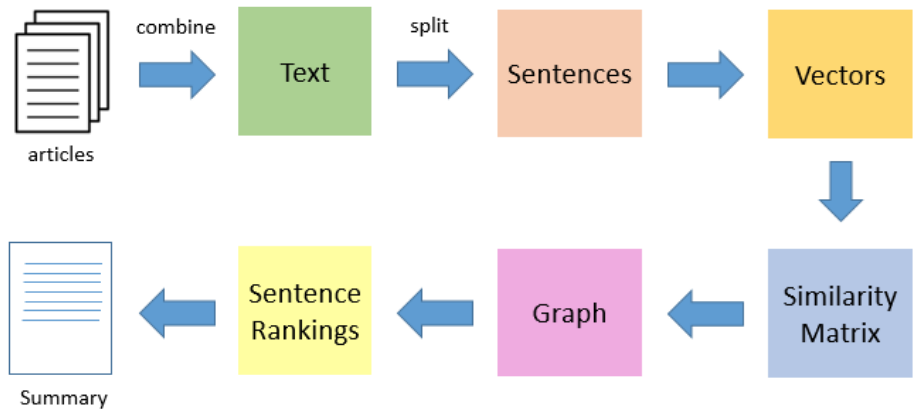
Text summarization is a very useful and important part of Natural Language Processing (NLP). First let us talk about what text summarization is. Suppose we have too many lines of text data in any form, such as from articles or magazines or on social media. We have time scarcity so we want only a nutshell report of that text. We can summarize our text in a few lines by removing unimportant text and converting the same texText summarization is a very useful and important part of Natural Language Processing (NLP). First let us talk about what text summarization is. Suppose we have too many lines of text data in any form, such as from articles or magazines or on social media. We have time scarcity so we want only a nutshell report of that text. We can summarize our text in a few lines by removing unimportant text and converting the same text into smaller semantic text form.

Now let us see how we can implement NLP in our programming. We will take a look at all the approaches later, but here we will classify approaches of NLP.

TEXT SUMMARIZATION

In this approach we build algorithms or programs which will reduce the text size and create a summary of our text data. This is called automatic text summarization in machine learning.  
Text summarization is the process of creating shorter text without removing the semantic structure of text.



There are two approaches to text summarization.

1. Extractive approaches
2. Abstractive approaches

EXTRACTIVE APPROACHES:

Using an extractive approach we summarize our text on the basis of simple and traditional algorithms. For example, when we want to summarize our text on the basis of the frequency method, we store all the important words and frequency of all those words in the dictionary. On the basis of high frequency words, we store the sentences containing that word in our final summary. This means the words which are in our summary confirm that they are part of the given text.

ABSTRACTIVE APPROACHES:

An abstractive approach is more advanced. On the basis of time requirements we exchange some sentences for smaller sentences with the same semantic approaches of our text data.

**2.**

**What is Text Summarization?**

Text summarization is the process of distilling the essential information from a piece of text, creating a shorter version while retaining its core meaning. There are two main types: extractive, which selects and stitches together existing sentences, and abstractive, which generates new sentences to convey the main points.

**The BART Model:**

Our hero for this task is BART (Bidirectional and Auto-Regressive Transformers). BART is a transformer-based model pre-trained on a denoising autoencoder objective, meaning it’s great at reconstructing clean sentences from noisy ones. This ability makes it a fantastic choice for various NLP tasks, including summarization.

**Why BART:**

The choice of the BART model in the provided code example is just one of many possibilities. BART (Bidirectional and Auto-Regressive Transformers) is a transformer-based model that has shown strong performance in various natural language processing tasks, including abstractive text summarization. It was pre-trained on a denoising autoencoder objective, which makes it proficient at generating coherent and contextually relevant summaries.

The Hugging Face Transformers library provides a variety of pre-trained models, each with its strengths and use cases. BART is just one example; depending on your specific requirements, you might choose a different model. Some other popular models for text summarization include GPT-3 (Generative Pre-trained Transformer 3), T5 (Text-to-Text Transfer Transformer), and Pegasus.

The choice of the model depends on factors such as the size of the model, the amount of training data, the task complexity, and the specific characteristics of the input text and desired summaries. Experimenting with different models and fine-tuning them on your specific task or dataset can help you find the most suitable one for your needs.

BART (Bidirectional and Auto-Regressive Transformers) is a transformer-based model designed for various natural language processing tasks, including abstractive text summarization. Let’s break down the key components of BART in detail.

**1. Transformer Architecture:**

BART is built upon the Transformer architecture, introduced by Vaswani et al. in the paper “Attention is All You Need.” This architecture uses self-attention mechanisms to capture contextual relationships between words in a sequence.

**Self-Attention Mechanism**

The self-attention mechanism computes a weighted sum of values for each element in a sequence, where the weights are determined by the relevance of other elements to the current one. The formula for computing the attention score *Aij*​ between the *i*-th and *j*-th elements is as follows:

A mathematical equation with numbers and symbols

Description automatically generated

where *eij*​ is the attention energy, computed as the dot product of query, key, and a scaling factor:

A math equations and formulas

Description automatically generated with medium confidence

Here, *Qi*​ and *Kj*​ are the query and key vectors, and *dk*​ is the dimensionality of the key vectors.

**2. Tokenization:**

Before feeding text into the BART model, the input text is tokenized into smaller units called tokens. Tokenization is crucial for the model to process and understand the input. BART uses a tokenizer to perform this task.

The tokenization process involves breaking down a sequence of characters into tokens. In mathematical terms, if *X* is the input sequence and *T*(*X*) is the tokenized sequence, then:

A number of mathematical equations

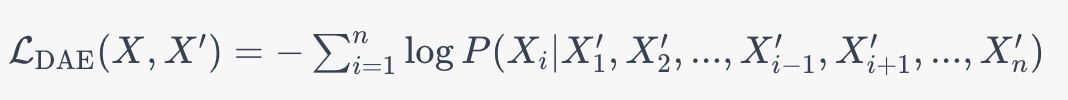
Description automatically generated with medium confidence

where *xi*​ represents the *i*-th token in the tokenized sequence.

**3. BART Architecture:**

BART extends the transformer architecture with a specific pre-training objective, making it effective for various tasks, including summarization. It is trained to denoise documents corrupted by various techniques like shuffling and removing sentences.

BART is pre-trained using a denoising autoencoder objective. Given a corrupted input sequence ′*X*′, the model is trained to reconstruct the original sequence *X*. The objective is to minimize the reconstruction loss:



**4. Summarization:**

During the fine-tuning phase for summarization, BART is further trained on summarization-specific datasets to fine-tune its ability to generate abstractive summaries.

The objective during summarization involves generating a summary *Y* from the input sequence *X*. BART is fine-tuned to minimize the negative log-likelihood of the target summary given the input:



BART’s strength lies in its transformer architecture, denoising autoencoder pre-training, and its ability to generate coherent and contextually relevant summaries during the fine-tuning phase. Understanding the self-attention mechanism, tokenization, and the specific pre-training and fine-tuning objectives provides insights into how BART operates and excels in text summarization tasks.

**Code Implementation:**

Now, let’s bring this to life with some code using the Hugging Face Transformers library. We’ll explore a Python script that leverages the PyMuPDF library for PDF processing and the Transformers library for natural language processing to perform extractive summarization. The goal is to summarize the content of a PDF file and save the summary in a new PDF document.

**Part 1: Importing Libraries**

The script begins by importing the necessary libraries:

import fitz # PyMuPDF  
from transformers import BartForConditionalGeneration, BartTokenizer  
import textwrap

* PyMuPDF (fitz): This library provides functionality for working with PDF files. In our case, we use it to extract text from PDF pages
* Transformers (BartForConditionalGeneration and BartTokenizer): These components come from the Hugging Face Transformers library and are essential for using the BART model, a sequence-to-sequence model widely used for natural language processing tasks.
* textwrap: A standard Python library module that helps with formatting text.

**Part 2: Extracting Text from PDFs**

The first function, extract\_text\_from\_pdf, handles the extraction of text from a PDF file:

def extract\_text\_from\_pdf(pdf\_path):  
 doc = fitz.open(pdf\_path)  
 text = ""  
 for page\_num in range(doc.page\_count):  
 page = doc[page\_num]  
 text += page.get\_text()  
 doc.close()  
 return text

This function takes the path to a PDF file as input and uses PyMuPDF to open the file. It then iterates through each page, extracting text and concatenating it. Finally, the function returns the combined text.

**Part 3: Generating Summaries with BART**

The next function text\_summarizer\_from\_pdf uses BART to generate a summary:

def text\_summarizer\_from\_pdf(pdf\_path):  
 pdf\_text = extract\_text\_from\_pdf(pdf\_path)  
  
 model\_name = "facebook/bart-large-cnn"  
 model = BartForConditionalGeneration.from\_pretrained(model\_name)  
 tokenizer = BartTokenizer.from\_pretrained(model\_name)  
  
 inputs = tokenizer.encode("summarize: " + pdf\_text, return\_tensors="pt", max\_length=1024, truncation=True)  
 summary\_ids = model.generate(inputs, max\_length=150, min\_length=50, length\_penalty=2.0, num\_beams=4, early\_stopping=True)  
  
 summary = tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)  
 formatted\_summary = "\n".join(textwrap.wrap(summary, width=80))  
 return formatted\_summary

This function utilizes the extracted text from the previous step. It loads the BART model and tokenizer and then generates a summary. The summary is obtained by encoding the input text, decoding the output summary\_ids, and formatting it using textwrap for better readability.

**Part 4: Creating a Summary PDF**

The third function, save\_summary\_as\_pdf, saves the generated summary in a new PDF document:

def save\_summary\_as\_pdf(pdf\_path, summary):  
 doc = fitz.open()  
  
 page = doc.new\_page()  
 page.insert\_text((10, 100), summary, fontname="helv", fontsize=12) # Adjust the vertical position as needed  
  
 output\_pdf\_path = pdf\_path.replace(".pdf", "\_summary.pdf")  
 doc.save(output\_pdf\_path)  
 doc.close()  
  
 return output\_pdf\_path

This function creates a new PDF document using PyMuPDF, adds a page, and inserts the formatted summary. The resulting PDF is then saved with a modified filename to indicate that it contains a summary.

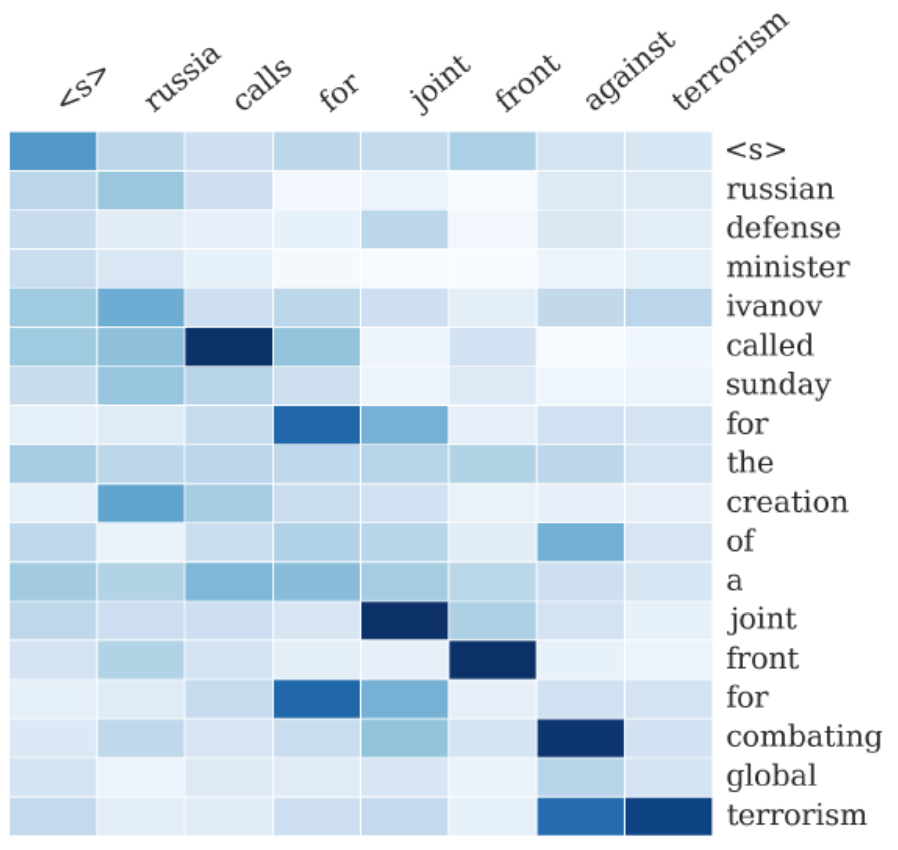
**Part 5: Example Usage**

The script concludes with an example demonstrating how to use the functions:

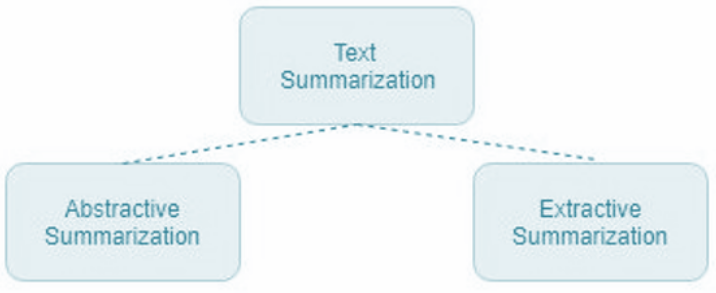
pdf\_file\_path = "path/to/your/file.pdf"  
summary = text\_summarizer\_from\_pdf(pdf\_file\_path)  
output\_pdf\_path = save\_summary\_as\_pdf(pdf\_file\_path, summary)  
print("Summary saved as PDF:", output\_pdf\_path)

This part shows how to apply the functions to a specific PDF file. The text is summarized using BART, and the summary is saved as a new PDF. The path to the saved summary PDF is then printed.

BART’s strength lies in its transformer architecture, denoising autoencoder pre-training, and ability to generate coherent and contextually relevant summaries during the fine-tuning phase. Understanding the self-attention mechanism, tokenization, and the specific pre-training and fine-tuning objectives provides insights into how BART operates and excels in text summarization tasks.



Here we generally use deep machine learning, that is transformers, bi-directional transformers(BERT), GPT, etc.



EXTRACTIVE APPROACHES:

We will take a look at a few machine learning models below.

TEXT SUMMARIZATION USING THE FREQUENCY METHOD

In this method we find the frequency of all the words in our text data and store the text data and its frequency in a dictionary. After that, we tokenize our text data. The sentences which contain more high frequency words will be kept in our final summary data.

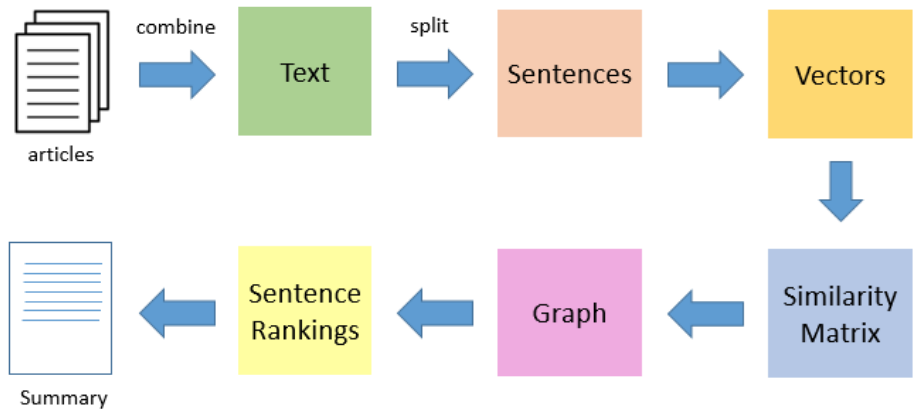
Explain

t into smaller semantic text form.

Now let us see how we can implement NLP in our programming. We will take a look at all the approaches later, but here we will classify approaches of NLP.

TEXT SUMMARIZATION

In this approach we build algorithms or programs which will reduce the text size and create a summary of our text data. This is called automatic text summarization in machine learning.  
Text summarization is the process of creating shorter text without removing the semantic structure of text.



There are two approaches to text summarization.

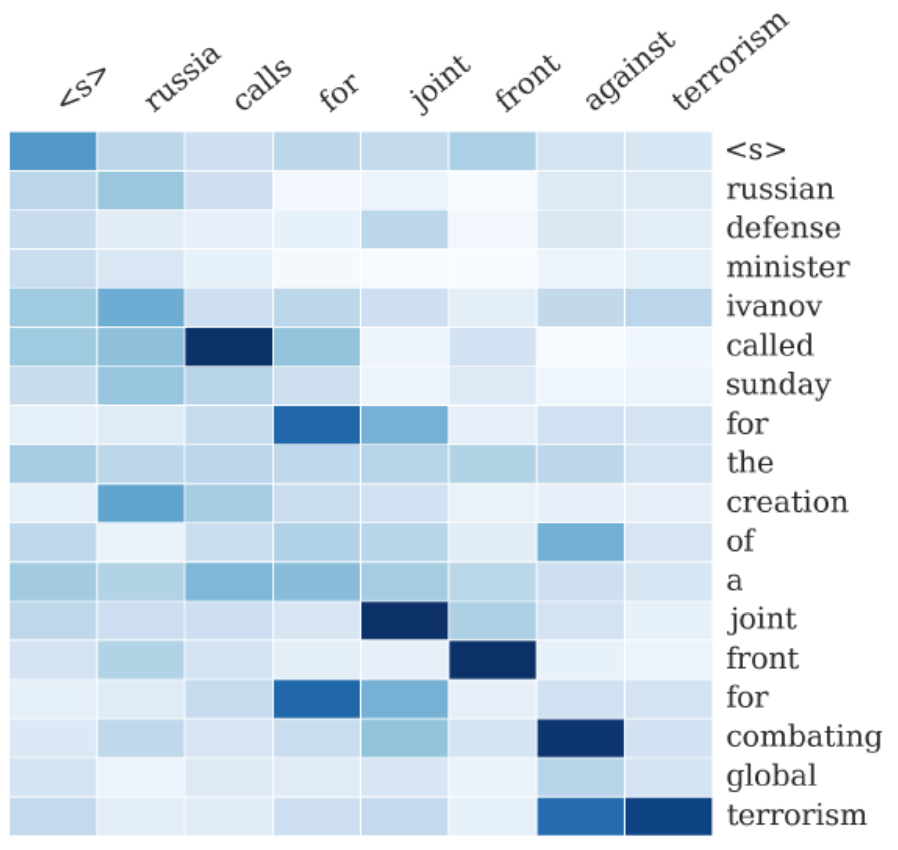
1. Extractive approaches
2. Abstractive approaches

EXTRACTIVE APPROACHES:

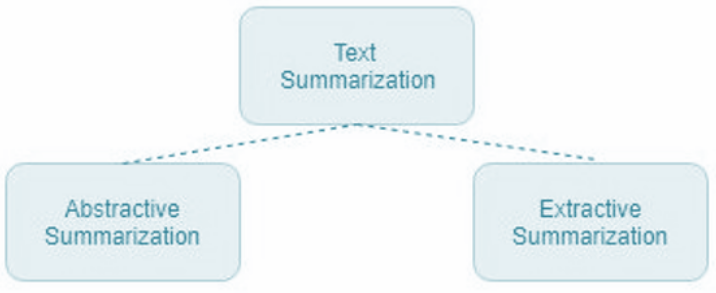
Using an extractive approach we summarize our text on the basis of simple and traditional algorithms. For example, when we want to summarize our text on the basis of the frequency method, we store all the important words and frequency of all those words in the dictionary. On the basis of high frequency words, we store the sentences containing that word in our final summary. This means the words which are in our summary confirm that they are part of the given text.

ABSTRACTIVE APPROACHES:

An abstractive approach is more advanced. On the basis of time requirements we exchange some sentences for smaller sentences with the same semantic approaches of our text data.



Here we generally use deep machine learning, that is transformers, bi-directional transformers(BERT), GPT, etc.



EXTRACTIVE APPROACHES:

We will take a look at a few machine learning models below.

TEXT SUMMARIZATION USING THE FREQUENCY METHOD

In this method we find the frequency of all the words in our text data and store the text data and its frequency in a dictionary. After that, we tokenize our text data. The sentences which contain more high frequency words will be kept in our final summary data.

Explain

**Convert-Text-to-Speech-in-Python**

[Text to speech](https://getprojects.org/convert-text-to-speech-in-python/) is a process to convert any text into voice. Text to speech project takes words on digital devices and converts them into audio with a button click or finger touch. Text to speech python project is constructive for people struggling with reading.

Project (Convert Text to Speech in Python) Prerequisites To implement this project, we will use the basic concepts of Python, Tkinter, gTTS, and playsound libraries.

1. ***Tkinter*** is a standard GUI Python library that is one of the fastest and easiest ways to build GUI applications using Tkinter.
2. ***gTTS (Google Text-to-Speech)*** is a Python library, an elementary library that converts the text into audio.
3. The ***playsound*** module is used to play audio files. With this module, we can play a sound file with a single line of code.