Optimization in Question Answering System

GROUP NO.38

SUNNY TYAGI (071260)
ADITYA DUTT (071288)
PARAS JAIN (071319)
KESHAV KUMAR (071324)

Under the Supervision of: **Mr. Ravikant Verma**

विद्या तत्व ज्योतिसमः

Submitted in partial fulfillment of the Degree of Bachelor of Technology

Department of Computer Science & Information technology

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT, SOLAN
HIMACHAL PRADESH, INDIA
2011

TABLE OF CONTENTS

CHAPTER 1—SRS DOCUMENT

CHAPTER 2—DIAGRAMS

CHAPTER 3—QUESTION ANSWERING SYSTEM

CHAPTER 4—INVERTED INDEX

CHAPTER 5—SVM

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT
SOLAN
HIMACHAL PRADESH

CERTIFICATE

This is to certify that the thesis entitled "**OPTIMIZATION IN QUESTION ANSWERING SYSTEM**" is submitted in the partial fulfilment of the award of degree of **Bachelor of Technology(CSE &IT)** by JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT.

Signature of Supervisor ……………….28/May/11

Name of Supervisor **Mr. Ravikant verma**

Designation **Lecturer, CSE& IT**

Date ……………………..

ACKNOWLEDGEMENT

"Success of any endeavor is always due to contribution from different people"

Some time expressions are more reliable than words. Today we are short of words to express our deepest and numerous feelings to Almighty (The most beneficial and merciful) who gave us the mind, ability and courage to perform in a creative way.

We would like to express our sincere thanks to **Mr. Ravikant Verma** for his invaluable guidance, patience, support and encouragement extended to us throughout the duration of the project.

Finally, we wish to express our gratitude to all those who have in one-way or other helped us in the successful completion of interim part of our project.

SUMMARY

The main focus of QA is to gain the knowledge of the user's question and retrieve the sentences that are close to the answer. The QA task expects the system to understand the question and retrieve the corresponding passage in the text which contains the answer.

The architecture of our QA system is

1) Question Analysis

2) Passage retrieval and

3) Passage selection.

Question analysis involves the classification of the question into pre-defined question types, extraction of query words and determining the answer type. Passage retrieval searches for passages in the document collection which are likely to contain the answer. Passage selection ranks the list of candidate answers to determine the final answer.

OBJECTIVE OF THE PROJECT

To classify the questions into some predefined classes and find out the answers using the keyword in questions. To rank the answers among a large no of candidate answers. To obtain the most optimized answer satisfying the question module.

LIST OF FIUGRES

LIST OF ABBREVIATION/SYMBOLS

Abbreviation	Meaning
II	Inverted Index
SVM	Support Vector Machine
SRS	Software Requirements specification
DFD	Data Flow Diagram
FDD	Functional Decomposition Diagram

CHAPTER 1—SRS DOCUMENT

Introduction

Purpose:

To classify the questions into some predefined classes and find out the answers using the keyword in questions. To rank the answers among a large no of candidate answers. To obtain the most optimized answer satisfying the question module.

Example:

What a current information retrieval system or search engine can do is just "document retrieval", i.e., given some keywords it only returns the relevant documents that contain the keywords. However, what a user really wants is often a precise answer to a question. For instance, given the question "Who was the first President of India?"

- What a user really wants is the answer "Dr. Rajendra Prasad", but not to read through lots of documents that contain the words "first", "President" and "India" etc.

- Hence in order to correctly answer a question, usually one needs to understand what the question asks for. Question Classification, i.e., putting the questions into several semantic categories, can not only impose some constraints on the possible answers but also suggest different processing strategies.

- For instance, if the system understands that the question "Who was the first President of India?"asks for a person name, the search space of possible answers will be significantly reduced.

Document Conventions

No conventions as such.

Intended Audience and Reading Suggestions:

The intended audience for this SRS will be general public, though it will be specifically useful to users wanting to find just the answer and not to read the whole document containing the text thereby saving time.The developers will also find the SRS useful in designing. No reading suggestions provided since the S.R.S is very much self explanatory.

Product Scope:

This software system will be an optimization of question answering system produced for the general public to save their time by giving just the thing asked for and not the whole document. This system will be designed to minimize the reader's effort and time in searching a question or a paragraph for that matter and giving just the answer to the question.

References

IEEE . IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

Overall Description

Product Perspective:

The product is a self contained product and the SRS contains all the information required for the user to understand the software. The user will not be needing any other guide or tutorial for the same.

Product Functions:

- Question Analysis.
- Question classification.
- Text Retrieval.
- Candidate answer extraction.
- Ranking of candidate answer.
- Answer selection.

User Classes and Characteristics:

As our project involves general public we don't need a classification as such. But, since literate users are more probable of using the software we can classify the users on the basis of literacy-illiteracy, technical and non-technical.

Operating Environment:

The software can run on any system having command line interface (MS-DOS).

Design and Implementation Constraints:

The software would be generic, open for the general public and would be made having minimum time complexity and space complexity to avoid any kind of problem for the user.

User Documentation:

The software is not at all complex but is very easy and self-explanatory and thus does not require any documentation in terms of manuals or tutorials.

Assumptions and Dependencies:

Our assumption is that users question will definitely fall under one of the coarse and fine grained category.

Other Assumptions:-

Query will happen frequently

- e.g. Find all documents that contain term t

– Delete will be rare

- e.g. Delete document 52

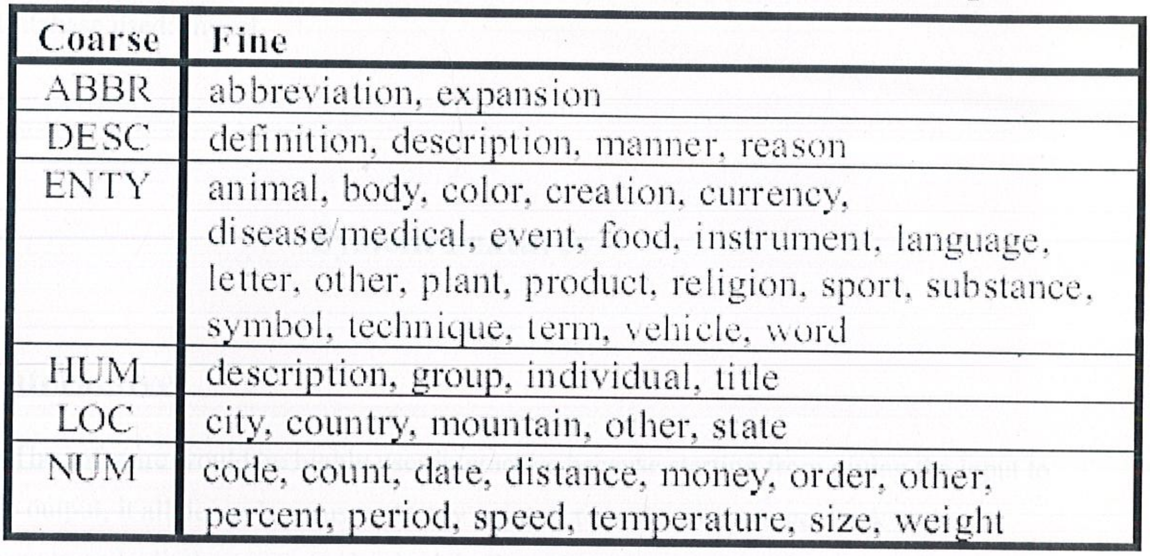
– Update will be rare

- e.g. Correct the spelling of term t in document 52

– add will not happen too often

- e.g. Add new documents

Table 1. The coarse and fine grained question categories.



Coarse	Fine
ABBR	abbreviation, expansion
DESC	definition, description, manner, reason
ENTY	animal, body, color, creation, currency, disease/medical, event, food, instrument, language, letter, other, plant, product, religion, sport, substance, symbol, technique, term, vehicle, word
HUM	description, group, individual, title
LOC	city, country, mountain, other, state
NUM	code, count, date, distance, money, order, other, percent, period, speed, temperature, size, weight

-

External Interface Requirements

User Interfaces:

The user will get a search space where he may right his query ,just like in google or blackle. And then the user will get the most optimized answer.

Hardware Interfaces:

No as such specific hardware requirements.

Software Interfaces:

- Software model used: Iterative model.
- Platform used for SVM: Statistical modeling software .e.g..

--SPSS(STATISTICAL PACKAGE FOR SOCIAL SCIENCE).

--DATA MINER by IBM

- Database used:-mysql

System Features

User Interactive:

The software would be highly user interactive because starting from giving the input to seeing the output, it all depends on user. At any point of time the user can get back to the previous menu and edit the query or simply exit. Everything is done keeping in mind the user's choice.

Popup menu:

The user will choose any one of following options and will then get a choice to either print the program, see the output or the related theory. like-

Enter choice

1. Print the program

2. Give related theory

3. Show output.

At each step special messages relating to the inappropriate functioning are generated as in case of a wrong input or any logical mistakes.

Proper validations are applied throughout the software.

Other Nonfunctional Requirements

Performance Requirements:

The code will be as light as possible having minimum time and space complexity. For this purpose we will try t use as less variables as possible and continually empty the buffer memory to retain the space complexity.

Safety Requirements:

Precautions should be taken in case of giving input don't try and give any redundant input, as it may result in bursty output.

Security Requirements:

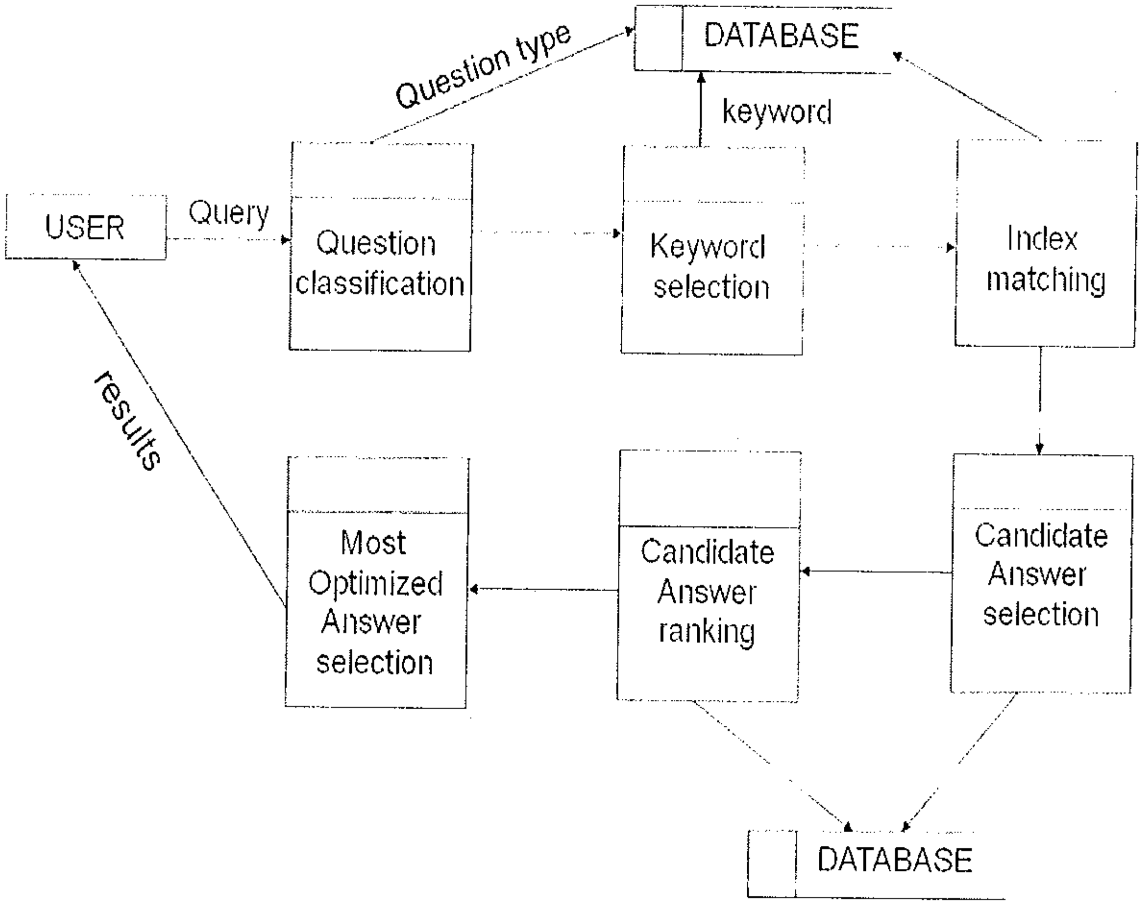
The codes will be provided but once the program code is approved by any higher authority the copyright of the code will be taken and the codes will be then made inaccessible to the users.

Software Quality Attributes:

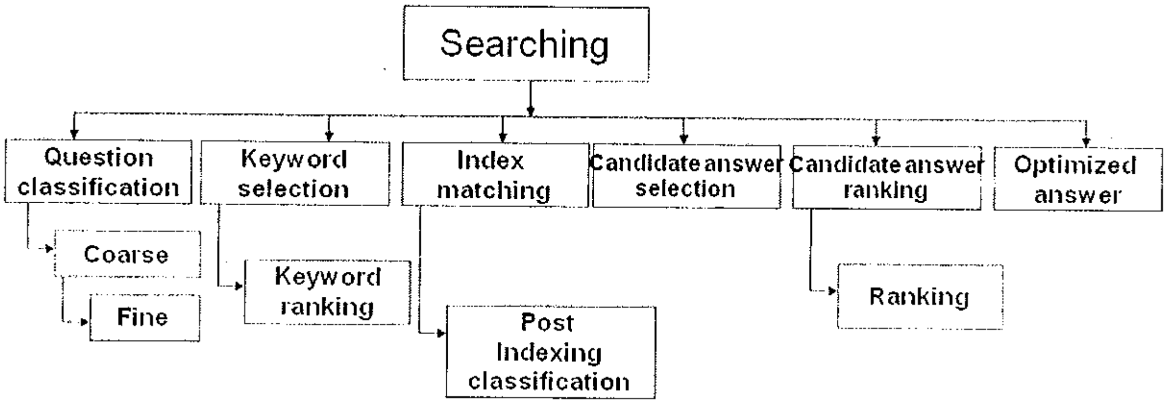
Adaptability: Will work on all environments like all versions of Windows having cli like (MS-DOS).
Availability: It would be made available to the recommended users.
Reliability: The software won't provide any wrong output.
Portability: the code will be light weighted and since it will run on any environment it is very portable.
Reusability: Since the program is provided as an output the user can reuse the code and manipulate it according to his needs or the user may contact the developer to do the needful.
Robustness: the software would handle any redundant input and will give the appropriate error message.
Testability: testability will be done for every kind of redundant input possible.

CHAPTER 2—SOFTWARE DIAGRAMS

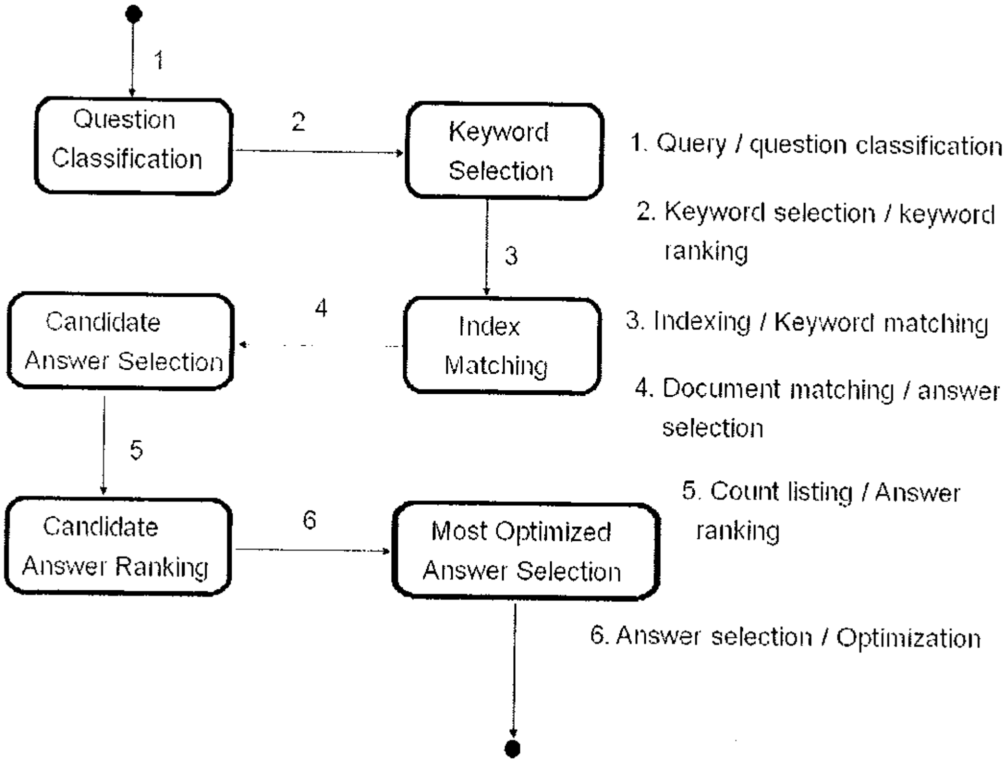
DATA FLOW DIAGRAM OF OPTIMIZATION IN QUESTION ANSWERING SYSTEM



FUNCTIONAL DECOMPOSITION DIAGRAM OF OPTIMIZATION IN QUESTION ANSWERING SYSTEM

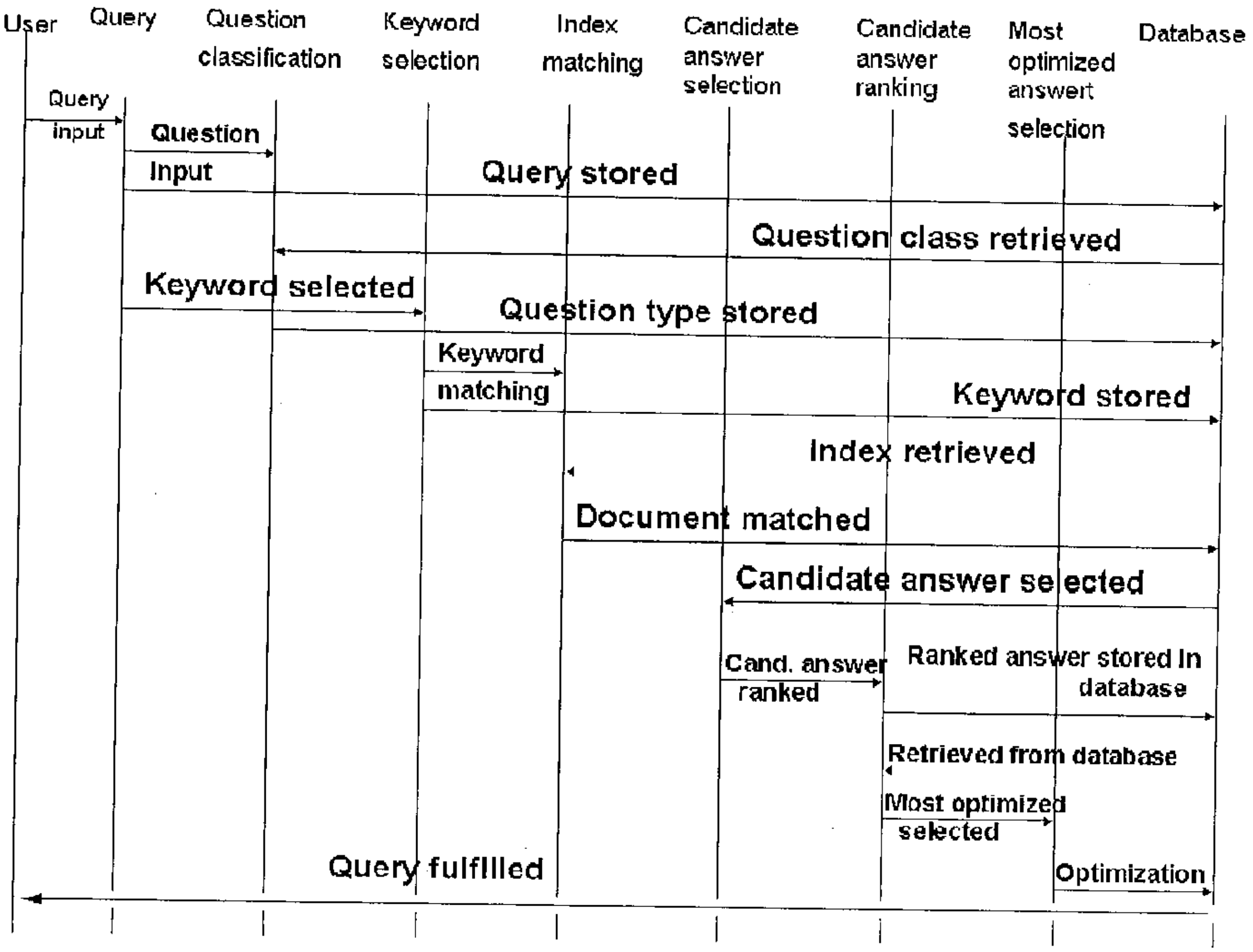


STATE CHART DIAGRAM OF OPTIMIZATION IN QUESTION ANSWERING SYSTEM



1. Query / question classification
2. Keyword selection / keyword ranking
3. Indexing / Keyword matching
4. Document matching / answer selection
5. Count listing / Answer ranking
6. Answer selection / Optimization

SEQUENTIAL DIAGRAM OF OPTIMIZATION IN QUESTION ANSWERING SYSTEM



CHAPTER 3—QUESTION ANSWERING SYSTEM

INTRODUCTION

- To classifies the questions into some predefined classes and find out the answers using the keyword in questions. The core of our question-answering mechanism is searching predefined pattern of textual expression that may be interpreted as answers to certain types of question. The presence of such patterns in analyzed answer-string candidates may provide evidence of the right answer.
- To rank the answers among a large no of candidate answers.
- To obtain the most optimized answer satisfying the question module.
- Question classification is very important for question answering.
- What a current information retrieval system or search engine can do is just "document retrieval", i.e., given some keywords it only returns the relevant documents that contain the keywords.
- However, what a user really wants is often a precise answer to a question. For instance, given the question "Who was the first President of India?"
- what a user really wants is the answer "Dr. Rajendra Prasad", but not to read through lots of documents that contain the words "first", "President" and "India" etc.
- Hence in order to correctly answer a question, usually one needs to understand what the question asks for. Question Classification, i.e., putting the questions into several semantic categories, can not only impose some constraints on the possible answers but also suggest different processing strategies.
- For instance, if the system understands that the question "Who was the first President of India?"asks for a person name, the search space of possible answers will be significantly reduced.

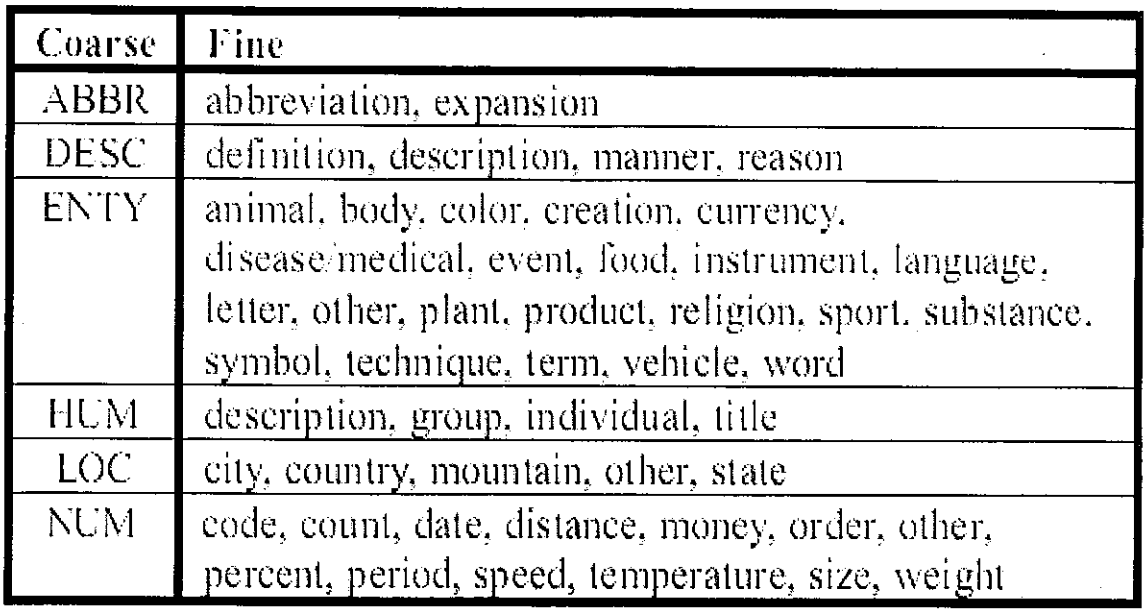
COMPONENTS OF Q/A SYSTEM

- Typically, a QA system has the following four components:
  1) Question Analysis,
  2) Document Retrieval,
  3) Passage Retrieval, and
  4) Answer Extraction.
- The question analysis component analyzes the question to determine its answer type, and to produce a list of keywords. Using these keywords document retrieval searches for a set of potentially relevant documents from the collection. From these documents, passage retrieval selects passages that are likely to contain the answer.
- Finally, answer extraction searches these passages for the final answer.
- Typically, question answering system uses the following components:
  1. Question analysis: analyzes a given question sentence and determines the question type and keywords.
  2. Text retrieval: finds the top N paragraphs that match the output of question analysis, such as keywords and question types.
  3. Answer candidate extraction: extracts answer candidates form the relevant documents retrieved by the text retrieval component.
  4. Answer selection: selects answer to the question from among the answer candidates based on the result of question analysis.

QUESTION CLASSIFICATION

- Question Classification means putting the questions into several semantic categories.
- We follow the two-layered question taxonomy, which contains 6 coarse grained categories and 50 fine grained categories, as shown in Table 1. Each coarse grained category contains a non-overlapping set of fine grained categories.
- Most question answering systems use a coarse grained category definition. Usually the number of question categories is less than 20. However, it is obvious that a fine grained category definition is more beneficial in locating and verifying the plausible answers.

Table 1. The coarse and fine grained question categories.



Coarse	Fine
ABBR	abbreviation, expansion
DESC	definition, description, manner, reason
ENTY	animal, body, color, creation, currency, disease/medical, event, food, instrument, language, letter, other, plant, product, religion, sport, substance, symbol, technique, term, vehicle, word
HUM	description, group, individual, title
LOC	city, country, mountain, other, state
NUM	code, count, date, distance, money, order, other, percent, period, speed, temperature, size, weight

To simplify the following experiments, we assume that one question resides in only one category. That is to say, an ambiguous question is labeled with its most probable category.

ALGORITHM USED

We use inverted index, tree map, hashed indexing here for indexing the tokens in the passage so that on matching the keywords of query with token we can conclude which passage to retrieve. And rank the passage as per the term frequency of the keyword.

OPTIMIZATION IN Q/A SYSTEM

We use inverted index and hash indexing over other sequential data structures to store the data and use brute force algorithm for pattern matching and candidate answer selection which reduces space and time complexity significantly. We use machine learning technique to optimize the answer .Here we use SVM(support vector machine)specifically to optimize the answer. We train the machine with training data and further test it with test data and check the optimality.

CHAPTER 4--INVERTED INDEX

INTRODUCTION

- Regardless of the retrieval strategy we need a data structure to efficiently store:

– For each term in the document collection

• The list of documents that contain the term

• For each occurrence of a term in a document

– The frequency the term appears in the document (tf).

– The position in the document for which the term appears.

  - Position may be expressed as section, paragraph, sentence, location within sentence.

- An inverted index contains two parts: an index of terms, (generally called simply the term index, lexicon, or term dictionary) which stores a distinct list of terms found in the collection and, for each term, a posting list, a list of documents that contain the term.
- Search engine indexing collects, parses, and stores data to facilitate fast and accurate information retrieval. Index design incorporates interdisciplinary concepts from linguistics, cognitive psychology, mathematics, informatics, physics and computer science. An alternate name for the process in the context of search engines designed to find web pages on the Internet is Web indexing.
- Popular engines focus on the full-text indexing of online, natural language documents.
- Computational and I/O costs are O (characters in collection).

Index Design Factors

Major factors in designing a search engine's architecture include

- **Merge factors:** How data enters the index, or how words or subject features are added to the index during text corpus traversal, and whether multiple indexers can work asynchronously. The indexer must first check whether it is updating old content or adding new content. Traversal typically correlates to the data collection policy. Search engine index merging is similar in concept to the SQL Merge command and other merge algorithms.
- **Storage techniques:** How to store the index data, that is, whether information should be data compressed or filtered.

- **Index size:** How much computer storage is required to support the index.
- **Lookup speed:** How quickly a word can be found in the inverted index. The speed of finding an entry in a data structure, compared with how quickly it can be updated or removed, is a central focus of computer science.
- **Maintenance:** How the index is maintained over time.
- **Fault tolerance:** How important it is for the service to be reliable. Issues include dealing with index corruption, determining whether bad data can be treated in isolation, dealing with bad hardware, partitioning, and schemes such as hash-based or composite partitioning, as well as replication.

Index Data Structures

- Where other data structures will take a minimum of O(logn) time to do any operation (best was B-trees (balanced trees), developed for efficient database access).
- Each internal node contains a key
  - Left subkey stores all keys smaller than the parent key.
  - Right subtree stores keys larger than the parent key.
- B-tree (balanced tree) of order m
  - Root has between m and 2m keys, as do all other internal nodes
  - if ki is the i-t key of a given internal node, then all keys in the (i-1)-th child are smaller than k, while all keys in the i-th child are bigger.
  - All leaves are at the same depth.
- Usually, a B-tree is used as an index, and all associated data are stored in the leaves .
- B-trees are mainly used as a primary key access method for large databases in secondary memory.
- To search a given key, we go down the tree choosing the appropriate branch at each step.
  - Number of disk accesses = height of the tree.

Inverted indices

- Many search engines incorporate an inverted index when evaluating a search query to quickly locate documents containing the words in a query and then rank these documents by relevance. Because the inverted index stores a list of the documents containing each word, the search engine can use direct access to find the documents associated with each word in the query in order to retrieve the matching documents quickly. The following is a simplified illustration of an inverted index:

INVERTED INDEX	
Word	**Documents**
the	document1, document3 ,document4, document5
cow	document2, document3, document4
says	document5
moo	document7

- This index can only determine whether a word exists within a particular document, since it stores no information regarding the frequency and position of the word; it is therefore considered to be a boolean index. Such an index determines which documents match a query but does not rank matched documents. In some designs the index includes additional information such as the frequency of each word in each document or the positions of a word in each document.

- Position information enables the search algorithm to identify word proximity to support searching for phrases; frequency can be used to help in ranking the relevance of documents to the query. Such topics are the central research focus of information retrieval.

- The inverted index is a sparse matrix, since not all words are present in each document. To reduce computer storage memory requirements, it is stored differently from a two dimensional array. The index is similar to the term document matrices employed by latent semantic analysis. The inverted index can be considered a form of a hash table. In some cases the index is a form of a binary tree, which requires additional storage but may reduce the lookup time. In larger indices the architecture is typically a distributed hash table.

Index Merging

The inverted index is filled via a merge or rebuild. A rebuild is similar to a merge but first deletes the contents of the inverted index. The architecture may be designed to support incremental indexing, where a merge identifies the document or documents to be added or updated and then parses each document into words. For technical accuracy, a merge conflates newly indexed documents, typically residing in virtual memory, with the index cache residing on one or more computer hard drives.

After parsing, the indexer adds the referenced document to the document list for the appropriate words. In a larger search engine, the process of finding each word in the inverted index (in order to report that it occurred within a document) may be too time consuming, and so this process is commonly split up into two parts, the development of a forward index and a process which sorts the contents of the forward index into the inverted index. The inverted index is so named because it is an inversion of the forward index.

The Forward Index

- The forward index stores a list of words for each document. The following is a simplified form of the forward index:

Forward Index

Documents	Words
Document1	the, cow, says, moo
Document2	the, cat, and, the, hat
Document3	the, dish, ran, away, with,

- The idea behind developing a forward index is that as documents are parsing, it is better to immediately store the words per document. The delineation enables Asynchronous system processing, which partially circumvents the inverted index update bottleneck. The forward index is sorted to transform it to an inverted index. The forward index is essentially a list of pairs consisting of a document and a word, collated by the document. Converting the forward index to an inverted index is only a matter of sorting the pairs by the words. In this regard, the inverted index is a word-sorted forward index.

Compression

Generating or maintaining a large-scale search engine index represents a significant storage and processing challenge. Many search engines utilize a form of compression to reduce the size of the indices on disk. Consider the following scenario for a full text, Internet search engine.

- An estimated 2,000,000,000 different web pages exist as of the year 2000
- Suppose there are 250 words on each webpage (based on the assumption they are similar to the pages of a novel.
- It takes 8 bits (or 1 byte) to store a single character. Some encodings use 2 bytes per character.
- The average number of characters in any given word on a page may be estimated at 5.
- The average personal computer comes with 100 to 250 gigabytes of usable space.

Given this scenario, an uncompressed index (assuming a non-conflated, simple, index) for 2 billion web pages would need to store 500 billion word entries. At 1 byte per character, or 5 bytes per word, this would require 2500 gigabytes of storage space alone, more than the average free disk space of 25 personal computers. This space requirement may be even larger for a fault-tolerant distributed storage architecture.

Depending on the compression technique chosen, the index can be reduced to a fraction of this size. The tradeoff is the time and processing power required to perform compression and decompression.

Notably, large scale search engine designs incorporate the cost of storage as well as the costs of electricity to power the storage. Thus compression is a measure of cost.

Document Parsing

Document parsing breaks apart the components (words) of a document or other form of media for insertion into the forward and inverted indices. The words found are called *tokens*, and so, in the context of search engine indexing and natural language processing, parsing is more commonly referred to as tokenization. It is also sometimes called word boundary disambiguation , tagging, text segmentation, content analysis, text analysis, text mining , concordance generation, speech segmentation, lexing, or lexical analysis. The terms 'indexing', 'parsing', and 'tokenization' are used interchangeably in corporate slang.

Tokenization presents many challenges in extracting the necessary information from documents for indexing to support quality searching. Tokenization for indexing involves multiple technologies, the implementation of which are commonly kept as corporate secrets.

Challenges in Natural Language Processing

Word Boundary Ambiguity: Native English speakers may at first consider tokenization to be a straightforward task, but this is not the case with designing a multilingual indexer. In digital form, the texts of other languages such as Chinese, Japanese or Arabic represent a greater challenge, as words are not clearly delineated by whitespace. The goal during tokenization is to identify words for which users will search. Language-specific logic is employed to properly identify the boundaries of words, which is often the rationale for designing a parser for each language supported (or for groups of languages with similar boundary markers and syntax).

Language Ambiguity: To assist with properly ranking matching documents, many search engines collect additional information about each word, such as its language or lexical category (part of speech). These techniques are language-dependent, as the syntax varies among languages. Documents do not always clearly identify the language of the document or represent it accurately. In tokenizing the document, some search engines attempt to automatically identify the language of the document.

Diverse File Formats: In order to correctly identify which bytes of a document represent characters, the file format must be correctly handled. Search engines which support multiple file formats must be able to correctly open and access the document and be able to tokenize the characters of the document.

Faulty Storage: The quality of the natural language data may not always be perfect. An unspecified number of documents, particular on the Internet, do not closely obey proper file protocol. Binary characters may be mistakenly encoded into various parts of a document. Without recognition of these characters and appropriate handling, the index quality or indexer performance could degrade.

Tokenization

Unlike literate humans, computers do not understand the structure of a natural language document and cannot automatically recognize words and sentences. To a computer, a document is only a sequence of bytes. Computers do not 'know' that a space character separates words in a document. Instead, humans must program the computer to identify what constitutes an individual or distinct word, referred to as a token. Such a program is commonly called a tokenizer or parser or lexer. Many search engines, as well as other natural language processing software, incorporate specialized programs for parsing, such as YACC or Lex.

During tokenization, the parser identifies sequences of characters which represent words and other elements, such as punctuation, which are represented by numeric codes, some of which are non-printing control characters. The parser can also identify entities such as email addresses, phone numbers, and URLs. When identifying each token, several characteristics may be stored, such as the token's case (upper, lower, mixed, proper), language or encoding, lexical category (part of speech, like 'noun' or 'verb'), position, sentence number, sentence position, length, and line number.

Language Recognition

- If the search engine supports multiple languages, a common initial step during tokenization is to identify each document's language; many of the subsequent steps are language dependent (such as stemming and part of speech tagging).
- Language recognition is the process by which a computer program attempts to automatically identify, or categorize, the language of a document. Other names for language recognition include language classification, language analysis, language identification, and language tagging.
- Automated language recognition is the subject of ongoing research in natural language processing. Finding which language the words belongs to may involve the use of a language recognition chart.

Assumptions

– query will happen frequently
- e.g.. Find all documents that contain term t

– delete will be rare
- e.g.. Delete document 52

– update will be rare
- e.g.. Correct the spelling of term t in document 52

– add will not happen too often
- e.g.. Add new documents

Building an Inverted Index

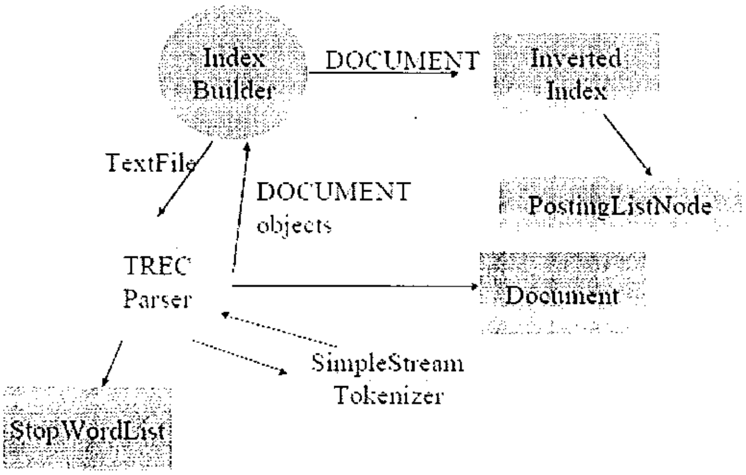
- Associates a posting list with each term.
- For each document d in the collection
  - For each term t in document d
    - Find term t in the term dictionary
    - If term t exists, add a node to its posting list
    - Otherwise,
      - Add term t to the term dictionary
      - Add a node to the posting list
- After all documents have been processed, write the inverted index to disk.
- Inverted because it lists for a term, all documents that contain the term.

Need of Inverted Index

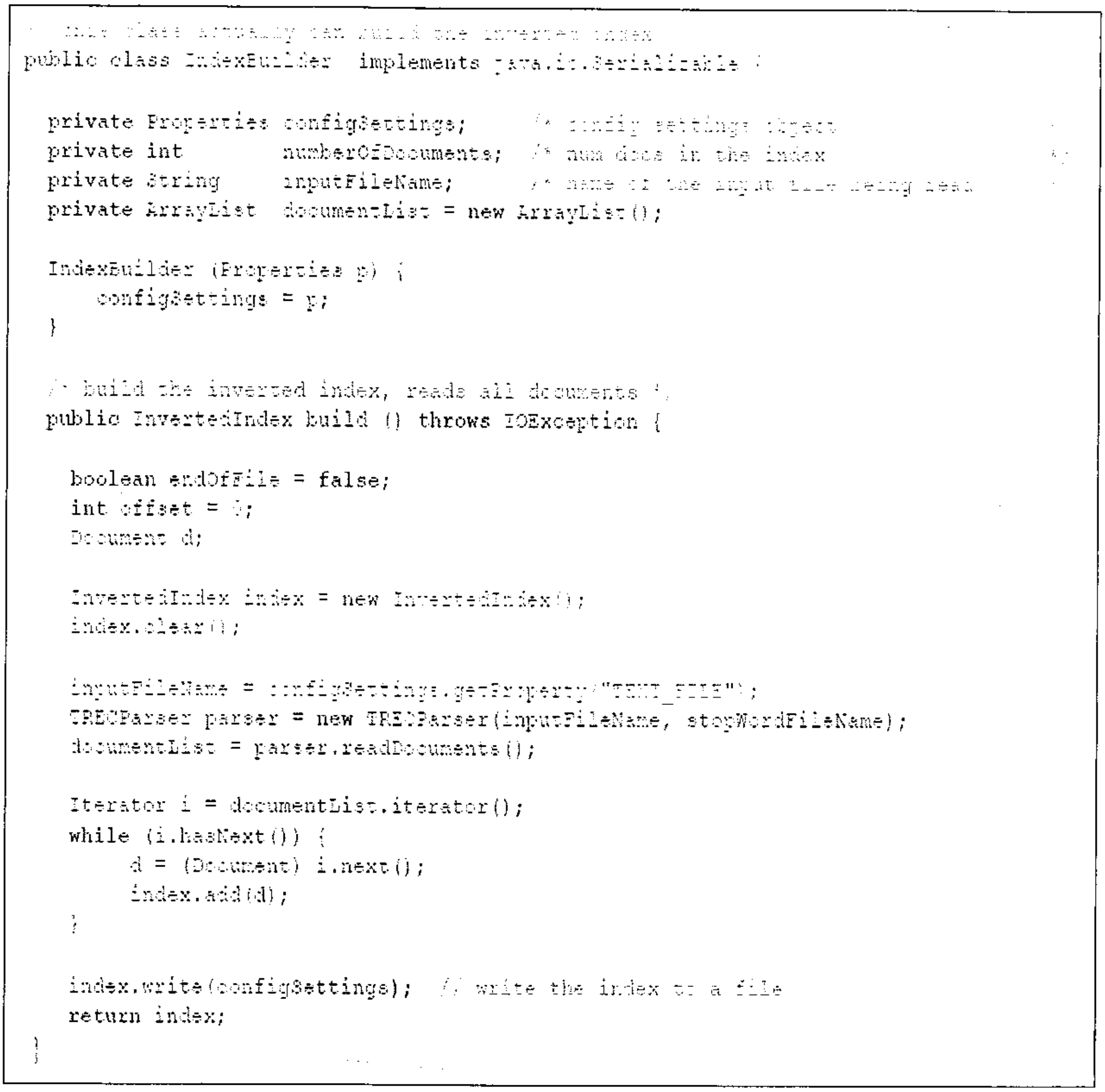
- The point of using an index is to increase the speed and efficiency of searches of the document collection. Without some sort of index, a user's query must sequentially scan the complete document collection, finding those documents containing the search terms.
- Consider the "Find" operation in Windows; a user search is initiated and a search starts through each file on the hard disk. When a directory is encountered, the search continues through each directory.
- With only a few thousand files on a typical laptop, a typical "find" operation takes a minute or longer.
- Currently, a web search covers at least one billion documents.
- Hence, a sequential scan is simply not feasible.
- Within the search engine domain, data are searched far more frequently than they are updated.
- An inverted index is able to do many accesses in O(1) time at a price of significantly longer time to do an update, in the worst case O(n).
- Where other data structures will take a minimum of O(logn) time to do any operation(best was B-trees (balanced trees), developed for efficient database access).
- For many systems, the inverted index can be compressed to around ten percent of the original document collection.

Index Construction

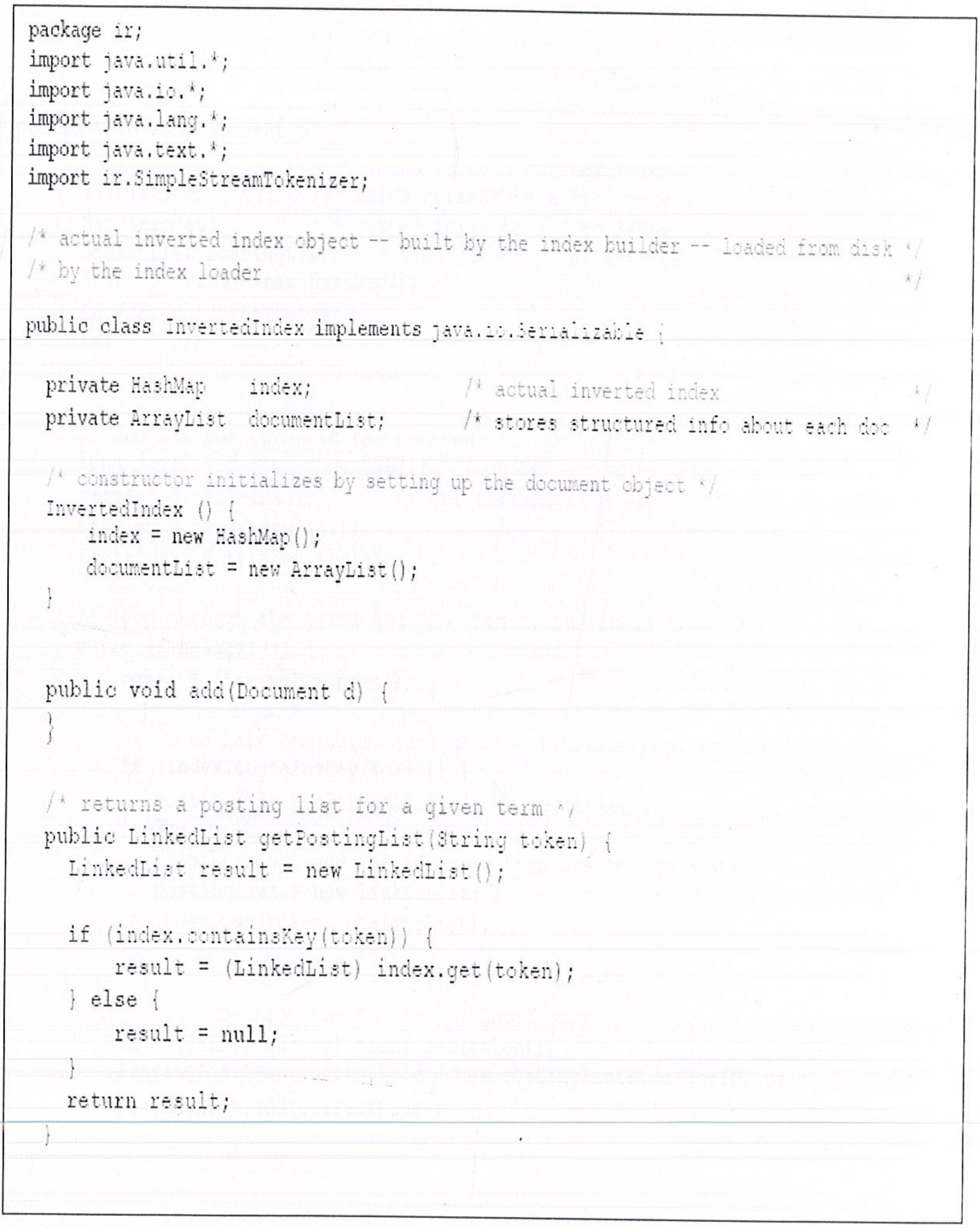
Figure 10: Indexing Architecture



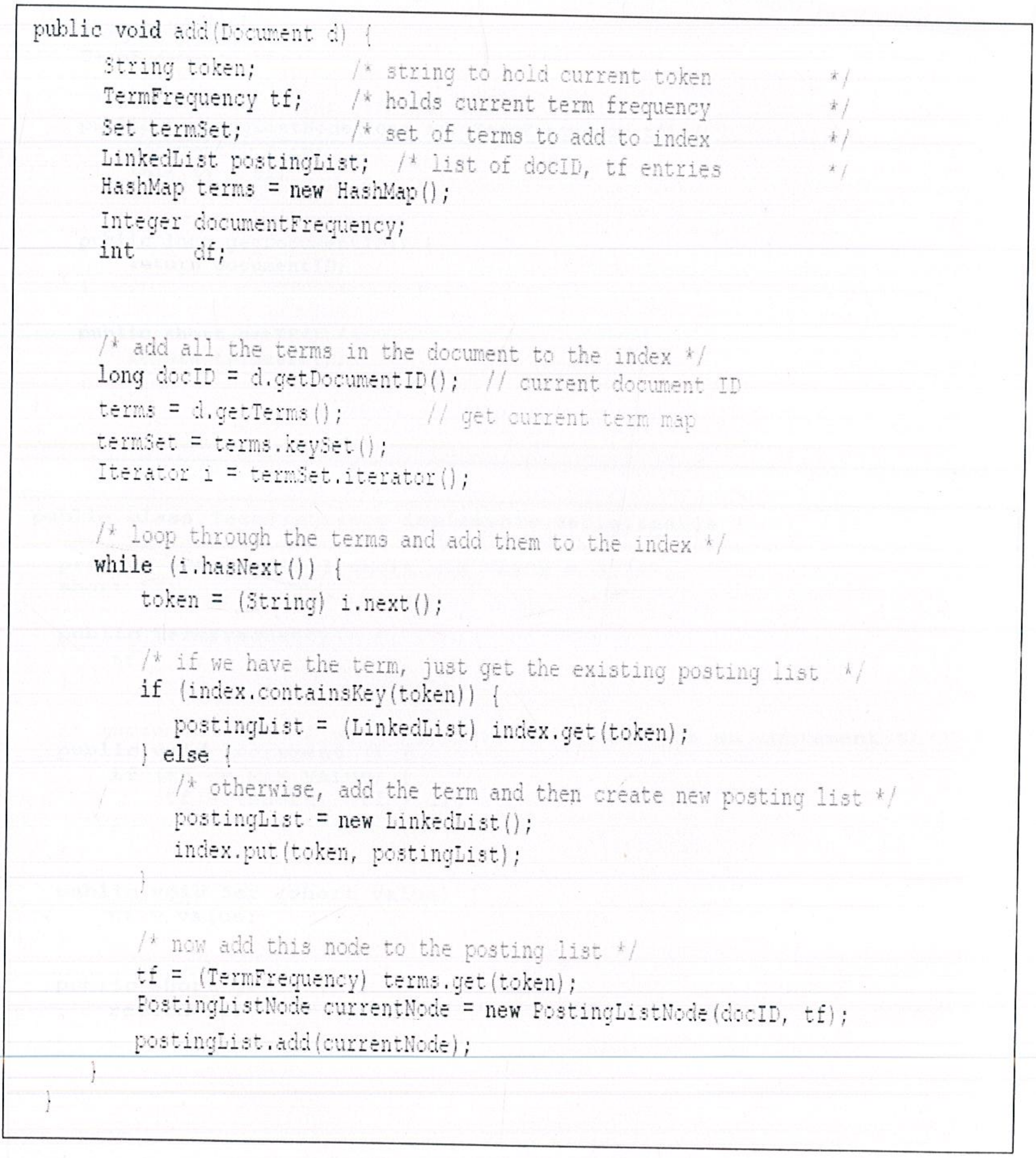
ALGORITHM



```
/* this class actually can build the inverted index
public class IndexBuilder  implements java.io.Serializable {

  private Properties configSettings;      /* config settings object
  private int        numberOfDocuments;  /* num docs in the index
  private String     inputFileName;      /* name of the input file being read
  private ArrayList  documentList = new ArrayList();

  IndexBuilder (Properties p) {
      configSettings = p;
  }

  /* build the inverted index, reads all documents */
  public InvertedIndex build () throws IOException {

    boolean endOfFile = false;
    int offset = 0;
    Document d;

    InvertedIndex index = new InvertedIndex();
    index.clear();

    inputFileName = configSettings.getProperty("TEXT_FILE");
    TRECParser parser = new TRECParser(inputFileName, stopWordFileName);
    documentList = parser.readDocuments();

    Iterator i = documentList.iterator();
    while (i.hasNext()) {
         d = (Document) i.next();
         index.add(d);
    }

    index.write(configSettings);  // write the index to a file
    return index;
 }
 }
```



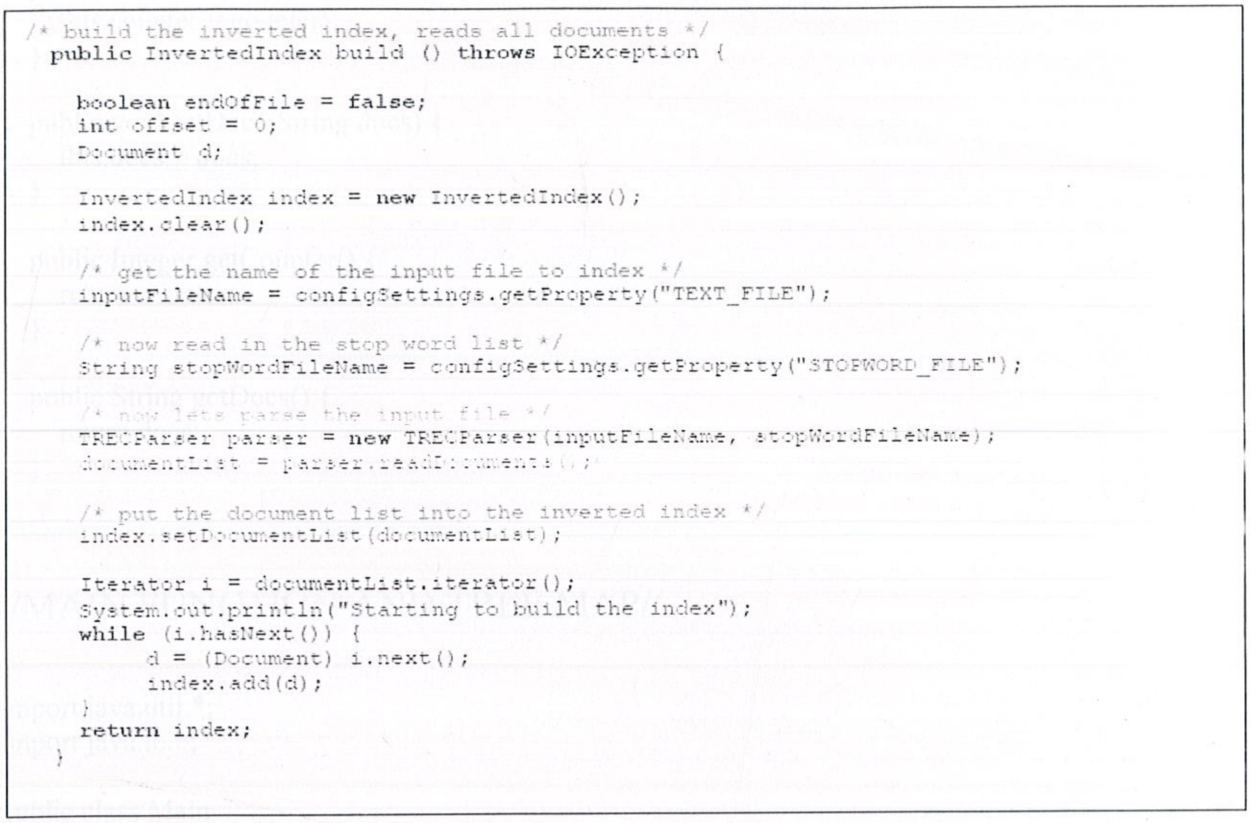
```java
package ir;
import java.util.*;
import java.io.*;
import java.lang.*;
import java.text.*;
import ir.SimpleStreamTokenizer;

/* actual inverted index object -- built by the index builder -- loaded from disk */
/* by the index loader                                                            */

public class InvertedIndex implements java.io.Serializable {

  private HashMap    index;              /* actual inverted index                  */
  private ArrayList  documentList;       /* stores structured info about each doc  */

  /* constructor initializes by setting up the document object */
  InvertedIndex () {
      index = new HashMap();
      documentList = new ArrayList();
  }

  public void add(Document d) {
  }

  /* returns a posting list for a given term */
  public LinkedList getPostingList(String token) {
    LinkedList result = new LinkedList();

    if (index.containsKey(token)) {
        result = (LinkedList) index.get(token);
    } else {
        result = null;
    }
    return result;
  }
```



```
public void add(Document d) {
      String token;          /* string to hold current token           */
      TermFrequency tf;      /* holds current term frequency           */
      Set termSet;           /* set of terms to add to index           */
      LinkedList postingList;  /* list of docID, tf entries            */
      HashMap terms = new HashMap();
      Integer documentFrequency;
      int     df;


      /* add all the terms in the document to the index */
      long docID = d.getDocumentID();  // current document ID
      terms = d.getTerms();          // get current term map
      termSet = terms.keySet();
      Iterator i = termSet.iterator();

      /* loop through the terms and add them to the index */
      while (i.hasNext()) {
          token = (String) i.next();

          /* if we have the term, just get the existing posting list  */
          if (index.containsKey(token)) {
             postingList = (LinkedList) index.get(token);
          } else {
             /* otherwise, add the term and then create new posting list */
             postingList = new LinkedList();
             index.put(token, postingList);
          }

          /* now add this node to the posting list */
          tf = (TermFrequency) terms.get(token);
          PostingListNode currentNode = new PostingListNode(docID, tf);
          postingList.add(currentNode);
      }
}
```

```
/* This will store and retreive an entry in a posting list              */
/* typically an entry in a posting list contains the document identifier */
/* and the term frequency                                                 */
public class PostingListNode implements Serializable {

    long documentID;
    TermFrequency tf;

    /* simple constructor with no args -- should not be called  */
    public PostingListNode(long id, TermFrequency tf) {
        documentID = id;
        this.tf = tf;
    }

    public long getDocumentID() {
        return documentID;
    }

    public short getTF() {
        return tf.getTF();
    }
}
```

```
public class TermFrequency implements Serializable {

  private static final short MAX_VALUE = 32767;
  short tf;

  public TermFrequency() {
      tf = 1;
  }

  /* increment the tf as long as we have room for an increment */
  public void Increment () {
      if (tf <= MAX_VALUE) {
          tf = (short) (tf + 1);
      }
  }

  public void Set (short value) {
      tf = value;
  }

  public short getTF () {
      return tf;
  }
}
```



```
/* build the inverted index, reads all documents */
 public InvertedIndex build () throws IOException {

   boolean endOfFile = false;
   int offset = 0;
   Document d;

   InvertedIndex index = new InvertedIndex();
   index.clear();

   /* get the name of the input file to index */
   inputFileName = configSettings.getProperty("TEXT_FILE");

   /* now read in the stop word list */
   String stopWordFileName = configSettings.getProperty("STOPWORD_FILE");

   /* now lets parse the input file */
   TRECParser parser = new TRECParser(inputFileName, stopWordFileName);
   documentList = parser.readDocuments();

   /* put the document list into the inverted index */
   index.setDocumentList(documentList);

   Iterator i = documentList.iterator();
   System.out.println("Starting to build the index");
   while (i.hasNext()) {
         d = (Document) i.next();
         index.add(d);
   }
   return index;
 }
```

SOURCE CODE

//   TO READ DOCUMENT   //

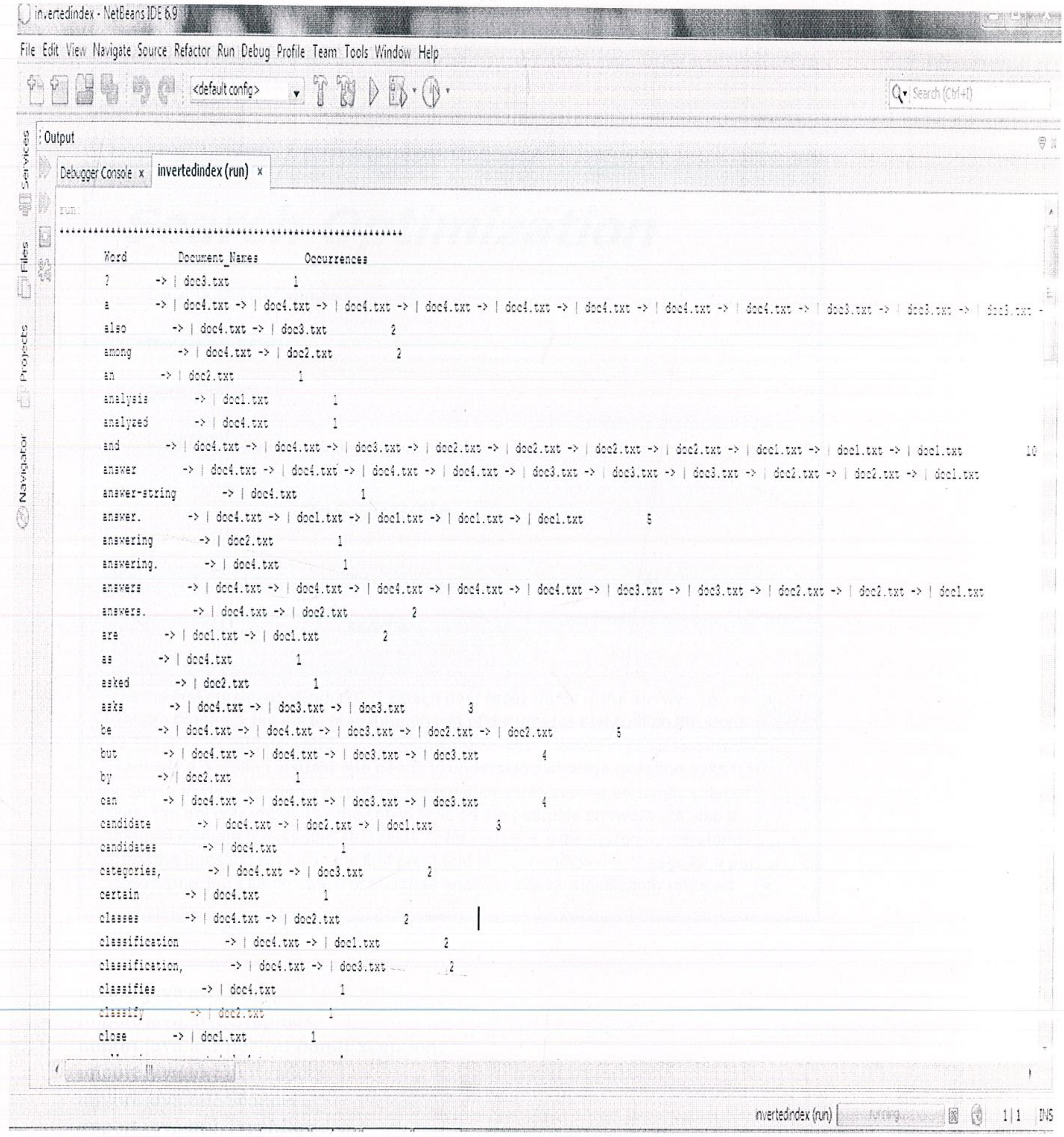
```
public class Data
{
Integer counter;
String docs="->";


    public Data(Integer counter, String docs) {
    this.counter = counter;
    this.docs =this.docs+" | "+docs;
  }

  public void setCounter(Integer counter) {
        this.counter = counter;
    }

    public void setDocs(String docs) {
        this.docs = docs;
    }

    public Integer getCounter() {
        return counter;
    }

    public String getDocs() {
        return docs;
    }
}
```

//MAIN FUNCTION AND TREE MAP//

```
import java.util.*;
import java.io.*;

public class Main
{
  public static void main(String[ ] args)
  {
    final TreeMap<String, Data> frequencyData = new TreeMap<String, Data>( );
java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new NewJFrame(frequencyData).setVisible(true);
        }
      });
    readWordFile(frequencyData);
    printAllCounts(frequencyData);

  }


  public static Integer getCount(String word, TreeMap<String, Data> frequencyData)
  {
    if (frequencyData.containsKey(word))
    {
        return frequencyData.get(word).getCounter();
    }
```

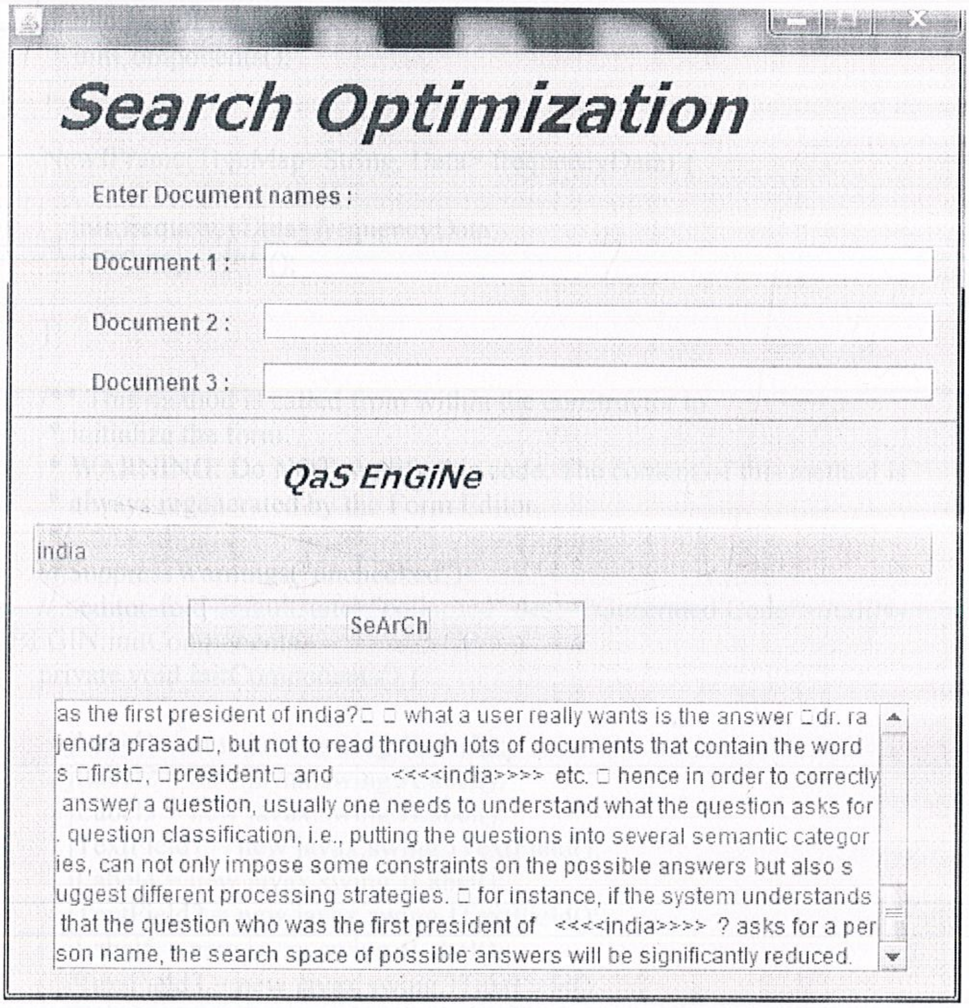
```
        else
        {
          return 0;
        }
      }

     public static void printAllCounts(TreeMap<String, Data> frequencyData)
     {
   System.out.println("******************************************************************")
   ;
       System.out.println("      Word      Document_Names       Occurrences");

       for(String word : frequencyData.keySet( ))
       {
         System.out.println("        " + word + "        " + frequencyData.get(word).getDocs()+"
   " +frequencyData.get(word).getCounter()+ "             ");
       }

   System.out.println("******************************************************************")
   ;
     }

     public static void readWordFile(TreeMap<String, Data> frequencyData)
     {
       Scanner wordFile;
       String word;
       Integer count;
        Data d;

  for (int x=0; x<Docs.length; x++)
  {

       try
       {
         wordFile = new Scanner(new FileReader(Docs[x]));
       }
       catch (FileNotFoundException e)
       {
   System.err.println(e);
   return;
       }
      while (wordFile.hasNext( ))
      {
   word = wordFile.next( );
 word = word.toLowerCase();
        count = getCount(word, frequencyData) + 1;

        if(x>=0)
        {
           try{


  if(frequencyData.get(word).getDocs()==null)
  {
     frequencyData.put(word, new Data(count,Docs[x]));
  }
  else
  {
          frequencyData.put(word, new Data(count,Docs[x]+"
"+frequencyData.get(word).getDocs()));
  }
          }
          catch(NullPointerException e)
          {
             frequencyData.put(word, new Data(count,Docs[x]));
          }
       }
     }

    }
  }


static String Docs [] = {"doc1.txt", "doc2.txt","doc3.txt","doc4.txt"};
}
```

//OUTPUT//



```
run:
**********************************************************
        Word          Document_Names        Occurrences
        ?         -> | doc3.txt            1
        a         -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc3.txt -> | doc3.txt -> | doc3.txt -
        also        -> | doc4.txt -> | doc3.txt           2
        among        -> | doc4.txt -> | doc2.txt           2
        an         -> | doc2.txt            1
        analysis        -> | doc1.txt            1
        analyzed        -> | doc4.txt            1
        and         -> | doc4.txt -> | doc4.txt -> | doc3.txt -> | doc2.txt -> | doc2.txt -> | doc2.txt -> | doc2.txt -> | doc1.txt -> | doc1.txt -> | doc1.txt            10
        answer        -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc3.txt -> | doc3.txt -> | doc3.txt -> | doc2.txt -> | doc2.txt -> | doc1.txt
        answer-string        -> | doc4.txt           1
        answer.        -> | doc4.txt -> | doc1.txt -> | doc1.txt -> | doc1.txt -> | doc1.txt           5
        answering        -> | doc2.txt           1
        answering.        -> | doc4.txt           1
        answers        -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc4.txt -> | doc3.txt -> | doc3.txt -> | doc2.txt -> | doc2.txt -> | doc1.txt
        answers.        -> | doc4.txt -> | doc2.txt           2
        are        -> | doc1.txt -> | doc1.txt           2
        as        -> | doc4.txt            1
        asked        -> | doc2.txt            1
        asks        -> | doc4.txt -> | doc3.txt -> | doc3.txt           3
        be        -> | doc4.txt -> | doc4.txt -> | doc3.txt -> | doc2.txt -> | doc2.txt           5
        but        -> | doc4.txt -> | doc4.txt -> | doc3.txt -> | doc3.txt           4
        by        -> | doc2.txt            1
        can        -> | doc4.txt -> | doc4.txt -> | doc3.txt -> | doc3.txt           4
        candidate        -> | doc4.txt -> | doc2.txt -> | doc1.txt           3
        candidates        -> | doc4.txt            1
        categories,        -> | doc4.txt -> | doc3.txt           2
        certain        -> | doc4.txt            1
        classes        -> | doc4.txt -> | doc2.txt           2
        classification        -> | doc4.txt -> | doc1.txt           2
        classification,        -> | doc4.txt -> | doc3.txt           2
        classifies        -> | doc4.txt            1
        classify        -> | doc2.txt            1
        close        -> | doc1.txt            1
```

//JFRAME//



india

```
as the first president of india?□ □ what a user really wants is the answer □dr. ra
jendra prasad□, but not to read through lots of documents that contain the word
s □first□, □president□ and      <<<<india>>>> etc. □ hence in order to correctly
 answer a question, usually one needs to understand what the question asks for
. question classification, i.e., putting the questions into several semantic categor
ies, can not only impose some constraints on the possible answers but also s
uggest different processing strategies. □ for instance, if the system understands
 that the question who was the first president of   <<<<india>>>> ? asks for a per
son name, the search space of possible answers will be significantly reduced.
```

```
import java.awt.Font;
import java.awt.Graphics;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;
import java.util.TreeMap;
```

```
public class NewJFrame extends javax.swing.JFrame {
    TreeMap<String, Data> frequencyData = new TreeMap<String, Data>( );

    /** Creates new form NewJFrame */
    public NewJFrame() {
        initComponents();
    }

    NewJFrame(TreeMap<String, Data> frequencyData) {

        this.frequencyData=frequencyData;
         initComponents();

    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        jTextField2 = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        jTextField3 = new javax.swing.JTextField();
        jLabel6 = new javax.swing.JLabel();
        jTextField4 = new javax.swing.JTextField();
        jSeparator1 = new javax.swing.JSeparator();
        jButton1 = new javax.swing.JButton();
        jPanel1 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();
```

```
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setResizable(false);

jLabel1.setFont(new java.awt.Font("Tahoma", 3, 36));
jLabel1.setForeground(new java.awt.Color(51, 0, 0));
jLabel1.setText("Search Optimization");

jLabel2.setText("Enter Document names :");

jLabel3.setText("Document 1 :");

jLabel4.setText("Document 2 :");

jTextField2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextField2ActionPerformed(evt);
    }
});

jLabel5.setText("Document 3 :");

jLabel6.setFont(new java.awt.Font("Tahoma", 3, 18));
jLabel6.setForeground(new java.awt.Color(51, 0, 0));
jLabel6.setText("          QaS EnGiNe");

jTextField4.setBackground(new java.awt.Color(204, 204, 204));
jTextField4.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextField4ActionPerformed(evt);
    }
});

jButton1.setText("SeArCh");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});

jTextArea1.setColumns(20);
jTextArea1.setEditable(false);
jTextArea1.setLineWrap(true);
jTextArea1.setRows(5);
jScrollPane1.setViewportView(jTextArea1);
```

```
        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 447,
Short.MAX_VALUE)
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 145, Short.MAX_VALUE)
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(43, 43, 43)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel4)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                                .addComponent(jTextField2, javax.swing.GroupLayout.DEFAULT_SIZE,
360, Short.MAX_VALUE))
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                                .addComponent(jTextField1, javax.swing.GroupLayout.DEFAULT_SIZE,
360, Short.MAX_VALUE))
                            .addComponent(jLabel2)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel5)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
                              .addComponent(jTextField3, javax.swing.GroupLayout.DEFAULT_SIZE,
360, Short.MAX_VALUE))))
                        .addGroup(layout.createSequentialGroup()
                              .addContainerGap()
                              .addComponent(jTextField4, javax.swing.GroupLayout.DEFAULT_SIZE, 467,
Short.MAX_VALUE)))
                  .addContainerGap())
               .addComponent(jSeparator1, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 487, Short.MAX_VALUE)
               .addGroup(layout.createSequentialGroup()
                  .addGap(96, 96, 96)
                  .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 212,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addContainerGap(179, Short.MAX_VALUE))
               .addGroup(layout.createSequentialGroup()
                  .addGap(94, 94, 94)
                  .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 186,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addContainerGap(207, Short.MAX_VALUE))
               .addGroup(layout.createSequentialGroup()
                  .addGap(23, 23, 23)
                  .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 378,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addContainerGap(86, Short.MAX_VALUE))
               .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                  .addContainerGap()
                  .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                  .addContainerGap())
         );
         layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addGroup(layout.createSequentialGroup()
                  .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 58,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addGap(11, 11, 11)
                  .addComponent(jLabel2)
                  .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                     .addComponent(jLabel3)
                     .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

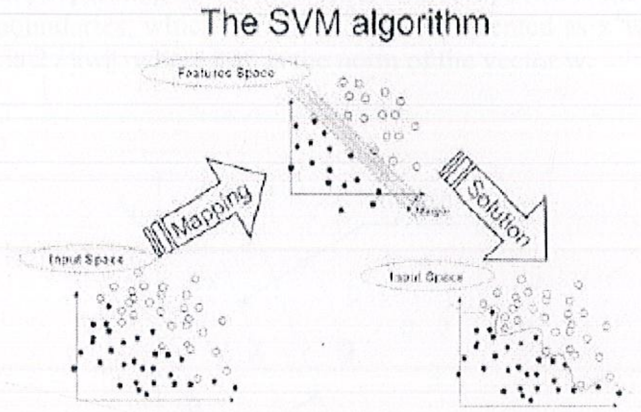
```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
               .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
               .addComponent(jLabel4))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
               .addComponent(jLabel5)
               .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(9, 9, 9)
            .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 31,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(11, 11, 11)
            .addComponent(jTextField4, javax.swing.GroupLayout.PREFERRED_SIZE, 28,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jButton1)
            .addGap(28, 28, 28)
            .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
      );

      pack();
   }// </editor-fold>//GEN-END:initComponents

   private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jTextField2ActionPerformed
      // TODO add your handling code here:
   }//GEN-LAST:event_jTextField2ActionPerformed

   private void jTextField4ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jTextField4ActionPerformed
      // TODO add your handling code here:
   }//GEN-LAST:event_jTextField4ActionPerformed

   private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
      Scanner wordFile;
      String text=null;
      Integer count;
       Data d;
       Graphics g;
       Font f,f_ini;
       f_ini=jTextArea1.getFont();

       for(String word : frequencyData.keySet( ))
      {
          if(jTextField4.getText().compareTo(word)==0)
          {
             String docs=frequencyData.get(word).getDocs().replace("-> | ", " ");
             docs=docs.substring(1);
       String sunny[]=docs.split("  ");
       System.out.println(docs);

       System.out.println(sunny.toString());
for(int i=0;i<sunny.length;i++)
{
  //System.out.println(sunny[i]);
             try
      {
        wordFile = new Scanner(new FileReader(sunny[i]));
        jTextArea1.append("\n"+sunny[i].toUpperCase()+"\n"+"-----------------\n");
      }
      catch (FileNotFoundException e)
      {
 System.err.println(e);
 return;
      }

             while (wordFile.hasNext( ))
      {
  word = wordFile.next( );
word = word.toLowerCase();
if(jTextField4.getText().compareTo(word)==0)
          {
   //jTextArea1.setFont(f_ini);
   jTextArea1.append(text);
   //f = new Font ("Monospaced", Font.BOLD | Font.ITALIC, 14);
     text="";
     //jTextArea1.setFont(f);
     jTextArea1.append("\t <<<<"+word+">>>>\t");
}
else
{
  text=text+" "+word;
}
   }
        // jTextArea1.setFont(f_ini);
  jTextArea1.append(text);
        /*f = new Font ("Monospaced", Font.BOLD | Font.ITALIC, 14);
        jTextArea1.setFont(f);
        jTextArea1.setText(text);*/

        //jTextArea1.setText("      " + word + "      " + frequencyData.get(word).getDocs()+"
" +frequencyData.get(word).getCounter()+ "        ");
      }
    //System.out.println("      " + word + "      " + frequencyData.get(word).getDocs()+"
" +frequencyData.get(word).getCounter()+ "        ");
   }
    }
  }//GEN-LAST:event_jButton1ActionPerformed
  public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
      public void run() {
        new NewJFrame().setVisible(true);
      }
    });
  }

  // Variables declaration - do not modify//GEN-BEGIN:variables
  private javax.swing.JButton jButton1;
  private javax.swing.JLabel jLabel1;
  private javax.swing.JLabel jLabel2;
  private javax.swing.JLabel jLabel3;
  private javax.swing.JLabel jLabel4;
  private javax.swing.JLabel jLabel5;
  private javax.swing.JLabel jLabel6;
  private javax.swing.JPanel jPanel1;
  private javax.swing.JScrollPane jScrollPane1;
  private javax.swing.JSeparator jSeparator1;
  private javax.swing.JTextArea jTextArea1;
  private javax.swing.JTextField jTextField1;
  private javax.swing.JTextField jTextField2;
  private javax.swing.JTextField jTextField3;
  private javax.swing.JTextField jTextField4;
  // End of variables declaration//GEN-END:variables   }
```

CHAPTER 5—SVM

Introduction

- Support vector machines (SVMs) are a set of related supervised learning methods that analyze data and recognize patterns, used for classification and regression analysis.
- The original SVM algorithm was invented by Vladimir Vapnik and the current standard incarnation (soft margin) was proposed by Corinna Cortes and Vladimir Vapnik.
- The standard SVM takes a set of input data, and predicts, for each given input, which of two possible classes the input is a member of, which makes the SVM a non-probabilistic binary linear classifier.
- SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.
- A support vector machine constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space, which can be used for classification, regression or other tasks.
- In SVM literature, a predictor variable is called an attribute, and a transformed attribute that is used to define the hyperplane is called a feature. The task of choosing the most suitable representation is known as feature selection. A set of features that describes one case is called a vector. So the goal of SVM modeling is to find the optimal hyperplane that separates clusters of vector in such a way that cases with one category of the target variable are on one side of the plane and cases with the other category are on the other size of the plane. The vectors near the hyperplane are the support vectors.

The figure represents the overview of SVM process:-



Margin maximization

Assume, there is a new company j, which has to be classified as solvent or insolvent according to the SVM score. In the case of a linear SVM the score looks like a DA or Logit score, which is a linear combination of relevant financial ratios $x_j = (x_{j1}, x_{j2}, \ldots x_{jd})$, where x_j is a vector with d financial ratios and x_{jk} is the value of the financial ratio number k for company j, k=1,…,d. So zj , the score of company j, can be expressed as

$$\mathbf{z_j = w_1x_{j1} + w_2x_{j2} + \ldots + w_dx_{jd} + b}$$

In a compact form:

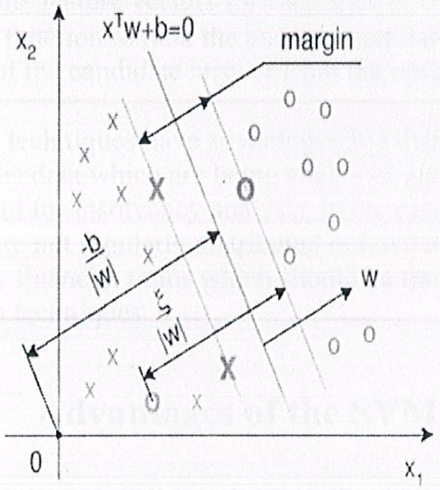
$$\mathbf{z_j = x_j^T w + b}$$

where w is a vector which contains the weights of the d financial ratios and b is a constant. The comparison of the score with a benchmark value (which is equal to zero for a balanced sample) delivers the "fore-cast" of the class – solvent or insolvent – for company j.

In order to be able to use this decision rule for the classification of company j, the SVM has to learn the values of the score parameters w and b on a training sample. Assume this consists of a set of n companies i = 1, 2, …, n. From a geometric point of view, calculating the value of the parameters w and b means looking for a hyperplane that best separates solvent from insolvent companies according to some criterion.

The criterion used by SVM's is based on margin maximization between the two data classes of solvent and insolvent companies. The margin is the distance between the hyperplanes bounding each class, where in the hypothetical perfectly separable case no observation may lie. By maximizing the margin, we search for the classification function that can most safely separate the classes of solvent and insolvent companies.

The graph below represents a binary space with two input variables. Here crosses represent the solvent companies of the training sample and circles the insolvent ones. The threshold separating solvent and insolvent companies is the line in the middle between the two margin boundaries, which are canonically represented as $x^T w + b = 1$ and $x^T w + b = -1$. Then the margin is $2 / ||w||$, where $||w||$ is the norm of the vector w.



Geometrical Representation of the SVM Margin

Key ideas of SVM

SVM offer the following advantages over conventional statistical learning algorithms:

1. High generalization performance even with feature vectors of high dimension.
2. The ability to manage kernel functions that input data to higher dimensional space without increasing computational complexity.

Need for SVM

- From the view point of machine learning, answer selection id defined as task of training and classifying candidate answer into correct and incorrect answers for a given question.
- To apply SVM, we have to prepare a set of training examples that contain feature vectors.
- For each question, the QA system analyzes the question, retrieve the document released to the question, and lists the candidate answers.
- The system parameters that were computed in the process are recorded and used to create the feature vectors for each answer candidate.
- We use a function to rank the answer candidate, where the function represents the distance of the candidate answer from the optimal hyper plane.

All classification techniques have advantages and disadvantages, which are more or less important according to the data which are being analysed, and thus have a relative relevance. SVMs can be a useful tool for insolvency analysis, in the case of non-regularity in the data, for example when the data are not regularly distributed or have an unknown distribution. It can help evaluate information, i.e. financial ratios which should be transformed prior to entering the score of classical classification techniques.

Advantages of the SVM technique

- By introducing the kernel, SVMs gain flexibility in the choice of the form of the threshold separating solvent from insolvent companies, which needs not be linear and even needs not have the same functional form for all data, since its function is non-parametric and operates locally. As a consequence they can work with financial ratios, which show a non-monotone relation to the score and to the probability of default, or which are non-linearly dependent, and this without needing any specific work on each non-monotone variable.
- Since the kernel implicitly contains a non-linear transformation, no assumptions about the functional form of the transformation, which makes data linearly separable, is necessary. The transformation occurs implicitly on a robust theoretical basis and human expertise judgement beforehand is not needed.
- SVMs provide a good out-of-sample generalization, if the parameters C and r (in the case of a Gaussian kernel) are appropriately chosen. This means that, by choosing an appropriate generalization grade, SVMs can be robust, even when the training sample has some bias.

- SVMs deliver a unique solution, since the optimality problem is convex. This is an advantage compared to Neural Networks, which have multiple solutions associated with local minima and for this reason may not be robust over different samples.
- With the choice of an appropriate kernel, such as the Gaussian kernel, one can put more stress on the similarity between companies, because the more similar the financial structure of two companies is, the higher is the value of the kernel. Thus when classifying a new company, the values of its financial ratios are compared with the ones of the support vectors of the training sample which are more similar to this new company. This company is then classified according to with which group it has the greatest similarity

Disadvantages of the SVM technique

- The biggest limitation of the support vector approach lies in choice of the kernel.
- Another limitation is speed and size, both in training and testing.
- Discrete data presents another problem.
- Although SVM's have good generalization performance, they can be abysmally slow in test phase.
- The most serious problem with SVM's is the high algorithmic complexity and extensive memory requirements of the required quadratic programming in large-scale tasks.

Conclusion

SVM's can produce accurate and robust classification results on a sound theoretical basis, even when input data are non-monotone and non-linearly separable. So they can help to evaluate more relevant information in a convenient way. Since they linearize data on an implicit basis by means of kernel transformation, the accuracy of results does not rely on the quality of human expertise judgement for the optimal choice of the linearization function of non-linear input data. SVM's operate locally, so they are able to reflect in their score the features of single companies, comparing their input variables with the ones of companies in the training sample showing similar constellations of financial ratios. Although SVM's do not deliver a parametric score function, its local linear approximation can offer an important support for recognizing the mechanisms linking different financial ratios with the final score of a company. For these reasons SVM's are regarded as a useful tool for effectively complementing the information gained from classical linear classification techniques.