Text as Data

Report on Multiple-choice Question-Answering System

Ronak Janawa 2933784J

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

March 2024

Introduction

Our task is to create a multiple-choice question-answering system in natural language processing and machine learning. Using techniques such as tokenization, set similarity measures, and cosine similarity of term frequency (TF) vectors, we aim to identify the most appropriate answer from a set of options. Our task is to choose the best answer among the four suitable options. The data we used is divided into three file train.json(Training data), val.json(Validation data) and test.json(Testing data).

Section 1: Dataset and Pre-Processing

Data Loading and Initial Observations

After downloading and loading the dataset, we segmented it into training, validation, and test splits. Each split consists of questions paired with four potential answers, where only one answer is correct (while others are factually correct). The pre-processing steps included tokenizing the questions and options using the spaCy library, which involved removing punctuation, whitespace, and converting tokens to lowercase.

Step 1: Load the Data: This part involves loading the training, validation, and testing datasets from JSON files into Pandas DataFrames. This is the initial step to work with the data in a structured format.

Step 2: Tokenization Function Using spaCy: spaCy library is used to load an English language model and remove unnecessary processing pipelines to speed up the tokenization process. The text_pipeline_spacy_special function is defined to tokenize text into lowercase tokens while excluding punctuation and whitespace.

Step 3: Pre-process and Analyze the Data: Various analyses are performed on the tokenized data, like counting the number of questions and options, calculating the average number of tokens per question/option, and examining lexical overlaps and semantic similarities between questions and their correct options.

Key Findings and Statistics

1. Questions and Options Count:

    - The training set comprises 741 questions and 2964 options.
    - The validation set comprises 103 questions and 412 options.
    - The test set comprises 202 questions and 808 options.

2. Average Tokens:

    - The average number of tokens per question in the training set is 6.2726.
    - The average number of tokens per option in the training set is 22.33.

- The average number of tokens per correct option in the training set is 26.03.

3. Additional Insights:

   - Lexical overlap between questions and correct options is higher than with incorrect options, suggesting that correct answers share more vocabulary with the question, as correct answers have 2.65 average lexical overlap while incorrect answers have 1.60 lexical overlap.

   - Correct options tend to be longer than incorrect ones, which could inform feature development, correct answers have an average length of 26.03 on the other hand incorrect answers have 21.10 average length.

   - The semantic similarity score between questions and correct options is only 0.3151 which is really low, indicating that simply relying on semantic similarity may not be effective in all cases.

Section 2: Set Similarity Measures

Methodology and Results

We applied three set similarity measures: overlap coefficient, Sorensen-Dice, and Jaccard similarity, to calculate scores for each question-option pair. The option with the highest score was selected as the answer.

Performance Evaluation

1. **Accuracy:**
   - Overlap coefficient achieved an accuracy of 0.523 on the training set, with 246 ties and 0.466 on the validation, with 29 ties.

   - Sorensen-Dice coefficient achieved an accuracy of 0.429 on the training set, with 20 ties and 0.359 on the validation set, with 4 ties.

   - Jaccard similarity achieved an accuracy of 0.429 on the training set with 20 ties and 0.35 on the validation set, with 4 ties.

Approach:

We first declare two variables with correct predictions and ties valued zero, then in a loop we count the scores that we need to see if we are right or wrong. If we choose a correct option for a question the score is appended. For ties first we check if the score is greater than 1 or not if it is then we increase the ties variable as it tied with the option. To check the accuracy of we will check if the score we got and the correct option match or not, if they match then one is added to the correct prediction variable which will help us deduce the accuracy of code.

Section 3: Cosine Similarity of TF Vectors

Approach and Findings:

Using CountVectorizer, we generated TF vectors for each question and its corresponding options, after generating TF vectors we store then in a matrix of for both question and answer separately. We then calculated the cosine similarity between each question and option pair, selecting the option with the highest similarity as the answer. We use *"cos_sim = cosine_similarity([q_vector], o_matrix)[0]"* for the cosine_similarity and then we choose the best answer option.

Performance and Improvements

1. Accuracy:

- The cosine similarity method achieved an accuracy of 0.446 on the training set and 0.456 on the validation set, which in comparison with set similarity is more than Jaccard similarity and Sorensen-Dice coefficient, however in comparison with Overlap coefficient the accuracy of training set it more then cosine similarity and just a slightly higher in validation set.

2. Modification and Evaluation:

- We propose modifying the TF vector generation process by including bigrams to capture more contextual information. This modification will improve accuracy by considering not just individual words but also the relationship between adjacent words.

- After implementing this change, the accuracy decreased to 0.453 on the training set and increase to 0.466 on the validation set, this shows that this was effective in validation set to considering more contextual information.

Section 4: Cosine similarity of vectors from bert-base-uncased

This part uses the BERT (Bidirectional Encoder Representations from Transformers) model to enhance the question-answering system's performance. It involves embedding extraction and cosine similarity calculations between questions and options to select the best answer.

Step 1: Load BERT Model and Tokenize

- The BertTokenizer and BertModel from the transformers library are initialized. These are set to the 'bert-base-uncased' version, indicating the use of a pre-trained model that handles lowercase input.

- A function named encode_with_bert is defined to encode a list of texts (questions and options) into BERT embeddings. It takes advantage of BERT's capacity to understand the context and semantics of words in sentences.

- Texts are first tokenized and encoded with the tokenizer, including padding and truncation for uniformity.

- The BERT model then processes these encoded inputs to produce embeddings. The pooled output is used, summarizing the entire input sequence into a single embedding vector representing each text.

Step 2: Getting Accuracy:

The select_best_answer function iterates through pairs of question embeddings and lists of option embeddings. It calculates the cosine similarity between the question embedding and each option embedding, selecting the option with the highest similarity as the correct answer.

The performance is a lot lower than the previous set similarity we have used, the accuracy for training set is only 0.147 and for the validation set is also 0.141 which is significantly lower than the pervious performances.

Limitations of Set Similarity and Cosine Similarity Methods:

- **Context Ignorance in Set Similarity:** Set similarity measures like Jaccard or Sorensen-Dice focus purely on the presence or absence of terms and ignore the context in which they appear. This approach might miss the meaning of words in context, leading to incorrect selections when options meaning is different but share common terms.

- **Lack of Semantic Understanding in TF Vectors:** TF vectors capture the frequency of terms; they do not understand the meaning of those terms. Words with similar meanings but different surface forms may be treated as entirely unrelated, and synonyms or related terms contribute no additional similarity score.

- **Fixed Representations in BERT Embeddings:** BERT captures contextual information, the use of a single [CLS] token embedding to represent the entire question or option might not capture all factors, especially for longer texts. The representation may lose vital information necessary for differentiating between closely related options.

Section 5: Fine-tuning a Transformer Modal

- **Creating Question-Option Pairs:** The function create_question_option_pairs generates pairs of questions and options along with their corresponding labels (1 for correct option, 0 for incorrect options) for both the training and validation sets.

- **Tokenization and Dataset Preparation:** It loads a pre-trained BERT tokenizer and model (bert-base-uncased) and tokenizes the question-option pairs. Then, it creates Dataset objects for both the training and validation sets.

- **Training Arguments:** Defines the training arguments including the output directory, batch size, learning rate, number of epochs, and weight decay.

- **Training:** Initializes a Trainer object with the defined model, training arguments, and datasets, and then trains the model.

- **Evaluation:** Evaluates the trained model on the validation set by making predictions and calculating evaluation metrics including accuracy, precision, recall, and F1-score.