Assignment 1 Analysis Report

Nakul Jadeja

Question 1: Cosine Similarity Implementation

I implemented the cosine similarity function using the standard formula:

cosine_similarity = $(A \cdot B) / (|A| \times |B|)$

Where:

- $A \cdot B$ is the dot product of vectors A and B
- $|A|$ and $|B|$ are the magnitudes of vectors A and B

Key Details of Implementation: -

- Union of vocabularies listed to get all listed words from both documents.
- For calculating the dot product, I multiplied the counts from both documents.
- For calculating the magnitude, I use the Euclidean norm for both vectors.,
- Also, added a check to prevent division by zero

Question 2 : - TF-IDF Implementation

TF-IDF(t,d) = TF(t,d) × IDF(t) IDF(t) = $\log_2$(N / DF(t))

I used the above formula to implement TF-IDF.

Where:

- TF(t,d) = term frequency of term t in document d
- DF(t) = document frequency of term t
- N = total number of documents (10^6)
- Log base 2 as specified

Implementation Details: -

- Used math.log2(N / df) with N = 10^6
- Skip words not present in the corpus (df = 0)

- TF-IDF = tf × idf for each term

Question 3: - BM 25 Bug Fix

The most critical error in this implementation was using tf(query term frequency) instead of candidate_tf(candidate document frequency) in the BM 25 formula.

The bug: -

```
# WRONG (original code):
for word, tf in query_word_cnt_dict.items():
    # ... other code ...
    numerator = tf * (k1 + 1)  # Using query tf instead of candidate tf
    denominator = tf + k1 * (1 - b + b * (candidate_length / avgdl))  # Same error
```

The fix: -

```
# CORRECT (fixed code):
for word, query_tf in query_word_cnt_dict.items():
    candidate_tf = candidate_word_cnt_dict.get(word, 0)  # Get candidate tf
    # ... other code ...
    numerator = candidate_tf * (k1 + 1)  # Use candidate tf
    denominator = candidate_tf + k1 * (1 - b + b * (candidate_length / avgdl))  # Use candidate tf
```

As we know that BM 25 should score based on term frequency in the candidate document not query, this fix was important. Using query tf would make all candidates with same query term to get identical scores, breaking the ranking logic. It would hence violate the BM 25 principle.

Question 4: Typo Identification

Typo Found:

Location: In the run_retrieval_algorithm function, line 1:

# WRONG: base_path = pathlib.Path("cs589/assignment1/dataset/")

# CORRECT: base_path = pathlib.Path("cs589assignment1/dataset/")

This typo would cause a FileNotFoundError when the function tries to access files, preventing the notebook from running to completion.

Question 5: BM 25 Parameter Tuning

Methodology:

I would systematically test different combinations of k1 and b parameters:

- **k1 range**: 0.5 to 5.0 (step 0.5) - controls term frequency saturation
- **b range**: 0.0 to 1.0 (step 0.1) - controls document length normalization

k1 Parameter (Term Frequency Saturation):

- **Lower k1 (0.5-1.5)**: Faster saturation, less emphasis on repeated terms
- **Higher k1 (3.0-5.0)**: Slower saturation, more emphasis on repeated terms
- **Optimal range**: Typically, 1.2-2.0 for most collections

Parameter (Document Length Normalization):

- **b = 0**: No length normalization (like TF-IDF)
- **b = 1**: Full length normalization
- **Optimal range**: Typically, 0.6-0.8 for most collections

Conclusion

This assignment demonstrates the evolution from simple cosine similarity through TF-IDF to the more sophisticated BM25 algorithm. Each method addresses limitations of the previous:

1. **Cosine Similarity**: Basic but effective for short texts
2. **TF-IDF**: Adds term weighting but ignores document length
3. **BM25**: Addresses document length and term frequency saturation