Information Retrival- search engine project

Scobo-Engine

Project report

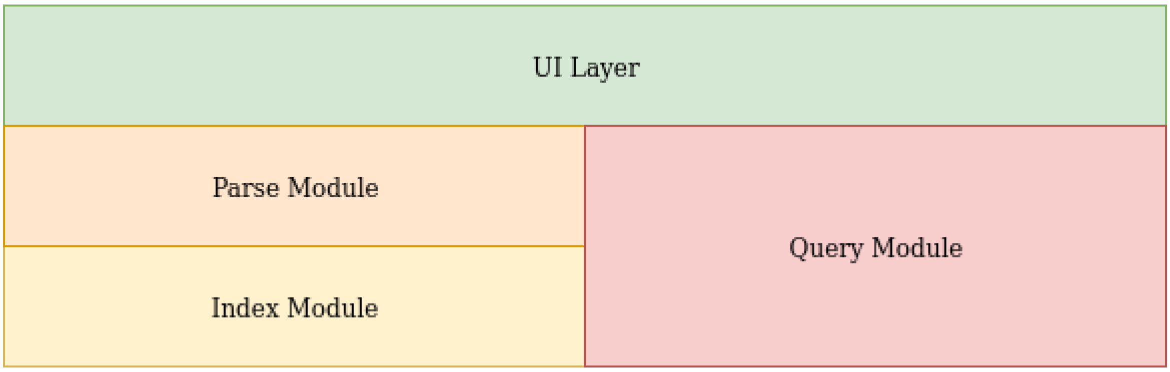
Shoham Zarfati- 318501418 Hod Twito- 315230482

19/12/2019

Scobo-Engine Architecture

Scobo can be divided into four modules - three core modules and a UI modules, these modules have a mostly hierarchical relationship and thus we will look at them as layers.

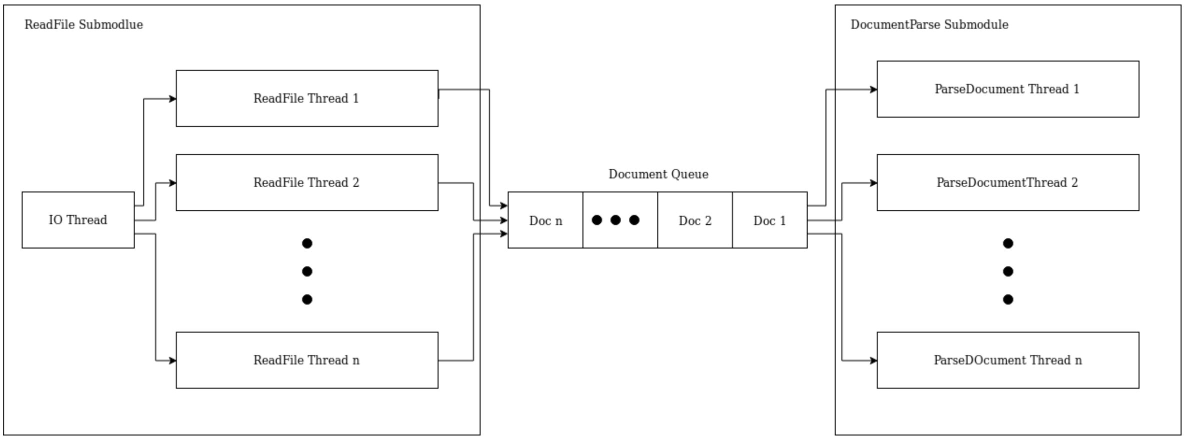
The core modules are **Parse**, **Index**, **Query**, where the first two modules are separate as they are used in the initial setup of the engine in order to create the necessary data to later be searched, and the Query module is used to search the data and retrieve results. finally the **UI module** is used to facilitate user communication with the other layers.



Parse Module

The Parse module receives the path to the corpus as input and is responsible for creating all document to termList maps that will later be used for indexing.

The Parse module is divided into two main parts, the first is the ReadFile submodule that is responsible for asynchronously taking multiple files from the corpus and splitting them up into documents these documents are then sent to the DocumentParse submodule that takes a document and finds all the terms appearing in it and creates `<document, termList>` mappings. (below is an abstract representation of the process)



Index Module

The Index module receives the mappings created by the Parse module and converts them into an inverted index mapping terms to a list of documents they appear in, this process is done in two phases:

- **Invert Phase**: first a batch of documents is taken from the parser and then is inverted into into term -> document mappings and then those mappings are written into a single posting file.
- **Merge Phase**: after all the documents have been inverted they are all merged into a single inverted file where each line is a term -> documents mapping

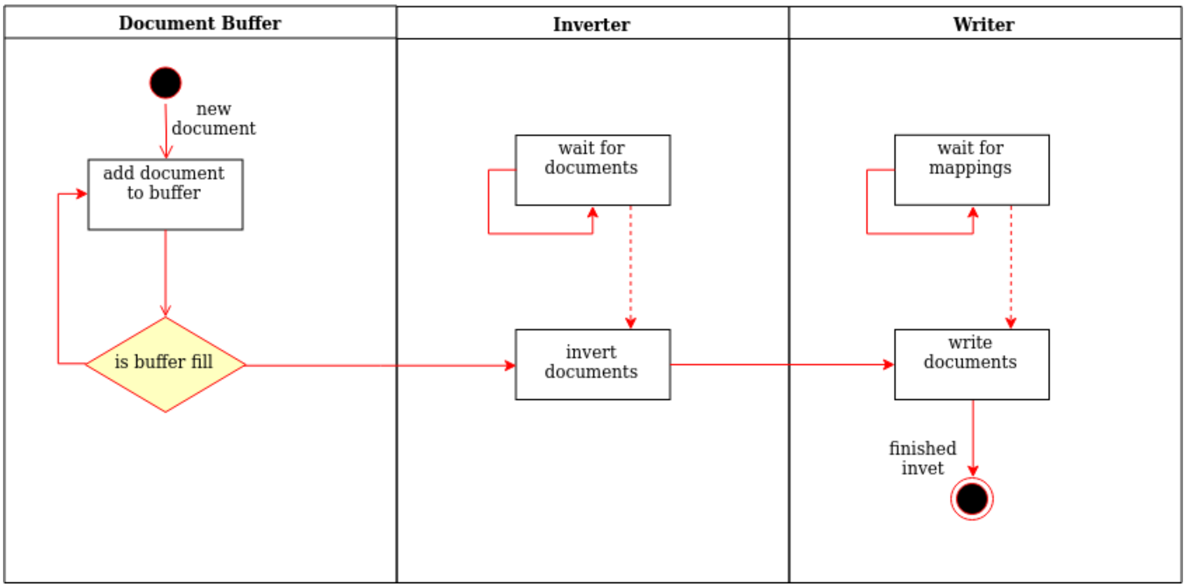
In addition to the inversion process the index module is tasked with the semantic analysis of the corpus, since implementing a full semantic analysis algorithm is beyond the scope of this project, the index module uses a prebuilt map of words -> similar words based on the GloVe project, the process of creating said map from the GloVe project is described below.

Invert Phase

The invert phase starts once the parser is finished with the first document, the document is inserted into the document buffer, once the document buffer fills up it sends all of its documents to be inverted and then continues collecting new documents.

When a group of document arrives to be inverted, they are converted from `<document, termList>`mappings into `<term, documentList>` mappings and then are sent to be written into a new posting file. In the process of inverting the the documents a dictionary and a document map are created,
the dictionary holds mappings from terms to their statistics and a pointer to the terms posting (is null at this stage), the document map holds mappings from document ids (generated by the mapper) to the documents data, such as document name, max term frequency and length.

The mappings are written to a file sorted alphabetically by the term string, such that each mapping is written in a new line with its document list. (below is an abstract representation of the process)



Merge Phase

When all documents have been written into posting files, the indexer initiates a merge, during the merge all the posting files are read concurrently line by line. In each iteration we check to see which is the alphabetically minimal term in all the lines, then we merge all the lines containing said term and write the merged line into the inverted file, at this point the dictionary pointer is updated to point to the new line written to the inverted file.

Semantic Analysis

As mentioned above a map of word -> similar words is used for semantic improvement of the queries, this map was created from the GloVe project in the following process:

1. The GloVe file consists of words -> vector mappings where each entry in the vector represents the words similarity to some concept. The following is done on the smallest provided GloVe file.
    - first clean the file of all non words and convert all the capitalized words to lower case.
    - create a new file where all the words are stemmed and if two or more words have the same stem the stems vector will be the average of all the vectors of the unstemmed words.
    - run a similarity function between each word and all the other words (we used simple cosine similarity), save the 10 words with the heights similarity for each word.
2. Since running the above process for the larger GloVe files would take too ling we ran a similar process with an added step for cleaning, we took a dictionary of English words form GitHub - https://github.com/dwyl/english-words/blob/master/words_alpha.txtand in the cleaning step we removed all words that do not appear in the English dictionary, this produced a much smaller file after the clean step on which we ran the reset of the steps described in step 1.
3. Now we have 4 files, two files generated from the small GloVe and two from the big GloVe, lastly we merged the files to produce two files, one stemmed and one unstemmed, for the merge we simply took the union of both files and for any intersection we replaced the entry from the file generated from the small GloVe by the one generated from the big GloVe (this because presumably the similarity generated from the bigger GloVe file is more generally accurate since the data set is much bigger)

Now we have two maps of word -> similar words, one for stemmed words and one for unstemmed words, when the parser finishes its work and all the words from the corpus are present in the dictionary, we will load the appropriate file (depending on if the user chose to run the engine with stemming or not) and compare it with the dictionary we created, we will then save a smaller file containing only words found in the dictionary, though the words in the similarity vector might still not be available in the dictionary, those words will simply be ignored in the query analysis.

Query Module

The Query Module is ran independently of the parse and index modules, though it uses their output - the inverted index in order to process queries and return documents quickly. The Query Module receives a query and returns a list of documents that are most relevant to the query.

The Query processing in split into three stages:

- Parsing - the query is treated as a document and the parser is ran on it in order to find all the terms preset in the query.
- Retrieval - all the documents that might be relevant to the terms in the query (the documents in the term's posting) are collected.
- Ranking - the retrieved documents are scored based on the BM25 similarity function and only the most relevant documents are then returned in order of relevance. we use two similarity function, one semantic and the bm25 function itself.
    - bm25 - the function iterates all terms in the intersection between the query and a given document, and uses weights (k, b) that we chose by trial and error, and normalization

factors like the terms document frequency, number of documents in the corpus, and the average document length to achieve a similarity measure between the query an the document.

  - semantic - for semantic ranking we added semantic fields to the query, an then created a measure that is a weighted average of the similarity between parts of the query and parts of the document, more precisely we calculated the bm25 measure between only the entities, then only the terms, then only the semantic fields, then only numbers, and then averaged the results using weights for each semantic part of the query, the weights given to numbers and semantic fields is 0.1 as they are less indicative of the topic of the query, and 0.4 to the entities and terms as they give a better indication for the topic of the query.

Scobo-Engine Documentation

This document contains implementation level documentation for the engine. In order to get an overview of how the engine operates and how the classes described here operate together see the Architecture document.

Parser Package

Document Class

Holds the information of a document including numbers, terms, and entities

- `public Document(String name)` : Creates a new document with the given name (DOCNO).
- `public void addNumber(String term)` : Adds a number term to the document.
- `public void addTerm(String word)` : Adds a non number or entity term to the document.
- `private boolean isWordNumber(String word)` :
  check to see if the word is a number with a postfix like 10m or 10M.
- `public void addEntity(String entity)` : Adds an entity term to the document.
- `private Integer computeAdd(String term, Integer frequency)` :
  function used to add a term to one of the maps.

Expression Class

Represents an expression within the text of a document

- `public Expression(int startIndex, int endIndex, String expression, String doc)` :
  Creates expression with the given indexes in the doc and the string itself.
- `public Expression()` : Creates empty expression.
- Getters for the expressions fields :
  - `public int getStartIndex()`
  - `public int getEndIndex()`
  - `public String getExpression()`
  - `public String getDoc()`
- `public boolean isPercentExpression()` :check if expression is a percent expression
  legitimate percent expressions are:
  - %
  - percent
  - percentage
- `public boolean isMonthExpression()` :
  check if expression is a percent expression
  legitimate percent expressions is defined by the month table built in the static function `buildMonthTable()`
- `public boolean isDollarExpression()` :
  check if expression is a dollar expression
  legitimate dollar expressions is defined by the dollar table built in the static function `buildDollarTable()`

- `public boolean isPostfixExpression()` :
  check if expression is a postfix expression
  legitimate postfix expressions is defined by the numbers postfixes table built in the static function `buildPostfixTable()`
- `public boolean isWeightExpression()` :
  check if expression is weight expression
  legitimate weight expressions are defined by the weight table built in the static function `buildWeightTable()`
- `public boolean isDistanceExpression()` :
  check if expression is distance expression
  legitimate distance expressions are defined by the distance table built in the static function `buildDistanceTable()`
- `public Expression getNextExpression()` :
  Get the next expression in the document
  the next expression is the string after this expression's space, until the next space in the text
- `public Expression getPrevExpression()` :
  Get the previous expression in the expression's document the previous expression is the string before this expression's, until the previous space in the text.
- `public Expression getNextWordExpression()` :
  Get the expression in the document
  the next word is the string after this expression's space, until the next space in the text
- `public void join(Expression exp)` : add exp to the end of this expression, with separating space
- `public String toString()` : gets String representation of the expression.
- `public int hashCode()`: gets hash code of the expression.
- Static functions to build the various tables :
  - `private static HashSet<String> buildDollarExpressions()`
  - `private static HashMap<String, String> buildMonthTable()`
  - `private static HashSet<Character> buildStoppingCharsTable()`
  - `private static HashMap<String, Double> buildNumbersPostfixTable()`
  - `private static HashMap<String, String> buildDistanceTable()`
  - `private static HashMap<String, String> buildWightTable()`

NumberExpression Class

Extension of the Expression class
holds also the numeric value of the number

- `public NumberExpression(int startIndex, int endIndex, String expression, String doc)` :
  Creates a number expression with given indexes in the doc and the string itself
- `public NumberExpression(Expression expression)` :
  Creates number expression from generic expression.
- `public double getValue()` : returns the value of the number expression.
- `protected boolean isPartOfMixedNumber()` :
  Check if number is part of mixed number mixed number is: number numerator/denominator

- `protected boolean isNumerator()` :
  Check if number is the numerator of a mixed number expression
- `protected boolean isDenominator()` :
  Check if number is the denominator of a mixed number expression
- `protected boolean isFullNumberOfMixedNumber()` :
  Check if number is the the full number part of a mixed number expression
- `private double translateValue()` : translate the value of the numeric string to number
- `public String toString()` : returns String representation of the expression.
- `public static double translateValue(NumberExpression exp)` :
  Translate the numeric expression to true number value, in order to assign the value to the number expression object
- `public static boolean isNumberExpression(Expression exp)` :
  Check if Expression is numeric expression
- `public static boolean isNumberExpression(String strExp)` :
  Check if string is numeric expression
- `public static NumberExpression createMixedNumber(NumberExpression numerator)` :
  Join fullNumber (if exist), numerator and denominator to one mixed number expression
  if no full number exist- only fraction, the number expression returned will be: 0 numerator/denominator
- `public static String getNumberString(double number)` :
  return string of the number expression, according to the rules- show only maximum three decimal numbers, if exist

Parse Class

Runnable Class, handles the parsing process for a single document.

- `protected Parse(String document, Parser parser)` :
  Creates instance of Parse for the given document.
- `private String genDocName(String document)` :
  extracts document name from the document.
- `public void run()` : Start the parsing on doc
- `private void parseText(String text)` : Parse every paragraph in the document
- `private void parseNumbers(NumberExpression numberExp)` :
  Parse numbers from the document.
- `private void parseHyphenSeparatedExp(Expression exp)` :
  parse all the hyphen separated words or numbers.
- `private void parseWords(Expression word, Matcher m)` : Parse every word in the document
- `private boolean tryDate(NumberExpression numberExp)` :
  Check the date rules date is:
  - DD Month,   add as MM-DD
  - Month DD,   add as MM-DD
  - Month YYYY, add as YYYY-MM
- `private boolean tryPercent(NumberExpression numberExp)` :
  Check percent rule legal percent term is:
  - Number%
  - Number percent
  - Number percentage

If number following this rules, it is added to the dictionary as number%

- `private boolean tryWight(NumberExpression numberExp)` :
  Check weight rule legal distance term is: number[wight postfix] where wight postfix is one of the following: {kg, KG, Kg, g, G, mg, MG} If the number follows this rule, it is added to the dictionary as number[lowercase postfix]
- `private boolean tryDistance(NumberExpression numberExp)` :
  Check distance rule legal distance term is: number[distance postfix] where distance postfix is one of the following: {km, KM, Km, cm, CM, mm, MM} If the number follows this rule, it is added to the dictionary as number[lowercase postfix]
- `private boolean tryDollars(NumberExpression numberExp)` :
  Check if number is part of price rules if number is below million the rules are:
    - price Dollars
    - price frac Dollars
    - $price

  adds to the dictionary as number Dollar
  if number is above million the rules are:
    - price Dollars
    - $price
    - $price million
    - $price billion
    - price postfix dollars
    - price million U.S. Dollars
    - price billion U.S. Dollars
    - price trillion U.S. Dollars

  adds to the dictionary as number M Dollars
- `private boolean tryPlainNumeric(NumberExpression numberExp)` :
  Check if number is plain number, without any addition plain number is every string contains only numeric chars or ,/. with or without decimal point or fraction
  adds to the dictionary as:
    - if number is bellow 1000- as it appears
    - if number is 1K <= number < 100000- as numberK
    - if number is 1M <= number < 1bn- as numberM
    - if number is 1bn <= number - as numberB

  if the number has decimal point- adds to the dictionary with maximum 3 decimal numbers
- `private boolean tryBetweenFirst(NumberExpression numberExp)` :
  Check if number is part of bigger expression in the format
  between number and number
  If it is- add to the dictionary as as hyphen separated numbers : number-number
- `private boolean tryCapitalLetters(Expression word, Matcher m)` :
  Check if word first char is upper case If it is- add to the dictionary by the rules of capital letters words:
  if exist in dictionary in low case- add in low case else- add whole word in capital letters check if the next word is also in capital, if it is, apply the same rule, and create an entity when the entity is big enough or the capital letters words are finished- add entity to dictionary.
- `private void moveUpperToLower(String upperCaseWord)` :
  check if low case word already added as capital word if it is- add as lower, and remove the capital.

- `private void handleSingleCapital(Expression word)` :
  if word exist in low case, add as low case else- add as capital word

Parser Class

Manages file reading from the corpus, and documents parsing.

- `public Parser(String path, Consumer consumer)` :
  Constructs a parser using the corpus path and a Consumer, initialize the task groups, give values to the parser elements, loads the stop words list
- `public Parser(String stopWordsPath, Consumer consumer, DocumentProvider provider)` :
  Constructs a parser using a DocumentProvider in place of using the corpus path in order to construct the documents.
- `private void loadStopWords(String path)` :
  loads the stop words list- all the words to ignore from
- `boolean isStopWord(String word)` : check if given word is stop word
- `String stemWord(String word)` :
  If configured to stem- stem a given word otherwise, return the same word.
- `void onFinishedParse(Document document)` : notify the parser that document parse is finished
- `public void start()` :
  Start the parsing process, if no DocumentProvider was set then read files from the corpus path.
- `private void finish()` : What to do when the parsing process is done
- `public void awaitRead()` : Wait until finished reading all the corpus files
- `public void awaitParse()` : Wait until parsing is done
- `public int getDocumentCount()` : return the number of documents parsed.
- `int getBatchSize()` : return the batch size configured for the parser.

ReadFile Class

Manages the reading and separating of files into documents

- `protected ReadFile(String, Parser parser)` :
  Creates a readfile that will read from the specified path and is associated with the given parser.
- `private void read(String[] batch, int start, int end)` :
  Reads a batch of files from 'batch', start reading from the 'start' index of batch up to the 'end' index.
- `private void separate(byte[][] batch)` :
  Separates the given files into documents, the files are given as an array of byte arrays where each byte array represents a files bytes.

Stemmer

This Class is the Porter Stemmer taken from : https://tartarus.org/martin/PorterStemmer/java.txt

For detailed documentation on how the stemmer works visit :
https://snowball.tartarus.org/algorightms/proter/stemmer.html

Indexer Package

Dictionary Class

Maps string representation of a term to a `Term` instance holding the relevant term statistics and posting file pointer.

Dictionary file format: Each line in the file represents a single entry in the Dictionary, each line will look like so: [term]|[document frequency]|[posting file]\n

- term - string representation of the term
- term frequency - number of times the term appears in the corpus
- document frequency - number of documents the term appears in
- posting pointer - index of the posting file this terms posting appears in

`Dictionary` is externally immutable meaning that it is immutable outside of the scope of its package (indexer)

- `protected Dictionary()` :
  Constructs a `Dictionary` with default parameters. This creates a *mutable* reference.
- `private Dictionary(int termCount, float loadFactor, int concurrencyLevel)` :
  Constructs a `Dictionary` with the given parameters.
- `protected void addNumberFromDocument(String number, int frequency)` :
  Adds the number to the dictionary, if the number was already contained its statistics will be updated, otherwise the number will be added to the dictionary.
- `protected void addTermFromDocument(String term, int frequency)` :
  Adds the term to the dictionary, if the word was already contained its statistics will be updated, otherwise the term will be added to the dictionary.
- `protected void addEntityFromDocument(String entity, int frequency)` :
  Adds the entity to the dictionary, if entity term was already contained its statistics will be updated, otherwise the entity will be added to the dictionary.
- `private void addTerm(String term, int count, int frequency)` :
  helper function to add a term to the dictionary. where frequency is the number of times the term appears in the new document.
- `public Optional<Term> lookupTerm(String term)` :
  Retrieves information about a term via a `Term` object
- `Optional<Term> lookupEntity(String term)` :
  Retrieves information about an entity via a `Term` object
- `boolean isEntity(String entity) :`
  returns true if the string is a valid entity, false otherwise.
- `public boolean contains(String term)` :
  returns true if the term exists in the dictionary, false otherwise.
- `public int size()` :
  Returns the number of key-value mappings in this map. If the map contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.
- `public Collection<Term> getTerms()` :
  returns Collection of all the terms in the dictionary.
- `void save()` :
  Saves the `Dictionary` to the directory specified by`Configuration`.
- `public void clear() throws IOException` :
  Removes all of the entries from the dictionary, and deletes the dictionary file.
- `public static Dictionary loadDictionary() throws IOException` :
  Loads the Dictionary into memory from the directory specified by `Configuration` and returns a reference to it.

- `private static String getPath()` :
  returns the path to the dictionary file as specified by `Configuration`.

DocumentBuffer Class

Represents a buffer of documents that builds up until a term limit is reached (or exceeded) at which point the buffer is flushed and queued to be inverted.

- `DocumentBuffer(Indexer indexer)` : constructs a new document buffer.
- `void addToBuffer(Document document)` : adds a document to the buffer.
- `void flush()` : flushes the document buffer.

DocumentMap Class

Maps document IDs to document data and generates said IDs.

Document Map file format: Each line in the file represents a document ID -> document data mapping. each line line will look like so: [document ID]|[(document data)]\n

- document ID - id given to the document by the map
- document data - csv data about the document including document name

`DocumentMap` is externally immutable meaning that it is immutable outside of the scope of its package (indexer)

- `protected DocumentMap(Indexer indexer)` :
  Creates a `DocumentMap` in ADD mode. This creates a *mutable* reference.
- `private DocumentMap(int mapSize, float loadFactor)` :
  private initialization constructor used by the package constructor and the external `loadDocumentMap` function.
- `int addDocument(Document document)` : Adds a document to the map, giving it a unique ID.
- `public Optional<DocumentMapping> lookup(int docID)` :
  Gets the document mapping of the given document ID.
- `void updateEntity(String termPostingStr)` :
  Parses the term posting string and updates all the documents that the entity appears in. If the entities frequency is high enough in some document it will entered into the list of most dominant entries in the document.
- `void save()` : saves the document map into the document map file.
- `public void clear() throws IOException` :
  Removes all of the document mappings from the map, and deletes the document map file.
- `public static DocumentMap loadDocumentMap() throws IOException` :
  Loads the document map in LOOKUP mode into memory and returns a reference to it.
- `private static String getPath()` :
  returns the path to the dictionary file as specified by Configuration.
- `DocumentMapping` Class

  Holds the data of the documents mapping:

  - `name` - the name of the document (DOCNO)
  - `maxFrequency` - frequency of the term or entity that is most frequent in the document
  - `length` - number of terms or entities that appear the document (not unique)

  - `dominantEntities` - list of the most dominant entities in the document.
  - `synchronized void updateEntity(String entity, int frequency)` :
    Updates the `dominantEntities` list with a new entity and its frequency.

Indexer Class

Manages the indexing of the documents produced by the `Parser`

indexing is done in two phases:

- first a batch of documents is taken from the parser and then is inverted into into term -> document mappings and then those mappings are written into a single posting file.
- after all the documents have been inverted they are all merged into a single inverted file where each line is a term -> documents mapping

during this process a `Dictionary` and `DocumentMap` are created in order to later retrieve information from the inverted file.

- public void onFinishParser() :
  Callback meant to be used by the parser to notify the indexer that the last of the documents has been parsed and the indexer can now start entering it's second phase.
- `public void awaitIndex()` :
  Waits until all indexing is done. when this method returns all posting files are gone and the the inverted file, dictionary, document map are ready to be used.
- `public void consume(Document document)` : Adds a document to the inverted index.
- `void queueInvert(LinkedList<Document> documents)` :
  Queues an invert task for the given list of documents.
- `private void invert(LinkedList<Document> documents)` :
  Inverts the given document list as described above and then queues a task to write the resulting map into a posting file.
  - `private void invertNumbers(int docId, PostingFile newPosting, Documetn document)` :
    inverts number terms.
  - `private void invertTerms(int docId, PostingFile newPosting, Documetn document)`:
    inverts terms that are not numbers or entities.
  - `private void invertEntities(int docId, PostingFile newPosting, Documetn document)` :
    inverts entity terms.
- `public int getTermCount()` : returns the number of terms in the dictionary.

PostingCache Class

Manages the creation and deletion of posting files and the inverted file

- `static void initCache(Indexer cacheIndexer)` :
  Initializes the cache, after this method is called it is possible to start using the cache to create posting files and later an inverted file.
- `static Optional<PostingFile> newPostingFile()` : Posting file factory, creates new posting files.
- `static void queuePostingFlush(PostingFile postingFile)` :
  Queues a flush of a posting file, this will write the posting file to the disk under a name

matching it's id.

- `private static void flushPosting(int postingFileId, TermPosting[] postings)` : Flushes the posting file to the disk.
- `static void merge(Dictionary dictionary)` : Merges all the posting files into an inverted file.
- `static void clean()` : Deletes all the posting files.
- `public static void deleteInvertedFile() throws IOException` : Deletes the inverted file.
- `private static String getPostingPath()` : get path to Posting file directory.
- `private static String getInvertedFilePath()` : get path to inverted file.
- `private static String getPostingFilePath(int postingFileID)` :
  get path to the posting file with the given id.
  note: this method does not guarantee that the file exists.

PostingFile Class

Represents a posting file while its in memory.

- `PostingFile(int postingFileID)` : Creates a posting file with the given id
- `public void addTerm(String term, int documentID, int documentFrequency)` :
  Adds a term -> document mapping to the posting file.
- `TermPosting[] getPostings()` :
  returns an alphabetically sorted array of the term postings in the posting file.
- `public int getID()` : returns posting file id.
- `public void flush()` : Writes the posting file to a file.

Term Class

Holds information about a term.

- `term`- string representation of the term.
- `termDocumentFrequency` - number of documents the term has appeared in.
- `termFrequency` : number of times the term occurred in the corpus.
- `pointer` : pointer to the terms line in the inverted file.

TermPosting Class

Represents a terms posting (a line) in a posting file.

- `public TermPosting(String term)` : Constructs a term posting with the given term.
- `public void addDocument(int documentID,int termFrequency)` :
  Add a document to the posting.
- `public String getTerm()` : return the term of the posting.
- `public String toString()` : return a string containing all the contents of this term posting.

Searcher Class

Manages the retrieval of results for a single query

- `public Searcher(Query query, QueryProcessor manager)` :
  Constructs Searcher for a given query that is managed by the given manager.
- `public void run()` : runs the searcher.

- `private void expandQuery()` : semantically expands the query, adding semantic fields.
- `private void expandTerm(String[] sim)` :
  semantically expands the term, adding fields that are semantically similar to the term.
- `private void loadDocuments() throws IOException` :
  loads all the documents that might be relevant to the query, from the inverted file.
- `private void addDocuments(String line)` :
  parses a line of the inverted file, adding the documents in it to the relevant documents.

Query Package

Query Class

Query represents a parsed query, a query is a specialized document where we add additional id and semanticTerms fields.

- `public Query(Document document)` : Creates a query from the document representing it.
- `public void addSemantic(String field)` :
  adds a semantic field to the query, these are fields that do not count as normal terms and are used in the ranking.
- `public int get(String term)` :
  returns the frequency of the term in the query, including semantic terms, exists as both a semantic field and a regular term its regular frequency will be returned.
- `public int length()` : length of the query (semantic fields included).
- `public Iterator<Map.Entry<String,Integer>> iterator()` :
  returns an iterator over all <term, frequency> mappings.
- `QueryIterator` class: implements an iterator that concatenates iteration over the queries terms, numbers, entities, and semantic fields.

QueryProcessor Class

Manages the querying process, may only process one request at a time, though a request may consist of multiple queries.

- `public QueryProcessor(String indexPath, Dictionary dictionary, DocumentMap documentMap)` :
  Initializes the query processor with the given dictionary and document map, this constructor blocks while loading the Similarity file.
- `private void loadGloSim()` : loads the similarity vectors.
- `public QueryResult query(String... queries)` :
  Request for a group of queries to be processed, where the queries may be any free text, the query result can later be used to see the documents most similar to each of the queries.
- `public QueryResult query(Pair<Integer,String>[] queries)` :
  Request for a group of queries to be processed, where the queries may be any free text, but are provided with a query id, the query result can later be used to see the documents most similar to each of the queries.
- `public void consume(Document document)` : initiates search using the given query.
- `public void onFinishParser()` :
  notifies the query processor that parsing of the queries is complete, and no new consume calls will be made.
- `private static List<String> asDocuments(String... queries)` :
  creates queries that fit the format our parser expects from the given free text queries.

- `private static List<String> asDocuments(Pair<Integer, String>[] queries)` : creates queries that fit the format our parser expects from the given `<queryID, query>` pairs.

QueryResult Class

Holds query results possibly of multiple queries, if only one query was requested, use `first()` in order to retrieve the ranking.

- `QueryResult(String... queries)` : constructs query result from free text queries.
- `QueryResult(Pair<Integer, String>[] queries)` : constructs query result from structured queries.
- `void updateResult(int queryID, int[] ranking)` : sets the result for the given queryID.
- `public Optional<int[]> resultOf(String query)` : return ranking for the given query if it exists.
- `public Optional<int[]> resultOf(int queryID)` : return ranking for the given queryID if it exists.
- `public int[] get()` : return one of the available results, if only one is available it will be returned.
- `public Set<Map.Entry<Integer, int[]>> sorted()` : return a sorted view of the results.
- `public Iterator<Map.Entry<Integer, int[]>> iterator()` : returns iterator over `<queryID, ranking>` pairs.
- `public void forEach(Consumer<? super Map.Entry<Integer, int[]>> action)` : applies the given action on each `<queryID, ranking>` .
- `public Spliterator<Map.Entry<Integer, int[]>> spliterator()` : returns spliterator over `<queryID, ranking>` pairs.

Ranker

This class calculates the similarity between a query and documents and ranks the documents according to said similarity.

- `private Ranker(Query query, QueryProcessor manager)` : constructs a ranker for a given query and QueryProcessor.
- `protected void updateRanking(int docID, double sim)` : updates the ranking for a given docID that has the given similarity.
- `public int[] getRanking()` : ranking of the docID's where the doc at index 0 is the one ranked the highest.
- `public abstract void rank(int docID, Map<String,Integer> tf)` : Adds the given doc to the ranking.
- `public static Ranker semantic(Query query, QueryProcessor manager)` : Creates a semantic Ranker.
- `public static Ranker bm25(Query query, QueryProcessor manager)` : Creates a bm25 Ranker.
- `SemanticRanker` class - implements the rank method using a semantic algorightm.
- `BM25Ranker` class - implements the rank method using the bm25 algorightm.

Gui Package

GUI Class

The GUI class is the application's entry point and not much more then that, it creates the stage and scene and then launches the application, the controller class handles the run of the application.

- `public void start(Stage primaryStage) throws Exception`
  the only method in this class, it sets up the application and launches it.

Controller Class

This class *controls* the application during its runtime, most of its functionality is event handling, launching the engine and displaying its data.

- `public void setStage(Stage stage)` :
  called when the application starts and the stage is initialized, serves as an init method for the controller.
- `private void updateOptions()` : updates the configurations of the application
- `public void onClickBrowseCorpus()` :
  event, triggered when the user browses for the corpus path, updates the text field as well as updating the configuration with the selected corpus path.
- `public void onClickBrowseIndex()` :
  event, triggered when the user browses for the index path, updates the text field as well as updating the configuration with the selected index path.
- `public void onClickBrowseLog()` :
  event, triggered when the user browses for the log path, updates the text field as well as updating the configuration with the selected log path.
- `public void onClickBrowseQuery()` :
  event, triggered when the user browses for a query file, updates the text field as well as updating the configuration with the selected query file.
- `public void onClickSaveOptions()` :
  event, triggered when the user clicks the save button, this saves the currently selected configuration to disk so it will be available in the next run of the application.
- `public void onClickRunIndex()` :
  event, triggered when the user clicks the run index button , runs the parser/indexer and creates the inverted index files.
- `public void onClickRunQuery()` :
  event, triggered when the user clicks the run query button, runs the query processor on the provided queries and produces a result file or a window with the results.
- `private void handleTextQuery()` : handles a free text query.
- `private void handleFileQuery()` : handles a query file.
- `public void onClickReset()` :
  event, triggered when the user clicks the reset button, clears the memory and disk of the dictionary and inverted file.
- `public void onClickLoadDict()` :
  event, triggered when the user clicks the load dictionary button, loads the dictionary into memory.
- `public void onClickShowDict()` :
  event, triggered when the user clicks the show dictionary button, opens a window with a table view containing two columns, all the terms as they appear in the dictionary, and their frequencies in the corpus.
- `private void makeViewable()` : creates a sorted view of the dictionary.

- `private void showAlert(String title, String message)` : shows alert with given text and message.
- `private Pair<Integer, String>[] getQueriesFromFile()` :
  parses the query file and returns an array of queries.
- `private void saveQueryResults(QueryResult result)` : saves the query results as a file.
- `private void showQueryResult(QueryResult result)` : shows the query result in a new window.
- `private void showDocumentEntities(String docName, List<Pair<String, Integer>> entities)` :
  displays alert showing the given dominant entities with the document name.

DictionaryEntry Class

This class represents a table entry in the dictionary table window

- `public DictionaryEntry(String term, int frequency)` :
  Constructor, creates new entry with the given term and frequency.
- `public String getTerm()` : get the term in the entry.
- `public String getFrequency()` : get the frequency in the entry.
- `public void setTerm(String term)` : change the entry's term to a given term.
- `public void setFrequency(String term)` : change the entry's frequency to a given frequency.

Util Package

Configuration

Scobo Engine Configuration manager. Handles the creation, loading, and updating of the engine's configuration.

- `public static Configuration getInstance()` : returns an instance of Configuration.
- `private void loadConfiguration()` : loads configuration file.
- `private void initConfiguration()` : initialize configuration file.
- `public void updateConfig()` :
  In order for the configuration changes to persist this method must be called explicitly otherwise the changed configuration will only apply to the current run.
- `public void setCorpusPath(String path)` :
  Changes the corpus path, this change only applies to the current run of the engine, and will not persist unless the `updateConfig()` method is called.
- `public void setIndexPath(String indexPath)` :
  Changes the index path, this change only applies to the current run of the engine, and will not persist unless the `updateConfig()` method is called.
- `public void setParserBatchSize(int parserBatchSize)` :
  Changes the parsers batch size, this change only applies to the current run of the engine, and will not persist unless the `updateConfig()` method is called.
- `public void setLogPath(String logPath)` :
  Changes the path to the log file, this change only applies to the current run of the engine, and will not persist unless the `updateConfig()` method is called.

- `public void setUseStemmer(boolean useStemmer)` :
  Changes weather or not the engine will use a stemmer, this change only applies to the current run of the engine, and will not persist unless the `updateConfig()` method is called.
- `public void setUseSemantic(boolean useSemantic)` :
  Changes weather or not the engine will use the semantic analysis, this change only applies to the current run of the engine, and will not persist unless the `updateConfig()` method is called.
- The following methods are getters for all the configurations :
  - `public String getCorpusPath()`
  - `public String getIndexPath()`
  - `public int getParserBatchSize()`
  - `public String getLogPath()`
  - `public boolean getUseStemmer()`
  - `public String getDictionaryPath()`
  - `public String getDocumentMapPath()`
  - `public String getInvertedFilePath()`
  - `public InputStream getGloVe()`
  - `public String getDictSimPath()`
- `private String getUseStemmerPath()` :
  returns the correct folder name for the index according to the value of `useStemmer` .

CountLatch Class

A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes.

[The `CountLatch` is very similar to the `CountDownLatch` it provides the same functionality only adding the ability to `countUp()` in addition to to `CountDownLatch.countDown()`.

The `CountLatch` is initialized to some count, the count may be thought of as the number of tasks/threads that need to complete before some other waiting threads can continue. The count can change through the `countDown()` and `countUp()`, whenever the count reaches 0 all the threads waiting using the `await()` method will be notified.

The maximum count is `Long.MAX_VALUE`.

- `public CountLatch(long initialCount)` :
  Creates a `CountLatch` with a specified initial count.
- `public void await() throws InterruptedException` :Causes the current thread to wait until the latch has counted down to zero, unless the thread is interrupted.

  If the current count is zero then this method returns immediately.

  If the current count is greater than zero then the current thread becomes disabled for thread scheduling purposes and lies dormant until one of two things happen:

  - The count reaches zero due to invocations of the `countDown()` method; or
  - Some other thread interrupts the current thread.

  If the current thread:

  - has its interrupted status set on entry to this method; or
  - is interrupted while waiting,

then `InterruptedException` is thrown and the current thread's interrupted status is cleared.

- `public long countUp()` : Increments the count of the latch.
- `public long countUp(int delta)` : Adds `delta` to the count of the latch.
- `public long countDown()` :
  Decrements the count of the latch, releasing all waiting threads if the count reaches zero.

Logger Class

Simplistic Logger class. This class is a singleton, as such in order to use it call `getInstance()`
Supports Three logging levels :

- MESSAGE - use `message(String)` to log message
- WARNING - use `warn(String)` to log warning
- ERROR - use `error(String)` to log error

The logs are not written to the log file instantly, this is done to improve performance and not force consecutive IO when it could be avoided. The logs will be written into the file when they reach a size of 1KB, otherwise to force the logger to write to the log file use `flushLog()`.

- `public static Logger getInstance()` : get instance of the logger class.
- `public void message(String message)` :
  Logs a message in the format: [TIME] [MESSAGE]: message
  Messages are intended as part of the regular operation of the program, and as a tool to output information such as debug data timings and more.
- `public void warn(String warning)` :
  Logs a warning in the format: [TIME] [WARNING]: warning
  Warnings are intended as errors or exceptions that the program can recover from and continue execution.
- `public void warn(Exception exception)` :

  Logs a warning in the format: [TIME] [WARNING]: warning
  Warnings are intended as errors or exceptions that the program can recover from and continue execution.
- `public void error(String error)` :

  Logs a error in the format: [TIME] [ERROR]: error
  Errors are intended as problems or exceptions that the program can *not* recover from.
- `public void error(Exception exception)` :

  Logs a error in the format: [TIME] [ERROR]: error
  Errors are intended as problems or exceptions that the program can *not* recover from.
- `private void logException(Exception exception, StringBuilder stringBuilder)` :
  formats the exception into a string and adds it to the log buffer.
- `public void flushLog()` :
  Flushes the log buffer into the log file, this method should be called when the logger is about to be destroyed or othera comparable runnable that can have a priority of execution.wise become unavailable.
- `private void tryFlushLog(int logSizeBytes)` :
  checks if the log buffer has become large enough to flush if so flush it.
- `private String getTime()` : creates a formatted string of the current date and time.

TaskExecutor Class

Thread pool used to execute tasks with varying priority. This implementation is practically identical to `ThreadPoolExecutor` apart from the ability to execute tasks with a given priority.

- `TaskExecutor(int poolSize, int initialSize)` :
  Creates new `TaskExecutor` with the given pool size and initial size.
- `public void execute(Runnable runnable)` :
  Enqueues a task with default priority to be executed when there is an available thread.
- `public void execute(Runnable runnable, int priority)` :
  Enqueues a task with default priority to be executed when there is an available thread.
- `private static Task taskOf(Runnable runnable, int priority)` :
  creates a task from the given runnable and priority.
- `Task` interface - a comparable runnable that can have a priority of execution.

TaskGroup Class

Encapsulates a group of tasks in order to be able to reason about them as one unit. Task Groups should be used when there is a clear grouping of tasks that can all be added to the group before all the added tasks are completed.
Example Usage:

```
//some CPU tasks that need to be executed
Runnable[] CPUTasks = getCPU();
//some IO tasks that need to be executed
Runnable[] IOTasks = getIO();

// create CPU task group
TaskGroup CPUGroup = TaskManager.getTaskGroup(GroupType.IO);
//create IO task group
TaskGroup IOGroup = TaskManager.getTaskGroup(GroupType.COMPUTE);

// open the group to ensure that awaitCompletion works as intended.
CPUGroup.openGroup();
// add the CPU tasks to the CPU group
for (Runnable r : CPUTasks)
     CPUGroup.add(r);
CPUGroup.closeGroup();

// open the group to ensure that awaitCompletion works as intended.
IOGroup.openGroup();
// add the IO tasks to the IO group
for (Runnable r : IOTasks)
     IOGroup.add(r);
CPUGroup.closeGroup();

CPUGroup.awaitCompletion();
System.out.println("all CPU tasks have been completed");

IOGroup.awaitCompletion();
System.out.println("all IO tasks have been completed");
```

- `TaskGroup(TaskManager manager, TaskType type, TaskPriority priority)` :
  Creates a task group that will be managed by the provided manager with the given type and priority.
- `public void add(Runnable task)` :
  Adds a task to the group and schedules it through the `TaskManager` as a task matching the groups type.
- `public void add(Collection<? extends Runnable> tasks)` :
  Adds a collection of tasks to the group and schedules them through the`TaskManager` as a task matching the groups type.
  Calling `awaitCompletion()` after using this method ensures that all the tasks added will complete before the thread calling `awaitCompletion()` will be notified.
- `public void openGroup()` :
  Ensures that if a threads calls `awaitCompletion()` on this group, it will not be notified before `closeGroup()` is called.
  This is intended in order to ensure that a batch of tasks that cannot be executed using `add(Collection)` will all be executed before a thread calling `awaitCompletion()` is notified. This is done by calling `openGroup` before calling `awaitCompletion()` then adding the tasks to the group, and calling `closeGroup()` when all tasks have been added. (see the example in the class documentation)
- `public void closeGroup()` :
  Notifies the group that all tasks have been added and threads that are waiting using `awaitCompletion()` will be notified once the tasks group is empty.
  Calling `closeGroup` more than once, has no effect.
- `public void complete()` :
  notifies the task group that a task has completed. calling this method is required when a task finishes in order to update the `awaitCompletion()`
- `public void awaitCompletion()` :
  Causes the calling thread to wait until every task that was added to the group using `add(Runnable)` has called the `complete()` method.
  The calling thread may be notified before all the planned tasks for this group have completed. for example if there was a delay in adding tasks and tasks 1,2,3 were completed before task 4 could be added to the group. When using Task Group take care to have a clear batch of tasks that can be executed.
  In order to be assured this issue does not arise use `openGroup()`

TaskManager Class

Simple task manager, servers as basically a wrapper for Executor service. This class implements the singleton pattern, in order to attain an instance use `getInstance()`.

You may also attain an instance of `TaskGroup` using it is recommended to use TaskGroups when it is needed to treat a group of tasks as one unit e.g - needing to wait for a batch of task to complete.

- `public static TaskManager getInstance()` : return instance of `TaskManager` class.
- `public static TaskGroup getTaskGroup(TaskType type)` :
  Creates a Task Group that can be used to execute tasks as part of group and treat all the tasks executed through the group as a single unit.
  The Task group will have MEDIUM priority
- `public static TaskGroup getTaskGroup(TaskType type, TaskPriority priority)` :
  Creates a Task Group that can be used to execute tasks as part of group and treat all the

tasks executed through the group as a single unit.
The Task group will have MEDIUM priority

- `public void executeIO(Runnable task, int priority)` :
  Enqueues the task into the IO task queue, the task will execute when its turn arrives.
- `public void executeCPU(Runnable task, int priority)` :
  Enqueues the task into the CPU task queue, the task will execute when its turn arrives.

Scobo-Engine report

b.

As we already know, indexing a corpus may take a lot of time, and a lot of memory. For the index part run the fastest, and run even on our “weak” PCs, we had to optimize memory usage and functions run time.

In order to optimize memory usage, we saved every batch of documents term in their own posting file and only in the end merged them- line by line, so we won’t need to upload almost 1GB of postings to the RAM. Also, we read batch of files every time, and not all at the same time, so we won’t upload all the corpus to the RAM.

In order to optimize run time, we implemented functions to handle strings on our own- for example we noticed that the java implemented functions for parse double to string is very expensive, so we implemented one of our own. Additionally, as explained before, all the process is done using threads, which managed by a task manager and task executer that we implemented in order to fit in the best way to the index phase of the engine.

c.

We chose to save our posting Terms as txt files, in the following format:

Term(|docID,tf)+

Every posting file contains about 2^16 terms- the capacity we chose to the term buffer for the postings, as described in the class DocumentBuffer we tried a lot of numbers and came up with the best results with this buffer size.

After running, we came up with between 170 to 190 posting files, and one big merged inverted file at the end.

d.

As explained in the architecture part of the report, Scobo divide the indexing phase to multiple parts:

At first the ReadFile of the Parse module reads batch of 10 files for every thread, which is about 3000 documents- a number we came up with after trying multiple options and got the best results with. The ReadFile split each file to its documents and sends it to be processed by the Parse.

The Parse get a document and catch all the terms in it according to the rules we gave it, and then sends a map of <document, Term> to the indexer. While the document is being parsed, each term found is added to the dictionary, with two numbers- counter for the tf, and pointer (for now it is Null) to the line of the term in the inverted file.

When the indexer buffer is full enough (as explained in c) it inverts it to maps <Term,document>, which is then written to a posting file.

After all the indexers are done, we have multiple posting files, which are getting merged by the merge function to one big sorted inverted file.

e.

As requested, we added two more fields to the information we collected while indexing the corpus.

In the dictionary we added the total frequency of every term in the corpus- how many times the term appears in the corpus. We saved this information because we believe we can use this information in the next phase, while retrieving data from the corpus, also, this data helps us answer future questions in the report.

In addition, for each document we saved the length of the document in words- how many words the document contains. We saved this information because we think this can help us normalize the numbers for every document.

f,g.

As requested, we added two additional rules to the parser:

1. <u>weight</u>: Any number, followed by a unit of weight- kg, g, mg, will be added to the dictionary as with that unit

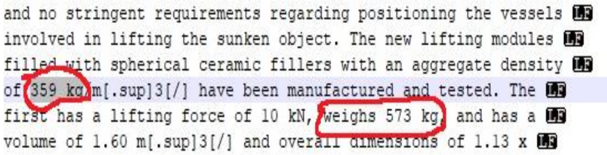
   For example:

Appears in the document	Will be saved in the dictionary as
13.6215 Kg	13.621 kg
20 1/3g	20 1/3 g
14 MG	14 mg

We added this rule because we knew weight is a subject a lot of people interested on and wanted to be able to cover queries on this subject better.

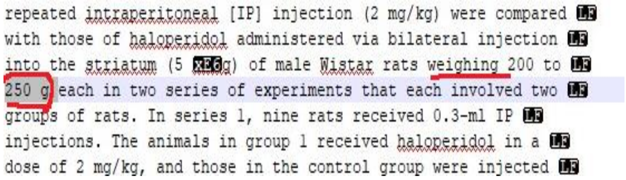
We can see terms from this subject in the corpus:

In doc FBIS4-47361



```
and no stringent requirements regarding positioning the vessels LF
involved in lifting the sunken object. The new lifting modules LF
filled with spherical ceramic fillers with an aggregate density LF
of 359 kg/m[.sup]3[/] have been manufactured and tested. The LF
first has a lifting force of 10 kN, weighs 573 kg, and has a LF
volume of 1.60 m[.sup]3[/] and overall dimensions of 1.13 x LF
```

In doc FBIS4-47446



```
repeated intraperitoneal [IP] injection (2 mg/kg) were compared LF
with those of haloperidol administered via bilateral injection LF
into the striatum (5 xE6g) of male Wistar rats weighing 200 to LF
250 g each in two series of experiments that each involved two LF
groups of rats. In series 1, nine rats received 0.3-ml IP LF
injections. The animals in group 1 received haloperidol in a LF
dose of 2 mg/kg, and those in the control group were injected LF
```

we implemented this rule using a dictionary of the weight units- when we got a number expression, we checked whether the next expression is a in the weight postfixes dictionary, and if it is- the number is a weight expression.

2. distance: Any number, followed by a unit of distance- km, cm, mm, nm will be added to the dictionary as with that unit

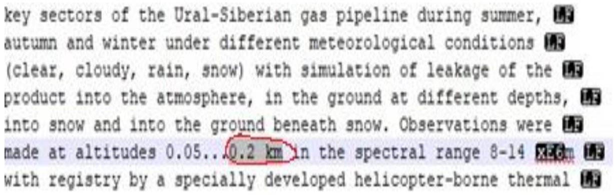
   For example:

Appears in the document	**Will be saved in the dictionary as**
703 Km	703 km
40.3152 cm	40.315 cm
70 1/2 nm	70 1/2 nm

   We added this rule because we knew measuring distances is a subject a lot of people interested on and wanted to be able to cover queries on this subject better.

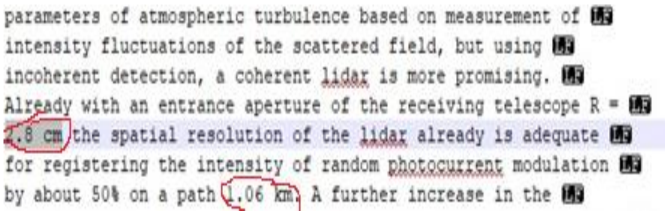
   We can see terms from this subject in the corpus:

   In document FBIS4-47306



```
key sectors of the Ural-Siberian gas pipeline during summer, LF
autumn and winter under different meteorological conditions LF
(clear, cloudy, rain, snow) with simulation of leakage of the LF
product into the atmosphere, in the ground at different depths, LF
into snow and into the ground beneath snow. Observations were LF
made at altitudes 0.05..0.2 km in the spectral range 8-14 XE6m LF
with registry by a specially developed helicopter-borne thermal LF
```

   In document FBIS4-47386



```
parameters of atmospheric turbulence based on measurement of LF
intensity fluctuations of the scattered field, but using LF
incoherent detection, a coherent lidar is more promising. LF
Already with an entrance aperture of the receiving telescope R = LF
2.8 cm the spatial resolution of the lidar already is adequate LF
for registering the intensity of random photocurrent modulation LF
by about 50% on a path 1.06 km. A further increase in the LF
```

   we implemented this rule using a dictionary of the distance units- when we got a number expression, we checked whether the next expression is a in the distance postfixes dictionary, and if it is- the number is a distance expression.

h.

In order to help us, we used on open source code- on the stemming, as suggested on. We downloaded the code for the Porter’s Stemmer and used it in the Stemmer class while parsing the documents to stem the words.

Link to the code we used: https://tartarus.org/martin/PorterStemmer/java.txt

2) Data after the indexing process

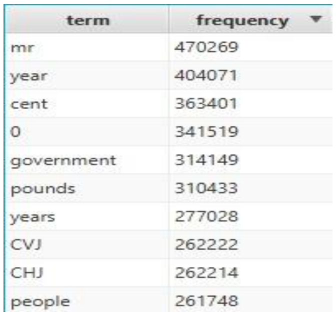
a. Without stemming we ended up with 1,404,418 different terms

b. With stemming we ended up with 1,275,692 different terms.

c. We found 125,601 different numeric terms in the corpus

d. The 10 most common words in the corpus,

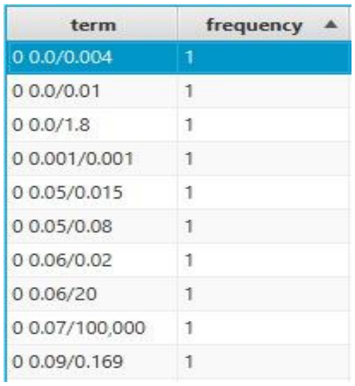
by total frequency are:



term	frequency ▼
mr	470269
year	404071
cent	363401
0	341519
government	314149
pounds	310433
years	277028
CVJ	262222
CHJ	262214
people	261748

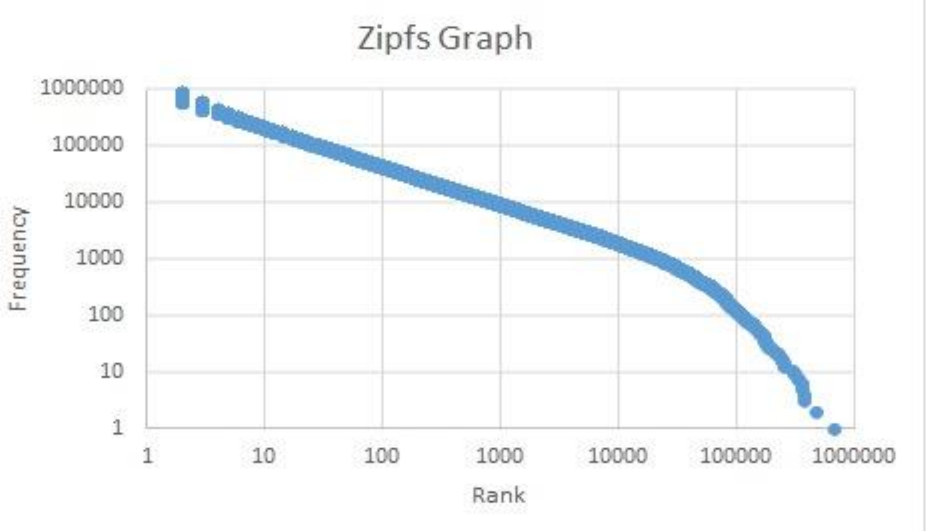
The 10 most uncommon words in the corpus,

by total frequency are:



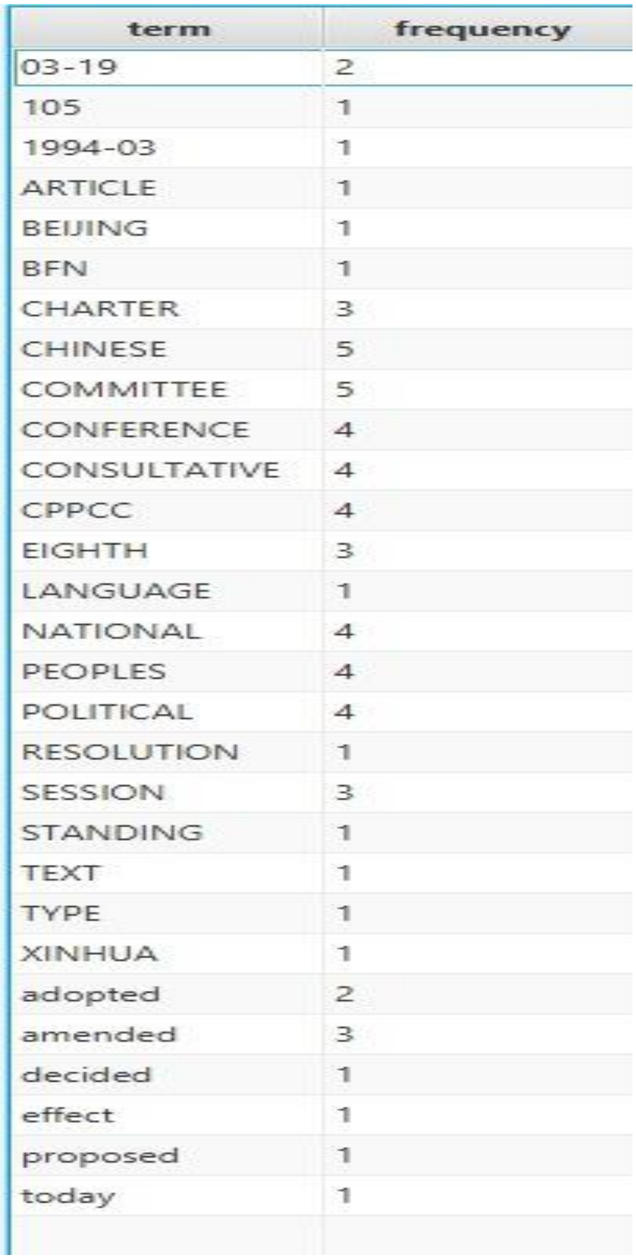
term	frequency ▲
0 0.0/0.004	1
0 0.0/0.01	1
0 0.0/1.8	1
0 0.001/0.001	1
0 0.05/0.015	1
0 0.05/0.08	1
0 0.06/0.02	1
0 0.06/20	1
0 0.07/100,000	1
0 0.09/0.169	1

e. The Zipf's law graph we came up with is:



The graph does looks like the Zipf's law we anticipated to get for a corpus as big and wide as this.

f. Below we can see the list of all the terms for “FBIS3-3366” document, sorted alphabetically:

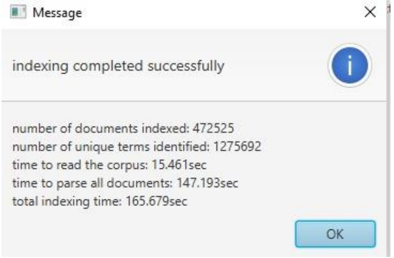


term	frequency
03-19	2
105	1
1994-03	1
ARTICLE	1
BEIJING	1
BFN	1
CHARTER	3
CHINESE	5
COMMITTEE	5
CONFERENCE	4
CONSULTATIVE	4
CPPCC	4
EIGHTH	3
LANGUAGE	1
NATIONAL	4
PEOPLES	4
POLITICAL	4
RESOLUTION	1
SESSION	3
STANDING	1
TEXT	1
TYPE	1
XINHUA	1
adopted	2
amended	3
decided	1
effect	1
proposed	1
today	1

g. The size the posting files require with stemming is 629,513kb

The size the posting files require without stemming is 677,981kb

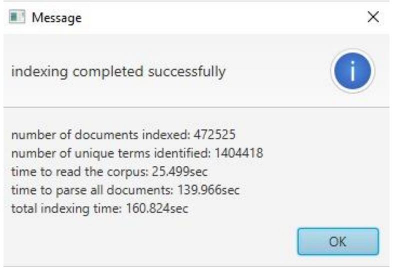
h. The engine took 165.679 seconds to produce the inverted index with stemming:



indexing completed successfully

```
number of documents indexed: 472525
number of unique terms identified: 1275692
time to read the corpus: 15.461sec
time to parse all documents: 147.193sec
total indexing time: 165.679sec
```

The engine took minutes to produce the inverted index without stemming:



indexing completed successfully

```
number of documents indexed: 472525
number of unique terms identified: 1404418
time to read the corpus: 25.499sec
time to parse all documents: 139.966sec
total indexing time: 160.824sec
```

Scobo-Engine report

Part B

c.

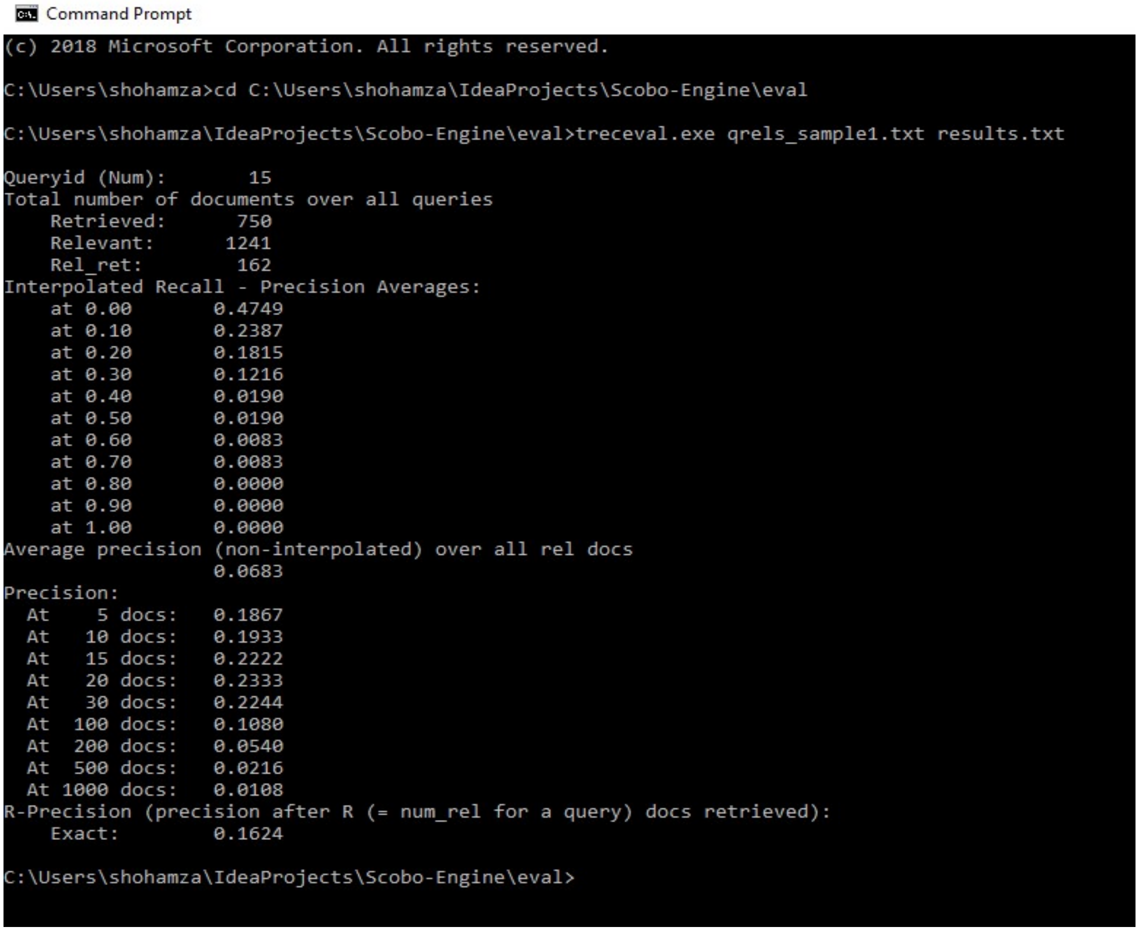
Ranking algorithms:

- **BM25-** in this ranking algorithm we used the BM25 NLP formula for documents ranking. We chose our constants by trying different numbers, until we reached the best results.

  The BM25 function: $f(q,d) = \sum_{w\in q\cap w} c(w,q)\frac{(k+1)c(w,d)}{c(w,d)+k(1-b+b\frac{|b|}{avg(d)}}$

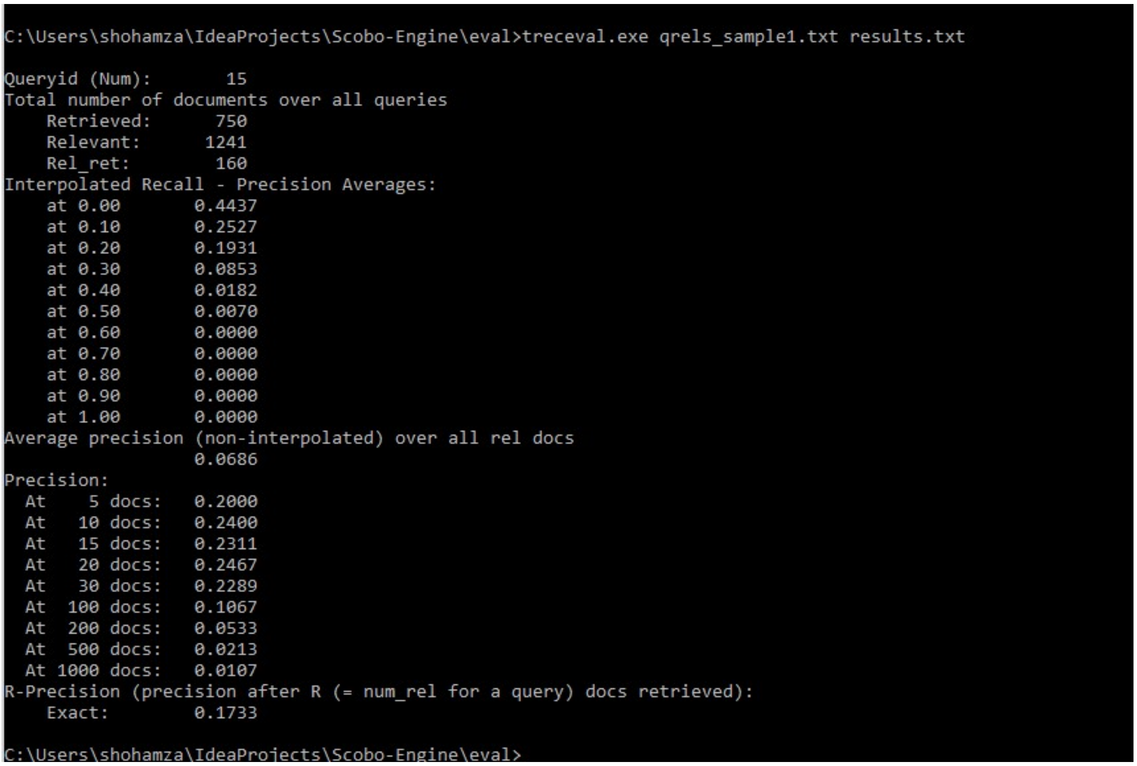
- **Semantic Rank-** in this ranking algorithm we decided to give weight to every part of the query, and for each part calculate its impact, using the BM25 NLP formula for documents ranking. We decided that the entities and the verbal terms in the query will have the most weight on the result- 0.4 each, because they describe the query the most, and the numbers and semantic expansion were given a weight of 0.1 because we believed they say less about the query.

These are the results when running BM25 without stemming



```
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\shohamza>cd C:\Users\shohamza\IdeaProjects\Scobo-Engine\eval

C:\Users\shohamza\IdeaProjects\Scobo-Engine\eval>treceval.exe qrels_sample1.txt results.txt

Queryid (Num):       15
Total number of documents over all queries
    Retrieved:      750
    Relevant:      1241
    Rel_ret:        162
Interpolated Recall - Precision Averages:
    at 0.00       0.4749
    at 0.10       0.2387
    at 0.20       0.1815
    at 0.30       0.1216
    at 0.40       0.0190
    at 0.50       0.0190
    at 0.60       0.0083
    at 0.70       0.0083
    at 0.80       0.0000
    at 0.90       0.0000
    at 1.00       0.0000
Average precision (non-interpolated) over all rel docs
                  0.0683
Precision:
  At    5 docs:   0.1867
  At   10 docs:   0.1933
  At   15 docs:   0.2222
  At   20 docs:   0.2333
  At   30 docs:   0.2244
  At  100 docs:   0.1080
  At  200 docs:   0.0540
  At  500 docs:   0.0216
  At 1000 docs:   0.0108
R-Precision (precision after R (= num_rel for a query) docs retrieved):
    Exact:        0.1624

C:\Users\shohamza\IdeaProjects\Scobo-Engine\eval>
```

these are the results when running Semantic Rank without stemming

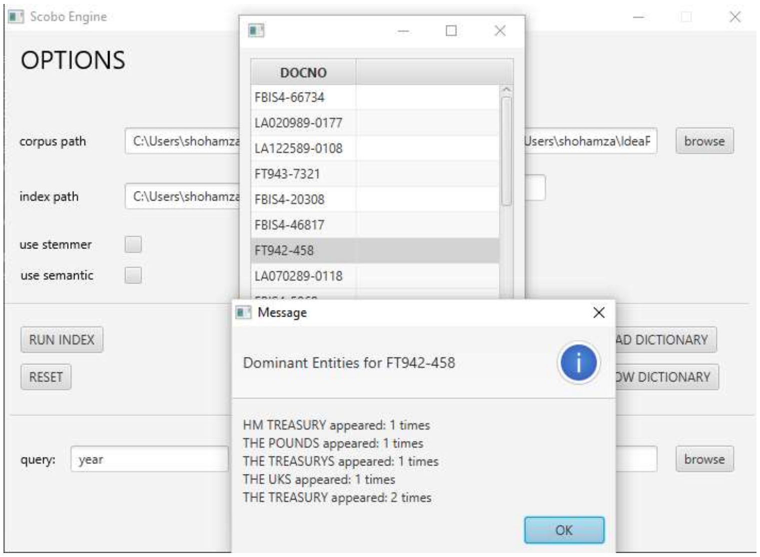


```
C:\Users\shohamza\IdeaProjects\Scobo-Engine\eval>treceval.exe qrels_sample1.txt results.txt

Queryid (Num):       15
Total number of documents over all queries
    Retrieved:      750
    Relevant:      1241
    Rel_ret:        160
Interpolated Recall - Precision Averages:
    at 0.00       0.4437
    at 0.10       0.2527
    at 0.20       0.1931
    at 0.30       0.0853
    at 0.40       0.0182
    at 0.50       0.0070
    at 0.60       0.0000
    at 0.70       0.0000
    at 0.80       0.0000
    at 0.90       0.0000
    at 1.00       0.0000
Average precision (non-interpolated) over all rel docs
                  0.0686
Precision:
  At    5 docs:   0.2000
  At   10 docs:   0.2400
  At   15 docs:   0.2311
  At   20 docs:   0.2467
  At   30 docs:   0.2289
  At  100 docs:   0.1067
  At  200 docs:   0.0533
  At  500 docs:   0.0213
  At 1000 docs:   0.0107
R-Precision (precision after R (= num_rel for a query) docs retrieved):
    Exact:        0.1733

C:\Users\shohamza\IdeaProjects\Scobo-Engine\eval>
```

as we can see although the number if relevant results has decreased by 2 we can see that the Average precision and R-Precision have improved when using the semantic model.

Algorithm to find most significant entities in document:

While parsing each document we go through all the entities, and for each entity give weight according to number of appearances in the document and save the 5 most significant in the documents file.



corpus path C:\Users\shohamza

Users\shohamza\IdeaF

index path C:\Users\shohamza

use stemmer □

use semantic □

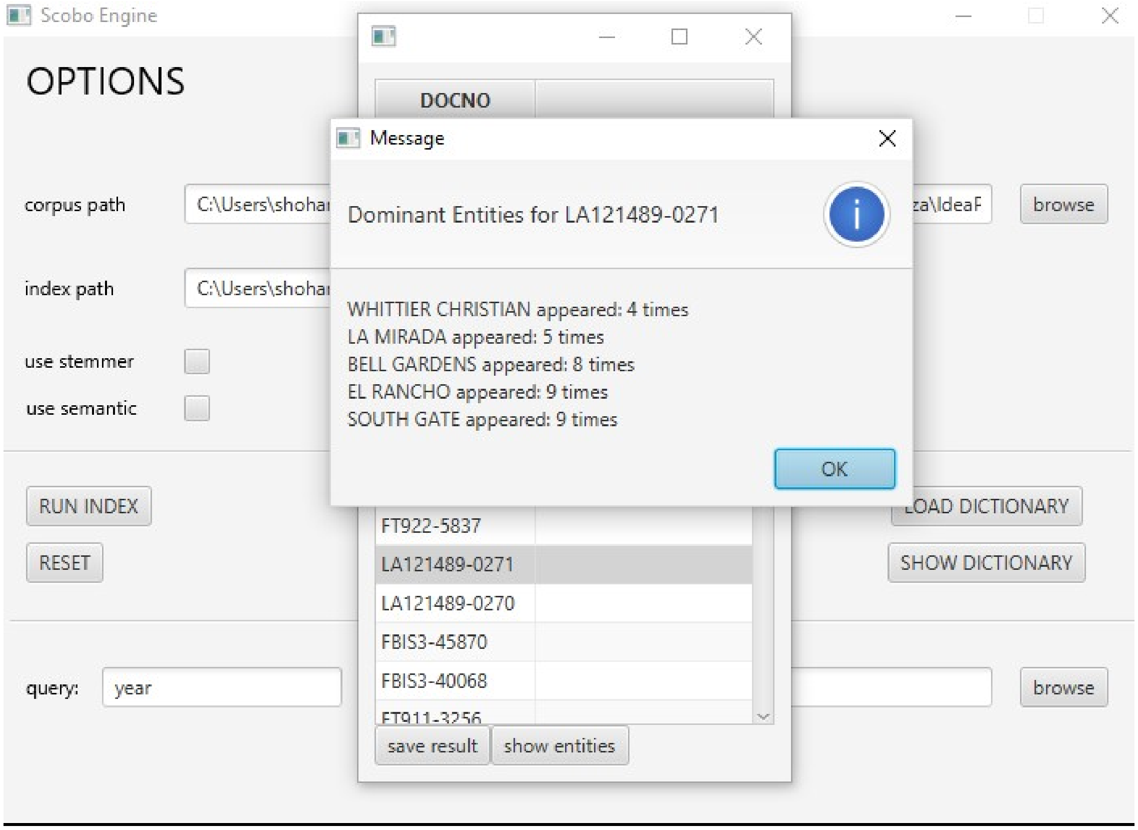
query: year

DOCNO
FBIS4-66734
LA020989-0177
LA122589-0108
FT943-7321
FBIS4-20308
FBIS4-46817
FT942-458
LA070289-0118

Message

Dominant Entities for FT942-458

HM TREASURY appeared: 1 times
THE POUNDS appeared: 1 times
THE TREASURYS appeared: 1 times
THE UKS appeared: 1 times
THE TREASURY appeared: 2 times



corpus path: C:\Users\shohar … za\IdeaF

index path: C:\Users\shohar

use stemmer ☐

use semantic ☐

query: year

```
Dominant Entities for LA121489-0271

WHITTIER CHRISTIAN appeared: 4 times
LA MIRADA appeared: 5 times
BELL GARDENS appeared: 8 times
EL RANCHO appeared: 9 times
SOUTH GATE appeared: 9 times
```

DOCNO
FT922-5837
LA121489-0271
LA121489-0270
FBIS3-45870
FBIS3-40068
FT911-3256

Semantic improvement algorithm:

We used a file containing words in the English language, and 10 words similar to that word. For creating this file, we used files from Glove- the files containing words and vector of numbers. We cleaned the files from words that are not real words, and then run over all the words, to find for each words the 10 most similar words using CosSim formula.
In the indexing part, we clean from this file all the words that don't appear in the dictionary.
While parsing a query, we check for each term if it appears in the semantic file, and if it does, we add two words similar to the term.

d.

In the posting files we saved for each term the frequency and the docID. We used the frequency in the ranking functions, and the docID in order to identify which document we are calculating now.

In the dictionary we saved for each term it’s df- which we used in the ranking functions.

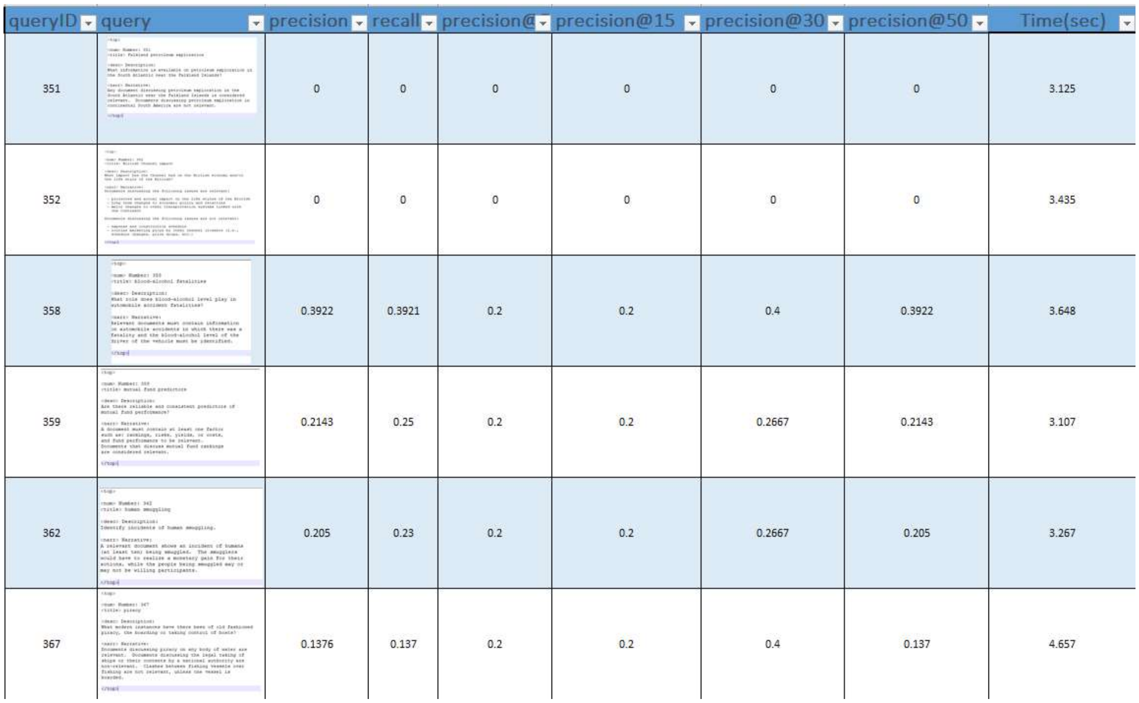
e.

We used the Porter’s Stemmer: https://tartarus.org/martin/PorterStemmer/java.txt

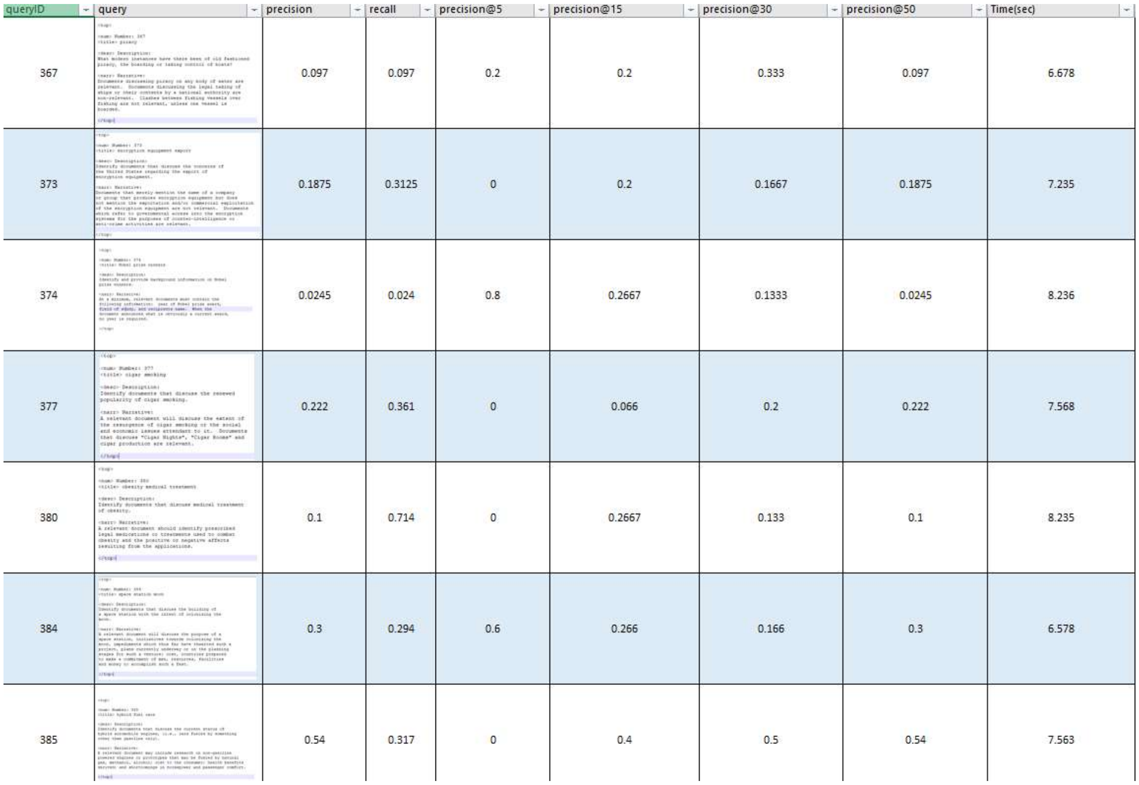
We also used files from GloVe for the Semantic improvement algorithm: https://nlp.stanford.edu/projects/glove/

Engine Evaluation

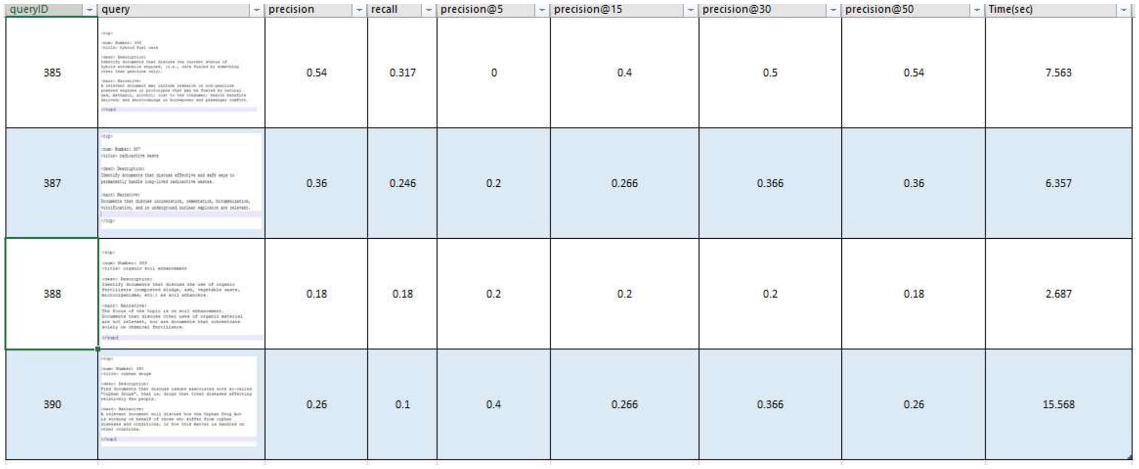
TrecEval output without stemming:



queryID	query	precision	recall	precision@	precision@15	precision@30	precision@50	Time(sec)
351	<top> <num> Number: 351 <title> Falkland petroleum exploration <desc> Description: What information is available on petroleum exploration in the South Atlantic near the Falkland Islands? <narr> Narrative: Any document discussing petroleum exploration in the South Atlantic near the Falkland Islands is considered relevant. Documents discussing petroleum exploration in continental South America are not relevant. </top>	0	0	0	0	0	0	3.125
352	<top> <num> Number: 352 <title> British Chunnel impact <desc> Description: What impact has the Chunnel had on the British economy and/or the life style of the British? <narr> Narrative: Documents discussing the following issues are relevant: - projected and actual impact on the life styles of the British - Long term changes to economic policy and relations - major changes to other transportation systems linked with the Continent Documents discussing the following issues are not relevant: - expense and construction schedule - routine marketing ploys by other channel crossers (i.e., schedule changes, price drops, etc.) </top>	0	0	0	0	0	0	3.435
358	<top> <num> Number: 358 <title> blood-alcohol fatalities <desc> Description: What role does blood-alcohol level play in automobile accident fatalities? <narr> Narrative: Relevant documents must contain information on automobile accidents in which there was a fatality and the blood-alcohol level of the driver of the vehicle must be identified. </top>	0.3922	0.3921	0.2	0.2	0.4	0.3922	3.648
359	<top> <num> Number: 359 <title> mutual fund predictors <desc> Description: Are there reliable and consistent predictors of mutual fund performance? <narr> Narrative: A document must contain at least one factor such as: rankings, risks, yields, or costs, and fund performance to be relevant. Documents that discuss mutual fund rankings are considered relevant. </top>	0.2143	0.25	0.2	0.2	0.2667	0.2143	3.107
362	<top> <num> Number: 362 <title> human smuggling <desc> Description: Identify incidents of human smuggling. <narr> Narrative: A relevant document shows an incident of humans (at least ten) being smuggled. The smugglers would have to realize a monetary gain for their actions, while the people being smuggled may or may not be willing participants. </top>	0.205	0.23	0.2	0.2	0.2667	0.205	3.267
367	<top> <num> Number: 367 <title> piracy <desc> Description: What modern instances have there been of old fashioned piracy, the boarding or taking control of boats? <narr> Narrative: Documents discussing piracy on any body of water are relevant. Documents discussing the legal taking of ships or their contents by a national authority are non-relevant. Clashes between fishing vessels over fishing are not relevant, unless one vessel is boarded. </top>	0.1376	0.137	0.2	0.2	0.4	0.137	4.657



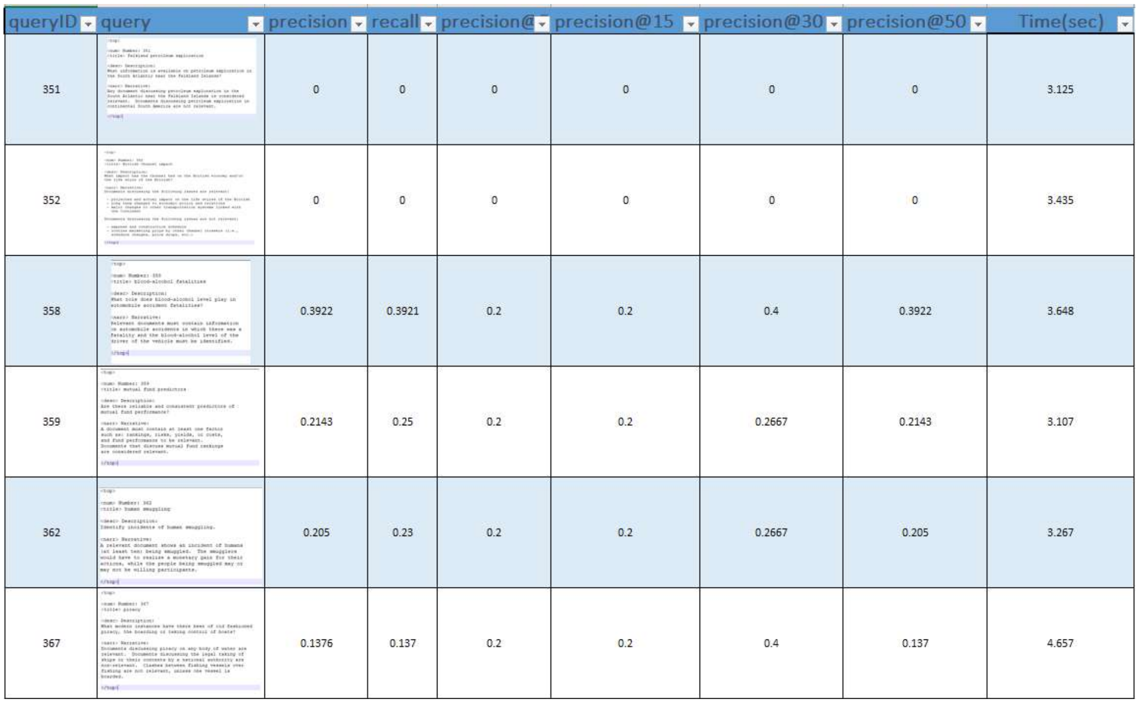
queryID	query	precision	recall	precision@5	precision@15	precision@30	precision@50	Time(sec)
367	<top> <num> Number: 367 <title> piracy <desc> Description: What modern instances have there been of old fashioned piracy, the boarding or taking control of boats? <narr> Narrative: Documents discussing piracy on any body of water are relevant. Documents discussing the legal taking of ships or their contents by a national authority are non-relevant. Clashes between fishing vessels over fishing are not relevant, unless one vessel is boarded. </top>	0.097	0.097	0.2	0.2	0.333	0.097	6.678
373	<top> <num> Number: 373 <title> encryption equipment export <desc> Description: Identify documents that discuss the concerns of the United States regarding the export of encryption equipment. <narr> Narrative: Documents that merely mention the name of a company or group that produces encryption equipment but does not mention the exportation and/or commercial exploitation of the encryption equipment are not relevant. Documents which refer to governmental access into the encryption systems for the purposes of counter-intelligence or anti-crime activities are relevant. </top>	0.1875	0.3125	0	0.2	0.1667	0.1875	7.235
374	<top> <num> Number: 374 <title> Nobel prize winners <desc> Description: Identify and provide background information on Nobel prize winners. <narr> Narrative: At a minimum, relevant documents must contain the following information: year of Nobel prize award, field of study, and recipients name. When the document announces what is obviously a current award, no year is required. </top>	0.0245	0.024	0.8	0.2667	0.1333	0.0245	8.236
377	<top> <num> Number: 377 <title> cigar smoking <desc> Description: Identify documents that discuss the renewed popularity of cigar smoking. <narr> Narrative: A relevant document will discuss the extent of the resurgence of cigar smoking or the social and economic issues attendant to it. Documents that discuss "Cigar Nights", "Cigar Rooms" and cigar production are relevant. </top>	0.222	0.361	0	0.066	0.2	0.222	7.568
380	<top> <num> Number: 380 <title> obesity medical treatment <desc> Description: Identify documents that discuss medical treatment of obesity. <narr> Narrative: A relevant document should identify prescribed legal medications or treatments used to combat obesity and the positive or negative affects resulting from the applications. </top>	0.1	0.714	0	0.2667	0.133	0.1	8.235
384	<top> <num> Number: 384 <title> space station moon <desc> Description: Identify documents that discuss the building of a space station with the intent of colonizing the moon. <narr> Narrative: A relevant document will discuss the purpose of a space station, initiatives towards colonizing the moon, impediments which thus far have thwarted such a project, plans currently underway or in the planning stages for such a venture; cost, countries prepared to make a commitment of men, resources, facilities and money to accomplish such a feat. </top>	0.3	0.294	0.6	0.266	0.166	0.3	6.578
385	<top> <num> Number: 385 <title> hybrid fuel cars <desc> Description: Identify documents that discuss the current status of hybrid automobile engines, (i.e., cars fueled by something other than gasoline only). <narr> Narrative: A relevant document may include research on non-gasoline powered engines or prototypes that may be fueled by natural gas, methanol, alcohol; cost to the consumer; health benefits derived; and shortcomings in horsepower and passenger comfort. </top>	0.54	0.317	0	0.4	0.5	0.54	7.563



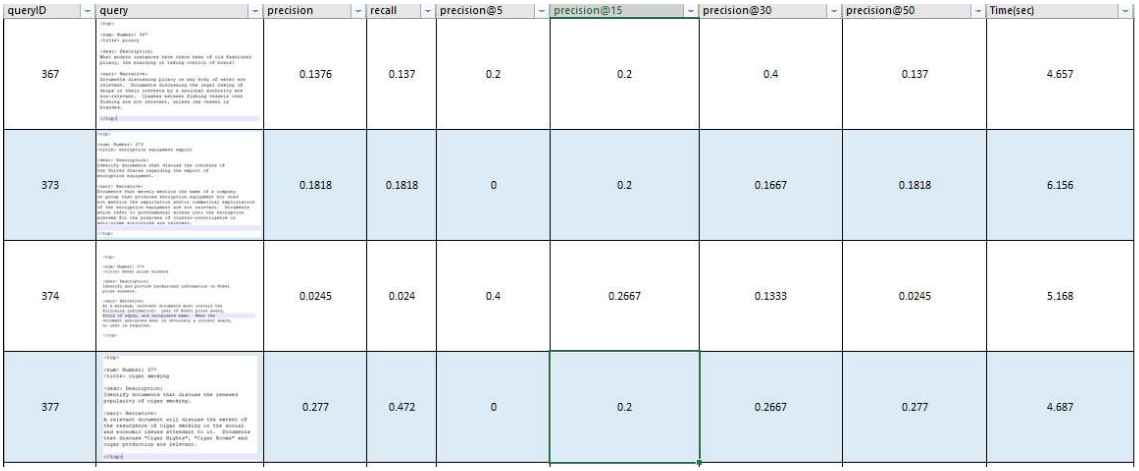
queryID	query	precision	recall	precision@5	precision@15	precision@30	precision@50	Time(sec)
385	<top> <num> Number: 385 <title> hybrid fuel cars <desc> Description: Identify documents that discuss the current status of hybrid automobile engines, (i.e., cars fueled by something other than gasoline only). <narr> Narrative: A relevant document may include research on non-gasoline powered engines or prototypes that may be fueled by natural gas, methanol, alcohol; cost to the consumer; health benefits; efficiency and shortcomings in horsepower and passenger comfort. </top>	0.54	0.317	0	0.4	0.5	0.54	7.563
387	<top> <num> Number: 387 <title> radioactive waste <desc> Description: Identify documents that discuss effective and safe ways to permanently handle long-lived radioactive wastes. <narr> Narrative: Documents that discuss incineration, cementation, bituminization, vitrification, and in underground nuclear explosion are relevant. </top>	0.36	0.246	0.2	0.266	0.366	0.36	6.357
388	<top> <num> Number: 388 <title> organic soil enhancement <desc> Description: Identify documents that discuss the use of organic fertilizers (composted sludge, ash, vegetable waste, microorganisms, etc.) as soil enhancers. <narr> Narrative: The focus of the topic is on soil enhancement. Documents that discuss other uses of organic material are not relevant, nor are documents that concentrate solely on chemical fertilizers. </top>	0.18	0.18	0.2	0.2	0.2	0.18	2.687
390	<top> <num> Number: 390 <title> orphan drugs <desc> Description: Find documents that discuss issues associated with so-called "orphan drugs", that is, drugs that treat diseases affecting relatively few people. <narr> Narrative: A relevant document will discuss how the Orphan Drug Act is working on behalf of those who suffer from orphan diseases and conditions, or how this matter is handled in other countries. </top>	0.26	0.1	0.4	0.266	0.366	0.26	15.568

We got a MAP value of 0.0683

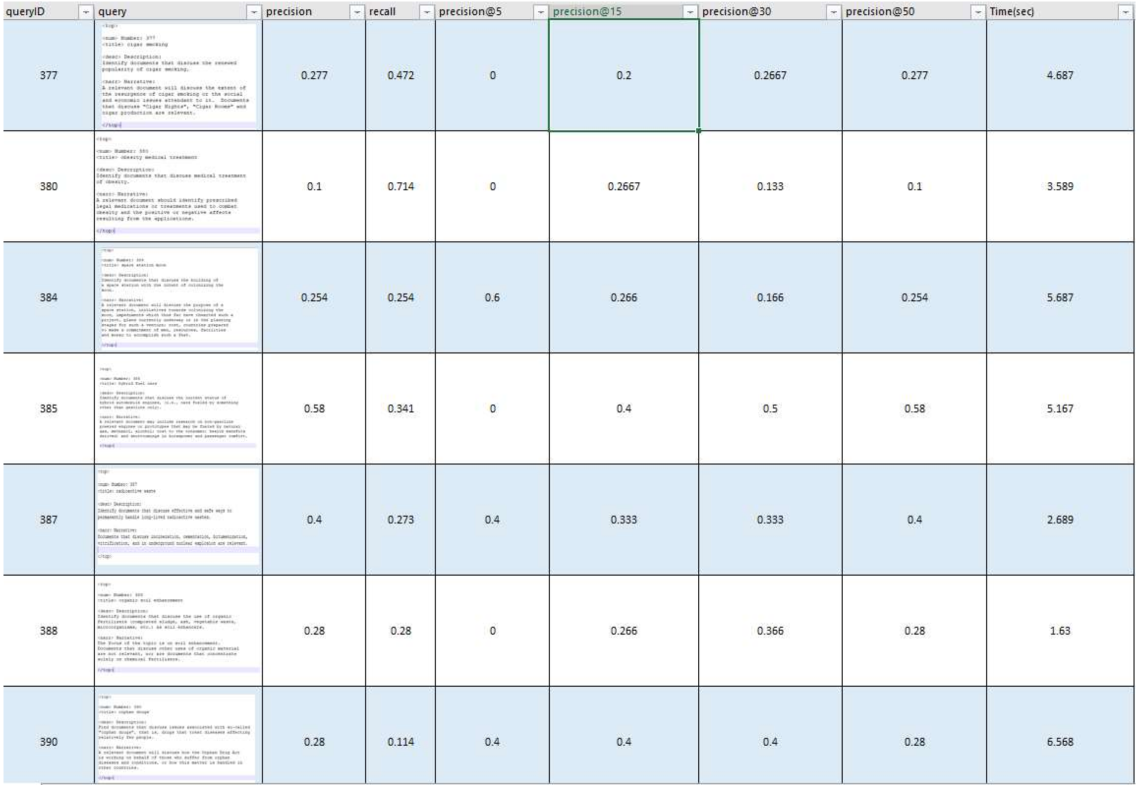
TrecEval Output with stemming:



queryID	query	precision	recall	precision@	precision@15	precision@30	precision@50	Time(sec)
351	<top> <num> Number: 351 <title> Falkland petroleum exploration <desc> Description: What information is available on petroleum exploration in the South Atlantic near the Falkland Islands? <narr> Narrative: Any document discussing petroleum exploration in the South Atlantic near the Falkland Islands is considered relevant. Documents discussing petroleum exploration in continental South America are not relevant. </top>	0	0	0	0	0	0	3.125
352	<top> <num> Number: 352 <title> British Chunnel impact <desc> Description: What impact has the Chunnel had on the British economy and/or the life style of the British? <narr> Narrative: Documents discussing the following issues are relevant: - projected and actual impact on the life styles of the British - Long term changes to economic policy and relations - major changes to other transportation systems linked with the Continent Documents discussing the following issues are not relevant: - expense and construction schedule - routine marketing ploys by other channel crossers (i.e., schedule changes, price drops, etc.) </top>	0	0	0	0	0	0	3.435
358	<top> <num> Number: 358 <title> blood-alcohol fatalities <desc> Description: What role does blood-alcohol level play in automobile accident fatalities? <narr> Narrative: Relevant documents must contain information on automobile accidents in which there was a fatality and the blood-alcohol level of the driver of the vehicle must be identified. </top>	0.3922	0.3921	0.2	0.2	0.4	0.3922	3.648
359	<top> <num> Number: 359 <title> mutual fund predictors <desc> Description: Are there reliable and consistent predictors of mutual fund performance? <narr> Narrative: A document must contain at least one factor such as: rankings, risks, yields, or costs, and fund performance to be relevant. Documents that discuss mutual fund rankings are considered relevant. </top>	0.2143	0.25	0.2	0.2	0.2667	0.2143	3.107
362	<top> <num> Number: 362 <title> human smuggling <desc> Description: Identify incidents of human smuggling. <narr> Narrative: A relevant document shows an incident of humans (at least two) being smuggled. The smugglers would have to realize a monetary gain for their actions, while the people being smuggled may or may not be willing participants. </top>	0.205	0.23	0.2	0.2	0.2667	0.205	3.267
367	<top> <num> Number: 367 <title> piracy <desc> Description: What modern instances have there been of old fashioned piracy, the boarding or taking control of boats? <narr> Narrative: Documents discussing piracy on any body of water are relevant. Documents discussing the legal taking of ships or their contents by a national authority are non-relevant. Clashes between fishing vessels over fishing are not relevant, unless one vessel is boarded. </top>	0.1376	0.137	0.2	0.2	0.4	0.137	4.657



queryID	query	precision	recall	precision@5	precision@15	precision@30	precision@50	Time(sec)
367	<top> <num> Number: 367 <title> piracy <desc> Description: What modern instances have there been of old fashioned piracy, the boarding or taking control of boats? <narr> Narrative: Documents discussing piracy on any body of water are relevant. Documents discussing the legal taking of ships or their contents by a national authority are non-relevant. Clashes between fishing vessels over fishing are not relevant, unless one vessel is boarded. </top>	0.1376	0.137	0.2	0.2	0.4	0.137	4.657
373	<top> <num> Number: 373 <title> encryption equipment export <desc> Description: Identify documents that discuss the concerns of the United States regarding the export of encryption equipment. <narr> Narrative: Documents that merely mention the name of a company or group that produces encryption equipment but does not mention the exportation and/or commercial exploitation of the encryption equipment are not relevant. Documents which refer to governmental access into the encryption systems for the purposes of counter-intelligence or anti-crime activities are relevant. </top>	0.1818	0.1818	0	0.2	0.1667	0.1818	6.156
374	<top> <num> Number: 374 <title> Nobel prize winners <desc> Description: Identify and provide background information on Nobel prize winners. <narr> Narrative: At a minimum, relevant documents must contain the following information: year of Nobel prize award, field of study, and recipients name. When the document announces what is obviously a current award, no year is required. </top>	0.0245	0.024	0.4	0.2667	0.1333	0.0245	5.168
377	<top> <num> Number: 377 <title> cigar smoking <desc> Description: Identify documents that discuss the renewed popularity of cigar smoking. <narr> Narrative: A relevant document will discuss the extent of the resurgence of cigar smoking or the social and economic issues attendant to it. Documents that discuss "Cigar Nights", "Cigar Rooms" and cigar production are relevant. </top>	0.277	0.472	0	0.2	0.2667	0.277	4.687



queryID	query	precision	recall	precision@5	precision@15	precision@30	precision@50	Time(sec)
377	<top> <num> Number: 377 <title> cigar smoking <desc> Description: Identify documents that discuss the renewed popularity of cigar smoking. <narr> Narrative: A relevant document will discuss the extent of the resurgence of cigar smoking or the social and economic issues attendant to it. Documents that discuss "Cigar Nights", "Cigar Rooms" and cigar production are relevant. </top>	0.277	0.472	0	0.2	0.2667	0.277	4.687
380	<top> <num> Number: 380 <title> obesity medical treatment <desc> Description: Identify documents that discuss medical treatment of obesity. <narr> Narrative: A relevant document should identify prescribed legal medications or treatments used to combat obesity and the positive or negative affects resulting from the applications. </top>	0.1	0.714	0	0.2667	0.133	0.1	3.589
384	<top> <num> Number: 384 <title> space station moon <desc> Description: Identify documents that discuss the building of a space station with the intent of colonizing the moon. <narr> Narrative: A relevant document will discuss the purpose of a space station, initiatives towards colonizing the moon, impediments which thus far have thwarted such a project, plans currently underway or in the planning stages for such a venture; cost, countries prepared to make a commitment of men, resources, facilities and money to accomplish such a feat. </top>	0.254	0.254	0.6	0.266	0.166	0.254	5.687
385	<top> <num> Number: 385 <title> hybrid fuel cars <desc> Description: Identify documents that discuss the current status of hybrid automobile engines, i.e., cars fueled by something other than gasoline only. <narr> Narrative: A relevant document may include research on non-gasoline powered engines or prototypes that may be fueled by natural gas, methanol, alcohol; cost to the consumer; health benefits derived; and shortcomings in horsepower and passenger comfort. </top>	0.58	0.341	0	0.4	0.5	0.58	5.167
387	<top> <num> Number: 387 <title> radioactive waste <desc> Description: Identify documents that discuss effective and safe ways to permanently handle long-lived radioactive wastes. <narr> Narrative: Documents that discuss incineration, cementation, bituminization, vitrification, and an underground nuclear explosion are relevant. </top>	0.4	0.273	0.4	0.333	0.333	0.4	2.689
388	<top> <num> Number: 388 <title> organic soil enhancement <desc> Description: Identify documents that discuss the use of organic fertilizers (composted sludge, ash, vegetable waste, microorganisms, etc.) as soil enhancers. <narr> Narrative: The focus of the topic is on soil enhancement. Documents that discuss other uses of organic material are not relevant, nor are documents that concentrate solely on chemical fertilizers. </top>	0.28	0.28	0	0.266	0.366	0.28	1.63
390	<top> <num> Number: 390 <title> orphan drugs <desc> Description: Find documents that discuss issues associated with so-called "orphan drugs", that is, drugs that treat diseases affecting relatively few people. <narr> Narrative: A relevant document will discuss how the Orphan Drug Act is working on behalf of those who suffer from orphan diseases and conditions, or how this matter is handled in other countries. </top>	0.28	0.114	0.4	0.4	0.4	0.28	6.568

We got a MAP value of 0.0794.

Summery and Reflection

Some problems we faced in this project:

- Not being able to load the entire corpus into memory, we had to create some sort of cache so we will be able to work on the maximal subset of the corpus we could load to memory at any time, this resulted in creating the custom TaskManager and TaskGroups to make synchronization easier and the two phase indexing process where we first save a partially indexed files to the disk and then merge them.
- At the beginning our idea for indexing was to make many small pasting files and constantly update them in order to avoid the merge phase at the end of the indexing, this turned out to be a bad idea because there would be some posting files where the terms are much more frequent then other and thus we had to write to a small number of files constently which slowed down the indexing significantly, at the end we switched to the two phase process, it turned out that the merge is not very time consuming as it usually takes about 10-30 seconds.
- When we began working on the Querying we needed to use the Parser in order to parse the queries, but the parser was dependent on the indexer and designed to specifically for the indexing, we had to change the dependency of the parser on ReadFile and the Indexer to the interfaces DocumentProvider and Parser.Consumer so that it could run well with the QueryProcessor

If given the chance we would probably work much harder on the parser and the indexer, because it had significant effect on the query results and it was very hard to trace the source of why some results are very good or very bad.