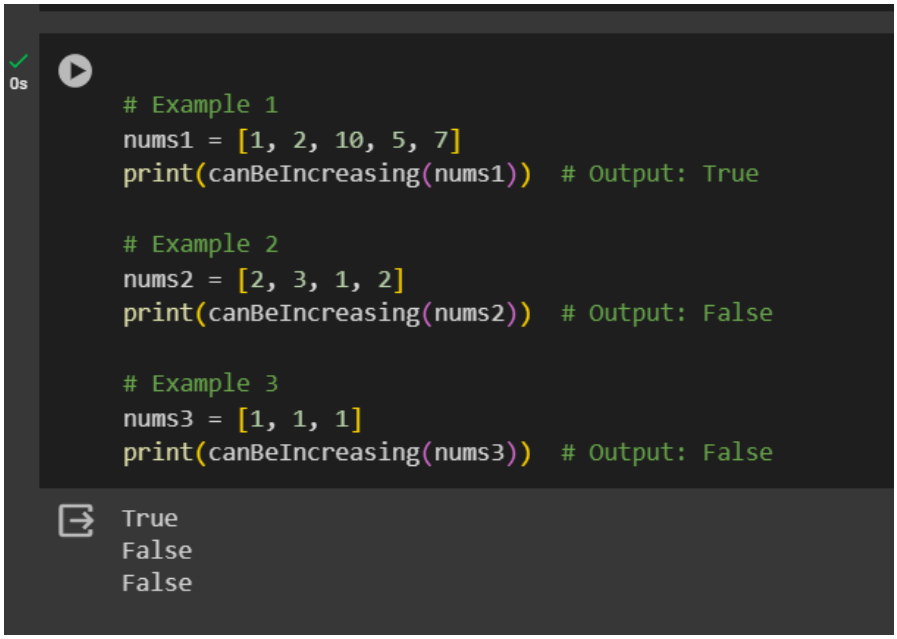
# Python and Machine Learning Assignment By Kashinath J

1. **Problem Statement: Given a 0-indexed integer array `nums`, return `true` if it can be made strictly increasing after removing exactly one element, or `false` otherwise. If the array is already strictly increasing, return `true`.**

**Solution :**

```
def canBeIncreasing(nums):
    """
    Checks if the given array can be made strictly increasing after removing exactly one element.

    Parameters:
    - nums (List[int]): A 0-indexed integer array.

    Returns:
    - bool: True if the array can be made strictly increasing, False otherwise.
    """

    # Variable to count non-strictly increasing pairs
    count = 0

    # Iterate through the array
    for i in range(len(nums) - 1):
        # Check if the current element is greater than or equal to the next element
        if nums[i] >= nums[i + 1]:
            # Increment the count of non-strictly increasing pairs
            count += 1

            # If there are more than one non-strictly increasing pairs, return False
            if count > 1:
                return False

            # Check conditions for continuing or returning False
            if i == 0 or i == len(nums) - 2 or nums[i - 1] < nums[i + 1] or nums[i] < nums[i + 2]:
                continue
            else:
                return False

    # If no more than one removal makes the array strictly increasing, return True
    return True
```


**Explanation :**

- The function canBeIncreasing takes a list of integers nums as input and returns True if the array can be made strictly increasing, False otherwise.
- The variable count is used to keep track of non-strictly increasing pairs.
- The function iterates through the array, checking each element against the next.
- If a non-strictly increasing pair is found, the count is incremented. If there are more than one such pair, the function returns False.

- Conditions are checked for continuing the loop or returning False based on the index and adjacent elements.
- If the loop completes without more than one non-strictly increasing pair, the function returns True.

**2. Problem Statement:** Given a string `s`, find the length of the longest substring without repeating characters.

Solution :

```
def length_of_longest_substring(s):
    """
    Finds the length of the longest substring without repeating characters.

    Parameters:
    - s (str): Input string.

    Returns:
    - int: Length of the longest substring without repeating characters.
    """
    # Initialize variables
    k = 0          # Start of the current substring
    max_len = 0    # Maximum length found
    count = 0      # Current substring length

    # Iterate through the string
    for i in range(1, len(s)):
        # Check for repeating characters in the current substring
        for j in range(k, i):
            if s[i] == s[j]:
                k = j + 1

        # Update the current substring length
        count = i - k + 1

        # Update the maximum length
        if count > max_len:
            max_len = count

    # Return the result
    return max_len
```

```python
def length_of_longest_substring(s):
    # Initialize variables
    k = 0          # Start of the current substring
    max_len = 0    # Maximum length found
    count = 0      # Current substring length

    # Iterate through the string
    for i in range(1, len(s)):
        # Check for repeating characters in the current substring
        for j in range(k, i):
            if s[i] == s[j]:
                k = j + 1
        # Update the current substring length
        count = i - k + 1
        # Update the maximum length
        if count > max_len:
            max_len = count

    # Return the result
    return max_len
```

```python
[6] # Example1
    s_example = "abcabcbb"
    print(length_of_longest_substring(s_example))
    # Example2
    s_example = "abcabcdebb"
    print(length_of_longest_substring(s_example))
```

```
3
5
```

**Explanation:**

- **Initialization:**

k: Represents the start of the current substring.

max_len: Represents the maximum length found.
count: Represents the current substring length.

- **Iteration through the string:**

for i in range(1, len(s)): iterates through the string starting from index 1.
The outer loop represents the end of the current substring.

- **Checking for repeating characters:**

for j in range(k, i): iterates through the current substring to check for repeating characters.
If a repeating character is found at index j, update the start index k to j + 1.
Updating current substring length:

count = i - k + 1 calculates the length of the current substring without repeating characters.

- **Updating the maximum length:**

If count is greater than the current max_len, update max_len with the new value.
Returning the result:

After iterating through the entire string, return the final max_len.

**3. Problem Statement:** Given an array `arr` of integers, check if there exist two indices `i` and `j` such that:
- i != j
- 0 <= i, j < arr.length
- arr[i] == 2 * arr[j]

**Solution :**

```
def check_double(arr):
# Iterate through each pair of indices i and j
for i in range(len(arr)):
for j in range(len(arr)):
# Check if the conditions arr[i] == 2 * arr[j] and i != j are satisfied
if i != j and arr[i] == 2 * arr[j]:
return True

# If no such pair is found, return False
return False
```

```python
def check_double(arr):
    # Iterate through each pair of indices i and j
    for i in range(len(arr)):
        for j in range(len(arr)):
            # Check if the conditions arr[i] == 2 * arr[j] and i != j are satisfied
            if i != j and arr[i] == 2 * arr[j]:
                return True

    # If no such pair is found, return False
    return False
```

```python
[8] # Example 1
arr1 = [10, 2, 5, 3]
print(check_double(arr1))  # Output: True

# Example 2
arr2 = [3, 1, 7, 11]
print(check_double(arr2))  # Output: False
```

```
True
False
```

**Explanation:**

- The function check_double takes an array of integers arr as input and returns True if there exist two indices i and j such that arr[i] is equal to 2 * arr[j], otherwise, it returns False.
- The function uses nested loops to iterate through each pair of indices i and j in the array.
- Inside the loop, it checks if the conditions arr[i] == 2 * arr[j] and i != j are satisfied.
- If a pair of indices satisfying the conditions is found, the function returns True.
- If no such pair is found after iterating through the entire array, the function returns False.

**NLP Task :**

**Summary of Implementation:**

**1. Libraries Installation:**

- Upgraded pip and installed necessary libraries, including torch, torchdata, transformers, and datasets.

**2. Loading Dataset:**

- Loaded the dataset named "knkarthick/dialogsum" using the load_dataset function from the datasets library.

**3. Example Indices:**

- Selected two example indices from the test set (example_indices = [140, 20]).

**4. Baseline Human Summary and Dialogue Display:**

- Displayed the baseline human summary and input dialogue for the selected examples.

**5. Model Configuration:**

- Used the FLAN-T5 base model for sequence-to-sequence tasks (model_name='google/flan-t5-base').
- Loaded the model and tokenizer from the Hugging Face model hub.

**6. Model Generation - Without Prompt Engineering:**

- Encoded the input sentence and decoded it to check the functionality of the tokenizer.
- Generated summaries using the model without prompt engineering for selected examples.
- Displayed the input prompt, baseline human summary, and model-generated summary.

**7. Model Generation - Zero Shot:**

- Created a prompt asking the model to summarize the conversation without prompt engineering.
- Generated summaries using the model for selected examples.
- Displayed the input prompt, baseline human summary, and model-generated summary.

**8. Model Generation - Zero Shot with Additional Context:**

- Modified the prompt by including additional context in the form of a question.
- Generated summaries using the model for selected examples.
- Displayed the input prompt, baseline human summary, and model-generated summary.

**9. Model Generation - One Shot Learning:**

- Created a prompt with one-shot learning context for a specific example.
- Generated a summary using the model.
- Displayed the input prompt, baseline human summary, and model-generated summary.

**10. Model Generation - Few Shot Learning:**

- Created a prompt with few-shot learning context for a specific example.
- Generated a summary using the model.
- Displayed the input prompt, baseline human summary, and model-generated summary.

**11. Model Generation - Few Shot Learning with Configured Tokens:**

- Configured the generation to allow a maximum of 150 new tokens.
- Generated a summary using the model.
- Displayed the model-generated summary.

**12. Summary Display:**

- Displayed a dash line, baseline human summary, and model-generated summary for each scenario.

**Key Observations:**

- The code demonstrates various ways to prompt the model for dialogue summarization, including zero-shot, one-shot, and few-shot learning approaches.
- Additional context in the form of questions or statements influences the generated summaries.
- Configuring the maximum number of new tokens can affect the length and content of the generated summaries.

**Challenges Faced**

**Complexity of Dialogue Summarization:**

- Dialogue summarization is a challenging task due to the dynamic and context-dependent nature of conversations.
- Developing an effective chatbot for dialogue summarization requires handling diverse input structures and generating coherent and concise summaries.

**Optimizing Prompt Engineering:**

- Identifying the most effective prompts for generating high-quality summaries is a non-trivial task.
- Experimentation with various prompt structures and techniques is necessary to find the best approach for a given model.

**Handling Model Limitations:**

- Pre-trained models may have limitations in understanding context and generating accurate summaries.
- Dealing with these limitations requires a combination of prompt engineering, model fine-tuning, and exploring different model architectures.

**Final Note:**

The implementation showcases different prompt engineering strategies for dialogue summarization using the FLAN-T5 model.
It serves as a comprehensive example of using Hugging Face's transformers library for sequence-to-sequence tasks.

```
example_indices_full = [45, 180, 200]
example_index_to_summarize = 200

few_shot_prompt = make_prompt(example_indices_full, example_index_to_summarize)

print(few_shot_prompt)
```

```
#Person1#: Maybe you should check your email.
#Person2#: Oh yes, I find it. Tonight at her home. Will you bring something?
#Person1#: Yes, a pair of wineglasses and a card to wish her happy marriage.
#Person2#: I will buy a tea set.

What was going on?
#Person1# tells #Person2# that Ruojia is married and will have a party tonight. #Person2#'s surprised to know that. They will bring their gifts to bless her.



Dialogue:

#Person1#: Hey Jack. How were your classes this semester?
#Person2#: They were not too bad. I really liked my poli-sci class.
#Person1#: Would you consider it your favorite class?
#Person2#: I don't know if I would call it my favorite, but it ranks up there.
#Person1#: What class was your favorite then?
#Person2#: I took a business communication class last year and it was terrific.
#Person1#: I never took that yet. If that was your favorite, I think I will check it out.

What was going on?
Jack tells #Person1# that business communication is his favorite last year and #Person1# will check it.
```

```
Dialogue:

#Person1#: Have you considered upgrading your system?
#Person2#: Yes, but I'm not sure what exactly I would need.
#Person1#: You could consider adding a painting program to your software. It would allow you to make up your own flyers and banners for advertising.
#Person2#: That would be a definite bonus.
#Person1#: You might also want to upgrade your hardware because it is pretty outdated now.
#Person2#: How can we do that?
#Person1#: You'd probably need a faster processor, to begin with. And you also need a more powerful hard disc, more memory and a faster modem. Do you have a CD-ROM drive?
#Person2#: No.
#Person1#: Then you might want to add a CD-ROM drive too, because most new software programs are coming out on Cds.
#Person2#: That sounds great. Thanks.

What was going on?
#Person1# teaches #Person2# how to upgrade software and hardware in #Person2#'s system.



Dialogue:

#Person1#: Have you considered upgrading your system?
#Person2#: Yes, but I'm not sure what exactly I would need.
#Person1#: You could consider adding a painting program to your software. It would allow you to make up your own flyers and banners for advertising.
#Person2#: That would be a definite bonus.
#Person1#: You might also want to upgrade your hardware because it is pretty outdated now.
#Person2#: How can we do that?
#Person1#: You'd probably need a faster processor, to begin with. And you also need a more powerful hard disc, more memory and a faster modem. Do you have a CD-ROM drive?
#Person2#: No.
#Person1#: Then you might want to add a CD-ROM drive too, because most new software programs are coming out on Cds.
```

**Colab link :**
[https://colab.research.google.com/drive/1OYGiatB11x_u90x8msAQN8RWliviwmM9?usp=sharing](https://colab.research.google.com/drive/1OYGiatB11x_u90x8msAQN8RWliviwmM9?usp=sharing)