Information Retrieval and Neural Network Relevance Scoring

Overview

This script performs a series of tasks for retrieving and ranking documents relevant to a user query. It includes text preprocessing, keyword extraction, query expansion using synonyms, document retrieval, and a basic neural network to rank the relevance of retrieved documents.

Code Breakdown

1. Imports and Setup

```python
import numpy as np
import nltk
from nltk.corpus import wordnet
import os
nltk.download('punkt_tab')
```

- `numpy`: Used for numerical operations, especially in the neural network.
- `nltk`: Natural Language Toolkit for text preprocessing and synonym extraction.
- `os`: Allows file navigation and reading.
- `nltk.download('punkt_tab')`: Ensures necessary NLTK data is downloaded.

---

2. Text Preprocessing

```python
def preprocess_text(query):
    words = nltk.word_tokenize(query.lower())
    return words
```

- Tokenizes the query into words.
- Converts the text to lowercase for uniformity.

---

3. Keyword Identification

```python
def identify_keywords(words):
    important_keywords = [word for word in words if word.isalpha()]
    return important_keywords
```

- Filters out non-alphabetic tokens, retaining only meaningful words.

---

4. Synonym Expansion

```
def get_synonyms(word):
    synonyms = set()
    for syn in wordnet.synsets(word):
        for lemma in syn.lemmas():
            synonyms.add(lemma.name())
    return list(synonyms)

def expand_query(keywords):
    expanded_query = []
    for word in keywords:
        expanded_query.append(word)
        synonyms = get_synonyms(word)
        expanded_query.extend(synonyms)
    return expanded_query
```

- `get_synonyms`: Uses WordNet to retrieve synonyms for a given word.
- `expand_query`: Adds the original keywords and their synonyms to the expanded query.

---

5. Document Retrieval

```
def retrieve_documents(expanded_query, folder_path):
    relevant_documents = []
    for filename in os.listdir(folder_path):
        if filename.endswith('.txt'):
            with open(os.path.join(folder_path, filename), 'r', encoding='utf-8')
as file:
                doc_text = file.read().lower()
                for term in expanded_query:
                    if term in doc_text:
                        relevant_documents.append((filename, doc_text[:200]))
                        break
    return relevant_documents
```

- Searches for `.txt` files in the specified folder.
- Checks if any term in the expanded query appears in the document.
- Returns the filenames and snippets of matching documents.

---

6. Relevance Scoring with Neural Network

6.1 Generating Random Relevance Scores

```
def calculate_relevance_score(relevant_docs):
    num_docs = len(relevant_docs)
    X = np.random.rand(num_docs, 10)
    return X
```

- Generates a random feature matrix `X` for the documents.

6.2 Neural Network Functions

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)
```

- Implements the Sigmoid activation function and its derivative.

6.3 Training the Neural Network

```
def train_neural_network(X, y, epochs, learning_rate):
    input_size = X.shape[1]
    hidden_size = 8
    output_size = 1
    W_input = np.random.randn(input_size, hidden_size) * 0.1
    B_input = np.zeros((1, hidden_size))
    W_output = np.random.randn(hidden_size, output_size) * 0.1
    B_output = np.zeros((1, output_size))

    for epoch in range(epochs):
        hidden_layer_input = np.dot(X, W_input) + B_input
        hidden_layer_output = sigmoid(hidden_layer_input)
        output_layer_input = np.dot(hidden_layer_output, W_output) + B_output
        output = sigmoid(output_layer_input)
        output_error = y - output
        output_delta = output_error * sigmoid_derivative(output)
        hidden_error = np.dot(output_delta, W_output.T)
        hidden_delta = hidden_error * sigmoid_derivative(hidden_layer_output)
        W_output += np.dot(hidden_layer_output.T, output_delta) * learning_rate
        B_output += np.sum(output_delta, axis=0) * learning_rate
        W_input += np.dot(X.T, hidden_delta) * learning_rate
        B_input += np.sum(hidden_delta, axis=0) * learning_rate
        if (epoch + 1) % 100 == 0:
            loss = np.mean(np.square(y - output))
            print(f'Epoch {epoch+1}, Loss: {loss}')
    return W_input, B_input, W_output, B_output
```

- Initializes a feedforward neural network with:
    - **Input layer**: Matches the feature dimensions.
    - **Hidden layer**: 8 neurons.
    - **Output layer**: Single output neuron.
- Trains the network using backpropagation for `epochs` iterations.

6.4 Predicting Relevance

```
def predict_relevance(X, W_input, B_input, W_output, B_output):
    hidden_layer_input = np.dot(X, W_input) + B_input
    hidden_layer_output = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, W_output) + B_output
    output = sigmoid(output_layer_input)
    return output
```

- Uses the trained network to predict relevance scores.

7. Integration

```
query = "Find me articles about the benefits of travelling for health."
folder_path = r'D:\IR\Information-Retrieval-Fall-2024\Assignment 6'
words = preprocess_text(query)
keywords = identify_keywords(words)
expanded_query = expand_query(keywords)
relevant_docs = retrieve_documents(expanded_query, folder_path)

for doc in relevant_docs:
    print(f"Document: {doc[0]}, Snippet: {doc[1]}")

if relevant_docs:
    X = calculate_relevance_score(relevant_docs)
    y_train = np.random.randint(0, 2, size=(len(relevant_docs), 1))
    W_input, B_input, W_output, B_output = train_neural_network(X, y_train,
epochs=1000, learning_rate=0.01)
    predicted_relevance = predict_relevance(X, W_input, B_input, W_output,
B_output)
    for i, doc in enumerate(relevant_docs):
        print(f"Document: {doc[0]}, Predicted Relevance Score:
{predicted_relevance[i][0]:.4f}")
else:
    print("No relevant documents found.")
```

- Query Preprocessing

  :

  - Processes the user query to extract keywords.
  - Expands keywords with synonyms.
- Document Retrieval

  :

  - Locates and ranks documents containing the keywords.
- Neural Network

  :

  - Trains a neural network to compute relevance scores.
  - Outputs predicted relevance scores for the retrieved documents.

Improvements and Suggestions

1. Keyword Scoring:
   - Use TF-IDF to score keywords for better precision in retrieval.
2. Neural Network Features:
   - Replace random features with meaningful embeddings (e.g., word2vec, BERT).
3. Performance:
   - Optimize retrieval by indexing the documents using a library like Elasticsearch.
4. Evaluation:
   - Introduce precision, recall, and F1-score metrics to evaluate performance.