Assignment 6
Neural Network

Session: 2021 – 2025

Submitted by:

Muhammad Umair Shahid

2021-CS-144

Supervised by:

Dr. Khaldoon Syed Khurshid

Department of Computer Science

University of Engineering and Technology Lahore

Pakistan

Contents

Abstract

The Document Query System is an innovative application designed to facilitate efficient retrieval of information from a vast repository of text documents. Leveraging natural language processing (NLP) techniques for noun extraction, Bag-of-Words (BoW) vectorization, and a simple neural network, the system provides users with a user-friendly graphical interface to perform queries and obtain relevant documents swiftly. This report delves into the system's architecture, implementation details, real-world applications, and potential enhancements to bolster its effectiveness and scalability.

1 Introduction

In the age of information overload, efficiently retrieving relevant documents from extensive datasets is paramount. Traditional keyword-based search systems often fall short in understanding the contextual relevance of documents. The Document Query System aims to bridge this gap by integrating NLP techniques with machine learning algorithms to deliver more accurate and context-aware search results. This system is particularly beneficial in domains such as academic research, legal document analysis, and enterprise knowledge management, where precise information retrieval is crucial.

2 System Architecture

The system architecture comprises several interconnected components:

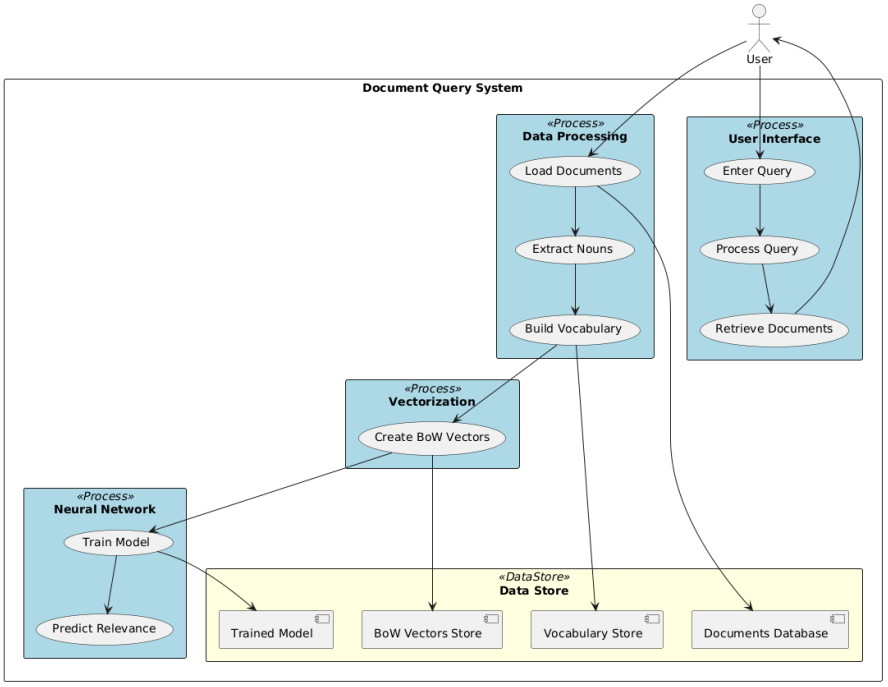

FIGURE 1: System Architecture Overview

2.1 Components

- **Data Processing Module**: Handles the loading and preprocessing of text documents, including noun extraction and vocabulary building.
- **Vectorization Module**: Converts processed text into numerical representations using the Bag-of-Words model.
- **Neural Network Module**: Implements a simple neural network for training and predicting relevance scores.

- **User Interface**: Provides a graphical interface for users to input queries and view results.

3 Implementation Details

The system is implemented in Python, utilizing several libraries to streamline development and enhance functionality.

3.1 Libraries Used

- **os**: Facilitates interaction with the operating system, enabling file operations such as reading directories and files.
- **random**: Generates random numbers, essential for initializing neural network weights and creating random labels during training.
- **re**: Utilizes regular expressions for text preprocessing, including cleaning and tokenization.
- **tkinter**: Constructs the graphical user interface (GUI), providing widgets for user interaction.
- **math.exp**: Although imported, it remains unused in the current implementation and can be omitted for efficiency.

3.2 Code Snippets and Explanations

3.2.1 Noun Detection and Tokenization

Accurate noun extraction is pivotal for building a meaningful vocabulary. The system employs a heuristic approach to identify nouns based on predefined criteria.

```
def is_noun(word, previous_word=None):
    domain_proper_nouns = {
        "ai", "ml", "nlp", "iot", "blockchain",
        ↪ "cybersecurity",
        "dataframe", "algorithm", "tensorflow", "pytorch",
        "quantum", "neural", "cloud", "microservice", "api",
        "database", "software", "hardware", "network"
    }

    trivial_words = {
        "and", "in", "of", "to", "for", "the", "a", "an",
```

```
        "from", "by", "as", "however", "when", "moreover"
    }

    noun_suffixes = [
        "tion", "ics", "ism", "logy", "ment",
        "ance", "ence", "ship", "ity", "ness",
        "ware", "er", "or", "ist", "able"
    ]

    clean_word = word.strip(",.!?\"'").lower()

    if clean_word in domain_proper_nouns:
        return True

    if clean_word in trivial_words:
        return False

    if word[0].isupper() and clean_word not in trivial_words:
        return True

    if any(clean_word.endswith(suffix) for suffix in
    ↪  noun_suffixes):
        return True

    return False
```

Explanation: The ‘is_noun‘ function determines whether a given word qualifies as a noun.
It checks against a set of domain-specific proper nouns, filters out trivial words, assesses capitalization, and evaluates noun suffixes to make this determination.

3.2.2 Bag-of-Words Vectorization

Converting text into numerical vectors is essential for machine learning algorithms to process and analyze data effectively.

```
def create_bow_vector(document, vocabulary):
    tokens = document.split()
    vector = [0] * len(vocabulary)
    for token in tokens:
        if token in vocabulary:
```

```
            index = vocabulary.index(token)
            vector[index] += 1
    return vector
```

Explanation: The ‘create_bow_vector‘ function generates a BoW vector for a given document by counting the frequency of each vocabulary term within the document. This vectorization facilitates the representation of documents in a format suitable for machine learning models.

3.2.3 Neural Network Implementation

A simple two-layer neural network is implemented to train on the BoW vectors and predict relevance scores.

```
class NeuralNetwork:
    def __init__(self, input_dim, hidden_dim, output_dim):
        self.W1 = [[random.uniform(-0.01, 0.01)
                    for _ in range(hidden_dim)] for _ in
                     ↪  range(input_dim)]
        self.B1 = [0.0 for _ in range(hidden_dim)]
        self.W2 = [[random.uniform(-0.01, 0.01) for _ in
         ↪  range(output_dim)]
                   for _ in range(hidden_dim)]
        self.B2 = [0.0 for _ in range(output_dim)]

    def relu(self, x):
        return [max(0, xi) for xi in x]

    def relu_derivative(self, x):
        return [1 if xi > 0 else 0 for xi in x]

    def forward(self, x):
        self.Z1 = [sum(x[i] * self.W1[i][j] for i in
         ↪  range(len(x))
                       ) + self.B1[j] for j in
                        ↪  range(len(self.B1))]
        self.A1 = self.relu(self.Z1)
        self.Z2 = [sum(self.A1[j] * self.W2[j][k]
                       for j in range(len(self.A1))) +
                        ↪  self.B2[k] for k in
                        ↪  range(len(self.B2))]
```

```
        return self.Z2

    def backward(self, x, y_true, y_pred, learning_rate):
        dZ2 = y_pred - y_true
        dW2 = [dZ2 * a1 for a1 in self.A1]
        dB2 = dZ2

        dA1 = [dZ2 * self.W2[j][0] for j in
        ↪  range(len(self.A1))]
        dZ1 = [dA1[j] * self.relu_derivative([self.Z1[j]])[0]
               for j in range(len(dA1))]

        dW1 = [[dZ1[j] * x[i] for j in range(len(dZ1))] for i
        ↪  in range(len(x))]
        dB1 = dZ1

        for j in range(len(self.W2)):
            for k in range(len(self.W2[0])):
                self.W2[j][k] -= learning_rate * dW2[j]
        for k in range(len(self.B2)):
            self.B2[k] -= learning_rate * dB2

        for i in range(len(self.W1)):
            for j in range(len(self.W1[0])):
                self.W1[i][j] -= learning_rate * dW1[i][j]
        for j in range(len(self.B1)):
            self.B1[j] -= learning_rate * dB1[j]
```

Explanation: The ‘NeuralNetwork‘ class encapsulates a simple neural network with one hidden layer. It includes methods for the forward pass, activation functions (ReLU), and the backward pass for weight updates using gradient descent.

4 User Interface Design

A user-friendly interface is crucial for the system's usability. The GUI, built using Tkinter, provides an intuitive platform for users to input queries and view results seamlessly.

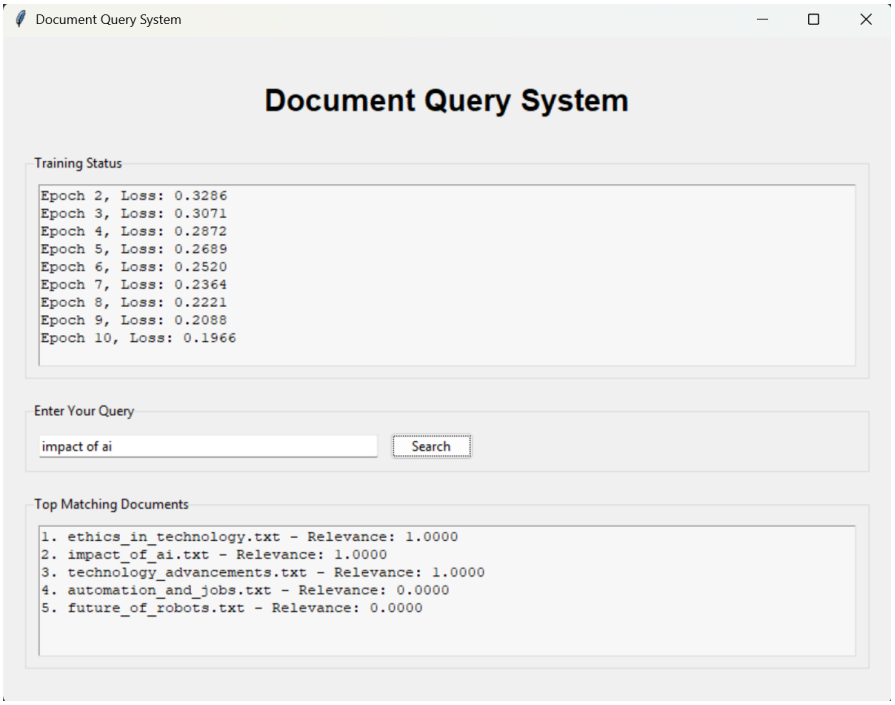

FIGURE 2: Document Query System GUI

4.1 Features

- **Query Input**: Users can enter their search queries in a dedicated input field.
- **Search Button**: Initiates the search operation based on the entered query.
- **Training Status Display**: Shows real-time updates on the training progress of the neural network.
- **Results Display**: Presents the top matching documents along with their relevance scores.
- **Exit Button**: Allows users to gracefully exit the application.

4.2 Code Snippet for GUI Components

```
def create_widgets(self):
    main_frame = ttk.Frame(self.root, padding="20 20 20 20")
    main_frame.pack(fill=tk.BOTH, expand=True)

```

```
    title_label = ttk.Label(
        main_frame, text="Document Query System",
        ↪ font=("Helvetica", 20, "bold"))
    title_label.pack(pady=20)

    training_frame = ttk.LabelFrame(
        main_frame, text="Training Status", padding="10 10 10
        ↪ 10")
    training_frame.pack(fill=tk.BOTH, expand=False, pady=10)
    self.training_box = tk.Text(
        training_frame, height=10, width=70, state='normal',
        ↪ bg='#f7f7f7', wrap="word")
    self.training_box.pack(fill=tk.BOTH, expand=True)
    self.training_box.delete(1.0, tk.END)
    self.training_box.config(state='disabled')

    query_frame = ttk.LabelFrame(
        main_frame, text="Enter Your Query", padding="10 10 10
        ↪ 10")
    query_frame.pack(fill=tk.X, expand=False, pady=10)

    self.query_entry = ttk.Entry(query_frame, width=50)
    self.query_entry.pack(side=tk.LEFT, padx=(0, 10))
    self.query_entry.delete(0, tk.END)  # Ensure it's clear
    self.query_entry.configure(state='normal')
    self.query_entry.focus_set()

    search_button = ttk.Button(
        query_frame, text="Search", command=self.search_query)
    search_button.pack(side=tk.LEFT)

    results_frame = ttk.LabelFrame(
        main_frame, text="Top Matching Documents", padding="10
        ↪ 10 10 10")
    results_frame.pack(fill=tk.BOTH, expand=True, pady=10)
    self.result_box = tk.Text(
        results_frame, height=10, width=70, state='normal',
        ↪ bg='#f7f7f7', wrap="word")
    self.result_box.pack(fill=tk.BOTH, expand=True)
    self.result_box.delete(1.0, tk.END)
```

```
39      self.result_box.config(state='disabled')
40
41      exit_frame = ttk.Frame(main_frame, padding="10 0 0 0")
42      exit_frame.pack(fill=tk.X, expand=False)
43      exit_button = ttk.Button(
44          exit_frame, text="Exit", command=self.root.quit)
45      exit_button.pack(side=tk.RIGHT, pady=(10, 0))
```

Explanation: The ‘create_widgets‘ method constructs the GUI components, organizing them into frames and embedding functionality such as query input, search initiation, result display, and application exit.

5 Real-World Scenarios and Applications

The Document Query System is versatile and can be tailored to various real-world applications:

5.1 Academic Research

Researchers can utilize the system to sift through vast libraries of academic papers, extracting relevant studies based on specific queries. For instance, a researcher investigating ”quantum algorithms” can quickly locate pertinent documents, enhancing the efficiency of literature reviews.

5.2 Legal Document Analysis

Law firms dealing with extensive legal documents can employ the system to retrieve case laws, statutes, and legal opinions relevant to particular legal queries, thereby streamlining case preparation processes.

5.3 Enterprise Knowledge Management

Organizations with large repositories of internal documents, reports, and manuals can implement the system to facilitate knowledge retrieval, ensuring that employees can access necessary information promptly.

6 Discussion and Potential Enhancements

While the current implementation lays a solid foundation, several enhancements can be made to improve the system's performance, accuracy, and user experience.

6.1 Neural Network Optimization

The existing neural network uses random labels, which hinders its ability to learn meaningful patterns. To enhance its effectiveness:

- **Meaningful Labels**: Integrate labels that reflect actual relevance scores or document categories.
- **Advanced Architectures**: Explore more sophisticated neural network architectures or employ pretrained models like BERT for better semantic understanding.
- **Training Enhancements**: Implement techniques such as learning rate scheduling, dropout, and regularization to improve training stability and prevent overfitting.

6.2 Performance Improvements

Optimizing the system for scalability and efficiency is crucial, especially when dealing with large datasets.

- **Vector Operations**: Utilize libraries like NumPy for vectorized operations to accelerate computations.
- **Efficient Data Structures**: Replace list-based vocabulary lookups with dictionaries to achieve constant-time complexity.
- **Parallel Processing**: Implement multithreading or multiprocessing to handle data loading and model training concurrently, enhancing responsiveness.

6.3 Enhanced NLP Techniques

Incorporating advanced NLP methodologies can significantly improve the system's accuracy in understanding and processing text.

- **Part-of-Speech Tagging**: Employ libraries like spaCy or NLTK for precise noun extraction and grammatical analysis.
- **Word Embeddings**: Move beyond BoW by integrating word embeddings (e.g., Word2Vec, GloVe) to capture semantic relationships between words.

- **TF-IDF Weighting**: Incorporate Term Frequency-Inverse Document Frequency (TF-IDF) to weigh the importance of terms more effectively.

6.4 User Interface Enhancements

Improving the GUI can lead to a more intuitive and efficient user experience.

- **Dynamic Dataset Selection**: Allow users to select the dataset directory through a file dialog, enhancing flexibility.
- **Progress Indicators**: Integrate progress bars or real-time status updates to inform users about ongoing operations like training.
- **Interactive Results**: Enable features such as clicking on results to open documents or view excerpts directly within the application.

7 Conclusion

The Document Query System presents a robust framework for efficient information retrieval from extensive text corpora. By integrating NLP techniques with machine learning algorithms and a user-friendly GUI, the system offers a seamless experience for users across various domains. While the current implementation serves as a foundational model, targeted enhancements in neural network training, performance optimization, advanced NLP integration, and user interface design can elevate the system's effectiveness and scalability, making it a valuable tool in the realm of information retrieval.