Phase3

Zahra Jamshidi, Sana Bavari

February 2026

1 Compare

For comparison, regular cosine similarity can be used. However, due to the fundamental differences between the two outputs, it is better to employ a different method.
The summary output by the TextRank algorithm consists of exact sentences from the original article placed side by side. On the other hand, the LLM-generated summary does not use the exact sentences from the input article; instead, it constructs its own sentences based on the concepts extracted from the input text. Therefore, a metric is needed that compares the meaning of the summaries, rather than just their lexical or structural similarity.

1.1 Match

For this reason, the Match[1] metric is introduced. $\text{match}(\hat{s}, s) = \frac{\mathbf{B}_\Phi(s) \cdot \mathbf{B}_\Phi(\hat{s})^\top}{\|\mathbf{B}_\Phi(s)\| \ \|\mathbf{B}_\Phi(\hat{s})\|}$
In this metric, before calculating cosine similarity, the embeddings of both summaries are first computed using a BERT[2] model (or generally, any transformer-based network). Cosine similarity is then measured on these embeddings. The Match metric calculates the semantic similarity between two texts, which allows for a more accurate comparison of summaries that differ in wording or structure.The Match value ranges between 0 and 1, where a value closer to 1 indicates a higher degree of semantic similarity between the summaries.

1.2 Code Explaining

```
class SummaryMerger:
    def __init__(self, model_name='all-MiniLM-L6-v2'):
        self.model = SentenceTransformer(model_name)
```

Creates a SentenceTransformer model that converts text to 384-dimensional vectors. all-MiniLM-L6-v2 is a small, efficient model good for semantic understanding.

```
def compare(self, textrank_summary, llm_summary, original_article):
```

```
        embeddings = self.get_embeddings([textrank_summary, llm_summary, original_article])
        emb_tr, emb_llm, emb_article = embeddings

        tr_vs_llm_score = cosine_similarity([emb_tr], [emb_llm])[0][0]

        tr_vs_article_score = cosine_similarity([emb_tr], [emb_article])[0][0]

        llm_vs_article_score = cosine_similarity([emb_llm], [emb_article])[0][0]

        comparison_results = {
            'tr_vs_llm': tr_vs_llm_score,
            'tr_vs_article': tr_vs_article_score,
            'llm_vs_article': llm_vs_article_score
        }
        return comparison_results
```

Calculates how similar summaries are to each other and to the original article.

- tr_vs_llm: Agreement between methods, High score = both found similar main points.
- tr_vs_article: TextRank's faithfulness Measures extractive accuracy (should be high).
- llm_vs_article: LLM's faithfulness Tests if LLM captured meaning (vs. hallucinating).

1.3 Output Example

As an example, the output of the compare function for the first article in the CNN/DailyMail dataset is shown below.

Comparison	**Similarity Score**
TextRank vs. LLM Summary	0.6216
TextRank vs. Original Article	0.8260
LLM vs. Original Article	0.6997

Table 1: Semantic similarity comparison results

In this example, the TextRank algorithm has performed better than the selected LLM.
Output of the TextRank :
Here, Soledad O'Brien takes users inside a jail where many of the inmates are mentally ill. An inmate housed on the "forgotten floor," where many mentally ill inmates are housed in Miami before trial. So, they end up on the ninth floor severely mentally disturbed, but not getting any real help because they're in jail. Even though we were not exactly welcomed with open arms by the guards,

we were given permission to shoot videotape and tour the floor.
Output of the LLM :
inmates with most severe mental illnesses are incarcerated until they're ready to appear in court . most often, they face drug charges or charges of assaulting an officer . they end up on the ninth floor severely mentally disturbed, but not getting real help .

2 Rank

As part of the ranking process, the similarity of each individual sentence in the summaries to the original text is measured, and this process continues toward constructing a combined summary. In this way, the similarity of each sentence from both summarization methods to the entire original text is calculated. These similarities are then stored in a list sorted in descending order (based on similarity) to be reviewed by the merge function for generating the final summary.

2.1 Code Explaining

```
nltk.download('punkt', quiet=True)
tr_sentences = sent_tokenize(textrank_summary)
llm_sentences = sent_tokenize(llm_summary)

all_sentences = []
sources = []

for sent in tr_sentences:
    all_sentences.append(sent)
    sources.append('TextRank')
for sent in llm_sentences:
    all_sentences.append(sent)
    sources.append('LLM')
```

Splits summaries into individual sentences.
Tracks which source (TextRank/LLM) each sentence came from.

```
article_embedding = self.get_embeddings([original_article])
sentence_embeddings = self.get_embeddings(all_sentences)
relevance_scores = cosine_similarity(article_embedding, sentence_embeddings)[0]
```

Creates embeddings for article and all sentences.
relevance_scores: How similar each sentence is to the article's main theme.

```
ranked_list = []

for i, (sentence, source, rel_score) in enumerate(zip(all_sentences, sources
    , relevance_scores)):
```

```
    ranked_list.append({
        'sentence': sentence,
        'source': source,
        'relevance': rel_score,
    })

ranked_list.sort(key=lambda x: x['relevance'], reverse=True)
```

sentence: The actual text
source: Where it came from ('TextRank' or 'LLM')
rel_score: Relevance score (0-1)
Loop iteration: For each sentence, defines these 3 pieces of information.
key=lambda x: x['relevance']: Sort by the 'relevance' value in each dictionary.

2.2 Output Example

Rank	Source	Score	Sentence
1	TextRank	0.8305	Here, Soledad O'Brien takes users inside a jail where many of the inma...
2	TextRank	0.6539	So, they end up on the ninth floor severely mentally disturbed, but no...
3	LLM	0.6446	inmates with most severe mental illnesses are incarcerated until they'...
4	LLM	0.5627	they end up on the ninth floor severely mentally disturbed, but not ge...
5	LLM	0.3782	most often, they face drug charges or charges of assaulting an officer...

Table 2: Top ranked sentences for merging

3 Merge

In the merge function, we pursue several objectives:
1- We aim to select sentences for the final summary that have the highest ranking scores in terms of similarity to the original text.
2- We strive to avoid selecting duplicate sentences (e.g., if a high-scoring sentence appears in both the TextRank and LLM summaries) by using a similarity threshold to filter them.
3- We ensure that the top-ranked sentences selected are merged in the correct semantic order, matching their sequence in the original text.
4- We ensure that not all sentences in the final summary are selected from just one algorithm, maintaining an approximate balance between the number of sentences chosen from the TextRank summary and the LLM summary.

3.1 Code Explanining

```
def merge(self, textrank_summary, llm_summary, original_article, max_sentences=3,
          balance_sources=True, similarity_threshold=0.9):
```

balance_sources=True: Ensure both TextRank and LLM are represented.
similarity_threshold=0.9: Prevent sentences >90.

```
ranked = self.rank_sentences(textrank_summary, llm_summary, original_article)
if not ranked:
    return ""
```

Calls rank_sentences() to get list of (score, sentence, source) tuples.
If empty, returns empty string.

```
selected_sentences = []
selected_embeddings = []
sources_used = {'TextRank': 0, 'LLM': 0}
```

selected_sentences: Final sentences that will be in summary.
selected_embeddings: BERT vectors of those sentences (for similarity check).
sources_used: Counts how many sentences from each source are selected.

```
ranked_sentences = [item[1] for item in ranked]
ranked_embeddings = self.get_embeddings(ranked_sentences)
sentence_to_embedding = {sent: emb for sent, emb in zip(ranked_sentences, ranked_embeddings)
```

1- Extracts the sentences from ranked list.
2- Gets BERT embeddings for ALL sentences.
3- Creates dictionary for quick lookup: "sentence text": [0.1, 0.3, ...], ...

```
for score, sentence, source in ranked:
    if len(selected_sentences) >= max_sentences:
        break

if balance_sources:
    max_per_source = (max_sentences + 1) // 2

    if sources_used[source] >= max_per_source:
        continue
```

Iterates through ranked sentences.
Stops when we have enough sentences.
max_sentences=3, Max 2 TextRank sentence, max 1 LLM sentences (or vice versa based on sentences ranking).

```
if selected_sentences:
    current_embedding = sentence_to_embedding[sentence]
    similarities = cosine_similarity([current_embedding], selected_embeddings)[0]
    max_similarity = np.max(similarities)
```

```
    if max_similarity >= similarity_threshold:
        continue
```

1- Gets BERT vector for current sentence.
2- Calculates similarity with ALL already-selected sentences.
3- If ANY similarity 0.9, skips this sentence.

```
selected_sentences.append(sentence)
selected_embeddings.append(sentence_to_embedding[sentence])
sources_used[source] += 1
```

Adds sentence text to final list.
Stores its embedding for future similarity checks.
Updates count for this source type.

```
def reorder_by_semantic_position(self, selected_sentences, original_article):
```

Reorder selected sentences to match where their content appears in the original article.

```
chunk_size = 200
article_chunks = [original_article[i:i+chunk_size]
                  for i in range(0, len(original_article), chunk_size)]
```

chunk_embeddings: Convert each 200-char chunk to 384D vector.
sentence_embeddings: Convert each selected sentence to 384D vector.

```
chunk_embeddings = self.get_embeddings(article_chunks)
sentence_embeddings = self.get_embeddings(selected_sentences)
```

Get embeddings.

```
sentence_positions = []

for i, sent in enumerate(selected_sentences):

    similarities = cosine_similarity([sentence_embeddings[i]], chunk_embeddings)[0]

    best_chunk_idx = np.argmax(similarities)
    sentence_positions.append((best_chunk_idx, sent))
```

Ex: i=0
1- sentence_embeddings[0]: Vector for first sentence.
2- [sentence_embeddings[0]]: Reshape to (1,384) for cosine_similarity.
3- chunk_embeddings: All chunk vectors.
4- cosine_similarity(...): Compare sentence to all chunks
5- Result: similarities = [0.42, 0.87, 0.15,...]
· Sentence is 42% similar to chunk 0
· 87% similar to chunk 1

· 15% similar to chunk 2, etc.
6- np.argmax([0.42, 0.87, 0.15,...]) returns 1, Sentence is most similar to chunk 1.
7- Stores tuple: (1, Sentence's Text).

```
sentence_positions.sort(key=lambda x: x[0])
return [sent for pos, sent, score in sentence_positions]
```

Lower chunk index = earlier in article.
Extract sentence text from each tuple.

3.2 Output Example

The output of the merge function for the example examined in the previous stages:
Here, Soledad O'Brien takes users inside a jail where many of the inmates are mentally ill. An inmate housed on the "forgotten floor," where many mentally ill inmates are housed in Miami before trial. inmates with most severe mental illnesses are incarcerated until they're ready to appear in court . So, they end up on the ninth floor severely mentally disturbed, but not getting any real help because they're in jail.

References

[1] H. Xie, Z. Qin, G. Y. Li, and B.-H. Juang, "Deep Learning Enabled Semantic Communication Systems," *IEEE Transactions on Signal Processing*, vol. 69, 2021.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota, Jun. 2019, pp. 4171–4186.