Interference Model

Overview

The interference model ranks a collection of documents based on their relevance to a user-provided query. It uses term frequencies and a probabilistic approach with add-one smoothing to compute relevance scores for each document.

Key Features

- **Query-based Ranking**: Users can input a query and receive a ranked list of documents.
- **Probabilistic Scoring**: Computes probabilities using term frequencies and smoothing to handle unseen terms.
- **Web Interface**: Interactive interface built using Flask.

Components

1. Loading Documents

Function: `load_documents(file_path)`

- Reads documents from a text file.
- Filters out empty lines and trims whitespace.
- Example:

```
documents = load_documents('documents.txt')
```

2. Tokenization

Function: `tokenize(text)`

- Splits a string into lowercase words.
- Example:

```
tokens = tokenize("The quick brown fox")
# Output: ['the', 'quick', 'brown', 'fox']
```

3. Term Frequency Computation

Function: `compute_term_frequencies(documents)`

- Calculates word frequencies for each document.
- Returns a list of dictionaries, one per document.
- Example:

```
tf = compute_term_frequencies(["hello world", "hello"])
# Output: [Counter({'hello': 1, 'world': 1}), Counter({'hello': 1})]
```

4. Probability Computation

Function: `compute_probabilities(query, document, tf, vocab_size)`

- Computes the probability of a query given a document:
  - Uses add-one smoothing: P(Query|Document)=∏termTFterm+1DocLength+VocabSizeP(\text{Query}|\text{Document}) = \prod*{\text{term}} \frac{\text{TF}}{\text{term}} + 1}{\text{DocLength} + \text{VocabSize}}
- Example:

```
prob = compute_probabilities("hello world", "hello world", tf, 5)
```

5. Ranking Documents

Function: `rank_documents(query, documents, tf)`

- Ranks documents based on relevance to the query.
- Example:

```
ranked = rank_documents("hello", ["hello world", "hi there"], tf)
```

Flask Web Application

Routes

1. `/`

   - Home Page
     - Displays sample queries.

2. `/rank`

   - Rank Documents
     - Accepts a query and returns ranked documents as JSON.

Running the Application

- Start the server:

```
python app.py
```

- Access the interface at `http://127.0.0.1:5000`.

Example Use Case

Input

Query: "quick fox" Documents:

1. "The quick brown fox jumps over the lazy dog"
2. "The lazy dog sleeps"
3. "Fast fox runs quickly"

Output

Ranked List:

1. "The quick brown fox jumps over the lazy dog"
2. "Fast fox runs quickly"
3. "The lazy dog sleeps"

Limitations

1. No semantic understanding; relies solely on term matching.
2. Add-one smoothing can overestimate probabilities for low-frequency terms.
3. Query performance depends on pre-computed term frequencies and vocabulary size.

Future Enhancements

- Implement stemming or lemmatization for better term matching.
- Use TF-IDF or neural models for more accurate ranking.
- Add support for larger datasets using a database.

Conclusion

The interference model demonstrates a simple and interpretable approach to ranking documents. With extensions, it can serve as a foundation for more sophisticated information retrieval systems.

Understanding the Belief Network and the Code

What is a Belief Network?

A Belief Network, or Bayesian Network, is a probabilistic graphical model that represents a set of variables and their conditional dependencies through a directed acyclic graph (DAG). It is used for decision-making and probabilistic reasoning by applying Bayes' theorem. The code above implements a simple belief network for calculating the relevance of a document based on a query and document features.

Explanation of the Code

BeliefNetwork Class

The `BeliefNetwork` class implements the Bayesian reasoning for document relevance.

1. **Initialization (`__init__`):**
   - `self.p_query`: Prior probability of the query being asked (`P(Query)`).
   - `self.p_relevance`: Prior probability of a document being relevant (`P(Relevance)`).
   - `self.p_query_given_relevance`: Conditional probability of the query given relevance (`P(Query | Relevance)`).
   - `self.p_relevance_given_query`: Conditional probability of relevance given the query (`P(Relevance | Query)`).
   - `self.p_doc_feature_given_relevance`: Probability of document features being relevant (`P(Feature | Relevance)`).
   - `self.p_doc_feature_given_non_relevance`: Probability of document features being non-relevant (`P(Feature | Non-Relevance)`).
2. **Bayesian Update (`bayesian_update`):**
   - Implements Bayes' theorem: P(Relevance|Query)=P(Query|Relevance)×P(Relevance)P(Query)P(Relevance | Query) = \frac{P(Query | Relevance) \times P(Relevance)}{P(Query)}
   - Returns the posterior probability of relevance given the query.
3. **Joint Probability (`joint_probability`):**
   - Calculates the joint probability of the query and relevance: P(Query,Relevance)=P(Query|Relevance)×P(Relevance)P(Query, Relevance) = P(Query | Relevance) \times P(Relevance)
4. **Marginal Probability (`marginal_probability`):**
   - Returns the prior probability of relevance, i.e., `P(Relevance)`.
5. **Relevance Given Feature (`relevance_given_feature`):**
   - Computes the conditional probability of relevance given a document feature (`doc_feature`).
   - If the feature is relevant, uses `P(Feature | Relevance)`.
   - If the feature is non-relevant, uses `P(Feature | Non-Relevance)`.
6. **Calculate Relevance (`calculate_relevance`):**
   - Calculates the final relevance score of a document by:
     - Using `bayesian_update` for posterior probability of relevance (`P(Relevance | Query)`).
     - Using `relevance_given_feature` for the likelihood of relevance given the feature.
     - Multiplying these probabilities to compute the final relevance score.

Flask Application

1. **Flask Initialization:**
   - `app = Flask(__name__)`: Sets up the Flask application.
2. **Loading Documents (`load_documents`):**
   - Reads a text file (`documents.txt`) containing documents, each line representing a document.
   - Associates each document with a unique ID and stores it in a list of dictionaries.
3. **Home Route (`/`):**

   Displays the index page with sample queries and documents.

   Uses `load_documents` to load and display the documents.
4. **Relevance Calculation (`/calculate`):**

   Accepts a user query via a POST request.

   Iterates through each document:
   - Checks if the query is present in the document to classify it as "relevant" or "non-relevant."
   - Calculates the relevance score using the `BeliefNetwork` class.

   Sorts the documents by relevance scores in descending order.

   Passes the sorted documents and relevance scores to the `results.html` template.
5. **Run Server (`if __name__ == '__main__':`):**

   Runs the Flask application in debug mode.

Functions and Their Purpose

Function	Purpose
`__init__`	Initializes prior and conditional probabilities for the belief network.
`bayesian_update`	Computes the posterior probability of relevance given a query.
`joint_probability`	Calculates the joint probability of the query and relevance.
`marginal_probability`	Returns the prior probability of relevance.
`relevance_given_feature`	Computes the likelihood of relevance based on document features.
`calculate_relevance`	Computes the final relevance score combining query and feature probabilities.
`load_documents`	Reads and processes documents from a text file.
`index`	Serves the home page with queries and documents.

Function	Purpose
`calculate`	Computes relevance scores for documents based on a user query.

Belief Network and Document Relevance System

Overview

This project implements a Belief Network to compute the relevance of documents based on user queries and document features. It uses Bayesian reasoning to calculate probabilities and integrates with a Flask web application to display results.

Belief Network

Probabilities

- **Prior Probabilities:**

  `P(Query)`: Probability of a query being asked.

  `P(Relevance)`: Probability of a document being relevant.

- **Conditional Probabilities:**

  `P(Query | Relevance)`: Probability of a query being generated given relevance.

  `P(Relevance | Query)`: Probability of relevance given a query.

  `P(Feature | Relevance)`: Probability of document features given relevance.

  `P(Feature | Non-Relevance)`: Probability of document features given non-relevance.

Conclusion

The **Belief Network and Document Relevance System** effectively demonstrates the application of Bayesian reasoning to compute document relevance based on user queries and document features. By integrating probability theory with real-world scenarios like information retrieval, this project showcases how belief networks can aid in decision-making by quantifying uncertainty.

The implementation is structured into two main components:

1. **Belief Network Model**: Encapsulates the logic for probabilistic computations, ensuring modularity and reusability.
2. **Flask Web Application**: Provides a user-friendly interface to interact with the model, process queries, and display ranked documents.

This approach highlights the synergy between theoretical models (Bayesian networks) and practical software engineering (Flask). The system can be extended further by incorporating:

- Advanced NLP techniques to refine query-document matching.

- Dynamic updates to probabilities based on user feedback.
- Machine learning models to complement Bayesian reasoning.