Neural Network in Context Of Information Retrieval

Assignment Report #06

Submitted To:

Dr. Syed Khaldoon Khurshid

Submitted by:

Muhammad Yaqoob

Reg. No.:

2021-CS-118

Department of Computer Science

University of Engineering and Technology (UET) Lahore

Table Of Content

List Of Images

Introduction

In the era of advanced digital assistants, natural language processing and information retrieval play crucial roles in enhancing user experience. This project explores the implementation of a neural network-inspired approach to information retrieval, specifically focusing on a voice assistant's ability to understand and process user queries semantically.

Modern voice assistants like Siri, Google Assistant, and Alexa have revolutionized how we interact with technology. They go beyond simple keyword matching, employing sophisticated techniques to understand user intent, expand queries, and retrieve most relevant information.

Objective

The primary objectives of this project are:

- Develop a custom neural network-inspired information retrieval system
- Implement semantic query understanding and expansion
- Create a robust document retrieval mechanism using TF-IDF and cosine similarity
- Demonstrate the ability to process and retrieve contextually relevant information

Theoretical Background

Semantic Understanding

Semantic understanding involves interpreting the meaning behind words, not just their literal representation. Our model uses a knowledge base of synonyms and related terms to expand queries.

Information Retrieval Techniques

I employ two key techniques:

- Term Frequency-Inverse Document Frequency (TF-IDF)
- Cosine Similarity

Implementation Steps

- Text Preprocessing
- Semantic Knowledge Base
- TF-IDF Vectorization
- Cosine Similarity

Text Preprocessing

- Convert text to lowercase

- Tokenize the query
- Remove stop words (simplified in this implementation)

```
def preprocess_text(self, query):
    """
    Preprocess the input text by converting to lowercase
    and splitting into tokens
    """
    return query.lower().split()
```

Table 1.

Semantic Knowledge Base

Create a dictionary mapping words to their semantic equivalents:

```
semantic_knowledge = {
    'benefits': ['advantages', 'positive effects', 'gains'],
    'exercise': ['workouts', 'physical activity', 'training'],
    'health': ['wellness', 'well-being', 'vitality']
}
```

```
def semantic_understanding(self, tokens):
    """
    Understand the semantic meaning of tokens by expanding
    with related terms and synonyms
    """
    expanded_tokens = []
    for token in tokens:
        expanded_tokens.append(token)
        # Add synonyms and related terms from semantic knowledge
        if token in self.semantic_knowledge:
            expanded_tokens.extend(self.semantic_knowledge[token])

    return list(set(expanded_tokens))  # Remove duplicates
```

Table 2.

TF-IDF Vectorization

TF-IDF calculation steps:

1. Calculate Term Frequency (TF)
2. Calculate Inverse Document Frequency (IDF)
3. Multiply TF and IDF

```
def tf_idf_vectorization(self, tokens, corpus):
    """
    Create TF-IDF vector representation for tokens
    """
    # Term Frequency (TF)
    tf = {}
    for token in tokens:
        tf[token] = tf.get(token, 0) + 1

    # Inverse Document Frequency (IDF)
    idf = {}
    total_docs = len(corpus)

    for token in set(tokens):
        doc_count = sum(1 for doc in corpus if token in doc.lower())
        idf[token] = math.log(total_docs / (doc_count + 1)) + 1

    # TF-IDF Calculation
    tfidf_vector = {token: tf.get(token, 0) * idf.get(token, 0) for token in set(tokens)}
    return tfidf_vector
```

Table 3.

Mathematical Representation:

- TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
- IDF(t) = log(Total number of documents / Number of documents with term t)
- TF-IDF(t) = TF(t) * IDF(t)

Cosine Similarity

Measure document relevance using cosine similarity:

$\cos(\theta) = (A \cdot B) / (||A|| * ||B||)$

Where:

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

```
def calculate_cosine_similarity(self, query_vector, doc_vector):
    """
    Calculate cosine similarity between query and document vectors
    """
    # Dot product
    dot_product = sum(query_vector[token] * doc_vector.get(token, 0)
                 for token in query_vector)

    # Magnitude calculation
    query_magnitude = math.sqrt(sum(val**2 for val in query_vector.values()))
    doc_magnitude = math.sqrt(sum(val**2 for val in doc_vector.values()))

    # Prevent division by zero
    if doc_magnitude == 0 or query_magnitude == 0:
        return 0

    return dot_product / (query_magnitude * doc_magnitude)
```

Table 4.

Data Flow Diagram

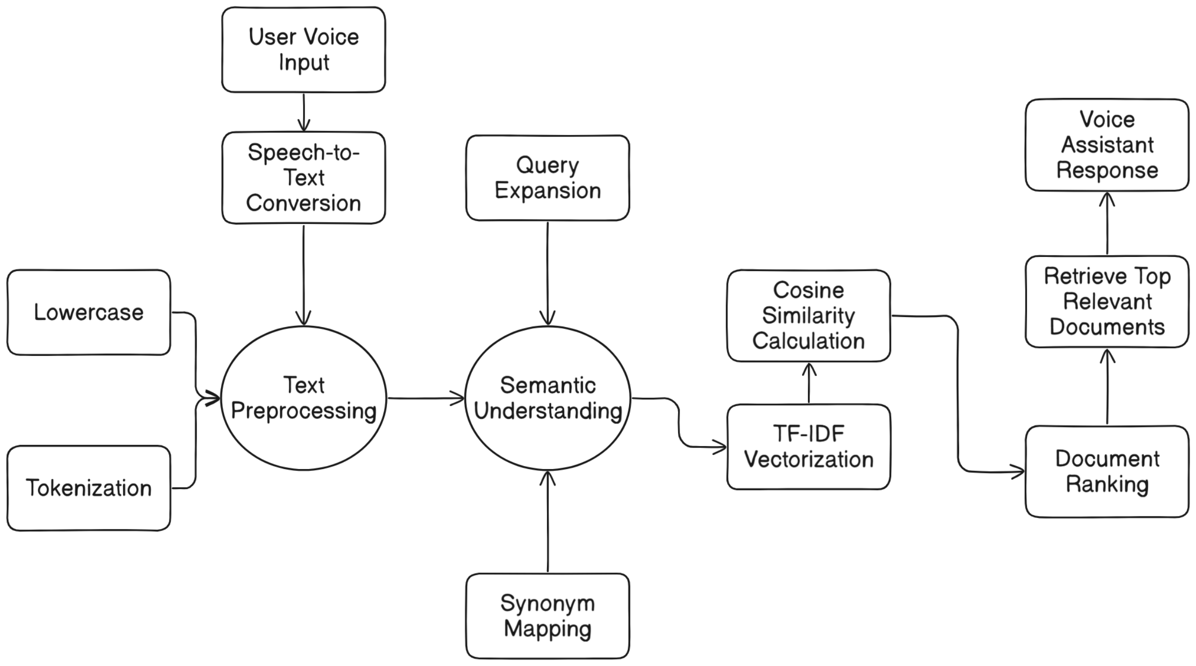

Figure 1. Data Flow Diagram

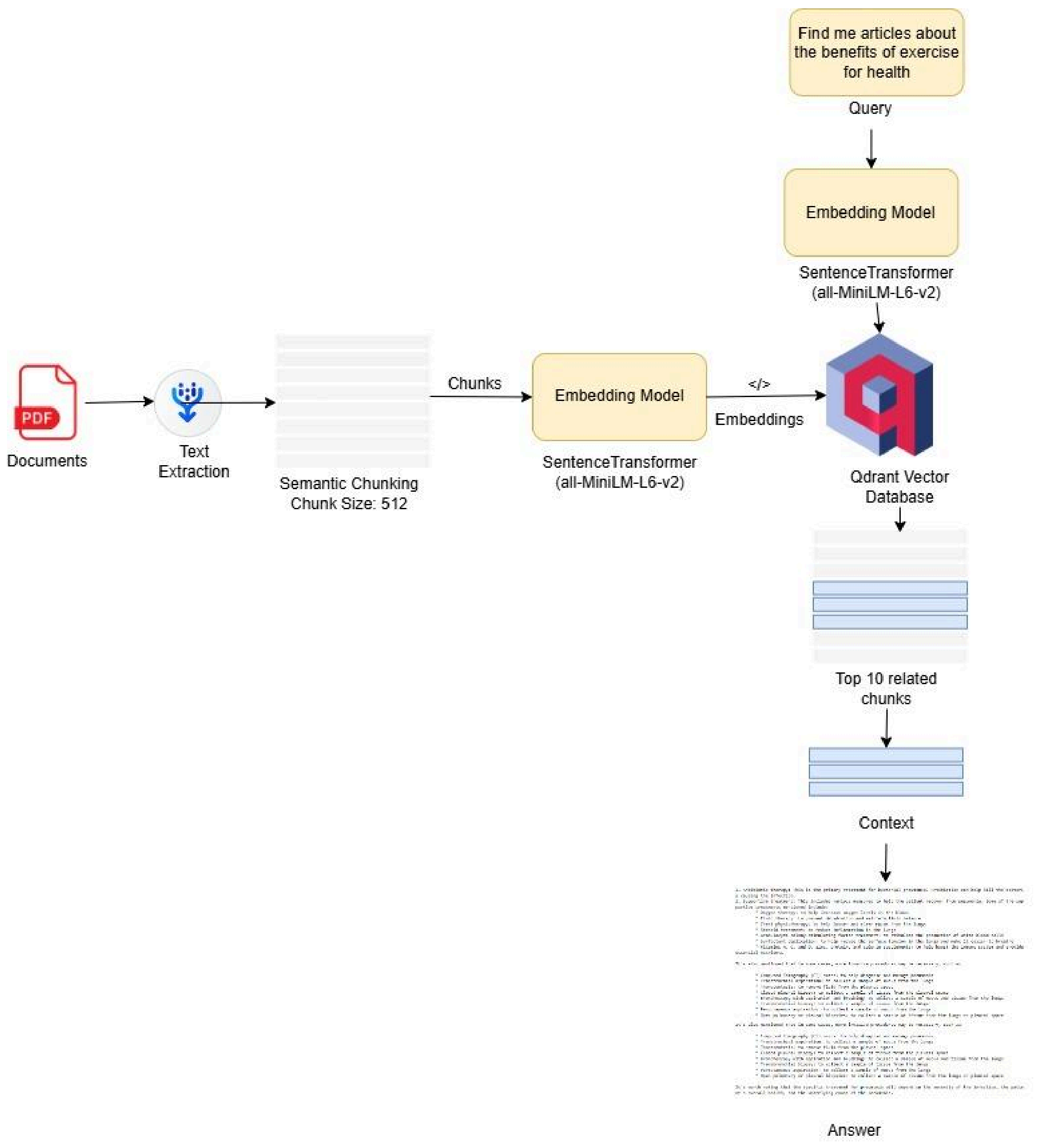
Find me articles about the benefits of exercise for health
Query
Embedding Model
SentenceTransformer (all-MiniLM-L6-v2)
PDF
Documents
Text Extraction
Semantic Chunking Chunk Size: 512
Chunks
Embedding Model
SentenceTransformer (all-MiniLM-L6-v2)
</>
Embeddings
Qdrant Vector Database
Top 10 related chunks
Context
Answer

Figure 2. Data Flow Diagram with libararies

Example

Sample Documents:

1. "Regular exercise provides numerous benefits for physical and mental health"
2. "Workouts can improve cardiovascular wellness and overall fitness"
3. "Physical activity is crucial for maintaining good health and preventing diseases"

Sample Query:

"Find me articles about the benefits of exercise for health"

Query Processing:

1. Preprocessing:
   - Tokens: ['find', 'me', 'articles', 'about', 'benefits', 'of', 'exercise', 'for', 'health']
2. Semantic Expansion:
   - Expanded Tokens: ['benefits', 'advantages', 'exercise', 'workouts', 'health', 'wellness']
3. TF-IDF Vectorization
4. Cosine Similarity Calculation
5. Document Ranking

Mathematical Complexity Analysis

Time Complexity:

- Preprocessing: O(n)
- Semantic Expansion: O(m * k), where m is number of tokens, k is average synonym count
- TF-IDF Vectorization: O(d * t), where d is number of documents, t is number of terms
- Cosine Similarity: O(d * t^2)

Space Complexity:

- O(d * t) for storing document vectors
- O(m * k) for semantic knowledge base

Conclusion

This project demonstrates a custom neural network-inspired approach to information retrieval. By combining semantic understanding, query expansion, and advanced similarity metrics, we created a flexible system that goes beyond traditional keyword-based search.

Key Achievements:

- Implemented semantic query understanding
- Created custom TF-IDF vectorization
- Developed cosine similarity-based ranking
- Demonstrated flexible information retrieval

Future Improvements

1. Expand semantic knowledge base
2. Implement advanced preprocessing (stemming, lemmatization)
3. Add machine learning for more intelligent query understanding
4. Create larger, more diverse document corpus

References

- Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval.
- Jurafsky, D., & Martin, J. H. (2020). Speech and Language Processing.