# CS 589 Assignment 2 Report

## ElasticSearch Ranking Evaluation

**Student Name:** Nakul Jadeja

---

## Table of Contents



---

# 1. Introduction

This assignment implements three information retrieval models using ElasticSearch:

- **BM25** (Okapi Best Match 25)
- **TF-IDF** (Term Frequency-Inverse Document Frequency)
- **Dirichlet Language Model**

The goal is to evaluate these models on StackOverflow question retrieval tasks across three programming languages: Python, Java, and JavaScript.

---

# 2. Dataset Description

## 2.1 Data Source

- **Source:** LinkSO dataset from StackOverflow
- **Languages:** Python, Java, JavaScript
- **Components:** Each question contains:
  - Title

- Question body
- Answer

## 2.2 Dataset Statistics

| Dataset | Total Documents | Test Queries |
|---|---|---|
| Python | 128,500 | 1,000 |
| Java | 159,263 | 1,000 |
| JavaScript | 174,015 | 1,000 |

## 2.3 Files Used

- `{lang}_qid2all.txt` - All StackOverflow questions
- `{lang}_cosidf.txt` - Relevance judgments between query-candidate pairs

---

# 3. Implementation Steps

## Step 1: Installing ElasticSearch and Kibana

**ElasticSearch Installation:**

1. Downloaded ElasticSearch 7.9.0 from official website
2. Extracted and ran `bin/elasticsearch.bat`
3. Verified installation at `http://localhost:9200`

**Kibana Installation:**

1. Downloaded Kibana 7.9.0
2. Extracted and ran `bin/kibana.bat`
3. Accessed Dev Tools at `http://localhost:5601`

## Step 2: Index Creation and Data Preparation

**2.1 Created 9 Indices**

Created indices for all combinations (3 languages × 3 similarity functions):

**BM25 Indices** (default similarity):

```
PUT /python_bm25
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_analyzer": {
          "type": "custom",
          "tokenizer": "whitespace",
          "filter": ["lowercase", "porter_stem"]
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "title": {"type": "text", "analyzer": "my_analyzer"},
      "body": {"type": "text", "analyzer": "my_analyzer"},
      "answer": {"type": "text", "analyzer": "my_analyzer"}
    }
  }
}
```

**TF-IDF Indices** (scripted similarity):

```
PUT /python_tfidf
{
  "settings": {
    "similarity": {
      "my_tfidf": {
        "type": "scripted",
        "script": {
          "source": "double tf = Math.sqrt(doc.freq); double idf = Math.l
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "title": {"type": "text", "similarity": "my_tfidf"},
      "body": {"type": "text", "similarity": "my_tfidf"},
      "answer": {"type": "text", "similarity": "my_tfidf"}
    }
```

```
    }
  }
```

**Dirichlet LM Indices**:

```
PUT /python_lmd
{
  "settings": {
    "similarity": {
      "my_lm": {
        "type": "LMDirichlet",
        "mu": 2000
      }
    }
  },
  "mappings": {
    "properties": {
      "title": {"type": "text", "similarity": "my_lm"},
      "body": {"type": "text", "similarity": "my_lm"},
      "answer": {"type": "text", "similarity": "my_lm"}
    }
  }
}
```

### 2.2 Data Conversion

Converted `{lang}_qid2all.txt` files to JSON format for bulk indexing:

**Python Script ( `convert_all_datasets.py` ):**

- Split data into batches of 10,000 documents each
- Created newline-delimited JSON format required by ElasticSearch
- Generated:
  - Python: 13 batch files
  - Java: 16 batch files
  - JavaScript: 18 batch files

### 2.3 Bulk Indexing

Used curl commands to bulk index all batches into all 9 indices:

```
curl.exe -s -H "Content-Type: application/json" \
  -XPOST localhost:9200/{index_name}/_doc/_bulk \
```

```
  --data-binary "@{batch_file}.json"
```

**Indexing Statistics:**

- Total batch files: 47
- Total curl commands executed: ~141 (47 batches × 3 indices)
- Total documents indexed: ~461,778 (across all 9 indices)

## Step 3: Generating Ratings (Algorithm 2)

Implemented Algorithm 2 to generate ground truth ratings for evaluation:

```
def generate_ratings(lang, index_name):
    # Read cosidf file containing relevance judgments
    # For each qid1:
    #   - Read 30 qid2s and their labels
    #   - Create ratings list with format:
    #     {"_index": index_name, "_id": qid2, "rating": label}
    # Save ratings as JSON file
```

**Output:** 9 rating files (one per index combination)

## Step 4: Ranking Evaluation (Algorithm 1)

Implemented Algorithm 1 using ElasticSearch's rank_eval API:

```
def evaluate_index(lang, similarity, index_name, ratings_file):
    # For each qid1:
    #   1. Get qid1_title from index
    #   2. Create ranking query with boost values:
    #      - title: 3.0
    #      - body: 0.5
    #      - answer: 0.5
    #   3. Execute es.rank_eval()
    #   4. Extract NDCG@10 score
    # Return average NDCG@10
```

**Query Structure (Figure 2):**

- Used boolean query with must_not (exclude query document)
- Used should clauses with different boost values
- Applied custom analyzer for text processing

**Evaluation Metric:**

- NDCG@10 (Normalized Discounted Cumulative Gain at rank 10)

---

# 4. Results

## 4.1 NDCG@10 Scores

| Dataset | BM25 | TF-IDF | Dirichlet LM |
|---|---|---|---|
| Python | 0.4957 | 0.4667 | 0.3540 |
| Java | 0.4754 | 0.4396 | 0.3452 |
| JavaScript | 0.4890 | 0.4746 | 0.3289 |

## 4.2 Performance Ranking

**Overall Ranking by Similarity Function:**

1. **BM25** - Best performer across all datasets
2. **TF-IDF** - Second best, consistent performance
3. **Dirichlet LM** - Lowest scores across all datasets

**Ranking by Dataset:**

1. **Python** - Highest scores overall
2. **JavaScript** - Mixed results
3. **Java** - Lowest scores overall

---

# 5. Analysis

## 5.1 Similarity Function Comparison

**BM25 Performance:**

- Consistently achieved highest NDCG@10 scores (0.47-0.50)
- Advantages:
  - Length normalization prevents bias toward long documents
  - Saturation function for term frequency
  - Proven effectiveness in information retrieval

**TF-IDF Performance:**

- Second-best performance (0.44-0.47)
- Simpler than BM25 but still effective
- Performs well when combined with proper normalization

**Dirichlet LM Performance:**

- Lowest scores (0.33-0.35)
- Possible reasons:
    - Default μ parameter (2000) may not be optimal
    - Language model requires more tuning
    - May need larger collection statistics

## 5.2 Dataset Comparison

**Python Dataset:**

- Best performance across all similarity functions
- Possible reasons:
    - Smaller dataset may have better quality questions
    - More focused technical vocabulary

**JavaScript Dataset:**

- Good BM25 and TF-IDF performance
    - Largest dataset with diverse content

**Java Dataset:**

- Lowest scores overall
- May have more complex or ambiguous queries

## 5.3 Component Weighting

The boost values used were:

- **Title: 3.0** - Highest weight (most discriminative)
- **Body: 0.5** - Lower weight (longer, noisier)
- **Answer: 0.5** - Lower weight (may contain irrelevant information)

This weighting strategy prioritizes title matching, which makes sense as titles are typically more concise and descriptive of the question's main intent.

---

# 6. Conclusion

## 6.1 Key Findings

1. **BM25 is the most effective** similarity function for StackOverflow question retrieval
2. **TF-IDF remains competitive** with simpler implementation
3. **Dirichlet LM requires tuning** to achieve competitive performance
4. **Title matching is crucial** for effective retrieval (as shown by high boost value)

## 6.2 Lessons Learned

1. **Index Configuration Matters:**
    - Proper analyzer configuration is essential
    - Similarity function must be correctly specified
2. **Bulk Indexing Challenges:**
    - Need to batch large datasets
    - Multiple indices require automation
3. **Evaluation Complexity:**
    - ElasticSearch's rank_eval API simplifies evaluation
    - Large-scale evaluation requires careful implementation

# Appendix

## A. Commands Used

**Index Creation:**

```
# Executed in Kibana Dev Tools
PUT /python_bm25
PUT /python_tfidf
PUT /python_lmd
# ... (repeated for java and javascript)
```

**Bulk Indexing:**

```
# Executed in terminal
python convert_all_datasets.py
python bulk_index_all.py
```

**Evaluation:**

```
python generate_ratings.py
python ranking_evaluation.py
```

## B. Files Submitted

1. `assignment2_complete.py` - Complete implementation
2. `assignment2_report.pdf` - This report
3. `ndcg_results.json` - Raw results
4. `ndcg_report.txt` - Formatted results table

---

**End of Report**