Phase1

Zahra Jamshidi, Sana Bavari

February 2026

1 Dataset

For the task of text summarization, a variety of datasets exist. We opted to utilize the CNN/DailyMail[1] dataset, which contains more than 300,000 unique news articles written by professional journalists. The dataset is annotated with extractive and abstractive summaries, specifically providing brief, multi-sentence summaries (commently comprising three sentences) for each article, thereby enabling straightforward comparative analysis.

2 TextRank

The TextRank[1] algorithm is a classic, graph-based summarization method. It models sentences as nodes in a graph, connecting them based on similarity, and ranks them using a voting mechanism. The goal is to identify the most central or important sentences.
How it works:
1-Split the input text into sentences.
2-Create a vector representation (embedding) for each sentence to capture its meaning. Calculate the similarity (e.g., cosine similarity) between every pair of sentence vectors.
3-Build a graph where sentences are nodes, and edges are weighted by similarity scores.
4-Run the ranking algorithm on this graph to score each sentence.
5-Select the top-ranked sentences to form the summary.

Steps 4 and 5 in more detail:
A "vote": When two sentences are similar, they "vote" for each other.
Weight of the vote: The strength of the similarity determines how powerful that vote is.
Importance is contagious: A sentence becomes important if it receives strong votes from other sentences that are themselves important.

[1] https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-dailymail

3 Code Explaining

```
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
```

Downloads NLTK's sentence tokenizer (punkt) and common English stopwords.

```
class TextRankSummarizer:
    def __init__(self, word_embeddings_path='glove.6B.100d.txt'):
        self.word_embeddings = self.load_embeddings(word_embeddings_path)
        self.stop_words = set(stopwords.words('english'))
```

Creates a class for TextRank summarization. Initializes by loading word embeddings and stopwords list. stopwords.words('english') returns words like "the", "a", "is" that carry little meaning.

```
def load_embeddings(self, path):
    embeddings = {}
    with open(path, 'r', encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], dtype='float32')
            embeddings[word] = vector
    return embeddings
```

Loads pre-trained GloVe word embeddings. Each line format: "the 0.123 0.456 ... -0.789" (word + 100 numbers). Creates dictionary: "the": [0.123, 0.456, ...], "cat": [...], ...

```
def sentence_to_vector(self, sentence):
    words = word_tokenize(sentence.lower())
    words = [w for w in words if w.isalpha() and w not in self.stop_words]
```

Converts sentence to lowercase and splits into words. Keeps only alphabetic words that aren't stopwords.

```
if len(words) == 0:
    return np.zeros((100,))

vector_sum = np.zeros((100,))
for w in words:
    if w in self.word_embeddings:
        vector_sum += self.word_embeddings[w]
return vector_sum / len(words)
```

Averages word vectors to create a single 100-dimensional sentence vector. This vector represents the overall meaning of the sentence.

```
def summarize(self, text, num_sentences=3):
    sentences = sent_tokenize(text)
    if len(sentences) <= num_sentences:
        return text
    sentence_vectors = [self.sentence_to_vector(sent) for sent in sentences]
```

Splits article into individual sentences. If article has $\leq$3 sentences, return it unchanged. Converts each sentence to its vector representation.

```
sim_matrix = np.zeros((len(sentences), len(sentences)))
for i in range(len(sentences)):
    for j in range(len(sentences)):
        if i != j:
            sim = cosine_similarity(
                sentence_vectors[i].reshape(1, -1),
                sentence_vectors[j].reshape(1, -1)
            )
            sim_matrix[i][j] = sim[0][0]
```

Creates similarity matrix where sim_matrix[i][j] = similarity between sentences i and j. Uses cosine similarity: measures angle between vectors (1=identical, 0=unrelated). reshape(1, -1) converts 1D array to 2D for sklearn compatibility.

```
nx_graph = nx.from_numpy_array(sim_matrix)
scores = nx.pagerank(nx_graph)
```

Builds graph where sentences are nodes, similarity scores are edge weights. PageRank algorithm: Sentences that are similar to other important sentences get higher scores.

```
ranked_sentences = sorted(((scores[i], i, sent) for i, sent in enumerate(sentences)), revers
top_sentence_indices = sorted([ranked_sentences[i][1] for i in range(num_sentences)])
```

Sorts sentences by their PageRank score (highest first). Gets indices of top 3 sentences, then sorts them by original position.

```
summary = ' '.join([sentences[i] for i in top_sentence_indices])
```

Reconstructs summary from selected sentences in original order.

References

[1] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into texts. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.