Keyword Search Engine (mini-IR)

Zbigniew Dendzik
Institute of Physics, University of Silesia

2025

Introduction

The goal of this exercise set is to introduce a simple but important class of applications of artificial intelligence: information retrieval systems (IR). The task of such a system is to take a textual query (e.g. one or several keywords) and return a list of documents that best match this query.

In practice, similar ideas underlie web search engines, searching in technical documentation, email or notes. Modern production systems use very complex language models (large neural networks) that can understand context and semantic relations between words. In this set, however, we focus on the simplest level: searching a collection of documents using keywords, counting word frequencies and implementing a simple document ranking.

Our "mini search engine" will consist of:

- a class representing a single text document,
- an inverted index: for each word a list of documents where it appears,
- simple one-word and two-word queries and a relevance-based ranking.

Such a simple system is a good starting point for understanding what more advanced IR systems and language models do later on: instead of raw keyword matching they use vector embeddings and semantic similarity.

Example I

Below you will find a skeleton definition of the `Document` class representing a single text document. Each document has an identifier, a title and a body and provides methods for simple text processing.

```
class Document
{
   private int id;
   private String title;
   private String body;

   public Document(int id, String title, String body)
   {
      this.id=id;
```

```
10          this.title=title;
11          this.body=body;
12      }
13
14      public int getId()
15      {
16          return id;
17      }
18
19      public String getTitle()
20      {
21          return title;
22      }
23
24      public String getBody()
25      {
26          return body;
27      }
28
29      //returns an array of "words" after simple normalisation
30      public String[] getWords()
31      {
32          String txt=body.toLowerCase();
33          txt=txt.replaceAll("[^a-z0-9 ]","");
34          String[] words=txt.split("\\s+");
35          return words;
36      }
37
38      //returns frequency of a given word in the document
39      public int wordFrequency(String word)
40      {
41          String[] words=getWords();
42          int count=0;
43          for(String s:words)
44          {
45              if(s.equals(word.toLowerCase()))
46                  count++;
47          }
48          return count;
49      }
50
51      public java.util.Set<String> wordSet()
52      {
53          java.util.Set<String> set=
54              new java.util.HashSet<String>();
55          for(String s:getWords())
56          {
57              if(s.length()>0)
58                  set.add(s);
59          }
60          return set;
```

```
61      }
62
63      public String toString()
64      {
65          return "Document["+id+"]: "+title;
66      }
67  }
```

Listing 1: Document.java

Ex. 1.1

Test the `Document` class. Write a program that:

- creates several `Document` objects with different contents,
- prints the id, title and number of words for each document,
- prints the frequency of a few selected words for one chosen document.

```
1   public class TestDocument
2   {
3       public static void main(String[] args)
4       {
5           Document d1=new Document(1,"Movies",
6               "This is a document about movies and series.");
7           Document d2=new Document(2,"Sports",
8               "This document is about football and volleyball.");
9
10          System.out.println(d1);
11          System.out.println("Word count: "+
12              d1.getWords().length);
13          System.out.println("freq 'document': "+
14              d1.wordFrequency("document"));
15
16          System.out.println(d2);
17          System.out.println("Word set: "+
18              d2.wordSet());
19      }
20  }
```

Listing 2: TestDocument.java – skeleton

Example II

The next step is to build a simple inverted index, i.e. a data structure storing information about which documents contain a given word.

```
1   import java.util.*;
2
3   class Index
```

```
{
    private List<Document> documents;
    private Map<String, List<Integer>> index;

    public Index()
    {
        documents=new ArrayList<Document>();
        index=new HashMap<String, List<Integer>>();
    }

    public void addDocument(Document d)
    {
        documents.add(d);
        for(String word : d.wordSet())
        {
            List<Integer> list=index.get(word);
            if(list==null)
            {
                list=new ArrayList<Integer>();
                index.put(word,list);
            }
            if(!list.contains(d.getId()))
                list.add(d.getId());
        }
    }

    public List<Document> searchWord(String word)
    {
        List<Document> result=new ArrayList<Document>();
        List<Integer> idList=index.get(word.toLowerCase());
        if(idList!=null)
        {
            for(Integer id : idList)
            {
                Document d=findDocumentById(id);
                if(d!=null)
                    result.add(d);
            }
        }
        return result;
    }

    private Document findDocumentById(int id)
    {
        for(Document d : documents)
        {
            if(d.getId()==id)
                return d;
        }
        return null;
    }
```

```
}
```

Listing 3: Index.java – skeleton

Ex. 1.2

Write a test program `TestIndex` that:

- creates an `Index` object,
- adds a dozen documents to it,
- asks the user for a keyword (from keyboard) and prints the titles of documents containing this word.

```
import java.util.*;

public class TestIndex
{
   public static void main(String[] args)
   {
      Index index=new Index();

      Document d1=new Document(1,"Movies",
         "This is a document about movies and series.");
      Document d2=new Document(2,"Sports",
         "This document is about football and volleyball.");
      Document d3=new Document(3,"Music",
         "A document about film music and concerts.");

      index.addDocument(d1);
      index.addDocument(d2);
      index.addDocument(d3);

      Scanner in=new Scanner(System.in);
      System.out.print("Keyword: ");
      String word=in.nextLine().toLowerCase();

      List<Document> results=index.searchWord(word);
      System.out.println("Found documents:");
      for(Document d : results)
      {
         System.out.println(" - "+d.getTitle());
      }
   }
}
```

Listing 4: TestIndex.java – skeleton

Example III

Now we extend the search engine with multi-word queries and simple relevance ranking.

```
import java.util.*;

class SearchEngine
{
   private Index index;
   private Set<String> stopWords;

   public SearchEngine(Index index)
   {
      this.index=index;
      stopWords=new HashSet<String>();
      stopWords.add("and");
      stopWords.add("the");
      stopWords.add("or");
      stopWords.add("a");
   }

   public List<Document> query(String word)
   {
      word=normaliseWord(word);
      if(stopWords.contains(word))
         return new ArrayList<Document>();
      return index.searchWord(word);
   }

   public List<Document> queryAND(String w1, String w2)
   {
      w1=normaliseWord(w1);
      w2=normaliseWord(w2);

      List<Document> l1=index.searchWord(w1);
      List<Document> l2=index.searchWord(w2);

      List<Document> result=new ArrayList<Document>();
      for(Document d1 : l1)
      {
         if(l2.contains(d1))
            result.add(d1);
      }
      return result;
   }

   public List<Document> queryOR(String w1, String w2)
   {
      w1=normaliseWord(w1);
      w2=normaliseWord(w2);

      List<Document> l1=index.searchWord(w1);
```

```
49        List<Document> l2=index.searchWord(w2);
50
51        List<Document> result=new ArrayList<Document>(l1);
52        for(Document d2 : l2)
53        {
54           if(!result.contains(d2))
55              result.add(d2);
56        }
57        return result;
58     }
59
60     public List<Document> sortByRelevance(
61        List<Document> list, String word)
62     {
63        word=normaliseWord(word);
64        Collections.sort(list,
65           new Comparator<Document>()
66           {
67              public int compare(Document d1,
68                                 Document d2)
69              {
70                 int f1=d1.wordFrequency(word);
71                 int f2=d2.wordFrequency(word);
72                 return Integer.compare(f2,f1);
73              }
74           });
75        return list;
76     }
77
78     private String normaliseWord(String s)
79     {
80        s=s.toLowerCase();
81        s=s.replaceAll("[^a-z0-9]","");
82        return s;
83     }
84  }
```

Listing 5: SearchEngine.java – skeleton

Ex. 1.3

Write a test program `TestSearchEngine` that:

- creates an `Index` and a `SearchEngine`,
- adds several documents to the index,
- lets the user choose the query type (one-word, AND, OR),
- prints the list of documents, and for a one-word query: documents sorted by relevance.

```java
import java.util.*;

public class TestSearchEngine
{
    public static void main(String[] args)
    {
        Index index=new Index();
        //TODO: add documents...

        SearchEngine se=new SearchEngine(index);

        Scanner in=new Scanner(System.in);
        System.out.println("1) one-word query");
        System.out.println("2) AND query");
        System.out.println("3) OR query");
        int choice=in.nextInt();
        in.nextLine();

        if(choice==1)
        {
            System.out.print("Word: ");
            String w=in.nextLine();
            List<Document> res=se.query(w);
            res=se.sortByRelevance(res,w);
            for(Document d : res)
                System.out.println(d.getTitle());
        }
        else if(choice==2)
        {
            System.out.print("Word 1: ");
            String w1=in.nextLine();
            System.out.print("Word 2: ");
            String w2=in.nextLine();
            List<Document> res=se.queryAND(w1,w2);
            for(Document d : res)
                System.out.println(d.getTitle());
        }
        else if(choice==3)
        {
            System.out.print("Word 1: ");
            String w1=in.nextLine();
            System.out.print("Word 2: ");
            String w2=in.nextLine();
            List<Document> res=se.queryOR(w1,w2);
            for(Document d : res)
                System.out.println(d.getTitle());
        }
    }
}
```

Listing 6: TestSearchEngine.java – skeleton

Example IV (extension)

Modern IR systems and large language models (LLMs) use much more advanced techniques: vector embeddings, semantic similarity, query expansion etc. The simple keyword search engine you build here is a miniature version of the first layer of such systems.

Ex. 1.4*

Extend the search engine with a simple TF-IDF weight:

- compute **TF** (Term Frequency) as the frequency of the word in the document,
- compute **IDF** (Inverse Document Frequency) as $\log(\frac{N}{df})$, where N is the number of documents and df is the number of documents containing the word,
- define the word weight in a document as $TF \cdot IDF$ and use it in document ranking.