A Project Report

On

QUESTIONS GENERATOR

BY

VEDA MIRACLE PALAPARTHY(SE21UARI188)

SAHITHI KOTAGIRI(SE21UARI122)

SRI SANJANA RAVURI(SE21UARI135)

DEEKSHITA KOMMI(SE21UARI033)

VAISHNAVI NEELA(SE21UARI179)

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

COURSE PROJECT IN AI 3106: Foundations of NLP

ECOLE CENTRALE SCHOOL OF ENGINEERING

HYDERABAD

(December 2023)

PROBLEM STATEMENT

The challenge we're addressing stems from the difficulty of creating comprehensive and appropriate questions for diverse topics. Currently, when formulating questions, there's a struggle to cover all aspects adequately, leading to the risk of missing key details. This limitation hinders effective understanding and learning. The goal is to develop a Solution, a tool, that specifically targets this issue. The tool will be designed to autonomously generate well-suited questions for a wide range of topics. This ensures a more thorough exploration of content, overcoming existing barriers in question generation and making the process more inclusive and adaptable to various subjects. Ultimately, the project seeks to provide a solution that enhances the quality and relevance of questions.

VEDA MIRACLE PALAPARTHY(SE21UARI188) - POS tagging and NER and report

SAHITHI KOTAGIRI(SE21UARI122) – Dataset, Summarization, accuracy

SRI SANJANA RAVURI(SE21UARI135) – Questions generator and accuracy

DEEKSHITA KOMMI(SE21UARI033) - Preprocessing and report

VAISHNAVI NEELA(SE21UARI179) – helped in report

CONTENTS

IMPLEMENTATION

Data sets used:

The dataset comprises articles sourced from the 'papers with code' website, covering various themes and publications. We have specifically chosen the second article, which focuses on the theme of crime.

Preprocessing done on it:

- As a first step, we renamed the 'content' column to 'articles' in our DataFrame.
- We filtered the DataFrame data to include only the columns 'human_summary', 'articles', and 'theme' using the filter method. This was done to narrow down the dataset and retain only the specific information relevant to our analysis or task, streamlining the data for further processing and exploration.
- Checked for empty data by calculating the sum of null values for each column in the DataFrame data using data**.**isnull().sum(). We check for missing information to understand how good the data is. This helps us decide how to deal with any missing values in the data later on. analyses.
- We created a dictionary called contraction_expander that connects common English contractions with their full forms. This helps us replace contractions with grammatically complete versions in the text.
- In our text cleaning process,
  - We first convert all letters to lowercase to maintain consistency.
  - When we encounter shortened forms like "can't," we substitute them with the complete forms, such as "cannot," for a more formal presentation.
  - we use a regular expression to find and remove content enclosed within parentheses, including the parentheses themselves, from the variable content. This is helpful for us in the preprocessing stage, as it allows us to eliminate information that may not be essential for our analysis.
  - We are also replacing occurrences of the sequence "______", "■", "•", "’s", "________ •", with an empty string with an empty string, removing it from the content.
  - Special characters, punctuation like exclamation marks or commas, are removed.
  - The consecutive whitespaces are with a single space and leading/trailing spaces are removed, ensuring consistent spacing.

- The URLs are replaced with a space, cleaning the text data of hyperlinks.

- We go through each summary in the original data's 'human_summary' column, clean it using the cleaning function, and add the cleaned version to our new list. This helps us have a cleaner and more processed set of human summaries for further analysis.
- we're doing a similar process as before. This time, we create a new list called clean_text. For each article in the original data's 'articles' column, we apply the cleaning function to make it cleaner and more suitable for analysis. The cleaned articles are then added to our new list, providing us with a refined set of text data.

Environment:

We have used Google Colab to run our scripts. We importing NumPy and Pandas which are pivotal for numerical operations and data manipulation, respectively. The **time** module is included to handle time-related operations, while the Natural Language Toolkit (NLTK) is introduced for its extensive functionalities in processing human language, covering tokenization, stemming, tagging, and parsing. The **re** module is utilized for regular expression operations, enabling efficient pattern matching and string manipulation. For graph-related tasks, the code imports NetworkX (as **nx**), and for plotting, it includes Matplotlib's **pyplot** module (as **plt**) and Seaborn (as **sns**). The spaCy library, geared towards advanced NLP tasks like named entity recognition, is brought in as **spacy**. The OpenAI library is imported for unspecified functionalities. The code also employs PrettyTable for creating ASCII tables, Summa for text summarization, and various NLTK components for sentiment analysis, including the SentimentIntensityAnalyzer class. Additionally, the fuzzywuzzy library is included for fuzzy string matching. Various NLTK resources are downloaded to support tasks such as tokenization, stop words, part-of-speech tagging, WordNet, and sentiment analysis. Lastly, external modules such as urllib.request and zipfile are imported for handling URLs and extracting data from zip files. We have also downloaded the 'glove.6d' file from the Stanford website to acquire word embeddings.

Proposed Technique / Methodology:

After loading the dataset and importing the required frameworks we preprocess the text. After which we are visualize the distribution of themes in the dataset using a count plot. The 'theme' column is converted to lowercase, and the themes are plotted on the y-axis. The count of occurrences for each theme is represented in the plot, and its title is set to 'Theme Distribution.'

Two new columns, 'clean_articles' and 'clean_summaries,' are added to the DataFrame named 'data.' The values for these columns are assigned from variables named 'clean_text' and 'clean_summary,' respectively. Subsequently, we extract and print the content of the 'clean_articles' column for the second row (index 1) of the DataFrame.We tokenize the text and perform POS tagging with the Universal POS tagset, we are printing the first 30 words according to their categories of POS tags due to space constraint, we can print how many ever we want by altering the parameters. We are using 'WordCloud' to visualize the POS tags.

Next, we load the spaCy English model ('en_core_web_md'), which includes word vectors used for Named Entity Recognition (NER). We perform NER tagging, extracting and printing 30 entities with spaCy, and visualize all the named entities.

We calculate the length of each article and plot the distribution of article lengths. Then, we compute the mean lengths for both articles and summaries, rounding the results to three decimal places. A new DataFrame is created with the mean values, allowing for the analysis of the average length of articles and summaries in the dataset.

Following that, we conduct statistical analysis on the lengths of articles and summaries in the dataset. We calculate specific percentiles (quantiles) for both 'Articles' and 'Summaries' and present the results in a tabular format. The output provides insights into the distribution of lengths, including the median, interquartile range, and extreme percentiles, aiding in the understanding of data variability and central tendency.

To visualize the relationship between the lengths of articles and summaries, a scatter plot with a regression line is created. The R-squared value quantifies the proportion of the variance in 'Summaries' that can be explained by the variance in 'Articles.' A higher R-squared value indicates a stronger relationship. The plot and R-squared value help understand how well the length of an article predicts the length of its corresponding summary.

In the word_embeddings function, we use the GloVe file which has pre-trained word embeddings. This function reads each line from the GloVe text file and then splits it into values, where the first value is the word or key and the remaining values are the components of the vector in the form of an array. This function stored the key and the remaining vector as its value in the word_embeddings dictionary defined above.

The sentence_vector function aims to generate sentence vectors from a list of sentences. First, we iterate through the list of sentences provided, where if the length of each sentence is not zero, it splits the sentence into words and maps the corresponding word embedding vector from the dictionary we created previously, if the word is not found the value of the vector is set to zero. Then we sum up all the vectors and divide the sum by the size of the sentence i.e., number of words and a constant of 0.001 to avoid the division by zero error. This value of the sentence vector is appended to the sentence_vectors list.

We use cosine similarity to analyze the similarity between two input sentences. We create a square matrix of the size of the sentence vectors named sim_mat which stores the value of cosine similarity between any 2 sentences. Every value in the matrix is 0 initially. Then we calculate the similarity value using the cosine_similarity() function. The PageRank algorithm is used to rank the sentences based on the cosine similarity scores and returns the top 5 sentences, which helps us in creating a concise summary. These sentences are meant to capture and communicate the most important and profound aspects of the overall text.

Now to generate the summary we take the first 10 clean_articles and iterates through them, generating summaries for each article using the above-defined functions sentence_vectors, cosine_similarity, and ranking. Then the generated summary is appended to the article_summary list.We print the summaries obtained in the above process.

The extract_keywords function generates keywords using the summa library. In the generate_question function, the text is tokenized into sentences, and questions are generated with the help of defining a template for the questions.

The question model we created lacks grammatical accuracy, and it relies on predefined question templates. That's why we explored an API-based model to gain a more comprehensive understanding of the outcomes.

We have initially set up OpenAI API key to authenticate and authorize the API request which takes user input for the number of questions needed to be generated in the context of the summary. Using “text-davinci-003” engine we generate the questions with respect to the summary of the text.

The techniques used in the project are a mix of older, well-known methods and newer, advanced ones. This mix reflects how the field keeps growing and changing, always looking for better ways to make computers understand and use human language. The project shows that we're always trying to improve how computers understand and generate language, and there's a lot more to explore and discover in this exciting field.

RESULTS AND DISCUSSION

For the model we have developed we are calculating the grammar score (94.00), diversity score(64.5), sentiment score(23.830000000000002), and soft revelance to context that uses fuzzywuzzy’s token_set_ration for soft matching (84.40%) to calculate the performance of our question generator that we have developed with respect to the text we have given. This attempt to build a questions generator was not very successful as the evaluation metric values were low and the questions generated were not of high quality.

So upon much more research about this concept from various sources, we have realized there are better ways to build a questions generator. We have worked on developing a questions generator model by using API whose performance of the questions generated have been calculated by grammar score(94.00), sentiment score(23.83), and revelance to context that uses fuzzywuzzy’s token_set_ration for soft matching(87.40%) which resulted in slightly better performance.

It shows how we can use different methods to analyze and understand text, like figuring out the main topics and themes. The project also gives us useful information about what works well and what challenges we face when we try to teach computers to summarize text or generate questions using traditional methods and modern methods

REFERENCES

APPENDIX A

URL TO COLAB PYTHON SCRIPT:
https://colab.research.google.com/drive/1ePXwSC5lp7WIeqY1Ua4rpdOkdcxKL3Iz?usp=sharing#scrollTo=a8HBNVUqvYaZ

DATASET HAS BEEN UPLOADED IN GITHUB REPOSITORY

LINK TO GITHUB REPO:
https://github.com/Sahithi-kotagiri/Questions-Generator

APPENDIX B