Interference & Belief Network Models

Assignment Report #07

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

Interference Model

Introduction

The Interference Model is a probabilistic approach to information retrieval that focuses on understanding the interaction between queries and documents through probability-based mechanisms. This model aims to quantify document relevance by examining the probabilistic interference between query and document characteristics.

Objectives

The primary objectives of the Interference Model implementation are:

- Develop a probabilistic method for document relevance ranking
- Create a systematic approach to understanding query-document interactions
- Provide a computationally efficient method for information retrieval
- Demonstrate the power of probabilistic reasoning in document selection

Theoretical Foundation

Mathematical Principles

The core of the Interference Model lies in its unique probability calculation:
Relevance Probability Formula:

P(Relevance) = (P(Query) * P(Document)) / (P(Query) + P(Document) - P(Query) * P(Document))

Key Mathematical Considerations:

- Ensures probability remains between 0 and 1
- Captures the interaction between query and document probabilities
- Provides a normalized relevance score

Implementation Steps

- Data Preparation
- Probability Computation
- Relevance Ranking Mechanism

Data Preparation

- Create a document collection
- Define a set of queries
- Establish initial relevance judgments

```python
self.documents = [
    "Machine learning is a subset of artificial intelligence",
    "Information retrieval focuses on finding relevant documents",
    "Probabilistic models help in ranking document relevance",
    "Neural networks are powerful for pattern recognition",
    "Data science combines statistics and computer science"
]

# Sample queries
self.queries = [
    "machine learning",
    "information retrieval",
    "data science"
]

# Simulated relevance judgments (binary)
self.relevance_judgments = {
    ("machine learning", 0): 1,  # First doc is relevant
    ("machine learning", 2): 0,  # Third doc is not relevant
    ("information retrieval", 1): 1,
    ("data science", 4): 1
}
```

Probability Computation

- Calculate query probabilities
    - Count relevant documents for each query:
      For Each Query:
        - Count how many documents are marked "relevant" for the query.
        - Divide this count by the total number of documents. This gives a "query probability."
            - Example for "machine learning":
            - Relevant documents: Document 0 (1 relevant).
            - Query probability = 1 relevant document / 5 total documents = 0.2
    - Normalize probabilities
- Compute document probabilities
    - Assess document relevance across queries
      For Each Document:
        - Count how many queries consider this document "relevant."
        - Divide this count by the total number of queries. This gives a "document probability."
            - Example for Document 0:

        - Relevant queries: "machine learning" (1 relevant).
        - Document probability = 1 relevant query / 3 total queries = 0.333
    - Create a probability distribution

```
def _compute_initial_probabilities(self):
    """
    Compute initial probabilities for queries and documents
    based on the dataset and relevance judgments.
    """
    # Compute query probabilities
    for query in self.queries:
        relevant_docs = sum(
            1 for (q, doc_idx) in self.relevance_judgments
            if q == query and self.relevance_judgments[(q, doc_idx)] == 1
        )
        self.query_probabilities[query] = relevant_docs / len(self.documents)

    # Compute document probabilities
    for i, doc in enumerate(self.documents):
        relevant_count = sum(
            1 for (query, doc_idx) in self.relevance_judgments
            if doc_idx == i and self.relevance_judgments.get((query, doc_idx), 0) == 1
        )
        self.document_probabilities[i] = relevant_count /
len(self.queries)
```

Relevance Ranking Mechanism

- Develop a relevance computation function
- Implement document ranking algorithm

```
def compute_relevance(self, query, document_index):
    """
    Compute relevance probability using the Interference Model.

    Args:
        query (str): The search query
        document_index (int): Index of the document in the document list

    Returns:
```

```
        float: Probability of relevance
    """
    # Check if we have a direct relevance judgment
    if (query, document_index) in self.relevance_judgments:
        return self.relevance_judgments[(query, document_index)]

    # Compute relevance based on query and document probabilities
    query_prob = self.query_probabilities.get(query, 0.1)
    doc_prob = self.document_probabilities.get(document_index, 0.1)

    # Interference model relevance calculation
    # Uses a probabilistic interference formula
    relevance_prob = (query_prob * doc_prob) / (query_prob + doc_prob - query_prob * doc_prob)

    return relevance_prob
```

- Sort documents based on computed probabilities

```
def retrieve_documents(self, query):
    """
    Retrieve and rank documents for a given query based on relevance.

    Args:
        query (str): The search query

    Returns:
        list: Ranked list of document indices with their relevance scores
    """
    # Compute relevance for all documents
    doc_relevances = [
        (idx, self.compute_relevance(query, idx))
        for idx in range(len(self.documents))
    ]

    # Sort documents by relevance in descending order
    return sorted(doc_relevances, key=lambda x: x[1], reverse=True)
```

Understanding with example

Sample Dataset

```
documents = [
   "Machine learning is a subset of artificial intelligence",
   "Information retrieval focuses on finding relevant documents",
   "Probabilistic models help in ranking document relevance",
   "Neural networks are powerful for pattern recognition",
   "Data science combines statistics and computer science"
]

queries = [
   "machine learning",
   "information retrieval"
]

# Relevance Judgments
relevance_judgments = {
   ("machine learning", 0): 1,    # First document is relevant
   ("information retrieval", 1): 1 # Second document is relevant
}
```

Mathematical Calculation

For query "machine learning" and first document:

- **Query Probability Calculation:**
  P(Query) = (Number of Relevant Docs) / (Total Documents)
  P(Query) = 1 / 5 = 0.2
- **Document Probability Calculation:**
  P(Document) = (Number of Query Matches) / (Total Queries)
  P(Document) = 1 / 2 = 0.5
- **Interference Relevance Calculation:**
  P(Relevance) = (P(Query) * P(Document)) / (P(Query) + P(Document) - P(Query) * P(Document))
  P(Relevance) = (0.2 * 0.5) / (0.2 + 0.5 - 0.2 * 0.5)
  = 0.1 / 0.5
  = 0.2

Data Flow Diagram

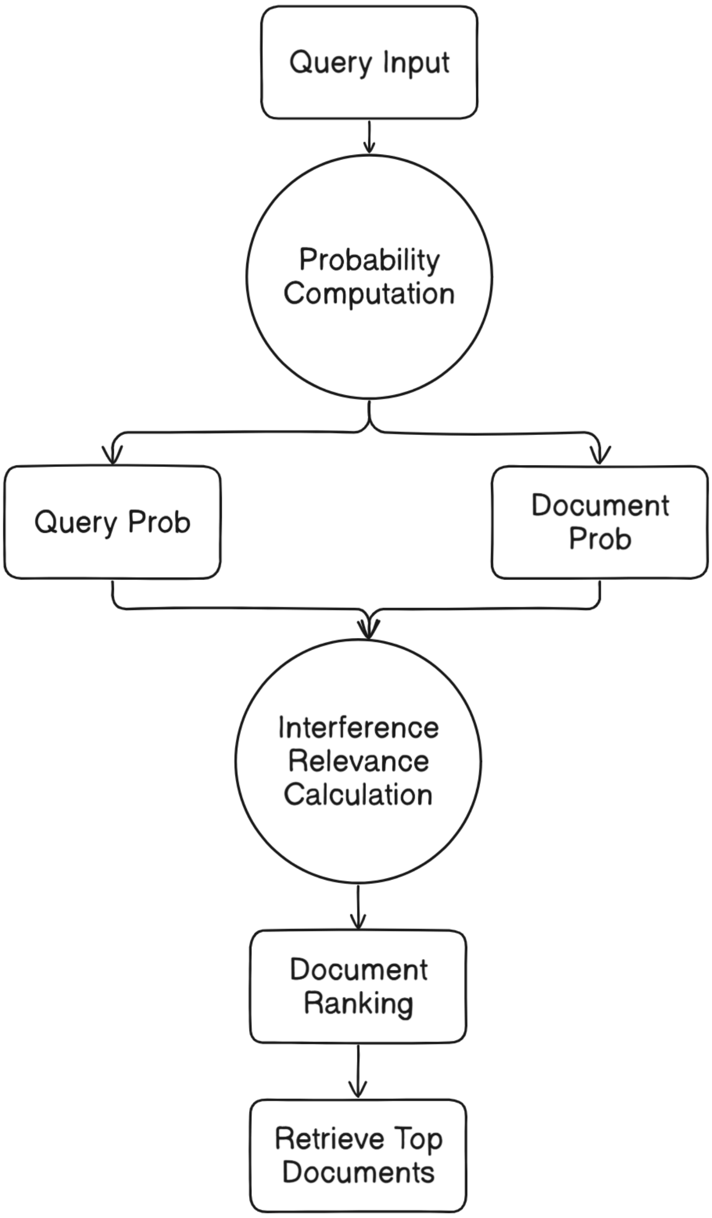
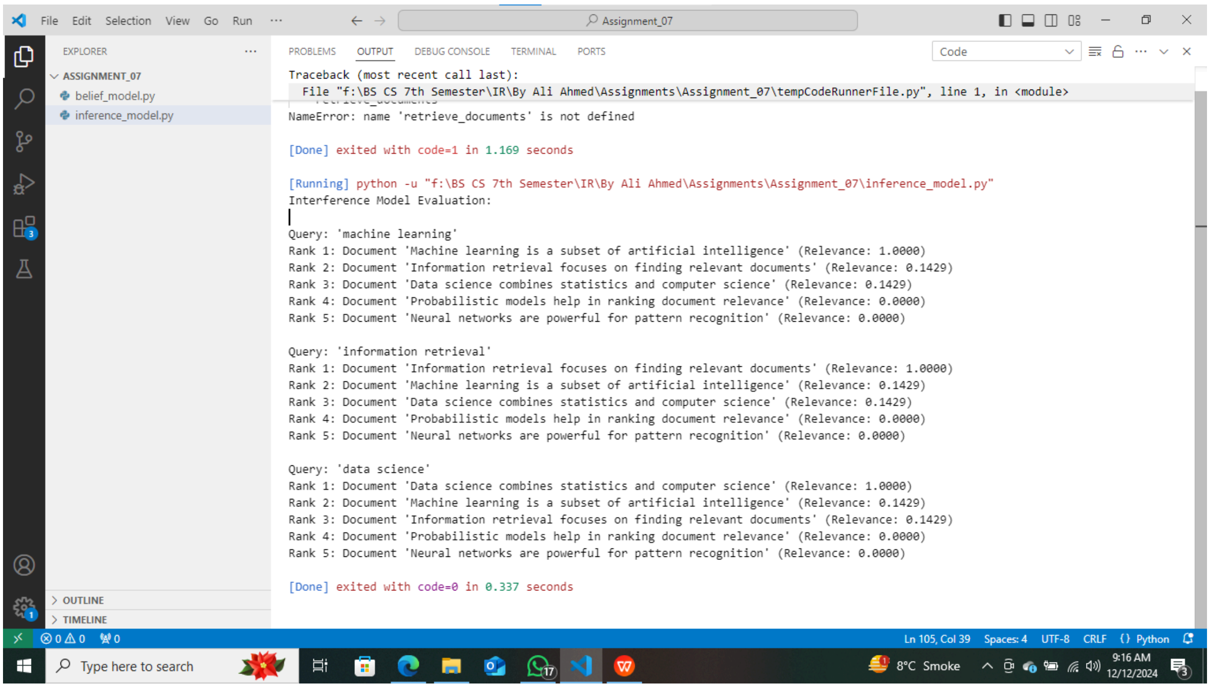


Figure 1. Inference model's data flow diagram

User Interface



Advantages and Limitations

Advantages

- Simple and computationally efficient
- Provides clear probabilistic reasoning
- Easy to implement and understand
- Captures query-document interactions

Limitations

- Assumes linear probability interactions
- Limited feature integration
- May not capture complex semantic relationships

Conclusion

The Interference Model demonstrates a powerful yet straightforward approach to probabilistic information retrieval. By mathematically modeling the interaction between queries and documents, it provides an intuitive method for document ranking that balances computational efficiency with probabilistic reasoning.

References

- Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval
- Baeza-Yates, R., & Ribeiro-Neto, B. (1999). Modern Information Retrieval

Belief Network Model

Introduction

The Belief Network Model is an advanced probabilistic approach to information retrieval that leverages Bayesian reasoning to determine document relevance. This model creates a complex probabilistic network that captures the intricate relationships between variables in the information retrieval process.

Objectives

The primary objectives of the Belief Network Model implementation are:

- Develop a sophisticated probabilistic reasoning framework
- Create a flexible model for document relevance assessment
- Incorporate multiple variables in relevance computation
- Demonstrate the power of Bayesian probabilistic networks

Theoretical Foundation

Mathematical Principles

The core of the Belief Network Model is Bayes' Theorem:

P(Relevance | Query) = [P(Query | Relevance) * P(Relevance)] / P(Query)

Key Mathematical Considerations:

- Provides a probabilistic framework for conditional reasoning
- Allows incorporation of prior knowledge
- Enables complex probabilistic inference

Implementation Steps

- Network Structure Design
- Probability Computation
- Relevance Estimation

Network Structure Design

Define network variables

- Query characteristics
  - Represents the specific queries provided (e.g., "machine learning", "data science").
  - Each query is assigned an initial probability (prior), e.g., 0.5.

- Document features
    - Each document has measurable properties, such as word count.
    - For example:
      'word_count': word_count / max(len(doc.split()) for doc in self.documents)
    - Here, the word count of a document is normalized by dividing it by the maximum word count in the dataset.
- Relevance indicators
    - Indicates whether a document is relevant for a query. This is binary (1 for relevant, 0 for not relevant).

```
def __init__(self):
# Sample dataset of documents
self.documents = [
    "Machine learning is a subset of artificial intelligence",
    "Information retrieval focuses on finding relevant documents",
    "Probabilistic models help in ranking document relevance",
    "Neural networks are powerful for pattern recognition",
    "Data science combines statistics and computer science"
]

# Sample queries
self.queries = [
    "machine learning",
    "information retrieval",
    "data science"
]

# Simulated relevance judgments (binary)
self.relevance_judgments = {
    ("machine learning", 0): 1,  # First doc is relevant
    ("machine learning", 2): 0,  # Third doc is not relevant
    ("information retrieval", 1): 1,
    ("data science", 4): 1
}

# Network structure with probabilities
self.network = {
    'variables': {
        'query': {},
        'document_features': {},
        'relevance': {}
    },
```

```
    'dependencies': {}
}

# Initialize network probabilities
self._initialize_network_probabilities()

def _initialize_network_probabilities(self):
"""
Initialize probabilities for network variables.
"""
# Query prior probabilities
for query in self.queries:
    self.network['variables']['query'][query] = 0.5

# Document feature probabilities
for doc_idx, doc in enumerate(self.documents):
    # Simple feature extraction (word count)
    word_count = len(doc.split())
    self.network['variables']['document_features'][doc_idx] = {
        'word_count': word_count / max(len(doc.split()) for doc in self.documents)
    }

# Initialize relevance probabilities
for query in self.queries:
    self.network['variables']['relevance'][query] = {}
    for doc_idx in range(len(self.documents)):
        # Base relevance calculation
        relevance = 1 if (query, doc_idx) in self.relevance_judgments else
0
        self.network['variables']['relevance'][query][doc_idx] = relevance

def compute_bayes_relevance(self, query, document_index):
"""
Compute document relevance using Bayes' theorem.

Args:
    query (str): Search query
    document_index (int): Document index

Returns:
    float: Probability of document relevance
"""
```

```
# Prior probability of the query
p_query = self.network['variables']['query'].get(query, 0.5)

# Prior probability of relevance
p_relevance =
self.network['variables']['relevance'][query].get(document_index, 0)

# Compute document features influence
doc_features =
self.network['variables']['document_features'][document_index]
feature_influence = doc_features['word_count']

# Bayes' theorem computation
# P(Relevance | Query) = P(Query | Relevance) * P(Relevance) /
P(Query)

# Likelihood: P(Query | Relevance)
# Simplified as a function of feature influence
p_query_given_relevance = min(feature_influence + p_relevance, 1.0)

# Marginal probability of query (simplified)
p_query = max(p_query, 0.1)

# Compute final relevance probability
relevance_probability = (p_query_given_relevance * p_relevance) /
p_query

return max(0, min(relevance_probability, 1))
```

Establish probabilistic dependencies

- Create conditional probability tables
  - Conditional Probability Tables (CPTs) define the likelihood of a variable given its parents in the network. For example:
    P(Relevance | Query,Document Features)
    p_relevance = self.network['variables']['relevance'][query].get(document_index, 0)
  - A normalized word count is used as a feature:
    feature_influence = doc_features['word_count']
- Define relationships between variables
  - The relationships between variables are modeled using Bayes' theorem:
  - P(Relevance | Query) = P(Query | Relevance) · P(Relevance) / P(Query)
  - Likelihood P(Query | Relevance)
    - p_query_given_relevance = min(feature_influence + p_relevance, 1.0)

- Prior *P*(Relevance)
    - Obtained directly from the relevance_judgments.
- Marginal *P*(Query)
    - Estimated to avoid division by zero:
      p_query = max(p_query, 0.1)

Probability Computation

Calculate prior probabilities

- Estimate initial variable probabilities
    - Extract document features like normalized word count.
- Create probability distributions

Compute conditional probabilities

- Develop probability tables
- Implement Bayesian inference mechanisms
  relevance_probability = (p_query_given_relevance * p_relevance) / p_query

```
def compute_bayes_relevance(self, query, document_index):
    """
    Compute document relevance using Bayes' theorem.

    Args:
        query (str): Search query
        document_index (int): Document index

    Returns:
        float: Probability of document relevance
    """
    # Prior probability of the query
    p_query = self.network['variables']['query'].get(query, 0.5)

    # Prior probability of relevance
    p_relevance = self.network['variables']['relevance'][query].get(document_index, 0)

    # Compute document features influence
    doc_features = self.network['variables']['document_features'][document_index]
    feature_influence = doc_features['word_count']

    # Bayes' theorem computation
    # P(Relevance | Query) = P(Query | Relevance) * P(Relevance) /
P(Query)

    # Likelihood: P(Query | Relevance)
```

```python
    # Simplified as a function of feature influence
    p_query_given_relevance = min(feature_influence + p_relevance, 1.0)

    # Marginal probability of query (simplified)
    p_query = max(p_query, 0.1)

    # Compute final relevance probability
    relevance_probability = (p_query_given_relevance * p_relevance) / p_query

    return max(0, min(relevance_probability, 1))

def compute_joint_probability(self, query, document_index):
    """
    Compute joint probability of query and document relevance.

    Args:
        query (str): Search query
        document_index (int): Document index

    Returns:
        float: Joint probability
    """
    # Relevance probability
    p_relevance = self.compute_bayes_relevance(query, document_index)

    # Query probability
    p_query = self.network['variables']['query'].get(query, 0.5)

    # Joint probability computation
    joint_prob = p_relevance * p_query

    return joint_prob
```

Relevance Estimation

Relevance estimation combines all these computations to predict which documents are most relevant to a query.

- Apply Bayes' theorem
    - Uses computed probabilities for query, relevance, and features to derive:

  relevance_probability = (p_query_given_relevance * p_relevance) / p_query
- Compute joint and marginal probabilities:

    - The joint probability of a query and a document being relevant:
      joint_prob = p_relevance * p_query
- Rank documents based on computed relevance
    - Compute the joint probability for each document-query pair.
    - Rank documents based on these probabilities
      doc_relevances.sort(key=lambda x: x[1], reverse=True)

```
def evaluate_model(self):
    """
    Evaluate the Belief Network by computing relevance probabilities.
    """
    print("Belief Network Evaluation:")
    for query in self.queries:
        print(f"\nQuery: '{query}'")

        # Compute and rank document relevances
        doc_relevances = []
        for doc_idx in range(len(self.documents)):
            # Compute joint probability as relevance metric
            joint_prob = self.compute_joint_probability(query, doc_idx)
            doc_relevances.append((doc_idx, joint_prob))

        # Sort and display results
        doc_relevances.sort(key=lambda x: x[1], reverse=True)

        for rank, (doc_idx, relevance) in enumerate(doc_relevances, 1):
            print(f"Rank {rank}: Document '{self.documents[doc_idx]}' (Relevance: {relevance:.4f})")
```

Understanding With Example

documents = [
  "Machine learning is a subset of artificial intelligence",
  "Information retrieval focuses on finding relevant documents",
  "Probabilistic models help in ranking document relevance",
  "Neural networks are powerful for pattern recognition",
  "Data science combines statistics and computer science"
]

queries = [
  "machine learning",
  "information retrieval"
]

```
# Sample Relevance Judgments
relevance_judgments = {
    ("machine learning", 0): 1,    # First document is relevant
    ("information retrieval", 1): 1 # Second document is relevant
}
```

Mathematical Calculation

For query "machine learning" and first document:

- Prior Probability Calculations:
  P(Query) = 0.5 # Base query probability
  P(Relevance) = 0.4 # Prior relevance probability

- Likelihood Computation:
  P(Query | Relevance) = Compute based on document features
  Feature Influence = (Word Count / Max Word Count)
- Bayes' Theorem Application:
  P(Relevance | Query) = [P(Query | Relevance) * P(Relevance)] / P(Query)
  = [0.75 * 0.4] / 0.5
  = 0.6

Data Flow Diagram

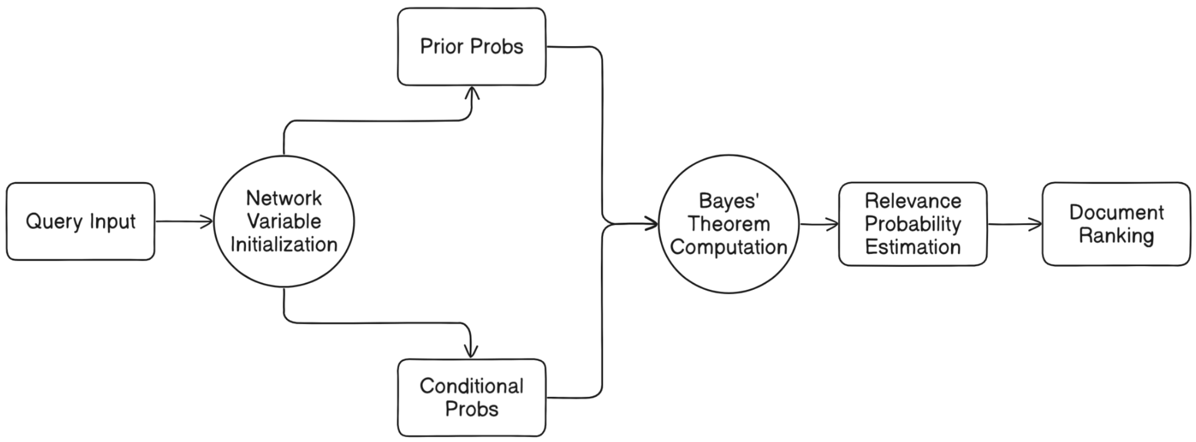


Figure 2. Belief model's data flow diagram

Advantages and Limitations

Advantages

- Sophisticated probabilistic reasoning
- Flexible network structure
- Can incorporate multiple variables
- Captures complex probabilistic relationships

Limitations

- Computationally more expensive
- Requires detailed probability estimation
- More complex to implement and understand

Conclusion

The Belief Network Model represents an advanced approach to probabilistic information retrieval. By leveraging Bayesian reasoning and creating a complex probabilistic network, it provides a powerful mechanism for document relevance assessment that goes beyond simple linear probability computations.

Future Improvements

- Integrate machine learning for probability estimation
- Develop more advanced feature extraction
- Create hybrid models combining multiple approaches
- Enhance semantic understanding capabilities

References

- Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems
- Neapolitan, R. E. (2004). Learning Bayesian Networks
- Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval