text_summarization

September 29, 2023

[1]:

1 Text Summarization with Generative Models on Vertex AI

Run in Colab

View on GitHub

Open in Vertex AI Workbench

1.1 Overview

Text summarization produces a concise and fluent summary of a longer text document. There are two main text summarization types: extractive and abstractive. Extractive summarization involves selecting critical sentences from the original text and combining them to form a summary. Abstractive summarization involves generating new sentences representing the original text's main points. In this notebook, you go through a few examples of how large language models can help with generating summaries based on text.

Learn more about text summarization in the official documentation.

1.1.1 Objective

In this tutorial, you will learn how to use generative models to summarize information from text by working through the following examples: - Transcript summarization - Summarizing text into bullet points - Dialogue summarization with to-dos - Hashtag tokenization - Title & heading generation

You also learn how to evaluate model-generated summaries by comparing to human-created summaries using ROUGE as an evaluation framework.

1.1.2 Costs

This tutorial uses billable components of Google Cloud:

- Vertex AI Generative AI Studio

Learn about Vertex AI pricing, and use the Pricing Calculator to generate a cost estimate based on your projected usage.

1.2 Getting Started

1.2.1 Install Vertex AI SDK

[2]:
```
!pip install google-cloud-aiplatform --upgrade --user
```

```
Requirement already satisfied: google-cloud-aiplatform in
/home/jupyter/.local/lib/python3.10/site-packages (1.33.1)
Requirement already satisfied: google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.2.*,!=2.
3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-aiplatform) (1.34.0)
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.0 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-aiplatform) (1.22.3)
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.2
1.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-aiplatform) (3.19.6)
Requirement already satisfied: packaging>=14.3 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-aiplatform) (23.1)
Requirement already satisfied: google-cloud-storage<3.0.0dev,>=1.32.0 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-aiplatform) (2.10.0)
Requirement already satisfied: google-cloud-bigquery<4.0.0dev,>=1.15.0 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-aiplatform) (3.11.4)
Requirement already satisfied: google-cloud-resource-manager<3.0.0dev,>=1.3.3 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-aiplatform) (1.10.3)
Requirement already satisfied: shapely<2.0.0 in /opt/conda/lib/python3.10/site-
packages (from google-cloud-aiplatform) (1.8.5.post1)
Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.56.2 in
/opt/conda/lib/python3.10/site-packages (from google-api-core[grpc]!=2.0.*,!=2.1
.*,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (1.60.0)
Requirement already satisfied: google-auth<3.0dev,>=1.25.0 in
/opt/conda/lib/python3.10/site-packages (from google-api-core[grpc]!=2.0.*,!=2.1
.*,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (2.22.0)
Requirement already satisfied: requests<3.0.0dev,>=2.18.0 in
/opt/conda/lib/python3.10/site-packages (from google-api-core[grpc]!=2.0.*,!=2.1
.*,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (2.31.0)
```

```
Requirement already satisfied: grpcio<2.0dev,>=1.33.2 in
/opt/conda/lib/python3.10/site-packages (from google-api-core[grpc]!=2.0.*,!=2.1
.*,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (1.48.1)
Requirement already satisfied: grpcio-status<2.0dev,>=1.33.2 in
/opt/conda/lib/python3.10/site-packages (from google-api-core[grpc]!=2.0.*,!=2.1
.*,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (1.48.1)
Requirement already satisfied: google-cloud-core<3.0.0dev,>=1.6.0 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-
bigquery<4.0.0dev,>=1.15.0->google-cloud-aiplatform) (2.3.3)
Requirement already satisfied: google-resumable-media<3.0dev,>=0.6.0 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-
bigquery<4.0.0dev,>=1.15.0->google-cloud-aiplatform) (2.5.0)
Requirement already satisfied: python-dateutil<3.0dev,>=2.7.2 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-
bigquery<4.0.0dev,>=1.15.0->google-cloud-aiplatform) (2.8.2)
Requirement already satisfied: grpc-google-iam-v1<1.0.0dev,>=0.12.4 in
/opt/conda/lib/python3.10/site-packages (from google-cloud-resource-
manager<3.0.0dev,>=1.3.3->google-cloud-aiplatform) (0.12.6)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/opt/conda/lib/python3.10/site-packages (from google-
auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,!=2.4
.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (4.2.4)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/opt/conda/lib/python3.10/site-packages (from google-
auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,!=2.4
.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (0.2.7)
Requirement already satisfied: rsa<5,>=3.1.4 in /opt/conda/lib/python3.10/site-
packages (from google-auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.*,!=2.1.*
,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (4.9)
Requirement already satisfied: six>=1.9.0 in /opt/conda/lib/python3.10/site-
packages (from google-auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.*,!=2.1.*
,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (1.16.0)
Requirement already satisfied: urllib3<2.0 in /opt/conda/lib/python3.10/site-
packages (from google-auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.*,!=2.1.*
,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (1.26.15)
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in
/opt/conda/lib/python3.10/site-packages (from google-resumable-
media<3.0dev,>=0.6.0->google-cloud-bigquery<4.0.0dev,>=1.15.0->google-cloud-
aiplatform) (1.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/conda/lib/python3.10/site-packages (from
requests<3.0.0dev,>=2.18.0->google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*
,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform)
```

```
(3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-
packages (from requests<3.0.0dev,>=2.18.0->google-api-core[grpc]!=2.0.*,!=2.1.*,
!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-
cloud-aiplatform) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.10/site-packages (from
requests<3.0.0dev,>=2.18.0->google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*
,!=2.4.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform)
(2023.7.22)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/opt/conda/lib/python3.10/site-packages (from pyasn1-modules>=0.2.1->google-
auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,!=2.4
.*,!=2.5.*,!=2.6.*,!=2.7.*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (0.4.8)
```

Colab only: Uncomment the following cell to restart the kernel or use the button to restart the kernel. For Vertex AI Workbench you can restart the terminal using the button on top.

[3]:
```
# # Automatically restart kernel after installs so that your environment can access the new packages
# import IPython

# app = IPython.Application.instance()
# app.kernel.do_shutdown(True)
```

1.2.2 Authenticating your notebook environment

- If you are using **Colab** to run this notebook, uncomment the cell below and continue.
- If you are using **Vertex AI Workbench**, check out the setup instructions here.
- If you are using **local Jupyter**, check out the setup instructions here.

[4]:
```
# from google.colab import auth
# auth.authenticate_user()
```

1.2.3 Import libraries

Let's start by importing the libraries that we will need for this tutorial

Colab only: Uncomment the following cell to initialize the Vertex AI SDK. For Vertex AI Workbench, you don't need to run this.

[5]:
```
# import vertexai

# PROJECT_ID = "[your-project-id]"  # @param {type:"string"}
# vertexai.init(project=PROJECT_ID, location="us-central1")
```

[6]:
```
from vertexai.language_models import TextGenerationModel
```

```
2023-09-29 15:49:04.298459: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
```

```
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2023-09-29 15:49:06.005763: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot
open shared object file: No such file or directory; LD_LIBRARY_PATH:
/usr/local/cuda/lib64:/usr/local/nccl2/lib:/usr/local/cuda/extras/CUPTI/lib64
2023-09-29 15:49:06.005976: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libnvinfer_plugin.so.7'; dlerror:
libnvinfer_plugin.so.7: cannot open shared object file: No such file or
directory; LD_LIBRARY_PATH:
/usr/local/cuda/lib64:/usr/local/nccl2/lib:/usr/local/cuda/extras/CUPTI/lib64
2023-09-29 15:49:06.006019: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot
dlopen some TensorRT libraries. If you would like to use Nvidia GPU with
TensorRT, please make sure the missing libraries mentioned above are installed
properly.
```

1.2.4 Import models

Here we load the pre-trained text generation model called `text-bison@001`.

[7]:
```python
generation_model = TextGenerationModel.from_pretrained("text-bison@001")
```

1.3 Text Summarization

1.3.1 Transcript summarization

In this first example, you summarize a piece of text on quantum computing.

[8]:
```python
prompt = """
Provide a very short summary, no more than three sentences, for the following␣
  ↪article:

Our quantum computers work by manipulating qubits in an orchestrated fashion␣
  ↪that we call quantum algorithms.
The challenge is that qubits are so sensitive that even stray light can cause␣
  ↪calculation errors - and the problem worsens as quantum computers grow.
This has significant consequences, since the best quantum algorithms that we␣
  ↪know for running useful applications require the error rates of our qubits␣
  ↪to be far lower than we have today.
To bridge this gap, we will need quantum error correction.
```

```
Quantum error correction protects information by encoding it across multiple␣
 ↪physical qubits to form a "logical qubit," and is believed to be the only␣
 ↪way to produce a large-scale quantum computer with error rates low enough␣
 ↪for useful calculations.
Instead of computing on the individual qubits themselves, we will then compute␣
 ↪on logical qubits. By encoding larger numbers of physical qubits on our␣
 ↪quantum processor into one logical qubit, we hope to reduce the error rates␣
 ↪to enable useful quantum algorithms.

Summary:

"""

print(
    generation_model.predict(
        prompt, temperature=0.2, max_output_tokens=1024, top_k=40, top_p=0.8
    ).text
)
```

```
Quantum computers are very sensitive and errors can occur easily.
Quantum error correction protects information by encoding it across multiple
physical qubits to form a logical qubit.
By encoding larger numbers of physical qubits on our quantum processor into one
logical qubit, we hope to reduce the error rates to enable useful quantum
algorithms.
```

Instead of a summary, we can ask for a TL;DR ("too long; didn't read"). You can compare the differences between the outputs generated.

[9]:

```
prompt = """
Provide a TL;DR for the following article:

Our quantum computers work by manipulating qubits in an orchestrated fashion␣
 ↪that we call quantum algorithms.
The challenge is that qubits are so sensitive that even stray light can cause␣
 ↪calculation errors - and the problem worsens as quantum computers grow.
This has significant consequences, since the best quantum algorithms that we␣
 ↪know for running useful applications require the error rates of our qubits␣
 ↪to be far lower than we have today.
To bridge this gap, we will need quantum error correction.
Quantum error correction protects information by encoding it across multiple␣
 ↪physical qubits to form a "logical qubit," and is believed to be the only␣
 ↪way to produce a large-scale quantum computer with error rates low enough␣
 ↪for useful calculations.
Instead of computing on the individual qubits themselves, we will then compute␣
 ↪on logical qubits. By encoding larger numbers of physical qubits on our␣
 ↪quantum processor into one logical qubit, we hope to reduce the error rates␣
 ↪to enable useful quantum algorithms.
```

```
TL;DR:
"""

print(
    generation_model.predict(
        prompt, temperature=0.2, max_output_tokens=1024, top_k=40, top_p=0.8
    ).text
)
```

```
Quantum computers are very sensitive and even stray light can cause calculation
errors. This problem worsens as quantum computers grow. To bridge this gap, we
will need quantum error correction which protects information by encoding it
across multiple physical qubits to form a "logical qubit".
```

1.3.2 Summarize text into bullet points

In the following example, you use same text on quantum computing, but ask the model to summarize it in bullet-point form. Feel free to change the prompt.

[10]:
```
prompt = """
Provide a very short summary in four bullet points for the following article:

Our quantum computers work by manipulating qubits in an orchestrated fashion␣
  ↪that we call quantum algorithms.
The challenge is that qubits are so sensitive that even stray light can cause␣
  ↪calculation errors - and the problem worsens as quantum computers grow.
This has significant consequences, since the best quantum algorithms that we␣
  ↪know for running useful applications require the error rates of our qubits␣
  ↪to be far lower than we have today.
To bridge this gap, we will need quantum error correction.
Quantum error correction protects information by encoding it across multiple␣
  ↪physical qubits to form a "logical qubit," and is believed to be the only␣
  ↪way to produce a large-scale quantum computer with error rates low enough␣
  ↪for useful calculations.
Instead of computing on the individual qubits themselves, we will then compute␣
  ↪on logical qubits. By encoding larger numbers of physical qubits on our␣
  ↪quantum processor into one logical qubit, we hope to reduce the error rates␣
  ↪to enable useful quantum algorithms.

Bulletpoints:

"""

print(
    generation_model.predict(
        prompt, temperature=0.2, max_output_tokens=256, top_k=1, top_p=0.8
    ).text
)
```

```
- Quantum computers work by manipulating qubits in an orchestrated fashion that
we call quantum algorithms.
- Qubits are so sensitive that even stray light can cause calculation errors.
- The best quantum algorithms require the error rates of our qubits to be far
lower than we have today.
- Quantum error correction protects information by encoding it across multiple
physical qubits to form a "logical qubit".
```

1.3.3 Dialogue summarization with to-dos

Dialogue summarization involves condensing a conversation into a shorter format so that you don't need to read the whole discussion but can leverage a summary. In this example, you ask the model to summarize an example conversation between an online retail customer and a support agent, and include to-dos at the end.

[11]:

```
prompt = """
Please generate a summary of the following conversation and at the end summarize the to-do's for the support Agent:

Customer: Hi, I'm Larry, and I received the wrong item.

Support Agent: Hi, Larry. How would you like to see this resolved?

Customer: That's alright. I want to return the item and get a refund, please.

Support Agent: Of course. I can process the refund for you now. Can I have your order number, please?

Customer: It's [ORDER NUMBER].

Support Agent: Thank you. I've processed the refund, and you will receive your money back within 14 days.

Customer: Thank you very much.

Support Agent: You're welcome, Larry. Have a good day!

Summary:
"""

print(
    generation_model.predict(
        prompt, temperature=0.2, max_output_tokens=256, top_k=40, top_p=0.8
    ).text
)
```

```
Larry received the wrong item and wants to return it for a refund.
The support agent processed the refund and Larry will receive his money back
within 14 days.
To-do's for the support agent:
- Process the refund.
- Send an email to Larry confirming the refund.
```

1.3.4 Hashtag tokenization

Hashtag tokenization is the process of taking a piece of text and getting the hashtag “tokens” out of it. You can use this, for example, if you want to generate hashtags for your social media campaigns. In this example, you take this tweet from Google Cloud and generate some hashtags you can use.

[12]:
```python
prompt = """
Tokenize the hashtags of this tweet:

Google Cloud
@googlecloud
How can data help our changing planet?

In honor of #EarthDay this weekend, we’re proud to share how we’re partnering with
@ClimateEngine
 to harness the power of geospatial data and drive toward a more sustainable future.

Check out how → https://goo.gle/3mOUfts
"""

print(
    generation_model.predict(
        prompt, temperature=0.8, max_output_tokens=1024, top_k=40, top_p=0.8
    ).text
)
```

```
EarthDay
```

1.3.5 Title & heading generation

Below, you ask the model to generate five options for possible title/heading combos for a given piece of text.

[13]:
```python
prompt = """
Write a title for this text, give me five options:
Whether helping physicians identify disease or finding photos of "hugs," AI is behind a lot of the work we do at Google. And at our Arts & Culture Lab in Paris, we’ve been experimenting with how AI can be used for the benefit of culture.
```

```
Today, we’re sharing our latest experiments-prototypes that build on seven␣
 ↪years of work in partnership the 1,500 cultural institutions around the␣
 ↪world.
Each of these experimental applications runs AI algorithms in the background to␣
 ↪let you unearth cultural connections hidden in archives-and even find␣
 ↪artworks that match your home decor."
"""

print(
    generation_model.predict(
        prompt, temperature=0.8, max_output_tokens=256, top_k=1, top_p=0.8
    ).text
)
```

```
1. How AI is used for the benefit of culture
2. Google Arts & Culture Lab experiments with AI
3. AI in the Arts & Culture Lab
4. AI for culture
5. AI in the Arts
```

1.4 Evaluation

You can evaluate the outputs from summarization tasks using ROUGE as an evalulation framework. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) are measures to automatically determine the quality of a summary by comparing it to other (ideal) summaries created by humans. The measures count the number of overlapping units such as n-gram, word sequences, and word pairs between the computer-generated summary to be evaluated and the ideal summaries created by humans.

The first step is to install the ROUGE library.

[14]:

```
!pip install rouge
```

```
Collecting rouge
  Downloading rouge-1.0.1-py3-none-any.whl (13 kB)
Requirement already satisfied: six in /opt/conda/lib/python3.10/site-packages
(from rouge) (1.16.0)
Installing collected packages: rouge
Successfully installed rouge-1.0.1
```

Create a summary from a language model that you can use to compare against a human-generated summary.

[15]:

```
from rouge import Rouge

ROUGE = Rouge()

prompt = """
Provide a very short, maximum four sentences, summary for the following article:
```

```
Our quantum computers work by manipulating qubits in an orchestrated fashion that we call quantum algorithms.
The challenge is that qubits are so sensitive that even stray light can cause calculation errors - and the problem worsens as quantum computers grow.
This has significant consequences, since the best quantum algorithms that we know for running useful applications require the error rates of our qubits to be far lower than we have today.
To bridge this gap, we will need quantum error correction.
Quantum error correction protects information by encoding it across multiple physical qubits to form a "logical qubit," and is believed to be the only way to produce a large-scale quantum computer with error rates low enough for useful calculations.
Instead of computing on the individual qubits themselves, we will then compute on logical qubits. By encoding larger numbers of physical qubits on our quantum processor into one logical qubit, we hope to reduce the error rates to enable useful quantum algorithms.

Summary:

"""

candidate = generation_model.predict(
    prompt, temperature=0.1, max_output_tokens=1024, top_k=40, top_p=0.9
).text

print(candidate)
```

```
Quantum computers work by manipulating qubits.
Qubits are sensitive to errors, and the problem worsens as quantum computers
grow.
Quantum error correction protects information by encoding it across multiple
physical qubits to form a "logical qubit".
By encoding larger numbers of physical qubits on our quantum processor into one
logical qubit, we hope to reduce the error rates to enable useful quantum
algorithms.
```

You will also need a human-generated summary that we will use to compare to the `candidate` generated by the model. We will call this `reference`.

[16]:

```
reference = "Quantum computers are sensitive to noise and errors. To bridge this gap, we will need quantum error correction. Quantum error correction protects information by encoding across multiple physical qubits to form a "logical qubit"."
```

Now you can take the candidate and reference to evaluate the performance. In this case, ROUGE will give you:

- `rouge-1`, which measures unigram overlap

- `rouge-2`, which measures bigram overlap
- `rouge-l`, which measures the longest common subsequence

[17]:

```
ROUGE.get_scores(candidate, reference)
```

[17]:

```
[{'rouge-1': {'r': 0.7333333333333333,
   'p': 0.4489795918367347,
   'f': 0.5569620206056722},
  'rouge-2': {'r': 0.53125, 'p': 0.2786885245901639, 'f': 0.3655913933356458},
  'rouge-l': {'r': 0.7, 'p': 0.42857142857142855, 'f': 0.5316455649094697}}]
```