Survey on Text Summarization Techniques

# 1. What is Summarization and its Types

Text Summarization is the process of condensing a piece of text to a shorter version while retaining the key information and main points. It is a crucial task in Natural Language Processing (NLP) with applications in various domains such as news, legal documents, research papers, and social media.

Types of Summarization:  
1. Extractive Summarization: Involves selecting sentences or phrases directly from the original text based on their importance.  
2. Abstractive Summarization: Generates new sentences that convey the same information as the original text, similar to human-written summaries.

# 2. Traditional Techniques

## NLTK (Natural Language Toolkit)

Pros:

* Comprehensive NLP library with tools for preprocessing.  
  Good for educational purposes and rapid prototyping.

Cons:

* Lacks advanced summarization algorithms.  
  Slower compared to some modern libraries.

Code Example:

import nltk  
from nltk.tokenize import sent\_tokenize, word\_tokenize  
  
text = "Your text here."  
sentences = sent\_tokenize(text)

## spaCy

Pros:

* Fast and efficient.  
  Provides pre-trained models.

Cons:

* Less intuitive for beginners.  
  Requires additional packages for full summarization capabilities.

Code Example:

import spacy  
  
nlp = spacy.load('en\_core\_web\_sm')  
doc = nlp("Your text here.")  
sentences = list(doc.sents)

## Gensim

Pros:

* Easy to use for topic modeling and summarization.  
  Efficient for large texts.

Cons:

* Limited to extractive summarization.  
  Older technology compared to modern transformers.

Code Example:

from gensim.summarization import summarize  
  
text = "Your text here."  
summary = summarize(text)

## Sumy

Pros:

* Offers several extractive summarization algorithms.  
  Easy to use and flexible.

Cons:

* Limited to extractive methods.  
  Not as widely maintained.

Available Methods:

* - LSA (Latent Semantic Analysis): Identifies important sentences based on topic modeling.  
  - LexRank: Uses a graph-based method similar to PageRank to score sentences.  
  - TextRank: Another graph-based ranking algorithm for scoring sentences.  
  - Edmundson: A heuristic-based method for summarization.  
  - KL-Sum: Uses Kullback-Leibler divergence to measure the information gain of sentences.

Code Example:

from sumy.parsers.plaintext import PlaintextParser  
from sumy.nlp.tokenizers import Tokenizer  
from sumy.summarizers.lsa import LsaSummarizer  
from sumy.summarizers.lex\_rank import LexRankSummarizer  
from sumy.summarizers.text\_rank import TextRankSummarizer  
  
text = "Your text here."  
parser = PlaintextParser.from\_string(text, Tokenizer("english"))  
  
summarizer = LsaSummarizer()  
summary = summarizer(parser.document, 2)  
for sentence in summary:  
 print(sentence)  
  
summarizer = LexRankSummarizer()  
summary = summarizer(parser.document, 2)  
for sentence in summary:  
 print(sentence)  
  
summarizer = TextRankSummarizer()  
summary = summarizer(parser.document, 2)  
for sentence in summary:  
 print(sentence)

## TextRank

Pros:

* Efficient and unsupervised.  
  Uses a graph-based approach to determine sentence importance.

Cons:

* Limited to extractive summarization.  
  May not capture deep semantic meanings.

Code Example:

import spacy  
import pytextrank  
  
nlp = spacy.load("en\_core\_web\_sm")  
tr = pytextrank.TextRank()  
nlp.add\_pipe("textrank", last=True)  
  
text = "Your text here."  
doc = nlp(text)  
for phrase in doc.\_.phrases:  
 print(phrase.text)

# 3. Modern Techniques

## Transformers (by Hugging Face)

Pros:

* Access to state-of-the-art models.  
  Supports both extractive and abstractive summarization.

Cons:

* Requires significant computational resources.  
  Can be complex to fine-tune.

Code Example:

from transformers import pipeline  
  
summarizer = pipeline("summarization")  
text = "Your text here."  
summary = summarizer(text, max\_length=130, min\_length=30, do\_sample=False)  
print(summary[0]['summary\_text'])

## BART (Bidirectional and Auto-Regressive Transformers)

Pros:

* Combines bidirectional and auto-regressive transformers.  
  Excellent performance in both generative tasks and summarization.

Cons:

* Requires significant computational resources.  
  Complexity in fine-tuning for specific tasks.

Code Example:

from transformers import BartForConditionalGeneration, BartTokenizer  
  
model = BartForConditionalGeneration.from\_pretrained('facebook/bart-large-cnn')  
tokenizer = BartTokenizer.from\_pretrained('facebook/bart-large-cnn')  
  
text = "Your text here."  
inputs = tokenizer([text], max\_length=1024, return\_tensors='pt')  
summary\_ids = model.generate(inputs['input\_ids'], num\_beams=4, max\_length=100, early\_stopping=True)  
summary = tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)  
print(summary)

## BERTSUM

Pros:

* Enhanced version of BERT for summarization tasks.  
  Fine-tuned for both extractive and abstractive summarization.

Cons:

* Complex setup and requires significant training data.  
  Computationally expensive.

Code Example:

Refer to the BERTSUM GitHub repository for detailed setup and usage: https://github.com/nlpyang/PreSumm

# 4. Evaluation Metrics

## ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

Description:

Measures the overlap between the n-grams (e.g., unigrams, bigrams) in the generated summary and a reference summary.

Calculation:

* - ROUGE-N: Measures n-gram overlap.  
  - ROUGE-L: Measures the longest common subsequence.  
  - ROUGE-W: Weighted longest common subsequence.

Interpretation:

High ROUGE scores indicate high similarity to the reference summary.

Pros:

* Widely used and accepted in summarization tasks.  
  Simple to implement and interpret.

Cons:

* Only measures lexical overlap, not semantic meaning.  
  Can be biased towards longer summaries.

Code Example:

Use Case 3: Creating an Executive Summary (Hybrid Approach):

def hybrid\_summary(text, extractive\_ratio=0.5, abstractive\_max\_length=130, abstractive\_min\_length=50):  
 extractive\_text = extractive\_summary(text, ratio=extractive\_ratio)  
 executive\_summary = abstractive\_summary(extractive\_text, max\_length=abstractive\_max\_length, min\_length=abstractive\_min\_length)  
 return executive\_summary  
  
executive\_summary = hybrid\_summary(financial\_document)  
print("\nExecutive Summary (Hybrid Approach):")  
print(executive\_summary)

Use Case 4: Generating Summarized Highlights for Different Sections (Extractive Summarization with TextRank):

from sumy.parsers.plaintext import PlaintextParser  
from sumy.nlp.tokenizers import Tokenizer  
from sumy.summarizers.text\_rank import TextRankSummarizer  
  
def textrank\_summary(text, sentences\_count=3):  
 parser = PlaintextParser.from\_string(text, Tokenizer("english"))  
 summarizer = TextRankSummarizer()  
 summary = summarizer(parser.document, sentences\_count)  
 return " ".join([str(sentence) for sentence in summary])  
  
highlights = textrank\_summary(financial\_document, sentences\_count=3)  
print("\nSummarized Highlights (TextRank):")  
print(highlights)

# Putting It All Together  
Below is the complete code, integrating all use cases:

from gensim.summarization import summarize  
from transformers import BartForConditionalGeneration, BartTokenizer  
from sumy.parsers.plaintext import PlaintextParser  
from sumy.nlp.tokenizers import Tokenizer  
from sumy.summarizers.text\_rank import TextRankSummarizer  
  
financial\_document = """ XYZ Bank has reported a significant increase in its net profit for the first quarter of 2024, reaching $2 billion. This is a 10% increase from the previous quarter, driven by higher interest income and reduced loan loss provisions. The bank's total assets grew to $150 billion, a 5% increase compared to the previous year.  
  
In the same period, XYZ Bank's operating expenses were reduced by 2%, reflecting the bank's efforts in cost management and efficiency improvements. The bank's CEO, John Doe, stated that the strong performance is a result of strategic investments in technology and customer service enhancements.  
  
The bank's loan portfolio expanded by 7%, with significant growth in commercial loans and mortgage lending. The non-performing loan ratio remained stable at 1.2%, indicating strong credit quality. Additionally, the bank's capital adequacy ratio improved to 13.5%, well above the regulatory requirement.  
  
Looking ahead, XYZ Bank plans to continue its focus on digital transformation and expanding its market presence in Asia. The bank is optimistic about maintaining its growth trajectory despite potential economic headwinds. """  
  
def extractive\_summary(text, ratio=0.3):  
 return summarize(text, ratio=ratio)  
  
def abstractive\_summary(text, max\_length=130, min\_length=30):  
 model = BartForConditionalGeneration.from\_pretrained('facebook/bart-large-cnn')  
 tokenizer = BartTokenizer.from\_pretrained('facebook/bart-large-cnn')  
 inputs = tokenizer([text], max\_length=1024, return\_tensors='pt')  
 summary\_ids = model.generate(inputs['input\_ids'], num\_beams=4, max\_length=max\_length, min\_length=min\_length, early\_stopping=True)  
 return tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)  
  
def hybrid\_summary(text, extractive\_ratio=0.5, abstractive\_max\_length=130, abstractive\_min\_length=50):  
 extractive\_text = extractive\_summary(text, ratio=extractive\_ratio)  
 executive\_summary = abstractive\_summary(extractive\_text, max\_length=abstractive\_max\_length, min\_length=abstractive\_min\_length)  
 return executive\_summary  
  
def textrank\_summary(text, sentences\_count=3):  
 parser = PlaintextParser.from\_string(text, Tokenizer("english"))  
 summarizer = TextRankSummarizer()  
 summary = summarizer(parser.document, sentences\_count)  
 return " ".join([str(sentence) for sentence in summary])  
  
# Use Case 1: Extracting Key Points  
key\_points = extractive\_summary(financial\_document, ratio=0.3)  
print("Key Points (Extractive Summarization):")  
print(key\_points)  
  
# Use Case 2: Generating a Brief Summary  
brief\_summary = abstractive\_summary(financial\_document, max\_length=100, min\_length=30)  
print("\nBrief Summary (Abstractive Summarization):")  
print(brief\_summary)  
  
# Use Case 3: Creating an Executive Summary  
executive\_summary = hybrid\_summary(financial\_document)  
print("\nExecutive Summary (Hybrid Approach):")  
print(executive\_summary)  
  
# Use Case 4: Generating Summarized Highlights for Different Sections  
highlights = textrank\_summary(financial\_document, sentences\_count=3)  
print("\nSummarized Highlights (TextRank):")  
print(highlights)