Patient-Doctor Chatbot Documentation

Project Overview

This project aims to develop a chatbot system for a patient-doctor interaction, where the patient's query is processed to find the most relevant response from a dataset of medical queries and answers. The project uses text preprocessing, TF-IDF-based similarity, and Word2Vec-based embeddings (Skip-Gram and CBOW) to determine the best response from the doctor.

1. Dependencies and Libraries

This project utilizes several Python libraries for natural language processing (NLP), machine learning, and visualization:

- **NLTK**: Used for text preprocessing, such as tokenization, stemming, and stopword removal.
- **Word2Vec (from Gensim)**: For training word embeddings (Skip-Gram and CBOW models).
- **TF-IDF Vectorizer (from sklearn)**: For computing TF-IDF matrices and similarity.
- **Cosine Similarity (from sklearn)**: To measure the similarity between the query and responses.
- **Matplotlib/Seaborn**: For visualizing the results (word clouds, frequency distributions, etc.).
- **WordCloud**: To generate word clouds based on unigrams, bigrams, and trigrams.

2. Data Preprocessing

Data preprocessing includes several text cleaning and normalization steps:

Functions:

- `tokenize_text(text)`: Tokenizes the input text into a list of words by splitting the string at spaces.
- `preprocess_text(text)`:
    - Converts text to lowercase.
    - Removes URLs and special characters (using regular expressions).
    - Removes stopwords (common words like "the", "is", etc.).
    - Applies stemming using the Porter Stemmer to reduce words to their root form.

Input:

- A raw text string (e.g., patient query).

Output:

- A list of preprocessed tokens (words).

3. Visualization

Word Frequency Visualization:

- `create_n_grams(df, n)`: Generates n-grams (unigrams, bigrams, trigrams) from the processed data in the dataframe `df` and saves them to CSV files.
- `plot_frequent_ngrams(unigram_file, bigram_file, trigram_file)`: Plots the top 10 most frequent unigrams, bigrams, and trigrams using bar plots.
- `word_cloud(filename, col)`: Generates a word cloud for a specific n-gram type (unigrams, bigrams, or trigrams).

4. TF-IDF Based Similarity

The TF-IDF approach converts the text into numerical vectors representing term frequencies across the dataset and compares them using cosine similarity.

Functions:

- `recommend_tfidf(query)`:
    - Takes a patient query as input.
    - Converts the query into a TF-IDF vector.
    - Finds the most similar responses in the dataset based on cosine similarity with the query.
    - Returns the most relevant doctor’s response.

Input:

- A patient query string.

Output:

- The doctor’s response corresponding to the most similar query in the dataset.

5. Word2Vec Based Similarity

Skip-Gram and CBOW Models:

- **Skip-Gram**: A Word2Vec model where the focus is on predicting the context (neighboring words) from a target word.
- **CBOW (Continuous Bag of Words)**: A Word2Vec model that predicts a target word from its surrounding context.

Functions:

- `convert_to_embeddings(query, model)`:
  - Converts a query sentence into its corresponding embedding vector using the specified Word2Vec model (Skip-Gram or CBOW).
- `recommend_word2vec(query, df)`:
  - Converts the query into embeddings using both Skip-Gram and CBOW models.
  - Computes cosine similarity between the query embeddings and the embeddings of the responses in the dataset.
  - Returns the most similar sentences (queries) and their corresponding responses from both models.

Input:

- A patient query string.

Output:

- The most similar sentences (queries) and their corresponding responses using both CBOW and Skip-Gram models.

6. Cosine Similarity Calculation

Cosine similarity is used to measure the similarity between two vectors (query vector and response vector). The cosine similarity value lies between -1 and 1, where 1 indicates a perfect match.

Functions:

- `find_cosine_similarity(embedded_query, total_embeddings)`: Computes the cosine similarity between the query embedding and all the response embeddings in the dataset.

7. Model Training and Evaluation

Skip-Gram and CBOW Model Training:

- The Word2Vec models (Skip-Gram and CBOW) are trained using the preprocessed text data.
- The trained models are saved as binary files (`skipgram_model_final.bin` and `cbow_model_final.bin`).

TF-IDF Model:

- The TF-IDF vectorizer is used to transform the input text into vectors based on term frequencies and inverse document frequencies.

8. Results and Recommendations

Example Queries and Responses:

Example queries like "I am feeling pain in stomach" are processed, and the most relevant doctor's responses are returned using both the TF-IDF and Word2Vec approaches.

9. Limitations

TF-IDF:

- **Context Loss**: TF-IDF does not maintain the context of the sentence and only focuses on individual words.
- **Sparse Vectors**: It leads to sparse vectors, which may result in poor performance when out-of-vocabulary (OOV) words are encountered.

Word2Vec:

- **Out-of-Vocabulary (OOV) Words**: Words not present in the training data will have zero embeddings.
- **Biased Embeddings**: Word2Vec embeddings may reflect biases in the training data.
- **Computational Complexity**: Training Word2Vec models is computationally expensive, especially for large datasets.

10. Conclusion

The project demonstrates two methods for text-based recommendations in a patient-doctor chatbot system: TF-IDF and Word2Vec. While TF-IDF is faster, Word2Vec maintains better semantic meaning and context. Depending on the query and response, the Word2Vec approach often produces more relevant and contextually accurate responses.