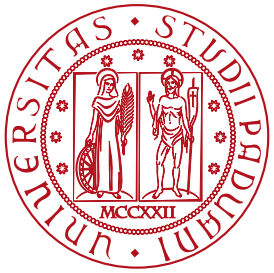

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

A visual analytics tool for the incremental evaluation of search engines

Laureando:

Fabio Giachelle

Relatore:

Prof. Gianmaria Silvello

18 Febbraio 2019

ANNO ACCADEMICO 2018/2019

Ai miei nonni, che mi hanno insegnato quali sono i valori per i quali vale la pena lottare.

Fabio Giachelle

ABSTRACT

In the field of information retrieval (IR), it is of fundamental importance the activity of evaluating the performance of an information retrieval system (IRS), by means of specific metrics. To consistently assess the performance of an IR system, so that it can be compared with other systems, it is necessary to choose a document collection as a reference, for all the IR systems considered. Since a collection can be composed of millions of documents, the indexing process can sometimes take a long time to index the entire collection. However, it is possible to index the collection incrementally to several index percentages, so that for each of them we can calculate the values for the different evaluation metrics chosen, without having to wait for the entire collection to be indexed. The values of the evaluation measures obtained at the various index percentages, allow outlining the performance trend of the retrieval models considered, while the collection of interest is indexed. In this way, we can collect useful information about the performances of the retrieval models as the indexing process proceeds, without having to wait for the full indexed document collection. For this reason, we developed AVIATOR, a software capable of incrementally indexing a document collection in order to help IR experts to automatically obtain the values for the evaluation metrics, so that they can be compared dynamically as the indexing proceeds. To facilitate the evaluation process, AVIATOR provides a web interface that allows experts to control each stage of the entire process with interactive charts and many other visual components. AVIATOR allows you to visually compare the performances of different retrieval models, for each combination of stoplist and stemmer, as the indexing process advances automatically in the background. In this way, it is possible to determine which system is the best at each index percentage, according to the several evaluation metrics chosen. In the version implemented in this thesis, Aviator has been tested on volumes 4 and 5 of the TIPSTER collection, for a total of over 500 000 indexed documents. Afterwards, the final validation phase shows that Aviator allows to easily achieve the same results known in the literature, by means of a simple and intuitive interface.

Sommario

Nel settore dell'information retrieval (IR), valutare le performance di un sistema di reperimento dell'informazione (IRS) è un'attività di fondamentale importanza, che richiede l'utilizzo di apposite misure di valutazione. Queste misure consentono in particolare di confrontare le performance di diversi modelli di reperimento dell'informazione, al fine di stabilire quali di essi risulta il migliore per la collezione di documenti considerata. Poiché la collezione può essere composta anche da milioni di documenti, il processo di indicizzazione può richiedere molto tempo prima di poter essere completato. Tuttavia, è possibile indicizzare la collezione in modo incrementale a varie percentuali (cut-off), così facendo per ogni cut-off è possibile calcolare i valori per le diverse metriche di valutazione, senza dover attendere che l'intera collezione sia indicizzata. I valori ottenuti per le misure di valutazione a vari cut-off, consentono di delineare l'andamento delle performance dei modelli di reperimento man mano che la collezione di interesse viene indicizzata. In questa tesi si propone AVIATOR: uno strumento di visual analytics in grado di indicizzare automaticamente, in modo incrementale, una data collezione di documenti, al fine di facilitare il lavoro di un team di esperti di IR nell'attività di valutazione di modelli e sistemi di reperimento dell'informazione. A tal fine, AVIATOR fornisce un'interfaccia web attraverso la quale è possibile controllare i risultati relativi alle metriche considerate progressivamente, attraverso grafici dinamici e interattivi. Inoltre, AVIATOR permette di confrontare visivamente le performance di differenti modelli di reperimento, per ogni combinazione di stoplist e stemmer considerata, man mano che il processo di indicizzazione avanza automaticamente in background. In questo modo è possibile determinare quale sistema risulta il migliore ad ogni cut-off, secondo un insieme di metriche prestabilito. Per la versione implementata in questa tesi, AVIATOR è stato testato sui volumi 4 e 5 della collezione TIPSTER, per un totale di oltre 500.000 documenti indicizzati. Infine, la fase finale di test e valutazione ha evidenziato che AVIATOR permette di ottenere i risultati noti in letteratura, attraverso un'interfaccia semplice e intuitiva.

CONTENTS

LIST OF THE FIGURES

FIGURE **Page**

LIST OF THE TABLES

CHAPTER 1

INTRODUCTION

Evaluating an IR system is essential to establish the quality of the results returned, analyse its behaviour and finally improve the effectiveness of the system for satisfying the expectations of the user. To compare the performances obtained by different IR systems, it is necessary to adopt standard and shared collections of documents and, at the same time, a specific set of evaluation metrics. In this way, the reproducibility of the experimental results is possible. In this context, due to the large number of measures and parameters to be considered, simplifying and speeding up the analysis process becomes of primary importance, as well as automating all the activities that do not require the direct supervision of a team of experts. The evaluation phase of one or more retrieval models requires that the considered collection has been previously indexed. This is necessary for querying the IRS and obtain the run files to be evaluated, using the chosen metrics and according to the related pool or ground truth.

1.1 The problem

When the given document collection is a very large collection (VLC), the indexing process can take a really long time to complete. For example, on a Dell Latitude E6230, indexing the document corpus of the TIPSTER collection requires at about three hours for over 528 000 documents. This means that we can not get any information regarding the performances of a set of retrieval models on the full collection, without waiting until the end of the indexing process.

1.2 Purpose of the thesis

Indexing a document collection is a very time-consuming task, specially when dealing with VLC. Besides, experiments in IR usually consists of many tests that require reindexing the document collection. In this context, performing dozens of tests on the whole indexed collection would take a long time, without potentially obtaining the desired results. In order to save time, we need to know how the experiments are going, while the indexing process incrementally proceeds. In this way, if the intermediate results of an experiment do not meet the expectations, it can be stopped in advance, without having to index the entire collection. For this reason, the purpose of this thesis is developing a tool that can help IR experts in the activity of evaluating one or more IRS, in a time efficient way, through a progressive visual analytics web interface.

1.3 State of the art

Unfortunately, there are no solutions known in the literature to solve the problem as it stands. Anyway, according to what reported in [1], if we consider some subsets obtained by uniformly sampling the given document collection, we have that, regardless of the topic, the expected probability of a document of being relevant in a sample is the same as in the full collection. Therefore, if in the sample a certain relation between the values of the evaluation measures is observed, for the different retrieval models, this relation can be projected on the whole collection with a percentage of reliability that is proportional to the size of the sample considered.

1.4 Proposed solution

To solve the problem, we can exploit the previous property and calculate the values of the evaluation measures for each version of the document collection's incremental index. In other words, the collection is incrementally indexed at various cut-offs and, for each of them, the evaluation measures are calculated. The values obtained for the evaluation measures at the various cut-offs, allow outlining the performance trend of the several IR models while the collection is being indexed, without having to wait for its full indexing. This strategy is the basis of the work performed by AVIATOR.

1.4.1 Aviator

In this thesis, we present AVIATOR, an all-in-one visual analytics tool for the evaluation of IRS. AVIATOR, through the combined use of an opensource framework for IR and a data visualisation library, allows the exploration of the evaluation data produced considering all the possible combinations generated by 4 stoplists, 4 stemmers and 4 models, for a total of 64 different combinations. All the 64 combinations are evaluated with over 20 different metrics. The document collection chosen includes as a corpus the volumes 4 and 5 of the TIPSTER, for a total of over 500.000 documents, while the set of 50 topics chosen comes from TREC7 and consider the topics from 351 to 400. The amount of data considered would require a phase of exploration too long and complicated to be carried out by hand, for this reason, AVIATOR allows you to automate the entire process of indexing, retrieval and evaluation thus simplifying the work of IR experts.

1.4.1.1 Backend

AVIATOR can incrementally index a given document collection at various cut-offs. It can also preprocess the collection to produce a set of buckets in which are arranged the documents uniformly sampled from the document collection. Besides, AVIATOR provides an exhaustive set of API to control each phase of the IR workflow: indexing, retrieval and evaluation. In addition, AVIATOR implements a REST server with which you can manage each IR task using any web client.

1.4.1.2 Frontend

To help IR experts in evaluating IRS, a user can use the web interface to analyse how the values, for the evaluation metrics, change during the incremental indexing process. For this purpose, AVIATOR provides useful dynamic charts that allow inspecting, both in general and in detail, all the information related to the evaluation phase.

1.4.1.3 Features

We can summarise the features provided by AVIATOR as follows:

- Preprocessing of the document collection, to produce a set of buckets made up of uniformly sampled documents.
- Incremental indexing of the document collection at various cut-off.
- Automatic IR workflow: indexing, retrieval, evaluation.
- A progressive visual analytics platform, for the dynamic exploration of the data regarding the evaluation measures.
- A controller for the Solr open source search platform.
- A controller for the Trec Eval evaluation tool.
- A REST web server which provides an exhaustive set of API to control each phase of the IR workflow.

From the visual analytics web interface, you can choose:

- The corpus of the document collection.
- The topic file.
- The pool file.
- The stoplist.
- The stemming algorithm, i.e. the stemmer.
- The retrieval model.

You can choose the value for all these options, in the web interface shown in Figure 1.1.

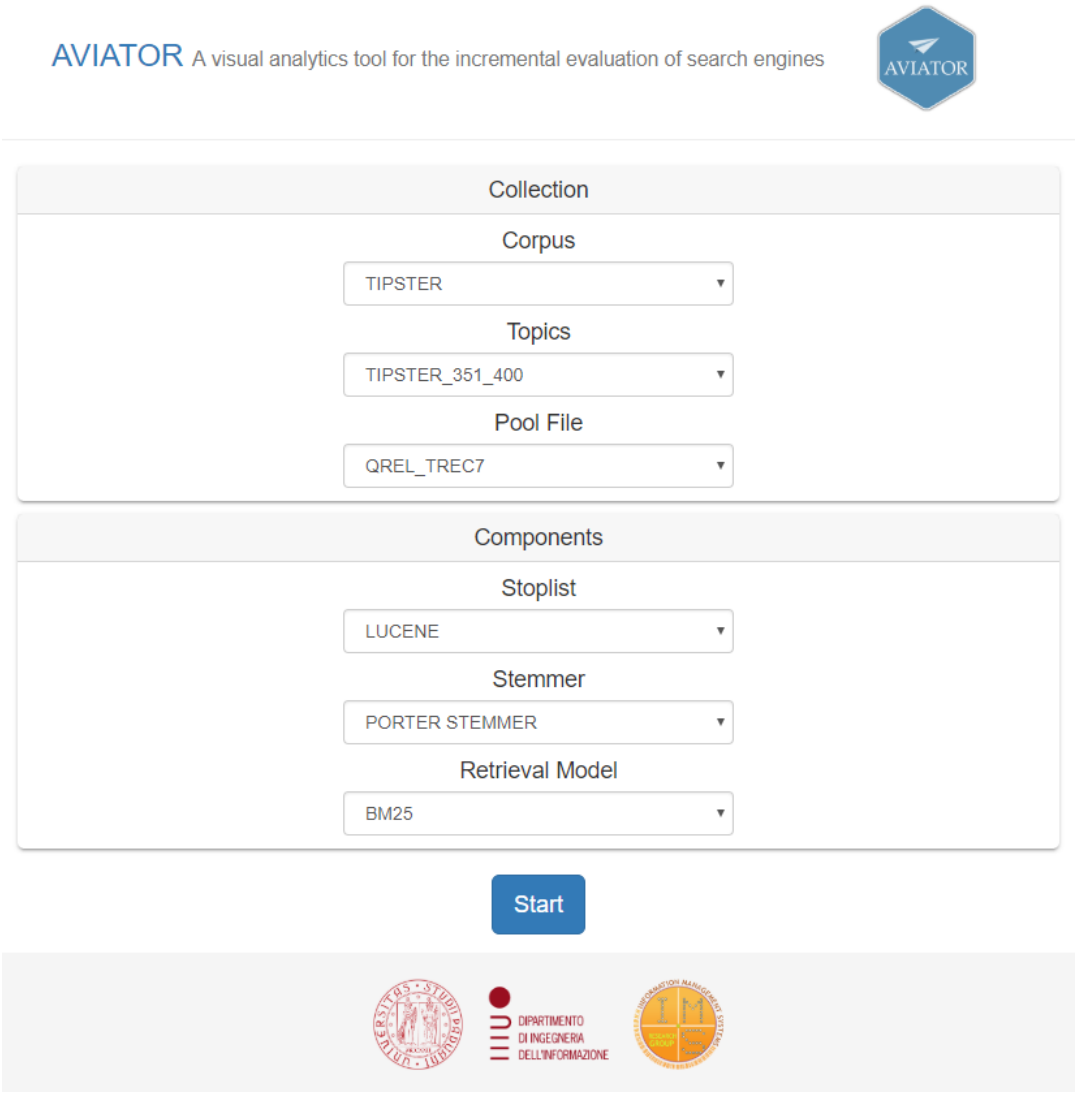

Figure 1.1: Example of the AVIATOR web interface.

1.5 Outline

The contents of this thesis are structured as follows:

- **Background** [2]: in this chapter, we explain the IR fundamentals and all the background notions, that are necessary to understand the whole work developed in this thesis. In particular, we describe the IR workflow: indexing, retrieval and evaluation process, with a focus on the incremental indexing process. Besides, we provide knowl-

edge related to the field of data visualisation, and then we show how to employ it to the IR field.

- **Conceptual Framework** [3]: in this chapter, we explain the specific details regarding how the proposed software solution works. In other words, we describe, at a conceptual level, the algorithm strategy and the methodology used to create the time efficient solution for the discussed problem.

- **Backend** [4]: in this chapter, we explain how we implemented the conceptual framework for the backend part, in terms of technologies used and provided functionalities. Besides, we show how to implement the incremental indexing using the Solr open source search platform. Finally, we show how to implement each phase of the IR workflow, as a pipeline of synchronized tasks.

- **Frontend** [5]: in this chapter, we explain how we implemented the Aviator front-end, in terms of technologies used and provided functionalities. Besides, we show how to use the AVIATOR visual analytics web interface for evaluating IR systems. In particular, we show how to perform the exploration of the evaluation data, by means of interactive charts developed using technologies like JavaScript and the D3.js library.

- **Experimental evaluation** [6]: in this chapter, we present data and results from tests done with AVIATOR on the TIPSTER corpus. Moreover, we compare the evaluation results to identify the best retrieval model, on average, and to quantify the performance differences between the various IR models.

- **Conclusions** [7]: in this chapter, we summarize all the work done to solve the initial problem, discussing the obtained results and the possible future developments.

CHAPTER 2

BACKGROUND

2.1 Information Retrieval

2.1.1 Introduction

Nowadays, the information retrieval (IR) field is often associated with web search engines, but actually, its history is much longer than the Internet one. Indeed, as reported in [2], the studies effectuated by information retrieval researchers started in the 1950s, and the first IR systems were found in commercial applications since the 1960s. Anyway, the main purpose of the information retrieval field has never changed: retrieve the information that satisfies the user information needs. This is what an information retrieval system (IRS) does: it takes the information need of a user, expressed as a query, and returns documents, hopefully relevant, to satisfy the information need inferred by the query. In IR the information retrieved for an input query comes from an unstructured source, which typically is a collection of unstructured data like web pages, documents, images, videos, etc. This is a significant difference from the database context, in which the data are structured. This is the reason why IR has become more and more popular in the last two decades, with the spreading of knowledge allowed by the Internet. The "infinite" knowledge, stored in the billions of web pages saved in servers distributed worldwide and interconnected by the Internet, would not be accessible in a time efficient way without search engines. They have become so essential for satisfying the information needs of billions of users, that nowadays search engines are considered the primary example of powerful IR systems. As any other

IRS, the purpose of web search engines is to locate the relevant information that satisfy the user queries. These usually are made up of a few keywords, while the related results consist of billions of pages, that are presented to the user as a list sorted according to a specific criterion of relevance. This one considers not only the content of the page but also the authoritativeness of the origin website. The increasing usage of IRS in everyday life motivates the last decade challenge, of the most famous web search engines, in improving the quality of the search service in terms of efficiency and effectiveness. This, accompanied by the development of additional but closely related services such as email and cloud, is what made the fortune of brilliant startups, e.g. Google, that now has become one of the most successful companies in the world.

2.1.2 Concepts and definitions

As we said, an information retrieval system (IRS) takes the information needs of users, expressed as queries, and retrieves for them the relevant judged documents for the given queries. Now we want to define accurately, the specific meaning of each concept as follows:

- **Information need**: lack of information that a user wants to fill to solve his problem and then make decisions.
- **Document**: intended as any form of manifestation of information, a document is any type of object, that contains useful information to satisfy the user information need.
- **Relevance**: the property of a document to contain useful information, that are necessary to satisfy the user information need.

Since a document is any type of object in which an IRS can look for useful information, we have to consider as a document, in addition to simple text documents, also mixed content documents like web pages and multimedia objects in general such as images, sounds and videos. In such a various scenario of many different unstructured sources of knowledge, it is of fundamental importance using a stable architecture framework. For this reason, since the beginning of the researches in the IR field, the experts have worked to develop an effective architecture shared by every IRS. A typical IRS architecture is shown in Figure 2.1, where we can see that a general IRS takes in input a query and a set of documents, in which carry out the search task. This set of documents constitutes the corpus of the document collection. Since queries and documents are intended to be any general container of information, the IRS takes them and converts, both the queries and the documents, into the proper

representation respectively. Once the conversion is done, the IRS can compare queries and documents using a similarity evaluation criterion. This one is of fundamental importance because it is used by the retrieval model, to produce the sorted list of results for the given query. The sorted list of results returned is composed of the subset of relevant judged documents, retrieved from the initial corpus of the considered document collection. Finally, the user can assess the quality of the relevant judged documents, to establish if they are, or not, really relevant for the submitted query. This relevance feedback mechanism is very important to prove the correctness of the returned results and to improve the performance of the retrieval model algorithm.

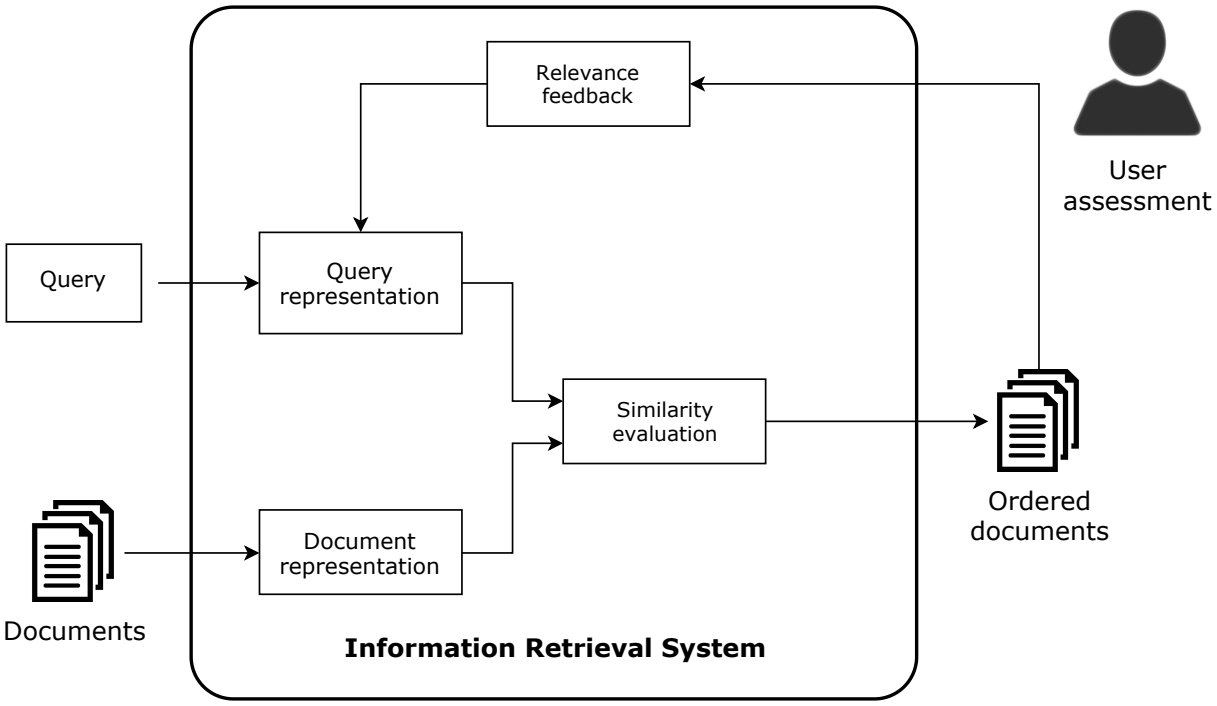

Figure 2.1: A typical architecture for an information retrieval system.

We can define the **retrieval model** as the set of constructs, formalised to make possible:

- The representation of the document's content.
- The query representation.
- The development and implementation of the retrieval algorithms.

As shown in Figure 2.2, the activities of converting queries and documents into the proper representation, are parts of the indexing process done by the information retrieval system. Besides, the activity of evaluating the similarity between a given query and the set of documents, from the document collection, is a task that regards instead the retrieval process. Both indexing and retrieval, are central phases included in the workflow of every IRS.

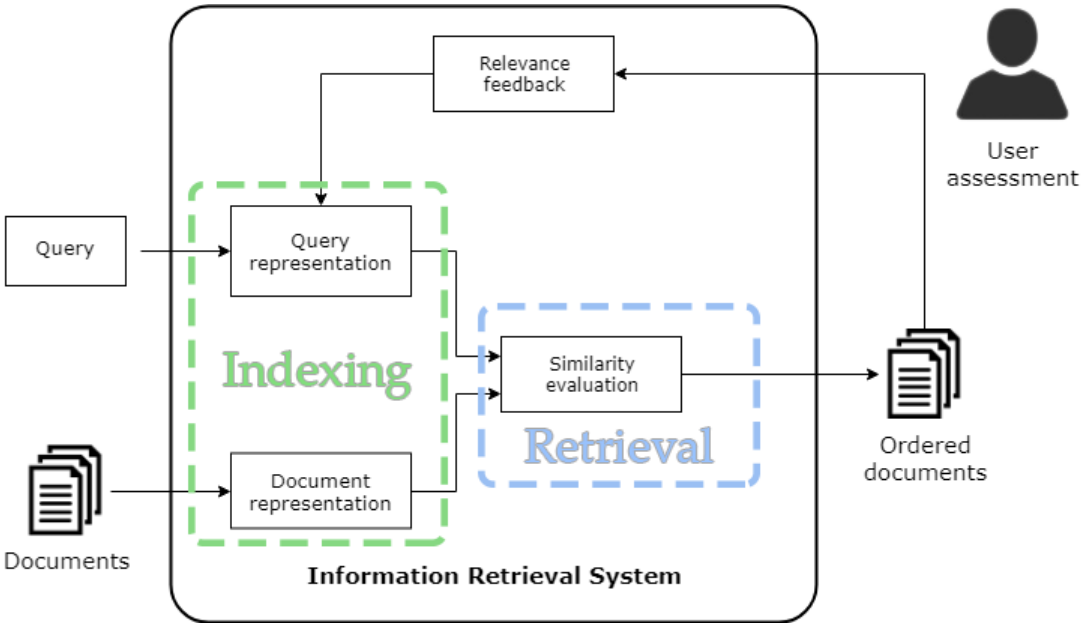

Figure 2.2: Indexing and retrieval phases in a typical architecture for an IRS.

The main purpose of the indexing process is to create the index, also known as the inverted index, for the document collection considered. The principal base function of the inverted index consists of making a link between terms and documents that contain them. The functioning principle of an inverted index is similar to what does an analytical index for a book: it tells the reader the pages where he can find the word, or the concept, for which is looking for. In the same way, the inverted index tells the IRS what documents contain the term considered or, in other words, indicates where the IRS can find the term in the document collection. Since the dimension of the inverted index is often comparable with the document collection size, it is necessary to adopt proper data structures to allow the efficient management of the queries.

We said that information need is the lack of information that a user wants to fill to solve his problem and then make decisions. Another way to express the same concept, in brief, is through the word **topic**. This one is used mainly in the evaluation contest, it represents the information need of a user, expressed by means of a query. A general user query is composed of one or more terms. We use the keyword **term** instead of word because a term can also be, for example, an image, a sound or a multimedia object in general. In Figure 2.3 is reported the information retrieval "U" scheme, in which is graphically described the role of each component in the workflow related to the IR architecture. In the centre of Figure 2.3, we can observe the block *Comparison*, which evaluates the similarity between a query and each document from the related document collection. This is possible thanks to a similarity criterion, which is the key for a good ranking of the returned set of relevant judged documents.

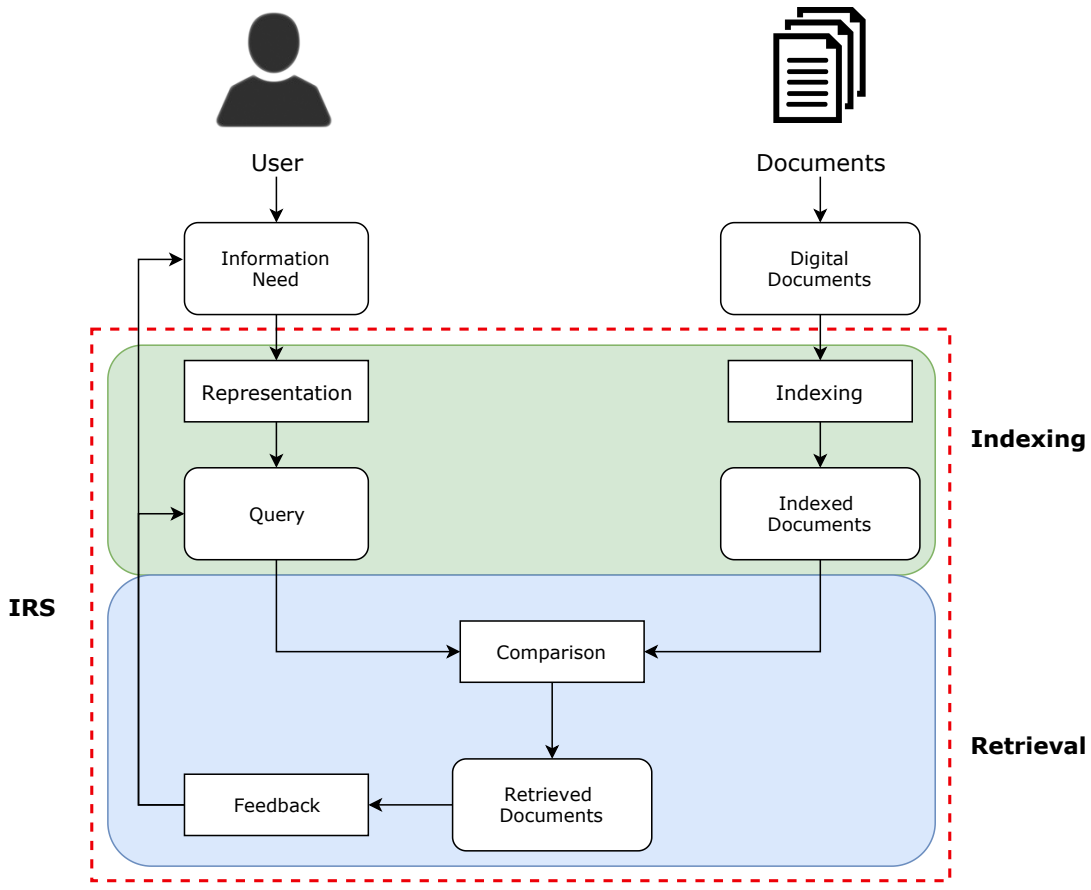

Figure 2.3: IR "U" scheme.

2.2 Indexing

As reported in [3], the purpose of the indexing process is to represent the document's information so that it can be accessed in an efficient fast way. To this end, the indexing process takes each document from a collection and creates a synthetic representation of its content, by means of keywords and terms. These terms are summary descriptors of the information content of each document and, for this reason, are used to build the index structure. This one, also known as the inverted index, indicates, for each index term, the documents that contain them. The inverted index is the most frequent type of index, implemented by information retrieval systems. The word "inverted" in the name suggests the fact that the index is designed to return the inverted list, or posting list, for each index term. The posting list is made up of the list of documents containing that index term. In Figure 2.4 are shown the high-level "building blocks" of the indexing process. These building blocks are:

- **Text Acquisition**: the purpose of this phase is to identify the documents of interest and make them available to be searched later. In simple cases, this consist of using an existing collection. Sometimes, instead, the text acquisition task involves crawling the Web to build the collection that will be later indexed.
- **Text Transformation**: at this stage, the input documents are transformed into index terms.
- **Index Creation**: once all the index terms are obtained, the index is created.

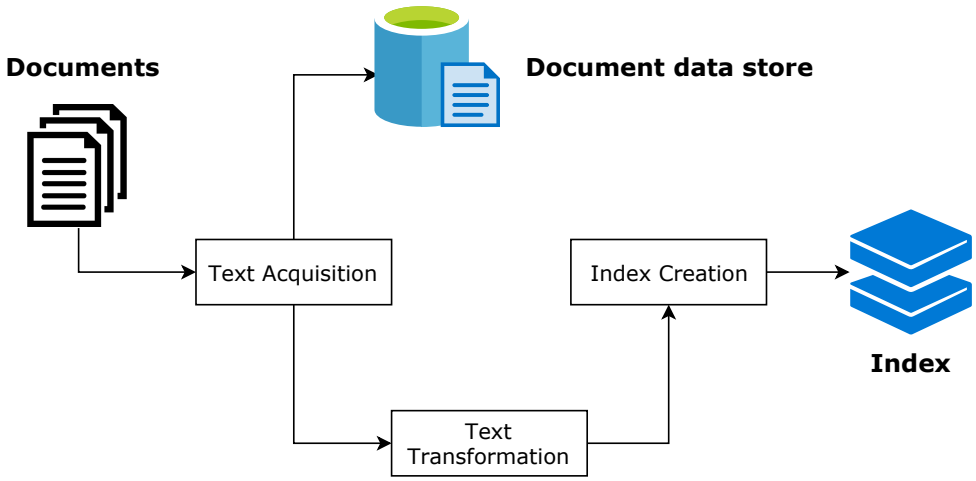

Figure 2.4: Indexing process overview.

In Figure 2.5 are reported, with a greater level of detail, the most important phases that concern the indexing process.

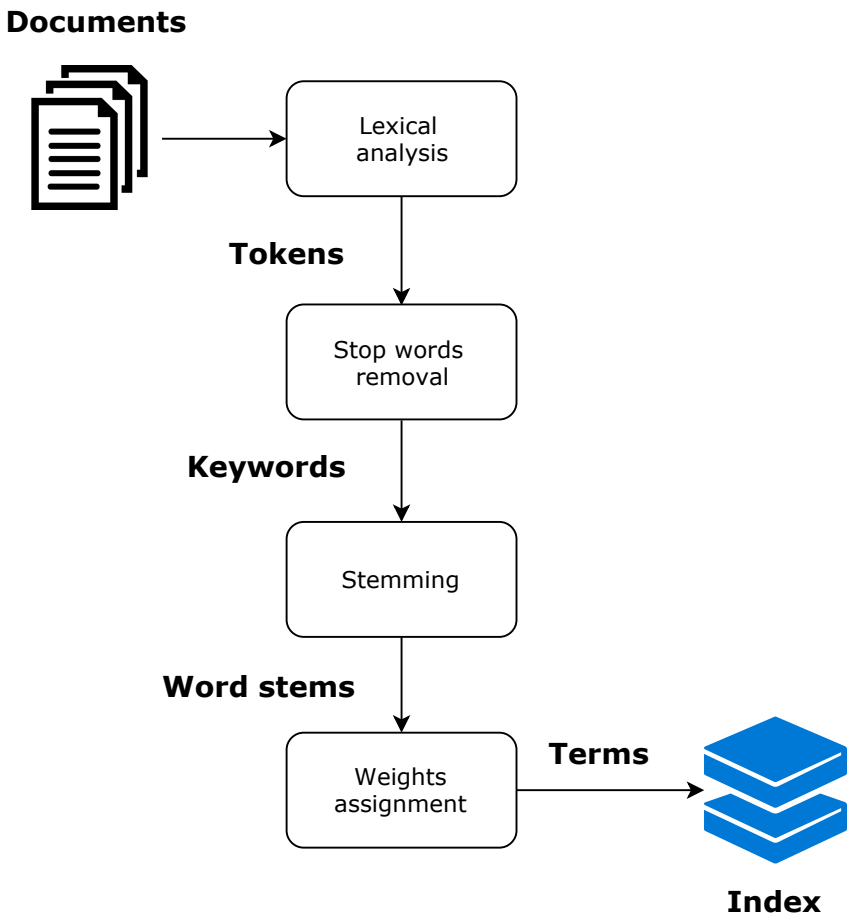

Figure 2.5: Indexing process: step by step.

The most important phases reported in Figure 2.5 are:

1. **Lexical analysis**: this phase consists of parsing and converting the words, of each document, into tokens. The tokens are the potential descriptors of every document. This phase is specific for every different language. If the document collection is made up of multimedia documents, this phase has to be specifically designed according to the given object type.

2. **Stop words removal**: the purpose of this phase is to delete the common words from the input stream of tokens. These common words are often function words, that are useful for the sentence structure but do not contribute to the information content of a document. Some examples of function words are: "the", "of", "for" and so on.

These words are also called stop words, because the indexing process stops when one of them is met, to decide whether keeping or not keeping it. Since stop words are common, removing them is useful for reducing the index size.

3. **Stemming**: also known as "conflation", is the phase in which are pointed out the stems of the keywords considered. The purpose is to group, all the words with the same stem, to reduce them to the common shared word, i.e. the stem. For example, if we consider the three words: "fish", "fishing" and "fisherman" the common stem is "fish", which is the reference index term for the three words in the index structure. The component that performs the stemming task is the stemmer. This one uses a stemming algorithm that is language specific. For this reason, there are different stemmers available in every IRS, the most popular for the English language is the Porter stemmer.

4. **Weights assignment**: once the candidate index terms are available, a phase of term composition can merge two or more single terms that are semantically linked. For example, the two words "tropical" and "fish" can be considered as a single term composed of two words: "tropical fish". This is meaningful since the word "tropical" specify a property of the word "fish". Finally, the weights assignment phase provides a weight for each index term, that reflects the relative importance of the index term in each document.

2.3 Incremental indexing

The indexing process is a complex task that involves a lot of activities such as stopwords removal, stemming and weights assignment. This implies that, when the given document collection is very large (VLC), the indexing process can take hours if not days to fully index the whole collection. For this reason, we need a method to continue using our IRS while the indexing process proceeds in background. This method consists of using an incremental index, capable of adding dynamically new terms to the index when they are found in documents, as the indexing process advances. In Figure 2.6 is shown, at a high-level of abstraction, how a general incremental indexing process works.The document collection is divided into n buckets, where each bucket represents a percentage of the collection. For example, if $n = 10$ then the first bucket represents the initial 10% of the document collection, the second is the first 20% and finally, the *n-th* bucket corresponds to the 100% of the indexed collection. Therefore, the incremental indexing process consists of indexing each bucket in order by increasing percentage. In this way, the user can query the system without having to wait for the full indexed collection. Obviously, every time a query is submitted only the indexed documents until then can be retrieved.

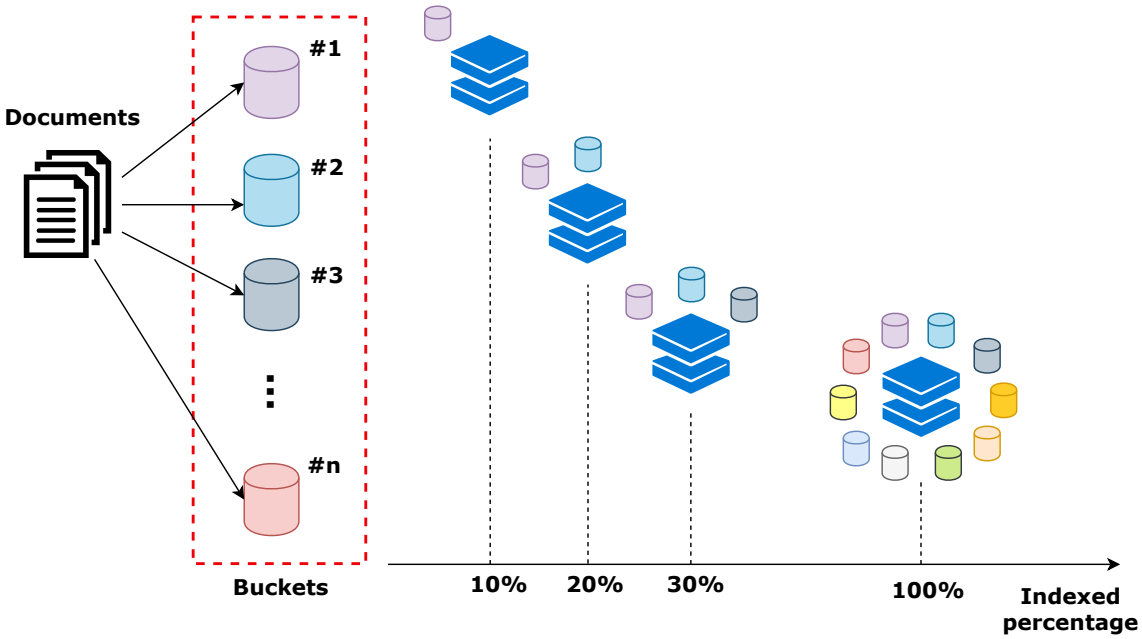

Figure 2.6: Incremental indexing process.

2.4 Retrieval models

A retrieval model is a set of rules, constructions and algorithms designed to make possible:

- The representation of the document's information content.
- The representation of the queries.
- The retrieval task.

The main activity, regarding the retrieval process, is evaluating the similarity between a given query and the set of documents, from the document collection. Once the similarity has been evaluated, the list of the relevant judged documents is ready to be proposed to the user. This list is made by sorting the selected documents, using the relevance score computed by the similarity evaluation process. The relevance score is strictly related to the document rank: the higher the score the lower is the rank. For this reason, at the top of the list, there are the most relevant documents with a high score and a low rank. In Figure 2.7 is shown the retrieval model architecture and its components.

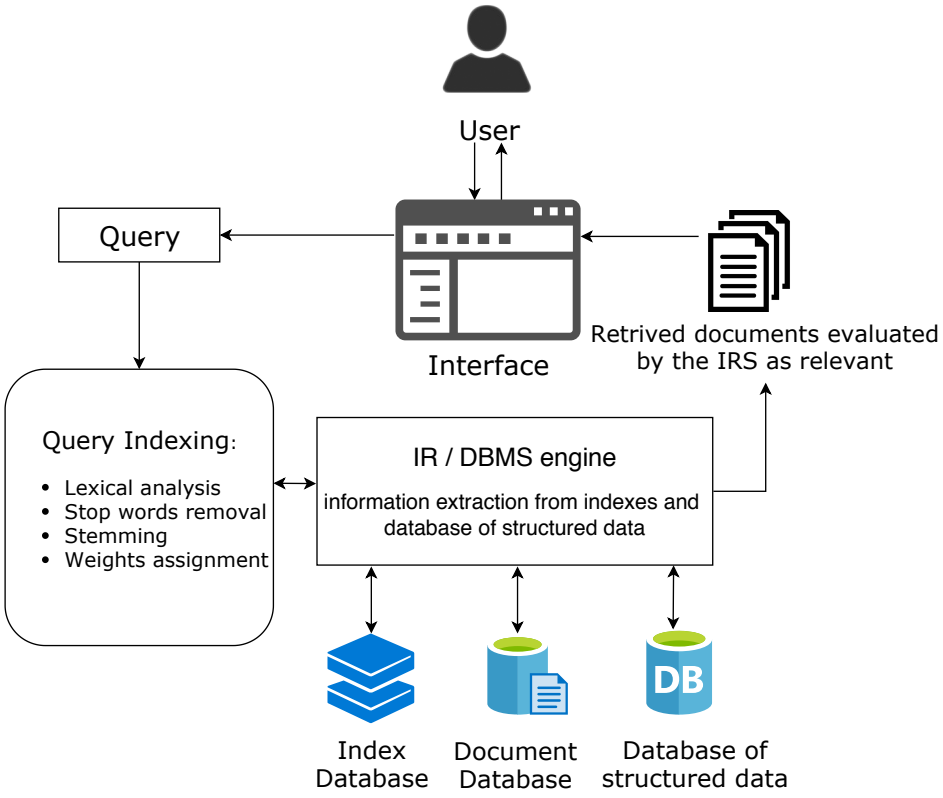

Figure 2.7: Retrieval model architecture and components.

2.4.1 Principal retrieval models

There are a lot of different aspects to consider for the retrieval process. For example, if we consider a general search engine it takes into account the authority of the source, the age of the document, the language in which is written and so on. For this reason, from the beginning of the information retrieval field, a lot of different IRS and retrieval model have been developed. Some of the most famous retrieval models are the boolean model, the vector space model and the probabilistic model.

2.4.1.1 Boolean model

The boolean model dates back to the 1950s, it was one of the first retrieval models to be used in industrial systems, search engines and digital libraries. This model owes its name to the boolean logic (TRUE or FALSE) used for the similarity evaluation task done for each input query. This means that every time a user submits a query to an IRS that adopts the boolean model, there are only two possible results for a document: relevant (TRUE) or not relevant (FALSE). This type of retrieval method is also known as exact-match retrieval, because a document, to be retrieved, has to exactly match the query, differently it is not considered. The boolean model also allows the user to write queries using the boolean operators such as AND, OR and NOT. In Figure 2.8 is reported an example document collection made up of three documents: D_1, D_2 and D_3 that contain three words: boat, fish and tropical, according to the plot intersections between the rectangular sets.

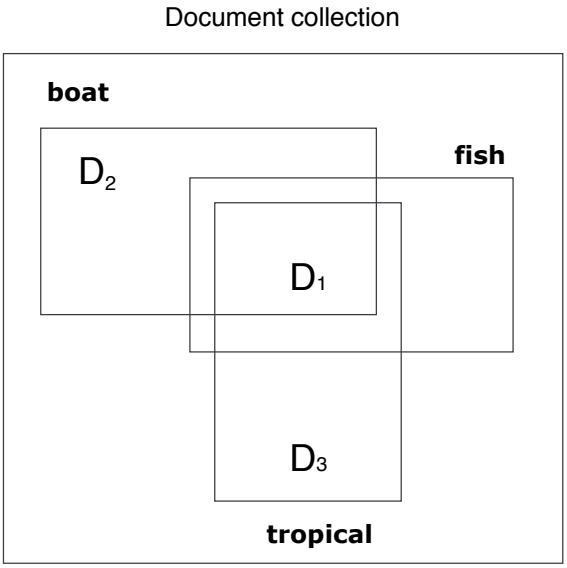

Figure 2.8: Example of overlappings in the content of three documents of the collection.

With reference to Figure 2.8 we list some queries examples as follows:

1. The query: "tropical AND fish" would return just the document D_1, because it is the only document that belongs to the intersection between the two sets: "tropical" and "fish".

2. The query: "tropical OR fish" would return the documents: D_1 and D_3 because the first one it is the only document that belongs to the intersection between the two sets: "tropical" and "fish" while D_3 belongs only to the "tropical" set, so the condition is satisfied.

3. The query: "tropical AND NOT fish" would return just the document D_3 because it is the only document that belongs to the "tropical" set that does not contain "fish".

4. The query: "tropical AND boat AND NOT fish" would return no documents, because the only document that contains the word "boat" contains also the word "fish".

It is important to notice that, since a document can be only "relevant" or "not relevant", there is no sorting criterion to order the list of the relevant judged documents.

2.4.1.2 Vector space model

The vector space model, or just vector model, was proposed by Gerard Salton in [4] and in the 1970s, it was one of the most important points of reference for the research in the field of information retrieval. The vector space model implements the term weighting and allows to realise the sorting of the list of the relevant judged documents. This is a great improvement if compared to the Boolean model which can only classify documents as "relevant" or "not relevant". The vector space model represents each document as a vector of t dimensions where t is the number of index terms:

$$D_i = (d_{i1}, d_{i2}, ..., d_{ij}, ..., d_{it})$$

where d_{ij} is the weight of the $j-th$ term in the $i-th$ document.In the same way, a query Q is represented by a vector of t weights, where t is the number of index terms:

$$Q = (q_1, q_2, ..., q_j, ..., q_t)$$

where q_j is the weight of the $j-th$ term in the query Q.

So if a collection is made up of n documents and t terms, we can represent it as a matrix where each row is a document and in each column, there is the weight of the $j-th$ term in the $i-th$ document.

	$Term_1$	$Term_2$	...	$Term_j$	...	$Term_t$
Doc_1	d_{11}	d_{12}	...	d_{1j}	...	d_{1t}
Doc_2	d_{21}	d_{22}	...	d_{2j}	...	d_{2t}
⋮	⋮	⋮		⋮		⋮
Doc_i	d_{i1}	d_{i2}	...	d_{ij}	...	d_{it}
⋮	⋮	⋮		⋮		⋮
Doc_n	d_{n1}	d_{n2}	...	d_{nj}	...	d_{nt}

Table 2.1: Term-document matrix.

To evaluate the similarity between a document and a query, the vector space model use the cosine correlation. This measure allows to compute the distance between the vector of each document and the vector of the considered query. In particular, the cosine correlation measure computes the cosine of the angle between the vector of each document and the query one. The mathmatical definition of the cosine correlation similarity follows:

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} \left(d_{ij} \cdot q_j\right)}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}} \tag{2.1}$$

To use the equation 2.1 we have to specify the weighting scheme adopted. The most famous weighting scheme is the $tf.idf$, where the term tf stands for "term frequency" while idf stands for "inverse document frequency". In particular, the tf term indicates the frequency of a term in the document D_i, so it reflects the importance of a term in a document. Instead, the idf term reveals the importance of a term in the document collection. This implies that if a term is presents in many documents, this would not be a good discriminator for the documents that contain it, during the retrieval phase.

2.4.1.3 Probabilistic model

When an IRS returns the list of the relevant judged documents for a query, not all the documents retrieved are effectively relevant. Indeed, there are many situations in which, for example, the query is ambiguous and, for this reason, the retrieved documents contain information that does not satisfy the information needs of the user. So we can reasonably assume that every time the value for the similarity evaluation measure is computed, each document has a probability to be considered relevant that is higher if the similarity value is higher, but it could be not relevant anyway even if the probability of being not relevant is very low. This is what probabilistic retrieval models try to model, according to the Probability Ranking Principle [5]. The Probability Ranking Principle states that if an IRS provides, for each query, a response as a list of ranked documents in order of decreasing probability of being not useful for the user and these probabilities are as accurate as possible, on the basis of the data available for this purpose, then the overall effectiveness of the system will be the best obtainable with that data. So if we can compute the probability of a document of being relevant or not relevant, we can classify documents just putting them into the set with the highest probability. According to this reasoning, we can decide that a document D can be considered relevant if the conditional probability of being relevant is greater than the one of being not relevant. We can express the same concept through the inequality: $P(R|D) > P(NR|D)$, where the $P(R|D)$ term is the probability for the document D of being relevant and $P(NR|D)$ is the probability for the document D of being not relevant. This approach is also called the Bayes Decision Rule. Consequently, we define as Bayes classifier every system that decides if a document is relevant, or not relevant, using this rule.

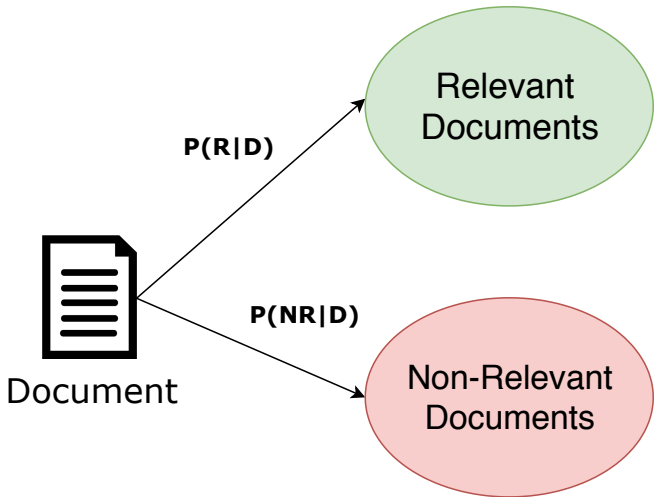

Figure 2.9: Bayes classifier.

Using the Bayes' rule we can find a direct relation between $P(R|D)$ and $P(D|R)$ as follows:

$$P(R|D) = \frac{P(D|R) \cdot P(R)}{P(D)} \tag{2.2}$$

Therefore, according to the equation 2.2, we can say that a document is relevant if $P(D|R) \cdot P(R) > P(D|NR) \cdot P(NR)$ or equivalently if the following inequality is met:

$$\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)} \tag{2.3}$$

Now we need to find a way for estimating $P(D|R)$ and $P(D|NR)$. The answer to this problem is given by the Binary Independence Model (BIM), proposed by Robertson and Jones in 1976 [6]. The BIM takes its name from two main assumptions:

1. Binary features.
2. Term independence (Naïve Bayes assumption).

The first one indicates that documents are represented by vectors with binary features. For example, a document D is represented by a vector of t components, where t is the number of terms: $D = (d_1, d_2, \ldots, d_t)$. Each d_i is a binary feature that indicates if the term i is present in the document ($d_i = 1$) or not ($d_i = 0$). The second assumption, claims that each term can appear in a document indipendently by the others, this allows to multiply the single term probabilities together. With these assumptions, we can estimate $P(D|R)$ as follows:

$$P(D|R) = \prod_{i=1}^{t} P(d_i|R) \tag{2.4}$$

Now we can rewrite the equation 2.4, using p_i as the probability that the term i occurs in a document from the relevant set. The resulting equation follows:

$$P(D|R) = \prod_{i:d_i=1} p_i \cdot \prod_{i:d_i=0} (1 - p_i) \tag{2.5}$$

In the same way, we can rewrite the equation for $P(D|NR)$ as follows:

$$P(D|NR) = \prod_{i:d_i=1} s_i \cdot \prod_{i:d_i=0} (1 - s_i) \tag{2.6}$$

where s_i represents the probability that the term i occurs in a document from the non-relevant set. Now we can express the likelihood ratio $\frac{P(D|R)}{P(D|NR)}$ as follows:

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{(1 - p_i)}{(1 - s_i)} \tag{2.7}$$

Finally, if we apply the logarithm function to the equation 2.7, the product becomes a sum, which is the score function for the probabilistic models:

$$\sum_{i:d_i=1} log \frac{p_i \cdot (1 - s_i)}{s_i \cdot (1 - p_i)} \tag{2.8}$$

The probabilistic models' score function, expressed in 2.8, is explained in detail in [7].

Some examples of probabilistic retrieval models are:

- **BM25**[1]: the BM25 model, also known as "Okapi BM25" from the Okapi IR system, is a retrieval model based on the probabilistic retrieval framework developed by Stephen E. Robertson and Karen Spärck Jones[8], between the 1970s and 1980s. This one is considered one of the most successful retrieval model algorithms [9].
- **DFR**[2]: the Divergence from Randomness (DFR) model is a generalisation of Harter's 2-Poisson indexing-model. This model is based on the following idea: "The more the divergence of the term-frequency within the document from its frequency within the collection, the more the information carried by the term t in document d". This idea has been developed in many different variants, that are called DFR models. Some examples of these models are PL2, InL2 and BB2 [10].

[1] https://en.wikipedia.org/wiki/Okapi_BM25
[2] http://terrier.org/docs/v2.2.1/dfr_description.html

2.5 Evaluating information retrieval systems

The purpose of the evaluation process, as the name suggests, consists of evaluating information retrieval systems, to improve them and to have a deeper understanding about their functioning, limits and strong points. The evaluation process provides useful data that can be used to make decisions and also to build better IRS. In IR the evaluation process evaluates the effectiveness of an IRS, measuring how well an IRS retrieves the documents that satisfy the user information need. Since the evaluation process regards mainly the effectiveness of an IRS, we need to outline the difference between effectiveness and efficiency. The effectiveness indicates how well the ranked list produced by an IRS, corresponds to the one that satisfies the information need of the user. The efficiency instead, measures how quickly the response is provided to the user. In addition, the evaluation process involves many tests that allow to understand how the system behaves in different contexts and situations. As we can see in Figure 2.10, the evaluation process is a transversal task, because every part of an IRS contributes to produce the results that will be evaluated. Besides, it is important to outline that the evaluation data indicate the performances of IR systems and allow to compare them. For this purpose, the experiments must be repeatable on the same comparative base. Otherwise, it is impossible to compare different IR systems.

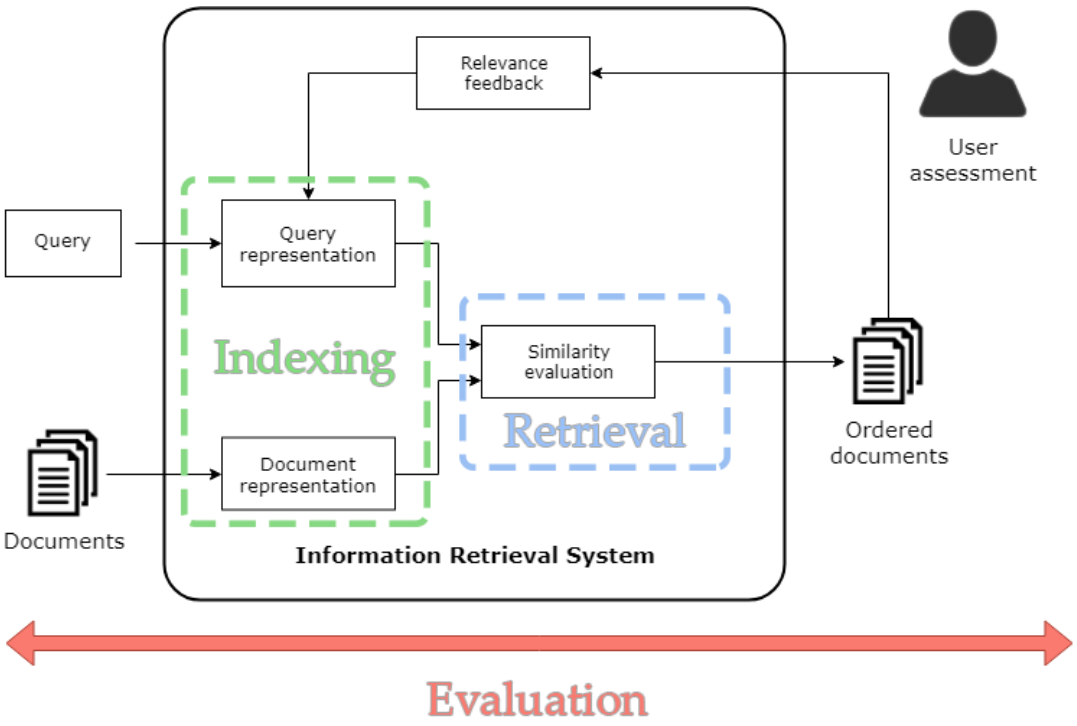

Figure 2.10: Evaluating an Information Retrieval System.

2.5.1 The Cranfield paradigm

The Cranfield paradigm is considered a standard for the evaluation of IR systems. As reported in [11], this paradigm was developed by Cyril Cleverdon, that in 1958 was the librarian of the College of Aeronautics, which was situated in Cranfield, hence the name of the paradigm. The birth of this paradigm was due to the necessity of indexing the huge amount of scientific documents produced after World War II. Make these indexes was very expensive, so the scientific community started wondering about which indexing system should be used. To answer this question, the only way was evaluating the efficiency of documentation systems, by means of experimental results. For these reasons, from 1958 to 1962, the first Cranfield I was run to test four different manual indexing methods. The results of Cranfield I were not so good as expected: the analysis did not point out a significant difference between the indexing system considered. Anyway, Cranfield I it is remembered because it was the first example of failure analysis. That is a very important practice in IR because allows others not to make the same mistakes. Afterwards, from 1962 to 1966, Cranfield II was run and, on this occasion, the concept of *test collection*, or experimental collection, was introduced for the first time. After the efforts of Cranfield I and II, the methodologies established since then, are now reused in evaluation campaigns around the world. The purpose of these evaluation campaigns is to create new test collections on which evaluating IRS, through the collaboration of many different academic research groups. Some examples of evaluation campaigns are:

- TREC[3] (Text REtrieval Conference): born in 1992, after the National Institutes of Standards (NIST) started the TIPSTER program, to evaluate the DARPA (Defense Advanced Research Projects Agency) system results. Nowadays, it is the evaluation campaign of reference for the USA.

- CLEF[4] (Conference and Labs of Evaluation Forum): started in 2000 after the 1999 TREC track for the evaluation of cross-language IRS, CLEF has been the IR reference organisation for the European area, since it is focused on European languages.

- NTCIR[5] (NII Testbeds and Community for Information access Research): born in 1997, it is the Japanese counterpart of TREC.

[3] http://trec.nist.gov/
[4] http://www.clef-initiative.eu/
[5] http://research.nii.ac.jp/ntcir/index-en.html

- FIRE[6] (Forum for Information Retrieval Evaluation) started in 2008, it is the South Asian counterpart for TREC, CLEF and NTCIR.

2.5.2 Concepts and definitions

In this section, we define the basic concepts that are necessary to understand how an IRS is evaluated and what are the metrics used in the evaluation process. The first concept we introduce is the definition of *test collection*. A test collection $C = \{D, T, RJ\}$ is defined as a triple made of:

1. The corpus of documents D.
2. The set of topics (information needs) T.
3. The set of relevance judgements RJ.

An example of a test collection is the CACM[7] (Communications of ACM), which is a collection born with the SMART project in 1982. In the TREC context, we can rewrite the test collection definition as follows:

$$C = \{D, T, RJ\} \longrightarrow C = \{D, T, P\} \longrightarrow C = \{D, T, GT\}$$

Instead of the relevance judgements RJ, we place the pool P which is our ground truth GT. The pool, or ground truth, it is a fundamental part of the evaluation process, since it tells if a document is relevant or not for the given topic (information need). In this way, we can compare the ranked list of results returned by an IRS with the pool, to see which relevant judged documents are really relevant and which are not relevant. After the comparison with the ground truth, we can estimate the effectiveness of an IRS using a set of evaluation measures. The pool creation activity is also known as pooling. Since in practice it is impossible to judge every document for every topic, usually the pool is made by judging only a sample of documents for every topic. The pooling strategy adopted by TREC consists of judging only the sample of documents that are obtained by experiments, or runs, made by the participants.

[6] http://fire.irsi.res.in/fire/2019/home
[7] http://ir.dcs.gla.ac.uk/resources/test_collections/cacm/

So now let us formally define all the useful notions, reported in [12], related to the evaluation process.

Definition 1. Let $D = \{d_1, d_2, \ldots, d_n\}$ be a set of documents, $T = \{t_1, t_2, \ldots, t_m\}$ a set of topics and a natural number $N \in \mathbb{N}^+$, a **run** is defined as a function:

$$\begin{aligned} R\colon\ & T \to D^N \\ & t \longmapsto \mathbf{r}_t = (d_1, d_2, \ldots, d_N) \end{aligned}$$

such that $\forall\, t \in T,\ \forall\, j,k \in [1,N] \mid j \neq k \implies \mathbf{r}_t[j] \neq \mathbf{r}_t[k]$ where $\mathbf{r}_t[j]$ it is the $j-th$ element of $\mathbf{r}_t$ vector.

Let REL be a finite set of relevance grades and let $\preceq$ be a total order relation on REL such that $(REL, \preceq)$ is a totally ordered set, then we can define the ground truth function as follows.

Definition 2. Let $D = \{d_1, d_2, \ldots, d_n\}$ be a set of documents and $T = \{t_1, t_2, \ldots, t_m\}$ a finite set of topics, the **ground truth** function follows:

$$\begin{aligned} GT\colon\ & T \times D \to REL \\ & (t,d) \longmapsto rel \end{aligned}$$

Definition 3. Given a run $R(t) = \mathbf{r}_t$, the **relevance score** of $R(t)$ is the following function:

$$\begin{aligned} \widehat{R}\colon\ & T \times D^N \to REL^N \\ & (t, \mathbf{r}_t) \longmapsto \widehat{\mathbf{r}}_t = (rel_1, rel_2, \ldots, rel_N) \end{aligned}$$

where

$$\widehat{\mathbf{r}}_t[j] = GT\left(t, \mathbf{r}_t[j]\right)$$

Let $W \subset \mathbb{Z}$ be a totally ordered finite set of integers, REL a finite set of relevance grades and let $RW : REL \longrightarrow W$ be a monotonic function which maps every relevance grade $\left(rel \in REL\right)$ into a relevance weight $\left(w \in W\right)$, we can define RW as follows.

Definition 4. Given a run $R(t) = \mathbf{r}_t$, the **relevance weight** of $R(t)$ is the following function:

$$\begin{aligned} \widetilde{R}\colon\ & T \times D^N \to W^N \\ & \left(t, \mathbf{r}_t\right) \longmapsto \widetilde{\mathbf{r}}_t = \left(w_1, w_2, \ldots, w_N\right) \end{aligned}$$

where

$$\widetilde{\mathbf{r}}_t\left[j\right] = RW\left(\widehat{\mathbf{r}}_t\left[j\right]\right)$$

Definition 5. Let D be the set of retrieved documents for a topic t and D^* the set of relevant documents for t, we can define the **precision** ($Prec$) as follows:

$$Prec = \frac{\left|D^* \cap D\right|}{|D|}$$

where $\left|D^* \cap D\right|$ is the number of relevant documents retrieved, while $|D|$ is the number of retrieved documents.
The precision is an evaluation measure that represents the ratio between the number of relevant documents retrieved $\left|D^* \cap D\right|$ and the number of retrieved documents $|D|$. In other words, it expresses how many relevant documents are retrieved, by an IRS, with respect to the total number of retrieved documents.

Definition 6. Let D be the set of retrieved documents for a topic t and D^* the set of relevant documents for t, we can define the **recall** (Rec) as follows:

$$Rec = \frac{\left|D^* \cap D\right|}{|D^*|}$$

where $\left|D^* \cap D\right|$ is the number of relevant documents retrieved, while $\left|D^*\right|$ is the number of relevant documents.
The recall is an evaluation measure that represents the ratio between the number of relevant documents retrieved $\left|D^* \cap D\right|$ and the number of relevant documents $\left|D^*\right|$. In other words, it expresses how many relevant documents are retrieved, by an IRS, with respect to the total number of relevant documents. Besides, the number of relevant documents $\left|D^*\right|$, for a topic

t, is also called **recall base** RB_t. It is important to notice that every topic has got a fixed recall base, according to the pool (ground truth).

Definition 7. Let $D = \{d_1, d_2, \ldots, d_n\}$ be a finite set of documents, $T = \{t_1, t_2, \ldots, t_m\}$ a finite set of topics, GT the ground truth defined on D and T and REL a totally ordered set of relevance judgements, then the **recall base** function follows:

$$\begin{aligned} RB:\ & T \to \mathbb{N} \\ & t \longmapsto RB_t = \left| \{ d \in D \mid GT(t,d) \succ min(REL) \} \right| \end{aligned}$$

The meaning of the symbol $\succ$, in the above relation, is that the recall base function returns only the documents such that their relevance judgements, in the ground truth, are greater than the minimum relevance grade in the set REL.

Definition 8. Given a topic $t \in T$, a recall base RB_t, $REL = \{nr, r\}$ and a run $\mathbf{r}_t$ of size $N \in \mathbb{N}^+$ such that:

$$\forall i \in [1, N], \widetilde{\mathbf{r}}_t = \begin{cases} 0, & \text{if } \widetilde{\mathbf{r}}_t[i] = nr \\ 1, & \text{if } \widetilde{\mathbf{r}}_t[i] = r \end{cases}$$

we can define the **Average Precision** (AP) as follows:

$$AP = \frac{1}{RB_t} \sum_{k=1}^{N} \widetilde{\mathbf{r}}_t[k] \frac{\sum_{h=1}^{k} \widetilde{\mathbf{r}}_t[h]}{k}$$

The Average Precision is one of the most important evaluation measure in IR. It can only assume values in the range between 0 and 1. Besides, the Average Precision is a **top heavy** measure, this implies that it gives a better score to the runs which have more relevant documents on the top of the list.

Definition 9. Given a run R and a set of topics T, the **Mean Average Precision** (MAP) is defined as follows:

$$MAP(R) = \frac{\sum_{t \in T} AP(\mathbf{r}_t)}{|T|}$$

The Mean Average Precision, like AP, is another important evaluation measure in IR. It is the mean of the AP measure, computed over all the possible topics of T.

Definition 10. Let $R(t)$ be a run of size $N \in \mathbb{N}^+$, where $t \in T$ is a given topic, RB_t is the related recall base and $j \in \mathbb{N}^+$ is a natural number such that $1 \leq j \leq N$, then the **Cumulative Gain** (CG) at cut-off j $\Big(CG\big[j\big]\Big)$ is defined as follows:

$$CG\big[j\big] = cg_{\mathbf{r}_t}\big[j\big] = \sum_{k=1}^{j} \widetilde{\mathbf{r}}_t\big[k\big]$$

It is important to notice that CG is not a top heavy measure.

Definition 11. Given a run $R(t)$ of size $N \in \mathbb{N}^+$ and a logarithmic base $b \in \mathbb{N}^+$, $\forall k \in \big[1, N\big]$ the **discounted gain** is defined as follows:

$$dg^b_{\mathbf{r}_t}\big[k\big] = \big(R\big) = \begin{cases} \widetilde{\mathbf{r}}_t\big[k\big], & \text{if } k < b \\ \frac{\widetilde{\mathbf{r}}_t[k]}{log_b k}, & \text{otherwise.} \end{cases}$$

Definition 12. Let $R(t)$ be a given run, the **Discounted Cumulative Gain** (DCG) at cut-off j $\Big(DCG\big[j\big]\Big)$ is defined as follows:

$$DCG\big[j\big] = \sum_{k=1}^{j} dg^b_{\mathbf{r}_t}\big[k\big]$$

It is important to notice that DCG is a top heavy measure, that can be computed at various cut-offs. Furthermore, the DCG values are not contained between 0 and 1.

Definition 13. The **ideal run** $I(t) = \mathbf{i}_t$ is the run that satisfies the following requirements:

(1) $$t \in T, \left|\Big\{j \in \big[1, N\big] \mid GT(t, \mathbf{i}_t[j]) \succ min\big(REL\big)\Big\}\right| = RB_t$$

(2) $$\forall\, t \in T, \forall\, j, k \in \big[1, N\big] \mid j < k \implies \widehat{\mathbf{i}}_t[j] \succeq \widehat{\mathbf{i}}_t[k]$$

Hence, the ideal run corresponds to the best sorting of the ground truth, according to the relevance judgements. In other words, the ideal run represents the perfect retrieval scenario.

Definition 14. Let $R(t)$ be a run of size $N \in \mathbb{N}^+$, where $t \in T$ is a given topic, $I(t) = \mathbf{i}_t$ is the ideal run and $j \in \mathbb{N}^+$ is a natural number such that $1 \leq j \leq N$, then the **normalized Cumulative Gain** (nCG) at cut-off j $\left(nCG\left[j\right]\right)$ is defined as follows:

$$nCG\left[j\right] = \frac{cg_{\mathbf{r}_t}\left[j\right]}{cg_{\mathbf{i}_t}\left[j\right]}$$

Definition 15. Let $R(t)$ be a given run, where $t \in T$ is a given topic and $I(t) = \mathbf{i}_t$ is the ideal run, then the **normalized Discounted Cumulative Gain** (nDCG) at cut-off j $\left(nDCG\left[j\right]\right)$ is defined as follows:

$$nDCG^b\left[j\right] = \frac{\sum_{k=1}^{j} dg^b_{\mathbf{r}_t}\left[k\right]}{\sum_{k=1}^{j} dg^b_{\mathbf{i}_t}\left[k\right]}$$

It is important to notice that nDCG allows to compare different runs on the same topic and it assumes values between 0 and 1.

2.6 Visual Analytics

2.6.1 Introduction

The purpose of visual analytics is to help users in data exploration activities. This kind of activities is common in every research field and with the advent of big data has become more and more popular. Indeed in many different domains, experts have to deal with large datasets of thousands or even millions of data, that are not feasible to be explored without the aid of appropriate data exploration tools. For example, these instruments are fundamental for data that comes from medical institutions such as hospitals and clinics. In this context, mining healthcare data requires long processing time and typically produce, in turn, a lot of evaluation data. To understand these data usually, analysts have to run many analytics processes, wait for them to complete and then analyze the results. Figure 2.11 shows the typical visual analytics workflow, in which data, after a manipulation phase, are displayed in the user interface.

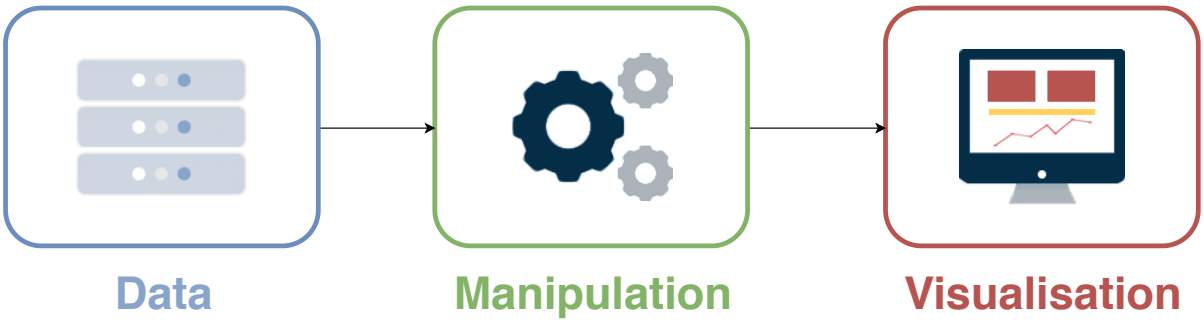

Figure 2.11: Visual Analytics workflow.

Visual analytics is a paradigm that provides visual components and dynamic charts useful to help experts in understanding data and results through interactive user interfaces. For example, the visual analytics tool developed in this thesis, AVIATOR, uses bar charts and scatter plots to provide two different analyses: "topic per topic" and "overall". Both analyses, as described in Chapters 3 and 5, are designed to help IR experts explore evaluation data. Since AVIATOR is a progressive visual analytics tool developed for IR purposes, the following sections explain the main concepts regarding visual analytics applied to the IR field.

2.6.2 Visual Analytics for Information Retrieval

Evaluating one or more IR systems is a complex task that involves a lot of components, parameters and measures. The evaluation process is done using dedicated test collections usually made during evaluation campaigns. A test collection could be made of millions of documents, and when this occurs, it is called Very Large Collection (VLC). Indexing all the documents of a VLC requires a lot of time and resources, but this is necessary if we want to use it for evaluating one or more IR systems. This implies a lot of effort for IR experts since they have to index the same collection for each IR system they want to evaluate. In this context, visual analytics techniques can help IR experts in the evaluation process to analyze and understand experimental results. To make the evaluation process more effective and to reduce the user effort, VIRTUE [12] and AVIATOR have been developed. In particular, the first one is designed to support and improve both performance and failure analyses, while AVIATOR uses the progressive visual analytics paradigm to make the evaluation process faster and more intuitive. For this purpose, it provides specific interactive charts by means IR experts can explore evaluation data and understand experimental results. In general, many different types of interactive charts can be developed according to the analyses of interest. For example, VIRTUE provides, for the performance analysis, the "Ranked Result Distribution Exploration" that allows the user to understand the overall performances of the systems considered. Another useful analysis is the "Failing Document Identification", which attempts to identify the documents that contribute the most to improve the performances of the chosen experiment.

2.6.3 Progressive Visual Analytics

As described in [13], Progressive Visual Analytics (PVA) "produces partial results during execution". There are several different ways to express the same concept related to PVA, the most famous are: Progressive Visualization [14], Progressive Analytics [15], Incremental Visualization [16] and Fine-Grain Visualization [17]. The main feature of the PVA paradigm is that users do not have to wait for the computation of the entire dataset to visualize data. This fundamental difference between visual analytics and its progressive version is explained in detail in [13] and [18]. PVA aims to reduce or eliminate the periods of time in which users are forced to wait for the end of analytics algorithms, before interacting with the system. A common alternative approach to PVA is to preprocess all the necessary data, before allowing interactive visualization. In this way, all the possible paths the user can explore are predetermined, thus there is no waiting time when switching from one configuration to

another. Unfortunately, this is not possible if there are a lot of different analytic parameters to consider because the number of possible combinations would explode. Besides, precomputing data limits creative explorations. To overcome these problems, the PVA approach requires analytic algorithms designed specifically to produce meaningful partial results at each execution round. Figure 2.12, from [13], shows the comparison between the traditional batch workflow and the progressive visual analytics one. The main difference is that batch visual analytics workflow forces the user to wait for the analytic process to complete, while the progressive version displays data as soon as they are available. Besides, the batch visual analytics workflow is associated with the paradigm of "compute-wait-visualize". Instead, PVA aims to eliminate the user's waiting time by means of meaningful partial results that describe, with the same format of the final results, the current state of the analytics process. Moreover, analysts can inspect partial results, when they become available, and then interact with the algorithm to change input parameters or stop analytics jobs that have produced results that are unlikely to be interesting. In this way, analysts can control analytic processes through PVA interfaces, that continuously provides useful information about the latest partial results, thus avoiding the inefficiencies of the "compute-wait-visualize" paradigm.

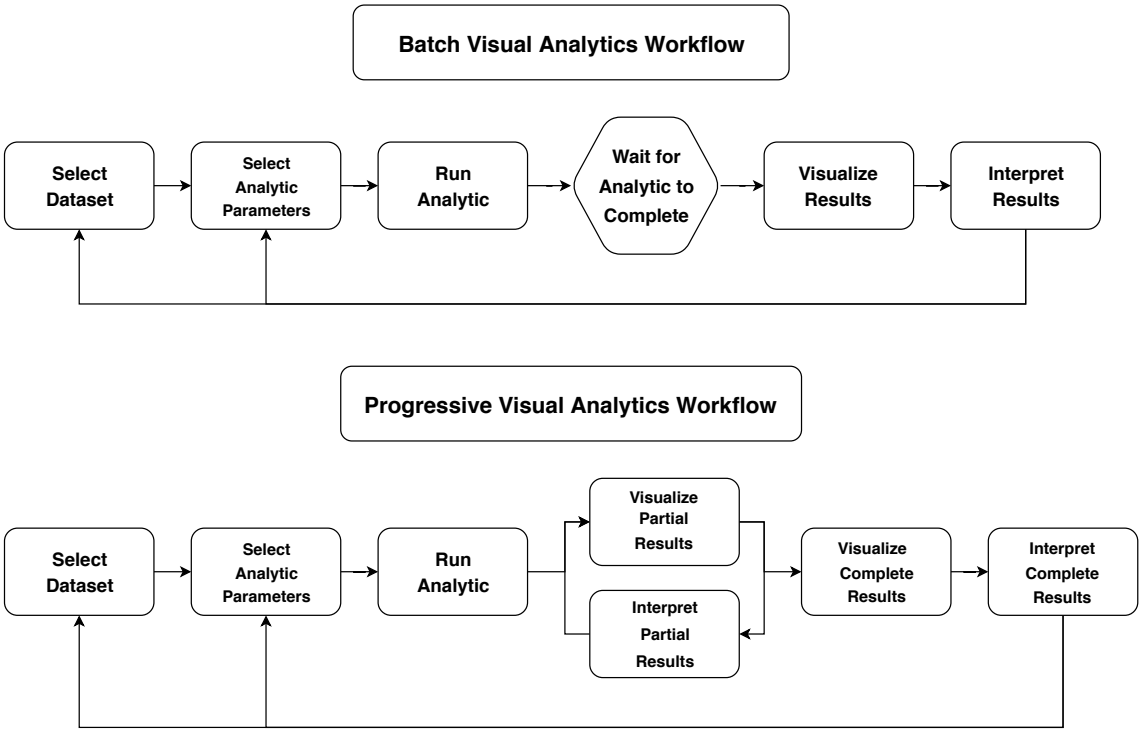

Figure 2.12: A comparison between batch and progressive visual analytics workflow.

CHAPTER 3

CONCEPTUAL FRAMEWORK

Since IR systems are becoming more and more sophisticated, at the same time, evaluating them has become a central task of fundamental importance. Unfortunately, we know that evaluating an IRS on a very large collection (VLC) is a hard task because it requires a lot of time and computational resources to have the whole collection indexed. Obviously, in many situations, we can not wait for the full-indexed collection. However, we can use the incremental indexing method to obtain useful evaluation data, while the collection is progressively indexed in the background. In this way, we do not have to wait for the full-indexed collection, we analyze and visualize data when they are available. This strategy is the basis for the conceptual framework developed and implemented in AVIATOR.

3.1 Background

All the activities performed by AVIATOR are related to a specific test collection of documents. For this reason, from the Background chapter 2, we recall the notion of test collection $C = \{D, T, RJ\}$, which is defined as a triple made up of:

1. The corpus of documents D.
2. The set of topics (information needs) T.
3. The set of relevance judgements RJ.

Another important concept to know is the definition of the bucket collection $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, which is a collection of disjoint uniform samples of documents, taken from the test collection C. These concepts are fundamental to understand the conceptual framework implemented in AVIATOR. We will discuss them in detail in the following section.

3.2 The conceptual framework

3.2.1 Process Overview

The AVIATOR process is composed of four principal phases: preprocessing, incremental indexing, retrieval and evaluation. This pipeline is more complicated than a general IR process, which is usually made up of three phases: indexing, retrieval and evaluation. The reason for this design is that we can not wait for the full-indexed collection to obtain the evaluation data. Therefore, the standard indexing phase is substituted by the incremental indexing one. Finally, the preprocessing phase is necessary to generate the bucket collection $\mathcal{B}$ used in the following phases.

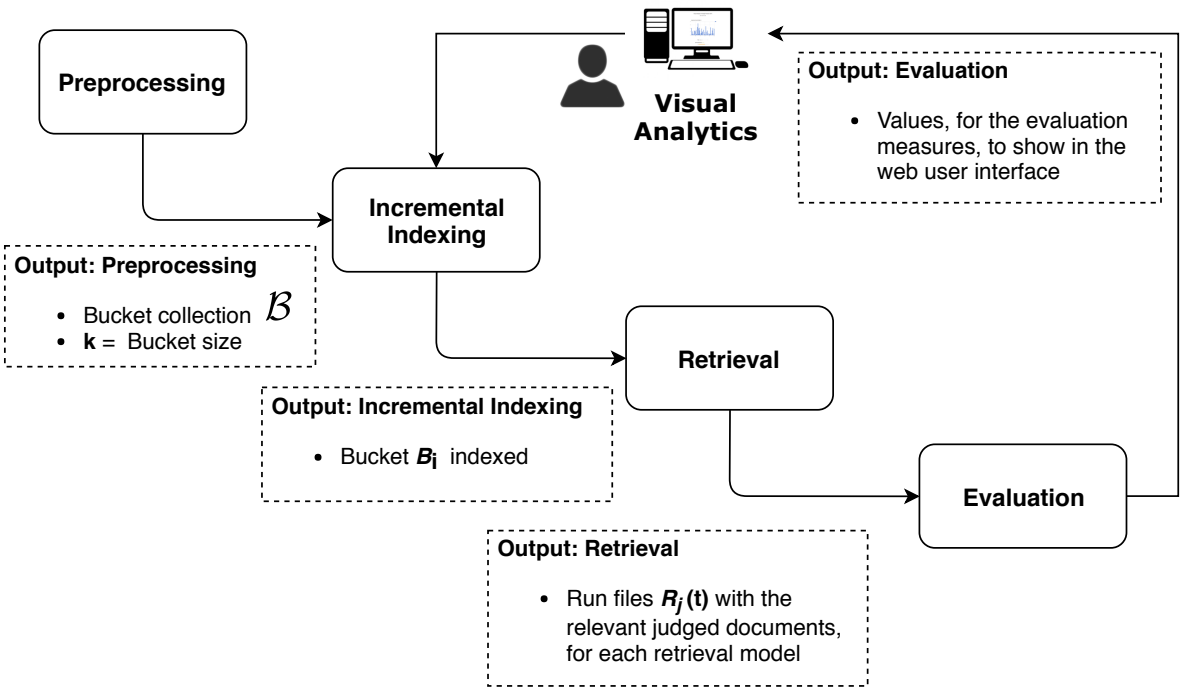

Figure 3.1: AVIATOR process: step by step.

3.2.2 Description

In Figure 3.1 are illustrated the principal phases that make up the AVIATOR process, which is described below:

1. **Preprocessing**: during this phase, the document collection C is divided into n buckets of the same size k. The bucket size k results from the following relation:

$$k = \left\lfloor \frac{|D|}{n} \right\rfloor \tag{3.1}$$

Where $|D|$ is the corpus size of the document collection C or, in other words, the number of documents that compose the document collection C. The floor function, applied to the fraction, means that we take an integer lower bound for the bucket size k. In this way, all the buckets are of the same size k with the exception, eventually, of the last one B_n, which could contain the documents due to the rest of the division. At the end of the preprocessing stage, the documents of C are rearranged into the n buckets of the bucket collection $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$. Each bucket B_i is populated by sampling uniformly the document collection C, so that $B_i \cap B_j = \emptyset$, $\forall\, i \neq j$. The uniform sampling is done for having all the buckets with different types of documents that, consequently, could be relevant for different topics. By doing so, at each stage of the incremental indexing process, every document has the same probability of being relevant for any topic. This strategy is described in [1] and we can summarize it as follows:

$$\mathcal{B} = \left\{ B_1, B_2, \ldots, B_n \right\}$$

Where B_i is the i-th bucket of size $k = |B_i| = |B_j|$, $\forall\, i \neq j$. Each bucket B_i is populated using the Bucket Populating Algorithm (BPA2), we describe it in the following section. So let us define some useful notions to understand how it works.

Definition 16. Given a finite set of documents $D = \{d_1, d_2, \ldots, d_N\}$, a finite set of topics $T = \{t_1, t_2, \ldots, t_m\}$, GT the ground truth defined on D and T and $REL = \{nr, r\}$ a totally ordered set of relevance judgements, we define Φ_t as the set of the relevant documents for a given topic $t \in T$ as follows:

$$\Phi_t = \{d \in D \mid GT(t,d) \succ min(REL)\}$$

Hence, Φ_t contains all the documents such as $GT(t,d) = r$. These documents are relevant for the topic t according to the ground truth or pool.

Definition 17. Given a finite set of documents $D = \{d_1, d_2, \ldots, d_N\}$, a finite set of topics $T = \{t_1, t_2, \ldots, t_m\}$ and the set of relevant documents Φ_t for a topic $t \in T$, we define Ω_t as the set of the not relevant documents for a given topic $t \in T$ as follows:

$$\Omega_t = \{d \in D \setminus \Phi_t\} = \{d \in D \mid d \notin \Phi_t\}$$

Hence, Ω_t contains all the documents such as $GT(t,d) = nr$. These documents are not relevant for the topic t according to the ground truth or pool.

As we said, uniform sampling is used to select documents from the collection $\mathcal{C}$, but we need to do that by ensuring each bucket is always composed both of relevant and not relevant documents. To assure this distribution, we use the following procedures and functions.

Algorithm 1 Uniform Sampling Without Replacement (SWR) algorithm

1: **procedure** SWR($\mathcal{S}$)
2: $d \leftarrow random(\mathcal{S})$ ▷ Save in d an element from $\mathcal{S}$ selected with uniform probability
3: $\mathcal{S} \leftarrow \mathcal{S} \setminus \{d\}$ ▷ Remove d from the set $\mathcal{S}$
4: **return** d

Definition 18. Given the Sampling Without Replacement (SWR1) algorithm, the set Φ_t of the relevant documents for a given topic $t \in T$ and the set of the not relevant documents Ω_t for the same topic t, we define the sampling without replacement function $s(t)$ as follows:

$$s(t) = \begin{cases} SWR(\Phi_t), & \text{if } \left(random\left(\{0,1\}\right) = 1 \wedge \Phi_t \neq \emptyset\right) \vee \Omega_t = \emptyset \\ SWR(\Omega_t), & \text{if } \left(random\left(\{0,1\}\right) = 0 \wedge \Omega_t \neq \emptyset\right) \vee \Phi_t = \emptyset \end{cases}$$

Hence, $s(t)$ will return $SWR(\Phi_t)$ in two cases:

a) when the result of $random\left(\{0,1\}\right)$ is equal to 1, which occurs with probability $p = \frac{1}{2}$, and Φ_t is not empty.

b) when Ω_t is empty.

Otherwise, $s(t)$ will return $SWR(\Omega_t)$ in two other cases:

a) when the result of $random\left(\{0,1\}\right)$ is equal to 0, which occurs with probability $q = \frac{1}{2}$, and Ω_t is not empty.

b) when Φ_t is empty.

It is important to notice, that the two cases of $s(t)$ can not happen together since one of the two sets Φ_t, Ω_t is always not empty, otherwise the whole collection C is already preprocessed, thus $s(t)$ would not be invoked. Now we have the necessary background to describe the Bucket Populating Algorithm (BPA2) as follows:

Algorithm 2 Bucket Populating Algorithm

1: **procedure** BPA($D, T, \mathcal{B}, k$)
2: $n \leftarrow |\mathcal{B}|$ ▷ Set n with the bucket collection size $|\mathcal{B}|$
3: $L \leftarrow getIDList(D)$ ▷ Save in L the document ID list
4: $i \leftarrow 1$ ▷ Initialize the bucket counter
5: **while** $(i < n)$ **do** ▷ Loop over the buckets $B_1, B_2, \ldots, B_{n-1}$
6: $c \leftarrow 1$ ▷ Initialize the topic counter
7: **while** $\left(|B_i| < k\right)$ **do** ▷ Loop until the bucket size k is reached
8: $d \leftarrow s(t_c)$ ▷ Save in d the document selected for the topic t_c
9: $l \leftarrow d.getID()$ ▷ Save in l the document ID of d
10: $B_i \leftarrow B_i \cup \{d\}$ ▷ Add d to the bucket B_i
11: $L \leftarrow L.remove(l)$ ▷ Remove l from the ID list L
12: **if** $(c < |T|)$ **then** ▷ Check if c is lower than the topic set size $|T|$
13: $c++$ ▷ Update the topic counter c
14: **else** ▷ Otherwise, reset c to 1
15: $c \leftarrow 1$ ▷ Reset the topic counter
16: $i++$ ▷ Update the bucket counter i
17: $B_n \leftarrow L.getDocuments()$ ▷ Save in B_n the documents whose id is left in L
18: **return** $\mathcal{B}$

The Bucket Populating Algorithm (BPA2), as the name suggests, populates each bucket B_i using the sampling without replacement function $s(t)$. The BPA iterates over the document ID list L and over the set of topics T. BPA uses L instead of D, to avoid any alteration for the original document corpus D. As we said, the document selection is made with the sampling without replacement function $s(t)$, which assures to have in each bucket a distribution of both relevant and not relevant documents. In this way, the generated buckets are representative samples of the given document collection $\mathcal{C}$.

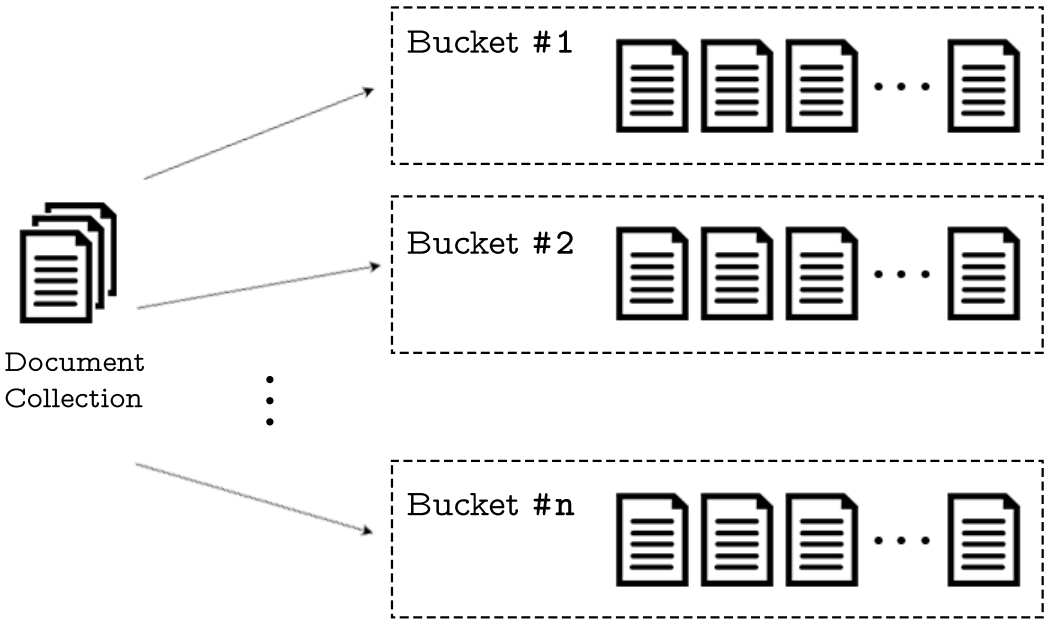

Figure 3.2: The document collection $\mathcal{C}$ is divided into n buckets of size k.

Since indexing a document collection is a task that requires a lot of time to be performed, especially in case of very large collections (VLC), to evaluate IR systems, in a time efficient way, we can not index the whole collection. However, we can evaluate data as they become available. For this reason, we use the approach described so far of dividing the corpus $\mathcal{D}$ of the document collection $\mathcal{C}$, into the n buckets of $\mathcal{B}$, to make the incremental indexing process as efficient as possible. As illustrated in Figure 3.1, the AVIATOR process is a cyclical task which loops through three phases: incremental indexing, retrieval and evaluation. At every iteration i, the B_i bucket is indexed; hence the documents contained in B_i become available for the retrieval phase. Doing so, iteration after iteration, the index grows and the performance of the system improves.

In this context, the choice of the bucket size k is essential to ensure the performance of the system. In particular, if the bucket size is too small we have a lot of buckets with just a few documents, this involves a lot of overhead since we need to index many buckets before having significant values for the evaluation measures. In the opposite case instead, with k too high, we have a few buckets with a lot of document, in this way we fall back into the starting problem: collection too big to be indexed in a reasonable time. To facilitate the testing task, AVIATOR allows the user to choose the number of buckets n, in which divides the document collection C. In this way, using the relation 3.1, k is uniquely determined. Anyway, we can also fix the bucket size k and then deduce the number of buckets n. A possible approach is to divide the document collection into buckets of size equal to the 10% of the collection size. Therefore, if the bucket size k is greater than a threshold, then k takes the threshold value, otherwise it does not change.

To choose the bucket size k, the following algorithm has been implemented.

Algorithm 3 Bucket size algorithm

1: **procedure** BUCKETSIZE($\mathcal{D}, \beta$)
2: $threshold \leftarrow \beta$ ▷ Set the default *threshold* for the bucket size
3: $k \leftarrow \frac{|\mathcal{D}|}{10}$ ▷ Set the bucket size k equal to the 10% of $|\mathcal{D}|$
4: **if** ($k > threshold$) **then** ▷ Check if the bucket size k is greater than *threshold*
5: $k \leftarrow threshold.$ ▷ if the condition is true then k takes the *threshold* value
6: **return** k

When the size of the document collection corpus $|\mathcal{D}|$ is very large, we cannot divide it just in ten buckets, because processing each bucket would require many computational resources and a long time. For this reason, the bucket size algorithm uses a threshold initialised with β, which is a value chosen by the user. In this way, if the bucket size k is greater than the threshold, then k is set equal to the threshold value. The threshold value can be chosen by the user, simply setting β, or can be computed using the following threshold algorithm.

The algorithm 4, instead, computes the threshold for the bucket size automatically: it doubles the threshold value until the time for indexing a bucket of that size, becomes greater than the time threshold τ set by the user. The time threshold τ is useful to

Algorithm 4 Threshold algorithm

```
1:  procedure THRESHOLD(β)
2:      threshold ← β                          ▷ Set the threshold with the default value β
3:      counter ← 0                            ▷ Initialize the iteration counter
4:      do
5:          counter++                          ▷ Update the iteration counter
6:          threshold ← 2 · threshold          ▷ Doubling the threshold
7:          t ← index()                        ▷ Save in t the time for indexing a bucket of threshold size
8:      while (t < τ)
9:      if (counter == 1) then                 ▷ Check if the iteration counter is equal to 1
10:         threshold ← β                      ▷ Restore the initial value for the threshold
11:     return threshold
```

guarantee the real-time performance of the AVIATOR system.
In particular, at the beginning, the threshold for the bucket size is initialized with β, later the threshold is doubled at each iteration while the loop condition is true. In this context, the iteration *counter* is useful because if the loop ends with *counter* equal to one, it means that the initial value for the threshold is already the best value, so we have to restore it.

2. **Incremental Indexing**: in this stage, the documents belonging to the i-th bucket B_i to process, are indexed by a batch job after the two phases of stop words removal and stemming. The incremental indexing process works by means of two indexing cores: Stable and Dynamic. As the name suggests, the Stable core is the one in which are consolidated the progress made by the Dynamic core, during the incremental indexing process. In Figure 3.3, it is shown how the incremental indexing process works and the roles of both the two cores. In particular, we can see that the Dynamic core takes in input the documents of the bucket B_i to be indexed, while the Stable core receives the requests from the user. At the beginning, both the cores are empty and the requests received by the Stable core can not be satisfied. After, when the Dynamic core finishes indexing the first bucket, e.g the first 10% of the document collection corpus $\mathcal{D}$, then there is a synchronisation phase between the two core: the index data of the

Dynamic core are copied to the Stable one. Since this moment, the Stable core can reply to the user requests, while the Dynamic core continues to index the documents of the next bucket in background. It is important to notice, that until the Dynamic core finishes indexing the next bucket, the progress of this one is not synchronised with the Stable core. This means that all the user requests received, by the Stable core before the synchronisation, are resolved with the latest version of the index available in the Stable core. Finally, when the last bucket B_n is indexed, the document collection C is full-indexed, the Dynamic core stops and the Stable one is synchronised for the last time.

3. **Retrieval**: given a set of topics T, the retrieval process consists of submitting a query, to the Stable core, for each topic $t \in T$. In particular, the Stable core takes each request and gives it to the search platform, where it is subjected to the two phases of stop words removal and then stemming. As we said, the Stable core reply to each request using the index version available after the last synchronisation phase with the Dynamic core. When the last bucket B_n is indexed, the Stable core is synchronised for the last time with the Dynamic core. After that, the index saved in the Stable core contains all the indexed documents of the collection C. For every submitted query, the Stable core returns, as a response, the sorted list of relevant judged documents. This list is the run $R_j(t)$ generated by the search platform, which is the IR system, for the topic t using the retrieval model j.

4. **Evaluation**: In this phase, the evaluation tool receives in input each run $R_j(t)$ and returns the list of the values for all the evaluation measures considered. As shown in Figure 3.1, these values are passed to the visual analytics web interface, so that they can be visualised by the user through a set of interactive charts. For this purpose, a lot of evaluation metrics are available in AVIATOR to be visualised. Some examples are: average precision (AP), mean average precision (MAP) and the normalised discounted cumulative gain (nDCG). The advantage, of having an exhaustive set of evaluation metrics to be displayed, is that we can deduce more useful information to improve the performance of the IR systems analyzed. This is the reason why the AVIATOR visual analytics web interface has been implemented.

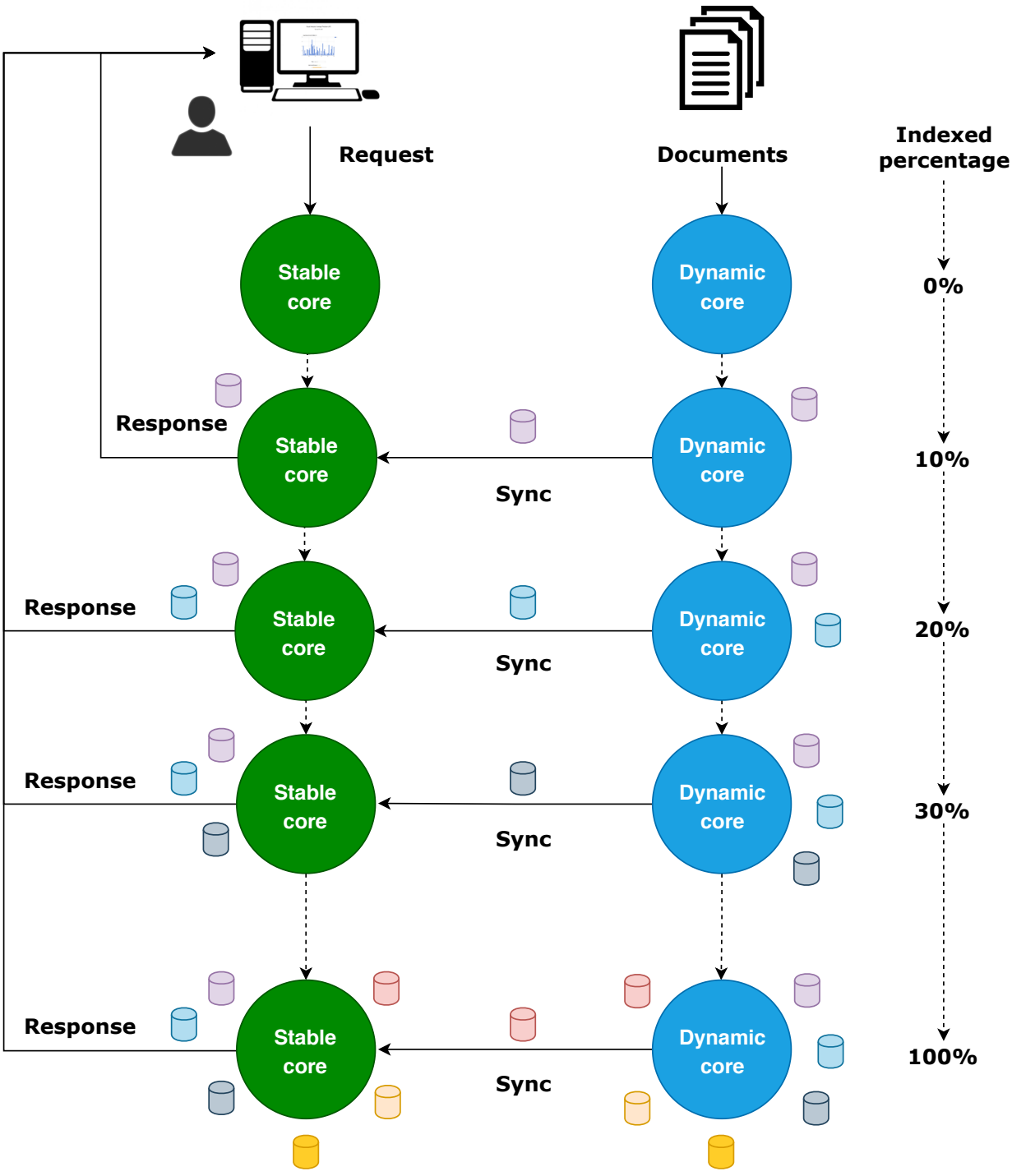

Figure 3.3: The incremental indexing process with Stable and Dynamic cores.

3.3 Mockup

3.3.1 Introduction

The definition of the term "mockup"[1] is: "a scale or full-size model of a design or device, used for teaching, demonstration, design evaluation, promotion, and other purposes". In other words, a mockup can be described as a functional representation of the reality of interest, used to quickly understand how it works. In software engineering and development, a mockup is often employed to build the front-end interfaces, that are used by the end user, to control the back-end software functionalities. Mockups are very useful to communicate:

- software design.
- provided functionalities
- interactions between UI components.

To create mockups there is a lot of software available both for desktop and online editing. In particular, using software for user interface (UI) mockups, you can design the software interface and see how it will appear in a specific device (e.g. desktop or mobile). Besides, you can even simulate it, by graphically defining the interaction between UI components, without having to write a single line of code. This approach follows the "Design with Data" paradigm, in which the user provides the data to be displayed in the UI and the software identifies, automatically, the best container component for the type of data provided. In this section, we describe the mockups realised for the AVIATOR platform.

3.3.2 First UI mockup

In Figure 3.4, is presented the first user interface mockup. As we can see, in the top-left side of the figure there is the name of the platform underlined, which act as a link to the home interface. Instead, in the top-right side of the figure, we can see the AVIATOR logo which provides a popup description when the user interacts with it. The most important part of this mockup regards the two rectangular boxes in the centre. These ones contain dropdown menus that allow the user to choose many parameters about the collection (document corpus, topic file and pool file) and other components of the IR process (stoplist, stemmer, retrieval model). Finally, the start button launches the process described in Figure 3.1

[1] https://en.wikipedia.org/wiki/Mockup

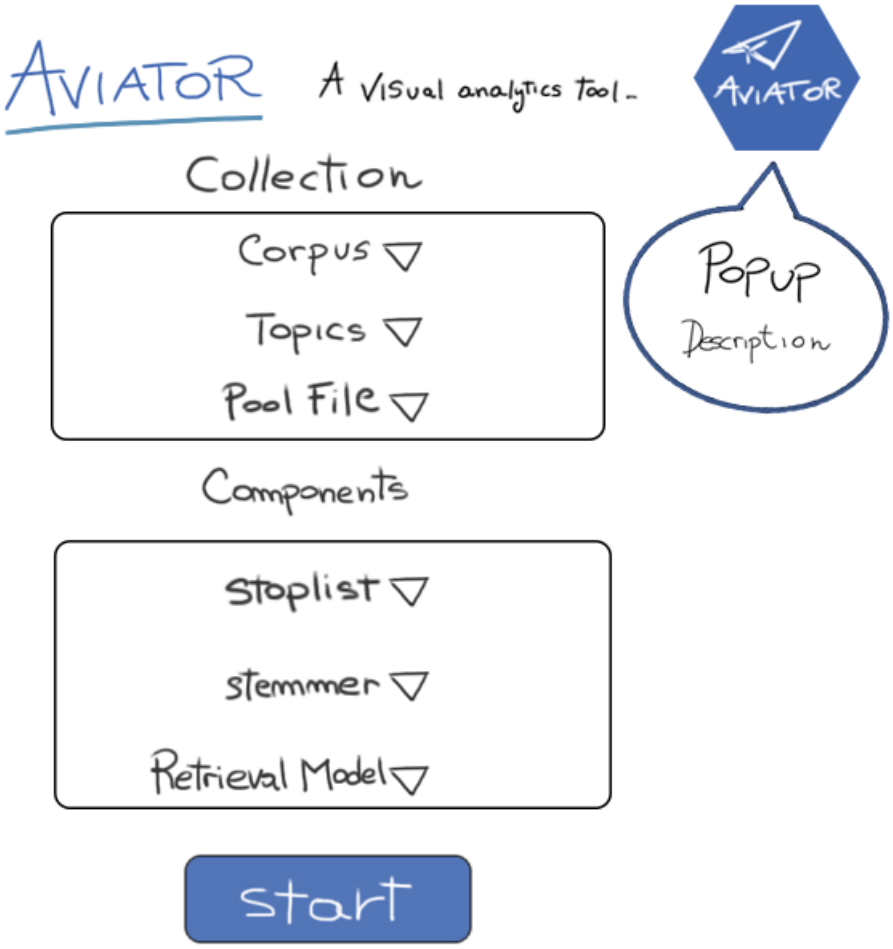

Figure 3.4: First UI mockup.

3.3.3 Second UI mockup

In Figure 3.5, is presented the second UI mockup. As we can see, the header of the user interface (platform name, logo and description) is shared by all the different UI mockups. In particular, this second mockup designs the "topic per topic analysis" done by means of an interactive scatter plot, in which each point represents a value for a specific evaluation measure. In the x-axis of the scatterplot is reported the topic, while in the y-axis there is the measure. The user can inspect each point, thus obtaining the information concerning the topic-measure pair. Besides, every retrieval model has a different colour, this helps the user in the exploration of the results. In the example reported in 3.5, we can see the information regarding the topic 352, which obtains a value, for the average precision (AP) measure, of 0,12. Obviously, we can change the measure just clicking in a dedicated button and the

scatter plot will be updated with the new data. Besides, we can also follow the progress, for the incremental indexing process, through the progress bar placed on the right side of the UI. This one shows the current percentage of indexed documents for the test collection. Finally, we can use the "Overall" button to switch to the dedicated interface, described in the third mockup.

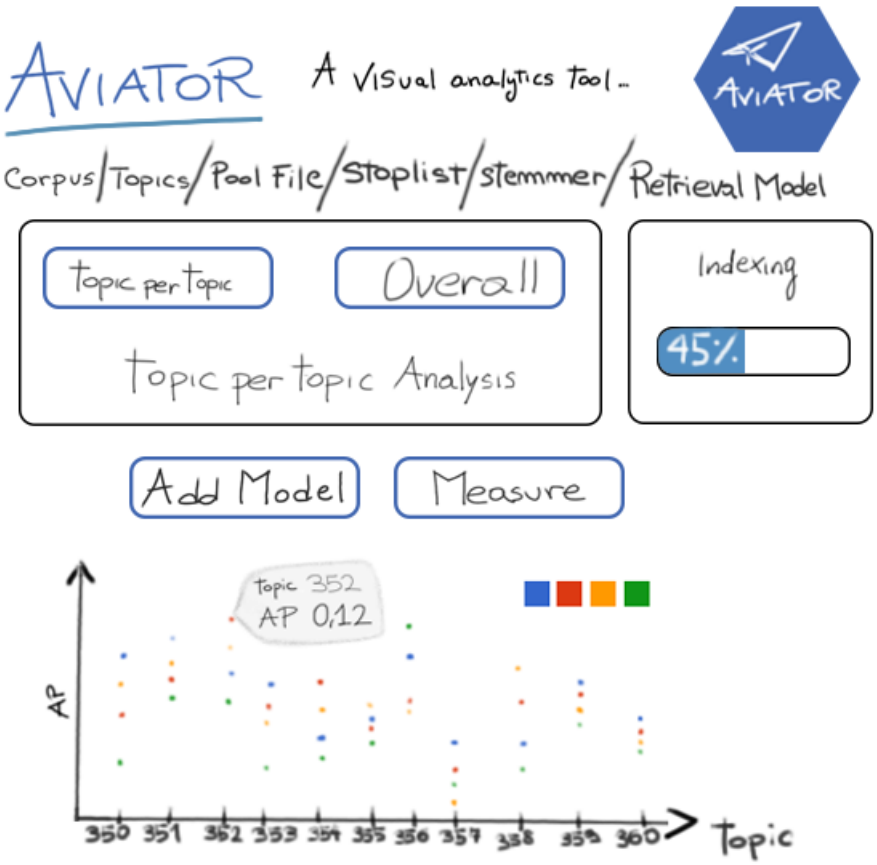

Figure 3.5: Second UI mockup.

3.3.4 Third UI mockup

In Figure 3.6, is presented the third UI mockup. As we can see, the interface is similar to the one of the second mockup. In particular, this third mockup designs the "Overall analysis" done by means of an interactive bar chart, in which each bar represents a value for a specific evaluation measure. In the x-axis of the bar chart is reported the retrieval model, while in the y-axis there is the measure. The user can inspect each bar, thus obtaining the information concerning the retrieval model-measure pair. As in the previous mockup, every retrieval

model has a different colour, this is useful to help users in the data exploration activity. In the example reported in Figure 3.6, there are four retrieval models, expressed in colours with: blue, red, yellow and green. In particular, if the user inspects the BM25 retrieval model, which in the example corresponds to the green colour, we can see a popup that shows the value for the evaluation measure considered (i.e. MAP). Besides, the progress bar, introduced in the second mockup, for monitoring the advances in the incremental indexing process, is included also in this mockup. Finally, we can use, as in the previous mockup, the "Topic per Topic" button to switch to the dedicated interface, described in the second mockup.

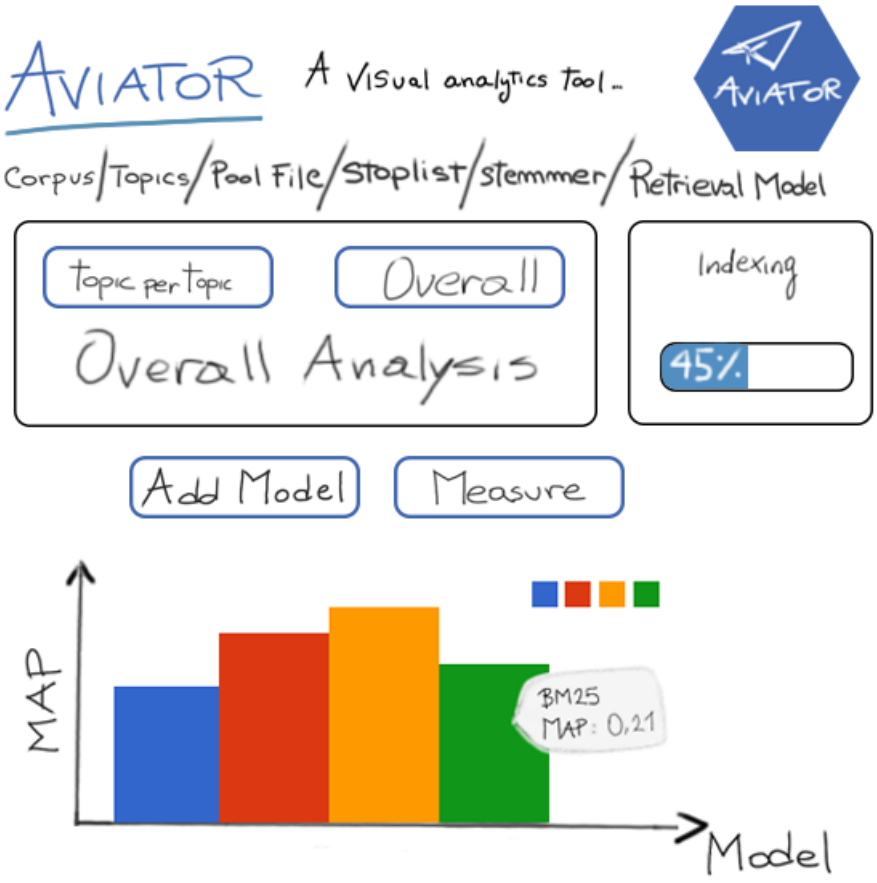

Figure 3.6: Third UI mockup.

3.3.5 Fourth UI mockup

In Figure 3.7, is presented the fourth UI mockup. As we can see, the interface is an improvement of the one presented in the second mockup. This fourth mockup designs the "topic per topic analysis", integrating new functionalities in the UI proposed in the second mockup.

In particular, the main feature added is the possibility of customising some retrieval model parameters. This involves an additional configuration of the search platform, that must be correctly set by AVIATOR, before the beginning of the retrieval process. In the example reported in Figure 3.7, there is a checkbox for the BM25 model. This checkbox can be used by a user for showing/hiding that model in the scatter plot. This feature is similar to one provided by the "Add Model" button, with a difference: when the checkbox is checked the user can change also some retrieval model parameters, like k_1 for the BM25 model in the figure. Using the dedicated slider, the user can change the value for k_1. Each time the value changes, a request for a new configuration is sent to the search platform which receives the request and performs the retrieval task, using the latest configuration requested. Later, when the evaluation phase is finished, the values are sent to the UI, thus the scatter plot is updated. The possibility of changing some parameters for a retrieval model is the only thing not implemented yet in AVIATOR. However, this feature will be realised in the future.

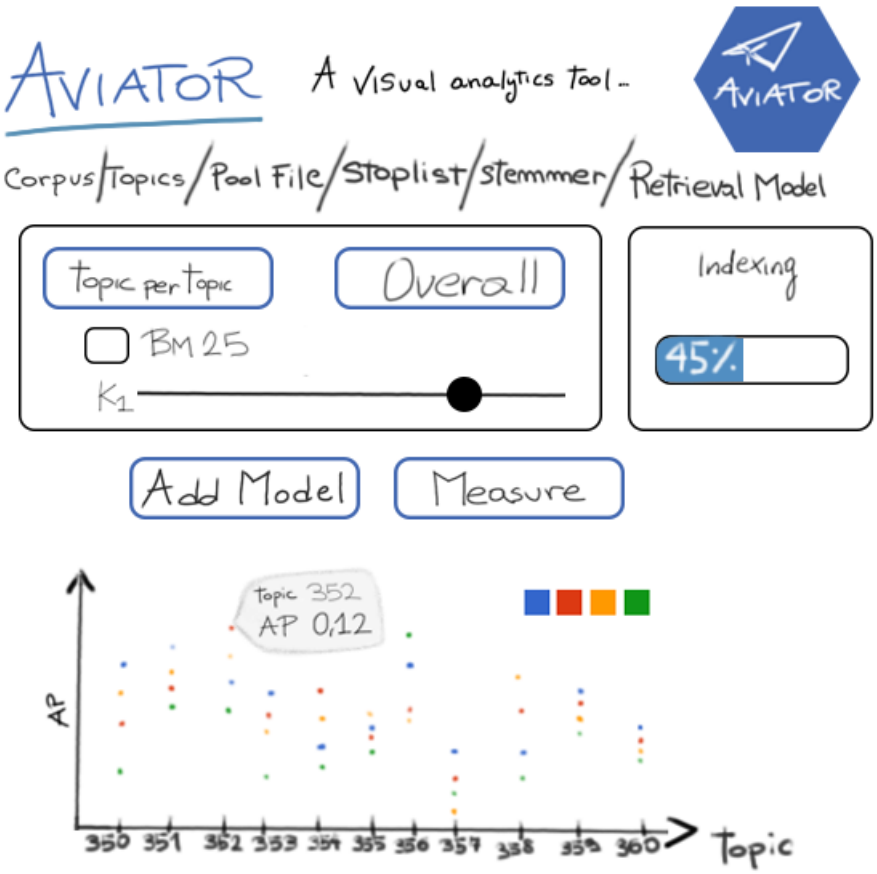

Figure 3.7: Fourth UI mockup.

3.4 Final remarks

In this chapter, we present the conceptual framework implemented in AVIATOR. In particular, we describe the mathematical notions necessary to understand the AVIATOR process. This one is composed of:

1. **Preprocessing**: takes the given input collection C and divides it into n buckets B_i of the bucket collection $\mathcal{B} = \left\{ B_1, B_2, \ldots, B_n \right\}$.
2. **Incremental Indexing**: indexes the bucket B_i at the i-th iteration.
3. **Retrieval**: for each topic $t \in T$ and retrieval model j, send a query to the IR system, which returns the list of the retrieved documents ordered by the relevance score, that is the run $R_j(t)$.
4. **Evaluation**: evaluate each run $R_j(t)$ using the ground truth (GT).

Therefore, we discuss the algorithms and the strategies used for the implementation of the AVIATOR process. Finally, we provide four mockups for the front-end visual analytics web interface.

CHAPTER 4

BACKEND

4.1 Introduction

Since AVIATOR is a full stack application, it is made of many different components that interact with each other to obtain and visualise useful evaluation data about IR systems. As every full stack application, AVIATOR has got a back-end for the data access layer and a front-end for the presentation one. We can define the notions of back-end and front-end [1] as follows:

- **Back-end**: indicates all the software parts that exhibit services and application functionalities, e.g. through the Application Programming Interface (API), that usually are not visible to the end-user. The back-end is strictly related to the data access layer because the provided functionalities need the input data to generate the expected outputs.

- **Front-end**: indicates all the software parts such that a user can interact with, e.g. the Graphical User Interface (GUI) or, in general, any software interface a user can see and use to send inputs, e.g. a command line.

In this chapter, we present and describe the AVIATOR back-end. As we said, the back-end functionalities regard accessing data, that, after some manipulations, are displayed to the user interface in the front-end. Since back-end and front-end are tightly

[1] https://stackoverflow.com/questions/18348612/

interconnected, to understand in depth how they work independently, we need to know how the AVIATOR architecture is organised. For this reason, in the following sections, first of all, we present the high-level AVIATOR architecture, after that, we describe in detail the back-end.

4.2 High-level architecture

AVIATOR is a client-server application that allows evaluating, one or more IR systems, on a given document collection $\mathcal{C}$. Since the purpose of this software is to help IR experts in evaluating IR systems, AVIATOR is strictly connected with the IRS to evaluate. It acts as a commander: it receives the user commands from the client interface and executes the dedicated procedures by automatically controlling the IRS for the task specified. In Figure 4.1, is reported the high-level client-server architecture adopted by AVIATOR.

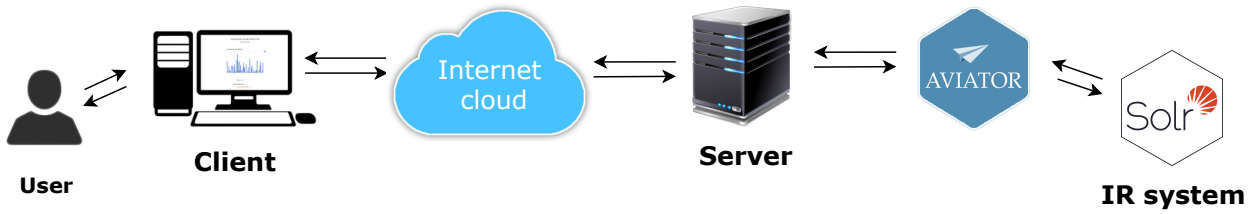

Figure 4.1: High level architecture: components, interactions.

As we can see, AVIATOR works on a server connected to the Internet, which can be remotely controlled by a user through a client web. This is a typical client-server architecture: the client sends a request for a resource or a service over a computer network, e.g. the Internet, and the server executes the right procedures to obtain the resource to send, as a response, to the client. According to this architecture, AVIATOR has a dedicated web server that receives the requests at a specific port and then executes the routine associated with the received request identifier. AVIATOR works as a wrapper for the IR system to evaluate: it can automatically control each stage of an IR process (e.g. indexing, retrieval, evaluation) with dedicated routines for every task to execute. These routines interface directly with the IR system, to perform the specific job for which are designed. The web server implemented in AVIATOR use the Representational State Transfer (REST[2]) architecture, which is an HTTP based web architecture that is designed for fast performance and to be reliable. Each resource or service is accessible through a unique Uniform Resource Locator (URL), according to the Uniform

[2]https://en.wikipedia.org/wiki/Representational_state_transfer

Resource Identifiers (URI) standard. Actually, using a REST web interface, we can do more than accessing resources, we can do any operation like create, read, update, and delete (CRUD). As shown in Figure 4.1 and 4.2, the IR system used is the Solr [3] search platform. The reason for this choice is that Solr is open source, stable and reliable since it is part of the Apache Lucene project. We describe it in detail in the dedicated section 4.3.

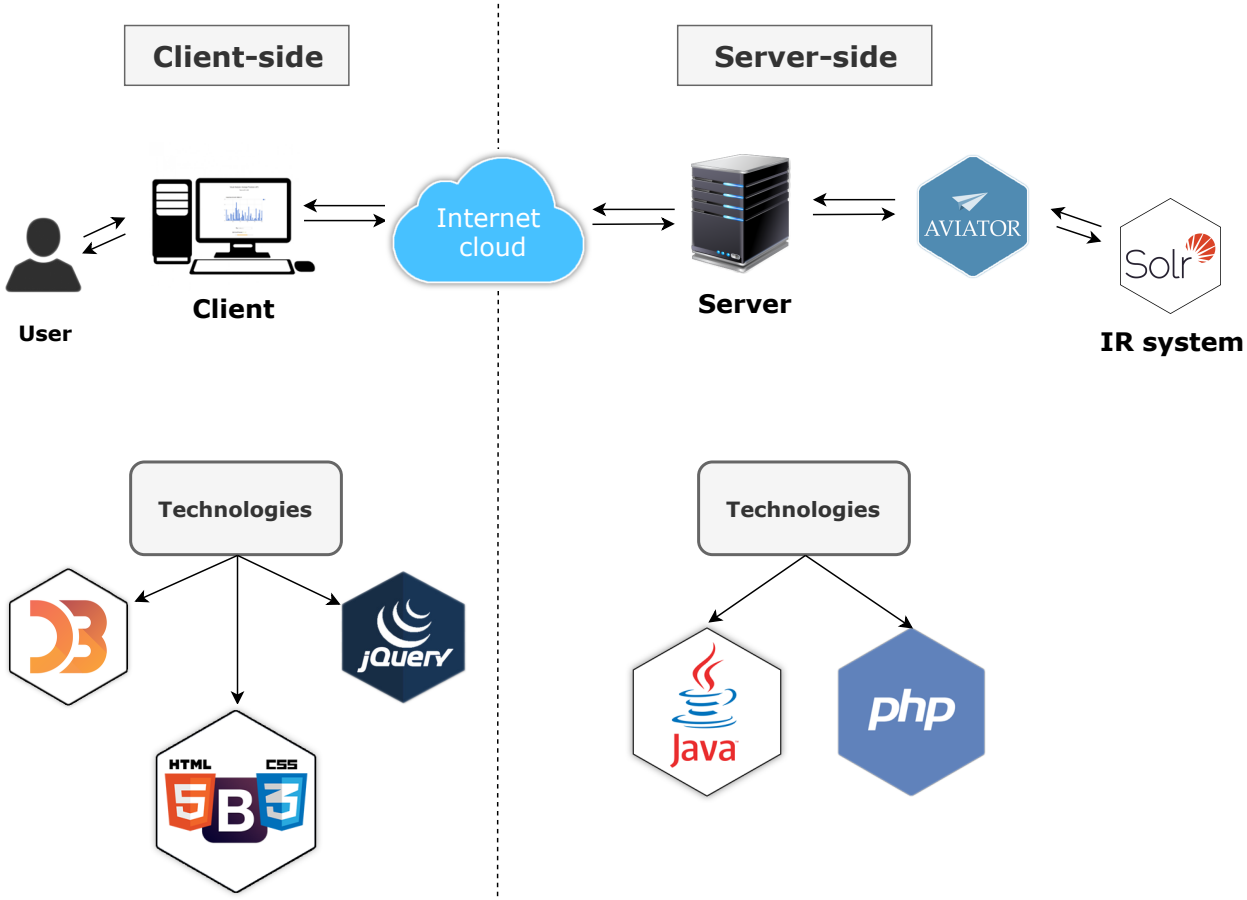

Figure 4.2: High level architecture: components, interactions and technologies.

In Figure 4.2, we can see both the *client-side*[4] and the *server-side*[5]. The first one regards the user and all the interfaces, i.e. web client, he can use to contact and interact with the AVIATOR web server situated on the server-side. An example of a client-side component is the web browser used by the user for connecting to the AVIATOR web interface. The server-side, instead, regards all the hardware and the software that work for satisfying the user request. On the server-side, there is the physical server on which operate both the AVIATOR and the

[3] http://lucene.apache.org/solr/
[4] https://en.wikipedia.org/wiki/Client-side
[5] https://en.wikipedia.org/wiki/Server-side

Solr platforms. Besides, in Figure 4.2 are reported the technologies employed to develop the different parts of AVIATOR. These technologies are divided according to the side on which are used. For example, in the server-side AVIATOR has been developed both in Java programming language and in Php scripting language. Let us briefly define all the technologies used in the client-side and the server-side. The technologies used in the client-side are:

- **D3.js**[6]: D3.js is a library, written in JavaScript[7] programming language, for data visualisation. By means of a powerful combination of visualisation components and a data-driven approach, this library allows analysing data in a fast and efficient way, through many different interactive charts. D3.js It is very useful in all the activities regarding data analysis since the same data can be quickly viewed using a different chart or visual component. For example, we can use D3 to produce an HTML table or, we can use the same data to create an interactive SVG bar chart with transitions and helpful animations. We used this library to develop all the interactive charts of the visual analytics web interface that makes up the AVIATOR front-end.

- **jQuery**[8]: "write less, do more" this is the slogan which present jQuery as a fast library for writing JavaScript code effectively and efficiently. This small library has become so popular since everyone knows JavaScript can use it to simplify tasks like document traversal, DOM manipulation and event handling. Finally, we can use it to make Ajax calls in a faster and concise way. This is the reason why we used it in AVIATOR. We used this library for creating all the interactive GUI components such as forms and input fields. Besides, we used jQuery for handling Ajax calls to the AVIATOR web server, which is placed on the server-side.

- **HTML5**[9]: is the fifth version of the Hypertext Markup Language (HTML), which is the standard markup language for developing web pages and web applications in general. Since HTML5 defines only the web page structure and its elements as static content, usually this markup language is combined with Javascript and the Bootstrap framework, to make web pages dynamic. This is the reason why in Figure 4.2 they are into the same hexagon. The structure of the AVIATOR web pages is described with the HTML5, which is the latest version available for this markup language.

[6] https://d3js.org/
[7] https://en.wikipedia.org/wiki/JavaScript
[8] https://jquery.com/
[9] https://it.wikipedia.org/wiki/HTML5

- **CSS3**[10]: is the third version of the Cascading Style Sheets (CSS), which is a style sheet language that defines how a document has to be displayed. CSS has been developed to separate the presentation layer from the content one. In this way, many different web pages can share the same style, saved in a separate ".css" file. Using CSS, we can specify, for example, the layout disposition, the colours of every page element and the dimensions of texts, images and other components. The style of the AVIATOR web pages is defined using custom CSS3 style sheets, that integrate the standard templates provided by Bootstrap.

- **Bootstrap**[11]: is an open source front-end framework to simplify web development with HTML, CSS and JavaScript. Bootstrap provides an exhaustive set of templates for any kind of web components such as buttons, menus and forms. All these components are realised according to the standards of the World Wide Web Consortium (W3C[12]). This means that the Bootstrap components are designed to be correctly visualised over all the most important browsers and devices, with the responsive grid layout system. This is the reason why, in the last years, it has become a standard framework for web development. The standard templates of Bootstrap have been used to develop the base of the AVIATOR front-end.

The technologies employed for AVIATOR in the server-side are:

- **Java**[13]: is one of the most popular object-oriented programming languages. Actually, according to TIOBE [14], Java is still the most popular programming language used in 2019. Java has become so successful because of its key strengths: simplicity, object-oriented, robustness, security, portable, threaded, and dynamic. This is the reason why nowadays Java runs on over 15 billion of different devices and its platform is the most employed for cloud development. We used Java for creating the AVIATOR back-end, which mainly consists of the AVIATOR web server. This one handles all the user requests by invoking the right routine, according to the received request.

[10] https://it.wikipedia.org/wiki/CSS
[11] https://getbootstrap.com/
[12] https://en.wikipedia.org/wiki/World_Wide_Web_Consortium
[13] https://en.wikipedia.org/wiki/Java_(programming_language)
[14] https://www.tiobe.com/tiobe-index/

Figure 4.3: AVIATOR: employed technologies.

- **PHP**[15]: is one of the most popular server-side scripting language specially designed for web development. In the beginning, PHP was the acronym for "Personal Home Page", but nowadays it stands for "Hypertext Preprocessor". This change reflects the close relation with HTML web pages that are the primary example of hypertext. The word "Hypertext" recalls that PHP is an HTML embedded scripting language: PHP scripts can be embedded into HTML pages. The purpose of this language is to develop dynamic pages, such that the content is established according to user inputs and preferences. The PHP reference site says: "Fast, flexible and pragmatic" these are only a few reasons for choosing the simplicity with which a blog, or even a popular website, can be created. We used PHP for creating the dynamic web pages of the AVIATOR front-end which a user interacts with.

[15]https://en.wikipedia.org/wiki/PHP

4.3 Apache Solr

Solr is one of the most popular enterprise search platforms and, for this reason, is broadly employed around the world for IR purposes. It is written on top of Lucene, a Java library for text searching, which is part of the Apache Software Foundation (ASF). Since it is based on Lucene[16], Solr is an open source project written in Java, part of the ASF too. There are many valid reasons to choose Solr for searching, but the most interesting features are:

- Full-text search.
- Faceted search.
- Hit highlighting.
- REST API.
- Web admin console interface.
- Support for geospatial search.
- Support for JSON, XML, TXT and many other output formats.
- Support for incremental and distributed indexing.
- Support for a cluster of Solr servers with SolrCloud.

Famous for being fast and scalable, Solr is also used by some famous companies and organizations like Apple, Instagram, Netflix, SourceForge and DuckDuckGo.

4.3.1 Solr functionalities

As we said, Solr provides a lot of useful functionalities but, for using it, we need to know how to handle and set the searching properties correctly. For this reason, in this section, we describe the Solr architecture and how it works. One of the most interesting features of Solr is the web interface that allows administrators to view Solr configuration details, run queries and analyse document fields; all these activities allow to tune a Solr configuration. In Figure 4.4 is reported the Solr administration dashboard. To access the Solr dashboard, first of all, we need to run the Solr launcher using the following command for Windows:

```
bin/solr.cmd start
```

[16] https://lucene.apache.org/core/

When Solr is started, we just need to go to: `http://hostname:8983/solr/`, which is the URL for accessing the Solr dashboard using a client web, e.g. a browser. For accessing Solr in a local machine, we can specify "localhost" as "hostname". Besides, the default port is 8983, but we can run Solr also in a different port. To do this on Windows, we have to specify the desired port when we run the Solr launcher, as follows:

```
bin/solr.cmd start -p 8080
```

In the dashboard we can see a lot of useful information such as the Solr version, both the physical and the JVM memory used, the number of processors available and much other information about the runtime environment. By default, the JVM memory available is 512 MB, but for our purposes, this limit is too low. For this reason, we raised the limit to 1024 MB. In this way, we can use more Solr core at the same time.

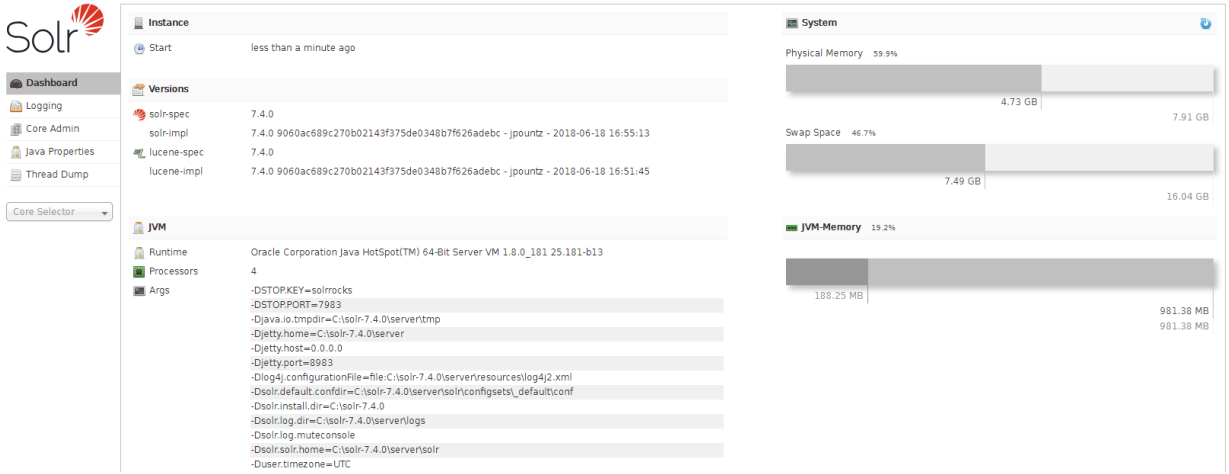

Figure 4.4: Solr dashboard.

We can highlight in red the sections reporting the information discussed so far, as reported in Figure 4.5. This one shows the information regarding the DELL Latitude E6230 on which the tests have been done. Looking at the figure below, we could think that 512 MB should be enough for work, but actually, even if memory usage is optimized when Solr starts, it loads all the Solr cores available, thus, if the number of cores to load is high, the memory required is greater than 512 MB. This justifies the new limit of 1024 MB set. On the left side of the window, there is the Solr menu from which we can view the log files, administrate the Solr cores and manage the Java properties. If we click in the "Core Admin" voice, indicated by the red arrow, we can access the Solr core admin interface reported in Figure 4.6.

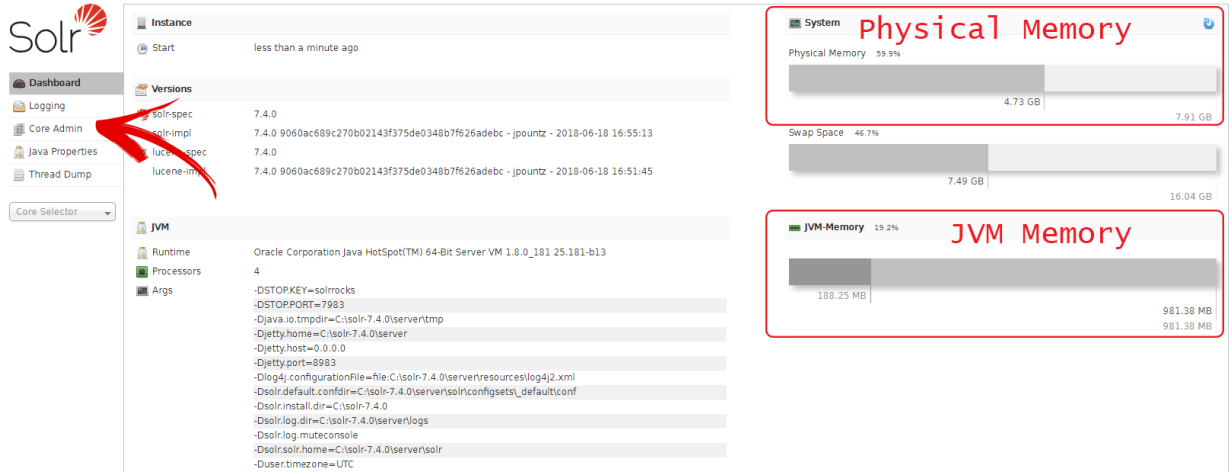

Figure 4.5: Solr dashboard detailed.

In Figure 4.6, are reported the information regarding the Solr core indicated by the red arrow, which is called: "TIPSTER_TERRIER_HUNSPELLSTEM_P100". In the two red boxes, we can see the information regarding this core, which is the core indexed using the Terrier[17] stoplist and the Hunspell stemming algorithm. The meaning of "P100" is that we index the 100% of the TIPSTER document corpus. Indeed, in the "Core index info" box we can see that the number of indexed documents is over 528 000, which corresponds to the size of the TIPSTER corpus.

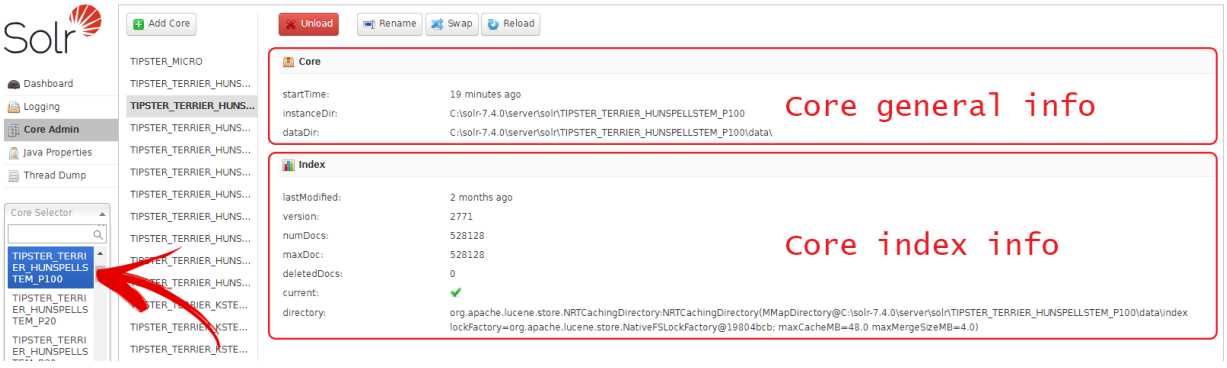

Figure 4.6: Solr core admin interface.

[17] http://terrier.org

In Figure 4.7 is shown the Solr interface for querying a specific core. On the left side, there are the input fields that allow specifying which document fields are used for searching in. Besides, on the same side, there is a field for the query text. On the right side, instead, is shown the list of the documents retrieved for the submitted query, ordered by the relevance score. This one depends on the retrieval model chosen, for example in Figure 4.7 the retrieval model used is the Dirichlet language model. In the following example, the query submitted to the Solr core "TIPSTER_TERRIER_HUNSPELLSTEM_P100" is "National Park", for which Solr has found 151 265 possible relevant documents. As reported in Figure 4.7, the response format is JSON[18] but many others are available e.g. XML and TXT. Besides, the JSON response contains an array of JSON object such that each object is a document with the values for the fields selected on the left side of the interface. In the example, the first two documents returned for the query "National Park" are identified respectively with "DOCNO": "LA040989-0218" and "LA052790-0064". In particular, the first one is about the Yellowstone National Park which matches perfectly with the submitted query.

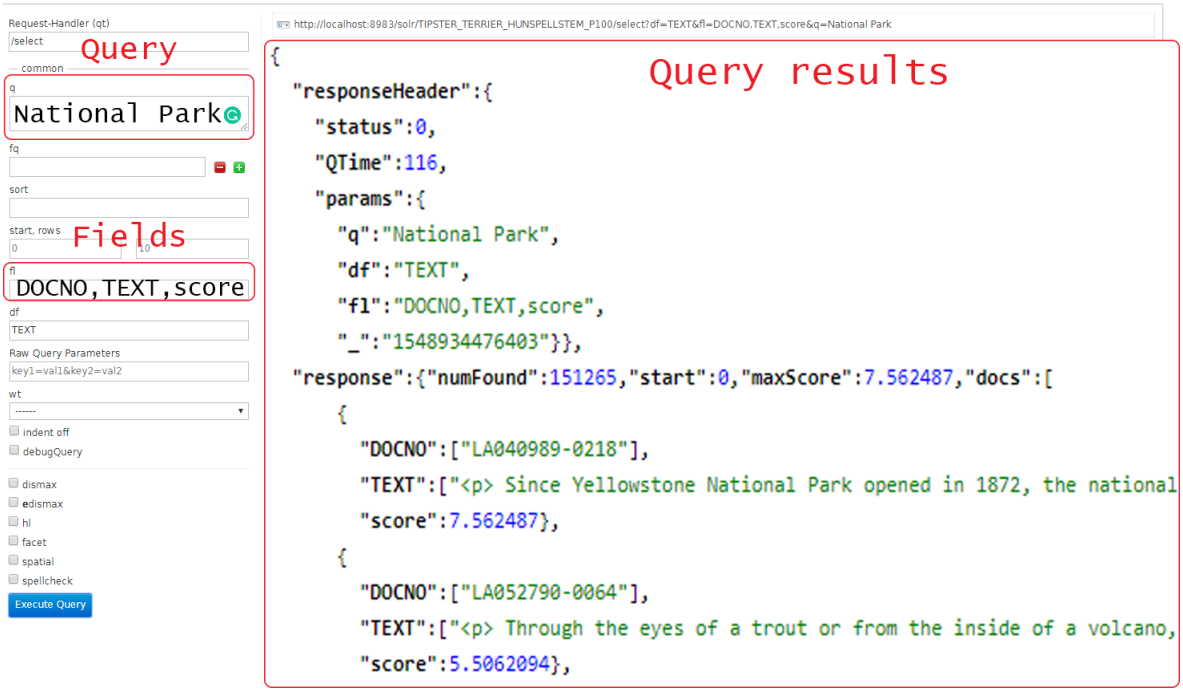

Figure 4.7: Solr query interface.

[18] https://www.json.org/

4.3.2 Aviator and Solr

As we said, Solr is the IR system used by AVIATOR both for the incremental indexing and the retrieval process. To perform these activities AVIATOR has a web server, written in Java, that receives the requests from the front-end web interface. It parses each request and then calls the right AVIATOR routine for the service requested. These routines automatically interact with Solr to satisfy the user request. The interaction between AVIATOR and Solr can be achieved in different ways, the most commons are by means of :

1. Solr REST APIs: in this way, all the activities related to the retrieval process are done. For example, to obtain the results for the query "National Park", reported in Figure 4.7, we can use this URL: `http://localhost:8983/solr/TIPSTER_TERRIER_HUNSPELLSTEM_P100/select?df=TEXT&fl=DOCNO,TEXT,score&q=NationalPark`.

2. AVIATOR functions: these are ad-hoc Java functions made for specific tasks, that could not be done through the Solr REST APIs. These functions are used, for example, for cloning a Solr core with the related configurations or even to reload the Solr server, since there is no way to do that using Solr REST APIs in the 7.4 version used for this thesis.

For every Solr core, there is a dedicated directory in which are saved all the information concerning the core, such as the configuration files and the compressed index. One of the most important configuration files is the "managed-schema.xml" or just "schema.xml". This one is an XML file in which are saved all the settings for the related core, such as the retrieval model, the stoplist and the stemmer used. An example of this file is reported in Figure 4.3.2, where we can see:

1. The retrieval model chosen, also known as "similarity retrieval model" or just "similarity", which in the figure example is the Dirichlet language model.

2. The list of stop words used for the indexing process, which is specified in the "stopwords.txt" file. The same stoplist can be used also for queries.

3. The stemming algorithm, which in the example is the Porter Stemmer, indicated with "PorterStemFilterFactory". As for the stoplist, even the stemmer can be used for queries.

4. The "LowerCaseFilterFactory" filter, which converts any uppercase alphabetic characters in the lowercase equivalent. This is a very common filter that is used both during the indexing process and for removing uppercase characters from user queries.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Solr managed schema -->
<schema name="default-config" version="1.6">
<!-- START SIMILARITY -->
<similarity class="org.apache.lucene.similarities.LMDirichletSimilarity">
</similarity>
<!-- END SIMILARITY -->
<uniqueKey>id</uniqueKey>
<fieldType name="ancestor_path" class="solr.TextField">
<analyzer type="index">
<tokenizer class="solr.KeywordTokenizerFactory"/>
</analyzer>
<analyzer type="query">
<tokenizer class="solr.PathHierarchyTokenizerFactory" delimiter="/"/>
</analyzer>
</fieldType>
<fieldType name="text_general" class="solr.TextField">
<!-- START INDEX SETTINGS-->
<analyzer type="index">
<tokenizer class="solr.StandardTokenizerFactory"/>
<filter class="solr.StopFilterFactory" words="stopwords.txt"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.PorterStemFilterFactory"/>
</analyzer>
<!-- END INDEX SETTINGS -->
<!-- START QUERY SETTINGS -->
<analyzer type="query">
<tokenizer class="solr.StandardTokenizerFactory"/>
<filter class="solr.StopFilterFactory" words="stopwords.txt"/>
<filter class="solr.SynonymGraphFilterFactory" synonyms="synonyms.txt"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.PorterStemFilterFactory"/>
</analyzer>
<!-- END QUERY SETTINGS -->
</fieldType>
</schema>
```

Figure 4.8: Solr managed schema.

Since the purpose of this thesis is developing a visual analytics software capable of helping IR experts in the evaluation of IR systems, we have generated a lot of evaluation data for each combination of stoplist, stemmer and retrieval model chosen. In other words, given the document corpus of the TIPSTER collection, for each combination of stoplist, stemmer and indexing percentage we create a core. The set of stoplist adopted are:

- **indri.txt**: this is the stoplist file which contains the stop words used by the Indri[19] IR system.
- **lucene.txt**: this is the default stoplist provided in Solr, inherited by the Apache Lucene project.
- **nostop.txt**: this is an empty stoplist, useful for evaluating an IR system which does not perform the stop words removal phase.
- **terrier.txt**: this is the stoplist file which contains the stop words used by the Terrier IR system.

The set of stemming algorithm, or stemmer, adopted are:

- **Hunspell Stemmer**: this is a dictionary based stemmer that provides support for many different languages. To work it requires a dictionary file ".dic" and a rule file ".aff" for each language we want to use.
- **Krovetz Stemmer**: this is an alternative to the Porter Stemmer, since it is less aggressive than the Porter Stemmer. The Krovetz Stemmer, also known as KStem, was written by Bob Krovetz, hence its name. This stemmer is only appropriate for the English language.
- **Porter Stemmer**: this is a stemmer coded directly in Java and is not based on Snowball, even if the results are similar. It is only appropriate for English language text. However, it has been benchmarked as four times faster than the English Snowball stemmer, so can provide a performance enhancement.
- **No Stemmer**: this option is useful for evaluating an IR system which does not perform the stemming phase.

[19] https://www.lemurproject.org/indri/

The set of retrieval models adopted are:

- **BM25**: this is a probabilistic model based on the probabilistic retrieval framework developed by Stephen E. Robertson and Karen Spärck Jones[8], between the 1970s and 1980s. This one is considered one of the most successful retrieval model algorithms [9]. In the Solr platform, this model is the default retrieval model for cores.
- **Boolean**: this is the retrieval model based on the boolean logic (TRUE or FALSE). Every time a user submits a query to an IRS that adopts the boolean model, there are only two possible results for a document: relevant (TRUE) or not relevant (FALSE). It was one of the first retrieval models to be adopted, but nowadays there are a lot of retrieval models that work better.
- **Dirichlet Language Model**: this is the retrieval model based on the Bayesian smoothing using Dirichlet priors. We chose to include this model in our tests since its performances are often comparable with the BM25 ones.
- **TF IDF**: this is the retrieval model based on the vector space model, proposed by Gerard Salton in [4] and in the 1970s, it was one of the most important points of reference for the research in the field of information retrieval. The vector space model implements the term weighting and allows to realise the sorting of the list of the relevant judged documents. In the Solr platform, this model is called "Classic Similarity" due to its historical importance, before BM25 it was the default retrieval model for Solr cores.

With 4 different stoplists (included the empty one), 4 stemming algorithms (included the "No Stemmer"), 4 retrieval models and 10 indexing percentages, the number of Solr core generated is:

$$4 \times 4 \times 4 \times 10 = 640 \quad \textit{Solr cores}$$

These 640 Solr cores correspond to 230 GB of indexing data, which is more than half the capacity of a standard SSD of 512 GB.

4.4 Backend functionalities

The most important part of the AVIATOR platform is the AVIATOR web server. This one is an HTTP REST server written in Java using sockets and other classes from the Java's network library. The AVIATOR web server receives the requests from the front-end web interface and, after a parsing phase, it calls the right Java routine for the service requested. A request, to be accepted by the web server, needs to be formatted using the API schema defined for the provided REST services. Indeed, the requests that do not respect the API schema defined for URL are rejected. We can summarise the major AVIATOR services as follows:

- **Start**, **stop** and **restart** the **Solr server**. These activities can be done just using the following URLs:
  - Start Solr: "`http://localhost:3000?command=startsolr`"
  - Stop Solr: "`http://localhost:3000?command=stopsolr`"
  - Restart Solr: "`http://localhost:3000?command=restartsolr`"

  It is important to notice that the value "localhost" for the hostname is only for testing purposes. Besides, the default port is 3000 but this value can be chosen using the AVIATOR configuration interface.

- **Preprocessing** the document collection: this service regards the creation of the buckets of the bucket collection $\mathcal{B}$ as described in the conceptual framework of Chapter 3. This can be done using the following URL:
  "`http://localhost:3000?command=preprocessing&collection=TIPSTER`"
  Where "TIPSTER" is the name of the collection to preprocess.

- **Index** a collection: this command creates the index for the collection specified in a Solr core with the same name. The URL for this API service is:
  "`http://localhost:3000?command=index&collection=TIPSTER`"
  Where "TIPSTER" is the name of the collection to index.

- **Index** an entire collection specifying the Solr core to use. This is similar to the previous command with the only difference that we can specify the Solr core to use by providing the name. The URL for this API service is:
  "`http://localhost:3000?command=indexCustomCore&collection=TIPSTER&coreName=C1`"

Where "TIPSTER" is the name of the collection to index and "C1" the name of the Solr core.

- **Create**, **delete** and **reload Solr core**. These activities can be done just using the following URLs:
  - Create a new Solr core: "`http://localhost:3000?command=createSolrCore&coreName=TIPSTER`"
  - Delete an existing Solr core: "`http://localhost:3000?command=deleteSolrCore&coreName=TIPSTER`"
  - Reload an existing Solr core: "`http://localhost:3000?command=reloadSolrCore&coreName=TIPSTER`"
- Set the **stoplist** for a Solr core. The URL for this API service is:
  "`http://localhost:3000?command=setStoplist&coreName=TIPSTER&stoplist=terrier.txt`"
  In this example URL, "TIPSTER" is the Solr core name and "terrier.txt" is the stoplist chosen.
- Set the **stemmer** for a Solr core. The URL for this API service is:
  "`http://localhost:3000?command=setStemmer&coreName=TIPSTER&stemmerName=PorterStemFilterFactory`"
  In this example URL, "TIPSTER" is the Solr core name and "PorterStemFilterFactory" indicates that the stemming algorithm chosen is the Porter Stemmer.
- Set the **similarity** for a Solr core. The URL for this API service is:
  "`http://localhost:3000?command=setSimilaritySchema&coreName=TIPSTER&similarity=BM25Similarity`"
  In this example URL, "TIPSTER" is the Solr core name and "BM25Similarity" is the similarity option for the BM25 retrieval model.
- Create an AVIATOR **job**. This service creates a new AVIATOR job which performs the process described in Figure 3.1 of Chapter
  refchap:Conceptual Framework. This process starts with the preprocessing of the selected collection and then continue with the loop composed of three phases: incremental indexing, retrieval and evaluation. The URL for creating a new AVIATOR job is:
  "`http://localhost:3000?command=createAviatorJob&collection=TIPSTER&topicFile=TIPSTER_351_400&poolFile=QREL_TREC7&JID=TestJID&similarity=ClassicSimilarity&`

`stemmer=PorterStemFilterFactory&stoplist=lucene.txt"`

Where "TIPSTER" is the name of the document collection, "TIPSTER_351_400" is the name of the TREC topic file, "QREL_TREC7" is the pool file adopted, "TestJID" is the job identifier, "ClassicSimilarity" is the option for the TFIDF retrieval model, "lucene.txt" is the chosen stoplist and "PorterStemFilterFactory" is the option for Porter Stemmer algorithm.

We can configure the AVIATOR web server using the configuration interface reported in Figure 4.9. This interface has been developed using Java Swing, which is a library designed for graphical user interface (GUI). The configuration interface uses a minimalist design which consists of a dropdown menu for the settings, a central button to start and stop the AVIATOR web server and a text field for displaying useful information. In particular, in Figure 4.9, this field report the message: "Server is OFF" which indicates that the AVIATOR web server is not active. But if we press the button in the middle of the GUI the server turns active on to the specified port. When this occurs the text field shows the message: "Server is ON" and the button label changes from "Start Server" to "Stop Server". This behaviour is shown in Figure 4.10.

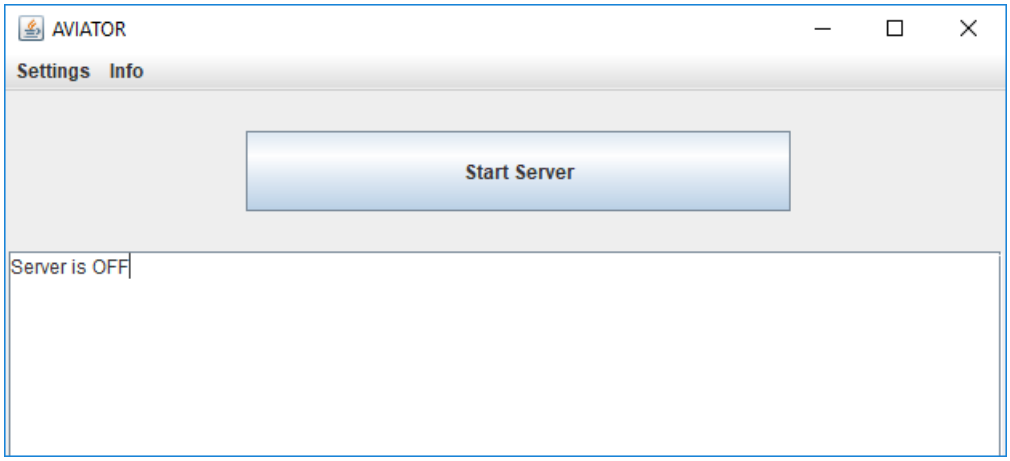

Figure 4.9: AVIATOR web server configuration interface: Server OFF.

According to Figure 4.11, from the "Settings" menu the parameters we can change are:

- The AVIATOR web server port.
- The collections settings such as name, path and type (TREC, PDF, TXT). In particular, these parameters can be set using the interfaces reported in Figure 4.12 and 4.13.
- The root directory of Solr.

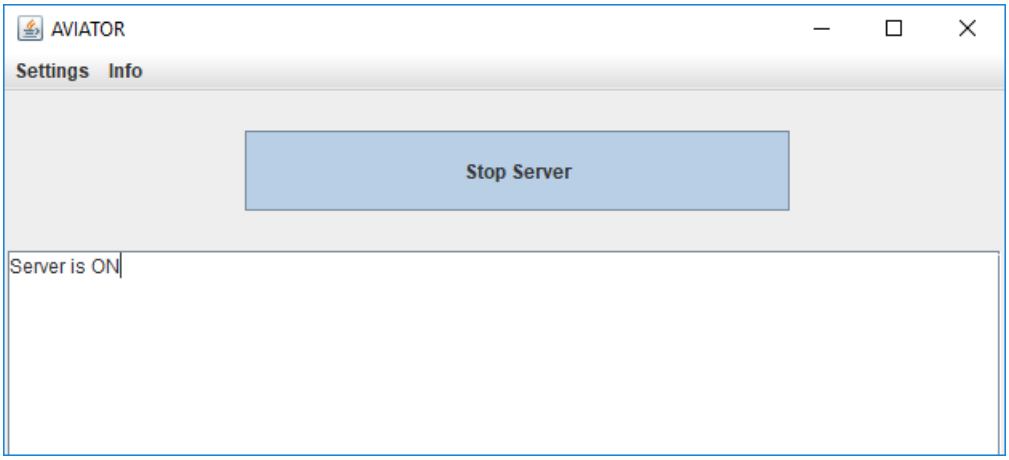

Figure 4.10: AVIATOR web server configuration interface: Server ON.

- The pool files settings, i.e. name and path of each pool file.
- The topic files settings, i.e. name and path of each topic file.
- The root directory of TREC eval, which is the evaluating tool used to evaluate every run file.
- The workspace directory, which is the directory used by AVIATOR to save its data, e.g. the preprocessed collection.

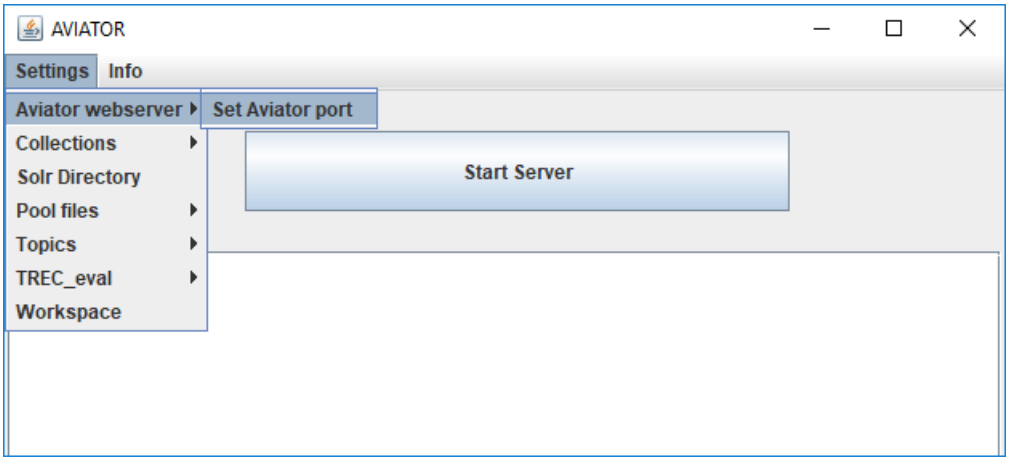

Figure 4.11: AVIATOR web server configuration interface: settings menu.

The following Figures 4.12 and 4.13 show how we can select a collection and change its properties, i.e. name, type and root directory.

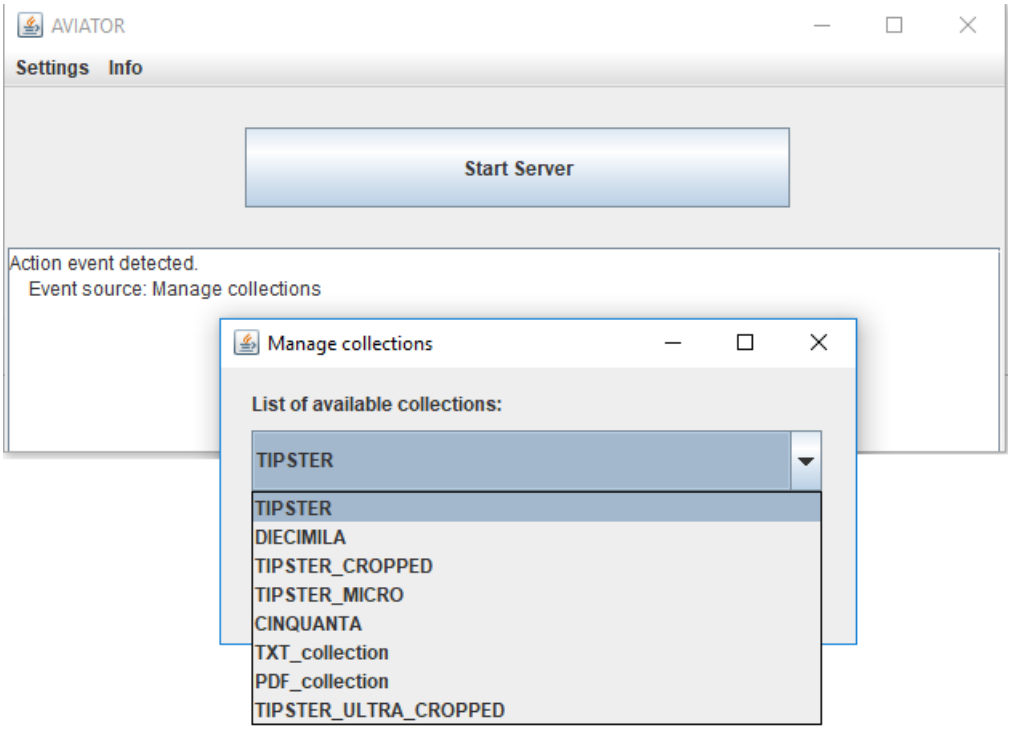

Figure 4.12: AVIATOR web server configuration interface: manage collections.

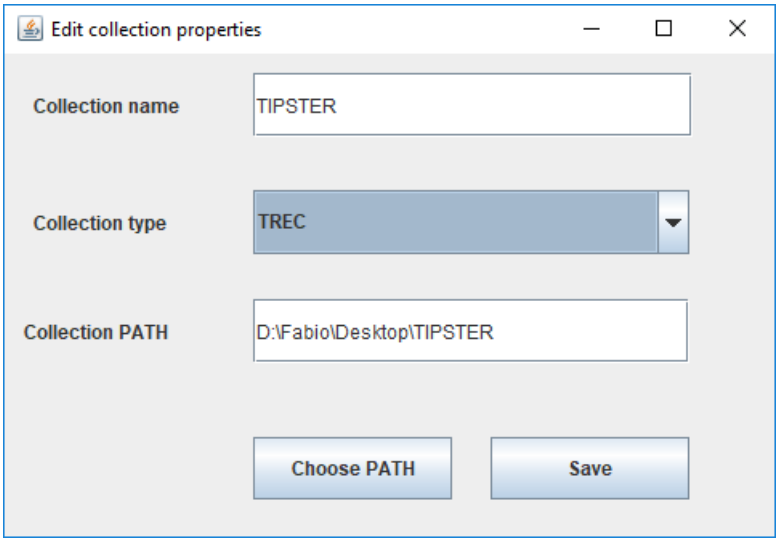

Figure 4.13: AVIATOR web server configuration interface: edit collection properties.

4.5 Final remarks

In this chapter, we define both the back-end and the front-end notions for a full stack application. We present the AVIATOR back-end in terms of technologies used and provided functionalities. Besides, we discuss the AVIATOR client-server architecture: the components, the interactions and the technologies adopted both for the client-side and the server-side. Therefore, we describe the Solr search platform, its functionalities and the integration with AVIATOR. Moreover, we present the back-end configuration tool developed to customize all the preferences and parameters for the correct functioning of AVIATOR.

CHAPTER 5

FRONTEND

In Chapter 4 we defined the notions of back-end and front-end, which are the fundamental components of every full-stack application, e.g. AVIATOR. This division, between back-end and front-end, aims to separate the data access layer from the presentation one. In particular, we can identify the front-end as the part of a computer system or application with which the user interacts directly. Therefore, every user interface is a front-end part designed for using a specific back-end service. In other words, this means that the front-end interfaces are designed to fit the provided functions of the back-end. For this reason, the front-end of an application is usually developed after the back-end. Nowadays, the separation between data and presentation has become not just a "best practice", it is a requirement for developing on many platforms such as Android and IOS. In particular, both Android and IOS require to use a specific design pattern called Model-View-Controller (MVC). This pattern assigns every object one of these three application roles:

- **Model**: this type of object is responsible for managing the data structures independently from the user interface. A model object might represent the text of an email message or the account information of a user profile.
- **View**: this type of object is usually displayed in the front-end interfaces so that the user can see it. An example of a view object is a scatter plot or a bar chart which both may refer to the same data model object.
- **Controller**: A controller object acts as an intermediary between both view and model objects. A controller object receives the user inputs and coordinates the application's

tasks to obtain the data to show, using view objects, in the front-end user interface. Besides, controller objects manage the life cycles of other objects.

The Model-View-Controller (MVC) design pattern architecture is shown in Figure 5.1.

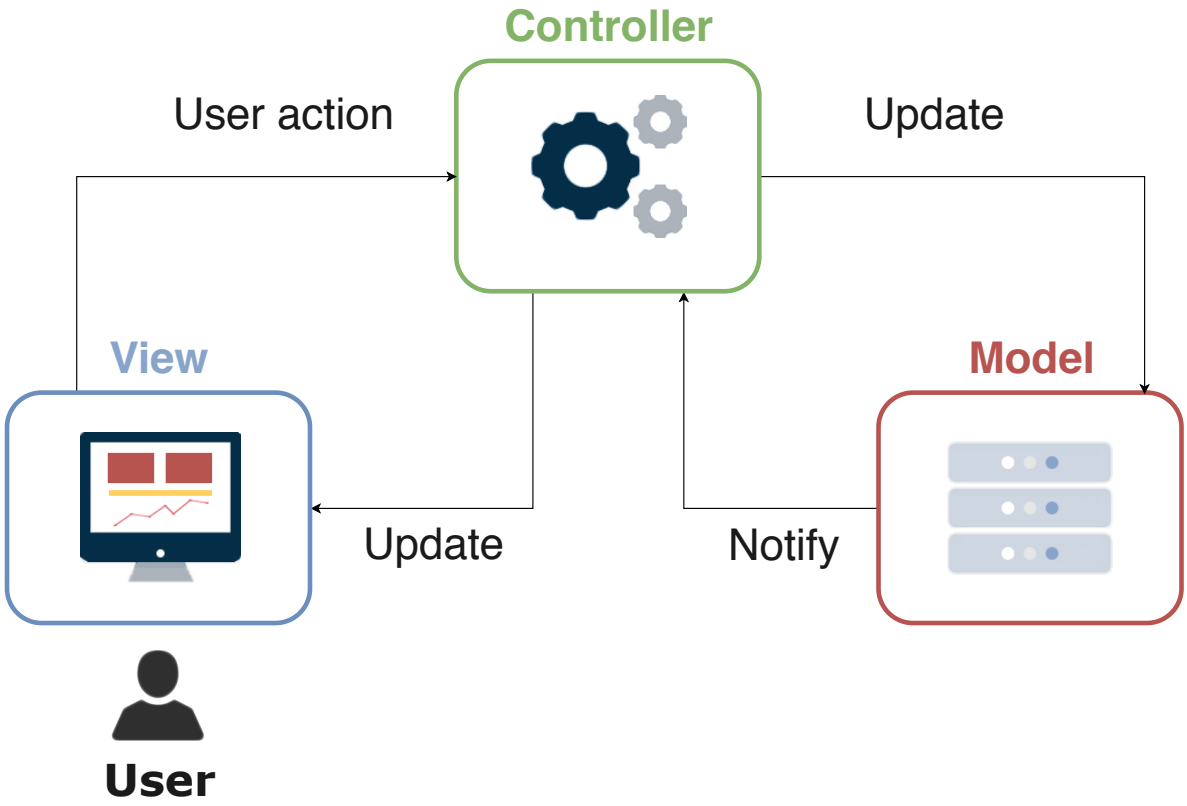

Figure 5.1: MVC architecture.

The Figure 5.1 shows how an MVC architecture works. First of all, the view receives the user actions and sends these inputs to the controller. This one executes all the necessary computation to update the model and, in turns, the view. AVIATOR uses MVC to separate the data access layer from the presentation one. In particular, all the data that come from the Solr search platform corresponds to the model, the AVIATOR web server acts as the controller and the web visual analytics interface represents the view. Besides, all the view objects belong to the front-end, which is updated by the controller every time data are available. In this context, the AVIATOR scatter plots and bar charts are view objects and the web server, with the user's authorization, update them every time the latest evaluation data are available. The AVIATOR front-end consists of the web interfaces that allow IR experts to control how the evaluation measures change, while the incremental indexing process advances in the background.

The technologies used for the front-end development are: D3.js, JavaScript, Bootstrap, HTML5, CSS3, jQuery and PHP. In particular, D3.js has been used for creating all the AVIATOR charts such as scatter plots and bar charts. To make all the interactive components of web pages JavaScript and jQuery have been used. The structure and the style of every web page have been defined using HTML5, CSS3 and the Bootstrap framework. Besides, PHP has been used for creating dynamic web pages capable of receiving HTTP requests with both POST and GET methods, as a result of user inputs.
The Figure 5.1 shows how an MVC architecture works. First of all, the view receives the user actions and sends these inputs to the controller. This one executes all the necessary computation to update the model and, in turns, the view. AVIATOR uses MVC to separate the data access layer from the presentation one. In particular, all the data that come from the Solr search platform corresponds to the model, the AVIATOR web server acts as the controller and the web visual analytics interface represents the view. Besides, all the view objects belong to the front-end, which is updated by the controller every time data are available. In this context, the AVIATOR scatter plots and bar charts are view objects and the web server, with the user's authorisation, update them every time the latest evaluation data are available. The AVIATOR front-end consists of the web interfaces that allow IR experts to control how the evaluation measures change, while the incremental indexing process advances in the background. The technologies used for the front-end development are: D3.js, JavaScript, Bootstrap, HTML5, CSS3, jQuery and PHP. In particular, D3.js has been used for creating all the AVIATOR charts such as scatter plots and bar charts. To make all the interactive components of web pages JavaScript and jQuery have been used. The structure and the style of every web page have been defined using HTML5, CSS3 and the Bootstrap framework. Besides, PHP has been used for creating dynamic web pages capable of receiving HTTP requests with both POST and GET methods, as a result of user inputs. Figure 5.2 shows the first AVIATOR web interface where a user can choose the parameters for the process described in Figure 3.1. These parameters are:

- **Corpus**: This is the corpus of the document collection. For the testing purposes of this thesis, the TIPSTER corpus has been chosen. The version used in this thesis consists of the disks 4 and 5 of the TREC TIPSTER project, without Congressional Record. In general, the AVIATOR platform works with any text document corpus, for example, we can provide as input a collection of TXT, PDF, HTML and XML too. Obviously, for evaluating it over a set of topics, contained in a topic file, we need to provide an appropriate pool file. Since for every TREC document collection, there are both a topic and a pool files available we chose the TIPSTER corpus to simplify the whole process

and to obtain comparable evaluation results.

- **Topics**: The user can choose the set of topics used for the evaluation process. The default set available is the TREC7 one, which contains the topics from 351 to the 400, for a total of 50 topics.
- **Pool file**: The user can choose the pool file used for the evaluation process. The default pool file available is the TREC7 one, which is combined with the TREC7 topic file.
- **Stoplist**: This is the list of stop words that are removed during the indexing process. The user can choose one of the four stoplists available:
  - **Indri**: this is the stoplist file which contains the stop words used by the Indri IR system.
  - **Lucene**: this is the default stoplist provided in Solr, inherited by the Apache Lucene project.
  - **No Stoplist**: this is an empty stoplist, useful for evaluating an IR system without performing the stop words removal phase.
  - **Terrier**: this is the stoplist file which contains the stop words used by the Terrier IR system.
- **Stemmer**: this is the stemming algorithm used during the indexing process. The user can choose one of the four stemmers available:
  - **Krovetz Stemmer**: this is an alternative to the Porter Stemmer available only for the English language. When it is necessary a less aggressive stemmer this is a good choice.
  - **Porter Stemmer**: this is one of the most famous stemming algorithms. It is very fast but, unfortunately, it is available only for the English language.
  - **Hunspell Stemmer**: this is a dictionary based stemmer available in many languages. For working it requires a dictionary and a set of stemming rules, provided by two separated text files.
  - **No Stemmer**: using this option the user can choose to not perform the stemming phase.

- **Retrieval model**: this is the retrieval model, also known as "similarity", used for the retrieval task. The user can choose one of the four retrieval models available:
  - **BM25**
  - **Boolean**
  - **Dirichlet Language Model**
  - **TF IDF**

In Figure 5.2, we can see that when the mouse pointer goes hover the AVIATOR logo, a short description of the system appears. Besides, a user can start a new AVIATOR job, by pressing the "Start" button.

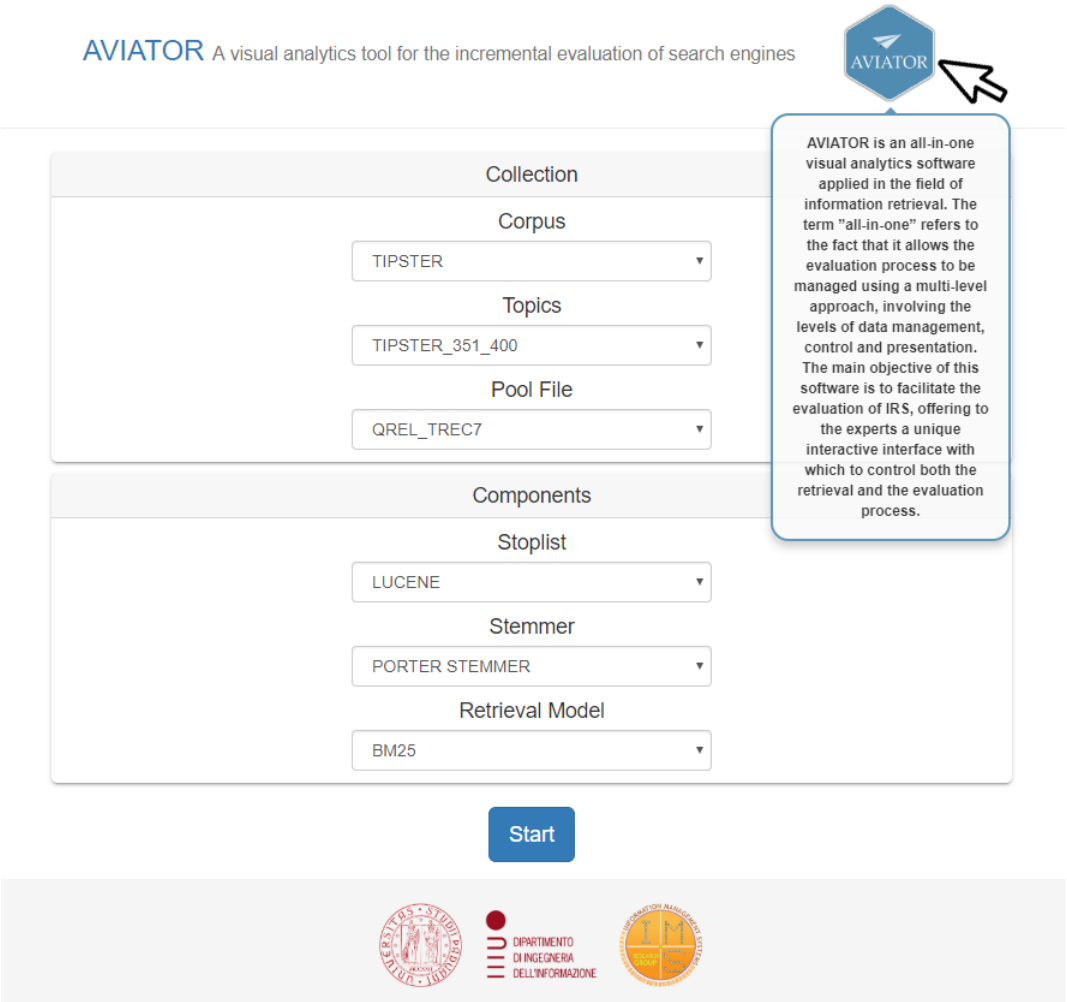

Figure 5.2: AVIATOR web user interface: homepage.

Figure 5.3 shows the AVIATOR visual analytics interface during the topic per topic analysis. This kind of analysis allows IR experts to see the values, of a chosen evaluation measure, for every topic $t \in T$. This is possible by means of a scatter plot in which every point represents the value, for the considered evaluation measure, obtained by a retrieval model for a specific topic. To easily distinguish the corresponding retrieval model of each point, a different colour and symbol are assigned to each retrieval model. The user can change the evaluation measure using the dedicated dropdown menu. An exhaustive set of measures is available for data exploration, e.g. Average Precision (AP), Mean Average Precision (MAP) and many others.

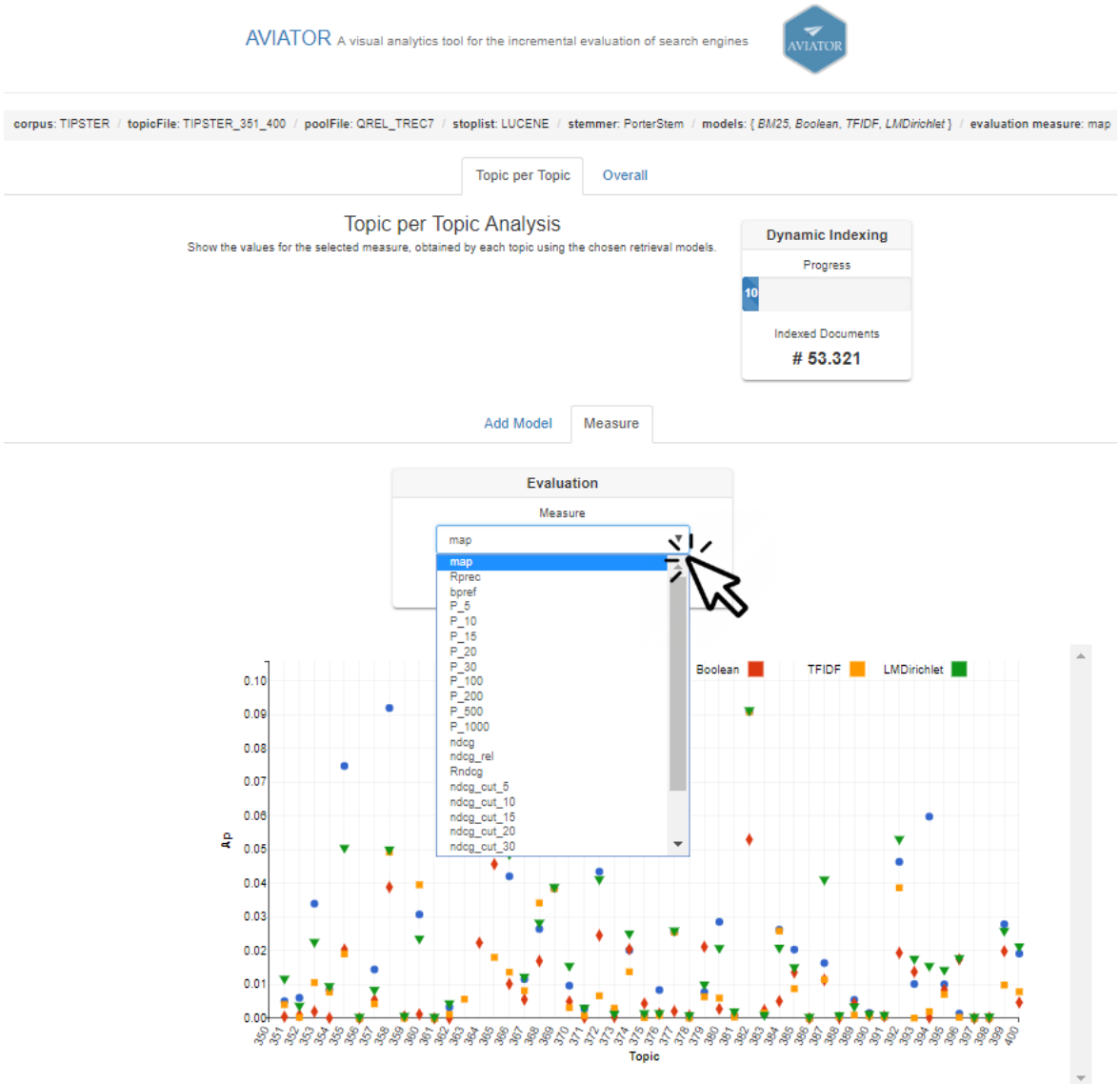

Figure 5.3: AVIATOR visual analytics UI: topic per topic, measure selection, progress 10%.

Figure 5.4 shows the topic per topic analysis with a focus on the retrieval models selection. The interface provides a checkbox for each retrieval model, the user can check the selected models and plot them using the dedicated button. On the right side of the user interface, we can see the progress bar and its value, i.e. 30%, for the incremental indexing process. The progress bar indicates the percentage of documents indexed from the document collection. This value is accompanied by the exact number of indexed documents, e.g. 160 669. On the top of the user interface, there is a light grey box in which are summarised all the information regarding the evaluation data represented in the scatter plot: document corpus, topic file, pool file, stoplist, stemmer, retrieval models and the evaluation measure adopted.

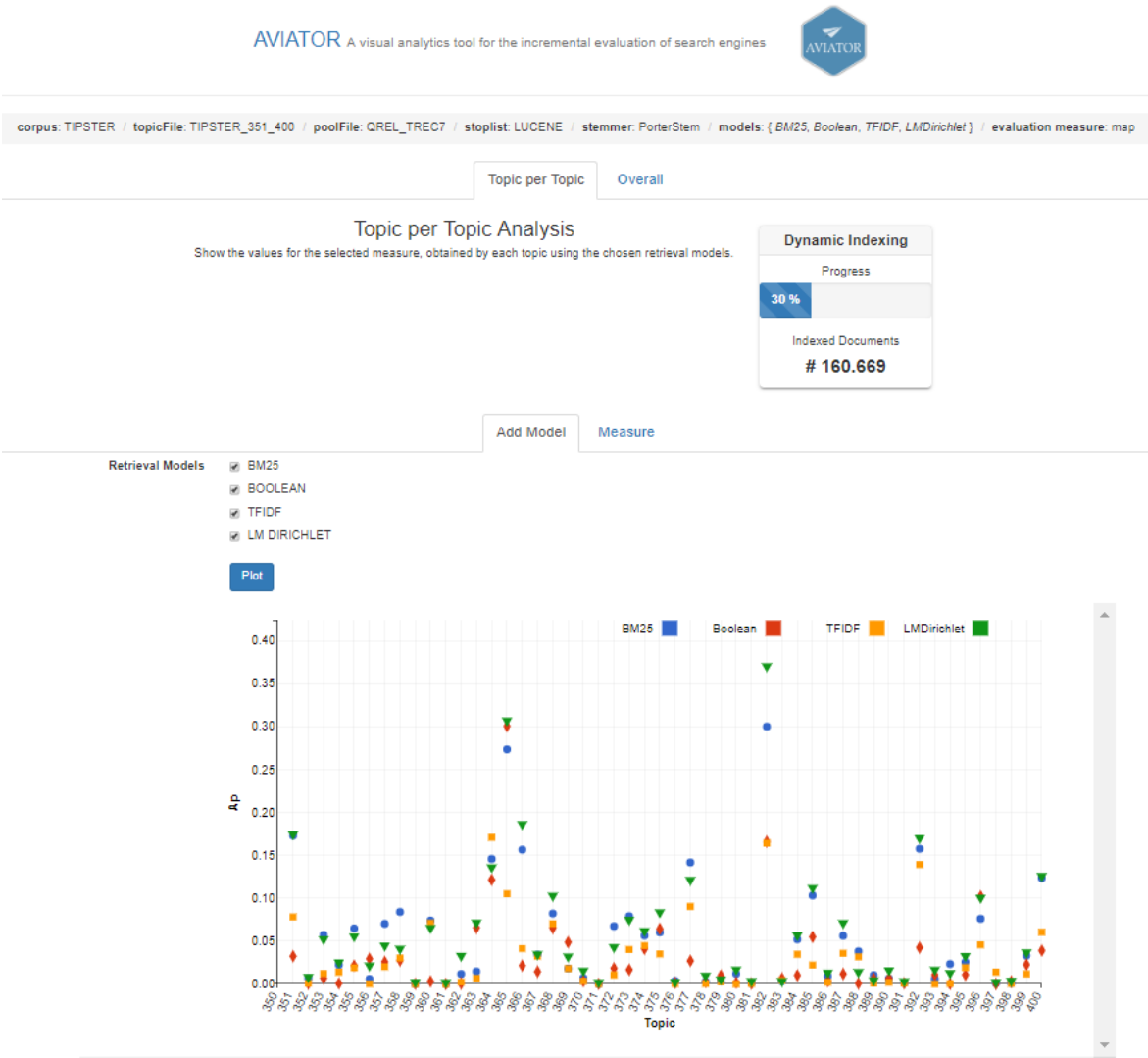

Figure 5.4: AVIATOR visual analytics UI: topic per topic, IR model selection, progress 30%.

Figure 5.5 shows the key functions available in the scatter plot provided for the topic per topic analysis. These functions are:

- **Inspect**: when the mouse pointer goes over a chart point a window appears with the related evaluation data for that point. In particular, we can see the retrieval model, the topic and the value for the considered evaluation measure.
- **Zoom**: this functionality is very useful for exploring the evaluation data, specially when the scatter plot contains many points. Indeed, inspecting a specific point is not an easy task if a lot of other points surround it, but using the zoom functionality, we can easily do that.
- **Pan**: this functionality allows IR experts to move on the entire plot space, thus making easy the point inspection activity.

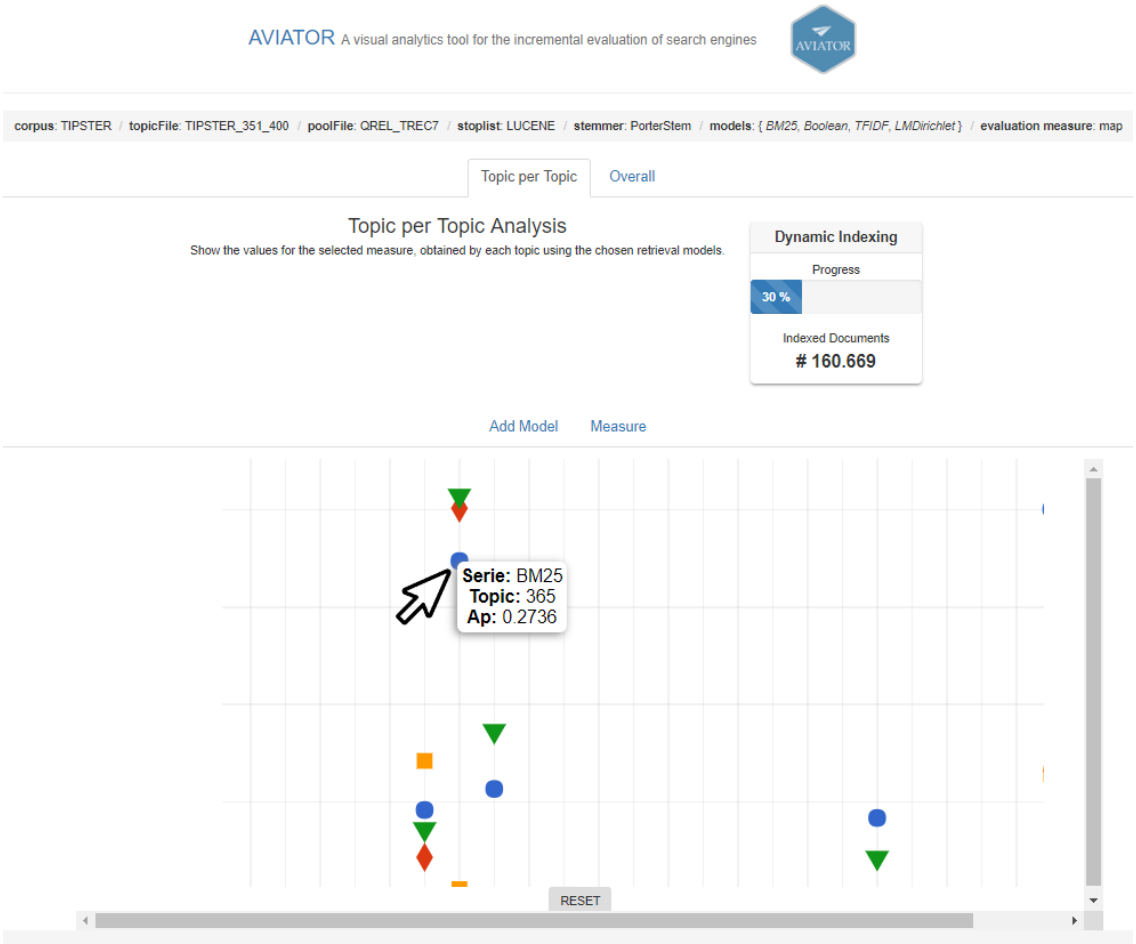

Figure 5.5: AVIATOR visual analytics UI: topic per topic, zoom and inspect, progress 30%.

Figure 5.6 shows the upgrade window that appears when a new index version is available. In particular, in the example of the figure below, we can see that the current progress for the incremental indexing process is 90%, which corresponds to 475 110 indexed documents. The upgrade window, of Figure 5.6, tells the user that the 100% of the document collection has been indexed, so the charts can be updated. The user can choose whether to update the charts or not: by clicking on the "Ok" button, the charts will be updated, otherwise not. However, if the user does not want to update the charts immediately, he can press the "Esc" button and keep working on the current index version. Later, after a timeout, the same message for the upgrade window will appear, reminding the user about the pending index update. In Figure 5.7 we can see the result of the upgrade confirmation.

Figure 5.6: AVIATOR visual analytics UI: topic per topic, progress 90%.

Figure 5.7 shows the AVIATOR visual analytics interface, for the topic per topic analysis, with the whole document collection indexed. Indeed, the progress bar shows that the 100% of the document collection has been indexed, which corresponds to 528 128 indexed documents. Observing the scatter plot, we can see that the best retrieval model, according to the Average precision (AP) measure, is the BM25. Another remarkable retrieval model is the Dirichlet Language Model, whose results are comparable with the BM25 ones. Besides, also the TFIDF model obtains good results, but lower than the two previous models. Hence, the model that gets the worst results is the Boolean model since its similarity criterion does not order the documents retrieved by a relevance score.
Since now, we presented the results only for the topic per topic analysis, but AVIATOR provides also the overall one. To switch to this analysis, the user can simply click on the related tab and the overall chart will appear. This chart is shown in Figure 5.8.

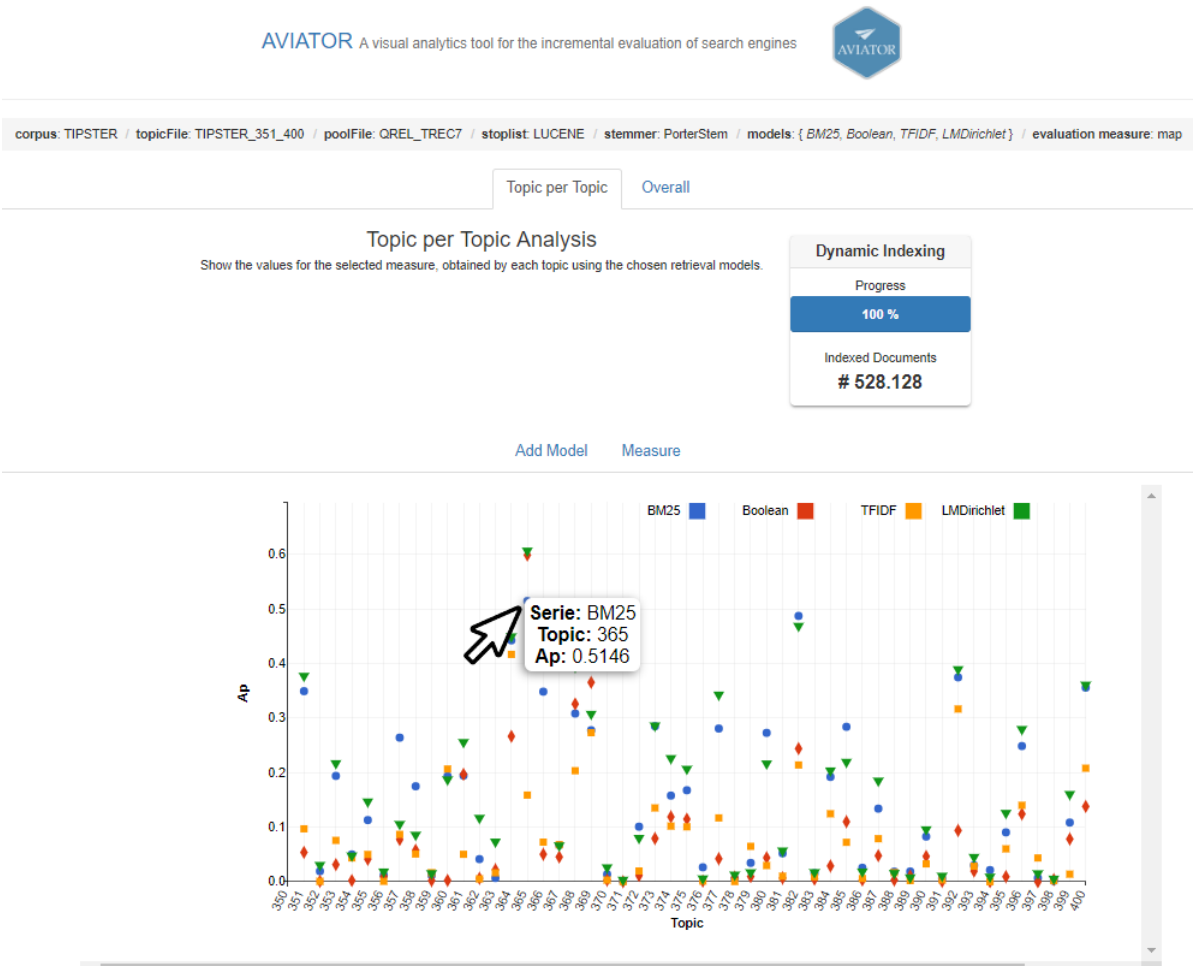

Figure 5.7: AVIATOR visual analytics UI: topic per topic, progress 100%.

Figure 5.8 shows the AVIATOR visual analytics interface, for the overall analysis. This kind of analysis is useful to understand, on average, what are the performances of the retrieval models considered. For this reason, the topic information disappears and the scatter plot is substituted by a bar chart. This one, on the x-axis, has got the list of the retrieval model considered and, on the y-axis, there are the values obtained by each retrieval model, for the measure considered. In the example of Figure 5.8, the evaluation measure is the Mean Average Precision (MAP). As we can see, the overall analysis reflects the same considerations made before. BM25 and Dirichlet Language Model obtain almost the same MAP value, then follows the TFIDF model and the last one is the Boolean.

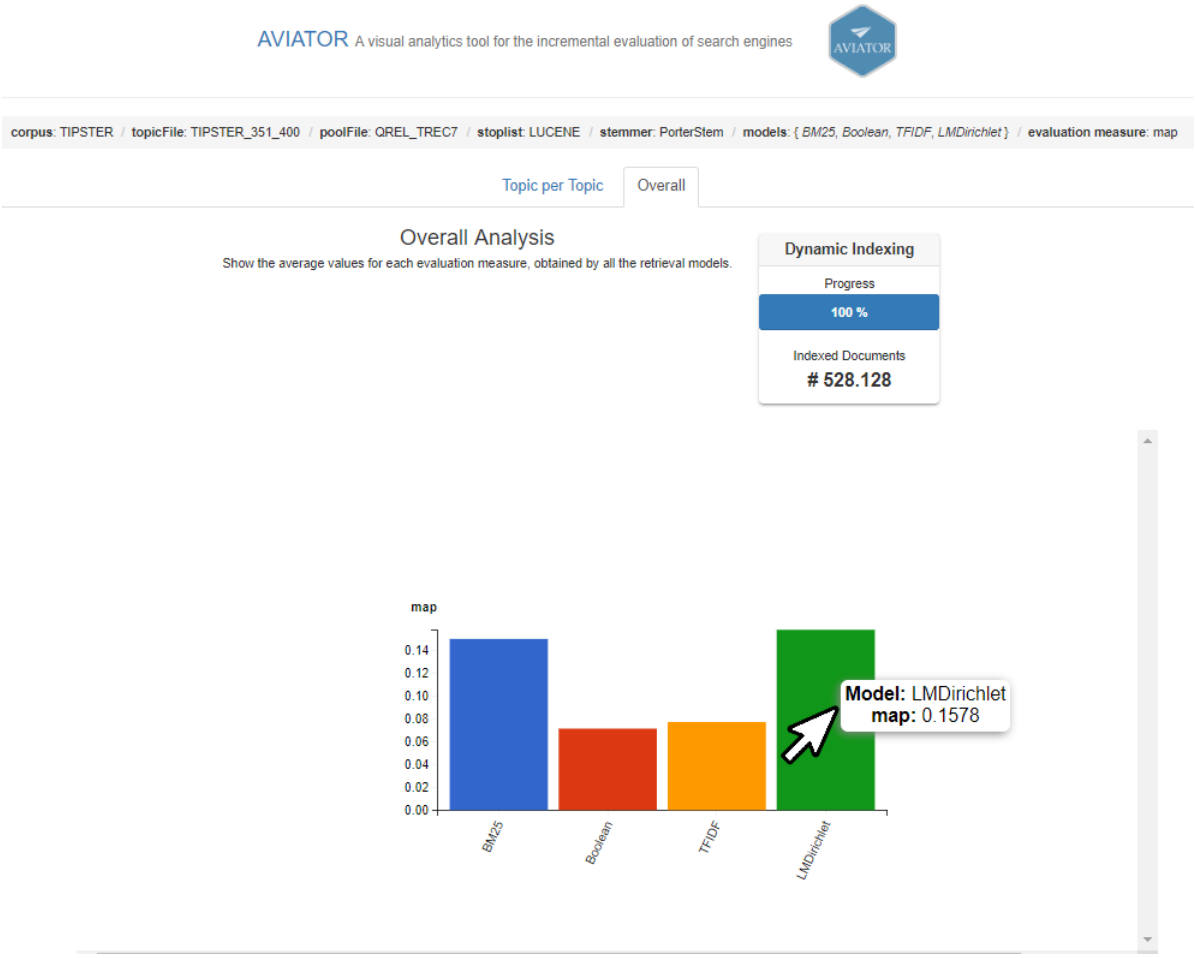

Figure 5.8: AVIATOR visual analytics UI: overall, progress 100%.

5.1 Final remarks

In this chapter, we explain how we implemented the AVIATOR front-end, in terms of technologies used and provided functionalities. In particular, the Model-View-Controller(MVC) architecture is explained, since it is used by AVIATOR to separate the data access layer from the presentation one. Besides, we show how to use the AVIATOR visual analytics web interface for evaluating IR systems. In particular, we show how to perform the exploration of the evaluation data, by means of interactive charts developed using technologies like JavaScript and the D3.js library. Moreover, we show that through the AVIATOR web interface we can use all the back-end services since they are provided as REST APIs.

CHAPTER 6

EXPERIMENTAL EVALUATION

In this chapter, we present the results obtained from the AVIATOR testing phase. We tested AVIATOR on 64 different combinations given by 4 stoplists, 4 stemming algorithms and 4 retrieval models. Each of these 64 different combinations was studied considering 10 index percentages, which gives $64 \times 10 = 640$ different Solr cores. The components used, for our testing purposes, are reported as follows:

- **Document corpus**: the document corpus chosen is the TIPSTER. In particular, for this thesis, we used the disks 4 and 5 of the TREC TIPSTER project.
- **Topic file**: the set of topics chosen comes from TREC7, which contains the topics from 351 to 400, for a total of 50 topics.
- **Pool file**: the pool file adopted, comes from TREC7 too.
- **Stoplist**: the stoplists chosen are:
  - **Indri**: this is the stoplist file which contains the stop words used by the Indri IR system.
  - **Lucene**: this is the default stoplist provided in Solr, inherited by the Apache Lucene project.
  - **No Stoplist**: this is an empty stoplist, useful for evaluating an IR system without performing the stop words removal phase.

  - **Terrier**: this is the stoplist file which contains the stop words used by the Terrier IR system.

- **Stemmer**: the stemming algorithms chosen are:

  - **Krovetz Stemmer**: this is an alternative to the Porter Stemmer available only for the English language. When it is necessary a less aggressive stemmer, this is a good choice.
  - **Porter Stemmer**: this is one of the most famous stemming algorithms. It is very fast but, unfortunately, it is available only for the English language.
  - **Hunspell Stemmer**: this is a dictionary based stemmer available in many languages. For working it requires a dictionary and a set of stemming rules, provided by two separated text files.
  - **No Stemmer**: using this option the user can choose not to perform the stemming phase.

- **Retrieval model**: the retrieval models chosen are:

  - **BM25**
  - **Boolean**
  - **Dirichlet Language Model**
  - **TF-IDF**

The 640 Solr cores, thus generated, correspond to 230 GB of memory on a disk. In this context, the amount of evaluation data produced is very high. For this reason, we decided to report in this chapter only a subset of the evaluation data generated by AVIATOR. In particular, we chose to include in this chapter only the evaluation data related to the combinations given by 2 stoplists (Indri and No Stoplist), 2 stemmers (Porter Stemmer and No Stemmer), 2 retrieval models (BM25 and TF-IDF) and 2 evaluation measures (Average Precision (AP) and normalised Discounted Cumulative Gain (nDCG)), for a total of: $2 \times 2 \times 2 \times 2 = 16$ different combinations. These combinations are described by the following sixteen tables, which show the values for the considered evaluation measures for each topic and bucket. Each of the fifty topics considered corresponds to a row in the table, while the columns are related to the n buckets B_i of the bucket collection $\mathcal{B} = \{B_1, B_2 \ldots, B_n\}$. In particular, the values reported under the B_i column, are obtained from the indexed documents of the buckets from B_1 to B_i. Furthermore, the MAP and the overall nDCG values, reported on top of each table, refer to the whole collection indexed.

BM25 - Average Precision (AP) values w.r.t. MAP = 0.1585

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0051	0.1111	0.1731	0.1676	0.1885	0.2311	0.3152	0.3379	0.3545	0.3776
t_{352}	0.0059	0.0081	0.0067	0.0061	0.0130	0.0191	0.0217	0.0206	0.0231	0.0263
t_{353}	0.0322	0.0458	0.0569	0.0493	0.0466	0.0721	0.0879	0.1403	0.1508	0.1844
t_{354}	0.0083	0.0154	0.0216	0.0230	0.0307	0.0397	0.0459	0.0532	0.0532	0.0533
t_{355}	0.0655	0.0793	0.0565	0.0535	0.0519	0.0568	0.1028	0.0994	0.0933	0.0911
t_{356}	0.0000	0.0000	0.0045	0.0104	0.0098	0.0095	0.0095	0.0077	0.0060	0.0050
t_{357}	0.0142	0.0646	0.0721	0.1057	0.1264	0.1934	0.2471	0.2782	0.2807	0.2784
t_{358}	0.0860	0.0782	0.0813	0.0970	0.1027	0.1010	0.0970	0.0914	0.1208	0.1820
t_{359}	0.0003	0.0002	0.0001	0.0107	0.0097	0.0080	0.0070	0.0085	0.0090	0.0105
t_{360}	0.0318	0.0449	0.0716	0.0811	0.0836	0.0888	0.0895	0.1538	0.1741	0.1886
t_{361}	0.0000	0.0000	0.0000	0.0000	0.0556	0.0556	0.0556	0.2130	0.2130	0.2794
t_{362}	0.0032	0.0103	0.0103	0.0071	0.0276	0.0360	0.0571	0.0551	0.0657	0.0672
t_{363}	0.0636	0.0188	0.0146	0.0125	0.0130	0.0142	0.0111	0.0104	0.0120	0.0074
t_{364}	0.0571	0.1110	0.1444	0.1314	0.1767	0.2837	0.2952	0.3690	0.4625	0.5008
t_{365}	0.0786	0.1971	0.2707	0.3536	0.4769	0.4722	0.4641	0.5148	0.6428	0.6541
t_{366}	0.0226	0.0501	0.0862	0.1584	0.2047	0.2223	0.2139	0.1951	0.2385	0.2693
t_{367}	0.0115	0.0398	0.0343	0.0319	0.0317	0.0516	0.0566	0.0654	0.0628	0.0636
t_{368}	0.0255	0.0413	0.0794	0.1196	0.1516	0.1521	0.1469	0.1453	0.2412	0.3477
t_{369}	0.0385	0.0085	0.0182	0.0822	0.0641	0.0502	0.0471	0.0278	0.1148	0.2679
t_{370}	0.0094	0.0092	0.0070	0.0102	0.0130	0.0129	0.0114	0.0129	0.0142	0.0139
t_{371}	0.0028	0.0013	0.0009	0.0014	0.0011	0.0008	0.0011	0.0015	0.0013	0.0008
t_{372}	0.0446	0.0485	0.0682	0.0625	0.0809	0.0805	0.0860	0.0849	0.1288	0.1150
t_{373}	0.0020	0.0502	0.0778	0.2350	0.2160	0.2241	0.2694	0.2626	0.2234	0.2204
t_{374}	0.0203	0.0312	0.0561	0.0637	0.0945	0.1062	0.1266	0.1343	0.1481	0.1847
t_{375}	0.0009	0.0316	0.0590	0.0710	0.0897	0.1065	0.1692	0.1793	0.1971	0.1889
t_{376}	0.0073	0.0042	0.0034	0.0047	0.0060	0.0096	0.0157	0.0176	0.0223	0.0253
t_{377}	0.0256	0.1670	0.1427	0.1393	0.1518	0.1835	0.1935	0.2396	0.2794	0.2882
t_{378}	0.0003	0.0010	0.0026	0.0042	0.0040	0.0043	0.0043	0.0046	0.0050	0.0053
t_{379}	0.0083	0.0045	0.0032	0.0054	0.0055	0.0046	0.0042	0.0035	0.0029	0.0352
t_{380}	0.0286	0.0143	0.0119	0.0102	0.0102	0.0102	0.0090	0.0085	0.1053	0.1855
t_{381}	0.0008	0.0008	0.0012	0.0019	0.0015	0.0033	0.0040	0.0078	0.0074	0.0507
t_{382}	0.0909	0.2007	0.3019	0.2780	0.3534	0.4057	0.4412	0.5023	0.5026	0.5580
t_{383}	0.0011	0.0022	0.0030	0.0122	0.0119	0.0129	0.0136	0.0151	0.0161	0.0154
t_{384}	0.0263	0.0292	0.0518	0.0608	0.0775	0.1394	0.1525	0.1487	0.1782	0.2067
t_{385}	0.0192	0.0623	0.1041	0.1359	0.1508	0.1953	0.2118	0.2894	0.3030	0.2989
t_{386}	0.0000	0.0144	0.0090	0.0085	0.0135	0.0113	0.0116	0.0387	0.0379	0.0276
t_{387}	0.0149	0.0123	0.0541	0.0599	0.0607	0.0785	0.1030	0.1321	0.1364	0.1326
t_{388}	0.0006	0.0050	0.0380	0.0316	0.0327	0.0291	0.0247	0.0224	0.0287	0.0264
t_{389}	0.0056	0.0062	0.0105	0.0124	0.0141	0.0135	0.0138	0.0131	0.0180	0.0205
t_{390}	0.0016	0.0120	0.0064	0.0300	0.0297	0.0292	0.0326	0.0359	0.0610	0.0838
t_{391}	0.0005	0.0006	0.0013	0.0038	0.0062	0.0060	0.0075	0.0072	0.0085	0.0080
t_{392}	0.0474	0.1454	0.1595	0.1917	0.2455	0.3223	0.3200	0.3653	0.4085	0.4373
t_{393}	0.0102	0.0062	0.0058	0.0044	0.0043	0.0029	0.0105	0.0097	0.0246	0.0323
t_{394}	0.0598	0.0299	0.0185	0.0149	0.0184	0.0212	0.0205	0.0193	0.0243	0.0219
t_{395}	0.0105	0.0195	0.0269	0.0380	0.0539	0.0714	0.0803	0.0926	0.0993	0.1040
t_{396}	0.0010	0.0407	0.0741	0.1413	0.1763	0.2222	0.2377	0.2396	0.2379	0.2718
t_{397}	0.0000	0.0006	0.0009	0.0006	0.0071	0.0059	0.0055	0.0132	0.0128	0.0106
t_{398}	0.0001	0.0006	0.0010	0.0008	0.0010	0.0016	0.0020	0.0017	0.0022	0.0020
t_{399}	0.0278	0.0376	0.0326	0.0417	0.0439	0.0668	0.0638	0.0734	0.0807	0.1095
t_{400}	0.0185	0.0575	0.1224	0.1753	0.2127	0.2542	0.2809	0.2974	0.3335	0.3896

Table 6.1: Measure: AP; Stoplist: INDRI; Stemmer: PORTERSTEM; Model: BM25.

TF-IDF - Average Precision (AP) values w.r.t. MAP = 0.0868

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0076	0.0351	0.0828	0.0740	0.0723	0.0834	0.0972	0.1005	0.0987	0.1061
t_{352}	0.0003	0.0016	0.0011	0.0009	0.0010	0.0011	0.0010	0.0009	0.0010	0.0012
t_{353}	0.0116	0.0118	0.0140	0.0227	0.0171	0.0237	0.0361	0.0513	0.0508	0.0712
t_{354}	0.0078	0.0109	0.0144	0.0151	0.0200	0.0295	0.0352	0.0457	0.0452	0.0457
t_{355}	0.0142	0.0186	0.0157	0.0112	0.0086	0.0129	0.0336	0.0347	0.0318	0.0299
t_{356}	0.0000	0.0000	0.0002	0.0006	0.0005	0.0005	0.0005	0.0004	0.0002	0.0002
t_{357}	0.0047	0.0216	0.0217	0.0371	0.0409	0.0620	0.0805	0.0966	0.0931	0.0917
t_{358}	0.0545	0.0436	0.0442	0.0357	0.0315	0.0262	0.0246	0.0118	0.0352	0.0429
t_{359}	0.0002	0.0004	0.0003	0.0371	0.0341	0.0253	0.0299	0.0315	0.0173	0.0161
t_{360}	0.0442	0.0534	0.0807	0.0919	0.0989	0.1233	0.1335	0.1823	0.2070	0.2249
t_{361}	0.0000	0.0000	0.0000	0.0000	0.0370	0.0370	0.0370	0.1093	0.0796	0.1009
t_{362}	0.0012	0.0017	0.0021	0.0010	0.0036	0.0045	0.0051	0.0049	0.0069	0.0092
t_{363}	0.0054	0.0105	0.0073	0.0104	0.0331	0.0338	0.0317	0.0237	0.0396	0.0378
t_{364}	0.0571	0.1119	0.1742	0.1303	0.1660	0.2780	0.2927	0.3670	0.4326	0.4657
t_{365}	0.0345	0.0856	0.1212	0.1283	0.2199	0.2177	0.2124	0.2123	0.2769	0.2706
t_{366}	0.0093	0.0188	0.0268	0.0321	0.0335	0.0354	0.0314	0.0272	0.0361	0.0507
t_{367}	0.0118	0.0365	0.0343	0.0322	0.0325	0.0520	0.0623	0.0697	0.0664	0.0672
t_{368}	0.0339	0.0466	0.0759	0.0977	0.1056	0.1024	0.0926	0.0872	0.1831	0.2280
t_{369}	0.0385	0.0085	0.0182	0.0822	0.0592	0.0471	0.0445	0.0270	0.1142	0.2632
t_{370}	0.0031	0.0038	0.0030	0.0042	0.0060	0.0051	0.0036	0.0024	0.0039	0.0037
t_{371}	0.0007	0.0003	0.0002	0.0003	0.0003	0.0002	0.0003	0.0002	0.0002	0.0002
t_{372}	0.0120	0.0135	0.0130	0.0141	0.0180	0.0169	0.0198	0.0166	0.0239	0.0235
t_{373}	0.0034	0.0078	0.0188	0.1025	0.1019	0.1048	0.1383	0.1357	0.1339	0.1297
t_{374}	0.0141	0.0254	0.0447	0.0494	0.0692	0.0831	0.0963	0.0963	0.1049	0.1206
t_{375}	0.0002	0.0204	0.0362	0.0417	0.0524	0.0596	0.1026	0.1201	0.1243	0.1187
t_{376}	0.0009	0.0005	0.0003	0.0003	0.0004	0.0010	0.0013	0.0018	0.0016	0.0012
t_{377}	0.0256	0.0947	0.0903	0.0879	0.0867	0.1103	0.1095	0.0984	0.1013	0.1101
t_{378}	0.0001	0.0002	0.0004	0.0007	0.0005	0.0007	0.0005	0.0005	0.0006	0.0007
t_{379}	0.0062	0.0034	0.0024	0.0040	0.0048	0.0039	0.0035	0.0031	0.0025	0.0658
t_{380}	0.0048	0.0022	0.0017	0.0013	0.0012	0.0013	0.0009	0.0007	0.0227	0.0291
t_{381}	0.0004	0.0004	0.0004	0.0009	0.0006	0.0010	0.0025	0.0061	0.0054	0.0102
t_{382}	0.0909	0.1522	0.1807	0.1664	0.2321	0.2661	0.2359	0.2647	0.2632	0.2867
t_{383}	0.0015	0.0016	0.0040	0.0092	0.0099	0.0105	0.0092	0.0095	0.0092	0.0098
t_{384}	0.0264	0.0271	0.0361	0.0429	0.0775	0.1234	0.1268	0.1304	0.1424	0.1554
t_{385}	0.0091	0.0144	0.0246	0.0250	0.0341	0.0502	0.0536	0.0808	0.0914	0.0874
t_{386}	0.0000	0.0030	0.0020	0.0016	0.0034	0.0030	0.0024	0.0096	0.0098	0.0087
t_{387}	0.0123	0.0085	0.0364	0.0381	0.0376	0.0484	0.0652	0.0783	0.0797	0.0779
t_{388}	0.0005	0.0009	0.0381	0.0367	0.0427	0.0379	0.0357	0.0355	0.0383	0.0258
t_{389}	0.0010	0.0005	0.0010	0.0010	0.0016	0.0016	0.0018	0.0015	0.0017	0.0026
t_{390}	0.0011	0.0020	0.0021	0.0131	0.0098	0.0084	0.0059	0.0075	0.0158	0.0304
t_{391}	0.0005	0.0005	0.0011	0.0032	0.0044	0.0039	0.0046	0.0042	0.0046	0.0045
t_{392}	0.0389	0.1289	0.1363	0.1634	0.2114	0.2980	0.2970	0.3348	0.3694	0.3659
t_{393}	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0002	0.0002	0.0147	0.0290
t_{394}	0.0021	0.0009	0.0005	0.0004	0.0010	0.0009	0.0009	0.0006	0.0005	0.0005
t_{395}	0.0076	0.0191	0.0227	0.0294	0.0402	0.0536	0.0582	0.0669	0.0709	0.0727
t_{396}	0.0002	0.0203	0.0395	0.0841	0.0927	0.1227	0.1235	0.1176	0.1180	0.1457
t_{397}	0.0000	0.0453	0.0244	0.0202	0.0427	0.0416	0.0350	0.0270	0.0416	0.0511
t_{398}	0.0001	0.0002	0.0005	0.0004	0.0004	0.0010	0.0021	0.0018	0.0019	0.0017
t_{399}	0.0098	0.0181	0.0122	0.0112	0.0221	0.0228	0.0221	0.0245	0.0249	0.0243
t_{400}	0.0082	0.0254	0.0620	0.0877	0.1025	0.1295	0.1355	0.1597	0.1911	0.2252

Table 6.2: Measure: AP; Stoplist: INDRI; Stemmer: PORTERSTEM; Model: TF-IDF.

BM25 - Average Precision (AP) values w.r.t. MAP = 0.1553

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0013	0.0607	0.1260	0.1242	0.1471	0.1881	0.2430	0.2635	0.2722	0.2959
t_{352}	0.0061	0.0082	0.0067	0.0063	0.0130	0.0192	0.0218	0.0207	0.0231	0.0263
t_{353}	0.0497	0.0664	0.0706	0.0543	0.0505	0.0784	0.1019	0.1533	0.1610	0.2023
t_{354}	0.0105	0.0085	0.0115	0.0145	0.0215	0.0321	0.0430	0.0578	0.0615	0.0635
t_{355}	0.0625	0.0772	0.0518	0.0408	0.0357	0.0397	0.0722	0.0737	0.0711	0.0703
t_{356}	0.0000	0.0000	0.0053	0.0128	0.0119	0.0115	0.0112	0.0091	0.0069	0.0056
t_{357}	0.0162	0.0603	0.0693	0.1090	0.1363	0.1991	0.2634	0.2996	0.3002	0.2953
t_{358}	0.0529	0.0435	0.0428	0.0583	0.0648	0.0622	0.0601	0.0536	0.0621	0.1040
t_{359}	0.0006	0.0005	0.0002	0.0403	0.0419	0.0229	0.0226	0.0197	0.0169	0.0168
t_{360}	0.0503	0.0877	0.1249	0.1424	0.1632	0.1753	0.2095	0.3249	0.3707	0.4106
t_{361}	0.0000	0.0000	0.0000	0.0000	0.0556	0.0556	0.0556	0.1963	0.1852	0.2513
t_{362}	0.0035	0.0108	0.0096	0.0086	0.0259	0.0435	0.0691	0.0658	0.0848	0.0903
t_{363}	0.0329	0.0550	0.0529	0.0378	0.0410	0.0376	0.0302	0.0286	0.0406	0.0311
t_{364}	0.0571	0.1244	0.1552	0.1377	0.1825	0.2956	0.3350	0.4178	0.5098	0.5512
t_{365}	0.0786	0.1971	0.2758	0.3623	0.4850	0.4819	0.4724	0.5316	0.6557	0.6728
t_{366}	0.0224	0.0501	0.0835	0.1480	0.1851	0.2022	0.1943	0.1712	0.2155	0.2463
t_{367}	0.0115	0.0415	0.0351	0.0324	0.0320	0.0520	0.0570	0.0658	0.0631	0.0639
t_{368}	0.0289	0.0432	0.0896	0.1289	0.1596	0.1645	0.1584	0.1577	0.2386	0.3672
t_{369}	0.0385	0.0085	0.0182	0.0822	0.0641	0.0502	0.0471	0.0278	0.1148	0.2679
t_{370}	0.0077	0.0095	0.0064	0.0094	0.0111	0.0104	0.0106	0.0120	0.0121	0.0111
t_{371}	0.0033	0.0016	0.0011	0.0016	0.0012	0.0009	0.0013	0.0017	0.0015	0.0009
t_{372}	0.0258	0.0292	0.0483	0.0468	0.0617	0.0611	0.0630	0.0622	0.0922	0.0894
t_{373}	0.0051	0.0606	0.0878	0.2861	0.2475	0.2578	0.2949	0.2857	0.2879	0.2897
t_{374}	0.0229	0.0420	0.0763	0.0843	0.1318	0.1475	0.1676	0.1817	0.1925	0.2340
t_{375}	0.0009	0.0317	0.0592	0.0711	0.0902	0.1070	0.1697	0.1803	0.1987	0.1903
t_{376}	0.0089	0.0051	0.0043	0.0058	0.0074	0.0118	0.0193	0.0223	0.0277	0.0306
t_{377}	0.0256	0.0732	0.0652	0.0616	0.0605	0.0776	0.0730	0.1184	0.1418	0.1350
t_{378}	0.0003	0.0010	0.0026	0.0040	0.0040	0.0045	0.0046	0.0045	0.0049	0.0056
t_{379}	0.0625	0.0625	0.0625	0.1250	0.1250	0.1250	0.1250	0.1250	0.1562	0.2677
t_{380}	0.0476	0.0238	0.0204	0.0159	0.0159	0.0169	0.0138	0.0137	0.0795	0.1472
t_{381}	0.0012	0.0006	0.0009	0.0010	0.0009	0.0013	0.0026	0.0084	0.0078	0.0523
t_{382}	0.0909	0.1727	0.2535	0.2269	0.3186	0.3747	0.3779	0.3739	0.3719	0.4248
t_{383}	0.0007	0.0018	0.0026	0.0074	0.0063	0.0069	0.0079	0.0110	0.0131	0.0128
t_{384}	0.0264	0.0299	0.0518	0.0613	0.0796	0.1410	0.1557	0.1548	0.1949	0.2240
t_{385}	0.0217	0.0568	0.0920	0.1227	0.1439	0.1775	0.1887	0.2477	0.2524	0.2506
t_{386}	0.0000	0.0115	0.0095	0.0093	0.0097	0.0081	0.0083	0.0236	0.0214	0.0202
t_{387}	0.0202	0.0167	0.0608	0.0628	0.0633	0.0802	0.1013	0.1301	0.1349	0.1321
t_{388}	0.0027	0.0100	0.0426	0.0394	0.0666	0.0642	0.0640	0.0575	0.0853	0.0776
t_{389}	0.0003	0.0007	0.0060	0.0046	0.0049	0.0034	0.0032	0.0029	0.0028	0.0044
t_{390}	0.0003	0.0083	0.0099	0.0316	0.0318	0.0321	0.0392	0.0468	0.0617	0.0824
t_{391}	0.0005	0.0006	0.0013	0.0038	0.0062	0.0060	0.0075	0.0072	0.0085	0.0080
t_{392}	0.0161	0.0499	0.0598	0.0606	0.0926	0.1146	0.1116	0.1244	0.1346	0.1299
t_{393}	0.0223	0.0124	0.0116	0.0103	0.0102	0.0093	0.0135	0.0159	0.0351	0.0512
t_{394}	0.0826	0.0425	0.0381	0.0323	0.0523	0.0490	0.0425	0.0406	0.0360	0.0264
t_{395}	0.0105	0.0194	0.0268	0.0381	0.0538	0.0713	0.0801	0.0925	0.0992	0.1039
t_{396}	0.0016	0.0448	0.0769	0.1469	0.1807	0.2299	0.2449	0.2475	0.2457	0.2733
t_{397}	0.0000	0.0000	0.0000	0.0000	0.0009	0.0007	0.0005	0.0013	0.0020	0.0093
t_{398}	0.0001	0.0002	0.0001	0.0001	0.0002	0.0007	0.0005	0.0005	0.0008	0.0009
t_{399}	0.0296	0.0276	0.0358	0.0325	0.0346	0.0540	0.0497	0.0556	0.0564	0.0829
t_{400}	0.0177	0.0561	0.1128	0.1579	0.1965	0.2342	0.2586	0.2828	0.3133	0.3642

Table 6.3: Measure: AP; Stoplist: INDRI; Stemmer: NOSTEM; Model: BM25.

TF-IDF - Average Precision (AP) values w.r.t. MAP = 0.0863

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0072	0.0214	0.0506	0.0485	0.0465	0.0532	0.0599	0.0641	0.0641	0.0671
t_{352}	0.0003	0.0016	0.0011	0.0009	0.0010	0.0011	0.0010	0.0009	0.0010	0.0012
t_{353}	0.0068	0.0073	0.0078	0.0236	0.0187	0.0236	0.0331	0.0527	0.0516	0.0717
t_{354}	0.0062	0.0054	0.0077	0.0083	0.0123	0.0189	0.0295	0.0451	0.0470	0.0479
t_{355}	0.0170	0.0288	0.0213	0.0165	0.0127	0.0160	0.0360	0.0327	0.0305	0.0291
t_{356}	0.0000	0.0000	0.0002	0.0007	0.0006	0.0005	0.0005	0.0004	0.0002	0.0002
t_{357}	0.0089	0.0326	0.0340	0.0568	0.0734	0.1142	0.1578	0.1834	0.1811	0.1826
t_{358}	0.0316	0.0212	0.0225	0.0256	0.0274	0.0238	0.0229	0.0150	0.0208	0.0425
t_{359}	0.0002	0.0005	0.0001	0.0096	0.0081	0.0050	0.0054	0.0060	0.0049	0.0045
t_{360}	0.0417	0.0587	0.0997	0.1115	0.1227	0.1334	0.1539	0.2115	0.2393	0.2651
t_{361}	0.0000	0.0000	0.0000	0.0000	0.0370	0.0370	0.0370	0.0898	0.0675	0.0814
t_{362}	0.0011	0.0013	0.0010	0.0008	0.0032	0.0050	0.0061	0.0063	0.0087	0.0102
t_{363}	0.0114	0.0225	0.0155	0.0215	0.0878	0.0873	0.0846	0.0846	0.1326	0.1275
t_{364}	0.0571	0.1264	0.1880	0.1378	0.1735	0.2917	0.3332	0.4183	0.4816	0.5193
t_{365}	0.0348	0.0858	0.1264	0.1334	0.2153	0.2112	0.2055	0.2038	0.2719	0.2694
t_{366}	0.0096	0.0174	0.0256	0.0294	0.0314	0.0343	0.0306	0.0263	0.0352	0.0568
t_{367}	0.0118	0.0372	0.0347	0.0325	0.0327	0.0523	0.0627	0.0700	0.0668	0.0676
t_{368}	0.0322	0.0467	0.0839	0.1149	0.1280	0.1326	0.1250	0.1239	0.2073	0.3007
t_{369}	0.0385	0.0085	0.0182	0.0822	0.0592	0.0471	0.0445	0.0270	0.1142	0.2632
t_{370}	0.0031	0.0039	0.0030	0.0037	0.0055	0.0047	0.0039	0.0030	0.0038	0.0041
t_{371}	0.0009	0.0004	0.0003	0.0005	0.0004	0.0003	0.0006	0.0005	0.0004	0.0005
t_{372}	0.0211	0.0122	0.0151	0.0119	0.0141	0.0131	0.0137	0.0126	0.0172	0.0181
t_{373}	0.0061	0.0091	0.0099	0.0791	0.0816	0.0782	0.1085	0.1063	0.1067	0.1050
t_{374}	0.0169	0.0340	0.0589	0.0621	0.0845	0.1029	0.1162	0.1196	0.1257	0.1477
t_{375}	0.0002	0.0173	0.0334	0.0388	0.0487	0.0559	0.1001	0.1194	0.1215	0.1171
t_{376}	0.0012	0.0006	0.0003	0.0004	0.0006	0.0012	0.0016	0.0023	0.0022	0.0019
t_{377}	0.0256	0.0596	0.0565	0.0467	0.0457	0.0477	0.0441	0.0526	0.0580	0.0564
t_{378}	0.0001	0.0002	0.0004	0.0006	0.0004	0.0006	0.0005	0.0005	0.0005	0.0006
t_{379}	0.0625	0.0625	0.0625	0.1250	0.1250	0.1250	0.1250	0.1250	0.1562	0.2833
t_{380}	0.0060	0.0026	0.0022	0.0017	0.0015	0.0016	0.0014	0.0009	0.0046	0.0067
t_{381}	0.0006	0.0002	0.0003	0.0004	0.0001	0.0002	0.0026	0.0137	0.0103	0.0161
t_{382}	0.0909	0.1407	0.1518	0.1494	0.2223	0.2540	0.2141	0.2071	0.2037	0.2239
t_{383}	0.0008	0.0014	0.0033	0.0057	0.0048	0.0059	0.0061	0.0076	0.0076	0.0071
t_{384}	0.0248	0.0268	0.0328	0.0396	0.0751	0.1181	0.1224	0.1263	0.1497	0.1595
t_{385}	0.0060	0.0108	0.0170	0.0175	0.0243	0.0292	0.0302	0.0403	0.0390	0.0383
t_{386}	0.0000	0.0018	0.0012	0.0009	0.0012	0.0010	0.0005	0.0032	0.0038	0.0029
t_{387}	0.0130	0.0087	0.0379	0.0382	0.0383	0.0480	0.0611	0.0761	0.0776	0.0752
t_{388}	0.0014	0.0031	0.0423	0.0388	0.0607	0.0606	0.0608	0.0607	0.0764	0.0480
t_{389}	0.0001	0.0002	0.0002	0.0003	0.0004	0.0005	0.0005	0.0004	0.0003	0.0007
t_{390}	0.0003	0.0010	0.0014	0.0041	0.0044	0.0044	0.0044	0.0043	0.0054	0.0192
t_{391}	0.0005	0.0005	0.0011	0.0032	0.0044	0.0039	0.0046	0.0042	0.0046	0.0045
t_{392}	0.0148	0.0496	0.0510	0.0514	0.0865	0.1075	0.1044	0.1152	0.1252	0.1106
t_{393}	0.0011	0.0006	0.0004	0.0002	0.0001	0.0001	0.0003	0.0003	0.0146	0.0289
t_{394}	0.0166	0.0064	0.0050	0.0045	0.0049	0.0042	0.0028	0.0030	0.0019	0.0017
t_{395}	0.0076	0.0190	0.0227	0.0293	0.0400	0.0536	0.0578	0.0668	0.0707	0.0728
t_{396}	0.0003	0.0208	0.0407	0.0814	0.0845	0.1129	0.1136	0.1097	0.1083	0.1305
t_{397}	0.0000	0.0000	0.0000	0.0000	0.0001	0.0001	0.0001	0.0001	0.0010	0.0021
t_{398}	0.0001	0.0001	0.0001	0.0000	0.0001	0.0006	0.0004	0.0003	0.0004	0.0004
t_{399}	0.0031	0.0035	0.0036	0.0039	0.0090	0.0101	0.0094	0.0095	0.0094	0.0116
t_{400}	0.0091	0.0252	0.0615	0.0864	0.0982	0.1270	0.1327	0.1559	0.1803	0.2103

Table 6.4: Measure: AP; Stoplist: INDRI; Stemmer: NOSTEM; Model: TF-IDF

BM25 - Average Precision (AP) values w.r.t. MAP = 0.1631

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0033	0.0995	0.1668	0.1614	0.1819	0.2229	0.3047	0.3289	0.3419	0.3643
t_{352}	0.0062	0.0083	0.0065	0.0060	0.0125	0.0193	0.0217	0.0204	0.0235	0.0266
t_{353}	0.0340	0.0475	0.0564	0.0476	0.0452	0.0708	0.0867	0.1383	0.1495	0.1838
t_{354}	0.0090	0.0166	0.0237	0.0257	0.0339	0.0443	0.0520	0.0602	0.0601	0.0609
t_{355}	0.0751	0.0965	0.0688	0.0666	0.0633	0.0612	0.1109	0.1055	0.1012	0.0985
t_{356}	0.0000	0.0000	0.0059	0.0124	0.0116	0.0116	0.0111	0.0093	0.0072	0.0059
t_{357}	0.0148	0.0614	0.0687	0.1019	0.1226	0.1899	0.2461	0.2827	0.2851	0.2840
t_{358}	0.0882	0.0778	0.0836	0.0975	0.1040	0.1020	0.0989	0.0946	0.1241	0.1814
t_{359}	0.0003	0.0001	0.0001	0.0107	0.0102	0.0084	0.0079	0.0093	0.0106	0.0121
t_{360}	0.0326	0.0438	0.0749	0.0843	0.0859	0.0959	0.0965	0.1635	0.1829	0.1979
t_{361}	0.0000	0.0000	0.0000	0.0000	0.0556	0.0556	0.0556	0.2130	0.2130	0.2690
t_{362}	0.0032	0.0124	0.0118	0.0093	0.0258	0.0424	0.0573	0.0531	0.0638	0.0651
t_{363}	0.0636	0.0192	0.0150	0.0128	0.0128	0.0133	0.0104	0.0100	0.0121	0.0073
t_{364}	0.0571	0.1150	0.1369	0.1204	0.1603	0.2690	0.2856	0.3624	0.4465	0.4841
t_{365}	0.0786	0.2019	0.2733	0.3558	0.4828	0.4792	0.4574	0.5162	0.6442	0.6571
t_{366}	0.0430	0.0860	0.1545	0.2546	0.3432	0.3711	0.3853	0.3569	0.3893	0.4310
t_{367}	0.0121	0.0393	0.0342	0.0314	0.0313	0.0522	0.0574	0.0663	0.0634	0.0643
t_{368}	0.0267	0.0412	0.0827	0.1202	0.1510	0.1497	0.1442	0.1423	0.2373	0.3382
t_{369}	0.0385	0.0070	0.0172	0.0913	0.0682	0.0527	0.0491	0.0286	0.1164	0.2685
t_{370}	0.0099	0.0113	0.0073	0.0120	0.0162	0.0150	0.0144	0.0162	0.0191	0.0185
t_{371}	0.0028	0.0000	0.0000	0.0004	0.0002	0.0002	0.0004	0.0007	0.0007	0.0003
t_{372}	0.0440	0.0476	0.0677	0.0742	0.0943	0.0931	0.0973	0.0965	0.1324	0.1226
t_{373}	0.0022	0.0553	0.0959	0.2792	0.2599	0.2722	0.3099	0.3051	0.3073	0.2959
t_{374}	0.0203	0.0315	0.0570	0.0633	0.0929	0.1027	0.1210	0.1301	0.1453	0.1795
t_{375}	0.0008	0.0313	0.0606	0.0721	0.0903	0.1046	0.1592	0.1704	0.1847	0.1784
t_{376}	0.0076	0.0042	0.0031	0.0048	0.0062	0.0100	0.0148	0.0167	0.0214	0.0242
t_{377}	0.0256	0.1631	0.1417	0.1402	0.1480	0.1807	0.1864	0.2260	0.2704	0.2826
t_{378}	0.0003	0.0009	0.0023	0.0036	0.0035	0.0036	0.0038	0.0036	0.0039	0.0044
t_{379}	0.0089	0.0048	0.0033	0.0054	0.0058	0.0049	0.0044	0.0037	0.0031	0.0667
t_{380}	0.0286	0.0143	0.0119	0.0102	0.0102	0.0101	0.0090	0.0084	0.1039	0.2053
t_{381}	0.0009	0.0007	0.0012	0.0020	0.0015	0.0037	0.0043	0.0075	0.0070	0.0513
t_{382}	0.0909	0.1890	0.2933	0.2715	0.3486	0.3985	0.4357	0.4968	0.5022	0.5537
t_{383}	0.0012	0.0023	0.0034	0.0119	0.0117	0.0127	0.0130	0.0138	0.0144	0.0135
t_{384}	0.0263	0.0290	0.0515	0.0610	0.0768	0.1366	0.1494	0.1476	0.1773	0.2036
t_{385}	0.0195	0.0629	0.1036	0.1344	0.1469	0.1898	0.2014	0.2814	0.2960	0.2917
t_{386}	0.0000	0.0152	0.0088	0.0094	0.0160	0.0137	0.0141	0.0667	0.0651	0.0380
t_{387}	0.0188	0.0159	0.0587	0.0636	0.0632	0.0811	0.1056	0.1333	0.1373	0.1334
t_{388}	0.0007	0.0050	0.0381	0.0304	0.0307	0.0267	0.0219	0.0199	0.0238	0.0241
t_{389}	0.0055	0.0061	0.0095	0.0110	0.0133	0.0129	0.0137	0.0128	0.0174	0.0202
t_{390}	0.0016	0.0116	0.0062	0.0285	0.0286	0.0278	0.0307	0.0340	0.0588	0.0819
t_{391}	0.0005	0.0005	0.0009	0.0036	0.0053	0.0053	0.0059	0.0061	0.0067	0.0072
t_{392}	0.0462	0.1398	0.1575	0.1879	0.2421	0.3188	0.3172	0.3621	0.4041	0.4254
t_{393}	0.0079	0.0047	0.0044	0.0035	0.0034	0.0024	0.0051	0.0059	0.0191	0.0286
t_{394}	0.0598	0.0299	0.0228	0.0173	0.0222	0.0251	0.0245	0.0234	0.0269	0.0252
t_{395}	0.0102	0.0200	0.0276	0.0388	0.0547	0.0722	0.0797	0.0914	0.0959	0.1009
t_{396}	0.0025	0.0454	0.0776	0.1461	0.1807	0.2299	0.2451	0.2473	0.2523	0.2857
t_{397}	0.0000	0.0006	0.0009	0.0006	0.0080	0.0064	0.0060	0.0137	0.0130	0.0109
t_{398}	0.0001	0.0006	0.0009	0.0008	0.0010	0.0016	0.0021	0.0017	0.0023	0.0020
t_{399}	0.0278	0.0386	0.0322	0.0373	0.0402	0.0612	0.0558	0.0639	0.0699	0.0983
t_{400}	0.0160	0.0560	0.1199	0.1729	0.2108	0.2511	0.2760	0.2922	0.3275	0.3829

Table 6.5: Measure: AP; Stoplist: NOSTOP; Stemmer: PORTERSTEM; Model: BM25.

TF-IDF - Average Precision (AP) values w.r.t. MAP = 0.0811

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0040	0.0300	0.0721	0.0636	0.0611	0.0685	0.0792	0.0805	0.0798	0.0844
t_{352}	0.0003	0.0007	0.0005	0.0005	0.0005	0.0006	0.0005	0.0004	0.0005	0.0006
t_{353}	0.0084	0.0085	0.0084	0.0190	0.0132	0.0185	0.0273	0.0436	0.0429	0.0605
t_{354}	0.0083	0.0113	0.0147	0.0158	0.0203	0.0288	0.0384	0.0486	0.0478	0.0490
t_{355}	0.0179	0.0204	0.0173	0.0128	0.0099	0.0143	0.0281	0.0284	0.0263	0.0242
t_{356}	0.0000	0.0000	0.0003	0.0009	0.0008	0.0007	0.0007	0.0005	0.0005	0.0004
t_{357}	0.0041	0.0175	0.0175	0.0312	0.0346	0.0531	0.0709	0.0859	0.0827	0.0803
t_{358}	0.0502	0.0298	0.0294	0.0284	0.0270	0.0237	0.0233	0.0149	0.0391	0.0541
t_{359}	0.0001	0.0003	0.0001	0.0445	0.0433	0.0424	0.0417	0.0429	0.0170	0.0178
t_{360}	0.0369	0.0465	0.0688	0.0758	0.0796	0.1048	0.1128	0.1545	0.1772	0.1940
t_{361}	0.0000	0.0000	0.0000	0.0000	0.0370	0.0370	0.0370	0.1111	0.0756	0.0920
t_{362}	0.0012	0.0019	0.0023	0.0011	0.0036	0.0043	0.0045	0.0040	0.0059	0.0081
t_{363}	0.0062	0.0107	0.0076	0.0104	0.0271	0.0274	0.0230	0.0189	0.0374	0.0338
t_{364}	0.0571	0.1150	0.1651	0.1284	0.1590	0.2667	0.2804	0.3590	0.4201	0.4526
t_{365}	0.0177	0.0676	0.1030	0.1107	0.1957	0.1937	0.1874	0.1849	0.2390	0.2270
t_{366}	0.0135	0.0299	0.0411	0.0534	0.0569	0.0589	0.0553	0.0509	0.0594	0.0678
t_{367}	0.0073	0.0319	0.0316	0.0308	0.0325	0.0521	0.0625	0.0702	0.0672	0.0682
t_{368}	0.0338	0.0464	0.0737	0.0930	0.0950	0.0907	0.0817	0.0777	0.1685	0.2030
t_{369}	0.0385	0.0070	0.0180	0.0848	0.0643	0.0512	0.0462	0.0280	0.1146	0.2636
t_{370}	0.0035	0.0044	0.0036	0.0044	0.0065	0.0057	0.0047	0.0034	0.0048	0.0046
t_{371}	0.0007	0.0004	0.0002	0.0004	0.0003	0.0003	0.0003	0.0002	0.0002	0.0002
t_{372}	0.0070	0.0082	0.0096	0.0113	0.0142	0.0135	0.0157	0.0130	0.0192	0.0190
t_{373}	0.0034	0.0109	0.0421	0.1204	0.1126	0.1138	0.1450	0.1425	0.1378	0.1351
t_{374}	0.0137	0.0249	0.0439	0.0460	0.0622	0.0750	0.0859	0.0871	0.0946	0.1104
t_{375}	0.0002	0.0160	0.0288	0.0343	0.0416	0.0471	0.0859	0.1028	0.1055	0.1019
t_{376}	0.0008	0.0003	0.0002	0.0003	0.0003	0.0007	0.0009	0.0012	0.0012	0.0009
t_{377}	0.0256	0.0928	0.0891	0.0870	0.0857	0.1026	0.1020	0.0894	0.1010	0.1067
t_{378}	0.0001	0.0001	0.0003	0.0006	0.0004	0.0005	0.0004	0.0003	0.0003	0.0002
t_{379}	0.0069	0.0037	0.0026	0.0043	0.0050	0.0041	0.0037	0.0032	0.0030	0.0659
t_{380}	0.0057	0.0000	0.0000	0.0000	0.0000	0.0012	0.0011	0.0008	0.0166	0.0229
t_{381}	0.0004	0.0004	0.0003	0.0009	0.0006	0.0010	0.0024	0.0055	0.0051	0.0087
t_{382}	0.0909	0.1191	0.1620	0.1453	0.2054	0.2430	0.2152	0.2458	0.2409	0.2611
t_{383}	0.0017	0.0019	0.0040	0.0087	0.0089	0.0094	0.0081	0.0084	0.0082	0.0085
t_{384}	0.0245	0.0263	0.0325	0.0396	0.0717	0.1084	0.1115	0.1153	0.1361	0.1383
t_{385}	0.0075	0.0115	0.0201	0.0214	0.0279	0.0391	0.0433	0.0699	0.0783	0.0751
t_{386}	0.0000	0.0032	0.0021	0.0023	0.0042	0.0036	0.0032	0.0090	0.0090	0.0080
t_{387}	0.0155	0.0109	0.0414	0.0401	0.0399	0.0499	0.0669	0.0793	0.0800	0.0773
t_{388}	0.0006	0.0010	0.0317	0.0304	0.0359	0.0330	0.0308	0.0298	0.0305	0.0297
t_{389}	0.0009	0.0005	0.0009	0.0009	0.0015	0.0016	0.0017	0.0016	0.0017	0.0025
t_{390}	0.0011	0.0017	0.0019	0.0138	0.0101	0.0089	0.0061	0.0081	0.0167	0.0303
t_{391}	0.0004	0.0005	0.0008	0.0026	0.0034	0.0034	0.0037	0.0035	0.0035	0.0037
t_{392}	0.0436	0.1307	0.1429	0.1670	0.2119	0.2945	0.2936	0.3283	0.3631	0.3503
t_{393}	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	0.0143	0.0237
t_{394}	0.0020	0.0009	0.0005	0.0004	0.0010	0.0009	0.0009	0.0008	0.0007	0.0005
t_{395}	0.0073	0.0155	0.0192	0.0252	0.0353	0.0483	0.0538	0.0620	0.0651	0.0656
t_{396}	0.0003	0.0206	0.0450	0.0877	0.1027	0.1346	0.1360	0.1290	0.1307	0.1554
t_{397}	0.0000	0.0427	0.0166	0.0167	0.0380	0.0356	0.0328	0.0244	0.0479	0.0553
t_{398}	0.0001	0.0002	0.0005	0.0004	0.0004	0.0011	0.0023	0.0021	0.0021	0.0020
t_{399}	0.0099	0.0167	0.0112	0.0101	0.0209	0.0224	0.0215	0.0233	0.0229	0.0232
t_{400}	0.0062	0.0224	0.0535	0.0742	0.0848	0.1073	0.1125	0.1365	0.1611	0.1863

Table 6.6: Measure: AP; Stoplist: NOSTOP; Stemmer: PORTERSTEM; Model: TF-IDF.

BM25 - Average Precision (AP) values w.r.t. MAP = 0.1533

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0012	0.0580	0.1222	0.1184	0.1418	0.1812	0.2307	0.2506	0.2536	0.2741
t_{352}	0.0063	0.0083	0.0067	0.0060	0.0127	0.0193	0.0217	0.0205	0.0235	0.0266
t_{353}	0.0478	0.0619	0.0663	0.0477	0.0459	0.0716	0.0907	0.1394	0.1464	0.1912
t_{354}	0.0117	0.0089	0.0124	0.0156	0.0229	0.0354	0.0471	0.0628	0.0666	0.0681
t_{355}	0.0654	0.0904	0.0562	0.0473	0.0432	0.0441	0.0773	0.0773	0.0754	0.0745
t_{356}	0.0000	0.0000	0.0084	0.0166	0.0152	0.0152	0.0144	0.0116	0.0084	0.0067
t_{357}	0.0153	0.0572	0.0640	0.1035	0.1309	0.1923	0.2588	0.2945	0.2946	0.2893
t_{358}	0.0562	0.0435	0.0432	0.0558	0.0636	0.0606	0.0592	0.0520	0.0613	0.1012
t_{359}	0.0005	0.0005	0.0002	0.0160	0.0177	0.0108	0.0095	0.0118	0.0146	0.0155
t_{360}	0.0488	0.0836	0.1209	0.1414	0.1625	0.1746	0.2043	0.3135	0.3568	0.3841
t_{361}	0.0000	0.0000	0.0000	0.0000	0.0556	0.0556	0.0556	0.1963	0.1852	0.2487
t_{362}	0.0036	0.0129	0.0111	0.0100	0.0260	0.0458	0.0649	0.0615	0.0731	0.0783
t_{363}	0.0329	0.0565	0.0533	0.0406	0.0434	0.0406	0.0326	0.0307	0.0436	0.0332
t_{364}	0.0571	0.1347	0.1527	0.1290	0.1690	0.2832	0.3252	0.4109	0.4923	0.5337
t_{365}	0.0786	0.2019	0.2885	0.3733	0.4965	0.4923	0.4678	0.5383	0.6619	0.6746
t_{366}	0.0237	0.0530	0.0871	0.1457	0.1873	0.2039	0.1982	0.1775	0.2208	0.2531
t_{367}	0.0121	0.0412	0.0351	0.0318	0.0317	0.0525	0.0578	0.0668	0.0638	0.0646
t_{368}	0.0289	0.0427	0.0916	0.1249	0.1582	0.1613	0.1571	0.1568	0.2382	0.3600
t_{369}	0.0385	0.0070	0.0172	0.0913	0.0682	0.0527	0.0491	0.0286	0.1164	0.2685
t_{370}	0.0092	0.0115	0.0068	0.0095	0.0122	0.0118	0.0124	0.0141	0.0162	0.0156
t_{371}	0.0033	0.0000	0.0000	0.0004	0.0002	0.0002	0.0014	0.0018	0.0016	0.0009
t_{372}	0.0309	0.0341	0.0515	0.0485	0.0699	0.0696	0.0703	0.0687	0.0966	0.0939
t_{373}	0.0052	0.0607	0.1048	0.3219	0.2885	0.2968	0.3334	0.3260	0.3247	0.3207
t_{374}	0.0228	0.0395	0.0736	0.0811	0.1264	0.1421	0.1614	0.1747	0.1865	0.2223
t_{375}	0.0008	0.0315	0.0609	0.0723	0.0904	0.1045	0.1626	0.1734	0.1879	0.1796
t_{376}	0.0105	0.0053	0.0040	0.0058	0.0076	0.0120	0.0187	0.0208	0.0254	0.0288
t_{377}	0.0256	0.0730	0.0651	0.0615	0.0605	0.0776	0.0729	0.1172	0.1417	0.1355
t_{378}	0.0003	0.0009	0.0023	0.0035	0.0035	0.0037	0.0039	0.0037	0.0041	0.0047
t_{379}	0.0625	0.0625	0.0625	0.1250	0.1250	0.1250	0.1250	0.1250	0.1562	0.2833
t_{380}	0.0476	0.0238	0.0238	0.0179	0.0238	0.0247	0.0186	0.0185	0.0813	0.1678
t_{381}	0.0013	0.0006	0.0009	0.0011	0.0008	0.0014	0.0026	0.0081	0.0080	0.0352
t_{382}	0.0909	0.1709	0.2622	0.2325	0.3072	0.3657	0.3682	0.3671	0.3645	0.4119
t_{383}	0.0007	0.0018	0.0025	0.0069	0.0059	0.0065	0.0073	0.0104	0.0122	0.0119
t_{384}	0.0265	0.0299	0.0485	0.0558	0.0745	0.1337	0.1468	0.1474	0.1899	0.2191
t_{385}	0.0205	0.0512	0.0852	0.1173	0.1350	0.1672	0.1731	0.2332	0.2441	0.2417
t_{386}	0.0000	0.0097	0.0082	0.0081	0.0084	0.0071	0.0071	0.0238	0.0196	0.0181
t_{387}	0.0232	0.0198	0.0665	0.0683	0.0676	0.0852	0.1039	0.1327	0.1366	0.1337
t_{388}	0.0022	0.0081	0.0471	0.0454	0.0702	0.0682	0.0634	0.0564	0.0777	0.0710
t_{389}	0.0003	0.0007	0.0061	0.0034	0.0036	0.0028	0.0028	0.0025	0.0024	0.0043
t_{390}	0.0002	0.0083	0.0101	0.0331	0.0336	0.0338	0.0409	0.0484	0.0637	0.0816
t_{391}	0.0005	0.0005	0.0009	0.0035	0.0053	0.0053	0.0059	0.0061	0.0067	0.0072
t_{392}	0.0161	0.0475	0.0535	0.0540	0.0881	0.1083	0.1057	0.1177	0.1276	0.1239
t_{393}	0.0221	0.0116	0.0110	0.0097	0.0097	0.0090	0.0114	0.0135	0.0314	0.0462
t_{394}	0.0786	0.0401	0.0455	0.0416	0.0610	0.0527	0.0436	0.0422	0.0374	0.0258
t_{395}	0.0101	0.0199	0.0272	0.0387	0.0545	0.0721	0.0795	0.0913	0.0958	0.1008
t_{396}	0.0050	0.0511	0.0822	0.1530	0.1843	0.2361	0.2529	0.2545	0.2581	0.2888
t_{397}	0.0000	0.0000	0.0000	0.0000	0.0008	0.0006	0.0005	0.0012	0.0019	0.0095
t_{398}	0.0001	0.0002	0.0001	0.0001	0.0001	0.0007	0.0006	0.0005	0.0009	0.0008
t_{399}	0.0296	0.0276	0.0358	0.0282	0.0315	0.0503	0.0454	0.0501	0.0486	0.0740
t_{400}	0.0165	0.0550	0.1095	0.1560	0.1947	0.2324	0.2540	0.2802	0.3076	0.3598

Table 6.7: Measure: AP; Stoplist: NOSTOP; Stemmer: NOSTEM; Model: BM25.

TF-IDF - Average Precision (AP) values w.r.t. MAP = 0.0791

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0044	0.0150	0.0445	0.0412	0.0408	0.0445	0.0501	0.0517	0.0528	0.0554
t_{352}	0.0003	0.0007	0.0005	0.0005	0.0005	0.0006	0.0005	0.0004	0.0005	0.0006
t_{353}	0.0058	0.0059	0.0057	0.0210	0.0169	0.0209	0.0288	0.0467	0.0456	0.0606
t_{354}	0.0074	0.0071	0.0098	0.0110	0.0155	0.0223	0.0364	0.0537	0.0551	0.0568
t_{355}	0.0231	0.0307	0.0226	0.0171	0.0130	0.0168	0.0353	0.0325	0.0303	0.0288
t_{356}	0.0000	0.0000	0.0003	0.0009	0.0008	0.0007	0.0007	0.0006	0.0005	0.0004
t_{357}	0.0086	0.0325	0.0316	0.0540	0.0688	0.1099	0.1538	0.1767	0.1729	0.1737
t_{358}	0.0374	0.0222	0.0248	0.0265	0.0281	0.0240	0.0232	0.0156	0.0221	0.0432
t_{359}	0.0001	0.0003	0.0000	0.0054	0.0049	0.0039	0.0040	0.0047	0.0042	0.0042
t_{360}	0.0385	0.0502	0.0855	0.0939	0.1005	0.1095	0.1266	0.1714	0.1898	0.1933
t_{361}	0.0000	0.0000	0.0000	0.0000	0.0370	0.0370	0.0370	0.0817	0.0574	0.0688
t_{362}	0.0011	0.0015	0.0014	0.0008	0.0032	0.0047	0.0054	0.0054	0.0078	0.0092
t_{363}	0.0088	0.0208	0.0148	0.0212	0.0558	0.0550	0.0529	0.0536	0.1034	0.0977
t_{364}	0.0571	0.1347	0.1835	0.1381	0.1700	0.2833	0.3253	0.4120	0.4692	0.5068
t_{365}	0.0131	0.0677	0.1118	0.1201	0.1943	0.1915	0.1840	0.1844	0.2444	0.2306
t_{366}	0.0102	0.0247	0.0317	0.0360	0.0378	0.0393	0.0356	0.0311	0.0370	0.0508
t_{367}	0.0073	0.0324	0.0319	0.0310	0.0327	0.0523	0.0628	0.0705	0.0675	0.0685
t_{368}	0.0301	0.0437	0.0798	0.1068	0.1147	0.1188	0.1104	0.1095	0.1938	0.2744
t_{369}	0.0385	0.0070	0.0180	0.0848	0.0643	0.0512	0.0462	0.0280	0.1146	0.2636
t_{370}	0.0042	0.0054	0.0047	0.0056	0.0077	0.0066	0.0061	0.0050	0.0059	0.0064
t_{371}	0.0009	0.0005	0.0003	0.0005	0.0004	0.0003	0.0006	0.0005	0.0005	0.0005
t_{372}	0.0110	0.0087	0.0107	0.0092	0.0111	0.0104	0.0104	0.0091	0.0126	0.0139
t_{373}	0.0076	0.0141	0.0147	0.0845	0.0879	0.0828	0.1056	0.1025	0.1028	0.1011
t_{374}	0.0162	0.0305	0.0548	0.0555	0.0742	0.0902	0.1018	0.1036	0.1096	0.1298
t_{375}	0.0002	0.0160	0.0283	0.0339	0.0403	0.0467	0.0842	0.0988	0.1015	0.0978
t_{376}	0.0009	0.0005	0.0003	0.0003	0.0004	0.0009	0.0013	0.0014	0.0014	0.0012
t_{377}	0.0256	0.0465	0.0435	0.0368	0.0349	0.0367	0.0361	0.0436	0.0487	0.0495
t_{378}	0.0001	0.0001	0.0003	0.0006	0.0003	0.0005	0.0003	0.0003	0.0003	0.0002
t_{379}	0.0625	0.0625	0.0625	0.1250	0.1250	0.1250	0.1250	0.1250	0.1562	0.2833
t_{380}	0.0065	0.0032	0.0026	0.0020	0.0018	0.0018	0.0013	0.0011	0.0040	0.0065
t_{381}	0.0006	0.0002	0.0003	0.0005	0.0001	0.0005	0.0021	0.0098	0.0091	0.0136
t_{382}	0.0909	0.1079	0.1353	0.1304	0.1925	0.2351	0.1957	0.1954	0.1928	0.2098
t_{383}	0.0007	0.0014	0.0032	0.0053	0.0047	0.0054	0.0056	0.0069	0.0071	0.0070
t_{384}	0.0240	0.0264	0.0309	0.0376	0.0700	0.1052	0.1081	0.1124	0.1345	0.1430
t_{385}	0.0047	0.0093	0.0140	0.0137	0.0193	0.0235	0.0262	0.0359	0.0321	0.0303
t_{386}	0.0000	0.0004	0.0003	0.0002	0.0004	0.0002	0.0001	0.0024	0.0008	0.0005
t_{387}	0.0168	0.0114	0.0429	0.0412	0.0399	0.0493	0.0641	0.0731	0.0746	0.0732
t_{388}	0.0015	0.0029	0.0413	0.0383	0.0613	0.0610	0.0610	0.0601	0.0718	0.0461
t_{389}	0.0001	0.0002	0.0002	0.0002	0.0004	0.0004	0.0004	0.0003	0.0003	0.0006
t_{390}	0.0002	0.0010	0.0010	0.0056	0.0044	0.0044	0.0042	0.0054	0.0067	0.0190
t_{391}	0.0004	0.0005	0.0008	0.0026	0.0034	0.0034	0.0037	0.0035	0.0035	0.0037
t_{392}	0.0148	0.0453	0.0473	0.0483	0.0819	0.1010	0.0965	0.1082	0.1173	0.1048
t_{393}	0.0010	0.0005	0.0003	0.0001	0.0001	0.0001	0.0002	0.0002	0.0143	0.0284
t_{394}	0.0215	0.0058	0.0044	0.0034	0.0032	0.0028	0.0022	0.0021	0.0015	0.0013
t_{395}	0.0072	0.0155	0.0188	0.0252	0.0352	0.0483	0.0538	0.0618	0.0651	0.0659
t_{396}	0.0003	0.0207	0.0429	0.0821	0.0908	0.1237	0.1227	0.1175	0.1160	0.1385
t_{397}	0.0000	0.0000	0.0000	0.0000	0.0001	0.0001	0.0001	0.0001	0.0011	0.0072
t_{398}	0.0001	0.0001	0.0001	0.0000	0.0001	0.0006	0.0004	0.0004	0.0005	0.0005
t_{399}	0.0031	0.0038	0.0038	0.0039	0.0140	0.0104	0.0095	0.0144	0.0140	0.0110
t_{400}	0.0066	0.0215	0.0538	0.0741	0.0820	0.1057	0.1079	0.1288	0.1539	0.1728

Table 6.8: Measure: AP; Stoplist: NOSTOP; Stemmer: NOSTEM; Model: TF-IDF.

BM25 - normalised Discounted Cumulative Gain (nDCG) values w.r.t. Overall nDCG = 0.3908

Bucket / Topic	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0451	0.2875	0.3747	0.3702	0.4022	0.4465	0.5362	0.5726	0.6087	0.6362
t_{352}	0.0553	0.0658	0.0611	0.0570	0.0870	0.0992	0.1051	0.1037	0.1170	0.1236
t_{353}	0.1154	0.1453	0.1727	0.1833	0.1898	0.2440	0.2737	0.3642	0.3809	0.4434
t_{354}	0.0814	0.1220	0.1524	0.1694	0.1986	0.2366	0.2721	0.2915	0.2959	0.2987
t_{355}	0.2318	0.2780	0.2481	0.2358	0.2249	0.2927	0.4121	0.4237	0.3936	0.3909
t_{356}	0.0000	0.0000	0.0414	0.0773	0.0757	0.0752	0.0752	0.0702	0.0653	0.0619
t_{357}	0.0909	0.1933	0.2233	0.2858	0.3258	0.4398	0.5326	0.5643	0.5632	0.5597
t_{358}	0.2588	0.2571	0.2604	0.3119	0.3347	0.3331	0.3294	0.3144	0.3717	0.4781
t_{359}	0.0170	0.0150	0.0138	0.0968	0.1175	0.1118	0.1082	0.1352	0.1477	0.1628
t_{360}	0.1434	0.1896	0.2544	0.2838	0.3058	0.3359	0.3560	0.5153	0.5480	0.5763
t_{361}	0.0000	0.0000	0.0000	0.0000	0.1483	0.1483	0.1483	0.3670	0.3670	0.4916
t_{362}	0.0606	0.1061	0.1325	0.1211	0.2071	0.2442	0.2749	0.3016	0.3068	0.3039
t_{363}	0.1875	0.1398	0.1282	0.1711	0.2025	0.2200	0.1956	0.2069	0.2415	0.1810
t_{364}	0.1608	0.2670	0.3153	0.3154	0.3747	0.4988	0.5413	0.6339	0.7598	0.8089
t_{365}	0.2032	0.3732	0.4569	0.5505	0.6516	0.6620	0.6586	0.7396	0.8562	0.8720
t_{366}	0.1122	0.1883	0.2773	0.3953	0.4715	0.4905	0.4953	0.4806	0.5531	0.5951
t_{367}	0.0682	0.1442	0.1499	0.1532	0.1659	0.2290	0.2579	0.2931	0.2931	0.3018
t_{368}	0.1095	0.1613	0.2441	0.3151	0.3700	0.3849	0.3800	0.3850	0.4980	0.6050
t_{369}	0.1178	0.0562	0.0995	0.2480	0.2236	0.2008	0.1970	0.1619	0.3443	0.5818
t_{370}	0.0751	0.0851	0.0820	0.1051	0.1163	0.1178	0.1149	0.1230	0.1313	0.1291
t_{371}	0.0353	0.0287	0.0260	0.0608	0.0577	0.0408	0.0583	0.0746	0.0733	0.0548
t_{372}	0.1474	0.1836	0.2186	0.2311	0.2654	0.2647	0.2898	0.2883	0.3839	0.3865
t_{373}	0.0256	0.1940	0.2604	0.4703	0.4835	0.4999	0.5485	0.5448	0.5313	0.5386
t_{374}	0.1146	0.1619	0.2335	0.2504	0.3185	0.3708	0.4092	0.4340	0.4670	0.5304
t_{375}	0.0267	0.1695	0.2535	0.2786	0.3132	0.3515	0.4241	0.4375	0.4778	0.4675
t_{376}	0.0693	0.0681	0.0736	0.0956	0.1125	0.1420	0.1799	0.1799	0.1933	0.2075
t_{377}	0.0917	0.3508	0.3447	0.3452	0.3784	0.4236	0.4716	0.5491	0.6296	0.6546
t_{378}	0.0197	0.0350	0.0616	0.0816	0.0894	0.0948	0.0989	0.1082	0.1134	0.1104
t_{379}	0.0936	0.0795	0.0735	0.1145	0.1294	0.1103	0.1083	0.1050	0.0868	0.1886
t_{380}	0.1063	0.0795	0.0743	0.0704	0.0704	0.1005	0.0969	0.0949	0.3258	0.4745
t_{381}	0.0438	0.0535	0.0662	0.0825	0.0792	0.0914	0.0952	0.1310	0.1293	0.2648
t_{382}	0.2179	0.3995	0.5196	0.5056	0.5971	0.6581	0.6906	0.7522	0.7678	0.8240
t_{383}	0.0505	0.0642	0.0762	0.1261	0.1287	0.1246	0.1259	0.1259	0.1212	0.1176
t_{384}	0.1152	0.1505	0.2065	0.2543	0.3294	0.4838	0.5232	0.5482	0.6160	0.6611
t_{385}	0.1010	0.2295	0.3120	0.3597	0.4043	0.4824	0.5154	0.5838	0.6124	0.6207
t_{386}	0.0000	0.1036	0.0889	0.0872	0.1154	0.1094	0.1253	0.1893	0.2036	0.1811
t_{387}	0.1188	0.1234	0.2495	0.2890	0.3105	0.3714	0.4503	0.5515	0.5692	0.5656
t_{388}	0.0239	0.0611	0.1679	0.1656	0.1771	0.1690	0.1611	0.1650	0.1910	0.1858
t_{389}	0.0507	0.0571	0.0774	0.0885	0.1136	0.1231	0.1237	0.1247	0.1376	0.1478
t_{390}	0.0507	0.1230	0.1019	0.1758	0.1752	0.1739	0.1785	0.1801	0.2347	0.2571
t_{391}	0.0304	0.0362	0.0542	0.0914	0.1224	0.1219	0.1361	0.1349	0.1479	0.1412
t_{392}	0.1740	0.3465	0.3850	0.4396	0.5284	0.6241	0.6372	0.7077	0.7654	0.8203
t_{393}	0.0654	0.0445	0.0434	0.0382	0.0378	0.0322	0.0661	0.0702	0.1255	0.1514
t_{394}	0.1955	0.1193	0.1172	0.1082	0.1678	0.2210	0.2188	0.2154	0.2416	0.2366
t_{395}	0.0882	0.1379	0.1633	0.2063	0.2648	0.3199	0.3445	0.3738	0.3924	0.4079
t_{396}	0.0253	0.1686	0.2270	0.3123	0.3537	0.4153	0.4315	0.4416	0.4499	0.4837
t_{397}	0.0000	0.0313	0.0444	0.0421	0.1069	0.1127	0.1216	0.1349	0.1341	0.1281
t_{398}	0.0093	0.0222	0.0351	0.0338	0.0390	0.0551	0.0633	0.0579	0.0699	0.0657
t_{399}	0.1162	0.1597	0.1499	0.1820	0.1786	0.2217	0.2204	0.2397	0.2562	0.3027
t_{400}	0.0910	0.1991	0.3151	0.4016	0.4679	0.5576	0.6050	0.6573	0.7024	0.7605

Table 6.9: Measure: nDCG; Stoplist: INDRI; Stemmer: PORTERSTEM; Model: BM25.

TF-IDF - normalised Discounted Cumulative Gain (nDCG) values w.r.t. Overall nDCG = 0.2975

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0532	0.1731	0.2912	0.2796	0.2951	0.3149	0.3633	0.3821	0.4106	0.4320
t_{352}	0.0194	0.0272	0.0249	0.0190	0.0293	0.0275	0.0271	0.0262	0.0292	0.0327
t_{353}	0.0804	0.0940	0.1110	0.1500	0.1364	0.1767	0.2072	0.2779	0.2814	0.3175
t_{354}	0.0792	0.0950	0.1101	0.1190	0.1420	0.1805	0.2080	0.2389	0.2410	0.2460
t_{355}	0.1247	0.1546	0.1547	0.1496	0.1331	0.1906	0.2717	0.2963	0.2754	0.2712
t_{356}	0.0000	0.0000	0.0183	0.0392	0.0383	0.0378	0.0374	0.0360	0.0197	0.0191
t_{357}	0.0593	0.1276	0.1437	0.1978	0.2169	0.2756	0.3346	0.3699	0.3592	0.3617
t_{358}	0.2185	0.2106	0.1938	0.1853	0.1899	0.1762	0.1723	0.1140	0.2037	0.2466
t_{359}	0.0147	0.0383	0.0359	0.1613	0.1710	0.1400	0.1648	0.1816	0.1477	0.1678
t_{360}	0.1591	0.2131	0.2769	0.3146	0.3370	0.3822	0.4105	0.5345	0.5646	0.5760
t_{361}	0.0000	0.0000	0.0000	0.0000	0.1175	0.1175	0.1175	0.2629	0.2310	0.3094
t_{362}	0.0485	0.0689	0.0939	0.0625	0.0887	0.1099	0.1209	0.1199	0.1443	0.1679
t_{363}	0.0639	0.1163	0.1063	0.1474	0.2396	0.2568	0.2360	0.2330	0.3131	0.3093
t_{364}	0.1608	0.2678	0.3701	0.3185	0.3680	0.4968	0.5409	0.6345	0.7086	0.7557
t_{365}	0.1415	0.2783	0.3502	0.3919	0.4977	0.5050	0.4900	0.5286	0.6038	0.5995
t_{366}	0.0759	0.1052	0.1491	0.2033	0.2242	0.2363	0.2247	0.2171	0.2404	0.2862
t_{367}	0.0688	0.1385	0.1506	0.1581	0.1692	0.2291	0.2657	0.2978	0.2971	0.3058
t_{368}	0.1227	0.1671	0.2372	0.2851	0.3234	0.3348	0.3248	0.3254	0.4572	0.5448
t_{369}	0.1178	0.0562	0.0995	0.2480	0.2176	0.1967	0.1935	0.1605	0.3437	0.5781
t_{370}	0.0529	0.0707	0.0691	0.0818	0.0905	0.0815	0.0646	0.0545	0.0620	0.0664
t_{371}	0.0243	0.0211	0.0195	0.0353	0.0339	0.0333	0.0336	0.0331	0.0325	0.0329
t_{372}	0.0766	0.0993	0.1042	0.1209	0.1440	0.1411	0.1565	0.1413	0.1935	0.1993
t_{373}	0.0309	0.0811	0.1351	0.3364	0.3475	0.3524	0.3963	0.3931	0.3809	0.3767
t_{374}	0.0978	0.1450	0.1988	0.2170	0.2784	0.3211	0.3526	0.3571	0.3800	0.4149
t_{375}	0.0203	0.1411	0.1961	0.1987	0.2237	0.2348	0.3179	0.3481	0.3695	0.3589
t_{376}	0.0333	0.0293	0.0226	0.0273	0.0322	0.0540	0.0642	0.0727	0.0673	0.0568
t_{377}	0.0917	0.2899	0.2938	0.2904	0.2974	0.3470	0.3834	0.3779	0.4118	0.4489
t_{378}	0.0112	0.0179	0.0244	0.0403	0.0340	0.0444	0.0385	0.0380	0.0425	0.0478
t_{379}	0.0851	0.0738	0.0684	0.1067	0.1240	0.1053	0.1033	0.1008	0.0828	0.2453
t_{380}	0.0555	0.0453	0.0429	0.0406	0.0395	0.0660	0.0374	0.0358	0.1584	0.2084
t_{381}	0.0381	0.0390	0.0380	0.0443	0.0311	0.0456	0.0662	0.1149	0.1012	0.1383
t_{382}	0.2179	0.3511	0.3879	0.3778	0.4520	0.5052	0.5005	0.5484	0.5471	0.5926
t_{383}	0.0548	0.0524	0.0860	0.1211	0.1319	0.1365	0.1297	0.1246	0.1229	0.1233
t_{384}	0.1159	0.1456	0.1811	0.2253	0.3330	0.4776	0.5088	0.5336	0.5703	0.6050
t_{385}	0.0717	0.1175	0.1647	0.1913	0.2363	0.2941	0.3084	0.3586	0.3788	0.3850
t_{386}	0.0000	0.0677	0.0619	0.0594	0.0827	0.0807	0.0642	0.0921	0.1235	0.1337
t_{387}	0.1130	0.1112	0.2213	0.2493	0.2584	0.3122	0.3857	0.4407	0.4464	0.4443
t_{388}	0.0222	0.0336	0.1652	0.1618	0.1894	0.1810	0.1843	0.1910	0.2212	0.1964
t_{389}	0.0238	0.0238	0.0329	0.0383	0.0518	0.0516	0.0556	0.0515	0.0578	0.0678
t_{390}	0.0442	0.0703	0.0681	0.1229	0.1243	0.0990	0.0926	0.0982	0.1328	0.1783
t_{391}	0.0299	0.0357	0.0500	0.0921	0.1072	0.1003	0.1105	0.1060	0.1095	0.1088
t_{392}	0.1510	0.3155	0.3412	0.3920	0.4846	0.6112	0.6247	0.6926	0.7473	0.7707
t_{393}	0.0147	0.0072	0.0068	0.0000	0.0000	0.0000	0.0100	0.0162	0.0844	0.1226
t_{394}	0.0645	0.0413	0.0227	0.0216	0.0571	0.0560	0.0555	0.0388	0.0380	0.0374
t_{395}	0.0806	0.1468	0.1648	0.2032	0.2533	0.2929	0.3087	0.3425	0.3572	0.3655
t_{396}	0.0179	0.1216	0.1809	0.2728	0.2919	0.3515	0.3533	0.3561	0.3638	0.4019
t_{397}	0.0000	0.1522	0.1205	0.1239	0.2172	0.2242	0.2234	0.2077	0.2662	0.3376
t_{398}	0.0090	0.0183	0.0310	0.0298	0.0298	0.0422	0.0583	0.0499	0.0540	0.0498
t_{399}	0.0719	0.1148	0.1134	0.1197	0.1569	0.1730	0.1761	0.1944	0.2037	0.2067
t_{400}	0.0700	0.1580	0.2616	0.3349	0.3820	0.4563	0.4868	0.5412	0.5909	0.6255

Table 6.10: Measure: nDCG; Stoplist: INDRI; Stemmer: PORTERSTEM; Model: TF-IDF.

BM25 - normalised Discounted Cumulative Gain (nDCG) values w.r.t. Overall nDCG = 0.3781

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0290	0.2215	0.3224	0.3112	0.3487	0.3958	0.4759	0.4955	0.5276	0.5496
t_{352}	0.0558	0.0685	0.0613	0.0575	0.0895	0.0994	0.1077	0.1038	0.1171	0.1236
t_{353}	0.1499	0.1862	0.1961	0.1919	0.1952	0.2522	0.3027	0.3839	0.4038	0.4577
t_{354}	0.0740	0.0794	0.0999	0.1209	0.1499	0.1817	0.2234	0.2587	0.2673	0.2783
t_{355}	0.2095	0.2565	0.2255	0.2131	0.2050	0.2507	0.3299	0.3639	0.3603	0.3590
t_{356}	0.0000	0.0000	0.0440	0.0833	0.0811	0.0804	0.0798	0.0743	0.0680	0.0638
t_{357}	0.0973	0.1885	0.2223	0.2889	0.3329	0.4287	0.5231	0.5712	0.5740	0.5739
t_{358}	0.1903	0.1844	0.1847	0.2368	0.2517	0.2491	0.2468	0.2354	0.2694	0.3452
t_{359}	0.0190	0.0406	0.0152	0.1703	0.1991	0.1532	0.1521	0.1691	0.1731	0.1939
t_{360}	0.1670	0.2263	0.2952	0.3147	0.3444	0.3740	0.4311	0.5741	0.6218	0.6512
t_{361}	0.0000	0.0000	0.0000	0.0000	0.1483	0.1483	0.1483	0.3567	0.3495	0.4410
t_{362}	0.0525	0.0984	0.1219	0.1176	0.1787	0.2280	0.2846	0.3105	0.3456	0.3401
t_{363}	0.1295	0.2027	0.1988	0.2272	0.2835	0.2912	0.2770	0.2880	0.3539	0.3047
t_{364}	0.1608	0.2779	0.3214	0.3192	0.3778	0.5039	0.5583	0.6527	0.7772	0.8267
t_{365}	0.2032	0.3732	0.4592	0.5540	0.6542	0.6654	0.6615	0.7457	0.8605	0.8783
t_{366}	0.1063	0.1832	0.2703	0.3802	0.4372	0.4614	0.4664	0.4538	0.5280	0.5663
t_{367}	0.0682	0.1460	0.1509	0.1538	0.1663	0.2293	0.2583	0.2934	0.2934	0.3021
t_{368}	0.1143	0.1481	0.2374	0.2966	0.3352	0.3473	0.3435	0.3431	0.4395	0.5870
t_{369}	0.1178	0.0562	0.0995	0.2480	0.2236	0.2008	0.1970	0.1619	0.3443	0.5818
t_{370}	0.0679	0.0881	0.0779	0.0955	0.1074	0.1077	0.1111	0.1224	0.1250	0.1199
t_{371}	0.0371	0.0302	0.0274	0.0625	0.0593	0.0422	0.0599	0.0765	0.0750	0.0561
t_{372}	0.1134	0.1390	0.1887	0.2145	0.2287	0.2277	0.2409	0.2397	0.3118	0.3080
t_{373}	0.0469	0.2112	0.2816	0.5232	0.5043	0.5221	0.5644	0.5597	0.5954	0.6043
t_{374}	0.1177	0.1798	0.2584	0.2809	0.3616	0.4121	0.4506	0.4812	0.5105	0.5668
t_{375}	0.0267	0.1696	0.2537	0.2787	0.3137	0.3519	0.4245	0.4380	0.4786	0.4736
t_{376}	0.0751	0.0718	0.0777	0.1047	0.1220	0.1529	0.1928	0.2024	0.2204	0.2256
t_{377}	0.0917	0.2422	0.2307	0.2247	0.2213	0.2552	0.2778	0.3704	0.4536	0.4292
t_{378}	0.0199	0.0350	0.0617	0.0762	0.0893	0.0998	0.1038	0.1034	0.1088	0.1157
t_{379}	0.1638	0.1638	0.1638	0.2671	0.2671	0.2671	0.2671	0.2671	0.3254	0.4826
t_{380}	0.1374	0.0979	0.0916	0.0827	0.0827	0.1163	0.1093	0.1084	0.2896	0.4246
t_{381}	0.0470	0.0413	0.0534	0.0645	0.0625	0.0768	0.0969	0.1317	0.1296	0.2659
t_{382}	0.2179	0.3784	0.4898	0.4703	0.5777	0.6421	0.6600	0.6913	0.7050	0.7635
t_{383}	0.0358	0.0627	0.0763	0.1192	0.1062	0.1146	0.1124	0.1194	0.1345	0.1352
t_{384}	0.1156	0.1519	0.2067	0.2545	0.3315	0.4853	0.5261	0.5534	0.6352	0.6840
t_{385}	0.1141	0.2183	0.2903	0.3485	0.3980	0.4707	0.5046	0.5707	0.5739	0.5784
t_{386}	0.0000	0.0933	0.0866	0.0853	0.1013	0.0822	0.0965	0.1500	0.1616	0.1872
t_{387}	0.1325	0.1420	0.2691	0.2984	0.3241	0.3796	0.4550	0.5467	0.5646	0.5663
t_{388}	0.0433	0.0925	0.1764	0.1890	0.2580	0.2628	0.2710	0.2628	0.3307	0.3356
t_{389}	0.0190	0.0331	0.0639	0.0557	0.0663	0.0651	0.0563	0.0551	0.0546	0.0663
t_{390}	0.0161	0.0754	0.1003	0.1557	0.1541	0.1548	0.1530	0.1660	0.2008	0.2439
t_{391}	0.0304	0.0362	0.0542	0.0914	0.1224	0.1219	0.1361	0.1349	0.1479	0.1412
t_{392}	0.0778	0.1635	0.2036	0.2150	0.2751	0.3119	0.3098	0.3379	0.3696	0.3745
t_{393}	0.0987	0.0703	0.0683	0.0651	0.0648	0.0622	0.0804	0.0932	0.1487	0.1837
t_{394}	0.2348	0.1469	0.1775	0.1676	0.2174	0.2277	0.2156	0.2128	0.1882	0.1533
t_{395}	0.0881	0.1378	0.1632	0.2090	0.2648	0.3198	0.3444	0.3737	0.3923	0.4078
t_{396}	0.0285	0.1761	0.2314	0.3238	0.3645	0.4271	0.4362	0.4467	0.4549	0.4855
t_{397}	0.0000	0.0000	0.0000	0.0000	0.0220	0.0202	0.0188	0.0362	0.0521	0.1182
t_{398}	0.0100	0.0104	0.0135	0.0093	0.0176	0.0330	0.0281	0.0276	0.0400	0.0433
t_{399}	0.1066	0.1154	0.1359	0.1370	0.1381	0.1636	0.1555	0.1661	0.1777	0.2226
t_{400}	0.0909	0.1951	0.3042	0.3830	0.4432	0.5200	0.5635	0.6231	0.6667	0.7154

Table 6.11: Measure: nDCG; Stoplist: INDRI; Stemmer: NOSTEM; Model: BM25.

TF-IDF - normalised Discounted Cumulative Gain (nDCG) values w.r.t. Overall nDCG = 0.2813

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0507	0.1342	0.2228	0.2192	0.2420	0.2677	0.3017	0.2999	0.3233	0.3419
t_{352}	0.0195	0.0273	0.0248	0.0190	0.0293	0.0301	0.0271	0.0263	0.0293	0.0327
t_{353}	0.0658	0.0836	0.0912	0.1668	0.1483	0.1862	0.2147	0.2990	0.3016	0.3411
t_{354}	0.0586	0.0674	0.0816	0.0952	0.1163	0.1376	0.1836	0.2259	0.2341	0.2380
t_{355}	0.1148	0.1730	0.1697	0.1663	0.1553	0.2013	0.2682	0.2811	0.2694	0.2661
t_{356}	0.0000	0.0000	0.0185	0.0397	0.0386	0.0382	0.0375	0.0364	0.0198	0.0192
t_{357}	0.0752	0.1539	0.1721	0.2201	0.2609	0.3307	0.4068	0.4526	0.4515	0.4584
t_{358}	0.1629	0.1509	0.1448	0.1667	0.1844	0.1764	0.1748	0.1499	0.1767	0.2515
t_{359}	0.0147	0.0487	0.0121	0.0901	0.0980	0.0768	0.0903	0.1039	0.0989	0.1080
t_{360}	0.1580	0.2116	0.2826	0.2900	0.3079	0.3405	0.3894	0.4988	0.5281	0.5572
t_{361}	0.0000	0.0000	0.0000	0.0000	0.1175	0.1175	0.1175	0.2451	0.2195	0.2905
t_{362}	0.0392	0.0583	0.0547	0.0526	0.0871	0.1121	0.1324	0.1415	0.1577	0.1619
t_{363}	0.0818	0.1439	0.1277	0.1566	0.3473	0.3628	0.3579	0.3736	0.4939	0.4710
t_{364}	0.1608	0.2793	0.3779	0.3230	0.3719	0.5026	0.5581	0.6544	0.7268	0.7750
t_{365}	0.1421	0.2787	0.3550	0.3968	0.4957	0.5016	0.4867	0.5242	0.6019	0.5997
t_{366}	0.0775	0.1049	0.1489	0.1911	0.2173	0.2265	0.2243	0.2165	0.2437	0.3102
t_{367}	0.0688	0.1391	0.1510	0.1584	0.1694	0.2293	0.2660	0.2979	0.2974	0.3061
t_{368}	0.1210	0.1530	0.2316	0.2862	0.3134	0.3252	0.3198	0.3191	0.4236	0.5589
t_{369}	0.1178	0.0562	0.0995	0.2480	0.2176	0.1967	0.1935	0.1605	0.3437	0.5781
t_{370}	0.0502	0.0686	0.0665	0.0769	0.0910	0.0855	0.0794	0.0753	0.0778	0.0800
t_{371}	0.0258	0.0223	0.0207	0.0373	0.0356	0.0350	0.0514	0.0505	0.0497	0.0505
t_{372}	0.0918	0.0772	0.0950	0.1011	0.1141	0.1110	0.1287	0.1254	0.1584	0.1665
t_{373}	0.0397	0.0952	0.1171	0.3082	0.3334	0.3271	0.3733	0.3693	0.3798	0.3669
t_{374}	0.1066	0.1697	0.2413	0.2576	0.3104	0.3635	0.3935	0.4034	0.4183	0.4689
t_{375}	0.0203	0.1405	0.1925	0.1949	0.2197	0.2308	0.3163	0.3483	0.3670	0.3582
t_{376}	0.0394	0.0303	0.0233	0.0281	0.0378	0.0594	0.0701	0.0834	0.0786	0.0729
t_{377}	0.0917	0.2237	0.2165	0.2008	0.1973	0.2126	0.2341	0.2792	0.3336	0.3325
t_{378}	0.0111	0.0179	0.0292	0.0397	0.0286	0.0390	0.0379	0.0374	0.0420	0.0423
t_{379}	0.1638	0.1638	0.1638	0.2671	0.2671	0.2671	0.2671	0.2671	0.3254	0.4939
t_{380}	0.0592	0.0475	0.0453	0.0431	0.0415	0.0685	0.0668	0.0375	0.0856	0.1216
t_{381}	0.0405	0.0252	0.0362	0.0472	0.0234	0.0249	0.0527	0.1252	0.0928	0.1315
t_{382}	0.2179	0.3419	0.3676	0.3646	0.4454	0.4985	0.4852	0.5113	0.4958	0.5402
t_{383}	0.0359	0.0596	0.0863	0.1012	0.0883	0.1011	0.1016	0.1109	0.1098	0.1113
t_{384}	0.1124	0.1448	0.1748	0.2187	0.3292	0.4716	0.5035	0.5286	0.5893	0.6193
t_{385}	0.0646	0.1110	0.1393	0.1590	0.2015	0.2425	0.2532	0.2836	0.2810	0.2937
t_{386}	0.0000	0.0601	0.0557	0.0392	0.0683	0.0665	0.0359	0.0682	0.0999	0.0813
t_{387}	0.1148	0.1122	0.2240	0.2498	0.2693	0.3125	0.3763	0.4388	0.4402	0.4281
t_{388}	0.0366	0.0681	0.1914	0.2012	0.2522	0.2596	0.2678	0.2820	0.3367	0.2917
t_{389}	0.0131	0.0201	0.0156	0.0187	0.0278	0.0307	0.0310	0.0275	0.0242	0.0313
t_{390}	0.0201	0.0442	0.0624	0.0678	0.0727	0.0728	0.0726	0.0723	0.0842	0.1335
t_{391}	0.0299	0.0357	0.0500	0.0921	0.1072	0.1003	0.1105	0.1060	0.1095	0.1088
t_{392}	0.0726	0.1641	0.1778	0.1884	0.2672	0.3055	0.3032	0.3295	0.3611	0.3400
t_{393}	0.0305	0.0199	0.0183	0.0094	0.0090	0.0088	0.0167	0.0232	0.0900	0.1280
t_{394}	0.0942	0.0669	0.0794	0.0765	0.0952	0.0920	0.0697	0.0856	0.0493	0.0482
t_{395}	0.0805	0.1467	0.1648	0.2032	0.2530	0.2930	0.3059	0.3425	0.3570	0.3656
t_{396}	0.0186	0.1303	0.1771	0.2633	0.2903	0.3422	0.3504	0.3545	0.3535	0.3877
t_{397}	0.0000	0.0000	0.0000	0.0000	0.0143	0.0138	0.0134	0.0132	0.0343	0.0540
t_{398}	0.0097	0.0093	0.0088	0.0084	0.0119	0.0253	0.0202	0.0199	0.0237	0.0235
t_{399}	0.0437	0.0638	0.0729	0.0865	0.1094	0.1086	0.1061	0.1108	0.1146	0.1419
t_{400}	0.0724	0.1537	0.2609	0.3297	0.3558	0.4321	0.4625	0.5149	0.5529	0.5844

Table 6.12: Measure: nDCG; Stoplist: INDRI; Stemmer: NOSTEM; Model: TF-IDF.

BM25 - normalised Discounted Cumulative Gain (nDCG) values w.r.t. Overall nDCG = 0.3951

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0385	0.2777	0.3705	0.3656	0.3974	0.4413	0.5309	0.5673	0.6021	0.6217
t_{352}	0.0535	0.0636	0.0558	0.0566	0.0816	0.0996	0.1053	0.1036	0.1177	0.1242
t_{353}	0.1187	0.1477	0.1724	0.1783	0.1865	0.2415	0.2713	0.3543	0.3756	0.4427
t_{354}	0.0830	0.1229	0.1542	0.1740	0.2077	0.2452	0.2826	0.3085	0.3092	0.3141
t_{355}	0.2424	0.2956	0.2615	0.2502	0.2458	0.2913	0.4210	0.4311	0.4183	0.4072
t_{356}	0.0000	0.0000	0.0456	0.0808	0.0789	0.0789	0.0781	0.0737	0.0680	0.0640
t_{357}	0.0920	0.1901	0.2200	0.2835	0.3258	0.4382	0.5260	0.5726	0.5735	0.5764
t_{358}	0.2608	0.2567	0.2717	0.3210	0.3441	0.3422	0.3393	0.3256	0.3825	0.4769
t_{359}	0.0163	0.0143	0.0133	0.0960	0.1185	0.1126	0.1109	0.1380	0.1532	0.1679
t_{360}	0.1442	0.1891	0.2610	0.2898	0.3115	0.3464	0.3661	0.5279	0.5566	0.5822
t_{361}	0.0000	0.0000	0.0000	0.0000	0.1483	0.1483	0.1483	0.3670	0.3670	0.4848
t_{362}	0.0607	0.1125	0.1374	0.1287	0.1992	0.2596	0.2961	0.3103	0.3191	0.3127
t_{363}	0.1877	0.1408	0.1294	0.1723	0.2023	0.2179	0.1937	0.2059	0.2418	0.1808
t_{364}	0.1608	0.2704	0.3089	0.2968	0.3501	0.4882	0.5331	0.6274	0.7430	0.7917
t_{365}	0.2032	0.3757	0.4573	0.5506	0.6532	0.6630	0.6551	0.7384	0.8662	0.8826
t_{366}	0.1489	0.2334	0.3482	0.4745	0.5730	0.6039	0.6289	0.6259	0.6719	0.7186
t_{367}	0.0695	0.1432	0.1499	0.1512	0.1635	0.2312	0.2600	0.2953	0.2949	0.3037
t_{368}	0.1116	0.1612	0.2479	0.3158	0.3697	0.3837	0.3786	0.3768	0.4898	0.5937
t_{369}	0.1178	0.0521	0.0988	0.2586	0.2285	0.2042	0.1998	0.1633	0.3463	0.5824
t_{370}	0.0756	0.0915	0.0801	0.1078	0.1232	0.1226	0.1195	0.1258	0.1415	0.1368
t_{371}	0.0353	0.0000	0.0000	0.0360	0.0188	0.0185	0.0364	0.0532	0.0523	0.0344
t_{372}	0.1464	0.1824	0.2262	0.2703	0.3009	0.2838	0.2996	0.2983	0.3854	0.3928
t_{373}	0.0262	0.2012	0.2829	0.5065	0.5197	0.5386	0.5744	0.5717	0.5962	0.5978
t_{374}	0.1145	0.1622	0.2340	0.2444	0.3140	0.3564	0.3959	0.4249	0.4587	0.5168
t_{375}	0.0256	0.1740	0.2579	0.2774	0.3145	0.3502	0.4121	0.4268	0.4599	0.4500
t_{376}	0.0697	0.0678	0.0638	0.0959	0.1128	0.1470	0.1722	0.1740	0.1875	0.2058
t_{377}	0.0917	0.3476	0.3436	0.3456	0.3741	0.4198	0.4645	0.5329	0.6157	0.6527
t_{378}	0.0192	0.0341	0.0559	0.0749	0.0873	0.0876	0.0966	0.0958	0.1009	0.1027
t_{379}	0.0952	0.0808	0.0741	0.1146	0.1310	0.1120	0.1100	0.1066	0.0880	0.2500
t_{380}	0.1063	0.0795	0.0743	0.0704	0.0704	0.0998	0.0964	0.0944	0.3235	0.4890
t_{381}	0.0441	0.0420	0.0660	0.0832	0.0797	0.1032	0.0962	0.1308	0.1289	0.2659
t_{382}	0.2179	0.3914	0.5152	0.5022	0.5951	0.6553	0.6887	0.7505	0.7684	0.8233
t_{383}	0.0508	0.0614	0.0842	0.1186	0.1280	0.1269	0.1243	0.1195	0.1170	0.1139
t_{384}	0.1153	0.1500	0.2063	0.2545	0.3286	0.4815	0.5205	0.5466	0.6149	0.6584
t_{385}	0.1013	0.2294	0.3106	0.3631	0.4012	0.4790	0.5192	0.5850	0.6140	0.6171
t_{386}	0.0000	0.1048	0.0879	0.1062	0.1364	0.1307	0.1463	0.2624	0.2749	0.2335
t_{387}	0.1271	0.1324	0.2558	0.2941	0.3094	0.3755	0.4585	0.5537	0.5712	0.5674
t_{388}	0.0243	0.0612	0.1680	0.1564	0.1647	0.1639	0.1555	0.1596	0.1676	0.1809
t_{389}	0.0504	0.0569	0.0749	0.0849	0.1085	0.1158	0.1230	0.1183	0.1335	0.1468
t_{390}	0.0505	0.1219	0.0978	0.1687	0.1765	0.1707	0.1676	0.1693	0.2249	0.2523
t_{391}	0.0299	0.0355	0.0439	0.0927	0.1139	0.1165	0.1207	0.1261	0.1300	0.1360
t_{392}	0.1729	0.3415	0.3841	0.4376	0.5280	0.6230	0.6363	0.7066	0.7640	0.8070
t_{393}	0.0576	0.0395	0.0383	0.0348	0.0344	0.0304	0.0469	0.0566	0.1149	0.1397
t_{394}	0.1958	0.1194	0.1265	0.1136	0.1773	0.2305	0.2287	0.2251	0.2484	0.2445
t_{395}	0.0876	0.1415	0.1728	0.2116	0.2702	0.3157	0.3431	0.3737	0.3810	0.3993
t_{396}	0.0328	0.1762	0.2318	0.3166	0.3572	0.4200	0.4360	0.4464	0.4577	0.4906
t_{397}	0.0000	0.0317	0.0445	0.0422	0.1100	0.1148	0.1231	0.1357	0.1231	0.1177
t_{398}	0.0091	0.0223	0.0350	0.0337	0.0390	0.0553	0.0638	0.0585	0.0706	0.0627
t_{399}	0.1161	0.1610	0.1496	0.1725	0.1749	0.2175	0.2161	0.2298	0.2454	0.2925
t_{400}	0.0869	0.1978	0.3133	0.4001	0.4668	0.5558	0.5986	0.6508	0.6920	0.7500

Table 6.13: Measure: nDCG; Stoplist: NOSTOP; Stemmer: PORTERSTEM; Model: BM25.

TF-IDF - normalised Discounted Cumulative Gain (nDCG) values w.r.t. Overall nDCG = 0.2899

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0410	0.1644	0.2785	0.2671	0.2722	0.2971	0.3428	0.3521	0.3669	0.3865
t_{352}	0.0185	0.0220	0.0160	0.0159	0.0212	0.0264	0.0231	0.0225	0.0249	0.0281
t_{353}	0.0712	0.0832	0.0912	0.1387	0.1275	0.1662	0.1896	0.2644	0.2674	0.2984
t_{354}	0.0842	0.0977	0.1147	0.1285	0.1510	0.1824	0.2227	0.2480	0.2488	0.2559
t_{355}	0.1334	0.1579	0.1665	0.1537	0.1445	0.1940	0.2681	0.2764	0.2721	0.2673
t_{356}	0.0000	0.0000	0.0210	0.0420	0.0406	0.0404	0.0397	0.0382	0.0374	0.0366
t_{357}	0.0575	0.1215	0.1349	0.1881	0.2090	0.2630	0.3216	0.3574	0.3550	0.3519
t_{358}	0.1987	0.1668	0.1545	0.1682	0.1800	0.1735	0.1740	0.1255	0.2252	0.2901
t_{359}	0.0137	0.0374	0.0240	0.1759	0.1978	0.1845	0.1833	0.2000	0.1361	0.1619
t_{360}	0.1509	0.2019	0.2660	0.2999	0.3200	0.3602	0.3966	0.5209	0.5503	0.5626
t_{361}	0.0000	0.0000	0.0000	0.0000	0.1175	0.1175	0.1175	0.2720	0.2268	0.2971
t_{362}	0.0490	0.0708	0.0958	0.0636	0.0971	0.1093	0.1179	0.1073	0.1320	0.1561
t_{363}	0.0664	0.1173	0.1075	0.1476	0.2108	0.2272	0.2164	0.2212	0.3062	0.2834
t_{364}	0.1608	0.2704	0.3627	0.3173	0.3631	0.4906	0.5339	0.6303	0.7030	0.7498
t_{365}	0.0989	0.2549	0.3325	0.3741	0.4816	0.4889	0.4732	0.5014	0.5848	0.5750
t_{366}	0.0933	0.1483	0.1924	0.2449	0.2783	0.2766	0.2720	0.2661	0.2908	0.3146
t_{367}	0.0502	0.1294	0.1447	0.1547	0.1709	0.2313	0.2686	0.3011	0.3009	0.3098
t_{368}	0.1222	0.1664	0.2352	0.2810	0.3136	0.3234	0.3135	0.3153	0.4469	0.5081
t_{369}	0.1178	0.0521	0.0999	0.2457	0.2203	0.2003	0.1925	0.1612	0.3436	0.5783
t_{370}	0.0520	0.0703	0.0690	0.0803	0.0940	0.0902	0.0863	0.0762	0.0827	0.0844
t_{371}	0.0247	0.0215	0.0198	0.0359	0.0344	0.0337	0.0339	0.0333	0.0327	0.0324
t_{372}	0.0613	0.0830	0.0939	0.1132	0.1351	0.1330	0.1535	0.1315	0.1819	0.1811
t_{373}	0.0309	0.0898	0.2043	0.3681	0.3822	0.3804	0.4306	0.4173	0.3908	0.3758
t_{374}	0.0969	0.1448	0.1952	0.2084	0.2648	0.3072	0.3324	0.3401	0.3645	0.4050
t_{375}	0.0199	0.1268	0.1723	0.1873	0.1978	0.2177	0.2963	0.3335	0.3530	0.3445
t_{376}	0.0322	0.0236	0.0218	0.0262	0.0312	0.0477	0.0527	0.0560	0.0603	0.0496
t_{377}	0.0917	0.2874	0.2919	0.2888	0.2952	0.3394	0.3665	0.3693	0.4117	0.4459
t_{378}	0.0111	0.0126	0.0239	0.0393	0.0330	0.0382	0.0372	0.0319	0.0315	0.0263
t_{379}	0.0873	0.0753	0.0695	0.1082	0.1256	0.1069	0.1049	0.1020	0.1004	0.2462
t_{380}	0.0585	0.0000	0.0000	0.0000	0.0000	0.0396	0.0387	0.0371	0.1464	0.1956
t_{381}	0.0385	0.0388	0.0376	0.0444	0.0310	0.0455	0.0655	0.1017	0.1002	0.1346
t_{382}	0.2179	0.2948	0.3645	0.3513	0.4232	0.4777	0.4741	0.5160	0.5104	0.5540
t_{383}	0.0589	0.0638	0.0860	0.1164	0.1291	0.1335	0.1235	0.1115	0.1105	0.1105
t_{384}	0.1118	0.1435	0.1741	0.2189	0.3248	0.4615	0.4925	0.5170	0.5768	0.5874
t_{385}	0.0679	0.1103	0.1544	0.1821	0.2218	0.2632	0.2823	0.3459	0.3641	0.3707
t_{386}	0.0000	0.0675	0.0615	0.0769	0.1013	0.0847	0.0827	0.0903	0.1211	0.1317
t_{387}	0.1207	0.1194	0.2293	0.2481	0.2676	0.3158	0.3849	0.4473	0.4521	0.4444
t_{388}	0.0234	0.0335	0.1546	0.1512	0.1782	0.1724	0.1680	0.1655	0.1828	0.1824
t_{389}	0.0233	0.0235	0.0351	0.0375	0.0538	0.0565	0.0579	0.0568	0.0625	0.0695
t_{390}	0.0442	0.0622	0.0604	0.1138	0.1110	0.0932	0.0826	0.0963	0.1352	0.1749
t_{391}	0.0294	0.0350	0.0428	0.0836	0.0919	0.0945	0.0980	0.0971	0.0945	0.1003
t_{392}	0.1692	0.3294	0.3665	0.4154	0.5006	0.6101	0.6237	0.6899	0.7449	0.7455
t_{393}	0.0143	0.0070	0.0066	0.0000	0.0000	0.0000	0.0075	0.0074	0.0750	0.1054
t_{394}	0.0639	0.0412	0.0227	0.0217	0.0571	0.0561	0.0554	0.0545	0.0535	0.0370
t_{395}	0.0800	0.1340	0.1551	0.1882	0.2377	0.2778	0.2971	0.3327	0.3443	0.3467
t_{396}	0.0188	0.1293	0.1818	0.2623	0.2999	0.3601	0.3758	0.3716	0.3803	0.4150
t_{397}	0.0000	0.1477	0.1011	0.1137	0.2019	0.2080	0.2154	0.2005	0.2850	0.3384
t_{398}	0.0088	0.0182	0.0313	0.0300	0.0299	0.0428	0.0529	0.0482	0.0521	0.0515
t_{399}	0.0720	0.1123	0.1115	0.1176	0.1587	0.1719	0.1745	0.1918	0.1909	0.2081
t_{400}	0.0643	0.1516	0.2516	0.3215	0.3660	0.4345	0.4676	0.5276	0.5531	0.5885

Table 6.14: Measure: nDCG; Stoplist: NOSTOP; Stemmer: PORTERSTEM; Model: TF-IDF.

BM25 - normalised Discounted Cumulative Gain (nDCG) values w.r.t. Overall nDCG = 0.3757

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0286	0.2176	0.3113	0.2989	0.3452	0.3919	0.4690	0.4886	0.5175	0.5380
t_{352}	0.0538	0.0638	0.0586	0.0567	0.0819	0.0997	0.1054	0.1037	0.1177	0.1241
t_{353}	0.1483	0.1783	0.1909	0.1779	0.1852	0.2352	0.2685	0.3612	0.3822	0.4518
t_{354}	0.0754	0.0798	0.1005	0.1255	0.1539	0.1917	0.2294	0.2667	0.2773	0.2839
t_{355}	0.2133	0.2738	0.2318	0.2282	0.2221	0.2609	0.3453	0.3698	0.3671	0.3657
t_{356}	0.0000	0.0000	0.0525	0.0895	0.0865	0.0865	0.0853	0.0789	0.0713	0.0663
t_{357}	0.0947	0.1811	0.2082	0.2796	0.3252	0.4167	0.5192	0.5692	0.5692	0.5670
t_{358}	0.1925	0.1811	0.1829	0.2316	0.2479	0.2448	0.2435	0.2331	0.2687	0.3343
t_{359}	0.0184	0.0404	0.0148	0.1119	0.1407	0.1211	0.1171	0.1467	0.1764	0.1987
t_{360}	0.1655	0.2235	0.2928	0.3220	0.3552	0.3848	0.4253	0.5699	0.6100	0.6386
t_{361}	0.0000	0.0000	0.0000	0.0000	0.1483	0.1483	0.1483	0.3567	0.3495	0.4396
t_{362}	0.0530	0.1047	0.1265	0.1223	0.1788	0.2328	0.2702	0.3034	0.3278	0.3128
t_{363}	0.1295	0.2047	0.1996	0.2315	0.2873	0.2974	0.2820	0.2927	0.3592	0.3098
t_{364}	0.1608	0.2854	0.3177	0.3019	0.3546	0.4942	0.5500	0.6460	0.7599	0.8092
t_{365}	0.2032	0.3757	0.4646	0.5580	0.6578	0.6675	0.6588	0.7461	0.8722	0.8885
t_{366}	0.1081	0.1832	0.2748	0.3812	0.4462	0.4702	0.4765	0.4669	0.5335	0.5763
t_{367}	0.0695	0.1453	0.1509	0.1518	0.1640	0.2316	0.2605	0.2957	0.2952	0.3040
t_{368}	0.1143	0.1475	0.2393	0.2937	0.3344	0.3449	0.3423	0.3421	0.4393	0.5837
t_{369}	0.1178	0.0521	0.0988	0.2586	0.2285	0.2042	0.1998	0.1633	0.3463	0.5824
t_{370}	0.0718	0.0906	0.0760	0.0915	0.1084	0.1112	0.1136	0.1246	0.1375	0.1347
t_{371}	0.0371	0.0000	0.0000	0.0368	0.0194	0.0190	0.0605	0.0771	0.0755	0.0564
t_{372}	0.1224	0.1485	0.1946	0.2260	0.2721	0.2788	0.2820	0.2570	0.3238	0.3125
t_{373}	0.0480	0.2113	0.3009	0.5431	0.5362	0.5522	0.5872	0.5836	0.6178	0.6224
t_{374}	0.1174	0.1750	0.2541	0.2738	0.3562	0.4041	0.4398	0.4677	0.4950	0.5442
t_{375}	0.0259	0.1743	0.2582	0.2776	0.3147	0.3502	0.4199	0.4286	0.4619	0.4508
t_{376}	0.0799	0.0724	0.0680	0.1003	0.1226	0.1536	0.1946	0.1876	0.1997	0.2184
t_{377}	0.0917	0.2418	0.2304	0.2245	0.2211	0.2550	0.2774	0.3690	0.4448	0.4300
t_{378}	0.0194	0.0341	0.0512	0.0747	0.0874	0.0879	0.0968	0.0960	0.1014	0.1082
t_{379}	0.1638	0.1638	0.1638	0.2671	0.2671	0.2671	0.2671	0.2671	0.3254	0.4939
t_{380}	0.1374	0.0979	0.0979	0.0867	0.0979	0.1307	0.1186	0.1179	0.2902	0.4443
t_{381}	0.0472	0.0413	0.0536	0.0649	0.0519	0.0777	0.0975	0.1526	0.1622	0.2583
t_{382}	0.2179	0.3771	0.4956	0.4743	0.5685	0.6364	0.6539	0.6868	0.7000	0.7563
t_{383}	0.0358	0.0625	0.0760	0.1143	0.1017	0.1098	0.1073	0.1146	0.1288	0.1361
t_{384}	0.1157	0.1519	0.2014	0.2458	0.3232	0.4746	0.5138	0.5421	0.6292	0.6788
t_{385}	0.1116	0.2087	0.2826	0.3435	0.3916	0.4637	0.4991	0.5632	0.5748	0.5740
t_{386}	0.0000	0.0887	0.0833	0.0819	0.0977	0.0795	0.0790	0.1521	0.1586	0.1835
t_{387}	0.1385	0.1436	0.2759	0.3053	0.3300	0.3858	0.4586	0.5552	0.5678	0.5694
t_{388}	0.0410	0.0873	0.1838	0.1976	0.2614	0.2666	0.2690	0.2600	0.3198	0.3263
t_{389}	0.0188	0.0307	0.0614	0.0533	0.0608	0.0619	0.0596	0.0526	0.0550	0.0676
t_{390}	0.0159	0.0750	0.1008	0.1461	0.1561	0.1530	0.1547	0.1675	0.2062	0.2355
t_{391}	0.0299	0.0355	0.0439	0.0896	0.1139	0.1165	0.1207	0.1261	0.1300	0.1360
t_{392}	0.0778	0.1571	0.1845	0.1954	0.2696	0.3066	0.3047	0.3318	0.3633	0.3690
t_{393}	0.0981	0.0684	0.0667	0.0636	0.0633	0.0613	0.0754	0.0874	0.1435	0.1779
t_{394}	0.2300	0.1434	0.1945	0.1872	0.2363	0.2370	0.2199	0.2177	0.1926	0.1518
t_{395}	0.0875	0.1414	0.1672	0.2115	0.2701	0.3155	0.3430	0.3736	0.3809	0.3992
t_{396}	0.0422	0.1847	0.2375	0.3287	0.3743	0.4308	0.4406	0.4506	0.4613	0.4928
t_{397}	0.0000	0.0000	0.0000	0.0000	0.0209	0.0197	0.0188	0.0357	0.0518	0.1187
t_{398}	0.0099	0.0104	0.0134	0.0093	0.0134	0.0295	0.0283	0.0278	0.0403	0.0399
t_{399}	0.1066	0.1151	0.1356	0.1325	0.1363	0.1620	0.1522	0.1574	0.1666	0.2170
t_{400}	0.0894	0.1941	0.2982	0.3765	0.4409	0.5176	0.5598	0.6251	0.6559	0.7053

Table 6.15: Measure: nDCG; Stoplist: NOSTOP; Stemmer: NOSTEM; Model: BM25.

TF-IDF - normalised Discounted Cumulative Gain (nDCG) values w.r.t. Overall nDCG = 0.2724

Topic \ Bucket	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
t_{351}	0.0413	0.1213	0.2139	0.2081	0.2319	0.2462	0.2713	0.2812	0.3057	0.3169
t_{352}	0.0186	0.0221	0.0160	0.0159	0.0212	0.0265	0.0257	0.0226	0.0251	0.0281
t_{353}	0.0629	0.0783	0.0824	0.1607	0.1432	0.1799	0.2064	0.2790	0.2816	0.3122
t_{354}	0.0649	0.0762	0.0929	0.1118	0.1344	0.1558	0.2034	0.2476	0.2520	0.2614
t_{355}	0.1312	0.1773	0.1729	0.1682	0.1563	0.2110	0.2749	0.2885	0.2764	0.2732
t_{356}	0.0000	0.0000	0.0211	0.0423	0.0414	0.0406	0.0401	0.0389	0.0376	0.0368
t_{357}	0.0747	0.1604	0.1735	0.2259	0.2578	0.3321	0.4122	0.4545	0.4505	0.4551
t_{358}	0.1800	0.1540	0.1503	0.1684	0.1856	0.1769	0.1756	0.1518	0.1803	0.2566
t_{359}	0.0135	0.0364	0.0115	0.0656	0.0764	0.0725	0.0844	0.0985	0.1066	0.1168
t_{360}	0.1509	0.1995	0.2568	0.2636	0.2829	0.3081	0.3601	0.4556	0.4799	0.4853
t_{361}	0.0000	0.0000	0.0000	0.0000	0.1175	0.1175	0.1175	0.2328	0.2052	0.2741
t_{362}	0.0395	0.0596	0.0744	0.0535	0.0949	0.1106	0.1292	0.1370	0.1542	0.1583
t_{363}	0.0751	0.1382	0.1247	0.1740	0.3022	0.3167	0.3125	0.3291	0.4518	0.4129
t_{364}	0.1608	0.2854	0.3730	0.3230	0.3689	0.4977	0.5535	0.6511	0.7213	0.7692
t_{365}	0.0865	0.2552	0.3413	0.3835	0.4808	0.4874	0.4707	0.5011	0.5979	0.5773
t_{366}	0.0792	0.1247	0.1682	0.2146	0.2448	0.2518	0.2495	0.2324	0.2598	0.2976
t_{367}	0.0502	0.1299	0.1450	0.1550	0.1712	0.2315	0.2689	0.3013	0.3011	0.3100
t_{368}	0.1171	0.1491	0.2280	0.2797	0.3029	0.3144	0.3079	0.3071	0.4160	0.5460
t_{369}	0.1178	0.0521	0.0999	0.2457	0.2203	0.2003	0.1925	0.1612	0.3436	0.5783
t_{370}	0.0550	0.0734	0.0764	0.0831	0.1013	0.0954	0.0937	0.0881	0.0956	0.1010
t_{371}	0.0264	0.0225	0.0208	0.0377	0.0360	0.0353	0.0517	0.0508	0.0499	0.0499
t_{372}	0.0633	0.0666	0.0822	0.0936	0.1066	0.1042	0.1196	0.1152	0.1456	0.1558
t_{373}	0.0441	0.1105	0.1331	0.3171	0.3432	0.3352	0.3740	0.3691	0.3796	0.3763
t_{374}	0.1046	0.1609	0.2303	0.2421	0.2902	0.3346	0.3614	0.3701	0.3878	0.4379
t_{375}	0.0199	0.1269	0.1718	0.1868	0.1965	0.2227	0.2936	0.3277	0.3471	0.3386
t_{376}	0.0331	0.0292	0.0224	0.0269	0.0318	0.0530	0.0674	0.0617	0.0613	0.0602
t_{377}	0.0917	0.2048	0.1976	0.1850	0.1800	0.1948	0.2201	0.2637	0.3094	0.3199
t_{378}	0.0111	0.0126	0.0238	0.0388	0.0279	0.0378	0.0319	0.0315	0.0311	0.0211
t_{379}	0.1638	0.1638	0.1638	0.2671	0.2671	0.2671	0.2671	0.2671	0.3254	0.4939
t_{380}	0.0608	0.0498	0.0471	0.0447	0.0434	0.0696	0.0404	0.0388	0.0829	0.1198
t_{381}	0.0410	0.0254	0.0372	0.0482	0.0241	0.0481	0.0504	0.0914	0.0895	0.1265
t_{382}	0.2179	0.2846	0.3432	0.3384	0.4138	0.4740	0.4525	0.4836	0.4814	0.5238
t_{383}	0.0359	0.0595	0.0853	0.0996	0.0940	0.0958	0.0960	0.1084	0.1113	0.1167
t_{384}	0.1106	0.1436	0.1706	0.2148	0.3222	0.4577	0.4882	0.5131	0.5752	0.6037
t_{385}	0.0595	0.1013	0.1269	0.1454	0.1862	0.2306	0.2442	0.2813	0.2699	0.2663
t_{386}	0.0000	0.0335	0.0318	0.0305	0.0460	0.0302	0.0150	0.0619	0.0520	0.0356
t_{387}	0.1231	0.1205	0.2315	0.2596	0.2675	0.3104	0.3861	0.4216	0.4275	0.4259
t_{388}	0.0375	0.0671	0.1897	0.1998	0.2522	0.2592	0.2669	0.2718	0.3151	0.2790
t_{389}	0.0130	0.0233	0.0152	0.0182	0.0272	0.0269	0.0302	0.0268	0.0235	0.0303
t_{390}	0.0152	0.0438	0.0506	0.0738	0.0730	0.0729	0.0685	0.0732	0.0860	0.1264
t_{391}	0.0294	0.0350	0.0428	0.0836	0.0919	0.0945	0.0980	0.0971	0.0945	0.1003
t_{392}	0.0726	0.1535	0.1722	0.1834	0.2629	0.3002	0.2959	0.3231	0.3541	0.3348
t_{393}	0.0296	0.0191	0.0175	0.0090	0.0087	0.0085	0.0157	0.0219	0.0806	0.1127
t_{394}	0.1205	0.0649	0.0768	0.0721	0.0720	0.0698	0.0664	0.0659	0.0468	0.0455
t_{395}	0.0800	0.1339	0.1496	0.1882	0.2376	0.2778	0.2970	0.3326	0.3442	0.3470
t_{396}	0.0194	0.1305	0.1871	0.2640	0.2959	0.3507	0.3577	0.3613	0.3603	0.3939
t_{397}	0.0000	0.0000	0.0000	0.0000	0.0143	0.0139	0.0135	0.0133	0.0348	0.0878
t_{398}	0.0095	0.0093	0.0088	0.0083	0.0118	0.0293	0.0207	0.0206	0.0281	0.0277
t_{399}	0.0439	0.0652	0.0738	0.0867	0.1272	0.1143	0.1068	0.1285	0.1274	0.1369
t_{400}	0.0644	0.1467	0.2520	0.3142	0.3419	0.4147	0.4348	0.4840	0.5284	0.5570

Table 6.16: Measure: nDCG; Stoplist: NOSTOP; Stemmer: NOSTEM; Model: TF-IDF.

6.1 Discussion of the experimental results

From the experimental results reported in this chapter, we can say with no doubt that the BM25 retrieval model is better than TF-IDF, according to the evaluation measures considered. In particular, the previous sixteen tables regards only two evaluation measures: Average Precision (AP) and normalised Discounted Cumulative Gain (nDCG). Despite these are different measures, they provide the same information:

1. Both BM25 and TF-IDF achieve better performances if the Porter Stemmer algorithm is used. Indeed, the best evaluation values obtained are:

   - **BM25**:
     - **MAP**: 0.1631, which is obtained for the couple: (No stoplist, Porter Stemmer).
     - **Overall nDCG**: 0.3951, which is obtained for the couple: (No stoplist, Porter Stemmer).
   - **TF-IDF**:
     - **MAP**: 0.0868, which is obtained for the couple: (Indri stoplist, Porter Stemmer).
     - **Overall nDCG**: 0.2975, which is obtained for the couple: (No stoplist, Porter Stemmer).

2. The worst results for both BM25 and TF-IDF, are obtained for the couple: (No stoplist, No stemmer). Indeed, stoplists help to reduce the noise produced by functional words, such as articles and conjunctions, which usually are considered stop words, since they do not add any information content. When these words are not removed, the index size increases and, at the same time, the IR system performance decreases. In practice the values obtained are:

   - **BM25**:
     - **MAP**: 0.1533.
     - **Overall nDCG**: 0.3757.
   - **TF-IDF**:
     - **MAP**: 0.0791.
     - **Overall nDCG**: 0.2724.

Besides, from the comparison between BM25 and TF-IDF, we can say that BM25 is better than TF-IDF not only in terms of MAP and nDCG but for all the evaluation measures available in AVIATOR. Moreover, from the comparison of all the retrieval models considered (BM25, TF-IDF, Dirichlet Language Model and Boolean) we can claim that BM25 and Dirichlet achieve the best performances for the retrieval process. After these models, between TF-IDF and Boolean the better is TF-IDF.

6.1.1 Topic per topic analysis

The sixteen tables, from 6.1 to 6.16, shows the values of AP and nDCG, obtained for each topic by the IR systems described in table 6.17. From the evaluation data of these tables, the following considerations can be drawn:

1. The topics for which the systems obtain low values of AP or nDCG in the first buckets, i.e. B_1, B_2, B_3, typically will get low values also for B_{10} (which corresponds to the whole indexed collection). This information is useful to understand how the system could evolve as the incremental indexing process advances. In other words, we can make a prediction about the final values of the evaluation measures, for the considered topic. For example, if we consider the topic t_{356}, we can see in table 6.5 that this topic gets low values of AP in the first three buckets $\left(B_1, B_2, B_3\right)$ and then the same occurs for bucket B_{10}. Besides, if the evaluation data regarding the first three buckets present low values for many different topics, the system may not use an effective configuration, so we can stop the AVIATOR job and try a different configuration. However, we have to keep in mind that this consideration is true in general, but some exceptions may occur. Indeed, if we consider the topic t_{361}, from the same table 6.5, it gets zero of AP in the first three buckets, but in bucket B_{10} it gets 0.269 which is a good result. To reduce the prediction error, a possible solution consists of indexing all the buckets from B_1 to B_5 at least. Actually, the value obtained for bucket B_5 is, in general, not so far from the one achieved for bucket B_{10}. This again implies that we can estimate the final value of a measure, without having to index the whole document collection.

2. The topics for which the systems obtain high values of AP or nDCG in the first three buckets typically will get high values also for B_{10}. For example, if we consider the topic t_{365} in table 6.5, this topic gets high values of AP in the first three buckets $\left(B_1, B_2, B_3\right)$ and then the same occurs for bucket B_{10}.

System	Description
S_1	(Indri, Porter stemmer, BM25)
S_2	(Indri, Porter stemmer, TFIDF)
S_3	(Indri, No stemmer, BM25)
S_4	(Indri, No stemmer, TFIDF)
S_5	(No stoplist, Porter stemmer, BM25)
S_6	(No stoplist, Porter stemmer, TFIDF)
S_7	(No stoplist, No stemmer, BM25)
S_8	(No stoplist, No stemmer, TFIDF)

Table 6.17: Description of each system S_j in terms of stoplist, stemmer and retrieval model.

Measure / System	MAP_1	MAP_2	MAP_3	MAP_4	MAP_5	MAP_6	MAP_7	MAP_8	MAP_9	MAP_{10}
S_1	0.0207 (-87%)	0.0394 (-75%)	0.0532 (-66%)	0.0670 (-58%)	0.0810 (-49%)	0.0959 (-39%)	0.1060 (-33%)	0.1212 (-24%)	0.1394 (-12%)	0.1585 (0%)
S_2	0.0124 (-86%)	0.0232 (-73%)	0.0314 (-64%)	0.0376 (-57%)	0.0465 (-46%)	0.0561 (-35%)	0.0595 (-31%)	0.0664 (-24%)	0.0766 (-12%)	0.0868 (0%)
S_3	0.0216 (-86%)	0.0369 (-76%)	0.0511 (-67%)	0.0660 (-58%)	0.0805 (-48%)	0.0938 (-40%)	0.1034 (-33%)	0.1174 (-24%)	0.1345 (-13%)	0.1553 (0%)
S_4	0.0130 (-85%)	0.0209 (-76%)	0.0291 (-66%)	0.0356 (-59%)	0.0452 (-48%)	0.0532 (-38%)	0.0575 (-33%)	0.0642 (-26%)	0.0743 (-14%)	0.0863 (0%)
S_5	0.0215 (-87%)	0.0402 (-75%)	0.0550 (-66%)	0.0702 (-57%)	0.0847 (-48%)	0.0997 (-39%)	0.1093 (-33%)	0.1252 (-23%)	0.1436 (-12%)	0.1631 (0%)
S_6	0.0117 (-86%)	0.0213 (-74%)	0.0296 (-64%)	0.0360 (-56%)	0.0439 (-46%)	0.0530 (-35%)	0.0559 (-31%)	0.0626 (-23%)	0.0721 (-11%)	0.0811 (0%)
S_7	0.0218 (-86%)	0.0372 (-76%)	0.0519 (-66%)	0.0664 (-57%)	0.0807 (-47%)	0.0940 (-39%)	0.1024 (-33%)	0.1167 (-24%)	0.1331 (-13%)	0.1533 (0%)
S_8	0.0124 (-84%)	0.0192 (-76%)	0.0273 (-65%)	0.0335 (-58%)	0.0417 (-47%)	0.0494 (-38%)	0.0531 (-33%)	0.0592 (-25%)	0.0686 (-13%)	0.0791 (0%)

Table 6.18: MAP for each system S_j and bucket B_i.

Measure / System	$nDCG_1$	$nDCG_2$	$nDCG_3$	$nDCG_4$	$nDCG_5$	$nDCG_6$	$nDCG_7$	$nDCG_8$	$nDCG_9$	$nDCG_{10}$
S_1	0.0886 (-77%)	0.1411 (-64%)	0.1742 (-55%)	0.2087 (-47%)	0.2422 (-38%)	0.2736 (-30%)	0.2957 (-24%)	0.3250 (-17%)	0.3599 (-8%)	0.3908 (0%)
S_2	0.0680 (-77%)	0.1094 (-63%)	0.1358 (-54%)	0.1621 (-46%)	0.1890 (-36%)	0.2131 (-28%)	0.2272 (-24%)	0.2458 (-17%)	0.2714 (-9%)	0.2975 (0%)
S_3	0.0870 (-77%)	0.1341 (-65%)	0.1681 (-56%)	0.2028 (-46%)	0.2334 (-38%)	0.2608 (-31%)	0.2818 (-25%)	0.3101 (-18%)	0.3459 (-9%)	0.3781 (0%)
S_4	0.0667 (-76%)	0.1020 (-64%)	0.1268 (-55%)	0.1513 (-46%)	0.1777 (-37%)	0.1992 (-29%)	0.2141 (-24%)	0.2346 (-17%)	0.2571 (-9%)	0.2813 (0%)
S_5	0.0897 (-77%)	0.1417 (-64%)	0.1761 (-55%)	0.2115 (-46%)	0.2450 (-38%)	0.2771 (-30%)	0.2983 (-25%)	0.3285 (-17%)	0.3630 (-8%)	0.3951 (0%)
S_6	0.0664 (-77%)	0.1052 (-64%)	0.1326 (-54%)	0.1591 (-45%)	0.1859 (-36%)	0.2089 (-28%)	0.2236 (-23%)	0.2416 (-17%)	0.2673 (-8%)	0.2899 (0%)
S_7	0.0875 (-77%)	0.1333 (-65%)	0.1674 (-55%)	0.2009 (-47%)	0.2326 (-38%)	0.2608 (-31%)	0.2802 (-25%)	0.3091 (-18%)	0.3440 (-8%)	0.3757 (0%)
S_8	0.0652 (-76%)	0.0980 (-64%)	0.1236 (-55%)	0.1484 (-46%)	0.1726 (-37%)	0.1941 (-29%)	0.2076 (-24%)	0.2267 (-17%)	0.2499 (-8%)	0.2724 (0%)

Table 6.19: Overall nDCG for each system S_j and bucket B_i.

6.1.2 Overall analysis

The two tables 6.18 and 6.19, report the values of MAP and overall nDCG, obtained for the IR systems described in table 6.17, by averaging over all the fifty topics considered. Besides, below each value is reported the $GAP_{\%}$ between the value obtained for the bucket B_i and the one related to bucket B_{10}. From the evaluation data of these tables, the following considerations can be drawn:

1. The $GAP_{\%}$ for the values obtained at bucket B_5, which corresponds to half document collection indexed, is lower than 50% for every system S_i considered. This implies that we can index just half of the document collection to obtain a $GAP_{\%}$ of less than 50% of the final value, related to bucket B_{10}. It is important to notice that this consideration is true for all the systems considered, no matter the stoplist, stemmer and retrieval model used. In this way, IR experts can estimate the final value of a measure, without having to index the whole document collection. The $GAP_{\%}$ between the value v for bucket B_i and the reference value v_r, which is the final value for bucket B_{10}, is computed with the following relation:
$$GAP_{\%} = \left| \frac{v - v_r}{v_r} \right| \times 100$$
According to the relation above, if $v = v_r$ we get $v - v_r = 0 \implies GAP_{\%} = 0\%$. This is the reason why, for bucket B_{10} the tables indicate $GAP_{\%} = 0\%$.

2. The previous consideration is valid both for MAP and overall nDCG. In particular, we have a $GAP_{\%}$ for the values obtained at bucket B_5 that is lower than 50% for the MAP measure while, for the same bucket, we have a $GAP_{\%}$ lower than 40% for the overall nDCG measure. Besides, this observation is true for all the systems considered.

CHAPTER 7

CONCLUSIONS

The information retrieval (IR) field has a long history which dates back to the 1950s when indexing large collections of scientific material was a very important problem to solve. Nowadays, IR is often associated with web search engines since they have become the primary source of knowledge used in everyday life. Despite the significant evolution of this field over the last 20 years, the main purpose of IR has never changed: retrieve the information that satisfies the user information needs. This is what an information retrieval system (IRS) does: it takes the information need of a user, expressed as a query, and returns documents, hopefully relevant, to satisfy the information need inferred by the query. To continuously improve the effectiveness and the efficiency of IR systems in meeting the information needs of users, they have become more and more complex. For this reason, a lot of metrics have been developed, also known as evaluation measures, which make it possible to evaluate and compare the performances of different IR systems. Besides, the evaluation measures, in general, do not have the same scale. For example, if we consider the Average Precision (AP) and the Discounted Cumulative Gain (DCG), the first one is always between 0 and 1 while the second is not. However, what is fundamental for evaluation measures is that they must be comparable for different IR systems. To be comparable, these measures require that each system adopts the same test collection C made of:

1. The document collection corpus D.
2. The set of topics (information needs) T.
3. The set of relevance judgements RJ, which corresponds to the ground truth or pool.

The document collection chosen for the testing phase of the software developed in this thesis is the TIPSTER, a collection of over 528 000 documents, created at NIST. Besides, the set of topics T considered, consists of fifty topics, from 351 to 400, chosen from TREC7. This is the same evaluation campaign from which comes the pool file. The purpose of the evaluation process, as the name suggests, consists of evaluating IR systems, to improve them and to have a deeper understanding of their functioning, limits and strong points. In particular, the evaluation process evaluates the effectiveness of an IR system, measuring how well an IR system retrieves the documents that satisfy the user information need. As we said, to compare different IR systems or retrieval models, we need to adopt the same test collection. This one could contain millions of documents, and when this occurs, it is called Very Large Collection (VLC). Indexing all the documents of a VLC requires a lot of time and resources, but this is necessary if we want to use it for evaluating one or more IR systems. Besides, IR experiments often require to reindex the same collection many times, e.g. for testing different stemming algorithms or stoplists. This implies a lot of effort and time for IR experts since they have to wait for the end of the indexing process, which coincides with the whole indexed collection. For this reason, the purpose of this thesis was the development of a visual analytics tool to make the evaluation process faster and more intuitive. The software developed for solving this problem is called AVIATOR. AVIATOR allows IR experts to evaluate one or more IR systems, in a time efficient way, through a progressive visual analytics web interface. To save time AVIATOR does not index the entire collection at once, instead, it uses the incremental indexing strategy, as described in Chapter 3. In this way AVIATOR, during the preprocessing phase, divides the document collection C into n buckets B_i of the bucket collection $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ and then at the i-th iteration of the loop: "Incremental Indexing", "Retrieval", "Evaluation", the documents of the bucket B_i are added to the index. Doing so, at each iteration, we evaluate IR systems not on the whole collection, but just on the latest version of the index available. As the incremental indexing process advances more and more documents are indexed and the values, for the evaluation measures, are closer to those calculated on the full index. This is very useful for IR experts since they know how the experiments are going, while the indexing process incrementally proceeds. In this way, if the intermediate results of an experiment do not meet the expectations, it can be stopped in advance, without having to index the entire collection. Using the progressive visual analytics interface provided by AVIATOR, IR experts can control each stage of the entire process with interactive charts and many other visual components. In particular, AVIATOR allows the user to visually compare the performance of the different retrieval models, for each combination of stoplist and stemmer, as the indexing process

advances automatically. In this way, it is possible to determine which system is the best, at each index percentage, according to the chosen evaluation measure. AVIATOR is a full stack application and it consists of a back-end and a front-end. The back-end is described in Chapter 4, where we show the client-server architecture adopted by AVIATOR and the technologies used to develop it. In particular, the back-end consists of an HTTP web server, written in Java, that acts as a wrapper for internal functionalities, e.g. the preprocessing function, and for the Solr search platform, which is the IR system adopted. The AVIATOR web server receives the user requests from the web interface at specific URLs, according to the URI-scheme defined for the REST APIs. Every time a valid request reaches the web server, it invokes the routine dedicated for the specific service requested. AVIATOR can be configured through a dedicated configuration tool, described in Chapter 4, from which a user can customise all the settings. The front-end, instead, is described in Chapter 5, where we show the Model-View-Controller architecture adopted by AVIATOR to separate the data access layer from the presentation one. In particular, the front-end consists of a web visual analytics interface, from which IR experts can control each stage of the AVIATOR process. Moreover, while the incremental indexing process advances in the background, the user can check in real-time the latest evaluation data available through interactive charts, i.e. scatter plots for the topic per topic analysis and bar charts for the overall one. The AVIATOR testing phase has been performed on the disks 4 and 5 of the TIPSTER document collection. In particular, we examined all the possible combinations generated by four different stoplists (including no stoplist at all), four stemming algorithms (including no stemming) and four retrieval models. This corresponds to a total of: $4 \times 4 \times 4 = 64$ possible combinations. Besides, we evaluated each combination at ten different indexing percentages, so the number of Solr core generated is: $64 \times 10 = 640$. In particular, these 640 Solr cores correspond to 230 GB of indexing data. From the analysis of the values for the considered evaluation measures, we can say that the two retrieval models that obtain the best results are: BM25 and Dirichlet Language Model. After these two models, the better between TFIDF and Boolean is TFIDF.

Future work

Using the AVIATOR version implemented for this thesis, IR experts can save a lot of time during the evaluation activities. However, evaluating an IR system is a complex task that involves many different measures and components. For this reason, despite the current version of AVIATOR provides already a lot of useful functionalities, many others can be added.

Hence, a lot of improvements can be included in the future versions of AVIATOR, the major are:

- The possibility of customising some retrieval model parameters such as k_1 for the BM25 retrieval model. The final result of this improvement can be visualised in the example of Figure 3.7 reported in Chapter 3. The figure shows a slider for the BM25 model, that can be used to select the value for k_1 and update the scatter plot with the new values for the evaluation measure considered, during the topic per topic analysis.
- The possibility of choosing the values for the probabilities p and q, described in Chapter 3. These probabilities regulate how the sampling is done during the preprocessing phase. By default, we have $p = q = \frac{1}{2}$, which gives balanced buckets with both relevant and not relevant documents. Giving the user the possibility of changing these values means that AVIATOR can be tuned according to the given document collection.
- The introduction of statistical and numerical analysis as a support for the visual analytics one. The statistical analysis is useful to estimate the gap between the values obtained for the evaluation measures at each percentage of the incremental indexing process, and the ones related to the whole document collection indexed.

RINGRAZIAMENTI

Ringrazio di cuore la mia famiglia, per avermi dato l'opportunità di intraprendere questo percorso di studi e per avermi sempre sostenuto e spronato ad arrivare sino alla fine.

Ringrazio il mio relatore, Prof. Gianmaria Silvello, per la disponibilità e la pazienza con cui mi ha sempre seguito durante tutto il lavoro di tesi.

Ringrazio i miei compagni dell'università per tutti i momenti di gioia e di difficoltà passati assieme. Grazie per avermi accompagnato in questa lunga avventura, da cui ho imparato che nessun ostacolo è insormontabile quando si è uniti e insieme.

Ringrazio tutti i miei amici per avermi regalato il loro tempo, la loro fiducia e l'entusiasmo con cui andare avanti.

Sarò sempre grato ai miei nonni, per i loro saggi consigli e per avermi insegnato i valori per i quali vale la pena lottare.

> "Remember to look up at the stars and not down at your feet. Try to make sense of what you see and wonder about what makes the universe exist. Be curious. And however difficult life may seem, there is always something you can do and succeed at.
> It matters that you don't just give up."
>
> Stephen Hawking

BIBLIOGRAPHY

[1] D. Hawking and S. Robertson, "On collection size and retrieval effectiveness," *Information Retrieval*, vol. 6, no. 1, pp. 99–150, 2003.

[2] M. Sanderson and W. Bruce Croft, "The history of information retrieval research," *Proceedings of The IEEE - PIEEE*, vol. 100, pp. 1444–1451, 05 2012.

[3] W. Croft, D. Metzler, and T. Strohman, *Search Engines: Information Retrieval in Practice*. Alternative Etext Formats, Addison-Wesley, 2010.

[4] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, pp. 613–620, Nov. 1975.

[5] S. Robertson, "The probability ranking principle in IR," *Journal of Documentation*, vol. 33, no. 4, pp. 294–304, 1977.

[6] S. E. Robertson and K. S. Jones, "Relevance weighting of search terms," *Journal of the American Society for Information Science*, vol. 27, no. 3, pp. 129–146, 1976.

[7] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[8] K. S. Jones, S. Walker, and S. Robertson, "A probabilistic model of information retrieval: development and comparative experiments: Part 1," *Information Processing & Management*, vol. 36, no. 6, pp. 779 – 808, 2000.

[9] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Found. Trends Inf. Retr.*, vol. 3, pp. 333–389, Apr. 2009.

[10] G. Amati and C. J. Van Rijsbergen, "Probabilistic models of information retrieval based on measuring the divergence from randomness," *ACM Trans. Inf. Syst.*, vol. 20, pp. 357–389, Oct. 2002.

[11] D. Harman, *Information Retrieval Evaluation*.
Morgan & Claypool Publishers, 1st ed., 2011.

[12] M. Angelini, N. Ferro, G. Santucci, and G. Silvello, "VIRTUE: A visual tool for information retrieval performance evaluation and failure analysis," *Journal of Visual Languages & Computing*, vol. 25, pp. 394–413, aug 2014.

[13] C. D. Stolper, A. Perer, and D. Gotz, "Progressive visual analytics: User-driven visual exploration of in-progress analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, pp. 1653–1662, Dec 2014.

[14] S. Frey, F. Sadlo, K. Ma, and T. Ertl, "Interactive progressive visualization with space-time error control," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, pp. 2397–2406, Dec 2014.

[15] J.-D. Fekete and R. Primet, "Progressive analytics: A computation paradigm for exploratory data analysis," 2016.

[16] D. Fisher, I. Popov, S. Drucker, and m. schraefel, "Trust me, i'm partially right: Incremental visualization lets analysts explore large datasets faster," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, (New York, NY, USA), pp. 1673–1682, ACM, 2012.

[17] D. Song and E. Golin, "Fine-grain visualization algorithms in dataflow environments," in *Proceedings of the 4th Conference on Visualization '93*, VIS '93, (Washington, DC, USA), pp. 126–133, IEEE Computer Society, 1993.

[18] M. Angelini, G. Santucci, H. Schumann, and H.-J. Schulz, "A review and characterization of progressive visual analytics," *Informatics*, vol. 5, no. 3, 2018.