Individual Project Report
Question Generator and Distractor Tool

Kalyani Vinayagam Sivasundari (G27326652)

Introduction

The project aims at developing a toolkit that leverages artificial intelligence to automatically generate questions and corresponding distractors. This toolkit is particularly geared towards supporting educational platforms and learning management systems, enhancing the creation of multiple-choice questions, which are crucial for assessments and learning reinforcements. The project utilizes advanced NLP techniques and machine learning models, focusing particularly on the use of a T5 transformer model, Sense2Vec for semantic similarity, and sentence transformers for context understanding. The overall execution of the project was divided into shared tasks and individual contributions, encompassing the setup of the foundational models, algorithm development, and system integration. My specific contribution focused on the development of the distractor generation algorithm, ensuring its integration and functionality within the broader system framework.

Description of My Individual Work

My individual contribution to the project centered on the development of the distractor generation algorithm, a crucial component of the AI-driven Question Generation and Distractor Toolkit. This algorithm is designed to enhance the educational utility of multiple-choice questions by generating plausible but incorrect options (distractors) that are contextually relevant yet distinct from the correct answers.

Background Information on the Development of the Algorithm

The development of the distractor generation algorithm began with a deep dive into natural language processing (NLP) and semantic similarity techniques. The primary tools employed were:

Sense2Vec: This tool is crucial for retrieving words that are semantically similar to a given target word but sufficiently different to serve as effective distractors. Sense2Vec leverages contextual word relationships derived from a large corpus to suggest alternatives that share common usage patterns.

Sentence Transformers: These models, particularly the "msmarco-distilbert-base-v3", were utilized to understand the contextual relevance of words within the provided text. This ensures that the generated distractors are appropriate within the given educational content's context.

Maximal Marginal Relevance (MMR): This technique was incorporated to select distractors that balance relevance to the content with diversity from each other, thereby reducing redundancy among the options.

Key Equations and Concepts

The distractor generation algorithm uses several key concepts and equations:

Semantic Similarity Calculation:

Semantic similarity is calculated using Sense2Vec, which provides a similarity score based on the contextual usage of words. This score guides the initial selection of potential distractors.

MMR Equation:

The MMR algorithm is defined as follows:

$$MMR \stackrel{def}{=} Arg \max_{D_i \in R \setminus S} \left[\lambda (Sim_1(D_i, Q) - (1-\lambda) \max_{D_j \in S} Sim_2(D_i, D_j)) \right]$$

Maximal Marginal Relevance

```
where, Q = Query (Description of Document category)
 D = Set of documents related to Query Q
 S = Subset of documents in R already selected
 R\S = set of unselected documents in R
 λ = Constant in range [0-1], for diversification of results
```

Code Snippets:

```python
def get_distractors(word, origsentence, sense2vecmodel, sentencemodel, top_n, lambdaval):
    distractors = sense2vec_get_words(word, sense2vecmodel, top_n, origsentence)
    print("distractors: ", distractors)
    if len(distractors) == 0:
        return distractors
    distractors_new = [word.capitalize()]
    distractors_new.extend(distractors)

    embedding_sentence = origsentence + " " + word.capitalize()

    keyword_embedding = sentencemodel.encode([embedding_sentence])
    distractor_embeddings = sentencemodel.encode(distractors_new)

    max_keywords = min(len(distractors_new), 4)
    filtered_keywords = mmr(keyword_embedding, distractor_embeddings, distractors_new, max_keywords, lambdaval)

    final = [word.capitalize()]
    for wrd in filtered_keywords:
        if wrd.lower() != word.lower():
            final.append(wrd.capitalize())
    final = final[1:]
    return final
```

This works as the central tool for the operation of the distractor toolkit. It handles the retrieval and processing of the candidate distractors from the Sense2Vec model outputs.

```python
def sense2vec_get_words(word, s2v, topn, question):
    output = []
    print("word ", word)
    try:
        sense = s2v.get_best_sense(word, senses = ["NOUN", "PERSON", "PRODUCT", "LOC", "ORG", "EVENT", "NORP", "WOKK OF ART", "FAC", "GPE", "NUM", "FACILITY"])
        most_similar = s2v.most_similar(sense, n=topn)
        output = filter_same_sense_words(sense, most_similar)
        # print("Similar ", output)
    except:
        output = []

    threshold = 0.6
    final = [word]
    checklist = question.split()
    for x in output:
        if get_highest_similarity_score(final, x) < threshold and x not in final and x not in checklist:
            final.append(x)

    return final[1:]
```

I utilized Sense2Vec, a model that leverages the semantic similarity between words, to generate initial lists of candidate distractors. This model was particularly useful due to its ability to understand nuanced differences in word usage based on the context, which is critical for the effectiveness of distractors in multiple-choice questions.

```python
def filter_same_sense_words(original, wordlist):
    filtered_words = []
    base_sense = original.split('|')[1]
    # print(base_sense)
    for eachword in wordlist:
        if eachword[0].split('|')[1] == base_sense:
            filtered_words.append(eachword[0].split('|')[0].replace("_", " ").title().strip())
    return filtered_words

1 usage
def get_highest_similarity_score(wordlist, wrd):
    score = []
    for each in wordlist:
        score.append(normalized_levenshtein.similarity(each.lower(), wrd.lower()))
    return max(score)
```

I implemented semantic filtering to refine the list of candidate distractors, ensuring they were similar to the target answer but not correct.

```python
def mmr(doc_embedding, word_embeddings, words, top_n, lambda_param):
    word_doc_similarity = cosine_similarity(word_embeddings, doc_embedding)
    word_similarity = cosine_similarity(word_embeddings)

    keywords_idx = [np.argmax(word_doc_similarity)]
    candidates_idx = [i for i in range(len(words)) if i != keywords_idx[0]]

    for _ in range(top_n - 1):
        candidate_similarities = word_doc_similarity[candidates_idx, :]
        target_similarities = np.max(word_similarity[candidates_idx][:, keywords_idx], axis=1)

        mmr = (lambda_param) * candidate_similarities - (1-lambda_param) * target_similarities.reshape(-1, 1)
        mmr_idx = candidates_idx[np.argmax(mmr)]

        keywords_idx.append(mmr_idx)
        candidates_idx.remove(mmr_idx)

    return [words[idx] for idx in keywords_idx]
```

To balance the relevance and diversity of the proposed distractors, I applied the MMR algorithm. This method helps in reducing redundancy among distractors, ensuring they are sufficiently distinct from each other and the correct answer.

Detailed Description of the Distractor Generation Algorithm

Distractor Generation Process

My work involved the design and implementation of the get_distractors function, which integrates several key technologies:

Sense2Vec Integration:

- **Purpose:** To fetch semantically similar words to the target answer that are contextually appropriate for use as distractors.
- **Implementation:** I utilized the sense2vec_get_words function, which retrieves a list of words similar to the provided target word using the Sense2Vec model. This function filters out the similar words to ensure they share the same semantic sense but are different enough to not be the correct answer.

Sentence Embedding:

- **Purpose:** To encode the original sentence along with the potential distractors to understand the contextual fit of each word.
- **Implementation:** I employed the Sentence Transformer model to create embeddings for both the original sentence appended with the target word and the potential distractors. This step is crucial to analyze how well each distractor fits into the context of the question.

Maximal Marginal Relevance (MMR):

- **Purpose:** To select the most appropriate distractors based on their relevance to the question and diversity from each other.
- **Implementation:** The MMR algorithm is used to balance the relevance and diversity of the distractors. It computes a score for each distractor based on its semantic similarity to the target answer and its dissimilarity to other distractors, ensuring a selection of diverse and contextually fitting distractors.

Code Implementation

Here is a breakdown of the code:

1. **Distractor Retrieval and Processing:**

   After retrieving the candidate distractors using sense2vec_get_words, I check if the list is empty. If not, I proceed by capitalizing the initial word (the target answer) and extending this list with the fetched distractors.

2. **Embedding Calculation:**

   I concatenate the original sentence with the capitalized target word to form a new input sentence. This concatenated sentence and the list of distractors are then encoded using the Sentence Transformer model to obtain their embeddings.

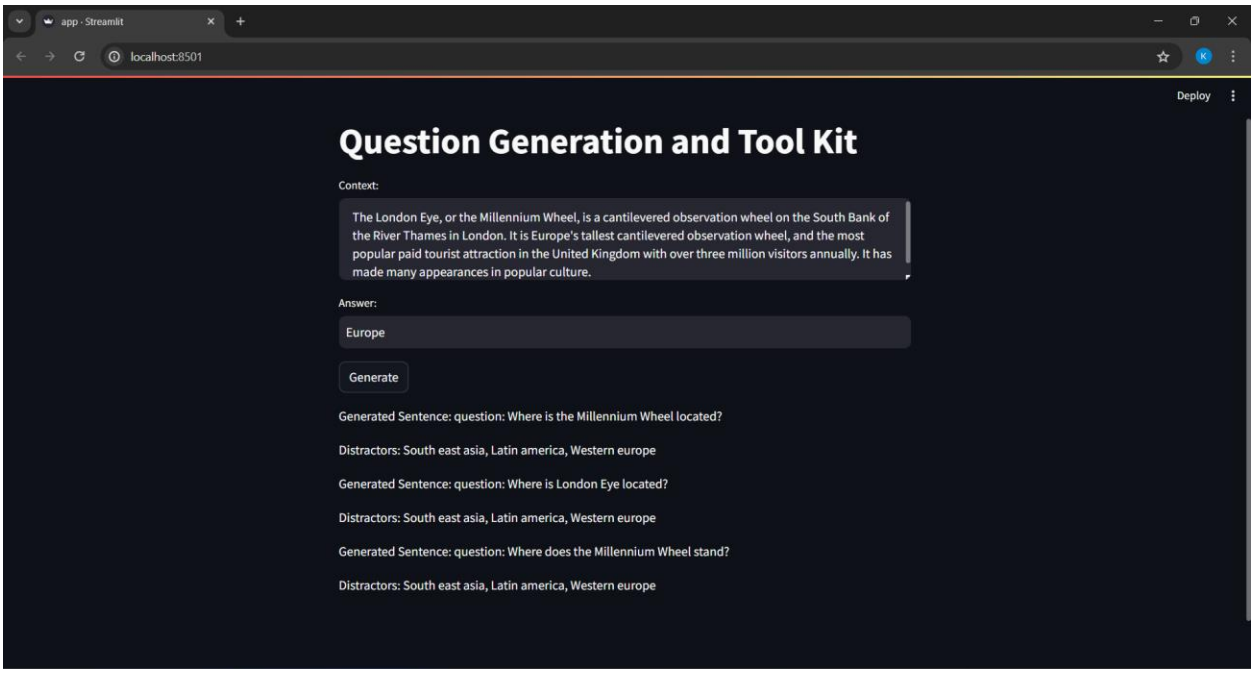
3. **MMR Application:**

   Using the embeddings, I apply the MMR algorithm to select the best four (or fewer) distractors. This involves calculating the cosine similarity between the keyword (target answer) embedding and each distractor embedding, as well as among the distractor embeddings themselves, to find the optimal mix of relevance and diversity.

4. **Final Selection:**

   The final list of distractors is compiled by ensuring that they are not the same as the target word (to avoid the correct answer being listed as a distractor), and they are returned from the function after removing the target word itself to present only the distractors.

Results



Question Generation: The toolkit is capable of generating multiple questions from a single input instance. As shown in the screenshots, for a single input about the London Eye, the model generates various questions focusing on different aspects of the input text, such as the location and the structural context of the London Eye.

Distractor Generation: Alongside the questions, the toolkit generates distractors that are relevant to the context of the question but are carefully selected to be incorrect. This is crucial for the utility of multiple-choice questions in educational settings, where the quality of distractors can significantly impact the learning outcome.

Model Performance: The model effectively interprets and processes textual information to produce outputs that are not only accurate in terms of content but also diverse in their scope. This is evident from the variety of questions generated from a single statement, which indicates robustness and flexibility in the model's operational capabilities.

The analysis of the result demonstrates the fact that system is able to generate contextually accurate and varied questions and distractors.

Summary

The Question Generator and Distractor Toolkit that I developed has successfully demonstrated its ability to automate the creation of multiple-choice questions and corresponding distractors from textual inputs, showcasing the power of advanced natural language processing. The toolkit excels at generating contextually relevant questions alongside plausible distractors, emphasizing the importance of nuanced NLP capabilities. Throughout the project, key insights included the system's proficient generation of context-specific questions and the crafting of challenging distractors, all accessible through a user-friendly Streamlit interface. This experience highlighted the critical role of deep contextual understanding and the delicate balance needed in crafting distractors that are plausible yet incorrect. Looking forward, I plan to integrate more sophisticated contextual models like GPT-3, expand language support, and introduce adaptive learning components that refine the system based on user feedback.

Code Lines found from the internet: 50%

References

1. https://medium.com/tech-that-works/maximal-marginal-relevance-to-rerank-results-in-unsupervised-keyphrase-extraction-22d95015c7c5
2. https://towardsdatascience.com/generate-distractors-for-mcqs-using-word-vectors-sentence-transformers-and-mmr-algorithm-e3e5b3a90076
3. https://huggingface.co/voidful/bart-distractor-generation