatever is typed could be stored too. Otherwise it is kept ‘True’.

Now all this accuracy can be measured too by using the response.confidence attribute. It gives the confidence on a scale of 0-1, and helps us to decide the accuracy level. It could also be increased by using the Logic adapters mentioned above.

For other NLP concepts, we used another module of Python named ‘scikit-learn’ or ‘sklearn’. Through them we implemented tf-idf and Multinomial naive bayes which converts a collection of raw documents to a matrix of TF-IDF features.TFIDF select the features by weighting the most

frequently used terms in a single document and penalising the terms that are used in a wide spectrum of documents.

* What is Multinomial NB?

Multinomial Naive Bayes is a specialized version of Naive Bayes that is designed more for text documents. Whereas simple naive Bayes would model a document as the presence and absence of particular words, multinomial naive bayes explicitly models the word counts and adjusts the underlying calculations to deal with in.

Hence, after all this our chatterbot is ready to take on any query within its bound and will try to answer it from the database. Firstly, if the response stored in the dataset to an asked query, based on the description given by the user has a satisfactory confidence level then the chatbot will show the reponse to the user else, it will find the contextual meaning of the query and will again check whether its response’s confidence level meets our needs or not and if it is satisfactory then it will display it, otherwise a provision had been made where the chatterbot will fetch top 5 links from Google to help user in solving the query.