Università degli Studi di Padova

Department of Information Engineering

Master Thesis in ICT for Internet and Multimedia

Predicting performance of combinations of IR System Components

Supervisor
Ferro Nicola
Università di Padova

Master Candidate
Padoan Alberto

Academic Year 2019-2020
Padova, April 20, 2020

Abstract

Information Retrieval systems are becoming a fundamental part in our lives. A IR system is essentially based on three components: a stop list, a stemmer and a IR model. Each of them has an important role during the Information Retrieval process, performing the task for which it is designed. During the years IR research produced different realisations of IR components, each of them with its advantages and deficiencies, with a precise scope or to solve a particular problem. This means that different components can be combined together, thus creating a unique realisation of a IR system.

Information Retrieval is also interested into improving performances and accuracy, and an important role in this research is taken by Machine Learning. So, from this union, starts the purpose of this thesis that is predicting performance of IR system components. The experiments are executed with different criteria and paying particular attention to the features involved, especially to those strictly related to the configuration of the IR system, and their influence on systems' ranking and performance.

Contents

Listing of figures

Listing of tables

1 Introduction

Open *google.com*, digit some words, wait less than one second, and obtain the result.
This is one of the most common operation that nowadays everyone perform - almost - everyday. The same concept can be applied when one searches a file in a personal device, a song on a streaming app, a book at the library or even a contact in its phone. Such actions are becoming automated mechanisms in our lives, to the point that we do not pay attention to what's going on in depth and we even get angry if the result of a complex and articulated query is not what we expected. Actually, when we search something in a search engine we ignore the complex mechanism that works behind what we see. First studies of this science started around 1920_s laying the groundwork for the discipline that, through a rapid evolution in the following decades, today we know as Information Retrieval (IR). IR cares of structure, store, search and retrieve information, and one commonly known realisation of such principles is, for example, the search engine.
There a lot of different realisations of IR systems, built on different concepts and for retrieving various types of information such as images, audio, videos etc. We are rather interested on text retrieval and on how a text IR system works. As we are going to deeply analyse in this thesis, a IR system is essentially formed by three components: a stop list, a stemmer and a IR model. These components permit to efficiently store the information both in terms of space and time and

to quickly satisfy user's needs when inserts a query. Over the years many studies have given rise to different types of stop list, stemmer and IR model, each of them with its advantages and deficiencies, with a precise scope or to solve a particular problem; however this search has allowed to progress fast in the IR field and to develop better technologies. Since there are a lot of realisations of IR system components, consequently there are different realisations IR systems. In fact, combining different types of IR components, it is possible to obtain a new configuration of a IR system. In this thesis we are going to present different types of IR components: 7 stop lists, 5 stemmers and 10 IR models. Thanks to a Java library, it is possible to build a **GoP**[1], i.e. a systematic series of experiments representing different combinations of retrieval methods. In this thesis we deal with 350 different IR models, obtained by all the possible combinations of our components.

Information Retrieval is constantly working for obtaining better and better performances, for satisfying user's needs faster and with accuracy. These needs led to the union between IR and Machine Learning: ML studies large amount of data, finding patterns that are impossible to detect for a human mind. For this reason the principles of ML can be applied to IR for conducting deeper studies and for improving its performances. In particular this thesis is going to study the prediction of performances of combinations of IR system components. Performances are obtained by the most common IR evaluation measures, as we going to see in chapter 2.

The experiments simulate two different scenarios. The first one assumes that the predictor knows some combinations of IR systems components and has to predict performances of the remaining systems, considering that in the training phase the predictor has seen, for at least in one system, each IR component. The second scenario assumes that the predictor has to predict performances of IR systems with a new component, that does not appear in its previous knowledge. The results of the predictor are analysed both in terms of performances related to the query of each system and in terms of ranking of the predicted systems. The core of this study is to comprehend how the predictor works considering the influence of a different set of features, each of them describing various aspects of our data, and paying particular attention to features relative to the IR system. Chapter 2

[1] http://gridofpoints.dei.unipd.it

introduces few notions about IR, the structure of a search engine and the evaluation measures. Then it proceeds with a description of the IR components involved in our experiments and finally it introduces few notions about Machine Learning and the predictors involved in out tests. Chapter 3 explains the methodology behind this work, presenting the two tasks of the thesis, the features involved, the predictors and their comparison measures. Chapter 4 describes the dataset, how the target measures are computed and the creation of our feature set. Chapter 5 presents the two main experiments of this thesis, describing the results obtained by each predictor and comparing their performances. Finally chapter 6 resumes the notions learned on this thesis, and provides some guidance for future works.

2 Background

This chapter introduces a few notions useful to fully comprehend this work since it is based on some fundamentals principles. For this reason, the first section will treat information retrieval. This thesis is based on search engines so first it is important to understand how a search engine works both in indexing and retrieval processes, paying particular attention to performance evaluation. The second section takes a review of the particular realization of IR systems adopted in this work, examining their components. Finally, in the third section, we explain the techniques used for the prediction task in our thesis, taking a brief overview of machine learning principles and the principal algorithms used.

2.1 Information Retrieval

Gerald Salton, pioneer of Information Retrieval (IR) research, proposed the following definition (Salton, 1968):

> Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information.

The meaning of 'information' is quite general. There is a wide range of types of information that can be applied to this definition. Usually, IR deals with

collections of unstructured or semi-structured data (e.g., web pages, documents, images, video, etc.). This is a relevant difference from databases, in which data is structured. Anyway, the final objective of IR is to develop methods, algorithms and systems which permit to a user to retrieve any information needed [2].

It is possible to find the first concepts of Information Retrieval at the beginning of the XX century. In the 1920_s and 1930_s, Emmanuel Goldberg was the first person to invent a mechanical device capable to search a catalog for a particular entry [13]. A significant contribution was given by Vannevar Bush in the 1930_s that invented a system named Memex. This system was never effectively created but was able to read microfilms with more functionality than any microfilm reader ever built [14]. So, starting from these mechanical prototypes, Information Retrieval found its fundamental application in computer systems becoming a crucial research field. Nowadays we associate IR to web search since this field had a huge growth thanks to the spread of the Internet giving rise to major companies like Google. Search engines are capable to retrieve an enormous amount of information from a bunch of terms (the query) inserted by a user. For this reason, such systems are one of the most powerful examples of the application of IR and gave a significant contribution to make this science growing. But *web search* is one of the possible scenario of IR. *Vertical search* is a web search performed on a domain regarding a specific topic. *Enterprise search* is a search made on files stored on an intranet. In this case, IR deals not only with web pages but with documents, email, reports and other types of information too. *Desktop search* is the private and local version of enterprise search. It deals with local files, email, messages, etc. Another example is *peer-to-peer search* which involves a peer-to-peer network and the information stored in the nodes of that network. Anyway, IR is applied also in other fields such as intelligence analysis, scientific research, health, etc. In general, any application which uses unstructured data takes advantage of IR principles for organizing and search that data [1].

Retrieve information given a user's query (*ad-hoc search*) is not the only IR task. There is *filtering*, which is a task that aims to provide correct information based on the user's interests. *Classification* task, instead, assigns class or labels to a set of documents. Finally, *question answering* tries to provide a pertinent and correct answer to a user's request such as "Who won 2014 Tour the France?".

As mentioned before there are a lot of types of documents that can be retrieved.

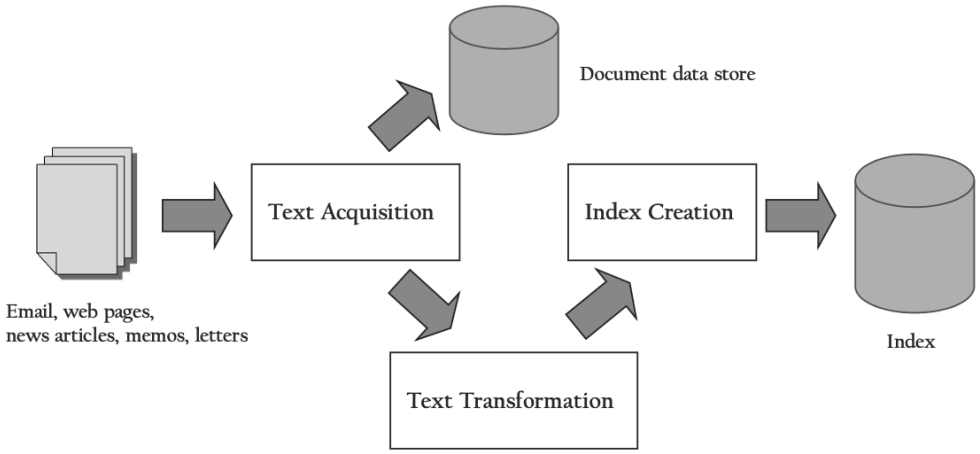

Figure 2.1: The indexing process [1].

This was done to give a brief overview of IR capabilities, but in this work and the following explanations, we are going to refer only to text retrieval.

2.1.1 Structure of a Search Engine

First of all one has to consider that IR systems' performance are based on two main concepts: *effectiveness* and *efficiency*. Efficiency (speed) is related to the speed of the task: a query needs to be processed as fast as possible. Efficiency is also related to a wide usage of resources, i.e. an efficient use of memory to store information needed. Effectiveness (quality) is referred to the will to retrieve the most relevant set of documents possible for a query and at the same time discard as most no-relevant documents as possible. As it will be presented, there are more specific measures to evaluate systems' performance, but all of them can be referred to those two concepts. A first classification of the components of a search engine could be done by dividing it into indexing and searching. Actually, before the search and the retrieval of the results, the data needs to be represented with their information content (figure 2.1). This procedure is known as indexing and permits to make the searching task working. The query process permits to retrieve documents related to a user's query. As it is shown in Figure 2.2 the whole process starts with user interaction (i.e. the query) and finishes with the

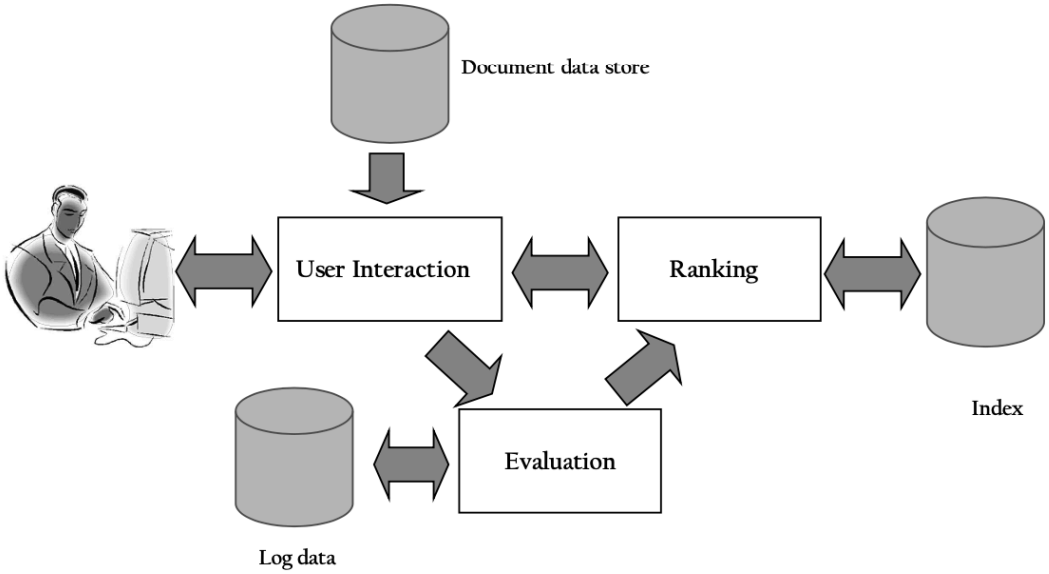

Figure 2.2: The query process [1].

results providing the ranked list of documents. This task needs to be efficient (this measure depends on the indexes) and effective depending on how well the raking list of documents is provided by the retrieval model. The evaluation permits to improve effectiveness by analyzing logs of previous interactions.

Indexing Process

As mentioned above, indexing permits to represent the informative content of a document and it is used to retrieve the document when a query is performed. The indexing process must be efficient both in terms of time and space and needs to be built in such a way that if a new document is added, the indexes can be updated easily. The best index for the information content of a document is the document itself but this practice does not respect time and space requirements. So, the text of a document needs to be processed to obtain an informative content as similar as possible to the original document, but considering the requirements of the indexing task. Doing this does not implies that the system has to comprehend the text; this operation would be too expensive and complex. For this reason a search engine captures statistical information regarding the text since it is a

faster and an efficient operation. There are several operations that can be done for indexing a document; none of them is mandatory and it is possible to use different combinations depending on the requirements and the corpus used.

First of all, if we are processing an European language, *tokenizing* is needed, since this practice is not used for Asiatic languages. It consists into obtaining a stream of words separated by a space starting from any type of document. So the tokenizer has to remove all the information that is not needed to index the document and to resolve some issues in order to improve effectiveness. For example, all the capital letters may be changed into lowercase, images and tables are substituted by their captions, abbreviations are expanded to their real meaning (this implies the usage of a dictionary), apostrophes and other punctuation are removed, etc. This permits us to have a lighter document without compromising its informative content. If the document parsed is written with a markup language, tags and other useful metadata are used by the parser in order to extract structure's information too.

Once we obtain a stream of words, it is necessary to remove the words that do not have a significant meaning and with low information. These words are known as *stop words* and usually are functional words extremely common and do not provide any additional characterization of the document. Stop words may help the reader to understand the document but they are not relevant if we consider the term statistics. Removing these words reduces index size and improves retrieval efficiency (and, usually, effectiveness too). Anyway, this process is not trivial; removing too many words may impoverish the information carried by the index. It is important how the stop words are chosen. The list of stop words considered usually includes the top-k (usually k = 50) most frequent words but it can vary from corpus to corpus and from language to language.

The following optimization step is known as *stemming*. The stemmer is a component that has the role of grouping words characterized by the same root and that are the variation of the same concept. Consider, for example, the words "fishing", "fisher" and "fish"; they represent similar concepts and they all related to the fish so, a stemmer would group these words into the shorter and basic stem, which is "fish". It is important to pay attention to the so-named *false positives*, which happens when two words are grouped with the same stem but they actually have a completely different meaning (for example when "is" is detected as the plu-

ral of "I"). Another issue happens when two words, which have actually the same stem, are not grouped together. This usually happens with plurals, for example, "century" and "centuries". These are known as *false negatives*. A stemmer can be of three types: *algorithmic*, *dictionary-based* and *statistical*. The first one searches a relation between words by means of algorithms and knowledge of words' suffixes. Instead, a dictionary-based stemmer relies on a dictionary with words and the related stem. It is more flexible in case of exceptions and irregularities but it needs to be updated frequently and it has to be built by linguistic experts. However, the number of false positives is expected to be quite low but there are some limitations since the dictionary can not be too long. For this reason, it is possible to find stemmers that combine both algorithm and dictionary-based approaches; in this way, it is possible to benefit from the advantages of the first two strategies. The first one can be used for unknown and particular words, the other one for common words. Finally statistical stemmers are built on a statistical analysis of text corpus, and are particularly useful for query expansion. It is important to notice that a statistical stemmer can be designed only if the text corpus is available. Summing up, stemming contributes to decreasing the dimension of the index and permits to perform *query expansion* i.e. the possibility for a user to obtain results similar to the concept expressed.
Finally, there is index creation. It tries to represent the document in order to permit a correct retrieval of the information. First of all it is important to create term weights. Weights are used for scoring and ranking and depends by the *IR model* used. A quite common practice is to compute the frequency of the term inside a document (*term frequency* or *tf*) and the term frequency among the collection of documents (*inverse document frequency* or *idf*). Then, the core of the whole indexing process is *inverted indexes*. It transforms the couples (document, term) into the component (term, document). This practice is done to enhance query performance but also to permit a faster insertion of new documents inside the index collection. Figure 2.3 resumes step by step the whole indexing process just described.

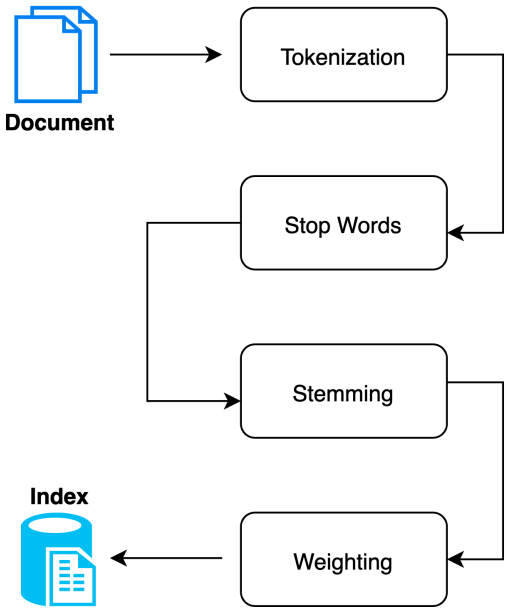

Figure 2.3: Detailed indexing process.

Query Process

The whole process begins with the insertion of the query by a user. Usually, a query may consist of a bunch of keywords related to the interested topic, however it may happen to deal with a more complex query. Normally, a query with unnecessary terms does not improve the final result and it could complicate the retrieval process. For this reason, search engines process the query in the same way they have processed the documents of their indexes. So the query is tokenized, it passes through a stop list and finally through a stemmer. This procedure is done also to make the query terms comparable to documents. Moreover, a search engine may correct spelling mistakes and perform query expansion which adds additional terms to the query in order to improve effectiveness. Both these operations are performed thanks to query logs, the analysis of documents, etc. The scoring process, known also as *ranking algorithm*, permits to obtain scores that are used to return the ranked list of documents. Scores are calculated by the IR model involved and do not depend by the indexing process. *IR models* are a fundamental component of any IR system. As mentioned before, they permit to represent the informative content of both documents and query and they are the core of the retrieval of the results by comparing documents' index and query.

There are a lot of versions of ranking algorithms, but generally a document score can be represented by the following formula:

$$\sum_i q_i d_i$$

Where q_i expresses the query term weight for the i-th term and d_i is the document term weight.

2.1.2 Cranfield

In IR it is fundamental to evaluate systems' performances. IR systems evaluation is crucial since it permits to understand what are the performance and how they can be improved. In 1960_s Cyril Cleverdon and his colleagues at the College of Aeronautics in Cranfield, near London, created the one that nowadays is known as *Cranfield pradigm* [15]. In that period there was more and more the need to define guidelines for indexing and retrieve documents in an efficient way. As explained in [16], the first project began in 1958 and published in 1962, treating four indexing schema. This project, known also as *Cranfield I*, provoked much debate since the indexing did not perform as expected. This was probably due to the low knowledge of the field but led to a *failure analysis* that permitted to identify the problems and the future improvements. In fact *Cranfield II* was a huge step forward; it introduces the concept of *collection* and inheriting some concepts from Cranfield I, define our notion of the methodology of IR experimentation [16]. Cranfield paradigm represents a model for evaluation of IR systems and in this paradigm relies information retrieval evaluation campaigns such as the *Text Retrieval Conference*[1] (TREC) and the *Conference and Labs of Evaluation Forum*[2] (CLEF). TREC was founded in 1992 by the *U.S. National Institute of Standards and Technology* (NIST) with the goal to create a large test collection for improving IR research [15]. CLEF instead, started in 2000 and was born to to promote research, innovation, and development of information retrieval systems [2]. In the first decade CLEF focused on the evaluation of systems developed to run on multiple languages but from 2010 it tried to cover more aspects of IR following its fast evolution over the years [2].

[1] https://trec.nist.gov/
[2] https://www.clef-initiative.eu/

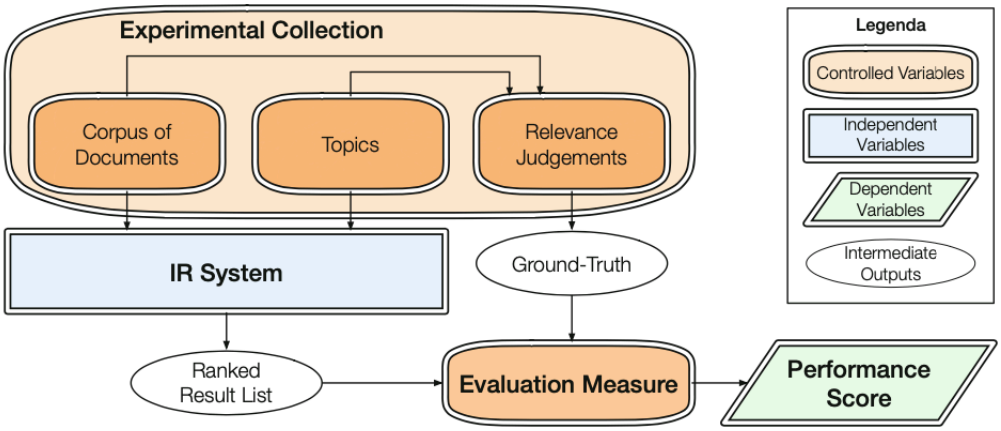

Figure 2.4: Cranfield paradigm [2].

Cranfield paradigm (Figure 2.4) is based on topic collections $\zeta = (D, T, RJ)$ where D represents the corpus of documents, T represents the user's information needs and RJ stands for the relevance judgments or ground-truth, that is, for each topic, the relevant set of documents [2]. The corpus represents the space where the IR system is going to run. Topics represents the user's information needs and are composed by four fields:

- **id**: is an identification number related to the topic;
- **title**: are just few words about the topic;
- **description**: which describes the topic's information;
- **narrative**: it is a narrative which hypothetically brings the user to that topic.

Then note that relevance judgments usually are labeled as *relevant* and *non-relevant* but it is possible to find also *relevant*, *partially relevant* and *non-relevant* or other more exhaustive scales. The relevance judgment can be expressed as a function [17]:

$$RJ : T \times D \rightarrow REL$$

$$(t, d) \rightarrow rel$$

Since examining all the documents for each topic is a long process, there exists a better and more efficient approach named *pooling*. But before it is necessary to

introduce the concept of *run* as expressed in [17].

Run: *Given a natural number* $N \in \mathbb{N}^+$ *called the length of the run, a run is a function*

$$R : T \to D^N$$

$$t \to r_t = (d_1, d_2, ..., d_N)$$

such that $\forall t \in T, \forall j, k \in [1, N] \mid j \neq k \Rightarrow r_t[j] \neq r_t[k]$ *where* $r_t[j]$ *denotes the* $j-th$ *element of the vector* r_t*, vectors start with index* 1*, and vectors end with index* N*.*

So a pool is obtained running several runs for a topic. Each run comes from a different search engine with a particular configuration. In each run only the top-k retrieved documents are considered: if a document belongs to that group, it is considered relevant, otherwise it is considered not relevant. In other words, all the top-k lists of the runs are merged obtaining the list of relevant documents. Usually in TREC the number top-k elements varies between 50 and 200 [1].

The goal of this setup is to have a repeatable and solid way for IR systems evaluation. This setup is based on experimental collection and evaluation measures that are also known as *controlled variables* since they are kept fixed during experimentation. Then we have *independent variables* that are IR systems and finally we have *dependent variables* which are performance scores and they change with respect to independent variables [18].

2.1.3 Evaluation Measures

In this paragraph follows the description of some of the most used evaluation measures in IR, paying attention to measures used in this thesis. In particular, in this work they are calculated using Matters[3], a MATLAB® Toolkit for Evaluation of information Retrieval Systems. It is complaint with the TREC-format and thus runs and pools from principal evaluation campaigns as TREC and CLEF.

Recall (figure 2.6) is a widely used measure introduced by Cranfield studies.

[3] https://matters.dei.unipd.it/

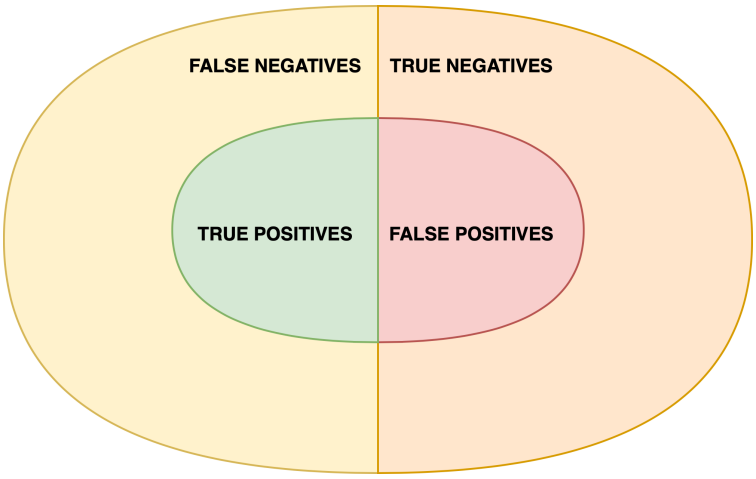

Figure 2.5: Set of documents.

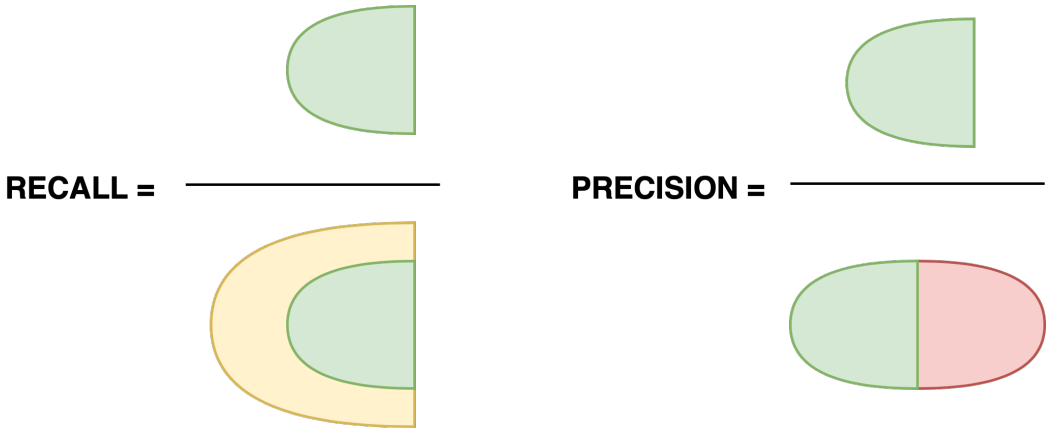

Figure 2.6: Visual representation of recall and precision .

It represents the portion of relevant items that are selected.

$$Recall = \frac{|RelevantDocs \cap RetrievedDocs|}{|RelevantDocs|}$$

Precision (figure 2.6) was introduced as recall by Cranfield studies. It represents how many of the retrieved items are relevant.

$$Precision = \frac{|RelevantDocs \cap RetrievedDocs|}{|RetrievedDocs|}$$

Figure 2.5 represents the set of documents. Note that the relevant set of doc-

uments is composed by *false negatives* (and so *true positives*). The retrieved set of documents is composed by the *true positives* set and the *false positives* set.

Average Precison (AP) is a measure that combines recall and precision for ranked retrieval results [19]. It is computed as follows:

$$AP = \frac{\sum_r P@r}{R}$$

Where r is the rank of each relevant document, $P@r$ is the precision of the top-r retrieved documents and R is the total number of relevant documents. AP is sensitive to the ranking of relevant documents and emphasizes if a relevant document is returned earlier. A relevant document that is ranked lower gives a minor contribution to this measure [19].

Mean Average Precison (MAP) is the average of AP of a IR system over a set of n query topics. It is computed as follows:

$$MAP = \frac{1}{n} \sum_n AP_n$$

Where AP is the average precision of a IR system over set of n query topics. MAP is one of the most frequently used measures over the years for TREC evaluations [20].

Precision at n (P@n) is the proportion of documents that are relevant among top-n retrieved documents.

$$P@n = \frac{r}{n}$$

Where r is the portion of relevant documents retrieved at rank n. The value of n could be chosen with respect to the expected number of documents that appear in a web search. Usually is set at 10 but in some experiments it may be convenient to assume that a user would check a lower number of results (5, for example) [21]. In this work, we consider $P@10$.

Normalized Discounted Cumulated Gain (nDCG) is a IR measures that

assumes that highly relevant documents are more valuable than less relevant documents. Moreover it assumes that if a relevant document has a bad rank, it is less valuable since the user is less likely to examine that [22]. The nDCG is based on the normalization of DCG, since the general formula of nDCG at rank position p:

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

Where $IDCG$ is the *ideal discounted cumulative gain*. More in detail, DCG_p is computed as:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_p(i+1)}$$

Note that rel_i is a binary value $rel_i \in (0, 1)$ and it is equal to 1 if the $i-th$ document is relevant. The normalization factor $IDCG_p$ expressed the maximum possible DCG at position p considering the retrieved documents as the pool ordered from the most relevant to the less relevant. The following is one possible realization of $IDCG_p$ designed by Microsoft:

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{log_2(i+1)}$$

Where REL_i represents the list of relevant documents (ordered by their relevance) in the corpus up to position p. Finally, note that averaging $nDCG$ among the topics it is not statistically reliable since the number of relevant documents may vary topic by topic [22].

Expected Reciprocal Rank (ERR) is a measure related to an assumption of user's behavior once he get the results. It is computed as follows:

$$ERR = \sum_{r=1}^{n} \frac{1}{r} P[\text{user stops at position r}]$$

Where n is the number of documents in the ranking and it is considered the probability that a user stops at position r. It means that a user, examining the ranked list of documents, stops the examination with a document at position r and no further documents are considered [23].

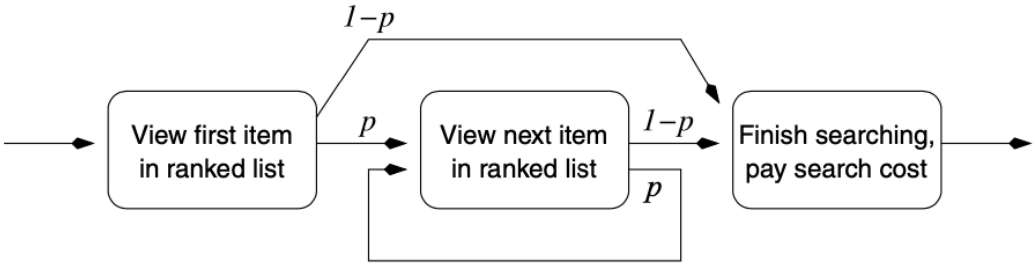

Figure 2.7: User's persistence model [3].

Reciprocal Rank (RR) is a measure which assumes that there is a set of relevant documents, and each of them is equally relevant. In some cases a user needs a relevant document, and it must be ranked as higher as possible in order to easily find it. So reciprocal rank is expressed as follows:

$$RR = \frac{1}{min\{k|Res[k] \in Rel\}}$$

Consider that if $|Rel| > 1$ the user task is completed as soon as he encounter a relevant document; all other documents are not considered. Instead, if there is only one relevant document ($|Rel| = 1$), the reciprocal rank is equal to the average precision [24].

Rank-Biased Precision (RBP) is another useful IR evaluation parameter. This measure considers a probability p, known also as *persistence*, of an user to view the next item in ranked list following the schema in figure 2.7. So, considering n ranked documents, the rank-biased precision is computed as:

$$RBP = (1-p)\sum_{i=1}^{n} rel_i p^{i-1}$$

Where $rel_i \in (0, 1)$ is the relevance degree of the $i-th$ document and it is equal to 1 if it is relevant. Choosing the proper value of p is not trivial. Usually, small values for p emphasizes top-ranked documents, instead higher values for p reduces importance of high ranked documents and fits better for more persistent users [3].

Kendall's τ is a coefficient that expresses the degree of concordance between two columns of ranked data. If it is equal to 1, there is perfect relationship between two lists. Instead, if data is equal to 0, there is no relationship. Note that the lowest possible value for this coefficient is -1, and it represents perfect inversion. Kendall's τ is computed as:

$$\text{Kendall's } \tau = \frac{C - D}{C + D}$$

Where C is the number of concordant pairs and D is the number of discordant pairs. A concordant pair is the number of observed ranks below a particular rank which are larger than that particular rank. On the other hand, a discordant pair is the number of observed ranks below a particular rank which are smaller in value than that particular rank.

Pearson Correlation is a correlation coefficient between -1 and 1 that indicates a linear relation between two statistical variables. Given two statistical variables X and Y, Pearson correlation is computed as the covariance divided by the product of the standard deviation of the two variables:

$$p_{XY} = \frac{covXY}{\sigma_X \sigma_Y}$$

If $p_{XY} > 0$ the two variables are directly correlated, if $p_{XY} = 0$ there is no linear correlation and if $p_{XY} < 0$ are inversely correlated.

2.2 Grid of Points and Lucene

The core of this thesis is around the *Grid of Points*[4], i.e. system runs originated by all the possible combinations of a set of targeted components, using Apache Lucene [5]. Runs are generated using different combinations of *stop lists*, *stemmers* and *IR models*; in particular in this work we use 7 stop lists, 5 stemmers and 10 IR models. Configurations used are listed below:

[4]http://gridofpoints.dei.unipd.it
[5]http://lucene.apache.org

- **stop lists**: glasgow, lucene, atire, nostop, indri, lingpipe, smart;
- **stemmers**: 5grams, porter, krovetz, nostem, harman;
- **IR models**: dfiis, bm25, lmd, af2exp, af2log, bool, dfrinexpb2, iblgd, lmjm, lucene.

2.2.1 Stop Lists

In this section there is a brief analysis about the stop lists adopted in this work:

- **nostop**: trivially defines the absence of any type of stop list. None word is stopped and all the tokens are passed to the stemmer;
- **glasgow**[6]: is composed by 319 terms;
- **lucene**[7]: is composed by only 33 words and it is the stop list adopted by Lucene;
- **atire** [8]: has 988 terms;
- **indri** [9]: it is used by the Indri IR system and contains 418 terms;
- **lingpipe** [10]: has 76 terms;
- **smart** [11]: is composed by 571 words.

[6] http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words
[7] https://github.com/apache/lucene-solr/blob/master/lucene/core/src/java/ ... /StandardAnalyzer.java
[8] http://www.atire.org/hg/atire/file/tip/source/stop_word.c
[9] https://sourceforge.net/p/lemur/code/HEAD/tree/indri/trunk/site-search/stopwords
[10] http://alias-i.com/lingpipe/docs/api/com/aliasi/tokenizer/EnglishStopTokenizerFactory.html
[11] http://ftp.gnome.org/mirror/archive/ftp.sunet.se/pub/databases/full-text/smart/english.stop

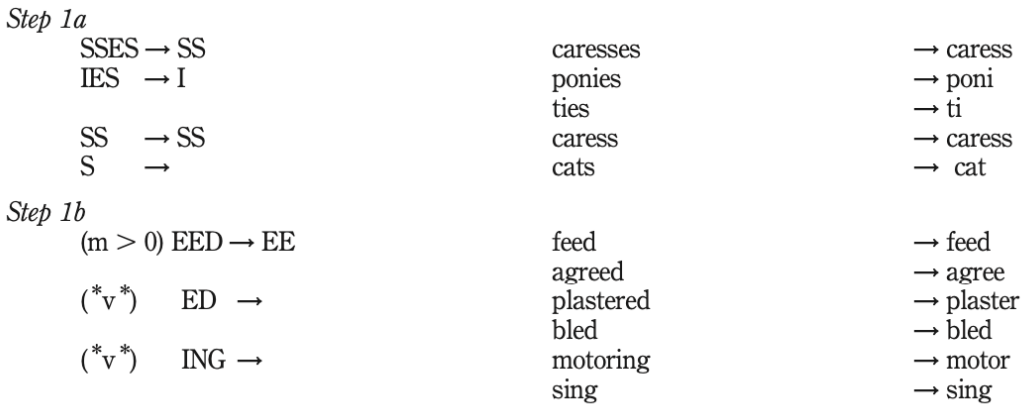

Figure 2.8: An example of first step of Porter stemmer [4].

2.2.2 Stemmers

In this thesis we use 5 types of stemmers, and each of them can be more or less aggressive than another.

- **nostem**: trivially defines the absence of any type of stem;
- **5grams**: is a sort of stemmer that creates sequences of n words. For example, given the word "tree", the output of a 2-gram would be "tr", "re", "ee". N-grams are particularly useful if it is needed to deal with language independent alternatives to more complex stemmers [25]; clearly n-grams produce larger indexes. In this work we are going to use a n-gram with $n = 5$ since it adapts to the English language [26].
- **porter**: this algorithmic stemmer is one of the most common in IR field. It is based on a set of rules for removing suffixes [1]. An example of the first rule of this algorithm is represented in figure 2.8. In its first version developed in 80_s, this stemmer had a lot of false positives or false negatives but these problems are fixed in its second version [1].
- **krovetz** [27]: it is a stemmer as aggressive as porter [26]. It is an hybrid approach between an algorithm and a dictionary-based stemmer [1]. This stemmer produces a lower rate of false positives than porter, but it may have an higher rate of false negatives depending on the size of the dictionary. However, the performances of these two stemmers are similar but sometimes krovetz is preferable since it produces full words where porter may produce word fragments.

- **harman**: or "S-stemmer" from Donna Harman [28]. In this work this stemmer is implemented by the class *EnglishMinimalStemmer* of Apache Lucene.

2.2.3 IR Models

As stated above, in this work we are going to use 10 different IR models. However, some of them are built on similar principles and so they can be grouped in 6 main categories. It follows an explanation of each of them.

Vector Space Model

In this category we can find the vector space model of **Lucene**[12]. Given a query q and a document d, it can be expressed by the following formula:

$$score(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{\text{t in q}} (tf(\text{t in d}) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t,d))$$

Let's have a more detailed analysis of these components.

- **tf**(t in d): it refers to the *term frequency* measure. So for each query term, it is computed its frequency in the document and so the tf according to Lucene method is:
$$tf(\text{t in d}) = \sqrt{frequency}$$
- **idf**(**t**): stands for *inverse document frequency*. In the formula above it is squared since the term appears both in query and in the document. Considering $docFreq$ the number of documents in which the term t appears, and $docCount$ the number of considered documents, $idf(t)$ is:
$$idf(t) = 1 + log(\frac{docCount + 1}{docFreq + 1})$$
- **coord**(**q**, **d**): is a score based on how many times query terms are found in the document;
- **queryNorm**(**q**): is a normalizing factor in order to make scores of different queries comparable. This factor does not affect document ranking;

[12] https://lucene.apache.org/core/6_6_0/core/org/apache/lucene/search/similarities/ClassicSimilarity.html

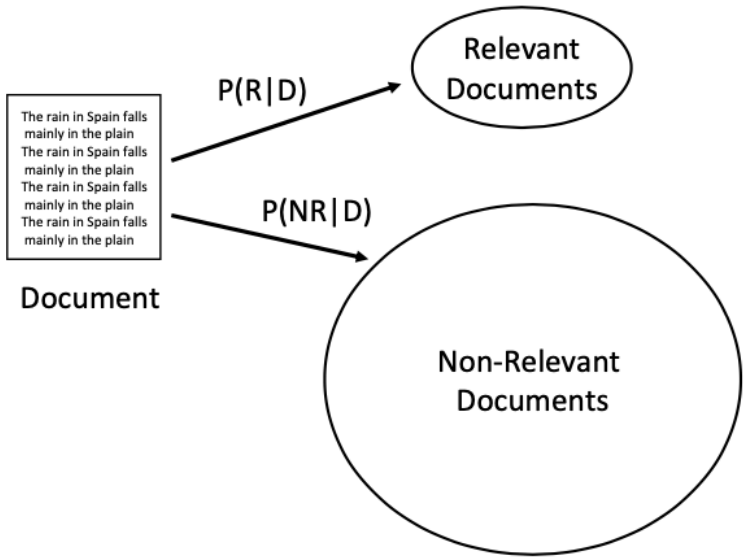

Figure 2.9: Bayes Classifier [1].

- $\mathbf{t}.\mathbf{getBoost}()$: is a search time boost of term t in the query q;
- $\mathbf{norm}(\mathbf{t}, \mathbf{d})$: encapsulates a few (indexing time) boost and length factors.

Probabilistic Model

Among used IR models, the **BM25** model fits this category. But first we have to comprehend how a probabilistic IR model works. Since we are dealing with probabilities, it is reasonable to classify a document D as relevant if the probability of being relevant given the document is higher than the probability of not being relevant given the document.

$$P(R|D) > P(NR|D)$$

This is known also as *Bayes Decision Rule* and this system is named *Bayes Classifier* 2.9. Thanks to the *Bayes' Rule*, we can state a relationship between $P(D|R)$

and $P(R|D)$ as follows:

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

Where$P(R)$ is *a priori* probability of relevance and $P(D)$ is like a normalizing constant. So, classifying a document as relevant is equal to have:

$$P(D|R)P(R) > P(D|NR)P(NR)$$

Or, alternately:

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

To determine values for $P(D|R)$ and $P(D|NR)$, we need to state some assumptions. First of all we need to represent the document with binary features as: $D = (d_1, d_2, ..., d_t)$ where d_i is equal to 1 if that term is in the document, otherwise it is 0. Another assumption we need is that term in a document are independent from each other (*Naïve Bayes* assumption). So now $P(D|R)$ (and similarly $P(D|NR)$) can be computed as:

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

Introducing p_i as the probability of term i occurring in a document of the relevant set, the above equation becomes:

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

And equivalently for $P(D|NR)$:

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

with s_i probability of term i occurring in the non-relevant set. So, going back, now we can rewrite the ratio between $P(D|R)$ and $P(D|NR)$ as:

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

Doing some mathematical manipulations, we obtain:

$$\prod_{i:d_i=1} \frac{p_i}{s_i} \cdot (\prod_{i:d_i=1} \frac{1-s_i}{1-d_i} \cdot \prod_{i:d_i=1} \frac{1-p_i}{1-s_i}) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i} = \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \prod_i \frac{1-p_i}{1-s_i}$$

The second product does not give any contributions in ranking since it is the same for all the documents. Moreover, since a product between a lot of small numbers may produce accuracy problems, we can instead use the logarithm where the product becomes a sum:

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

Since it may be difficult to obtain p_i and s_i, it is often used a *contingency table* (table 2.1) obtained through relevance feedback, where users identify relevant documents in initial rankings [1]. Where r_i is the number of relevant documents

	Relevant	Non-Relevant	Total
$d_i = 1$	r_i	n_i - r_i	n_i
$d_i = 1$	R $- r_i$	N $- n_i -$ R $+ r_i$	N-n_i
Total	R	N-R	N

Table 2.1: Contingency table [1].

with term i, n_i is the number of documents containing term i, R is the number of relevant documents or this query and N is the total number of documents. So, returning to our probabilities, p_i becomes r_i/R, and $s_i = (n_i - r_i)/(N - R)$. Since if $r_i = 0$ we would have $log 0$, in order to avoid this we add 0.5 to each count and 1 to the totals, obtaining $p_i = (r_i + 0.5)/(R + 1)$ and $s_i = (n_i - r_i + 0.5)/(N - R + 1)$. So, returning to the scoring function, we obtain:

$$\sum_{i:d_i=1} log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)}$$

This scoring function is not so effective as it is, but is the basis for **BM25**, one of the most used ranking algorithm. Its scoring function is:

$$\sum_{i \in Q} log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1)f_i}{K + f_i} \cdot \frac{(k_2 + 1)qf_i}{k_2 + qf_i}$$

In addiction to the general scoring function presented above, we have two additional factors with new components. We have the frequency f_i of term i in the document, the frequency qf_i of term i in the query and parameters k_1, k_2 and K. A more accurate description of these parameters is given in [1]. So **BM25** is a strong ranking algorithm, focused on topical relevance with the assumption that relevance is binary.

Boolean Models

In this category we can find the **bool** IR model. This model is not a proper ranking model since it does not rank the documents, but retrieves them according to a boolean evaluation: an outcome can have two possible results, TRUE and FALSE. It is known also as *exact match retrieval* since a document is retrieved only if respect exactly the query. According to this, the query is expressed by means of boolean operators (AND, OR, NOT). For example if we want to retrieve something about sun and mountain but not about snow, a possible query would be: "sun AND mountain AND NOT snow". But this model could have problems since it is possible that relevant documents may be completely discarded and documents with an informative content similar to the query may be not retrieved.

Divergence from Randomness Models

The model we adopted for this work that belongs to this category is the **dfrinexpb2**. It is a DFR model with Inverse Expected Document Frequency model with Bernoulli after-effect and normalisation 2. DFR models can be considered as a type of probabilistic models and they are based on a generalization of the Harter 2-Poisson model [29]. The 2-Poisson rather that a model is a framework to weighting terms using a probabilistic method. The DFR is based on the following idea: "The more the divergence of the within-document term-frequency from its frequency within the collection, the more the information carried by the word

t in document d. In other words, the term-weight is inversely related to the probability of term-frequency within the document d obtained by a model M of randomness"[13]. Note that in **dfrinexpb2** the model M used is the *Inverse Expected Document Frequency* model but there exist others like the Inverse Term Frequency model, the Inverse Document Frequency model or the Bose-Einstein distribution. Then in our case the *Bernoulli after-effect* acts as *first normalization* smoothing the information weight provided by the model chosen. Also in this case there exist different first normalization methods. Finally *normalization 2* stands for the *term frequency normalization* used.

Divergence from Independence Models

In this thesis we adopted **dfiis** (Divergence from Independence (DFI) with saturated measure of distance from independence), an application of Divergence from Independence models. DFI models gives performance on TREC test collections comparable to the state of the art models [30]. They are based on the following hypothesis: "a semantically selective word occurs in a semantically related content with a frequency different from the frequency of the word in common use, which can be determined by the saturated model of independence" [30].

Language Models

The model we adopted from this category is the Language model with Dirichlet smoothing (**lmd**). The language model concept wast first introduced by Ponte and Croft in their SIGIR 98 paper in which they proposed a query likelihood method. The main idea is to build a language model for each document and to rank them according to the likelihood of the query. More in general, the language model is built on a probabilistic model of the text [31].

Information Based Models

Another family of IR models are the Information Based Models [32], and among them we used the **iblgd** model (Information-based similarity with log-logistic distribution, lambda as average number of documents where w occurs, and normalization 2). These models are based on the concept that a term has not the

[13]http://terrier.org/docs/current/dfr_description.html

same statistics on the document and the collection. So it is possible to get the word's Shannon information in order to create a model. It is worth to pay particular attention on the distribution used, since in this model it makes the difference between a good or bad retrieval model. The **iblgd** model is based on a power-law distribution, and in particular on the log-logistic distribution. The evaluations conducted in [32] show that information based models have good performance comparing to other IR models such as language models, the Okapi BM25 and some DFR models.

Axiomatic Models

In [33] axiomatic models are introduced. In this work we used two different approaches of this type of models: the **af2exp** (axiomatic F2 exponential model) and the **af2log** (axiomatic F2 logarithmic model). As reported in Lucene documentation [14], this family of IR model is based on BM25, Pivoted Document Length Normalization and Language model with Dirichlet prior with some components modified in order to follow some axiomatic constraints. In particular, according to Lucene documentation, the actual realization of **af2exp** is:

$$\sum_{terms} (tfln(term\_doc\_freq, docLen) * IDF(term))$$

$$\text{with } IDF(term) = pow(\frac{N+1}{df(t)}, k)$$

Where $tfln$ is the the mixed term frequency and document length component, N is the total number of documents, df is the document frequency and k hyperparam for the primitive weighting function. The realization of **af2log** is similar:

$$\sum_{terms} (tfln(term\_doc\_freq, docLen) * IDF(term))$$

$$\text{with } IDF(term) = ln\frac{N+1}{df(t)}$$

The parameters are the same but the Inverse Document Frequency component is calculated using the logarithm instead the exponential.

[14] http://lucene.apache.org/core/8_1_1/core/org/apache/lucene/search/similarities/Axiomatic.html

2.3 Machine Learning

Together with Information Retrieval, this work is based on Machine Learning algorithms. In this paragraph an introduction to machine learning is presented, together with the explanation of the algorithms and models used. Most of the following treatment is based on concepts from [34].
Machine Learning (ML) is a branch of the AI (Artificial Intelligence) and studies the process of converting experience into knowledge, i.e. organize data and "learn" from them. A supervised ML process, receives in input training data (the experience) and produces output data (the knowledge) based on the input.
There are a lot of tasks performed by humans that can benefit from machine learning. Think about self-driving, speech recognition, computer vision etc. Machine learning can help humans in these tasks improving the quality, making life easier for people with disabilities and saving time. On the other hand these tasks are quite complex to realize and need to be refined with a huge amount of training samples. Moreover, there are other types of tasks that go over human capabilities since they are dealing with an huge amount of complex data that are impossible to fully comprehend for humans. Some examples of this type of tasks are made of complex data sets such as astronomic data, turning medical archives into medical knowledge, prediction and analysis of genomic data and the field treated in this thesis: *search engines and information retrieval*. Machine learning is very helpful with these types of data sets since it permits to find meaningful patterns obtaining substantial benefits. In addition with the significant increment of memory and computational speed of nowadays technologies, machine learning is becoming more and more important, reaching new horizons.
There exist many types of machine learning with different sub fields and tasks; in this work we are going to concentrate about *supervised learning*. This type of learning defines how the learner interacts with the environment: data received contains additional information about the task and this information is also known as label. Considering a task in which the learner has to define if an apple is good or not; in supervised learning the data contain, together with some information about the apple, the label good/bad. So we may say that the learner, in the training phase, has a *supervisor* that guides it in order to detect the best pattern in the data. For completeness, we introduce the opposite type of interaction

between learner and environment. We talk of *unsupervised learning* when the learner is not provided with labels in the training phase. In this case the learner has to obtain a summary of the data by means of other types of algorithms such as *clustering*.
After this brief introduction about a general idea of ML and its goals, it follows a more detailed an theoretical explanation about ML and its models.

2.3.1 General Model

First of all we need to define a formal model for learning tasks.

Learner's input:

- **Domain set:** an arbitrary set $\mathcal{X}$, i.e. the set of data that the learner wants to label. Items of this domain are represented by a vector of *features* and points of $\mathcal{X}$ (the *instance space*) are known as *instances*.
- **Label set:** represents labels $\mathcal{Y}$, i.e. the output of the learning process of the model. The output set may be different depending on the task, this aspect will be treated in following paragraphs.
- **Training data:** $S = ((x_1, y_1), ..., (x_n, y_n))$ is a finite sequence of pairs in $\mathcal{X} \times \mathcal{Y}$. This is the input provided to the learner.

Learner's output: the learner gives in output a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ known also as *predictior*, *hypotesis* or *classifier*. This function can be used to predict labels given unseen instances, i.e. new domain points.

Data generation model: data is generated by some probability distribution. More in general, we can identify as $\mathcal{D}$ the probability distribution over $\mathcal{X}$. Moreover, labels are generated by a "correct" labeling function $f : \mathcal{X} \to \mathcal{Y}$ with $y_i = f(x_i)$, $\forall\, i$. Note that this function is not known to the learner as the probability distribution $\mathcal{D}$ as well, in fact the learner goal is to find a model of the function f.

Measures of success: let's introduce the concept of *error of a classifier* as the probability of not predict correctly the label of an instance among the distribution

$\mathcal{D}$, i.e., given an instance x on the distribution $\mathcal{D}$ ($x \sim \mathcal{D}$), the probability that $h(x)$ is different from $f(x)$. So, the error of a predictor $h : \mathcal{X} \to \mathcal{Y}$ is defined as:

$$L_{\mathcal{D},f}(h) = \mathop{\mathbb{P}}_{x \sim \mathcal{D}}[h(x) \neq f(x)]$$

That is the probability of randomly choosing an instance x among $\mathcal{D}$ for which $h(x) \neq f(x)$. $L_{\mathcal{D},f}(h)$ is known as *generalization error*, *true error* or *risk*.

So, summing up, the learner receives a training set $\mathcal{S}$ and tries to find the best $h_s : \mathcal{X} \to \mathcal{Y}$. The goodness of h_s is evaluated by its true error, however the learner does not know $\mathcal{D}$ and f so it is impossible to calculate the true error of its predictor. For this reason, it is introduced the concept of *training error* (or *empirical error* or *empirical risk*), that is an error directly available for the learner in order to evaluate its prediction rule over training samples.

$$L_{\mathcal{S}}(h) = \frac{|\{i \in [m] \ : \ h(x_i) \neq y_i\}|}{m}$$

where $[m] = \{1, ..., m\}$.

So we can now identify the learner's goal, that is to find a prediction rule h which minimizes $L_{\mathcal{S}}(h)$. This practice is called *Empirical Risk Minimization* (ERM). However, not all the predictors are a good solution for the proposed task. In fact some predictors generalize "too well" training data leading to bad predictions if applied to real word data. This problem is known as *overfitting*. Despite that, we are still interested in using ERM for evaluate predictor's performances. For this reason we need to restrict the possible predictors in order to avoid overfitting. This procedure needs to be performed in advance and aims finding a set, an *hypothesis class* $\mathcal{H}$, and choose the predictor h within this set. So the problem is reformulated by using the ERM rule among all the possible $h \in \mathcal{H}$.

$$ERM_{\mathcal{H}}(S) \in \mathop{argmin}_{h \in \mathcal{H}} L_{\mathcal{S}}(h)$$

This practice introduces a bias in the choice of the predictor among the hypothesis class, this restriction is named *inductive bias*. However restricting the hypothesis class does not guarantee us to avoid overfitting and furthermore may introduce "too much bias" preventing us from finding a good solution. This may lead to the

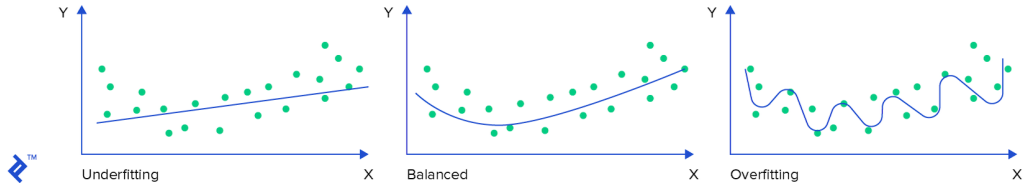

Figure 2.10: Underfitting and overfitting [5].

opposite concept of overfitting, namely *underfitting*. Underfitting occurs when the model is too simple with respect training data and not generalize them properly. In figure 2.10 it is possible to inspect visually difference between a underfitted model and a overfitted one. There are a lot techniques that permit to avoid and control overfitting, in the following paragraphs we are going to treat them.

2.3.2 Machine Learning Tasks

As mentioned before, machine learning can be applied to different types of tasks. However it is possible to divide them into two main groups, paying particular attention to the task of this thesis.

- **Classification:** the apple example mentioned before (classify whether an apple is good or not) is an example of a simple binary classification problem, i.e. associating to an instance a binary lable $0 - 1$. In addition, there exist more complex classification problems involving not a simple binary labels but a *class* of different labels; this task is know as *multiclass classification*.

- **Regression:** this task aims into finding a simple pattern in the data, more precisely a functional relationship between $\mathcal{X}$ and $\mathcal{Y}$. Usually $\mathcal{Y}$ is a real number and in this cases is called *target set*. A typical example of this set of problems is to predict house prices given a set of features describing the house, or predicting some measures related to IR system performances, topic of this thesis. For regression tasks we need to define a more accurate error to evaluate the goodness of the model. So, given a predictor $h : \mathcal{X} \to \mathcal{Y}$, the *expected square difference* between target values and predicted values is:

$$L_{\mathcal{D}}(h) = \underset{(x,y)\sim\mathcal{D}}{\mathbb{E}}(h(x) - y)^2$$

2.3.3 Validation

As mentioned before, the learner's goal is to find the best predictor among a finite class of predictors $\mathcal{H} = \{h_1, ..., h_n\}$. This process however is not trivial since the learner has to find the best predictor avoiding both underfitting and overfitting. To evaluate performances and obtain a more accurate measure of the true risk, we need to divide the training set into two parts: the training set and the *validation set*. So different algorithms (or the same algorithm with different parameters) are trained with the training set obtaining a finite class of predictors $\mathcal{H} = \{h_1, ..., h_n\}$. The validation set, since it is independent from the training set and $\mathcal{H}$, is used to evaluate predictor performances and the one with the smallest error is picked. Figure 2.11 gives a visual representation of the validation principle just described. As it is possible to see, the training error decreases as the model complexity grows since the algorithm learns better from the data. On the other hand, validation error decreases growing the model complexity but at a certain level it starts to increase. This behavior is due to overfitting since the algorithm fits "too well" training data and does not generalize just as well as validation data. So this gives us an important clue on the model to choose: the best option is to pick a model with a complexity relative to the minimum point of the validation curve diminishing so the risk of underfitting and overfitting.

k-Fold Cross Validation

Assume for example that the learner has not enough data for the training-validation split and the learner wants to have a more accurate error of the predictor. k-fold cross validation is a widely used solution to this problem and to tune algorithm's parameters; it increases performances' accuracy and it further defend us from overfitting. k-fold cross validation consists into dividing the training set into k folds: the training is performed on all the folds except one, and the validation error is computed on the remaining fold. This procedure is repeated using each fold once for validation, then all the errors are averaged producing an accurate version of the validation error (figure 2.12).

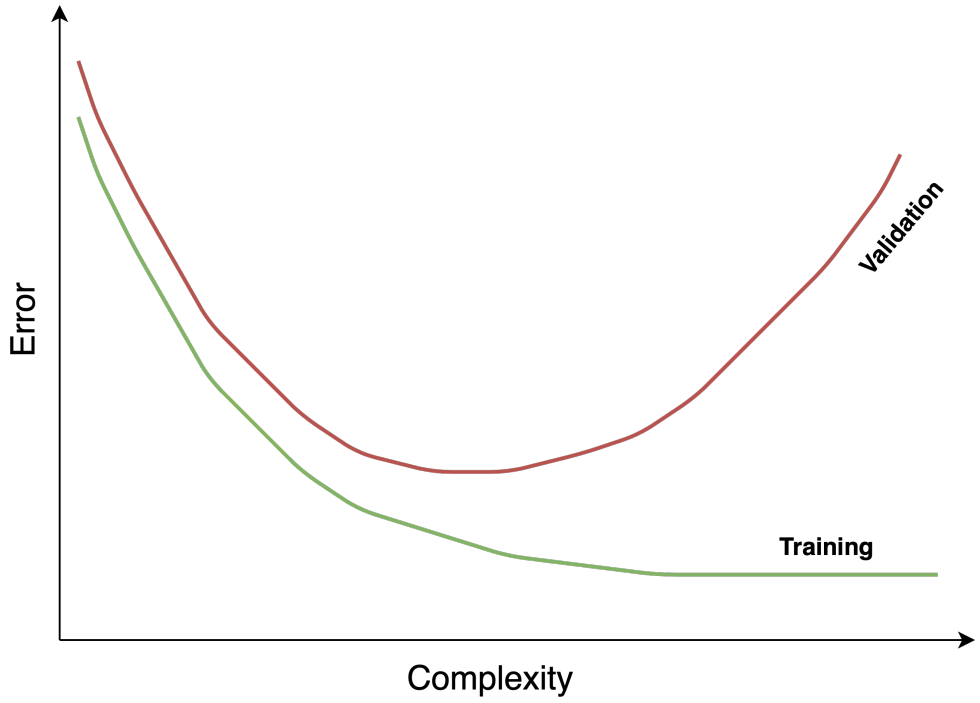

Figure 2.11: Model selection curve.

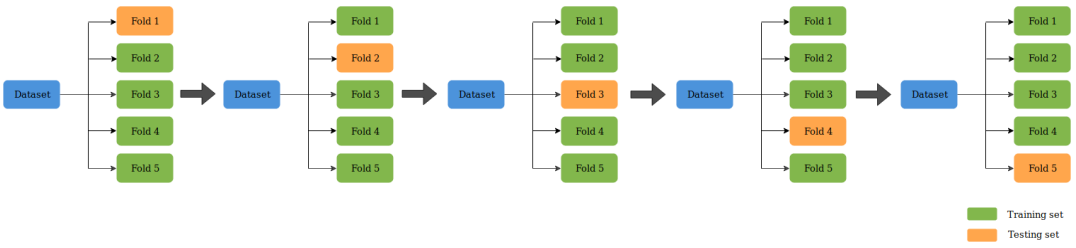

Figure 2.12: Example with a 5-fold cross validation [6].

2.3.4 Linear Models

Linear models are a family of predictors widely used for many problems. Most importantly, a lot of more complex predictors rely on linear predictors' principles. In this paragraph there is an overview on linear models and the presentation of two linear algorithms used for regression, main task of this thesis. First of all define the class of affine functions:

$$L_d = \{h_{w,b} \ : \ w \in \mathbb{R}^d, \ b \in \mathbb{R}\}$$

where

$$h_{w,b} = \langle w, x \rangle + b = \left(\sum_{i=1}^{d} w_i x_i \right) + b$$

In this notation L_d is a set of function parameterized by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, and each function, receiving as input a vector x, gives in output the scalar $\langle w, x \rangle + b$. The different hypothesis classes are compositions of a function $\phi : \mathbb{R} \to \mathcal{Y}$ where ϕ is related to the problem to solve. For example in binary classification problems ϕ could be the sign function, or in regression problems, where $\mathcal{Y} = \mathbb{R}$, it is just the identity function. For simplicity it could be convenient to incorporate the term b, the bias, into w as an extra coordinate. It is done just adding an extra coordinate with a value of 1 to all $x \in \mathcal{X}$. So we obtain $w' = (b, w_1, ..., w_d) \in \mathbb{R}^{d+1}$ and $x' = (1, x_1, ..., x_d) \in \mathbb{R}^{d+1}$. Obtaining the compact formula:

$$h_{w,b} = \langle w, x \rangle + b = \langle w', x' \rangle$$

For this reason in a lot of notation the bias is omitted since it is just included in w. The hypothesis classes used for binary classification are the *halfspaces*. Let us consider $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{-1, +1\}$ and so the class of halfspaces is defined as:

$$HS_d = sign \circ L_d = \{\mathbf{x} \rightarrowtail sign(h_{w,b}(\mathbf{x})) \ : \ h_{w,b} \in L_d\}$$

That is basically a combination of the class of affine functions L_d with the *sign* function. Each hypothesis is an hyperplane that is perpendicular to the vector w; the instances that are above the hyperplane are labeled as positive, the others as negative.

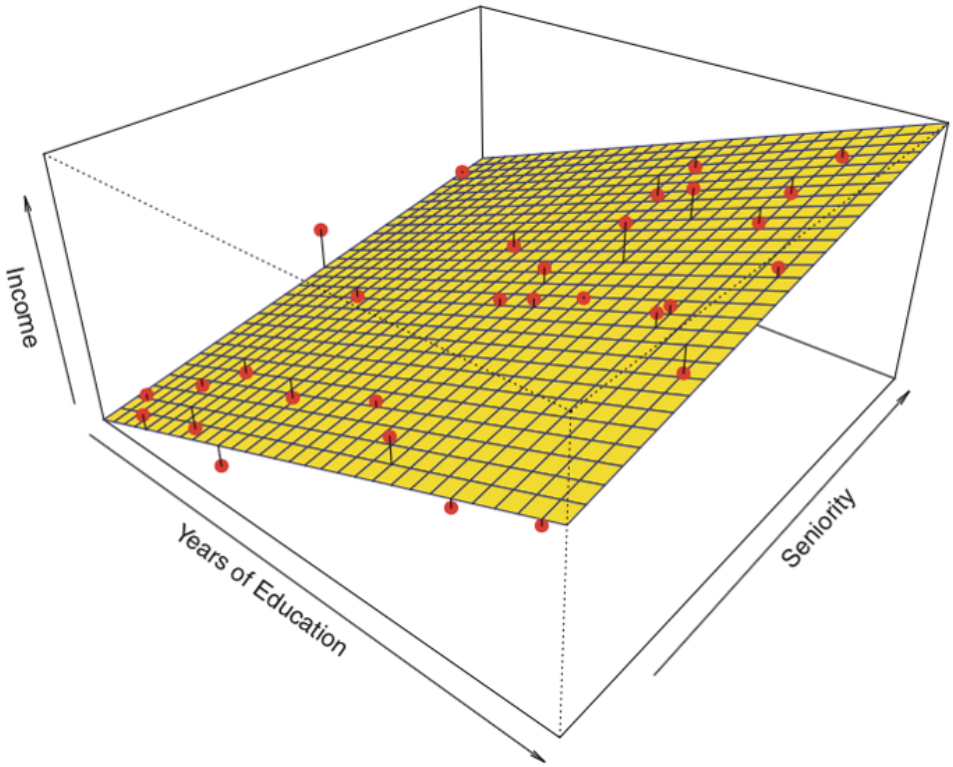

Figure 2.13: Linear regression example with $d = 2$ [7].

Linear Regression

Linear regression is one of the possible applications of linear models for regression tasks. The goal is to find a relationship between a set of variables and a real valued outcome. So the predictor tries to learn some linear function $h : \mathbb{R}^d \to \mathbb{R}$ that best approximates the relationship between training variables. In figure 2.13 there is an example with $d = 2$; the predictors tries to create a relationship (i.e. to find a function) between training variables (the years of education and the seniority) and the predicted output (the income). The set of hypothesis class for linear regression is just the set of linear functions, so:

$$\mathcal{H}_{reg} = L_d = \{x \rightarrowtail \langle w, x \rangle + b \ : \ w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

In this case, the loss function has to penalize the difference between $h(x)$ and

y, so it is widely used the *squared loss function*, namely:

$$l(h, (x, y)) = (h(x) - y)^2$$

For this loss function, the empirical risk function is called *Mean Squared Error* (**MSE**) and it is,

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

Another widely used function is the *absolute value loss function*, namely:

$$l(h, (x, y)) = |h(x) - y|$$

Ant its empirical risk function is the *Mean Absolute Error* (**MAE**), that is:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} |h(x_i) - y_i|$$

It is worth introducing a widely used metric for regression evaluation: the *coefficient of determination* known also as $\mathbf{R^2}$. It is a statistic used to evaluate the goodness of a regression. Its values are from $-\infty$ to 1; where 1 stands for perfect correlation between predicted data and true data. The coefficient of determination is evaluated as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (h(x_i) - y_i)^2}{\sum_{i=1}^{m} (y_i - \bar{y})^2}$$

Where $\bar{y}$ is the average among all the labels.

2.3.5 Support Vector Machines

Support Vector Machines (SVM) are a useful and well-known prediction tools. They are based on the assumption that data is linearly separable, and so there exists an hyperplane that classifies the training set. In practice, there are a lot of candidate hyperplanes and the aim of this predictor is to find the best one. Intuitively the best separating halfspace is the one with the largest **margin** since it can tolerate more noise. The *margin* is the minimum distance of the halfspace to a sample of the training set and the closest examples of the training set are

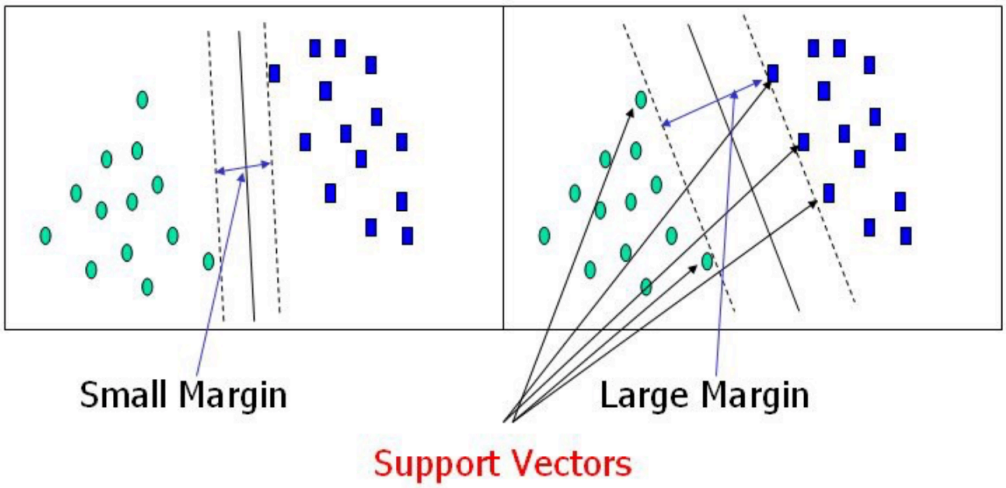

Figure 2.14: Support Vector Machines [8].

known as *support vectors* (figure 2.14).

Hard-SVM is a rule based on SVM principle which seeks for the hyperplane with the largest margin for linearly separable data. The problem can be formulated as:

$$arg \max_{(w,b):||w||=1} \min_{i\in\{1,...,m\}} |\langle w, x_i\rangle + b|$$

with $\forall i : y_i(\langle w, x_i\rangle + b) > 0$. This can be implemented as a quadratic optimization problem where, given in input a training set $(x_1, y_1), ..., (x_m, y_m)$ it is needed to solve:

$$(w_0, b_0) = \underset{(w,b)}{argmin} ||w||^2 \text{ s.t.} \forall i,\ y_i(\langle w, x_i\rangle + b) \geq 1$$

Giving the output $\hat{w} = \frac{w_0}{||w_0||}, \hat{b} = \frac{b_0}{||w_0||}$.

Hard-SVM are based on the constraint that training data must be linearly separable, but this assumption can not be always be true. For this reason, **Soft-SVM** modifies constraints of hard-SVM in order to adapt to non linearly separable data. Soft-SVM introduce a non-negative slack constraint $\xi_1, ..., \xi_m$ in order to modify the hard-SVM constraint into $y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i$. In practice, ξ_i determines how much the hard-SVM constraint is violated. So, soft-SVM both minimizes the norm of w (the margin) and the average of ξ_i (violation constraints). The pa-

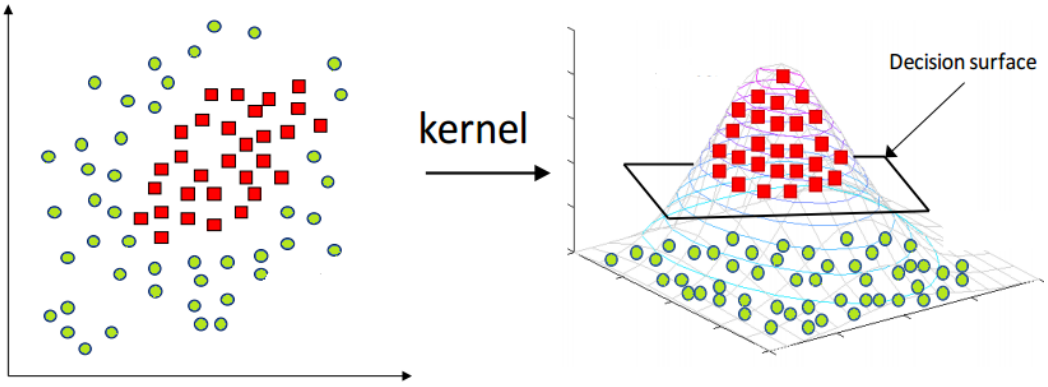

Figure 2.15: Non-linearly separable data and data after kernel application [9].

rameter λ controls the tradeoff between w and ξ_i. So the soft-SVM problem can be formulated for a training set and a parameter $\lambda > 0$ as solving the problem:

$$\underset{w,b,\xi}{min} \left(\lambda ||w||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right)$$

with the constraints $y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$. Giving in output w and b.

Kernel

However, it may happen that data are not linearly separable at all and even soft-SVM can not find a good separation rule (figure 2.15). In this case we need to find a non-linear transformation that maps original data in another (possibly) higher dimension space in which it is possible to find a good predictor, and so a good halfspace in that space.

First of all we need to find a non-linear transformation $\psi()$ in order to obtain a new set $S' = ((\psi(x_1), y_1), ..., (\psi(x_m), y_m))$ starting from the original set $S = ((x_1, y_1), ..., (x_m, y_m))$. Then it is possible to train a linear predictor h over S' and finally predict the label of a test point x by applying $h(\psi(x))$. More in general, we can define a *kernel function*:

$$K_\psi(x, x') = \langle \psi(x), \psi(x') \rangle$$

where $\psi(x)$ is the transformation of x.
Some of the most used kernel functions are:

- **Linear kernel**: $\psi(x) = x$.
- **Gaussian-radial basis function (RBF) kernel:** $K(x, x') = e^{-\gamma(x-x')^2}$ with parameter $\gamma > 0$.
- **Degree-Q polynomial kernel:** $K(x, x') = (\epsilon + \gamma \langle x, x' \rangle)^Q$ with parameters $\gamma, \epsilon > 0$ and for $Q \in \mathbb{N}$.
- **Gaussian kernel:** $K(x, x') = e^{-\frac{||x-x'||^2}{2\omega}}$ where ω is the control parameter.

Support Vector Regression

Support Vector Regression (**SVR**) is a prediction tool based on SVM principles but for solving regression problems instead of classification ones. It is based on the following function to be minimized:

$$\frac{\lambda}{2}||w||^2 + \sum_{i=1}^{m} V_\epsilon(y_i - \langle x, w \rangle - b)$$

where

$$V_\epsilon(r) = \begin{cases} 0, & \text{if } r < \epsilon \\ |r| - \epsilon, & \text{otherwise} \end{cases}$$

Where ϵ defines a margin of tolerance as shown in figure 2.16.

2.3.6 Random Forests

Random Forests are a prediction tool for classification and regression widely used in the machine learning environment. They are based on multiple decision trees and the output of each tree is combined in order to obtain the final output of the algorithm. But, before understanding how a random forest works, here follows a brief overview on decision trees.

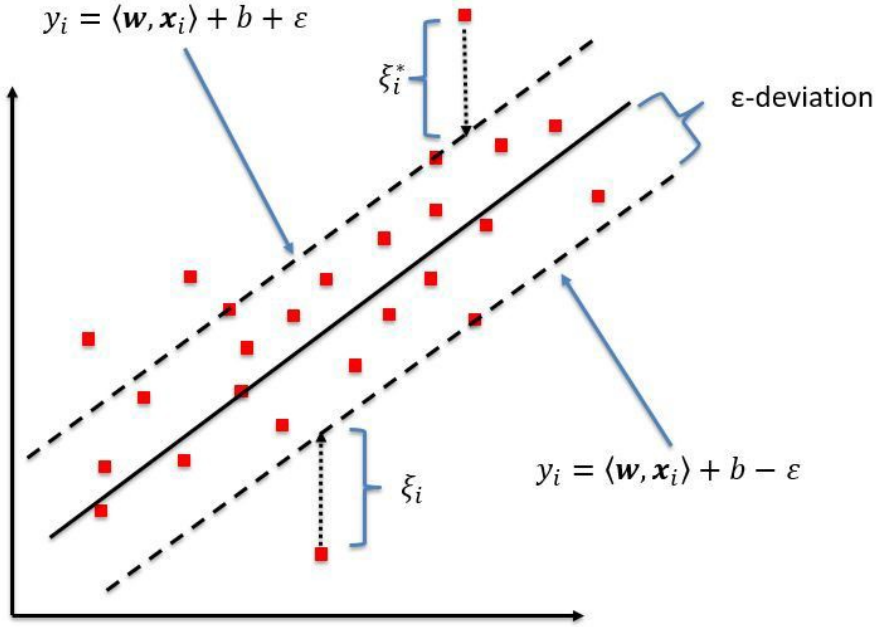

Figure 2.16: SVR rule [10].

Decision Trees

A decision tree is a predictor $h : \mathcal{X} \to \mathcal{Y}$ which predict the y value related to an instance x starting from the root node and arriving to a leaf node, i.e. the predicted output (figure 2.17). Usually the splitting is based on the set of features of the dataset or on another specified set of splitting rules. Leaf nodes contain the classes to predict in case of a classification task, or a real value for regression tasks. It is easier to understand looking at the figure 2.17: this simple decision tree tries to predict the type of car owned by a person basing on if it is married or not and if it is over or under 30 years.

However, decision trees tend to overfit; the whole dataset is processed through a decision tree and it may be difficult finding a good splitting rule or some correct features in order to make the two splitted groups as different from each other as possible. Random forests correct this bad behavior of decision trees: they combine multiple decision trees in a clever way and the result of a random forest comes out from the results of multiple decision trees (figure 2.18). In fact, the main concept

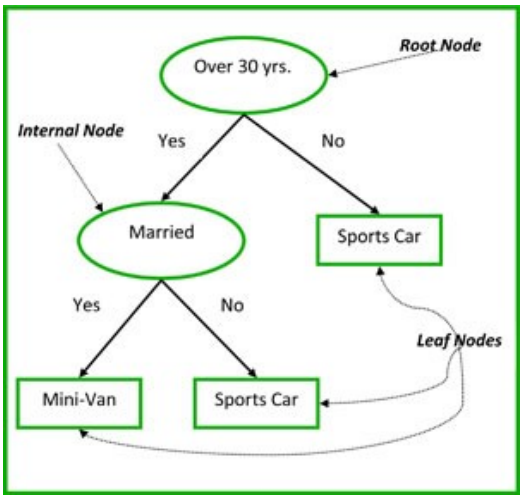

Figure 2.17: Example of a decision tree [11].

on which random forests rely is that decision trees are uncorrelated. In practice, the trees protect each other from their individual errors since individual trees have low correlation. This happens since at each node the number of features that can be split is a percentage of the total. This property ensures that among all the trees there is not a dominant feature, but instead each feature can give its equal predictive contribution. Moreover each tree processes a subset of the whole training set, providing further randomness and thus preventing overfitting. The final prediction of the random forest is the average of all the outputs of the decision trees of the forest. In this work we are interested in regression tasks but it is worth mentioning that random forests are accurate also for classification tasks. In addition this predictor works well with large databases and with a large number of input variables.

2.3.7 Gradient Boosting Regressor

Gradient Boosting Regressor is a machine learning technique for regression tasks. As random forest it is based on several decision trees, but with some differences. Below there is a brief overview of all the steps of a Gradient Boosting Regressor:

- The first step is to build a decision tree in which internal nodes provide splitting based on input features and leaves nodes contain residuals of target values. Residuals are computed by subtracting to each target value the

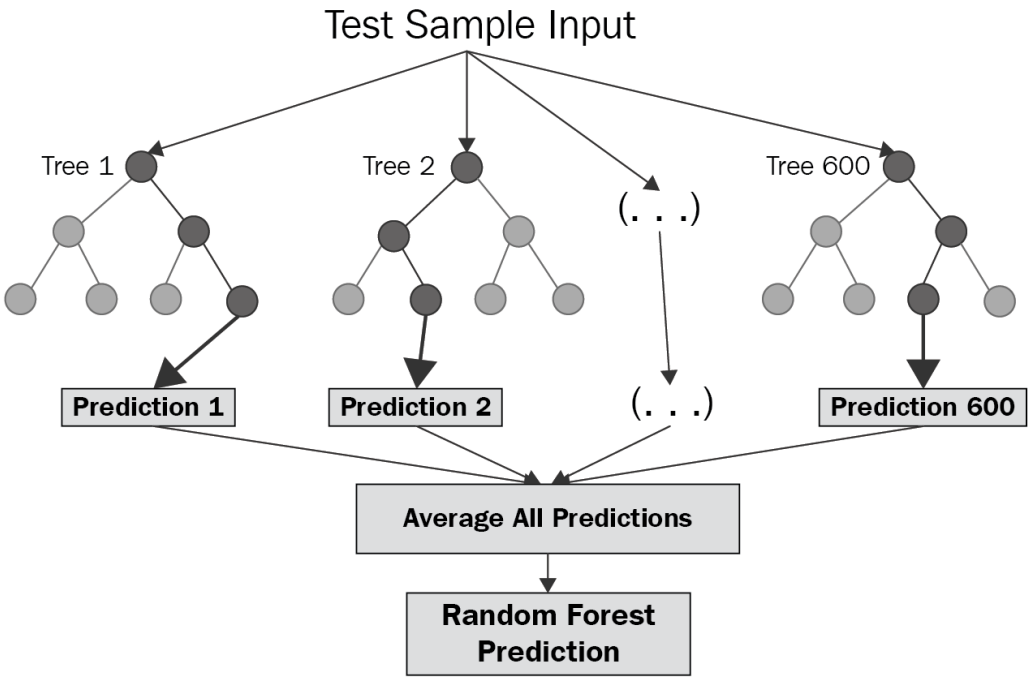

Figure 2.18: Example of a random forest regression [12].

average of all the target values. If the decision tree is built in a way that in a leaf there can be multiple residuals, the value associated to that leaf is the average of those residuals.

- Then each sample is passed through the decision tree falling into a residual. The temporary predicted value is computed as:

$$[\text{predicted value}] = [\text{avg target value}] + LR \cdot [\text{predicted residual}]$$

where LR is the *learning rate*, that is passed as algorithm parameter to prevent overfitting. Adopting this technique we need more trees, and each tree provides a further step to the final solution.

- Then, for each instance, we need to compute the new residuals as the difference between the actual value and the predicted value in the previous step. These residuals are used still as leaves values for the decision tree.

- Step 2 and 3 are repeated as many times as specified by the parameter *'number of estimators'*.

- Once trained, all the trees can be used to make the final prediction of the input variables as (figure 2.19):

  $\hat{\mathbf{y}} = [\text{avg targets}] + LR \cdot [\text{predicted residual 1}] + LR \cdot [\text{predicted residual 2}] + ...$

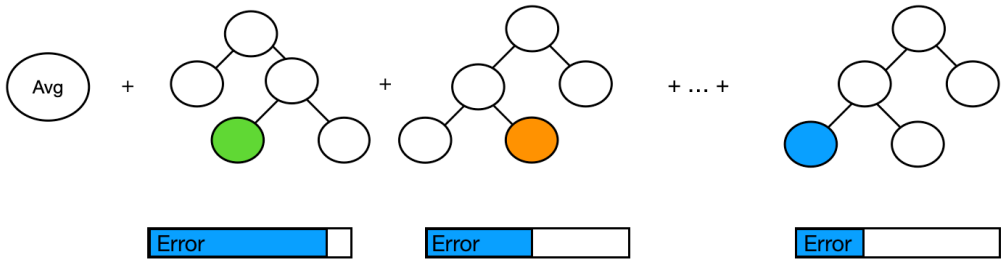

Figure 2.19: Gradient Boosting Regressor schema.

3 Methodology

In this chapter we define the methodology behind our work. First we introduce the two main tasks, explaining the idea of dataset splitting that is going to be tested in chapter 5. Then we introduce the features used for producing our evaluation. We divide them into four main groups, explaining how they are combined in our experiments and the criteria behind that. Finally, we introduce the models that permit us to run our tasks and the measures that give us a comparison tool.

3.1 Tasks

Our work is articulated on two main tasks, each based on a different approach for splitting the dataset. As previously mentioned, a IR system can be described by three components: a stop list, a stemmer and a IR model. **GoP**[1] permits us to obtain different IR systems by the combinations of our components. To fully comprehend our work, we can imagine our **GoP** as a parallelepiped where each dimension represents the realisations of one of the three components. In our experiment, inside our box, we can identify 350 different points, each of them representing a unique realisation of a IR system.

[1] http://gridofpoints.dei.unipd.it

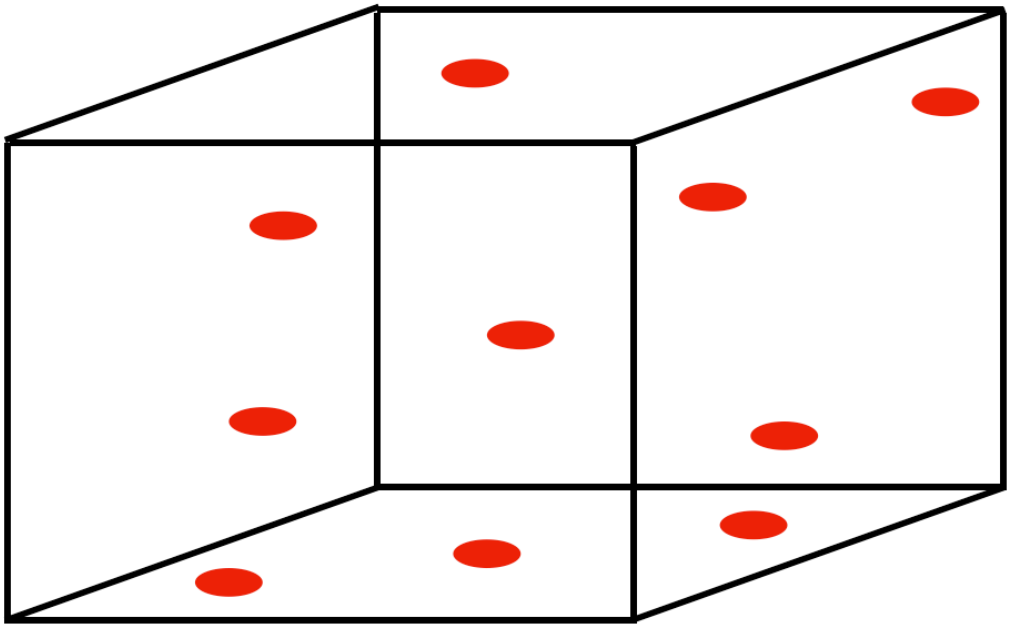

Figure 3.1: Visual representation of predicted IR systems in the first task.

3.1.1 First task

The first task consists into randomly pick some configurations and use them as test set: in this way the model is trained on some configurations and tested on another set of systems. A widely used practice in machine learning, is to use the 30% of the original data for testing and the remaining for training. So, 105 configurations among 350 are randomly selected as test set, and the remaining 245 are used for the training set. Recalling the visual representation of the **GoP** as a parallelepiped, in figure 3.1 the systems on which performing the prediction are represented as dots randomly selected among all the dots of our **GoP**.

3.1.2 Second task

The problem changes in the second task. IR systems of the test set are no longer randomly selected. In this task the test set is composed by all the systems that have a selected component, for example all the systems with a particular realisa-

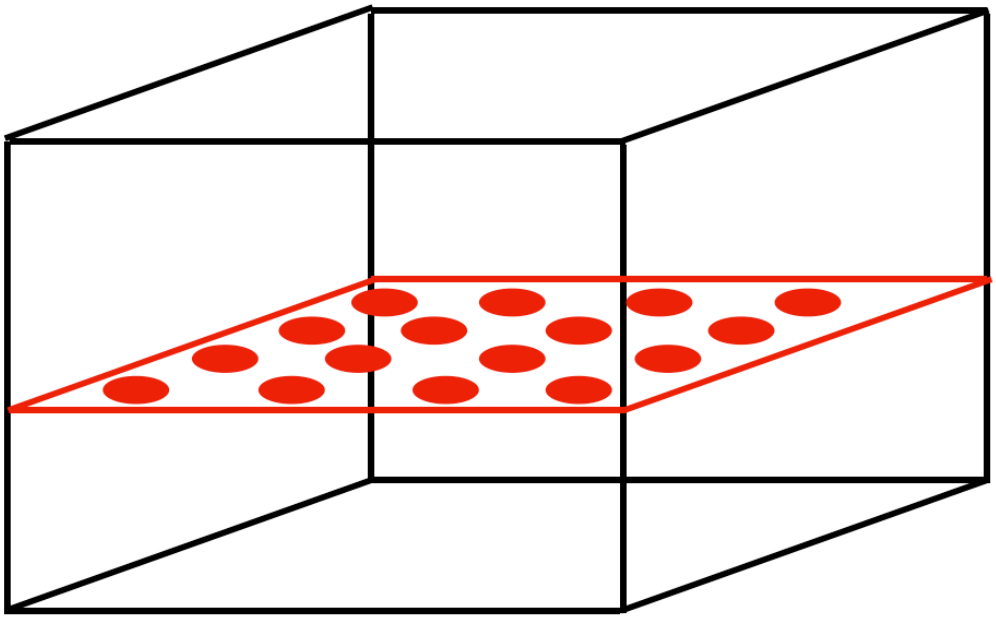

Figure 3.2: Visual representation of predicted IR systems in the second task.

tion of the stemmer. Figure 3.2 gives a visual representation of this idea, showing the plane of points with the same component. This introduces a further difficulty with respect to the previous task. Formerly, the test set was composed by some system configurations that were not present in the training set, but, however, the algorithm had the chance to learn the model using each IR component. In this task, instead, the learning process is not built on every possible component, forcing the model to give more importance to other components rather than finding a particular pattern among them.

3.2 Features

In this section we present the features that are going to compose our dataset, while how they are computed and further details are provided in the following chapter. Our features can be divided in to four main groups.
In the first group we find the set of categorical features describing the IR system involved, specifying its stop list, stemmer and IR model realisations. This fea-

ture set counts 22 features, one for each component involved in our task and, if a component is part of a specific system, it is represented by a 1, otherwise by a 0. The second feature set contains further information about the components that describe the IR system. There are four features: two for the stop list, one for the stemmer and one for the IR model.
The third group is composed by 8 features. Each of them is a statistical description of the query involved for a specific topic. In particular two of them express statistics regarding the number of query words and their length, while the others express different query word statistics within the collection.
The last feature set counts two features that provide a description of query result following the retrieval process by analysing its ranking score. Since these features describe a information following the retrieval task, they are named *post retrieval features*.
Summing up, for each corpus we have 350 different IR system configurations, and each of them has 50 samples (one for each topic) with a total of 17500 samples. Each sample is described by a set of 36 features: 22 are categorical and related to the IR system configuration, 4 describe the system's component, 8 are query-related features and 2 features belong to the post-retrieval category (figure 3.3). For our experiments we combine these sets of features in different way, in order to capture their differences and to analyse their behaviour for different tasks. Each feature set is tested alone, in order to compare its performances with other feature set. We are also interested into inspecting the performance when a predictor is trained on the whole feature set, i.e. when we provide our model with all possible information. Then we combine system related features, i.e. categorical features and component properties, in order to obtain a super-group strictly related to the IR system. In contrast to this set, we create a feature set strictly related to the topic by the union of the query feature set and post retrieval features. Forming these groups permits us to inspect their differences, analysing when the IR system or the topic have a major weight and if the combination of two sets may lead to an improvement.

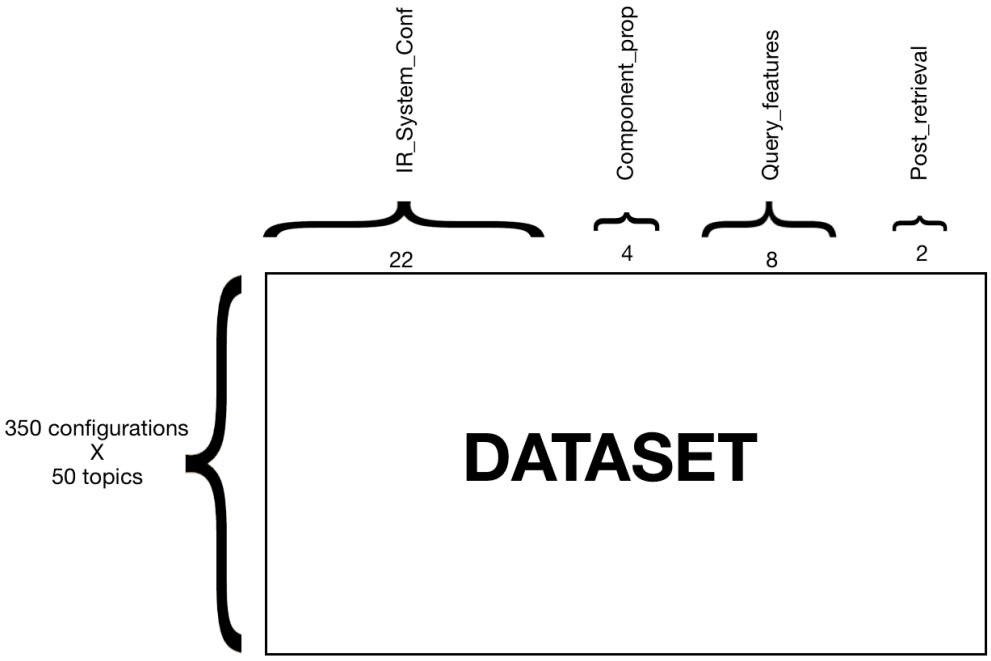

Figure 3.3: Dataset components.

3.3 Predictors

The tasks previously presented are tested on four different predictors. The first model used for our purposes is Linear Regression; it provides the simpler regression model by means of a linear function, as explained in the chapter 2. Then we test our dataset on a Support Vector Machine, and in particular on its realisation for regression tasks, the Support Vector Regression (SVR). Also this predictors assumes that data is linearly separable (if not it uses a kernel function), and it aims into finding the best hyperplane that best fits our data. However, it is a more complex model than Linear Regression, since it searches the halfspace with the largest margin, that is the distance between the halfspace and the support vectors.

Then we increase the complexity of our models, by passing from linear predictors to ensemble models. We implement a Random Forest Regression, that is an ensemble model by means of a bagging technique. This means that it is a model that combines predictions of multiple learners (the trees) run independently and

with no iteration between them. We use also an ensemble model by means of boosting technique, that is the Gradient Boosting Regressor. This predictor combines the resources of multiple learners but, unlike Random Forest, its workers cooperate each other, by using the result of a learner to improve the performance of another one thus making weak learners into strong learners.
For each task we run the four predictors evaluating their performances. We study their results for query prediction for each system, by analysing both training and test scores. These evaluations are based on the most used scores for regression problems. We use the **R2 score** with its range of values from $-\infty$ to 1 that gives an immediate view of the prediction goodness, the **Mean Absolute Error** (MAE) that gives a clue of the real gap between the predicted value and the real one, the **Mean Squared Error** (MSE) and its squared root, i.e. the **Root Mean Squared Error** (RMSE) that are widely used measures for IR tasks.
To compare the IR systems we use the Pearson correlation coefficient, that is an index that expresses the linear correlation between the performance of the predicted systems and their real performance. Moreover, we are interested into ranking IR system performances and it is done by using the Kendall's tau ranking coefficient that defines the goodness of the predicted ranking with respect to the correct ranking.

4

Experimental Setup

Before introducing the experimental results, it is necessary to fully comprehend the data on which the work is done. First of all we describe the corpus of documents and the relevance judgements used to obtain the evaluations, giving some statistical information on the documents and topics involved. Then, we explain how works the process to create the Grid of Points with Java: first we describe the classes and their scope and following the proper creation of runs, obtained executing the Grid of Points on the corpus presented in the first paragraph. Once we have the runs, it is possible to build the dataset on which performing the predictions. First of all we explain how to obtain evaluation measures, which are going to be the target of the predictions. Next we proceed extracting from the corpus and the runs useful data that are going to compose the feature set, i.e. input data, for prediction algorithms.

4.1 Collection

4.1.1 Documents

In this thesis we use the TREC 08 (1999) collection which is TIPSTER disk 4 and 5 minus Congressional Record, counting an amount of 528155 documents. Table 4.1 gives a general overview of the collection involved.

Disk	Size (Mbytes)	# Documents	Median # Words/Doc	Mean # Words/Doc
Disk 4				
the Financial Times, 1991-1994 (FT)	564	210,158	316	412.7
Federal Register, 1994 (FR)	395	55,630	588	644.7
Disk 5				
Foreign Broadcast Information Service (FBIS)	470	130,471	322	543.6
the LA Times	475	131,896	351	526.5

Table 4.1: TREC 08 collection.

As it is possible to notice in the table, TREC 08 collection has four main groups of documents (Financial Times, Federal Register, Foreign Broadcast Information Service and LA Times) and each document of the collection has in its ID a suffix that represents its shard (FT, FR, FBIS and LA).

4.1.2 Topic

TREC 08 collection is composed by 50 topics with IDs from 401 to 450. As previously stated, each topic is described by three fields: a title, a description and a narrative section (figure 4.1).

```
<top>

<num> Number: 401
<title> foreign minorities, Germany

<desc> Description:
What language and cultural differences impede the integration
of foreign minorities in Germany?

<narr> Narrative:
A relevant document will focus on the causes of the lack of
integration in a significant way; that is, the mere mention of
immigration difficulties is not relevant.  Documents that discuss
immigration problems unrelated to Germany are also not relevant.

</top>
```

Figure 4.1: TREC 08 topic sample.

Table 4.2 provides a brief summary of topic statistics for TREC 08. Those statistics are computed on token length including stopwords and without applying any stemming technique.

	Min	Max	Mean
Title	1	4	2.5
Description	5	32	13.8
Narrative	14	75	35.5
TREC 08 (401-450)	**23**	**98**	**51.8**

Table 4.2: TREC 08 topic statistics.

4.1.3 Pool

Relevance judgments are crucial for IR evaluation since for each topic it is necessary to make a list as complete as possible of relevant and non-relevant documents (relevance is assumed to be binary). The result of pooling is *qrels* (query relevances), i.e. judgments, stating whether a document is relevant to a topic. Note that a document is assumed to be relevant also if a piece of the document is relevant, even if this piece is very small. TREC guidelines [1] declare that a *qrels* file has to be composed by four columns:

- **topic**: topic number, i.e. the topic ID;
- **iteration**: is the feedback iteration (almost always zero and not used);
- **document #**: in this field there is the document ID;
- **relevancy**: in our case a binary value where 0 is non relevant and 1 is relevant.

Note that the order of documents does not have any influence on document relevance since it depends only by the relevancy grade (0-1). Then, if a document is not judged by human assessors it is assumed to be non-relevant. Figure 4.2 provides an example of a piece of TREC 08 *qrels*.

[1] https://trec.nist.gov/data/qrels_eng

```
401 0 FBIS3-19879 0
401 0 FBIS3-19881 0
401 0 FBIS3-19884 1
401 0 FBIS3-19893 0
401 0 FBIS3-19895 1
401 0 FBIS3-19897 1
401 0 FBIS3-19898 0
401 0 FBIS3-19905 0
401 0 FBIS3-19935 0
401 0 FBIS3-19951 0
401 0 FBIS3-19957 0
```

Figure 4.2: TREC 08 qrels sample.

4.2 GoP Creation

The first step in order to create the dataset is to build a **GoP**[2]. As mentioned before a GoP can be defined as system runs originated by all the possible combinations of a set of targeted components, using Apache Lucene [3]. Basically, given IR systems components, i.e. stop lists, stemmers and IR models, the idea is to create all the possible combinations of available components, this obtaining different IR systems. The components used are:

- **stop lists** (7 components): glasgow, lucene, atire, nostop, indri, lingpipe, smart;
- **stemmers** (5 components): 5grams, porter, krovetz, nostem, harman;
- **IR models** (10 components): dfiis, bm25, lmd, af2exp, af2log, bool, dfrinexpb2, iblgd, lmjm, lucene.

So all the possible combinations lead to $7 \times 5 \times 10 = 350$ different IR systems runs, providing components with different approaches (chapter 2) and an overview of some of the possible solutions achievable with Apache Lucene.

GoPs creation is managed by the *gopal* Java library[4]. The library can be easily configured thanks to a parametric Java properties file where it is possible to define

[2]http://gridofpoints.dei.unipd.it
[3]http://lucene.apache.org
[4]https://bitbucket.org/frrncl/gopal/

```
# The base directory where indexes are stored
it.unipd.dei.gopal.index.path = /ssd/data/padoan/Materiale/gopal/tipster/indexes

# The size in megabytes of the RAM buffer to be used for indexing
it.unipd.dei.gopal.index.ramBuffer = 5120

# The number of parallel workers to be used for indexing
it.unipd.dei.gopal.index.workers = 72

# The number of parallel workers to be used for searching
it.unipd.dei.gopal.search.workers = 72

# The expected number of documents in the corpus

# TIPSTER, TREC Disks 4 and 5 minus Congressional record
it.unipd.dei.gopal.corpus.size = 528155

# The base directory where runs are stored
it.unipd.dei.gopal.run.path = /ssd/data/padoan/Materiale/gopal/tipster/runs

# Declaration of the stop lists to be used
it.unipd.dei.gopal.stopList.labels = glasgow, lucene, atire, nostop, indri, lingpipe, smart

# Declaration of the stemmers to be used
it.unipd.dei.gopal.stemmer.labels = 5grams, porter, krovetz, nostem, harman

# Declaration of the IR models to be used
it.unipd.dei.gopal.model.labels = dfiis, bm25, lmd, af2exp, af2log, bool, dfrinexpb2, iblgd, lmjm, lucene
```

Figure 4.3: Parametric Java properties file for *gopal*.

which components use for each block (stop list, stemmer, IR model), their Java classes and some input parameters, such as the files containing the stop words [26]. This library is built to work in a multithreaded way since both indexing and searching processes work on independent workers. Therefore the configuration file provides also to state how many workers to use for indexing and searching and the ram buffer allocated for each worker. In the Java properties files it is possible to define paths for saving results and retrieve useful informations (such as corpus, pools, runs etc.) too. An explicative example of some of the most important parameters we used is represented in figure 4.3. In this example we have 72 workers both for indexing and searching so 72 different configurations are run in parallel until all the 350 configurations are processed.

The library works on the following packages [26]:

- **it.unipd.dei.gopal.analysis**: extends Lucene text processing by adding Lovins stemmer and *n-grams* (4-grams, 5-grams and the combination of them, i.e. 45-grams).
- **it.unipd.dei.gopal.gop**: it manages the multithread aim of gopal by worker creation for indexing and searching. Moreover in this package there are the

two classes we have used to run our processes, as we are going to explain later.

- **it.unipd.dei.gopal.index**: this package permits indexing document and processing of a corpus.
- **it.unipd.dei.gopal.parser**: implements parsing methods for different corpus such as Tipster and NYT (New York Times). Moreover implements XML document parsing and the class that represents a document to be indexed.
- **it.unipd.dei.gopal.search**: extends Lucene searcher implementing a multithread approach for GoP searching process.
- **it.unipd.dei.gopal.similarities**: extends Lucene default IR models adding support to Divergence from Independence (DFI) models, Divergence From Randomness (DFR) models and information-based models.

4.2.1 Runs

Once the configuration is done, it is possible to start with the indexing process. To start the process we have ran the **GopIndexerDriver** class inside the **it.unipd.dei.gopal.gop** package.

At this point, we have obtained the indexes for each of the 350 configurations and thus we have proceeded to create GoP runs. So we executed the **GopSearcherDriver** class inside the **it.unipd.dei.gopal.gop** package. The execution of this class gives in output 350 runs, one for each GoP configuration. Each run contains the ranking list of the first 1000 documents for each topic (the number of documents to retrieve is a parameter of the configuration properties file). A row in a run file respects TREC standards and in more in detail it contains:

- **topic**: topic number, i.e. the topic ID;
- **query number within that topic**: it is most of the times unused and usually set as Q0;
- **document #**: in this field there is the document ID;

- **rank**: progressive index that goes from 0 to 999 for each topic and indicates the rank of a document inside the topic;
- **score**: in this column there is the score of the retrieved document rigorously in descending order;
- **run tag**: this must be unique for the group and the method used.

Figure 4.4 represents a sample of one run.

4.3 Data extraction

In this section we describe how we have extracted data that is going to be used in learning models. First of all we describe how we have obtained, from the evaluation process, the measures that are going to be the target values of our prediction task. In the second part we explain how we have obtained features values, dividing them into pre-retrieval and post-retrieval categories.

4.3.1 Evaluation

At the end of the execution of the *gopal* Java library we have obtained the 350 runs, one for each IR system configuration. Consequently the following step is to evaluate performances of the systems obtained. Evaluation is performed using a Matlab library named *MATlab Toolkit for Evaluation of information Retrieval Systems (MATTERS)* [5]. This library is compliant with the TREC-format for pool and runs and moreover supports the computation of the most common measures for GoP runs. The measures computed by MATTERS for this work are widely used in Information Retrieval (see chapter 2) and are the following:

- **Average Precision** (AP);
- **Recall**;
- **Precision at 10** (P10);
- **Normalized Discounted Cumulated Gain** (nDCG);
- **Expected Reciprocal Rank** (ERR);

[5] http://matters.dei.unipd.it

```
401     Q0      FBIS4-18182     0       37.384987       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-18684     1       36.097157       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-59436     2       35.924698       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-18916     3       35.303467       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-9104      4       34.971752       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-18833     5       34.416161       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-31659     6       32.823460       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-31704     7       32.725719       upd_atire_krovetz_bm25_td
401     Q0      LA110990-0013   8       32.288155       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-39506     9       32.205700       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-39117     10      32.071655       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-9582      11      31.579205       upd_atire_krovetz_bm25_td
401     Q0      FT932-10861     12      31.480968       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-7981      13      31.443726       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-18223     14      31.189651       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-19881     15      31.170267       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-19951     16      31.169670       upd_atire_krovetz_bm25_td
401     Q0      LA110690-0075   17      30.964756       upd_atire_krovetz_bm25_td
401     Q0      LA123089-0101   18      30.913403       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-55036     19      30.607422       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-30880     20      30.246376       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-31541     21      30.202169       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-19212     22      30.086830       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-4201      23      30.058849       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-19829     24      29.989431       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-15890     25      29.888800       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-19645     26      29.887699       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-19702     27      29.687454       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-59042     28      29.619045       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-17077     29      29.520264       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-64139     30      29.484095       upd_atire_krovetz_bm25_td
401     Q0      LA032590-0082   31      29.457581       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-55980     32      29.446259       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-30735     33      29.403099       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-20335     34      29.358288       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-9504      35      29.334085       upd_atire_krovetz_bm25_td
401     Q0      FBIS4-7690      36      29.319391       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-8502      37      29.283882       upd_atire_krovetz_bm25_td
401     Q0      LA060389-0035   38      29.204670       upd_atire_krovetz_bm25_td
401     Q0      FBIS3-38677     39      29.151142       upd_atire_krovetz_bm25_td
```

Figure 4.4: Example of a run of the first 40 documents retrieved for topic 401 of TREC 08 collection with the atire-krovetz-bm25 IR system configuration.

- **Reciprocal Rank** (RR);
- **Rank-Biased Precision** (RBP).

In order to have an easy access to measures, we have written a simple Matlab script to convert evaluations obtained with MATTERS into a *csv* file since we have found this solution simpler to import data in python using *Pandas* library. So, for each of the seven measures, we have created a *csv* file with 50 rows (one for each topic) and 350 columns (one for each IR system configuration) making the value inside a cell the measure obtained for that topic searched with a specific search engine configuration 4.5.

Row	upd_atire_5grams_af2exp_td	upd_atire_5grams_af2log_td	upd_atire_5grams_bm25_td	upd_atire_5grams_bool_td	upd_atire_5grams_dfiis_td
401	0.000751821245937559	0.000997822371748018	0.00319178502406502	0.00026408352929186	0.0175311753362425
402	0.0762228354244716	0.074657989099879	0.117951024755476	0.027542574693085	0.216634531268084
403	0.370359358456423	0.353735365880026	0.530798463350594	0.0588238920016034	0.646244417506915
404	0.122837485613198	0.127065318695204	0.174326386948016	0.0563384473588928	0.174652712138327
405	0.0942555042879615	0.0835077042335675	0.129825700714469	0.0320288021216192	0.168510055811817
406	0.189757259375016	0.198353825911256	0.212472822245409	0.0841329690599358	0.22427957584045
407	0.117835013972484	0.118104062220311	0.1348632875261	0.0303858483536317	0.162884505379044
408	0.143496611200617	0.160331101108495	0.198074935688401	0.0629078916600984	0.245652506516198
409	0.0347344761169184	0.0371143907130476	0.0761733694010408	0.000409261205699992	0.116353989613564
410	0.390696133436918	0.349745943447171	0.62249282682325	0.0194492780266241	0.730529246326975
411	0.15783308968519	0.159204559204392	0.237634698893566	0.0185821260250912	0.308860246420723
412	0.0315270881059969	0.0317540921222187	0.0523808281556273	0.00754619412639439	0.271299657105155
413	0.134504097735976	0.135864280267047	0.157532925698561	0.008099325628337	0.110553684071032
414	0.0179470151833566	0.0177961649214553	0.0241367329599991	0.0106748754891474	0.0267396527032102
415	0.306785797450718	0.296077136634337	0.314581316707175	0.226622516413945	0.316965901978888
416	0.277663543190338	0.261244434734719	0.291733495532835	0.0609890314883606	0.273792465845537
417	0.00823696720657387	0.00826489962050848	0.00983513069578741	0	0.0251205300670033
418	0.319029936461954	0.29407419891603	0.313258991181248	0.00616757622589266	0.252588485343013
419	0.106987822119401	0.102071622114726	0.104971953278063	0.0263017703330802	0.07145964406031
420	0.0858213614762343	0.0795176503426968	0.144089274502442	0.0658797010906968	0.273991240287564
421	0.0296192811791515	0.0274327183262033	0.0320771752201599	0.00821030624687582	0.0349591643015492
422	0.00429898267861746	0.00470160371274251	0.0139892871058575	0.000992836090505384	0.0164816163058518
423	0.523426946161175	0.594554240181042	0.586797574860643	0.477174733682958	0.589927316835819
424	0.0773663549904291	0.0789514581960628	0.101681757385468	0.0357442106308154	0.130227712728918
425	0.000351601162293562	0.000219060308023524	0.000252696209791919	0	0.0704968652949406

Figure 4.5: Example of ap measures of the first 25 topics using 5 different configurations.

4.3.2 Pre-Retrieval Features

With pre-retrieval features are usually indicated those features that are obtained before the search takes place. This category of features is often easy to compute

glasgow	lucene	atire	nostop	indri	lingpipe	5grams	porter	krovetz	nostem	harman	dfiis	bm25	lmd	af2exp	af2log	bool
0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

Figure 4.6: Categorical vector for atire-5grams-af2exp system.

but generally the performances are lower than post-retrieval ones. However, since the computation of such features can be done quickly during indexing process, there are applications in which pre-retrieval features are preferred to post-retrieval ones, since they are computed after the searching process. So here follows all the pre-retrieval features we have used for our task.

Categorical Features

Since the core of this thesis are IR systems, it is fundamental to introduce features that describe the specific configuration adopted. Therefore, we used categorical features to represent the IR system configuration. We have built a vector of length 22 (7 stop lists + 5 stemmers + 10 IR models) where a 1 in the $i-th$ position indicates that a system adopts the $i-th$ component. Summing up, each configuration has three ones and nineteen zeros. For example, a system that uses *atire* stop list, *5grams* stemmer and the *af2exp* IR model, is described by the vector represented in figure 4.6.

Component Properties

Since we need to represent more accurately the IR system, we introduce features to describe the block of configuration used.

Regarding the stop lists we added two more features: one represents the number of words in the stop list and the other the average length of the words of the stop list. Table 4.3 resumes the features related to each stop list used in the Grid of Points.

Stop List	Words #	Avg Word Length
Glasgow	319	6.009
Lucene	33	3.909
Atire	988	6.516
Indri	418	6.502
Lingpipe	76	4.197
Smart	571	6.283
Nostop	0	0

Table 4.3: Stop list features.

Then we calculated a stemmer-related feature as ratio between the number of unique words in the corpus after stemming and the number of unique words in the corpus without applying a stemmer (both are computed with no stop list). We have found these values using methods provided by Java Lucene. Table 4.4 reports the ratio used for each stemmer.

Stemmer	Ratio
5grams	0.38433
Harman	0.97118
Krovetz	0.95600
Porter	0.93396
Nostem	1

Table 4.4: Stemmer features.

Finally, we introduced one more feature to better represent IR models too. In order to doing this, we have created GoP runs with *nostop*, *nostem* and with all the IR models listed above. Since we wanted a measure as much topic-independent as possible, we have designed a query containing only the top 760 most frequent words in English language from *https://www.wordfrequency.info*. The number of the top most frequent words is chosen to have as many words as possible but we had to consider Lucene limitations and time-efficiency too, so we found that number a good tradeoff. Clearly, runs with this type of query produce high scores

but we are rather interested into comparing IR models retrieval performance. As comparison performance measure, we have choose to use the *Normalized Query Commitment* (**NQC**), that is a widely used measure for post-retrieval tasks and it permitted me to compare different IR models. Although it is a post-retrieval measure since it is based on query scores, for our purpose it can be considered as a pre-retrieval one since it is computed on top-k most frequent English words and this computation can be done whenever before the proper retrieval task. Normalized Query Commitment measures the standard deviation of the retrieval scores on the set of top-k ranked documents, and it is normalized by the score of the whole collection. So, considering a corpus D, the set of relevant documents D_q of the query q, the top-k ranked documents D_q^k, and the average score μ, the NQC is computed as:

$$NQC(q) = \frac{\sqrt{\frac{1}{k}\sum_{d \in D_q^k}(Score(d) - \mu)^2}}{|Score(D)|}$$

Where for our purpose we have decided to set $k = 100$ comparing to 1000 retrieved documents.

The results obtained for each IR model is reported in table 4.5.

IR Model	**NQC**
dfiis	0.03779
dfrinexpb2	0.03637
lmd	0.062
lmjm	0.04127
af2exp	0.04525
af2log	0.04802
bool	0.11365
iblgd	0.04819
lucene	0.0371
bm25	0.03189

Table 4.5: IR model features.

Query-related features

Another set of pre-retrieval features is related to the query. It is worth noticing that this set of features is primarily related to the topic (each topic has its own query) but also to the specific configuration used since the query is processed differently on the basis of the selected stop list and the stemmer.

Two features are based on a naively linguistic approach, and are explained below:

- **number of query words**: which is simply the count of the words that compose the query (after applying the stop list and the stemmer);
- **average query word length**: that is, as the name states, the average number of characters that compose query words.

In addition, we computed six more query-related features which are derived from a statistical approach. These features are based on term frequency (*tf*) and inverse document frequency (*idf*) and measure the distribution of the query term frequencies within the collection [35]. More specifically, for both *tf* and *idf* we have extracted the average, the variance and the maximum among query terms, thus obtaining:

- **avgIDF**
- **varIDF**
- **maxIDF**
- **avgTF**
- **varTF**
- **maxTF**

4.3.3 Post-Retrieval Features

This category of features are related to search results, thus can be extracted only after the searching process. In particular, we adopted two features related to the ranking score following the retrieval, that are the *Unormalized Query Commitment* (**UQC**) and the *Normalized Query Commitment* (**NQC**). They are very similar, since UQC measures the standard deviation of the retrieval scores on the set of top-k ranked documents and NQC is the normalized version of UQC. So, as previously stated, given a corpus D, the set of relevant documents D_q of the query q, and the top-k ranked documents D_q^k, the UQC is computed as:

$$UQC(q) = \sqrt{\frac{1}{k} \sum_{d \in D_q^k} (Score(d) - \mu)^2}$$

And consequently NQC is:

$$NQC(q) = \frac{UQC}{|Score(D)|}$$

5 Experiments

This chapter contains the experimental evaluations of this thesis. Experiments follows essentially two different tasks, predicting a never seen configuration and predicting a never seen component. This means that in the first task, as we are going to see, the test set of our predictor is composed by IR systems with a configuration (stop list + stemmer + IR model) that does not appear with any sample in the training set. On the other hand, the second task is performed on a test set composed by a never seen IR component. This task is divided in three different test: in the first one the never components are two stop lists, in the second test we deal with a stemmer and in the third and last experiment the never seen components are two different IR models. Each task is repeated for each predictor chosen for this work: Linear Regression, Support Vector Regression, Random Forest Regressor and Gradient Boosting Regressor. Except Linear Regression that does not have parameters, each predictor is trained with a grid of possible hyperparameters, in order to obtain an optimal model. For doing this, we used the **GridSearchCV** python tool, that permits to select the best set of parameters and preventing overfitting thanks to an integrated Cross Validation mechanism during the training phase. For each predictor we are going to analyse query performance prediction results, i.e. prediction made on each of the 50 topics for each IR system configuration. Scores are presented considering the differentiation between training and test set and each set is described by mul-

tiple scores, improving the accuracy of the study. In the first task, scores are presented for each of the seven available target measures (AP, nDCG, recall, rbp, p@10, err, rr), instead the second task is focused only on AP. Then, an important aspect to analyse, is the Kendall's tau ranking coefficient for the Mean Average Precision(MAP) of each predicted system. All those evaluations are made on different feature subset in order to inspect the pros and cons of each subset and the task in which they obtain the best results.

5.1 First task

The first task is to predict performance of a never seen configuration. As previously mentioned, our dataset provides 350 different IR search engine configurations, considering all the combinations of stop lists, stemmers and IR models involved. In this experiment, the test set is composed by 105 randomly chosen IR systems, while the remaining configurations are used for training. Since each configuration has 50 samples (one for each topic), the total amount of data for training is $245 \times 50 = 12'250$ and for the test set $105 \times 50 = 5'250$.

5.1.1 Linear Regression

First of all we present our prediction results on the linear regression. Since this predictor does not need hyperparameter tuning, we start with the prediction task.

Average Precision (AP) is the first target measure we predict since by using MAP we can obtain a reliable goodness measure for IR systems. Table 5.1 reports training and test scores related to AP predictions. First it is worth to notice that Linear Regression does not provide good results on sample prediction and GoP features (IR system configuration and component properties) does not have a strong influence on prediction quality. Best results are reached when the predictor uses *query features* and most of all *post retrieval* features. In fact, it is possible to notice that predictions on post retrieval features reach a R2 test score slightly worse than the prediction made on the whole feature set (0.42268 vs 0.48179) and this statement is supported by the comparison of MSE, RMSE and MAE test scores too. The difference with respect to the whole feature set is further reduced if we merge post retrieval features with query related features obtaining a R2 test score equal to 0.45298.

	Training		Test			
	R2 score	**MSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.48370**	**0.02101**	**0.48179**	**0.02110**	**0.14525**	**0.10852**
IR system conf	0.09255	0.03694	0.09282	0.03694	0.19219	0.14621
Component prop	0.07305	0.037707	0.07224	0.03786	0.19457	0.14819
IR system conf + Component prop	0.09232	0.03695	0.09251	0.03695	0.19222	0.14623
Query features	0.24896	0.03055	0.24860	0.03066	0.17509	0.13165
Post retrieval	0.42456	0.02346	0.42268	0.02344	0.15310	0.11649
Query features + Post retrieval	0.45522	0.02217	0.45298	0.02228	0.14926	0.11150

Table 5.1: Training and test scores for each feature subset on AP predictions with Linear Regression.

Since *post retrieval features* and *query features* are strictly related to the query (i.e. the topic) rather than the IR system configuration, and since the GoP features do not provide good results on AP, it is evident that topic has the main weight on such predictions. However the best prediction scores are obtained using GoP features too, slightly improving the scores of the *query features + post retrieval features* subset, showing that IR systems features can give a little contribution.

The most interesting results are reached involving MAP. This measure is computed averaging the 50 AP predictions for each IR configuration, thus obtaining the MAP related to each of the 105 predicted IR systems. This vector, containing 105 MAP measures, is compared to the real MAP of the test set IR systems leading to interesting results reported in figure 5.1.

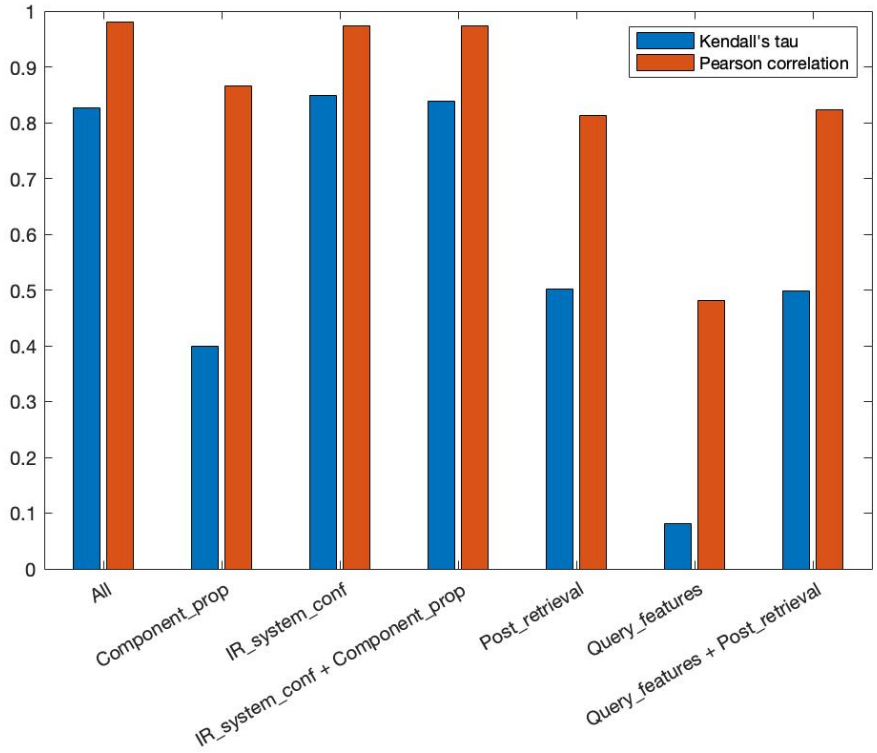

Figure 5.1: Kendall's tau and Pearson correlation of MAP related to different feature combinations with Linear Regression.

In this case it is evident that all the sets containing features related to the IR system configurations leads to better results. Pearson correlation, in fact, is worse for *post retrieval features*, *query features* and the combination of them. Moreover, Kendall's tau is sensibly better for *IR system configuration* and *IR system configuration + component properties* and for the whole feature set. Curious is the case of the *component properties* feature set, where there is a bad Kendall's tau but a good Pearson correlation; this is probably due to the fact that the predicted MAP are quite accurate but too close to each other leading to some misplacements and thus to bad ranking (i.e. a bad Kendall's tau coefficient).
It is worth noticing that the feature sets that give the worst results for AP predictions (R2 close to zero), are the best in terms of MAP prediction and thus for correlation coefficients. We first had a clue of this behaviour by inspecting the parallel plot obtained picking randomly three of the 105 predicted systems (this

number is chosen only for figure clarity) and comparing their predicted AP to the real AP. In figure 5.2 it is possible to see the relation between the actual AP value (right column) and the predicted AP value (left column). It is evident that the predicted values converges in three different points, showing that for each IR system the algorithms predict the same value for each topic, that is, for each system, a good approximation of its MAP.

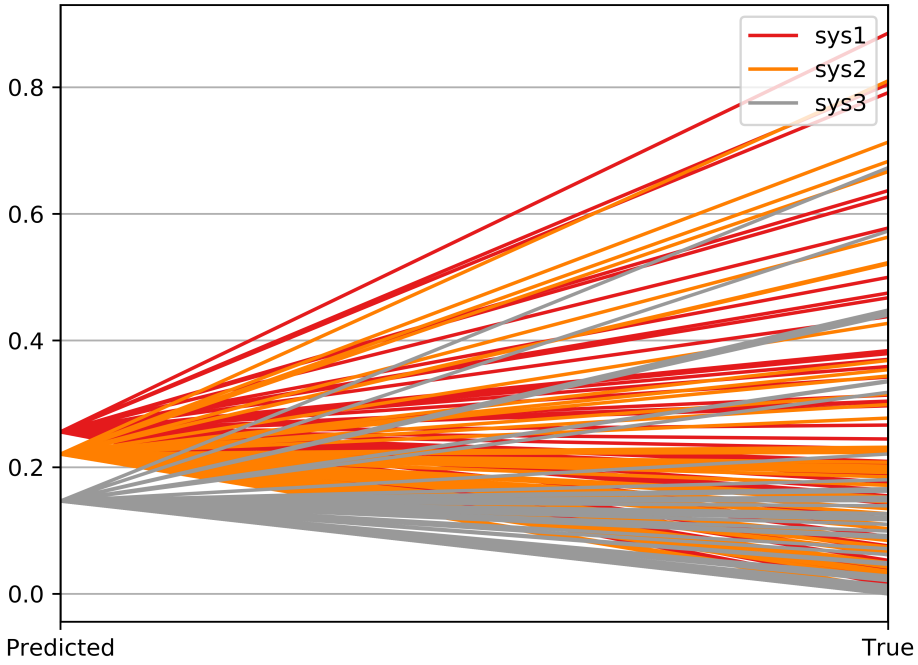

Figure 5.2: Relation between predicted AP (on the left) and true AP (on the right) with Linear Regression.

For sake of completeness, we report also the prediction scores on **nDCG**, **recall**, **rbp**, **p@10**, **err** and **rr** (tables 5.2, 5.3, 5.4, 5.5, 5.6, 5.7). The overall results using linear regression are not satisfactory. **nDCG** is the measure with better prediction and scores quite close to the ones obtained by AP, while **RR** is the one that performs worst. Anyway, in general, it is possible to see a behaviour similar to one seen for **AP**: the best predictions are when the feature set contains information strictly related with the topic (query features and post retrieval

features).

	Training		Test			
	R2 score	**MSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.39836**	**0.03633**	**0.39726**	**0.03640**	**0.19080**	**0.15948**
IR system conf	0.12879	0.05262	0.12797	0.05267	0.22951	0.19062
Component prop	0.09910	0.05433	0.10365	0.05435	0.23313	0.19506
IR system conf + Component prop	0.12694	0.05265	0.13048	0.05272	0.22962	0.19081
Query features	0.19912	0.04830	0.19921	0.04853	0.22030	0.18564
Post retrieval	0.30626	0.04184	0.30658	0.04202	0.20499	0.17196
Query features + Post retrieval	0.35133	0.03920	0.34467	0.03953	0.19882	0.16608

Table 5.2: Training and test scores for each feature subset on **nDCG** predictions with Linear Regression.

	Training		Test			
	R2 score	**MSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.32301**	**0.05341**	**0.31923**	**0.05361**	**0.23154**	**0.19389**
IR system conf	0.10092	0.07094	0.09902	0.07095	0.26637	0.22065
Component prop	0.07820	0.07276	0.07293	0.07295	0.27009	0.22561
IR system conf + Component prop	0.10252	0.07084	0.09745	0.07102	0.26649	0.22092
Query features	0.17943	0.06469	0.17675	0.06496	0.25487	0.21740
Post retrieval	0.22583	0.06103	0.22614	0.06106	0.24710	0.20665
Query features + Post retrieval	0.27769	0.05694	0.27897	0.05688	0.23850	0.20097

Table 5.3: Training and test scores for each feature subset on **recall** predictions with Linear Regression.

	Training		Test			
	R2 score	**MSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.32019**	**0.06075**	**0.31729**	**0.06108**	**0.24715**	**0.20036**
IR system conf	0.12398	0.07829	0.12202	0.07855	0.28027	0.23468
Component prop	0.10531	0.07997	0.10675	0.07989	0.28264	0.23642
IR system conf + Component prop	0.12324	0.07837	0.12401	0.07834	0.27990	0.23425
Query features	0.15790	0.07531	0.15130	0.07582	0.27534	0.22693
Post retrieval	0.22015	0.06974	0.21812	0.06984	0.26428	0.21769
Query features + Post retrieval	0.27446	0.06492	0.26629	0.06546	0.25584	0.20909

Table 5.4: Training and test scores for each feature subset on **rbp** predictions with Linear Regression.

	Training		Test			
	R2 score	**MSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.29851**	**0.06081**	**0.29759**	**0.06075**	**0.24646**	**0.19389**
IR system conf	0.11757	0.07650	0.11824	0.07626	0.27614	0.22721
Component prop	0.10233	0.07775	0.09947	0.07801	0.27931	0.23033
IR system conf + Component prop	0.11919	0.07629	0.11469	0.07670	0.27694	0.22810
Query features	0.14282	0.07427	0.14071	0.07441	0.27277	0.22287
Post retrieval	0.22583	0.06103	0.22614	0.06106	0.24710	0.20665
Query features + Post retrieval	0.20204	0.06916	0.19955	0.06924	0.26313	0.21465

Table 5.5: Training and test scores for each feature subset on **P@10** predictions with Linear Regression.

	Training		Test			
	R2 score	**MSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.28519**	**0.04708**	**0.28143**	**0.04728**	**0.21744**	**0.18826**
IR system conf	0.12392	0.05771	0.12083	0.05785	0.24051	0.21826
Component prop	0.10794	0.05874	0.10826	0.05872	0.24232	0.22012
IR system conf + Component prop	0.12335	0.05773	0.12313	0.05774	0.24029	0.21801
Query features	0.13883	0.05667	0.13491	0.05701	0.23878	0.21126
Post retrieval	0.19336	0.05308	0.19204	0.05324	0.23075	0.20439
Query features + Post retrieval	0.23497	0.05040	0.22978	0.05062	0.22500	0.19745

Table 5.6: Training and test scores for each feature subset on **ERR** predictions with Linear Regression.

	Training		Test			
	R2 score	**MSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.25151**	**0.11744**	**0.24435**	**0.11817**	**0.34376**	**0.30111**
IR system conf	0.11492	0.13887	0.10816	0.13947	0.37345	0.34393
Component prop	0.10033	0.14094	0.09941	0.14126	0.37584	0.34637
IR system conf + Component prop	0.11246	0.13904	0.11144	0.13937	0.37333	0.34283
Query features	0.11871	0.13832	0.11418	0.13839	0.37200	0.33292
Post retrieval	0.16902	0.13026	0.16937	0.13014	0.36074	0.32204
Query features + Post retrieval	0.20295	0.12494	0.20062	0.12524	0.35389	0.31340

Table 5.7: Training and test scores for each feature subset on **RR** predictions with Linear Regression.

5.1.2 SVR

In this section we comment results regarding *Support Vector Regression* (SVR). This prediction algorithm is provided by the *scikit-learn*[1] tool for python. For our task we use the *rbf* kernel; the library permits to use also other kernels such as the polynomial or the sigmoid, but, after some tests, we have decided to opt for rbf, and for prediction goodness and time performances. Another important parameter is the regularization parameter **C**. This determines the strength of regularization and it is inversely proportional to the margin: so, the smaller C the larger margin, the larger C the smaller margin. There not exists a rule of thumb to set properly C since it mostly depends on the dataset. So we have executed some tests to choose some potential good values. We saw that most of the better results came from a C between 1 and 100, so in our grid of hyperparameters we have inserted $C = \{1, 10, 50, 100, 500\}$. We set 500 as maximum value of C since higher values required too much computational time and does not provide a significant improvement. The last hyperparameter to tune is gamma, that is the kernel coefficient ($K(x, x') = e^{-\gamma||x-x'||^2}$ with $\gamma \leq 1$) and acts as a sort of similarity measure between two points. Also in this case there is not a rule of thumb to properly set gamma. In our hyperparameter grid we inserted the *'auto'* value that is, referring to the scikit-learn documentations, $\frac{1}{\#\text{ features}}$ and the *'scale'* value that is $\frac{1}{\#\text{ features} * \text{X.var()}}$. In addition, we have inserted also values that after some tests leaded to good performance. So, resuming, we have $gamma = \{auto, scale, 1, 0.1, 0.01\}$. The training is performed using the *GridSearchCV* toolkit that chooses the best set of hyperparameters among the ones given and moreover performs the k-fold cross validation (we set cv = 5 performing a 5-fold cross validation). Moreover we used RMSE as scoring function since it is one of the most used evaluation parameters in IR. Resuming, we have trained the SVR with the following parameters:

- $C = \{1, 10, 50, 100, 500\}$;
- $gamma = \{auto, scale, 1, 0.1, 0.01\}$;
- kernel = *rbf*;
- cv = 5;

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

- scoring function = RMSE.

Also for this predictor we start our analysis with AP measure. In table 5.8 is it possible to inspect prediction scores on AP both for training and test on different feature sets. The first result that comes up is that in general SVR works much better than linear regression. The feature set that performs better has a R2 train score of 0.869 and a 0.072 RMSE with respect to a R2 score equal to 0.481 and a RMSE equal to 0.145 for linear regression: a significant improvement. Another difference is that the feature set that achieves the best results it is not the one with all the features (as we have seen for Linear Regression), but the one that comprehends *query features* and *post retrieval* ones. This feature set is followed by the *query feature* set and the whole feature set, which obtain similar performance on the test set. Finally, as before, feature sets related to the IR system configuration do not provide any good result since those features are not related with topic at all. The most interesting result is that the whole feature set does not reach better results with respect to the other sets. In our opinion this happens because categorical variables increase considerably the SVR dimension space and the predictor struggles to handle them.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.79002	0.00850	0.09222	0.78506	0.00885	0.09408	0.07510
IR system conf	0.06413	0.03791	0.19471	0.05080	0.03909	0.19772	0.14270
Component prop	0.04629	0.03888	0.19719	0.06933	0.03777	0.19435	0.13978
IR system conf + Component prop	0.05412	0.03856	0.19638	0.07510	0.03753	0.19374	0.13982
Query features	0.80252	0.00806	0.08979	0.77133	0.00925	0.09622	0.07066
Post retrieval	0.55240	0.01827	0.13518	0.52955	0.01905	0.13802	0.10572
Query features + Post retrieval	**0.87979**	**0.00493**	**0.07026**	**0.86996**	**0.00518**	**0.07201**	**0.05941**

Table 5.8: Training and test scores for each feature subset on **AP** predictions with SVR.

SVR leads to good results also regarding ranking and correlation coefficients of the MAP of predicted IR systems (figure 5.3). Generally, all the scores are better with SVR with respect to linear regression. Kendall's tau for the whole feature set suffers a slight drop (0.77 with respect to 0.82 for linear regression

and a Pearson correlation of 0.96 against 0.98). Anyway, also with SVR there are better results in terms of correlation and ranking if the involved feature set contains the *IR system configuration* and the *component properties*.

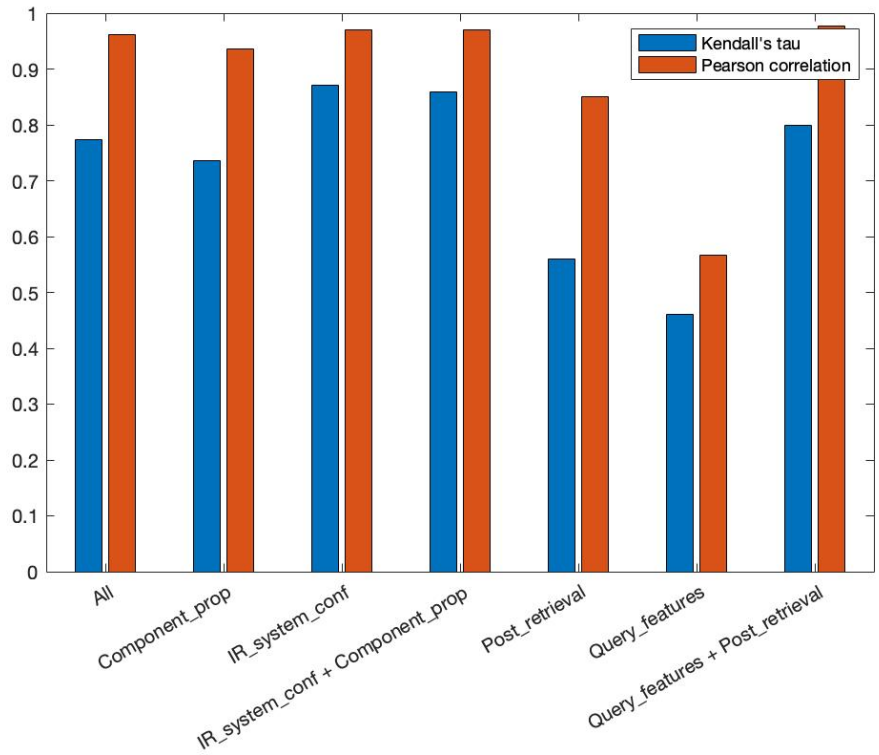

Figure 5.3: Kendall's tau and Pearson correlation of MAP related to different feature combinations with SVR.

Tables 5.9, 5.10, 5.11, 5.12, 5.13, 5.14 report a significant improvement also for other measures, mainly for nDCG, recall and RBP with the *query + post retrieval* feature set. Also the whole feature set and the *query features* bring good results on the test set.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.68782	0.01881	0.13718	0.68253	0.01928	0.13888	0.11165
IR system conf	0.13064	0.05240	0.2289	0.13658	0.05246	0.22904	0.18944
Component prop	0.12561	0.05311	0.23046	0.09454	0.05398	0.23235	0.19388
IR system conf + Component prop	0.13820	0.05235	0.22880	0.11565	0.05272	0.22962	0.19018
Query features	0.80084	0.01203	0.10970	0.78099	0.01323	0.11502	0.08513
Post retrieval	0.39763	0.03640	0.19079	0.39963	0.03626	0.19044	0.15774
Query features + Post retrieval	**0.91906**	**0.00481**	**0.06939**	**0.90000**	**0.00624**	**0.07901**	**0.06496**

Table 5.9: Training and test scores for each feature subset on **nDCG** predictions with SVR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.73216	0.02094	0.14472	0.68762	0.02512	0.15849	0.12163
IR system conf	0.09288	0.07093	0.26634	0.12034	0.07074	0.26597	0.21979
Component prop	0.10669	0.07142	0.26725	0.05184	0.07213	0.26858	0.22316
IR system conf + Component prop	0.11555	0.07071	0.26592	0.06454	0.07117	0.26677	0.22021
Query features	0.79595	0.01614	0.12707	0.78965	0.01646	0.12832	0.09368
Post retrieval	0.30556	0.05495	0.23442	0.30155	0.05467	0.23382	0.18975
Query features + Post retrieval	**0.92935**	**0.00562**	**0.07500**	**0.89748**	**0.00790**	**0.08890**	**0.07062**

Table 5.10: Training and test scores for each feature subset on **recall** predictions with SVR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.73480	0.02287	0.15123	0.68867	0.02725	0.16507	0.12695
IR system conf	0.08581	0.07884	0.28078	0.12469	0.07661	0.27679	0.22129
Component prop	0.07805	0.08011	0.28304	0.08292	0.07877	0.28067	0.22208
IR system conf + Component prop	0.08958	0.07911	0.28126	0.10643	0.07675	0.27705	0.22089
Query features	0.70860	0.02526	0.15893	0.65566	0.02980	0.17264	0.12220
Post retrieval	0.29907	0.06076	0.24650	0.27863	0.06244	0.24988	0.19267
Query features + Post retrieval	**0.88190**	**0.01020**	**0.10102**	**0.83902**	**0.01403**	**0.11847**	**0.08977**

Table 5.12: Training and test scores for each feature subset on **P@10** predictions with SVR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.73535	0.02377	0.15420	0.68860	0.02753	0.16594	0.12703
IR system conf	0.12131	0.07895	0.28098	0.10021	0.07957	0.28208	0.23348
Component prop	0.11374	0.07928	0.28156	0.07289	0.08240	0.28706	0.23967
IR system conf + Component prop	0.12783	0.07802	0.27932	0.08311	0.08149	0.28547	0.23660
Query features	0.72605	0.02436	0.15608	0.69612	0.02752	0.16592	0.12149
Post retrieval	0.30803	0.06153	0.24806	0.34403	0.05942	0.24377	0.19084
Query features + Post retrieval	**0.91958**	**0.00720**	**0.08487**	**0.87389**	**0.01123**	**0.10601**	**0.08118**

Table 5.11: Training and test scores for each feature subset on **RBP** predictions with SVR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.59860	0.06298	0.25097	0.50979	0.07673	0.27700	0.21128
IR system conf	-0.08338	0.17001	0.41232	0.04093	0.15012	0.38746	0.33257
Component prop	-0.14650	0.18056	0.42492	-0.14563	0.17779	0.42166	0.32664
IR system conf + Component prop	-0.07715	0.16964	0.41187	0.04324	0.14848	0.38533	0.33368
Query features	0.58580	0.06473	0.25443	0.53127	0.07403	0.27210	0.19416
Post retrieval	0.19884	0.12522	0.35386	0.21366	0.12420	0.35242	0.27860
Query features + Post retrieval	**0.85670**	**0.02240**	**0.14968**	**0.76455**	**0.03716**	**0.19277**	**0.13318**

Table 5.14: Training and test scores for each feature subset on **ERR** predictions with SVR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.66299	0.02203	0.14843	0.59655	0.02699	0.16428	0.13032
IR system conf	0.07695	0.06034	0.24565	0.12831	0.05831	0.24148	0.21703
Component prop	0.06398	0.06186	0.24871	0.08758	0.05964	0.24423	0.21137
IR system conf + Component prop	0.07728	0.06098	0.24694	0.12099	0.05746	0.23971	0.21597
Query features	0.62673	0.02453	0.15665	0.61237	0.02565	0.16016	0.12463
Post retrieval	0.26921	0.04804	0.21918	0.27165	0.04820	0.21955	0.17761
Query features + Post retrieval	**0.86050**	**0.00913**	**0.09556**	**0.80620**	**0.01292**	**0.11369**	**0.08939**

Table 5.13: Training and test scores for each feature subset on **ERR** predictions with SVR.

5.1.3 Random Forest Regressor

In this section we treat results obtained with Random Forest Regressor [2]. Since this predictor has various parameters, before proceeding with the experiments it is necessary performing hyperparameter tuning. The parameters tuned are:

- **max_depth:** is the maximum depth allowed for the three expansion;

[2]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

- **min_samples_split:** that is the minimum number of samples required to split an internal node;
- **min_samples_leaf:** refers to the minimum number of samples required to be a leaf node;
- **max_features:** is the number of features to consider when looking for the best split;
- **n_estimators:** that is the number of trees in the forest.

Hyperparameter tuning is performed on the whole feature set: it is the biggest dataset, it has often high performances and it may be particularly sensible to overfitting. Since the choice of hyperparameters is done also for other feature sets, for each parameter we choose a range of values that is a good tradeoff between accurate scores and avoiding overfitting: python GridSearchCV then has the role to choose the best values among those ranges.

The first parameter to tune is *max depth* and it is tested on values in range $\{1, 32\}$. Other parameters are setted on default values, so resuming:

- **max_depth** = $\{1, 32\}$;
- **min_samples_split** = 2;
- **min_samples_leaf** = 2;
- **max_features** = 'auto';
- **n_estimators** = 100.

Figure 5.4 represents training and test R2 scores for all the tested values of *max depth*. As we can see by inspecting R2 score, the model tends to overfit for *max depth* over values around 12. So, a good set of *max depth* values for the final set of hyperparameters can be $[10, 12]$.

Next we need to study the behaviour of *min samples split* parameter. The procedure is the same as before, and max depth is set equal to 10.

- **max_depth** = 10;
- **min_samples_split** = $\{2, 60\}$;

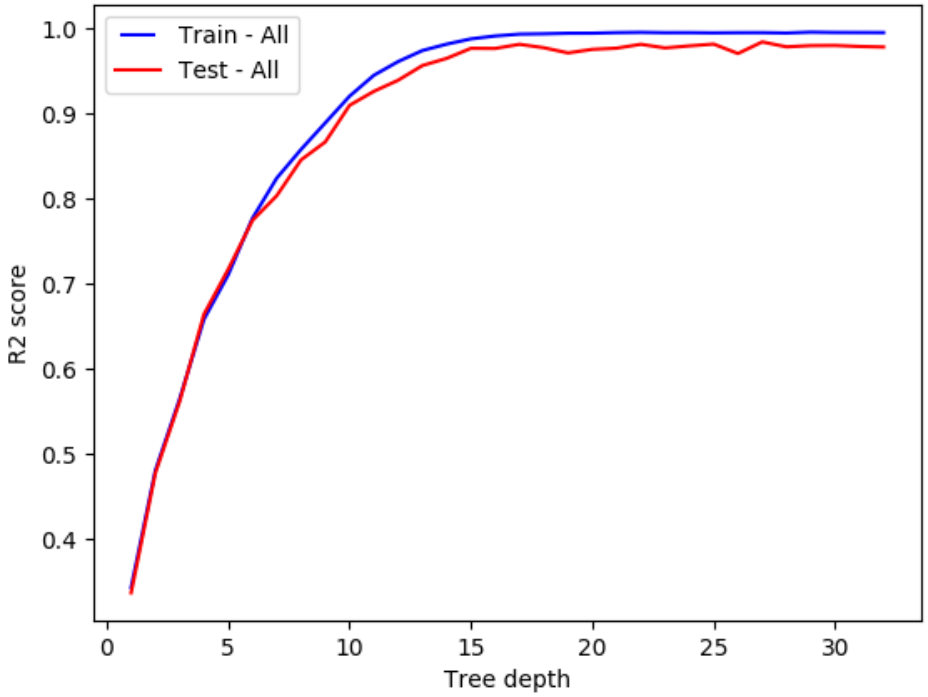

Figure 5.4: Training and test R2 scores for *max_depth* tuning with RF.

- **min_samples_leaf** = 2;
- **max_features** = 'auto';
- **n_estimators** = 100.

Figure 5.5 represents the results of *min samples split* tuning. An evident pattern is that the most the *min samples split* value grows, the worse R2. However, there are some perturbations so a good approach for selecting *min samples split* values is to choose them in the set $[2, 4, 6]$, knowing that better results are likely for lower values.

The next step is to tune *min samples leaf* parameter. *Min samples split* is set equal to 2 for the reason just explained and the parameters of the predictor are:

- **max_depth** = 10;
- **min_samples_split** = 2;

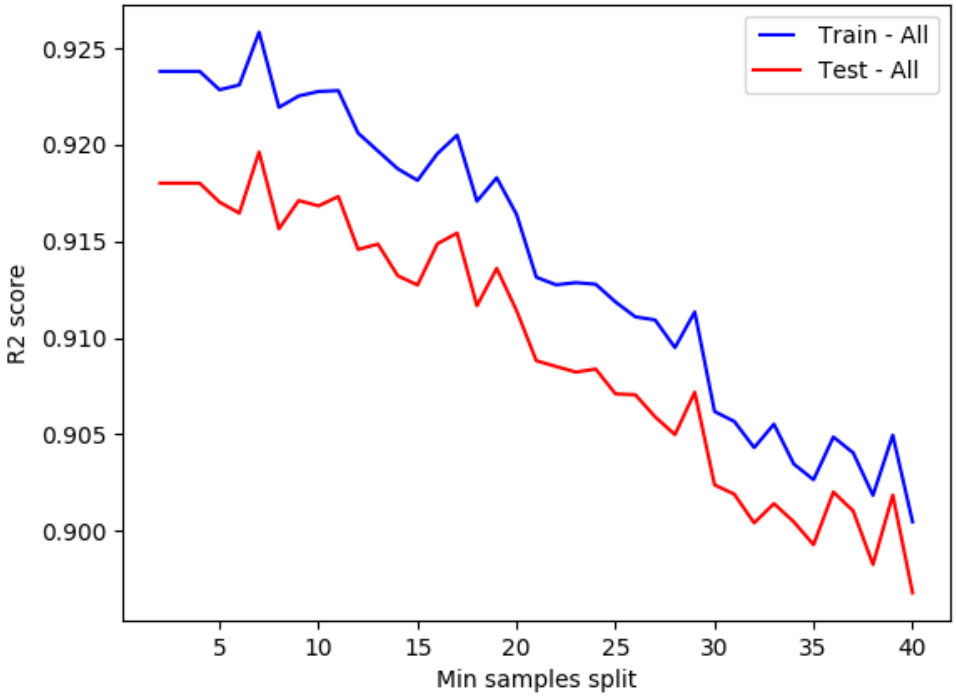

Figure 5.5: Training and test R2 scores for *min_samples_split* tuning with RF.

- **min_samples_leaf** = $\{2, 60\}$;
- **max_features** = 'auto';
- **n_estimators** = 100.

Figure 5.6 explains clearly the behaviour of *min samples leaf* parameter: R2 gets worse if the parameter grows. The pattern is more clear than the one observed with *min samples split* since the score gets sensibly worse for higher values. So *min samples leaf* parameter in the final hyperparameter grid will be [2], since higher values may rarely lead to better results.

The following parameter to tune is *max feature*. Since this parameter is related to the number of features involved, it is better to use a range of float values that stands for the the portion of the total number of features used. *Min_samples_leaf*, the last parameter tuned, is setted equal to 2 and so we have:

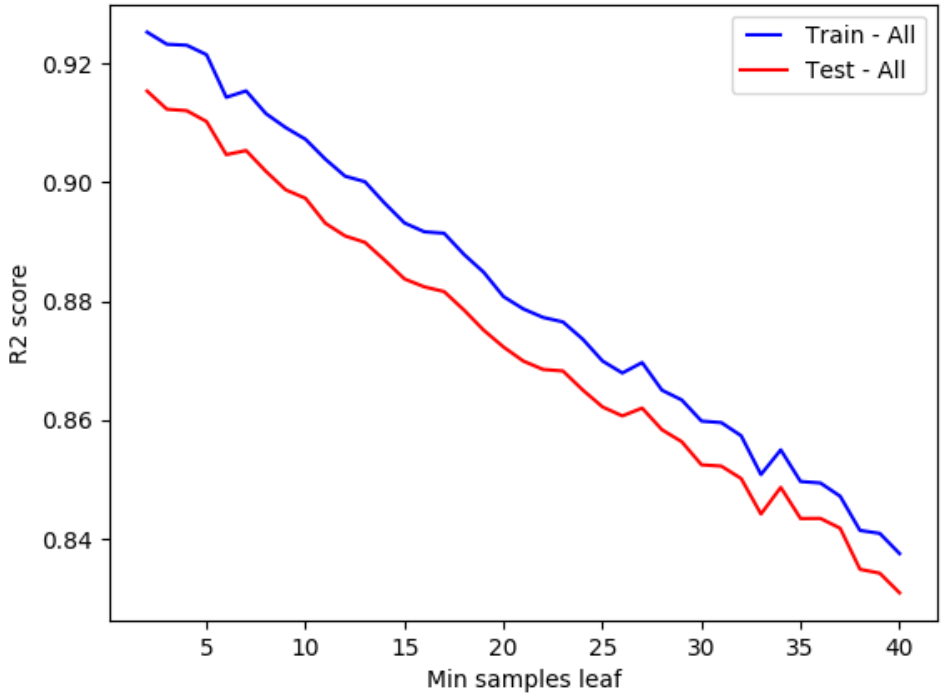

Figure 5.6: Training and test R2 scores for *min_samples_leaf* tuning with RF.

- **max_depth** = 10;
- **min_samples_split** = 2;
- **min_samples_leaf** = 2;
- **max_features** = $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$;
- **n_estimators** = 100.

From figure 5.7 is possible so see that values higher than 0.3 and lower than 0.9 lead to good results, while other values cause a significant drop of the R2 score. Since the model is going to be used with various set of features but results are quite similar in the range $\{0.3, 0.9\}$, the final hyperparameter grid will include the values $[0.4, 0.6, 0.8]$ in order to best fit each set of features.

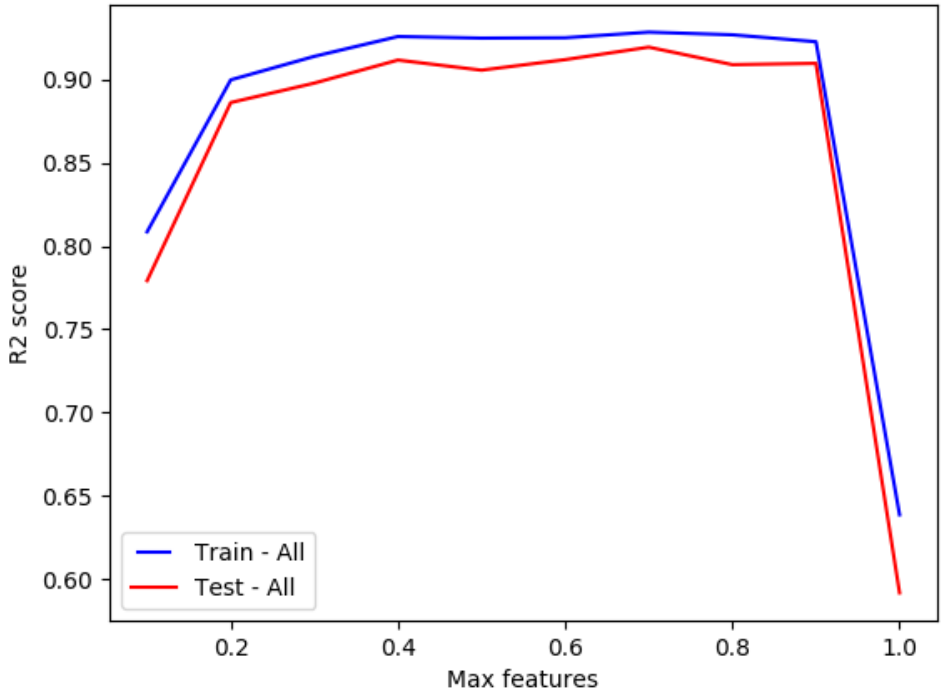

Figure 5.7: Training and test R2 scores for *max_features* tuning with RF.

N estimators determines the number of trees in the forest. This parameter is tuned with the same procedure as before with *max features* set equal to 0.4 obtaining:

- **max_depth** = 10;
- **min_samples_split** = 2;
- **min_samples_leaf** = 2;
- **max_features** = 0.4;
- **n_estimators** = $\{100, 1500\}$ with steps of 100.

Figure 5.8 shows the relation between R2 score and different values of the *n estimators* parameter. The accuracy of the model slightly grows with a larger parameter but, when reaches a value equal to 600 estimators, the R2 score becomes

more stable. For this reason, in the final hyperparameter grid, the *n estimators* parameter will be equal to 700.

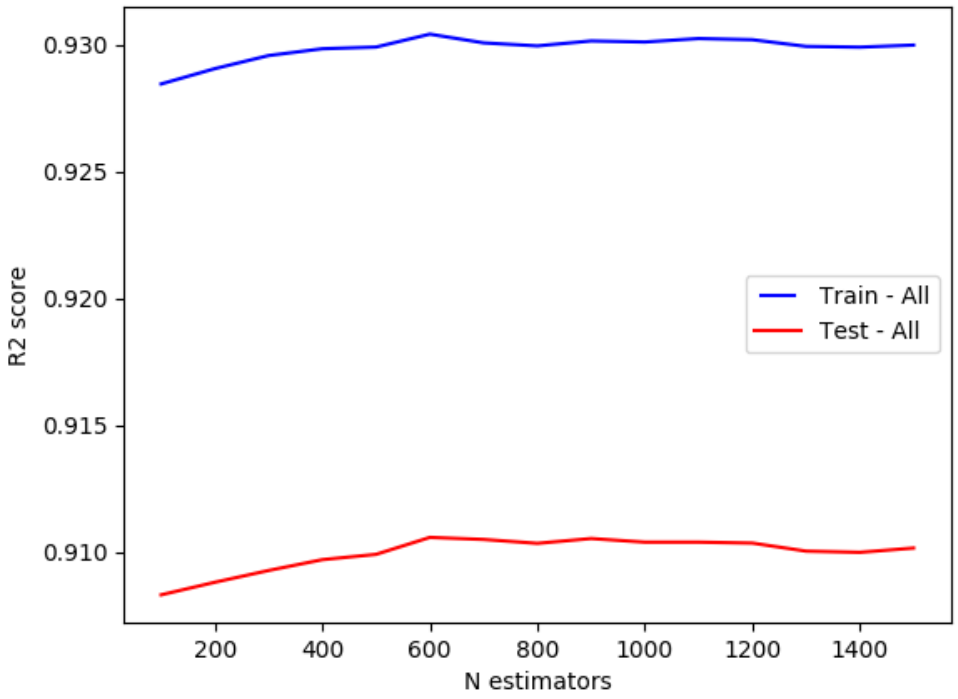

Figure 5.8: Training and test R2 scores for *n_estimators* tuning with RF.

So the final set of hyperparameters used in following tests is:

- **max_depth** = $[10, 12]$;
- **min_samples_split** = 2;
- **min_samples_leaf** = 2;
- **max_features** = $[0.4, 0.6, 0.8]$;
- **n_estimators** = 700;
- **criterion** = MSE.

Please note that in scikit-learn, Random Forest Regressor has only two different quality function to measure the quality of a split in the tree: the Mean Absolute

Error, and the Mean Squared Error. Since MSE is in general more accurate and widely used for IR tasks, we have decided to use it as criterion.

First we use Random Forest to predict AP measure using different sets of features as done with previous predictors. Table 5.15 reports all the obtained results. The whole feature set obtains the best scores, that are slightly better than the *query + post retrieval features* ones. This shows the ability of Random Forests to manage categorical features, permitting to IR system features to give a contribution also for query prediction.

	Training			**Test**			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.96582**	**0.00137**	**0.03708**	**0.95418**	**0.00191**	**0.04374**	**0.02906**
IR system conf	0.09841	0.03629	0.19050	0.09349	0.03785	0.19457	0.14791
Component prop	0.10000	0.03661	0.19134	0.09105	0.03711	0.19265	0.14619
IR system conf + Component prop	0.09999	0.03661	0.19134	0.09152	0.03709	0.19260	0.14610
Query features	0.85373	0.00594	0.07709	0.82670	0.00706	0.08393	0.04953
Post retrieval	0.76524	0.00953	0.09765	0.63447	0.01496	0.12233	0.09185
Query features + Post retrieval	0.96499	0.00141	0.03757	0.94941	0.00210	0.04587	0.03102

Table 5.15: Training and test scores for each feature subset on **AP** predictions with RF.

Random Forests permit to obtain easily - and clearly - the feature importance graph; figure 5.9 represents features importance of the whole feature set. From the image we can first notice that the most influence features are the ones related to the query and post retrieval features. This statement is evident also by inspecting score tables since feature sets with the best query prediction performances, are composed by features with a higher weight. However it is worth noticing that also *NQC_IR_model* and *stem ratio* features have an influential weight, together with some components such as *bool* IR model and *5grams* stemmer. This behaviour is more evident in figure 5.10 where are plotted features importance relative to the *IR system* feature set. There is a clear imbalance of the *bool* and *5grams*

components: they have a major impact on feature weights since systems built on those components tend to have bad performances [26] (in this case in terms of Average Precision) and consequently they are more 'recognizable' by the predictor.

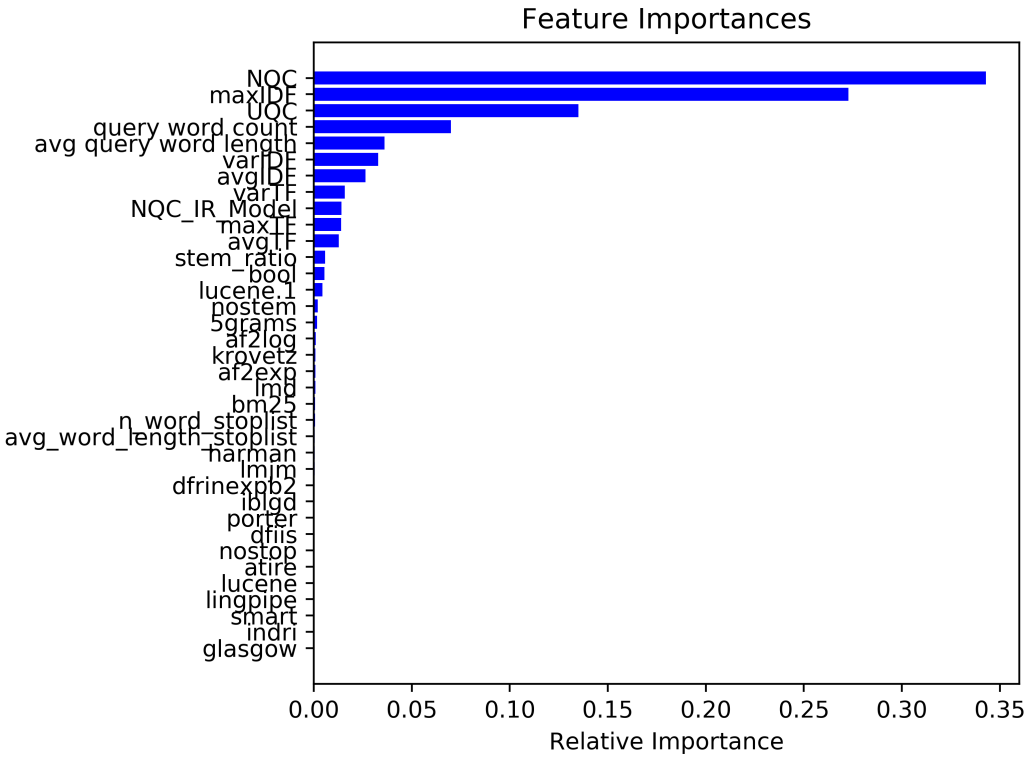

Figure 5.9: Feature importance of the whole feature set with RF.

Random Forest brings also great results for Kendall's Tau and Pearson correlation coefficients. Figure 5.11 reports those results showing an important improvement of Person correlation. Kendall's tau is improved as well, in particular feature sets with features related with IR systems confirm their goodness for ranking tasks, showing the importance of correctly describing the IR system. *Query features + post retrieval* feature set obtains good results (but not as good as the ones of the first four feature sets) although it does not contains any feature related to the IR system: this phenomena is due to the fact that this feature set has excellent scores for single query prediction, thus the average of such prediction gets closer to the real MAP.

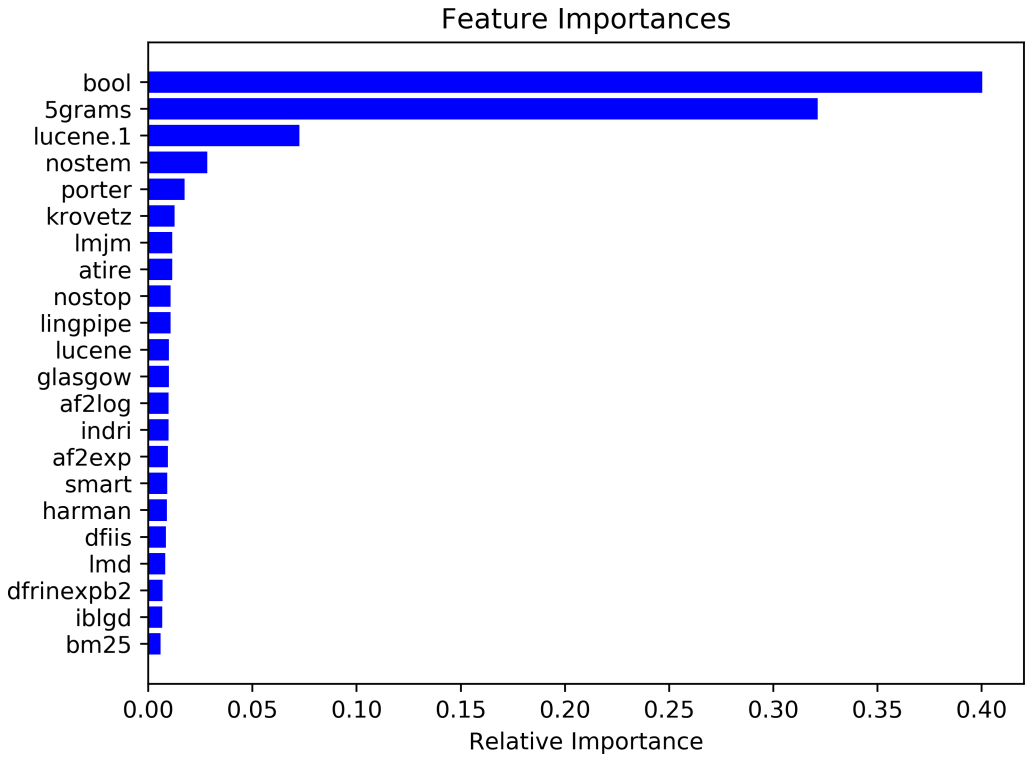

Figure 5.10: Feature importance of the *IR configuration* feature set with RF.

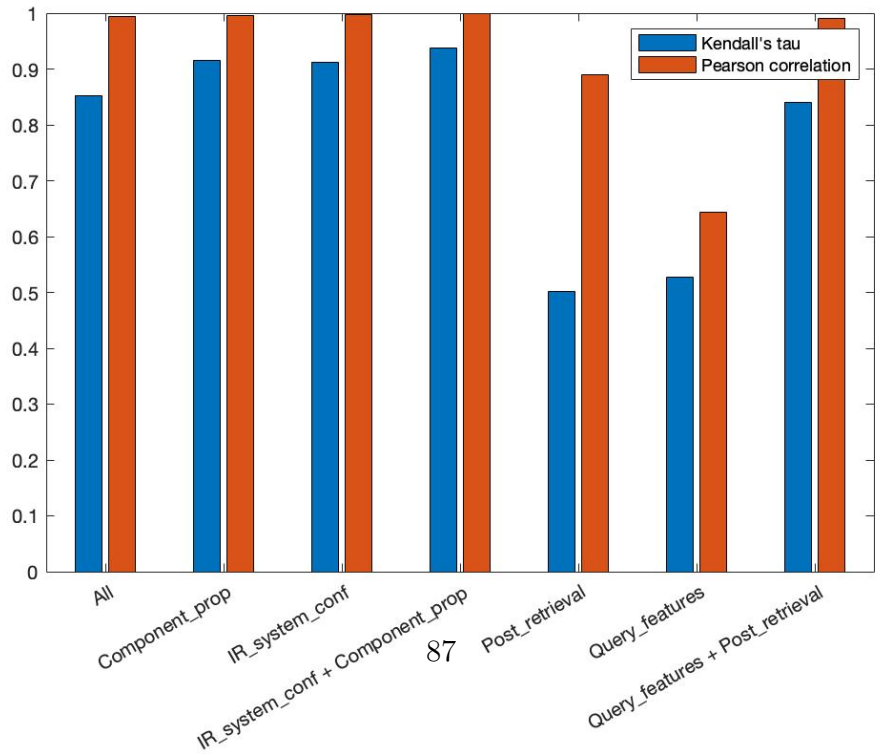

Figure 5.11: Kendall's tau and Pearson correlation of MAP related to different feature combinations with RF.

Tables 5.16, 5.17, 5.18, 5.19, 5.20, 5.21 report results of Random Forest prediction on nDCG, recall, RBP, P@10, ERR and RR scores. With respect to SVR we can notice an improvement for the other measures too. In particular, the most significant improvement with Random Forest is for the whole feature set and the *post retrieval* feature set. Anyway, we can notice also slight better performances also for *query features* and *query + post retrieval* feature set.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.94415**	**0.00337**	**0.05810**	**0.92437**	**0.00456**	**0.06754**	**0.04687**
IR system conf	0.13406	0.05234	0.22879	0.13669	0.05208	0.22821	0.18862
Component prop	0.13431	0.05243	0.22898	0.13660	0.05187	0.22776	0.18842
IR system conf + Component prop	0.13431	0.05243	0.22898	0.13677	0.05186	0.22774	0.18845
Query features	0.83723	0.00991	0.09958	0.83032	0.01004	0.10024	0.06848
Post retrieval	0.65051	0.02129	0.14593	0.49796	0.02973	0.17244	0.13845
Query features + Post retrieval	0.93416	0.00397	0.06305	0.91479	0.00515	0.07179	0.04902

Table 5.16: Training and test scores for each feature subset on **nDCG** predictions with RF.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.94877**	**0.00408**	**0.06389**	**0.92424**	**0.00581**	**0.07622**	**0.05242**
IR system conf	0.12283	0.06991	0.26442	0.06561	0.07167	0.26772	0.22187
Component prop	0.11589	0.07036	0.26526	0.08351	0.07062	0.26575	0.22059
IR system conf + Component prop	0.11590	0.07036	0.26526	0.08370	0.07060	0.26572	0.22053
Query features	0.84335	0.01239	0.11135	0.81609	0.01437	0.11990	0.08203
Post retrieval	0.56545	0.03439	0.18546	0.44501	0.04338	0.20830	0.16700
Query features + Post retrieval	0.93784	0.00486	0.06973	0.90033	0.00801	0.08950	0.06042

Table 5.17: Training and test scores for each feature subset on **recall** predictions with RF.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.89306**	**0.00923**	**0.09609**	**0.87205**	**0.01117**	**0.10573**	**0.07349**
IR system conf	0.12274	0.07575	0.27523	0.13900	0.07523	0.27428	0.22536
Component prop	0.13259	0.07546	0.27471	0.11570	0.07592	0.27554	0.22736
IR system conf + Component prop	0.13264	0.07546	0.27470	0.11607	0.07589	0.27549	0.22711
Query features	0.73647	0.02287	0.15123	0.74010	0.02243	0.14978	0.10844
Post retrieval	0.59977	0.03473	0.18637	0.43782	0.04852	0.22029	0.17201
Query features + Post retrieval	0.88885	0.00963	0.09818	0.85000	0.01295	0.11383	0.08145

Table 5.19: Training and test scores for each feature subset on **P@10** predictions with RF.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.89998**	**0.00898**	**0.09476**	**0.85860**	**0.01252**	**0.11191**	**0.07850**
IR system conf	0.13304	0.07785	0.27902	0.12957	0.07710	0.27768	0.23254
Component prop	0.13680	0.07729	0.27801	0.12129	0.07839	0.27999	0.23506
IR system conf + Component prop	0.13680	0.07729	0.27801	0.12153	0.07837	0.27995	0.23486
Query features	0.76229	0.02134	0.14611	0.72400	0.02444	0.15634	0.11111
Post retrieval	0.59868	0.03604	0.18985	0.44002	0.04959	0.04959	0.17554
Query features + Post retrieval	0.89769	0.00921	0.09599	0.84224	0.01386	0.11773	0.08400

Table 5.18: Training and test scores for each feature subset on **RBP** predictions with RF.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.86025**	**0.00918**	**0.09583**	**0.80686**	**0.01278**	**0.11308**	**0.08303**
IR system conf	0.13274	0.05700	0.23874	0.12502	0.05792	0.24068	0.21773
Component prop	0.12277	0.05715	0.23906	0.14244	0.05767	0.24016	0.21671
IR system conf + Component prop	0.12272	0.05715	0.23907	0.14314	0.05763	0.24006	0.21605
Query features	0.67970	0.02110	0.14528	0.64145	0.02359	0.15361	0.11111
Post retrieval	0.55251	0.02948	0.17172	0.40574	0.03911	0.19776	0.16028
Query features + Post retrieval	0.84380	0.01038	0.10189	0.78349	0.01394	0.11810	0.08817

Table 5.20: Training and test scores for each feature subset on **ERR** predictions with RF.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.83605**	**0.02581**	**0.16065**	**0.77063**	**0.03562**	**0.18875**	**0.13593**
IR system conf	0.12455	0.13782	0.37125	0.10984	0.13826	0.37184	0.34188
Component prop	0.11836	0.13760	0.37094	0.12509	0.13861	0.37230	0.34203
IR system conf + Component prop	0.11833	0.13760	0.37095	0.12520	0.13859	0.37228	0.34117
Query features	0.64339	0.05602	0.23669	0.57463	0.06635	0.25760	0.18429
Post retrieval	0.52403	0.07477	0.27344	0.38753	0.09554	0.30910	0.24937
Query features + Post retrieval	0.81888	0.02836	0.16840	0.74164	0.04063	0.20157	0.14911

Table 5.21: Training and test scores for each feature subset on **RR** predictions with RF.

5.1.4 Gradient Boosting Regressor

Gradient Boosting Regressor (GBR) is a powerful machine learning technique. The implementation of this predictor is done through the *scikit-learn*[3] python tool. The model is built on various parameters so, in order to obtain accurate

[3]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

results, we need to perform hyperparameter tuning. The parameters involved in this process are:

- **max_depth:** maximum depth of the individual regression estimators;
- **min_samples_split:** that is the minimum number of samples required to split an internal node;
- **min_samples_leaf:** refers to the minimum number of samples required to be a leaf node;
- **max_features:** is the number of features to consider when looking for the best split;
- **n_estimators:** number of boosting stages to perform;
- **learning_rate:** learning rate shrinks the contribution of each tree by a certain factor;
- **subsample:** the fraction of samples to be used for fitting the individual base learners.

Regarding the criterion used for measuring the quality of a split, we choose *"friedman_mse"*. GBR supports also MSE and MAE but the documentation suggests to use *"friedman_mse"* since it is generally the best as it can provide a better approximation in some cases. The first parameter to tune is *max_depth*. This parameter is tested in the range $\{1, 32\}$ while other parameters are fixed on their default values. So, overall, we have:

- **max_depth** = $\{1, 32\}$;
- **min_samples_split** = 2;
- **min_samples_leaf** = 2;
- **max_features** = 'auto';
- **n_estimators** = 100;
- **learning_rate** = 0.1;
- **subsample** = 1.

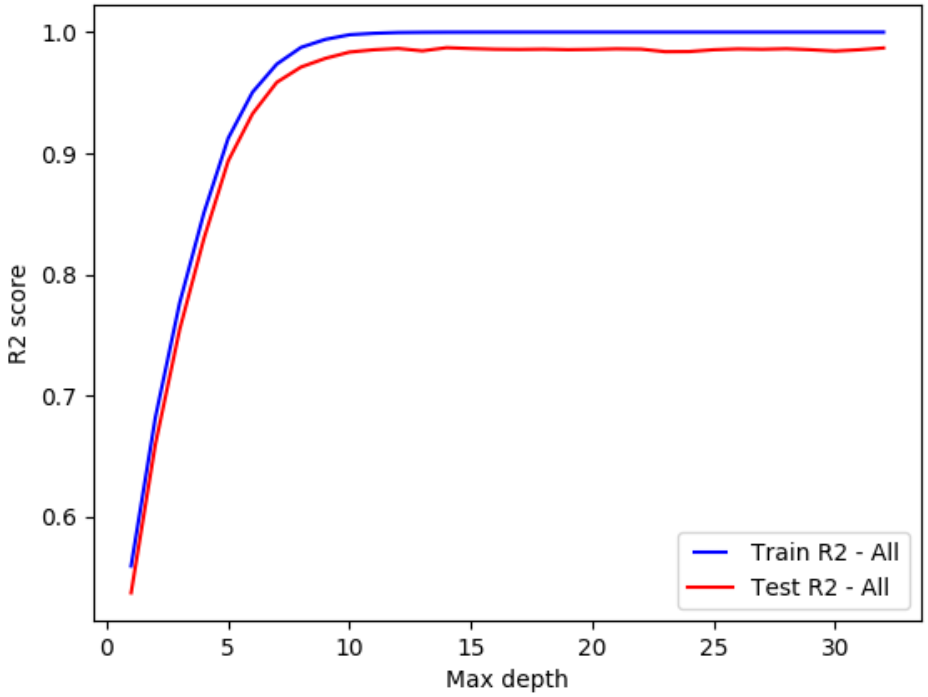

Figure 5.12: Training and test R2 scores for *max_depth* tuning with GBR.

The result, shown in figure 5.12, is that the model tends to saturate when *max_depth* is grater than 7/8. So, for the final parameter grid, a good choice for setting *max_depth* is [6].

Next we deal the *min_samples_split* parameter. It is tested on the range $\{2, 64\}$ with steps of 2. So the parameters for *min_samples_split* tuning are:

- **max_depth** = [6];
- **min_samples_split** = $\{2, 64\}$ with step of 2;
- **min_samples_leaf** = 2;
- **max_features** = 'auto';
- **n_estimators** = 100;
- **learning_rate** = 0.1;

- **subsample** = 1.

The relation between R2 training and test scores and the parameter *min_samples_split* is represented in figure 5.13. Although some perturbations, the score seems to get worse for higher values of *min_samples_split*. So, in order to capture the best model, we are going to use for the final grid values in the range $\{2, 20\}$ with steps of 2.

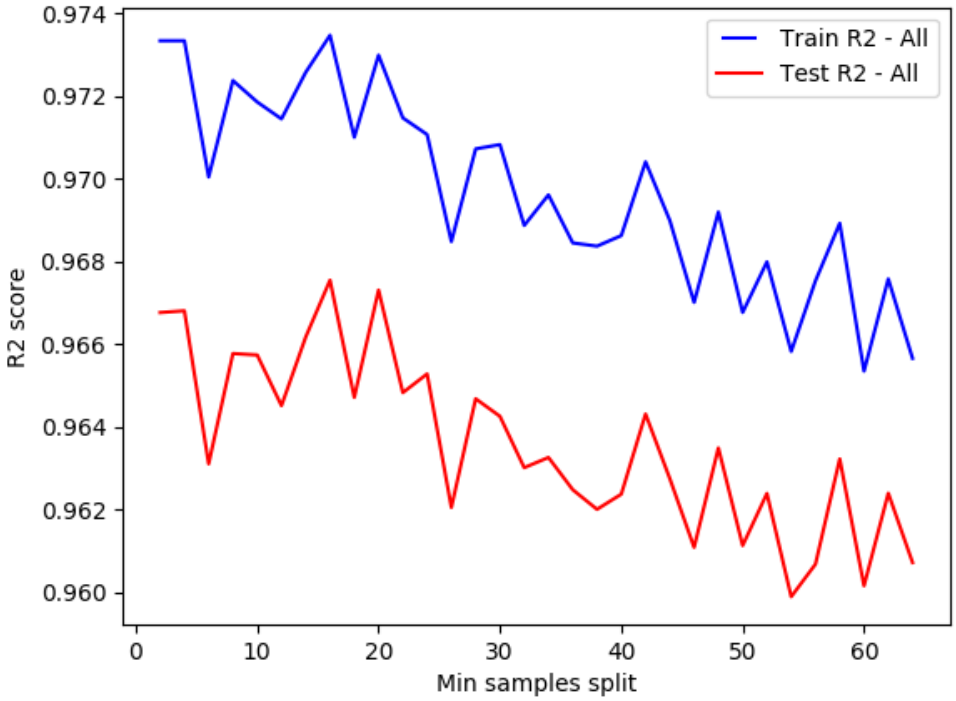

Figure 5.13: Training and test R2 scores for *min_samples_split* tuning with GBR.

Next, it is necessary to inspect the behaviour of *min_samples_leaf*. The model is run with usual parameters and setting *min_samples_split* equal to 2:

- **max_depth** = [6];
- **min_samples_split** = [2];
- **min_samples_leaf** = $\{2, 64\}$ with step of 2;

- **max_features =** 'auto';
- **n_estimators =** 100;
- **learning_rate =** 0.1;
- **subsample =** 1.

The behaviour of *min_samples_leaf* is more marked, as we have seen for Random Forest. The scores get clearly worse for higher values of *min_samples_leaf* (figure 5.14). For this reason we retain that [2] is the optimal value for *min_samples_leaf*, since using more values may be useless in terms of performances and may increase the computational time required to fit the predictor.

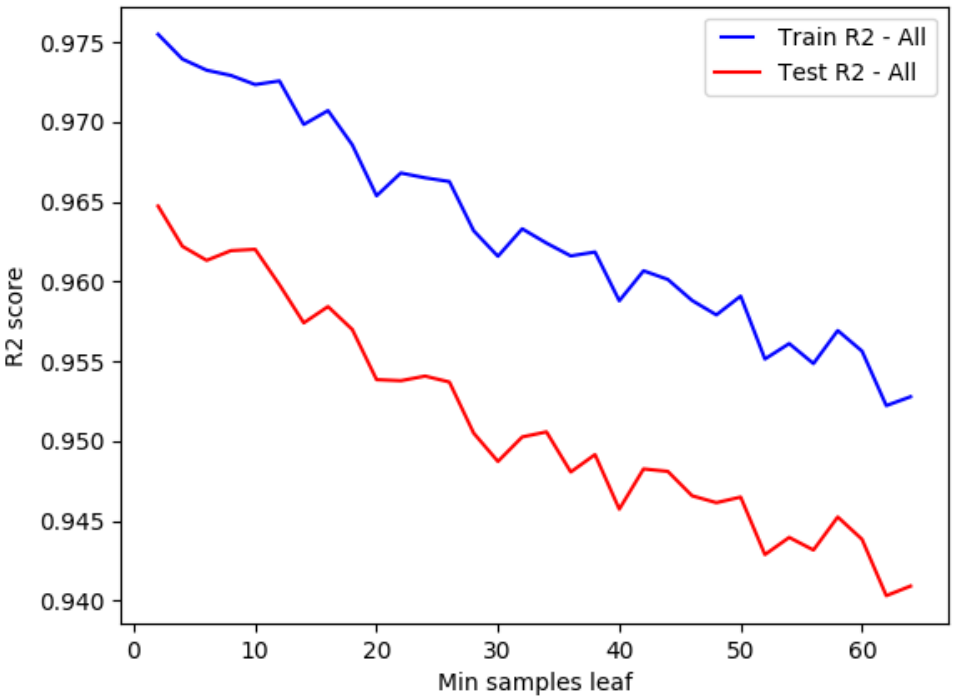

Figure 5.14: Training and test R2 scores for *min_samples_leaf* tuning with GBR.

Max_features is an important parameter for the task of this thesis since the predictor is used with different number of features. For this reason the parameter

is tuned using float values which represent a portion of the total number of input features, making this parameter suitable for different feature set. The parameters used to tune *max_features* are:

- **max_depth =** [6];
- **min_samples_split =** [2];
- **min_samples_leaf =** [2];
- **max_features =** $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$;
- **n_estimators =** 100;
- **learning_rate =** 0.1;
- **subsample =** 1.

Figure 5.15 represent the result of *max_features* for Gradient Boosting Regressor. The result is very similar to the one obtained for Random Forest Regressor. The best scores are reached with *max_features* values between 0.4 and 0.9: in this range scores are very similar each other, while outside this range there is an evident drop of performance. For the final parameter grid we are going to use values in the set $[0.4, 0.6, 0.8]$

N_estimators refers to the number of boosting stages to perform. This parameter is strictly related to the dataset, so the tuning is performed on a wide range:

- **max_depth =** [6];
- **min_samples_split =** [2];
- **min_samples_leaf =** [2];
- **max_features =** [0.4];
- **n_estimators =** $\{50, 1000\}$ with steps of 50;
- **learning_rate =** 0.1;
- **subsample =** 1.

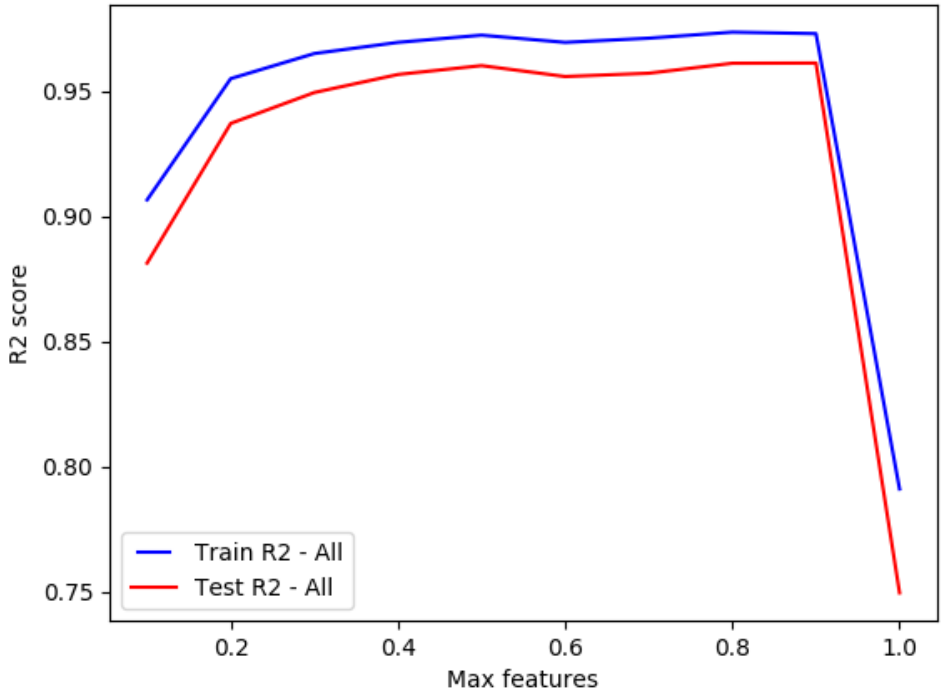

Figure 5.15: Training and test R2 scores for *max_features* tuning with GBR.

In figure 5.16 we can see that R2 train score tends to stabilize on a value around 0.99 when *n_estimators* is larger than 200. In order to prevent overfitting, we are going to use *n_estimators* equal to 150 for the following tests, since we retain this value a good tradeoff for preventing overfitting and fitting the model faster.

Next step is to tune *learning_rate* parameter, and it is performed with the following values:

- **max_depth** = [6];
- **min_samples_split** = [2];
- **min_samples_leaf** = [2];
- **max_features** = [0.4];
- **n_estimators** = [150];
- **learning_rate** = $[0.01, 0.05, 0.1, 0.15, 0.2, 0.3]$;

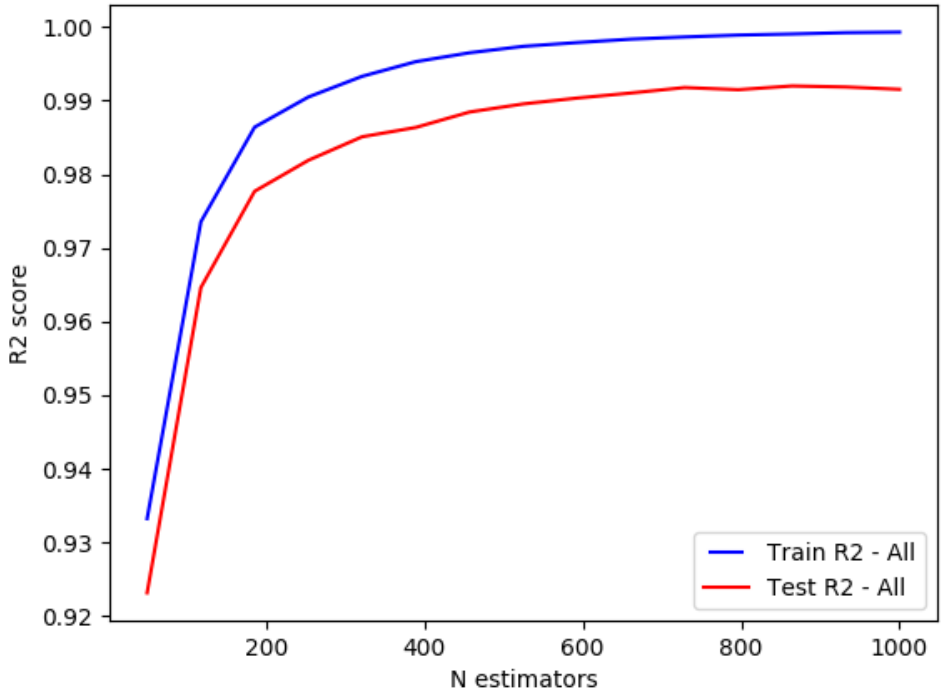

Figure 5.16: Training and test R2 scores for *n_estimators* tuning with GBR.

- **subsample** = 1.

Figure 5.17 represents the relation between R2 scores and learning rate. In this case we prefer to use for our tests the default value provided in GBR documentation, so a *learning_rate* equal to 0.1.

Finally we have to tune the *subsample* parameter. This value represents the fraction of samples to be used for fitting the individual base learners so it accepts values less or equal than 1. The parameters are:

- **max_depth** = [6];
- **min_samples_split** = [2];
- **min_samples_leaf** = [2];
- **max_features** = [0.4];

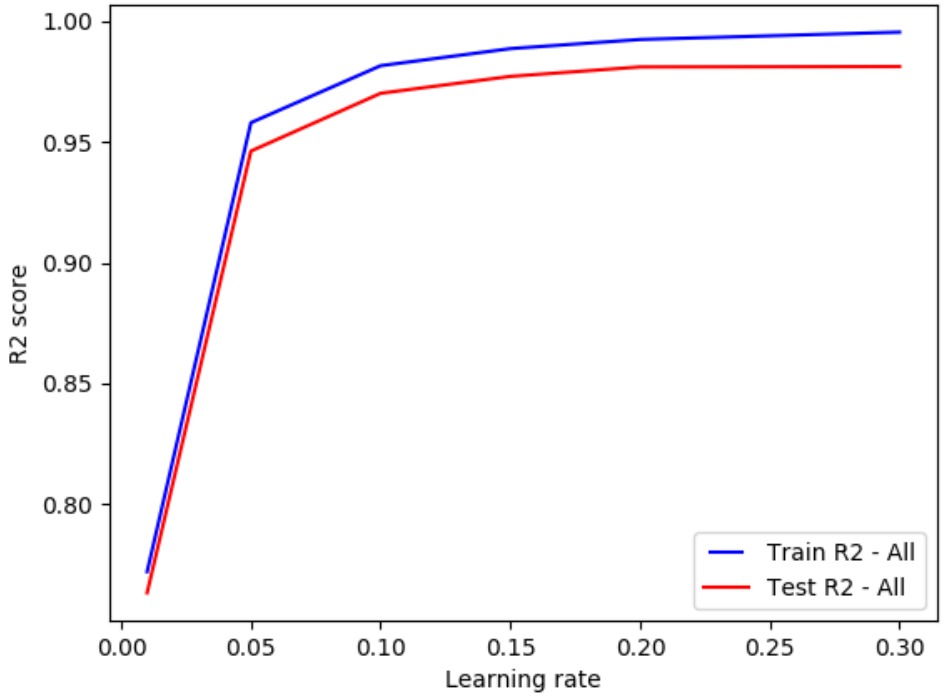

Figure 5.17: Training and test R2 scores for *learning_rate* tuning with GBR.

- **n_estimators** = [150];
- **learning_rate** = [0.1];
- **subsample** = $\{0.4, 1\}$.

In figure 5.18 we can see the results applied on different values of *subsample*. This parameter permits to obtain better results with values in range $\{0.75, 0.85\}$ that is the range of values that we are going to use for our final test.

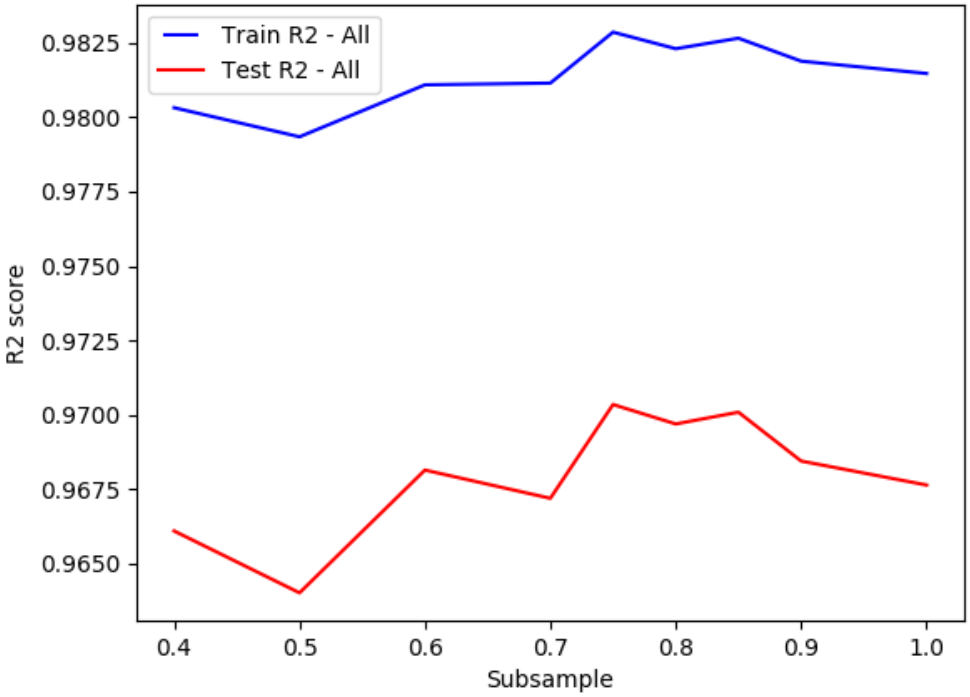

Figure 5.18: Training and test R2 scores for *subsample* tuning with GBR.

We have inspected the behaviour of all the parameters with our dataset, obtaining a grid of values that is going to be used to make predictions on the evaluation measures. For each set of features the algorithm tries every combination of parameters among the given values, in order to obtain the best predictor and thus the best results. The parameters evaluated are:

- **max_depth** = $[6]$;
- **min_samples_split** = $\{2, 20\}$ with steps of 2;
- **min_samples_leaf** = $[2]$;
- **max_features** = $[0.4, 0.6, 0.8]$;
- **n_estimators** = $[150]$;
- **learning_rate** = $[0.1]$;

- **subsample** = $[0.75, 0.8, 0.85]$.

First we discuss results regarding AP scores. Gradient Boosting Regressor brings further improvements, obtaining fine prediction for all feature sets except *IR system conf* and *component prop*, which confirms their bad behaviour for topic performance prediction since they not contain any feature related to the topic. Anyway those set of features, as we have seen for Random Forests, give an important contribution in the whole feature set since it obtains best scores, even better than the *query + post retrieval features*.

	Training			**Test**			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.98464**	**0.00062**	**0.02500**	**0.97375**	**0.00106**	**0.03261**	**0.02307**
IR system conf	0.09802	0.03674	0.19167	0.09345	0.03684	0.19192	0.14555
Component prop	0.09761	0.03701	0.19239	0.09491	0.03622	0.19030	0.14363
IR system conf + Component prop	0.09759	0.03701	0.19239	0.09516	0.03621	0.19028	0.14388
Query features	0.85663	0.00586	0.07656	0.81983	0.00724	0.08505	0.05020
Post retrieval	0.74100	0.01059	0.10292	0.61300	0.01557	0.12476	0.09395
Query features + Post retrieval	0.97902	0.00085	0.02917	0.96516	0.00142	0.03778	0.02648

Table 5.22: Training and test scores for each feature subset on **AP** predictions with GBR.

Gradient Boosting Regressor permits to inspect feature performances too. Figure 5.19 represents feature importance of the whole feature set for AP prediction. The figure states that feature weights for GBR are very similar to the ones obtained with Random Forest in figure 5.9. The few variations can be considered normal since feature weights can slightly vary for each fitting processes of the algorithm; anyway, comparing those figures, it is clear that the two algorithms treat features in the same way. Figure 5.20 permits to focus on *IR system configuration* features. Comparing this plot with figure 5.10 obtained with Random Forest Regressor, shows us that also this set of features is treated in a similar way by the two algorithms. The features with the highest weight are the same for the two algorithms and they have the same order. However, Random Forest gives a

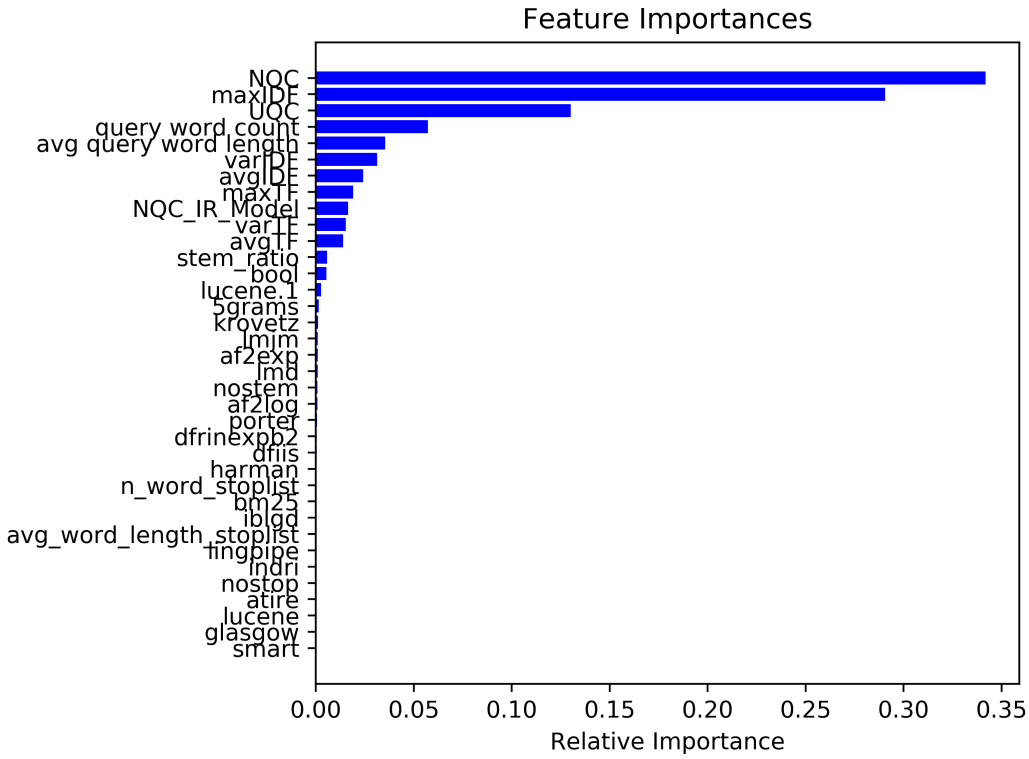

Figure 5.19: Feature importance of the whole feature set with GBR.

little more importance to most evaluated features: for example *bool* feature has a weight around 0.4 for RF and around 0.35 for GBR.

MAP is computed for each system of the test set and thus compared with the real MAP of the predicted IR systems, permitting to calculate Kendall's Tau and Person correlation coefficients. Figure 5.21 compares such results. First it is possible to appreciate the goodness of Pearson correlation coefficient and the Kendall's Tau ranking coefficient, especially for feature sets which contain *IR system conf* and/or *component prop*. The whole feature set obtains best results but they are slightly better than the ones achieved with sets containing features related to the IR system and the feature set formed by *query + post retrieval features*.

Tables 5.23, 5.25, 5.24, 5.26, 5.27, 5.28 report results of Gradient Boosting Regressor prediction on nDCG, recall, RBP, P@10, ERR and RR scores. With respect to Random Forest Regressor, GBR improves performances for all the

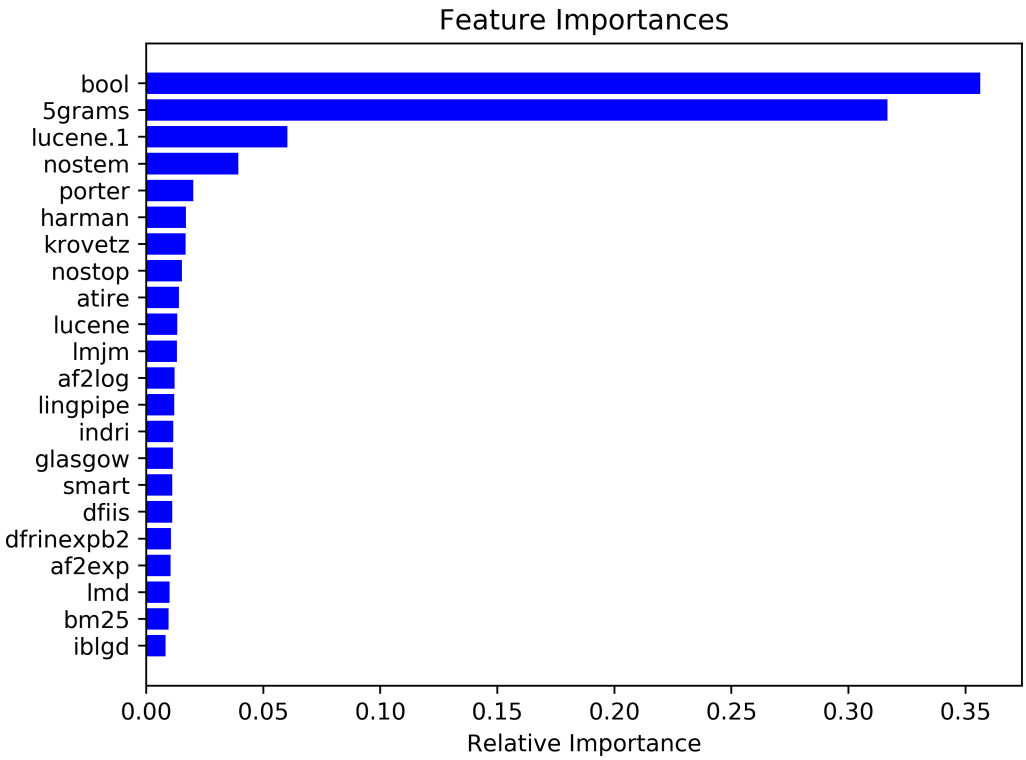

Figure 5.20: Feature importance of the *IR configuration* feature set with GBR.

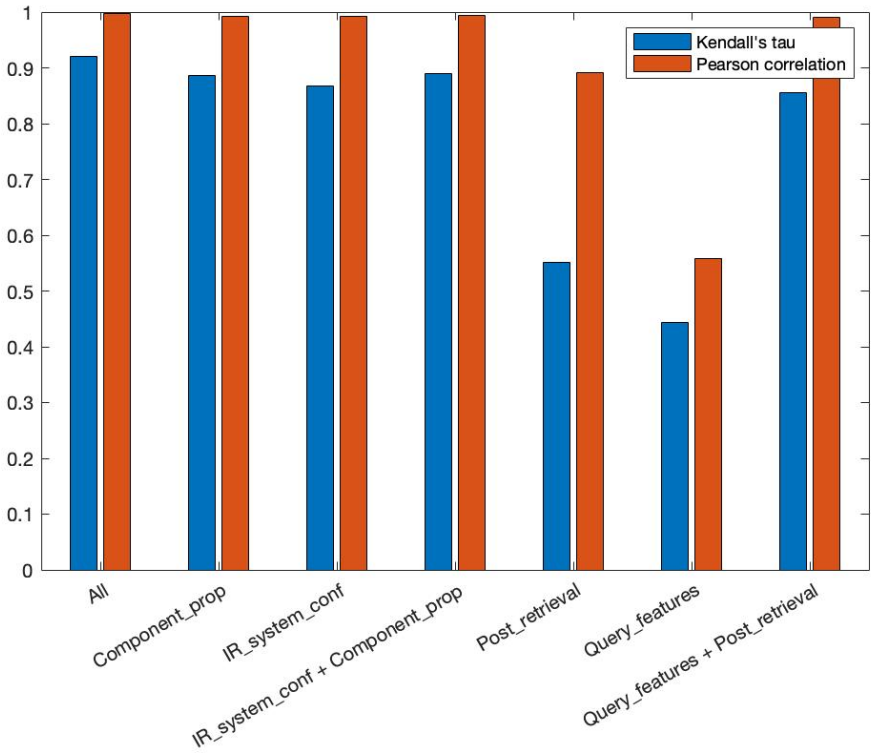

Figure 5.21: Kendall's tau and Pearson correlation of MAP related to different feature combinations with GBR.

measures with the whole feature set and the *query + post retrieval* feature set, while the results for *query features* and *post retrieval features* kept separately are almost the same for the two algorithms.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.97684**	**0.00140**	**0.03742**	**0.96293**	**0.00222**	**0.04717**	**0.03433**
IR system conf	0.13591	0.05226	0.22860	0.12923	0.05238	0.22888	0.19010
Component prop	0.13599	0.05211	0.22828	0.13051	0.05269	0.22955	0.19049
IR system conf + Component prop	0.13599	0.05211	0.22828	0.13098	0.05267	0.22949	0.19043
Query features	0.84339	0.00946	0.09728	0.84306	0.00946	0.09721	0.06575
Post retrieval	0.61143	0.02349	0.15326	0.48724	0.03092	0.17586	0.14341
Query features + Post retrieval	0.97145	0.00172	0.04151	0.95475	0.00273	0.05229	0.03786

Table 5.23: Training and test scores for each feature subset on **nDCG** predictions with GBR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.95511**	**0.00401**	**0.06339**	**0.92816**	**0.00639**	**0.07993**	**0.05870**
IR system conf	0.13529	0.07743	0.27827	0.12236	0.07813	0.27951	0.23439
Component prop	0.12803	0.07794	0.27918	0.13854	0.07700	0.27750	0.23199
IR system conf + Component prop	0.12802	0.07794	0.27918	0.13895	0.07697	0.27743	0.23182
Query features	0.76621	0.02079	0.14420	0.73967	0.02357	0.15350	0.10942
Post retrieval	0.57384	0.03790	0.19469	0.44896	0.04988	0.22334	0.17694
Query features + Post retrieval	0.94194	0.00519	0.07207	0.90880	0.00812	0.09010	0.06523

Table 5.24: Training and test scores for each feature subset on **RBP** predictions with GBR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.94523**	**0.00473**	**0.06883**	**0.91422**	**0.00745**	**0.08632**	**0.06385**
IR system conf	0.12552	0.07567	0.27508	0.13102	0.07554	0.27485	0.22632
Component prop	0.13111	0.07537	0.27454	0.11573	0.07625	0.27613	0.22797
IR system conf + Component prop	0.13111	0.07537	0.27454	0.11593	0.07623	0.27609	0.22805
Query features	0.75512	0.02120	0.14558	0.70318	0.02573	0.16030	0.11342
Post retrieval	0.56418	0.03773	0.19426	0.43407	0.04907	0.22151	0.17524
Query features + Post retrieval	0.93352	0.00577	0.07597	0.89914	0.00868	0.09319	0.06898

Table 5.26: Training and test scores for each feature subset on **P@10** predictions with GBR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.96964**	**0.00239**	**0.04897**	**0.94677**	**0.00417**	**0.06455**	**0.04590**
IR system conf	0.10620	0.07065	0.26580	0.10582	0.07010	0.26477	0.21936
Component prop	0.10474	0.07030	0.26515	0.10945	0.07090	0.26628	0.22054
IR system conf + Component prop	0.10474	0.07030	0.26515	0.10976	0.07088	0.26623	0.22056
Query features	0.86177	0.01084	0.10399	0.81921	0.01447	0.11972	0.07923
Post retrieval	0.55156	0.03515	0.18749	0.41023	0.04706	0.21693	0.17466
Query features + Post retrieval	0.96384	0.00285	0.05346	0.94159	0.00457	0.06756	0.04816

Table 5.25: Training and test scores for each feature subset on **recall** predictions with GBR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.91369**	**0.00568**	**0.07541**	**0.85970**	**0.00922**	**0.09601**	**0.06859**
IR system conf	0.13354	0.05710	0.23896	0.12129	0.05777	0.24035	0.21812
Component prop	0.13168	0.05721	0.23919	0.12542	0.05756	0.23991	0.21772
IR system conf + Component prop	0.13171	0.05721	0.23919	0.12639	0.05749	0.23978	0.21739
Query features	0.68368	0.02085	0.14442	0.64288	0.02344	0.15312	0.10998
Post retrieval	0.54477	0.03002	0.17326	0.38865	0.04014	0.20035	0.16304
Query features + Post retrieval	0.89220	0.00707	0.08412	0.83396	0.01100	0.10487	0.07472

Table 5.27: Training and test scores for each feature subset on **ERR** predictions with GBR.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.89300**	**0.01672**	**0.12932**	**0.82221**	**0.02805**	**0.16745**	**0.11764**
IR system conf	0.11873	0.13777	0.37118	0.12259	0.13842	0.37205	0.34087
Component prop	0.12058	0.13772	0.37111	0.11758	0.13863	0.37233	0.34202
IR system conf + Component prop	0.12061	0.13772	0.37110	0.11814	0.13855	0.37222	0.34111
Query features	0.64170	0.05599	0.23660	0.58733	0.06517	0.25514	0.18224
Post retrieval	0.51293	0.07612	0.27589	0.36470	0.10029	0.31668	0.25903
Query features + Post retrieval	0.87143	0.02013	0.14189	0.78993	0.03295	0.18151	0.12744

Table 5.28: Training and test scores for each feature subset on **RR** predictions with GBR.

5.2 Second task

In this section we analyse the behaviour of prediction algorithms when one or more IR components (realisations of stop list, stemmer and/or IR model) are present only in the test set. The algorithms involved are the same of the ones used for the

first prediction task, i.e. Linear Regression, Support Vector Regression, Random Forest Regressor and Gradient Boosting Regressor. The main difference is the criterion used to split the dataset: for this task, all the systems that contain one or more selected components are used excursively for the test set, thus eliminating those components from the fitting process. Basing on feature importance images shown in previous paragraphs, we have chosen some components to exclude from the training set and for forming the test set. We have chosen components with a low feature importance (so components like *bool, 5grams, lucene, nostem and porter* are excluded), since excluding from the training set important features may lead to very poor results. In fact, the systems formed by those components tend to have bad evaluations, very far from results achieved by other systems. As we are going to discuss in the last chapter, this problem may be solved by introducing new features that better describe the system and its performance. Then, among the remaining components, we have chosen two stop lists for one experiment, a stemmer for the second experiment and finally two IR models for the last experiment. This choice permits me to have a balanced training/test set separation; the first test set is composed by $2 \times 5 \times 10 = 100$ IR system combinations, the second has $7 \times 1 \times 10 = 70$ combinations and the last one has $7 \times 5 \times 2 = 70$ system combination on total of 350. For each of the three experiments, the components involved are:

1. *glasgow, smart*;
2. *harman*;
3. *af2exp, af2log*.

Those experiments are executed on four different feature set partitions: the whole feature set, IR system configuration, IR system configuration + component properties, query features + post retrieval features. This choice is due to the fact that the scope of this experiment is to show the difference between a feature set composed by IR system configuration, and a feature set with none information about the system configuration. Moreover it is interesting analyse the differences of the IR system configuration feature set adding the component properties.

5.2.1 Linear Regression

Linear Regression obtains poor results for each of the three experiments. Pearson correlation is - almost - always around 0 and Kendall's tau is always lower than 0.5, pointing out a bad ranking, considering that most of the systems have a similar MAP and thus it is possible to obtain a Kendall's tau higher than zero even with a bad prediction. Also the query performance prediction obtains low scores for the whole feature set with a R2 training score around 0.4/0.5 for each test and a R2 test score even less than zero. The only - poor - result comes from the *query features + post retrieval* feature set: it obtains R2 training and test scores around 0.45 for each experiment and a test RMSE around 0.15. Consequently Pearson correlation is not so bad, stabilizing around 0.8 for each experiment, instead Kendall's tau is quite poor being equal to 0.49, 0.43 and 0.18 in the three experiments. Clearly, results obtained by the set *query features + post retrieval*, are not satisfying at all, but are the best among the other tested feature sets, proving that this predictor suffers a lot the presence of a never seen categorical feature (i.e. IR component).

5.2.2 SVR

FIRST EXPERIMENT

This test is performed using all the IR systems that contain *glasgow* and *smart* stop list as test set. Figure 5.22 reports Kendall's tau and Person correlation of MAP related to different feature combinations. So AP predictions of each system configuration are averaged thus obtaining the MAP of each system. Pearson correlation is fine for each feature set, but Kendall's tau is slightly better for those sets that contain the features related to the system configuration. In particular, *IR system conf.* and *IR system conf. + component properties* obtain best results, followed by the whole feature set. The *query features + post retrieval* set, that does not contain information regarding the system configuration, obtains the worst results. This confirms the importance of introducing features relative to the system configuration.

It is worth mentioning also scores relative to query performance prediction for each system. We report only results related to the whole feature set and the *query features + post retrieval* set since, as we have seen in previous paragraph,

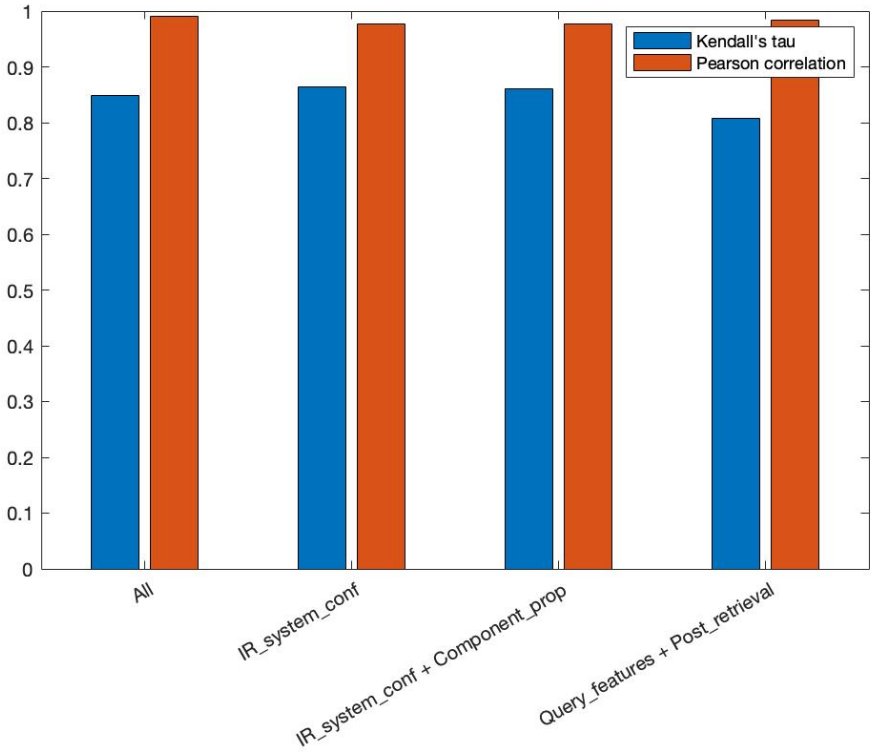

Figure 5.22: Kendall's tau and Pearson correlation of MAP related to different feature combinations with SVR and *never seen stop list*.

IR system conf. and *IR system conf. + component properties* obtain bad results for query prediction. Table 5.29 resumes the scores, showing that the *query features + post retrieval* set achieves the best results for every score. This fact, in our opinion, is due to the presence of the categorical feature set in the whole feature set that adds noise to the predictor; during the training phase, two of the categorical features do not appear at all, introducing a further difficulty in test prediction phase.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.77518	0.00911	0.09547	0.71674	0.01165	0.10798	0.09108
Query features + Post retrieval	**0.88939**	**0.00448**	**0.06697**	**0.88262**	**0.00483**	**0.06950**	**0.06023**

Table 5.29: Training and test scores for each feature subset on **AP** predictions with SVR and *never seen stop list*.

SECOND EXPERIMENT

This experiment is performed using all the IR systems that contain *harman* stemmer as test set. Figure 5.23 reports Kendall's tau and Person correlation of MAP related to different feature combinations. As illustrated in figure 5.23 Kendall's tau is worse than the previous test. This behaviour, in effect, is quite expected since all the stemmers have a relevant feature importance. Anyway, also in this case the best results are achieved by *IR system conf.* and *IR system conf. + component properties* and in particular the impact of the component properties is more relevant since they contribute to slightly improve the Kendall's tau ranking coefficient. Also in this test the *query features + post retrieval* obtains the worst result.

Also training and test scores suffer a slight worsening too. Table 5.23 reports the results confirming the difficulty of this task and the supremacy of *query features + post retrieval* with respect to the whole feature set for this task.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.77530	0.00900	0.09491	0.70887	0.01248	0.11172	0.09374
Query features + Post retrieval	**0.88553**	**0.00458**	**0.06774**	**0.83357**	**0.00713**	**0.08446**	**0.06813**

Table 5.30: Training and test scores for each feature subset on **AP** predictions with SVR and *never seen stemmer*.

THIRD EXPERIMENT

This experiment is performed using all the IR systems that contain *af2exp, af2log* IR models as test set. Figure 5.24 reports Kendall's tau and Person correlation of

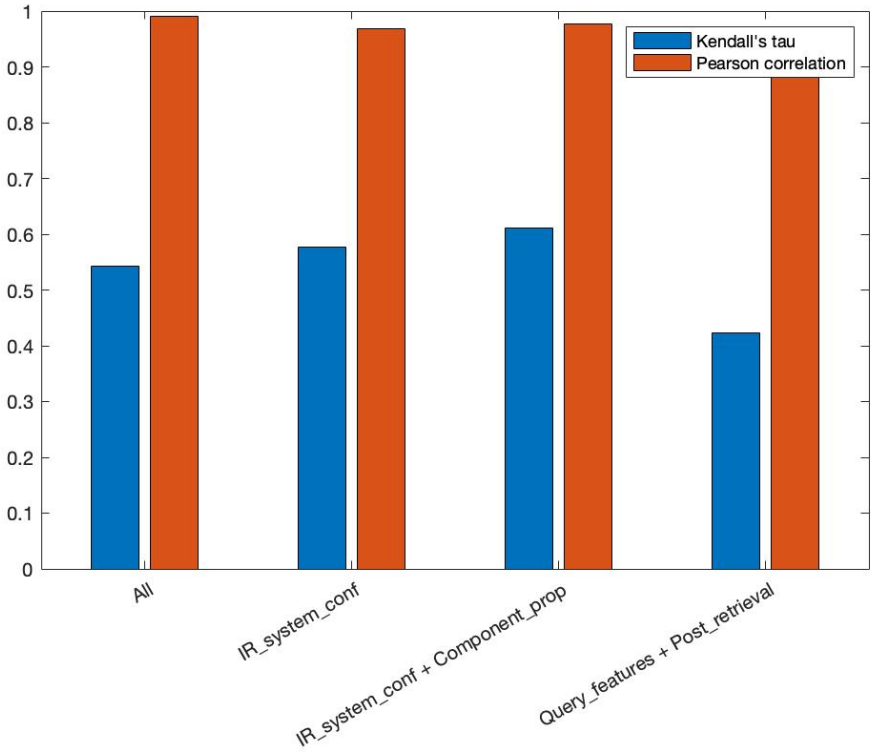

Figure 5.23: Kendall's tau and Pearson correlation of MAP related to different feature combinations with SVR and *never seen stemmer*.

MAP related to different feature combinations. The supremacy of *IR system conf.* and *IR system conf. + component properties* is more evident in this test since they obtain a Kendall's tau ranking coefficient sensibly better than the other two feature sets. In particular, it is worth pointing out the drop of the whole feature set performances. This is mainly due to bad query performance prediction, as we going to see later, that leads to bad MAP approximation and thus to worse Kendall's tau and Person correlation coefficients.

Table 5.31 reports query performance prediction scores of the whole feature set and the *query features + post retrieval* feature set. Although the scores for training are quite good, they drastically drop for test set demonstrating that excluding *af2exp, af2log* from training set makes the model more difficult for the predictor. Anyway, the fact that *query features + post retrieval* reports better test scores, confirms that it obtained better results related to Kendall's tau coefficient

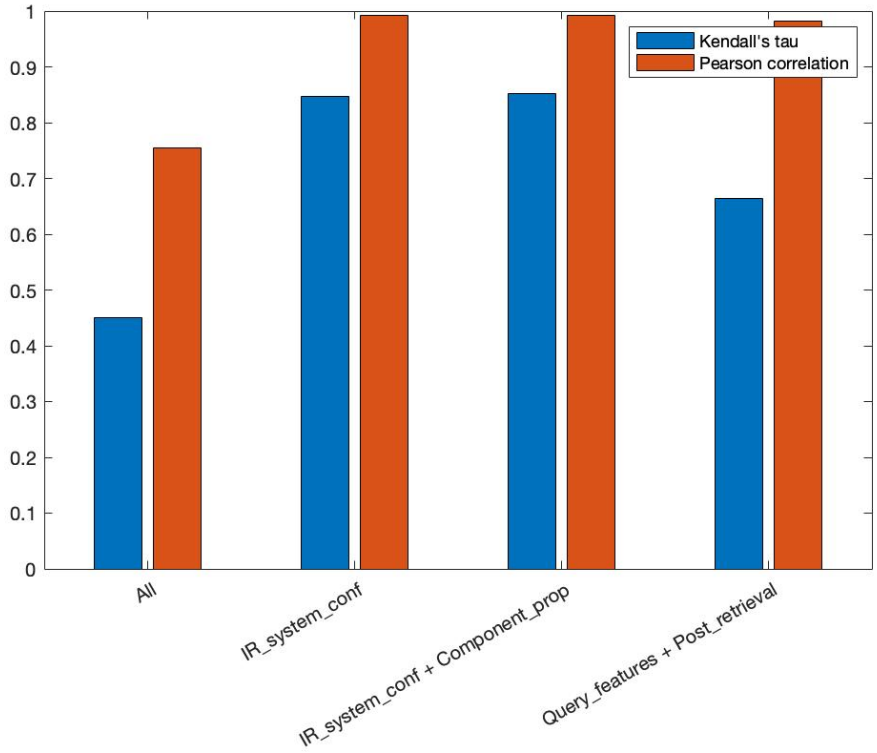

Figure 5.24: Kendall's tau and Pearson correlation of MAP related to different feature combinations with SVR and *never seen IR model*.

with respect to the whole feature set.

5.2.3 Random Forest Regressor

FIRST EXPERIMENT

The first experiment performed with Random Forest Regressor is made using all the IR systems that contain *glasgow* and *smart* stop list as test set. Figure 5.25 reports Kendall's tau and Person correlation of MAP related to different feature combinations. The first thing we can evince is that, for this experiment, Random Forest has performances slightly better that SVR for each feature set. In particular, we can notice that the feature set *IR system conf. + component properties* obtains the best Kendall's tau, confirming that the *component property* feature helps the algorithm to train in absence of samples containing the categorical fea-

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	0.77992	0.00878	0.09374	0.57601	0.01850	0.13602	0.10384
Query features + Post retrieval	**0.88113**	**0.00474**	**0.06889**	**0.65243**	**0.01516**	**0.12316**	**0.08058**

Table 5.31: Training and test scores for each feature subset on **AP** predictions with SVR and *never seen IR model*.

ture of *glasgow* and *smart*. Also in this case the worst result is obtained by the *query features + post retrieval* feature set, the only that does not contain the *IR system conf.* feature set.

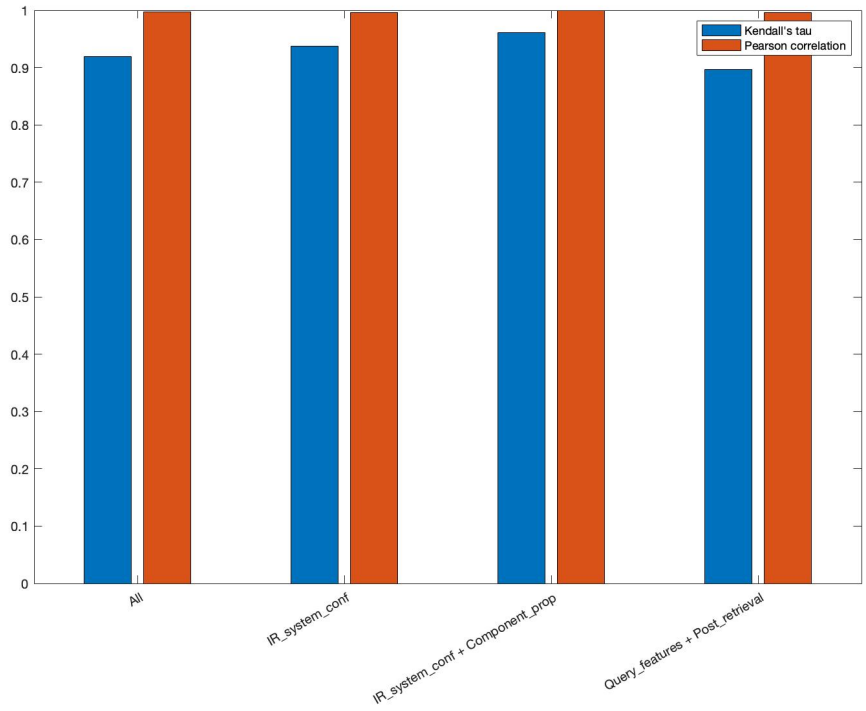

Figure 5.25: Kendall's tau and Pearson correlation of MAP related to different feature combinations with RF and *never seen stop list*.

Table 5.32 shows that Random Forest obtains better scores than SVR also for query performance prediction for each IR system. In particular, the most significant upgrade is achieved by the whole feature set obtaining better results than SVR especially in the test set. In addition, the whole feature set obtains better

results than the *query features + post retrieval*: the Random Forest penalizes less the absence of information for the missing components in the training set, permitting the predictor to obtain better results.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.96574**	**0.00138**	**0.03727**	**0.95147**	**0.00199**	**0.04469**	**0.02876**
Query features + Post retrieval	0.96424	0.00144	0.03807	0.94777	0.00214	0.04636	0.02995

Table 5.32: Training and test scores for each feature subset on **AP** predictions with RF and *never seen stop list*.

SECOND EXPERIMENT

This test is performed using all the IR systems that contain *harman* stemmer as test set. Figure 5.26 reports Kendall's tau and Person correlation of MAP related to different feature combinations. As we have seen for SVR, the absence of *harman* stemmer in the training set affects sensibly the ranking performances for the MAP of the involved IR systems. However, this second experiment confirms the goodness of Random Forest predictions since the results are better than the ones obtained with SVR. Consider, for example, the *IR system conf. + component properties* feature set: with Random Forest it obtains a Kendall's tau of 0.82111 against the 0.61076 obtained with SVR: it is a significant improvement. Also in this experiment the best results are achieved by *IR system conf.* and *IR system conf. + component properties*, followed by the whole feature set and the *query features + post retrieval* that still obtains the worst result with a Kendall's tau equal to 0.57101.

The drop of the raking coefficient goodness is due also to the lower results obtained for query performance prediction scores. Table 5.33 reports such results showing the scores for the whole feature set and the *query features + post retrieval* feature set. The best scores are still achieved by the whole feature set, and the score improvement with respect to SVR is confirmed also in this experiment.

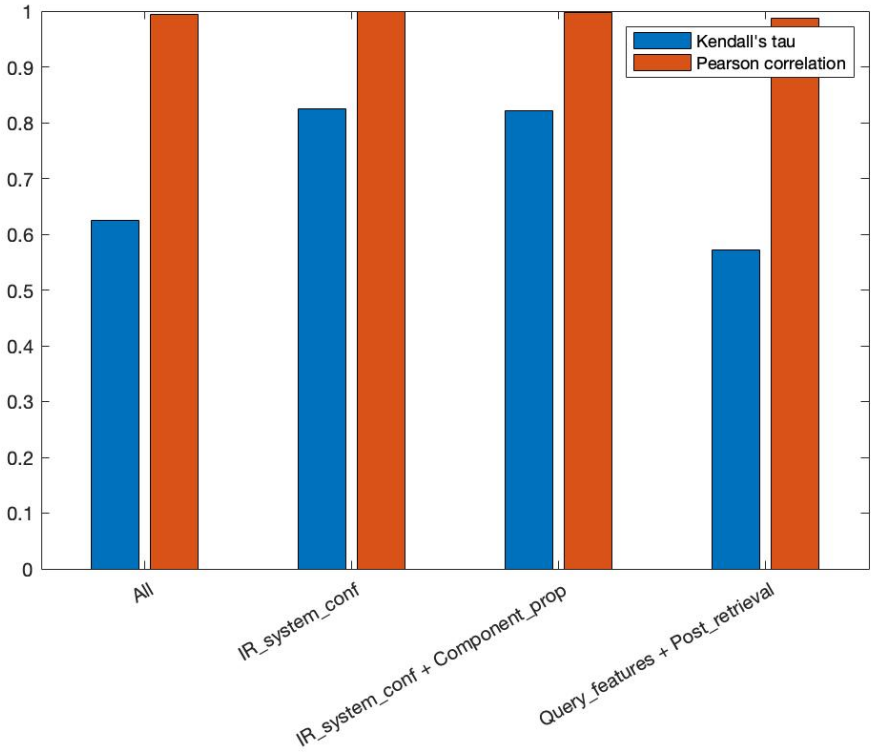

Figure 5.26: Kendall's tau and Pearson correlation of MAP related to different feature combinations with RF and *never seen stemmer*.

THIRD EXPERIMENT

This test is performed using all the IR systems that contain *af2exp, af2log* IR models as test set. Figure 5.27 reports Kendall's tau and Person correlation of MAP related to different feature combinations. The Kendall's tau ranking coefficient is slightly better than the second test especially for *IR system conf.* and *IR system conf. + component properties*, marking the different with the other two feature sets.

The absence of *af2exp, af2log* in the training set affects the results of the test set, that are even worse than the scores obtained in table 5.33 for the second experiment. However, comparing this results (table 5.34) with the ones obtained for the same experiment with SVR, we can notice a great improvement.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.97106**	**0.00115**	**0.03405**	**0.90267**	**0.00417**	**0.06459**	**0.04313**
Query features + Post retrieval	0.96788	0.00128	0.03588	0.89623	0.00444	0.06670	0.04464

Table 5.33: Training and test scores for each feature subset on **AP** predictions with RF and *never seen stemmer*.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.96662**	**0.00133**	**0.03650**	**0.83135**	**0.00736**	**0.08579**	**0.05747**
Query features + Post retrieval	0.96247	0.00149	0.03871	0.81535	0.00805	0.08976	0.05839

Table 5.34: Training and test scores for each feature subset on **AP** predictions with RF and *never seen IR model*.

5.2.4 Gradient Boosting Regressor

FIRST EXPERIMENT

The first test, performed with Gradient Boosting Regressor, is made using all the IR systems that contain *glasgow* and *smart* stop list as test set. Figure 5.28 reports Kendall's tau and Person correlation of MAP related to different feature combinations. The first main difference is that the whole feature set obtains the highest Kendall's tau ranking coefficient. The other feature set obtains slightly lower results, and the *query features + post retrieval* feature set still obtains the worst result. It is important to notice also that for the first experiment this algorithm obtains results as not as good as Random Forest Regressor. For example, Kendall's tau coefficient of the *IR system conf. + component properties* set gets worse passing from 0.96161 to 0.89710. The only improvement is encountered on the whole feature set with a Kendall's tau equal to 0.93438 against the 0.91878 of the Gradient Boosting Regressor.

The cause of the improvement of the Kendall's tau for the whole feature set can be easily traced back to the improvement of the scores for query performance prediction. Table 5.35 reports the scores for the whole feature set and the *query features + post retrieval* feature set. Also in this case the set containing all the

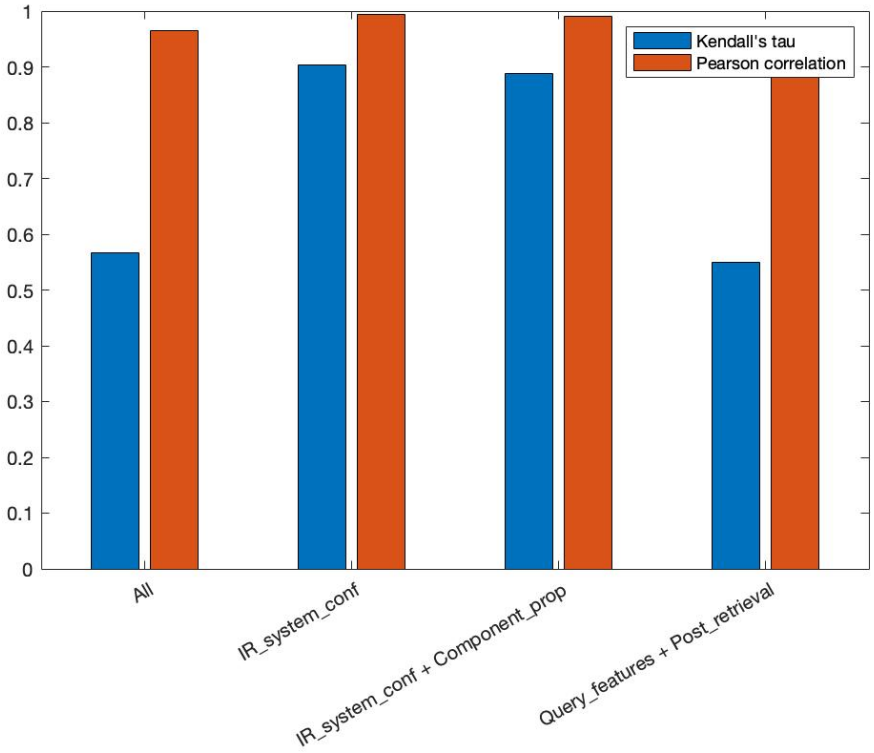

Figure 5.27: Kendall's tau and Pearson correlation of MAP related to different feature combinations with RF and *never seen IR model*.

features obtains the best scores, confirming the goodness of Random Forest and Gradient Boosting Regressor for treating categorical features. In addition, as we mentioned before, it is interesting the improvement of the test scores for this algorithm: the Gradient Boosting Regressor obtains a R2 score 0.97063 against the 0.95147 of the Random Forest for the whole feature set and a R2 score equal to 0.96371 with respect to a value of 0.94777 of the Random Forest for *query features + post retrieval*. From tables 5.35 and 5.32 we can see a improvement of other scores as well.

SECOND EXPERIMENT

This experiment is performed using all the IR systems that contain *harman* stemmer as test set. Figure 5.29 reports Kendall's tau and Person correlation of MAP related to different feature combinations. It is immediately evident that, for this

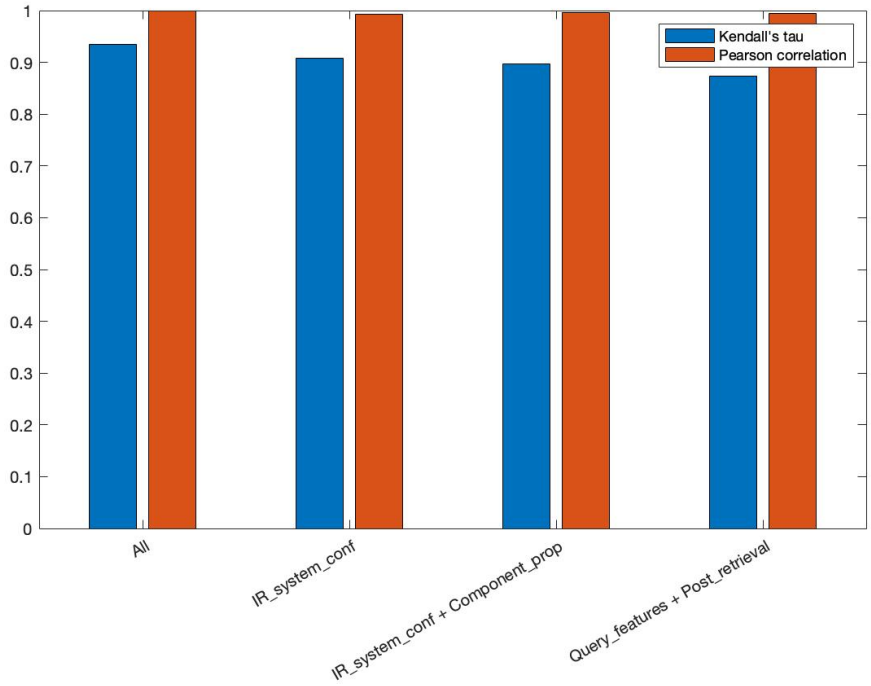

Figure 5.28: Kendall's tau and Pearson correlation of MAP related to different feature combinations with GBR and *never seen stop list*.

experiment, Kendall's tau coefficient is not as good as the one obtained with Random Forest. *IR system conf.* and *IR system conf. + component properties* feature sets have the worst decrease (0.82443 vs 0.66683 and 0.82111 vs 0.68844 respectively), instead the whole feature set benefits of an increase passing from the value 0.62401 of Random Forest to the actual 0.69929. Anyway, as we have seen for other algorithms, the coefficients of the second experiment are worse than the ones of the first one.

Table 5.36 shows the query performances of the two feature sets. The scores of the test set decrease with respect to the ones of the first experiment but they are very similar to the test scores obtained by Random Forest Regressor in the same experiment.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.98369**	**0.00066**	**0.02570**	**0.97063**	**0.00120**	**0.03475**	**0.02405**
Query features + Post retrieval	0.97819	0.00088	0.02972	0.96371	0.00149	0.03864	0.02674

Table 5.35: Training and test scores for each feature subset on **AP** predictions with GBR and *never seen stop list*.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.98526**	**0.00059**	**0.02430**	**0.90978**	**0.00386**	**0.06216**	**0.04174**
Query features + Post retrieval	0.98000	0.00080	0.02831	0.89896	0.00433	0.06577	0.04445

Table 5.36: Training and test scores for each feature subset on **AP** predictions with GBR and *never seen stemmer*.

THIRD EXPERIMENT

This experiment is performed using all the IR systems that contain *af2exp, af2log* IR models as test set. Figure 5.30 reports Kendall's tau and Person correlation of MAP related to different feature combinations. The ranking coefficient, also for this predictor, gets better in the third experiment with respect the second one and highlights more clearly the gap between *IR system conf.* and *IR system conf. + component properties* and the other two feature set. However, with respect to Random Forest Regression, the Kendall's tau is worse except for the whole feature set and the *query features + post retrieval* set which obtain slightly better results.

The improvement of the Kendall's tau for the whole feature set and the *query features + post retrieval* set can be noticed also in table 5.37. In fact, test scores are better for this predictor than the ones obtained by Random Forest Regressor. Using the R2 test score as comparison parameter, the whole feature set in Gradient Boosting Regressor is 0.90314 with respect to the 0.83135 of the Random Forest and the *query features + post retrieval* obtains a score 0.61540 against the previous 0.55031.

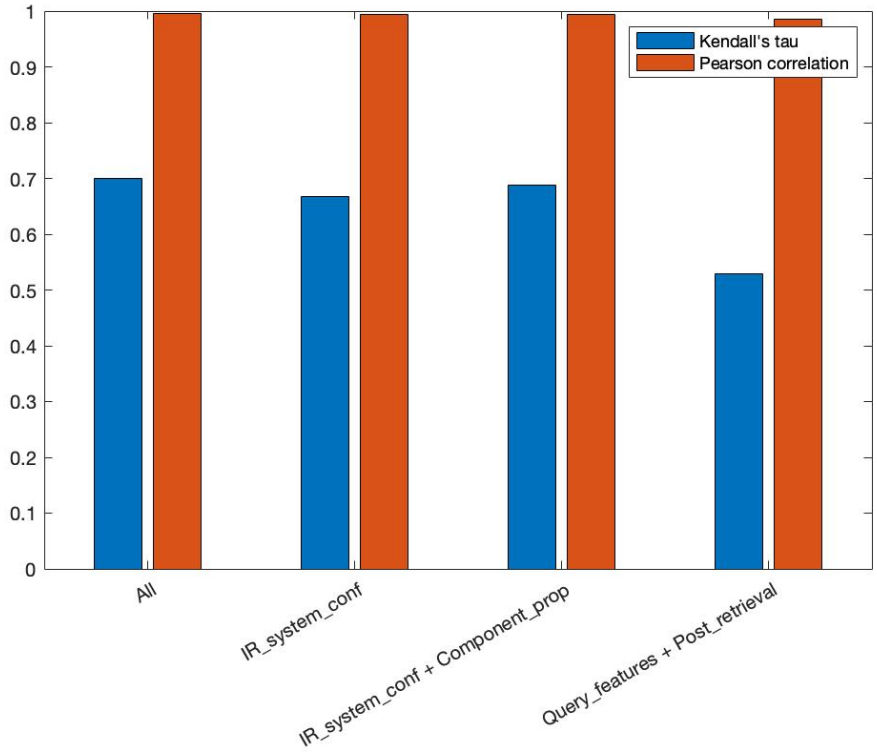

Figure 5.29: Kendall's tau and Pearson correlation of MAP related to different feature combinations with GBR and *never seen stemmer*.

5.3 Overall results

In this paragraph we present and compare the results of the two tasks.
In the first task 105 on a total of 350 IR system configurations, are used excursively in the test set: this permits to inspect the behaviour of the different predictors when they are trained on a set of IR systems, and tested on another one. The measures predicted for each topic of each system are quite satisfactory, especially using Random Forest and Gradient Boosting Regressor. Table 5.38 gives an overview of the scores achieved by the best feature set for each predictor.

Also other measures obtain results similar to AP, except for **ERR** and **RR** which have scores slightly lower. Anyway, performances obtained with AP reflect performances of other measures too, with Random Forest and Gradient Boosting Regressor that still achieve the best performances.

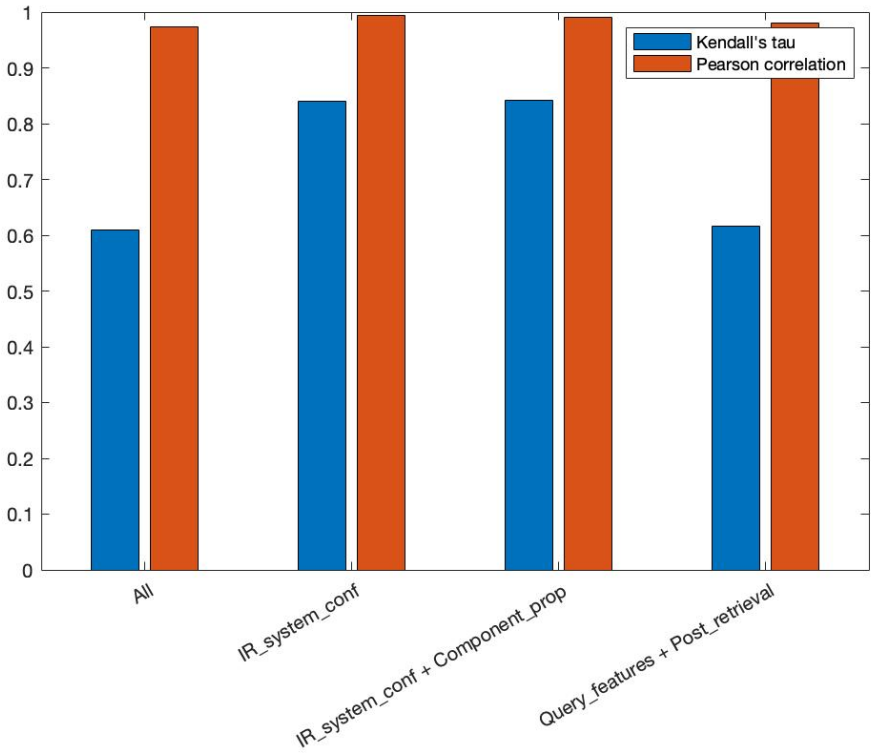

Figure 5.30: Kendall's tau and Pearson correlation of MAP related to different feature combinations with GBR and *never seen IR model*.

	Training			Test			
	R2 score	**MSE**	**RMSE**	**R2 score**	**MSE**	**RMSE**	**MAE**
All	**0.98491**	**0.00060**	**0.02453**	**0.90314**	**0.00422**	**0.06498**	**0.04244**
Query features + Post retrieval	0.98001	0.00079	0.02823	0.88494	0.00502	0.07084	0.04631

Table 5.37: Training and test scores for each feature subset on **AP** predictions with GBR and *never seen IR model*.

	Feature Set	Training			Test			
		R2 score	MSE	RMSE	R2 score	MSE	RMSE	MAE
Linear Regression	All	0.48370	0.02101	0.14494	0.48179	0.02110	0.14525	0.10852
SVR	Query features + Post retrieval	0.87979	0.00493	0.07026	0.86996	0.00518	0.07201	0.05941
Random Forest	All	0.96582	0.00137	0.03708	0.95418	0.00191	0.04374	0.02906
GBR	All	**0.98464**	**0.00062**	**0.02500**	**0.97375**	**0.00106**	**0.03261**	**0.02307**

Table 5.38: Best training and test scores on **AP** predictions for each algorithm of the first task.

Then, for each system of the test set, we have computed the average of its AP predictions in order to obtain its Mean Average Precision (MAP). This measure is crucial for comparing systems' performance. Then we compared MAP of systems of the test set with their real MAP, obtaining interesting results both in terms of ranking correlation (Kendall's tau coefficient) and Pearson Correlation. The most interesting result is that the best Kendall's tau is achieved by feature sets containing features related to the configuration of the IR systems, i.e. the categorical features describing the combination of components. In particular, it is worth noticing that feature sets with the best Kendall's tau obtained the worst scores for query performance prediction of the systems. This was due to the fact that this task is strictly related to the topic involved and those feature sets have features related only to the system; for this reason, for each system, the predictions related to each of the 50 topics, are the predictions of the MAP of that system (see figure 5.2). So, for each system, averaging the 50 predictions, the result is exactly the MAP of that system. In particular this measure seems to be quite accurate since they obtain a satisfactory Pearson correlation and a high Kendall's tau. Also for this task Random Forest Regression and Gradient Boosting Regressor reveal to be the best predictors. In particular, Random Forest obtains the best Kendall's tau for the feature sets strictly related to the system configuration, while Gradient Boosting Regressor is the best for the whole feature set (figure 5.31).

The second task of this thesis is to predict performances of IR systems with a never seen feature. This means that, among the 22 IR components (i.e. categorical features), some of them appear only in the test set, thus training the predictor without any sample of that component. This task increases the difficulty of the

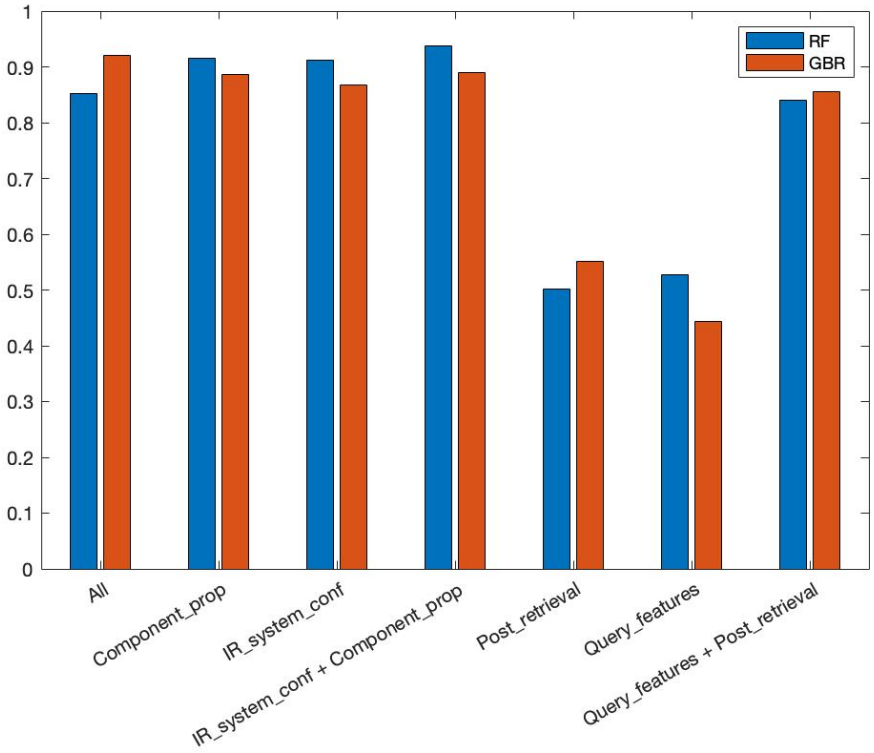

Figure 5.31: Comparison of Kendall's tau between Random Forest Regression and Gradient Boosting Regressor of the first task.

prediction and tests the robustness of our dataset and our models. For each of the four predictors, this task is articulated on three different experiments, each of them including different components in the test set. The first test set is composed by systems with *glasgow* and *smart* stop list, the second one has systems with *harman* stemmer and the last one includes systems with *af2exp* and *af2log* IR models. Linear Regression revealed to be not suitable for this task, leading to bad results. However, the other three predictors, achieved good results in terms of Kendall's tau. Their results, for each feature set of each experiment, are reported in figures 5.32, 5.33 and 5.34. From those graphs we can see that the absence of the two stop list in the first experiment is the situation in which the predictors suffer less the absence of one component in the training. The second experiment obtains values slightly lower since the stemmer has a great influence, as we have seen in the feature importance plot. Finally, the third experiment highlights the

ability of feature sets related to the IR system into predict a correct rank and other feature sets are more penalised. For each experiment the Random Forest achieves the best results, except for the whole feature set where, as we have seen in the first task, Gradient Boosting Regressor works better.

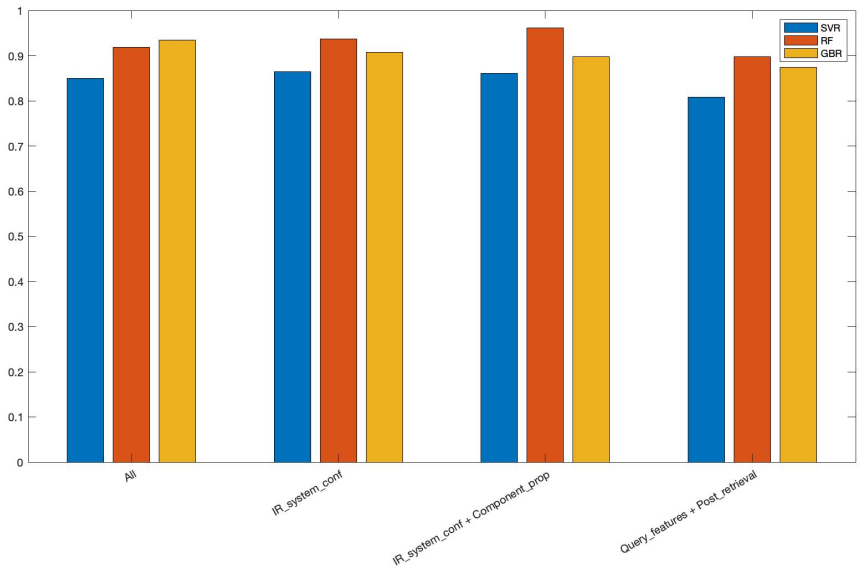

Figure 5.32: Comparison of Kendall's tau between SVR, Random Forest Regression and Gradient Boosting Regressor for first experiment of never seen component task.

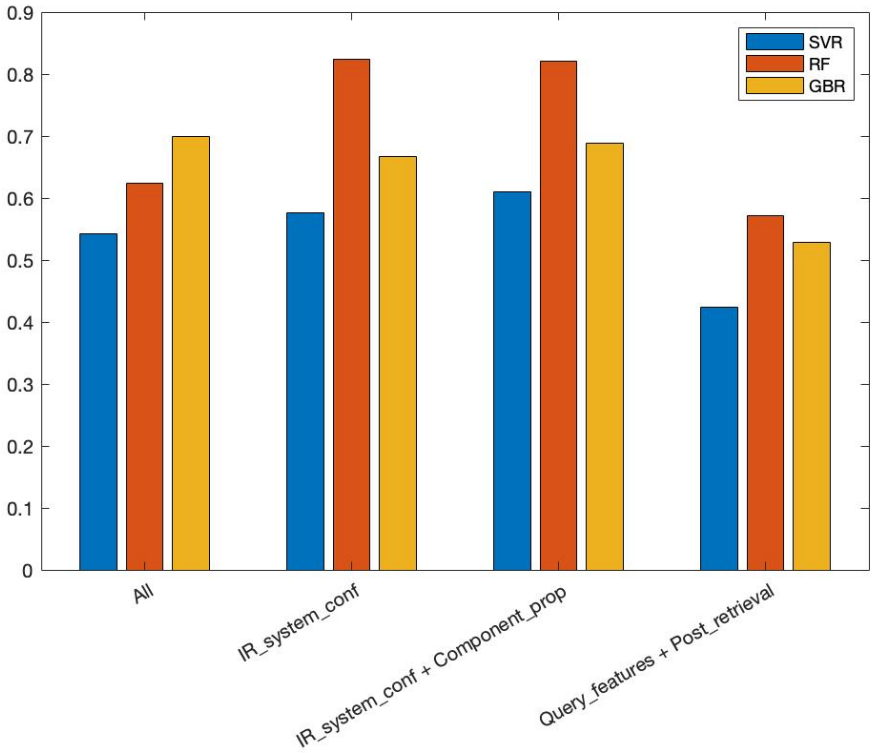

Figure 5.33: Comparison of Kendall's tau between SVR, Random Forest Regression and Gradient Boosting Regressor for second experiment of never seen component task.

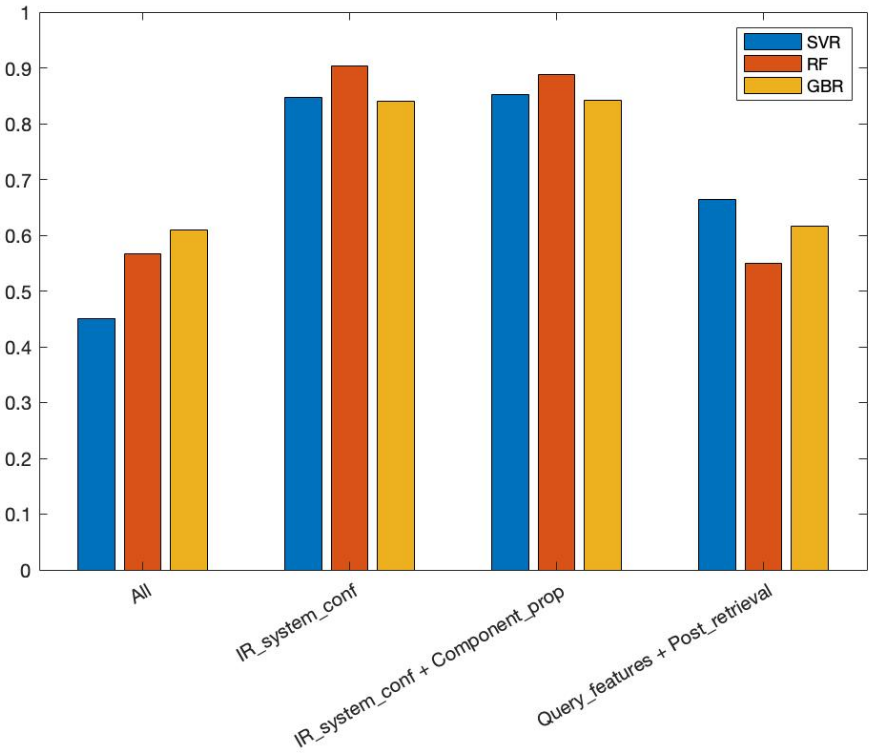

Figure 5.34: Comparison of Kendall's tau between SVR, Random Forest Regression and Gradient Boosting Regressor for third experiment of never seen component task.

6

Conclusion

In this thesis we have seen two prediction tasks applied to Information Retrieval. Machine learning and Information Retrieval are two search fields that converges very often and for a huge variety of purposes. In particular, our contribution is to predict performances of combinations of IR systems. **GoP**[1] permitted us to obtain runs originated by combinations of IR system components (stop list, stemmer and IR models) and therefore to create our dataset, formed by 36 different features and 7 target measures, i.e. the labels of our predictions. The job is articulated on two task, namely *never seen configuration* and *never seen component*. Moreover, the involved predictors are four: Linear Regression, Support Vector Regression, Random Forest Regression and Gradient Boosting Regressor.
Most of the times Random Forest Regression and Gradient Boosting Regressor achieved the best results. The difference with Linear Regression and SVR is more evident when we introduce categorical features: they sensibly increase the dimension of linear predictors. Instead, ensemble predictors have a better management of the problem since they split it in multiple learners and consider a portion of the total number of features, defined by the proper parameter. We have seen the importance of topic-related features for query prediction, but we have seen that, by introducing system-related features to the topic-related ones, may increase the

[1] http://gridofpoints.dei.unipd.it

performances. System-related features give the main contribution for the ranking task, when we compare the MAP of the predicted systems. In this context, system-related features achieve the best performances, obtaining better results than query-related features. Anyway, we have to consider the space in which we are ranking the systems. The MAP that we have seen in this work is located in a narrow range, thus producing a very dense classification of systems. As we have seen, this lead to a good Person correlation coefficient but may lead to some swaps that alter the ranking.

However, as it is pioneering work in this area, we have no previous experience and this leads to numerous scenarios for future work. A first interesting argument is to verify the performance on new topics. In fact, performing our experiments on query performance prediction, we based on a finite set of queries well known by the predictor. Our models obtains satisfying results on repeated queries, so a future work is to perform the same test on never seen queries.
Certainly, it would be interesting to repeat our test on a lager dataset, also by introducing a new corpus and new queries. By increasing the complexity of the problem, we would also need better features. It is possible to introduce a new set of query-related features extracted by means of a linguistic approach. Some linguistic features are the average number of morphemes per query word, the number of proper nouns, the number of acronyms, numeral values, unknown tokens, [35] etc. These features usually do not achieve good performances, but it would be interesting to inspect if they can increase query prediction performance in our study, especially expanding the number of topic, the corpus and making predictions on never seen queries. Others examples of pre-retrieval features to introduce are the *similarity* between the query and the collection and the *term relatedness* for exploring term co-occurrence statistics. It is also possible to introduce a new set of post-retrieval features, such as *clarity*, *Weighted Information Gain* (WIG) and *Query Feedback* (QF) [35].
An interesting task is to enforce system-related features in order to improve the contribution of system configuration both in query prediction and system ranking. This can be done by adding new features that describe the system components and the relation among them.
It is possible to obtain a new feature that describes the IR model similar to the one that we have described in chapter 4, but instead using the most frequent

English words as query to compute the NQC, we can use some key words of the corpus. The same procedure can be repeated by combining all the stop list and IR models: the feature just mentioned was computed without the application of any stop list, and for this reason we may be interested into analysing the behaviour of this feature using each available stop list combined with the IR model.
Another feature can be computed by creating a sort of histogram where the values are the scores of a specific IR model obtained with unique words taken from the corpus. When a query is submitted, it is possible to compute the ratio between the sum of the values of query words and the total sum of all terms. Although this feature could be very useful since it correlates the query with the IR model, it may increase the retrieval time since for computing this feature we need to search the score of each query word among the whole set of terms. If we want to save time, we can use this concept to compute a feature that is no more related with the query but that expresses the system configuration. In fact, it is possible to calculate the ratio between the sum of IR model scores for all the terms, except the ones discarded by the involved stop list and/or stemmer, and the sum of the whole set of terms. A similar idea can be applied to compute a feature that describes a stop list. It is possible to create a sort of histogram where the values are the term frequency of all the terms in the collection. Therefore, the feature is computed as a ratio between the sum of tf of stop list words, and the sum of the tf of all the terms.

References

[1] W. B. Croft, D. Metzler, and T. Strohman, *Search Engines - Information Retrieval in Practice*. Pearson Education, Inc., 2015.

[2] N. Ferro and C. Peters, *Information Retrieval Evaluation in a Changing World - Lessons Learned from 20 Years of CLEF*. Springer Nature Switzerland, 2019.

[3] A. Moffat and J. Zobel, "Rank-biased precision for measurement of retrieval effectiveness," *ACM Trans. Inf. Syst.*, vol. 27, no. 1, Dec. 2008.

[4] M. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[5] S. N. Kasturi, "Underfitting and overfitting in machine learning and how to deal with it," Jun 2019. [Online]. Available: https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf

[6] Krishni, "K-fold cross validation," Dec 2018. [Online]. Available: https://medium.com/datadriveninvestor/k-fold-cross-validation-6b8518070833

[7] S. Swaminathan, "Linear regression - detailed view," Jan 2019. [Online]. Available: https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86

[8] R. Gandhi, "Support vector machine - introduction to machine learning algorithms," Jul 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[9] G. Zhang, "What is the kernel trick? why is it important?" Nov 2018. [Online]. Available: https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d

[10] I. Bhattacharyya, "Support vector regression or svr," Jul 2018. [Online]. Available: https://medium.com/coinmonks/support-vector-regression-or-svr-8eb3acf6d0ff

[11] R. S. Brid, "Decision trees-a simple way to visualize a decision," Oct 2018. [Online]. Available: https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb

[12] A. Chakure, "Random forest and its implementation," Jan 2020. [Online]. Available: https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f

[13] M. Sanderson and W. B. Croft, "The History of Information Retrieval Research," *Proceedings of the IEEE*, vol. 100, pp. 1444–1451, 2012.

[14] M. K. Buckland, "Emanuel Goldberg, Electronic Document Retrieval, And Vannevar Bush's Memex," *American Society for Information Science*, vol. 4, pp. 284–294, 1992.

[15] E. M. Voorhees, *The Evolution of Cranfield*. Cham: Springer International Publishing, 2019, pp. 45–69.

[16] S. Robertson, "On the history of evaluation in IR," *Journal of Information Science*, vol. 4, pp. 439–456, 2008.

[17] 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 and Computing*, vol. 25, no. 4, pp. 394–413, 2014.

[18] N. Fuhr, "Salton award lecture: information retrieval as engineering science." *SIGIR Forum*, pp. 19–28, 2012.

[19] E. Zhang and Y. Zhang, *Average Precision*. Boston, MA: Springer US, 2009, pp. 192–193.

[20] S. M. Beitzel, E. C. Jensen, and O. Frieder, *MAP*. Boston, MA: Springer US, 2009, pp. 1691–1692.

[21] N. Craswell, *Precision at n.* Boston, MA: Springer US, 2009, pp. 2127–2128.

[22] K. Järvelin and J. Kekäläinen, *Discounted Cumulated Gain.* Boston, MA: Springer US, 2009, pp. 849–853.

[23] O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan, "Expected reciprocal rank for graded relevance," 01 2009, pp. 621–630.

[24] S. Büttcher, C. Clarke, and G. V. Cormack, *Information Retrieval: Implementing and Evaluating Search Engines.* The MIT Press, 2010.

[25] N. Ferro and G. Silvello, "Toward an anatomy of ir system component performances," *Journal of the Association for Information Science and Technology*, vol. 69, no. 2, pp. 187–200, 2018.

[26] N. Ferro, "Ims @ trec 2017 core track," *TREC 2017 Notebook – Core Track*, 2017.

[27] R. Krovetz, "Viewing morphology as an inference process," *Artificial Intelligence*, vol. 118, no. 1, pp. 277 – 294, 2000.

[28] D. K. Harman, "How effective is suffixing?" *JASIS*, vol. 42, pp. 7–15, 1991.

[29] G. Amati and C. Rijsbergen, "Probabilistic models of information retrieval based on measuring the divergence from randomness," *ACM Transactions of Information Systems (TOIS)*, vol. 20, 10 2002.

[30] I. Kocabas, B. Dincer, and B. Karaoğlan, "A nonparametric term weighting method for information retrieval based on measuring the divergence from independence," *Information Retrieval*, pp. 1–24, 05 2013.

[31] C. Zhai, *Statistical Language Models for Information Retrieval.* Hanover, MA, USA: Now Publishers Inc., 2008.

[32] S. Clinchant and E. Gaussier, "Information-based models for ad hoc ir," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 234–241.

[33] H. Fang and C. Zhai, "An exploration of axiomatic approaches to information retrieval," in *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 480–487.

[34] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014.

[35] D. Carmel and E. Yom-Tov, *Estimating the Query Difficulty for Information Retrieval*. Morgan Claypool, 2010.