Inference Model & Belief Network

Information Retrieval

Submitted to
Dr. Syed Khaldoon Khurshid

Submitted By

Shahzaib Irfan 2021-CS-07

University of Engineering and Technology
Lahore, Pakistan

Contents

1 Introduction

1.1 Project Overview

This project implements advanced Information Retrieval Probabilistic Models using a sophisticated Python-based framework. The system focuses on developing probabilistic approaches for document retrieval and relevance ranking, incorporating:

- Advanced relevance computation techniques
- Probabilistic interference and belief network models
- Dynamic document and query processing

1.2 Objectives

The main objectives of this project are:

- To develop robust probabilistic models for information retrieval
- To implement advanced document relevance computation methods
- To create an interactive web application for document analysis
- To demonstrate sophisticated probabilistic inference techniques

2 System Design

2.1 System Architecture

The system comprises multiple interconnected components:

1. **Document Processing Module**
    - Handles document upload and preprocessing
    - Extracts document features and metadata
2. **Probabilistic Inference Engine**
    - Implements two primary retrieval models:
        (a) Interference Model
        (b) Belief Network Model

3. **Relevance Computation Subsystem**
    - Computes advanced relevance scores
    - Manages term significance and prior probabilities
4. **Interactive Web Interface**
    - Built using Streamlit
    - Allows dynamic document upload and query processing

3 Implementation

3.1 Libraries and Tools

The project utilizes the following Python libraries:

- **Streamlit:** Web application framework
- **NumPy:** Numerical computing
- **Collections:** Advanced data structures
- **Math:** Mathematical computations

3.2 Core Components and Algorithms

3.2.1 Term Statistics Computation

The `_compute_term_statistics()` method calculates advanced term characteristics:

```
def _compute_term_statistics(self):
    term_doc_frequencies = defaultdict(int)
    total_docs = len(self.documents)

    # Compute term frequencies across documents
    for _, content in self.documents:
        terms = set(content.lower().split())
        for term in terms:
            term_doc_frequencies[term] += 1

    # Compute term significance using inverse document
       frequency
    for term, freq in term_doc_frequencies.items():
        self.prior_probabilities['term_significance'][term] =
            math.log(total_docs / (freq + 1))
```

Key Features:

- Computes term document frequencies
- Calculates term significance using logarithmic inverse document frequency
- Stores term significance in prior probabilities

3.2.2 Relevance Judgment Creation

The `create_relevance_judgments()` method generates sophisticated relevance scores:

```
def create_relevance_judgments(self, queries):
    self._compute_term_statistics()

    for query in queries:
        query_relevance = {}
        query_terms = set(query.lower().split())

        for doc_idx, (title, content) in enumerate(self.
            documents):
            doc_terms = set(content.lower().split())

            # Advanced term overlap computation
            overlap_score = sum(
                self.prior_probabilities['term_significance'
                    ][term]
                for term in query_terms.intersection(
                    doc_terms)
            )

            # Normalized relevance scoring
            relevance_score = min(1, overlap_score / len(
                query_terms)) if overlap_score > 0 else 0
            query_relevance[doc_idx] = relevance_score

        self.relevance_judgments[query] = query_relevance
```

3.2.3 Interference Model

The `interference_model()` computes document relevance:

```
def interference_model(self, query):
    relevance_scores = []
    for doc_idx, (title, content) in enumerate(self.documents
        ):
```

```
        base_relevance = self.relevance_judgments.get(query,
            {}).get(doc_idx, 0)

        query_terms = set(query.lower().split())
        doc_terms = set(content.lower().split())

        overlap_score = sum(
            self.prior_probabilities['term_significance'][
                term]
            for term in query_terms.intersection(doc_terms)
        )

        overlap = overlap_score / len(query_terms) if
            query_terms else 0
        relevance_score = base_relevance * (1 + overlap)
        relevance_scores.append((doc_idx, relevance_score))

    return sorted(relevance_scores, key=lambda x: x[1],
        reverse=True)
```

4 Results and Evaluation

4.1 Model Performance

- Successfully implemented two probabilistic retrieval models
- Demonstrated advanced relevance computation techniques
- Provided interactive web interface for document analysis

5 Conclusion

The project successfully implemented sophisticated probabilistic information retrieval models, showcasing advanced techniques in document relevance computation and ranking strategies.

Future Enhancements:

- Implement more advanced natural language processing techniques
- Integrate machine learning for dynamic model improvement
- Enhance query expansion capabilities