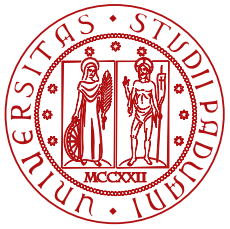



Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering


**MASTER DEGREE IN COMPUTER ENGINEERING**

# Desing, Development and Benchmarking of Algorithms for Conversational Search


Master Candidate

**Mattia Romanello**

**Student ID 2020379**

Supervisor

**Prof. Nicola Ferro**

**University of Padua**

Co-supervisor

**Guglielmo Faggioli**

**University of Padua**



Academic Year: **2021/2022**

Graduation date: $13^{th}$ **October 2022**

*To those who believed in me from the beginning.*

*And until the end.*

**Abstract**

Developing an intelligent dialog system that not only emulates human conversation, but also answers to difficult topics is one of the most important fields on several research area. In recent years, great strides have been made in this area and several companies and research groups create competitions that aim to find solutions to problems like *Conversational Information Seeking* and *Natural Language Generation*. On this work we see one of them in particular: **TREC CaST** *(Conversational Assistance Track)*. We analyze several techniques that allow to create a conversational system and how we can improve the results by using neural techniques. On this work we examine how to retrieve relevant documents by using Lucene and then to re-rank this documents by using a neural text-classifier like BERT.

**Sommario**

Lo sviluppo di un sistema intelligente che non emula solamente una conversazione umana, ma risponde anche a difficili argomenti è uno dei campi più importanti in diverse aree di ricerca. Negli ultimi anni sono stati svolti numerosi passi in avanti in questo settore e gruppi di ricerca tra cui Università e famose aziende stanno investendo nella creazione di competizioni che mirano a trovare soluzioni a questo problema. In questo documento guarderemo una in particolare: TREC CaST (Conversational Assistance Track). Analizzeremo diverse tecniche che mirano alla creazione di un sistema conversazionale, andremo a vedere come migliorare il sistema in modo da dare sempre i migliori risultati all'utente. In questo documento esamineremo come costruire un sistema conversazionale con *Lucene* per poi svolgere un re-ranking mediante un sistema di rete neurale, BERT. Per ogni metodologia di risoluzione sarà mostrato anche il risultato ottenuto per poi analizzare il tutto al termine del documento.

# Contents

# 1

# Introduction

Developing an intelligent dialog system that not only emulates human conversation, but also answer to topics ranging from mathematical concepts to current news is one of the most important fields on several research areas like *Natural Language Processing* (NLP), *Information Retrieval* (IR) and *Machine Learning* (ML) [5]. Over the years, information retrieval and search systems have become more conversational and a lot of personal assistant such as Amazon Alexa, Apple Siri, Google Home and Microsoft Cortana are becoming more important in our life. What these tools and technologies offer is certainly a searching method very similar to how human thinks, which isn't to write complex queries to obtain certain information, but using natural language with the system that will direct you to satisfy your information need. Systems with multi-turn capabilities and natural language capabilities have been studied for decades, but only in the last few years we have seen a growing evolution. There are many factors which brought about this grow, for example one of the most important is the progress on machine learning in fields like Natural Language Understanding and Spoken Language Understanding. The latest generation systems have been developed to support queries that refer indirectly to previous answers, previous questions or to specific arguments discussed during one of the previous

iterations. The growing interest on this field prompts many research group in Universities, Industries and Governments to invest and create competitions that aim to find solutions on this growing topic. *TREC CAsT* [4] 2022 is the fourth edition of *Conversational Assistance Track* where the goal is to pursue creating large-scale reusable test collections for open-domain conversational search [22]. The task aim to satisfy a user's information need which is expressed or formalized through a conversation turns. On this document we adopt terminology used in the speech and dialog context, for example the most basic unit is an *utterance* (equal to a single query on an information retrieval system). All contiguous utterances in the same context represent a single *turn* on a specif *topic*. During these years, several techniques have been implemented in CAsT, from the most advanced systems that involve neural query rewriting and neural re-ranker to the traditionals `NLP` approaches [3] which still present good results in terms of recall performance. One important change in CAsT 2022 is that for each turn in the conversation, the system may return a response or ask a question to clarify the user's information need [4]. On this work we concentrate on the main task of TREC CAsT: retrieve relevant documents and answer to each utterance. For the competition three collections are used MSMARCO, WaPo and KILT Benchmarking. On this work we will describe some approaches to solve the competition of *TREC CAsT* 2021, the 2022 competition had no way to be tested and analyzed yet (but the same techniques were used to produce 4 runs to be submitted to the 2022 competition). For example we tried the combination between BM25 and BERT in order to obtain a final ranked list or simpler solutions in which current utterance was expanded with previous utterances. On the first chapter we will describe the background part, in which we will analyze what is a retrieval system, the main difference with a conversational retrieval system and we will describe *TREC CAsT*. On the second chapter we will describe some libraries and methodologies (with code examples) that can be used to build a conversational system. For example, we will talk about *BERT (Bidirectional Encoder Representations from Transformers)* and *NeuralCoref* (a python library that resolves

coreferences using a neural network) that allowed us to obtain our best result in the 2021 competition. On the third chapter we will describe the methodologies used to produce the runs submitted to *TREC CAsT* 2022, we will talk about the results obtained in the 2021 competition and, on the last chapter, we will analyze the results obtained in the 2021 competition.

# 2

# Background

Before starting to see the details of the project done, let's start from the basics in order to understand what a *Retrieval* is and the differences between a classic Information Retrieval and a Conversational Retrieval System.

## 2.1 Information Retrieval

*Information Retrieval* can be defined as a set of techniques that deal with organization, storage, retrieval and evaluation of information from document repositories (particularly textual information as web documents). An information retrieval process can begin when a user enters a query into the system. Queries are formal statements of information need, for example what a user writes in a normal search engine in order to obtain the weather information, the age of an actor or the last match of a football team [9]. In information retrieval a query doesn't uniquely identify a single object (structured entities) in the collection (like in a database query), several objects may match the same query with different degrees of relevance. Opposed to classical SQL queries, in information retrieval the results may be correct or not to the query, so results are typically ranked.

| | **Databases** | **IR** |
|---|---|---|
| `Unit` | Structured data | Mostly unstructured data |
| `Queries` | Formally defined queries | Vague and imprecise information needs |
| `Results` | Correct in formal sense | Sometimes relevant, often not |
| `Match` | Exact | Best match (ranked) |
| `Interaction` | one shot query | Interaction is fundamental |

Table 2.1: Difference between databases and IR

This ranking of results, with different degrees of relevance, is one of the main differences that characterizes an information retrieval system from a database system.

## 2.2 Conversational Information Retrieval

A *Conversational Information Retrieval* (CIR) system is an information retrieval (IR) system with a conversational interface which allows users to interact with the system to seek information via a multi-turn conversation of natural-language, spoken or written [6]. Conversational Information Seeking *(CIS)* [30] is concerned with a sequence of interactions between a user (or more) and an information system. Interactions are primarily based on natural language dialogue and, for more sophisticated systems, they may include interactions like click, touch or body gestures. A distinctive property of a *CIS* system is the ability of the systems to understand multi-turn interactions expressed in natural language. One important aspect in a conversational system is that a user's query can contains few information (which allow to match documents) and it can refer to previous utterance (or arguments discussed in previous interactions). Conversational Information Seeking is adopted in new generation of conversational assistant systems like: *Amazon Alexa*, *Cortana*, *Google Assistant* and many others.

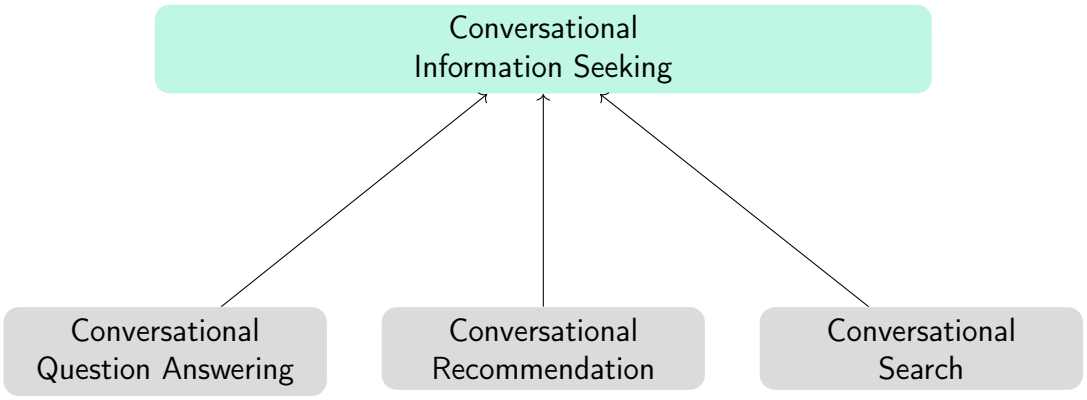



Figure 2.1: Conversational Information Seeking and example subdomains including conversational search, conversational recommendation, and conversational question answering [30].

The growing of this area in recent years is driven by two factors. The first is the ease of internet access in any part of the world: now people are doing almost everything on the web. The second factor is the rapid adoption of new conversational assistants installed on smartphone and devices of all kinds.

### 2.2.1 Conversational Search

*Conversational Search* refers to the use of complete sentences and other natural language phrases in search queries and how those queries are used in search engines using artificially intelligent algorithms. Historically, most internet searches were based on keyword phrases, such as "Italian food" or "Microsoft Corporation"; conversational searches use grammatical and syntactical patterns that closely resemble the way people talk. Conversational Search, or the process of interacting with a conversational system to search for information, is a popular research area and an important new frontier in *IR*. With the rise in machine learning (*ML*) and natural language processing many user's statements written in natural language are becoming more feasible, for example, the evolution of natural language processing brings improvements in the query rewriting of an

information retrieval . One of the main challenges to Conversational Search is that the system must take into account the evolution of the whole conversation. As we can see on Table 2.2, the system must "understand" a natural language utterance and represent the current state based on its knowledge, this step is required in order to build a more detailed query that can be used to retrieve relevant documents/answers to the user's information need.

| | |
|---|---|
| *USER* | How do genes work? |
| *SYSTEM* | A gene is a short piece of DNA. Genes tell the body how to build specific proteins... |
| *USER* | What others are caused by a single change? |
| *SYSTEM* | The four types of human Genetic diseases are: (1) Single-gene/monogenic Genetic Diseases... |
| *USER* | ... |

Table 2.2: CaST 2021, topic 113

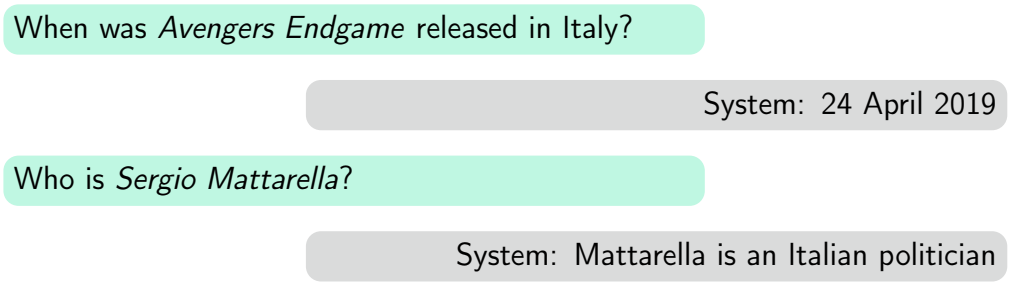
### 2.2.2 Conversational Recommendation

On Figure 2.1 we can notice that a conversational information seeking has a subdomain called: *Coversational Recommendetion*. On this section we introduce the general concept in order to explain the image. Recommender systems can be seen as information seeking systems that provide users with potentially relevant items based on historical interactions [30]. A recommender system uses the past user-item interactions as a way to produce (and re-rank) relevant information and aims to help users in order to select items for their information need, often in a closed domain such as books, resturants or movies.

### 2.2.3 Conversational Question Answering

Conversational Question Answering aims to provide one or more answers to a given question, it is an important research area in information retrieval *(IR)*

and natural language processing *(NLP)* communities [30]. Conversational question answering is a branch of conversational information seeking *(CIS)* and the user's need is expressed in a natural language (as in conversational search). In contrast to classical information retrieval *(IR)* systems, in which full documents are considered relevant to the user's need, conversational question answering aims to find short pieces of information to answer the queries. Therefore, this system uses *NLP* and *IR* techniques to retrieve small pieces of text that exact answer the queries (instead of the classic document list returned by IR systems). On Chapter 4 we will use some QA models in order to answer to the utterances during the conversation: these answers can be used during the query expansion or query rewriting phase.




## 2.3 TREC CAsT

*TREC* 2022 is the thirty-first edition of the ***T**ext **RE**trieval **C**onference* [7] in which the main goal is to create the evaluation infrastructure required for large-scale testing information retrieval *(IR)* technologies. Each TREC is organized around a set of focus areas called *tracks* [22], in this work we focus on **Conversational Assistance Track (CaST)**. This one is recent compared the other tracks, it started in 2019 and it focuses on creating large-scale reusable test collections for open-domain conversational search [3]. The document collection of this track is the union between:

- **MS MARCO** [13];

- **KILT** (benchmark version of Wikipedia) [8];
- **WAPO** (Washington Post collection) [28];

for a total of approximately 18 million of documents.

The goal of the task is to satisfy a user's information need expressed as a multi-turn conversational queries *(u)* for each turn:

$$T = u_1, u_2, u_3, u_4, ..., u_n. \tag{2.1}$$

On this work we talk more about CAsT 2021, the results of 2022 cannot yet be analyzed. CAsT 2021 has 26 information needs (topics) with an average length of 9.2, for a total of 239 turns; CAsT 2022 has 17 information needs, for a total of 205 turns. The main difference between the two competition is the structure of the topics:

- in 2021 each topic is a simple list of utterances expressed in natural language;
- in 2022 the topics are structured as a tree in which each level corresponds to one subtopic;

On a conversation we adopt the terminology *utterance* to refer a user's information need and we use the terminology *query* to refer the rewritten, expanded and reworked utterance used by the system in order to retrieve relevant information. All contiguous utterance from a single speaker form a single *turn*. An example of a 2021 topic is shown in Table 1.1.
Another big difference between CAsT 2021 and 2022 is the way to return relevant information:

- in 2021: the result is a list of relevant documents. The entire document is used without any division into passages, we split the document into passages (if you want, it depends on how a developer wants to reprocess

the documents) only to extract a response to the user (e.g. summarization of the passage);

- in 2022: documents are split into passages of length 250 words (there was a tool provided by CAsT) and the result is a ranked list of textual passages that can be used as a response to the user. The response to the user can come from a single document or from several, therefore, for each response is defined the source (corresponding to the ID of the document and to the ID of the passage);

In CAsT 2021 and 2022 there are three categories of run based on the data used in the testing phase [3]:

- *Manual*: Runs that use the manually rewritten (resolved) context-free queries, these use manual human rewritten queries;
- *Automatic-Canonical*: Automatic runs that use the provided automatic canonical system responses;
- *Automatic-Raw*: Runs that only use the provided raw conversational queries, these use raw utterances (with automatic rewrite/expansion) methods.

On this work we see the last category and we evaluate the system based on the same measure used by CaST: *Recall (Recall@500), Mean Average Precision (MAP@500), Mean Reciprocal Rank (MRR) and Normalized Discounted Gain (NDCG@500 and NDCG@3)*, we will see better this measures on the next chapters.

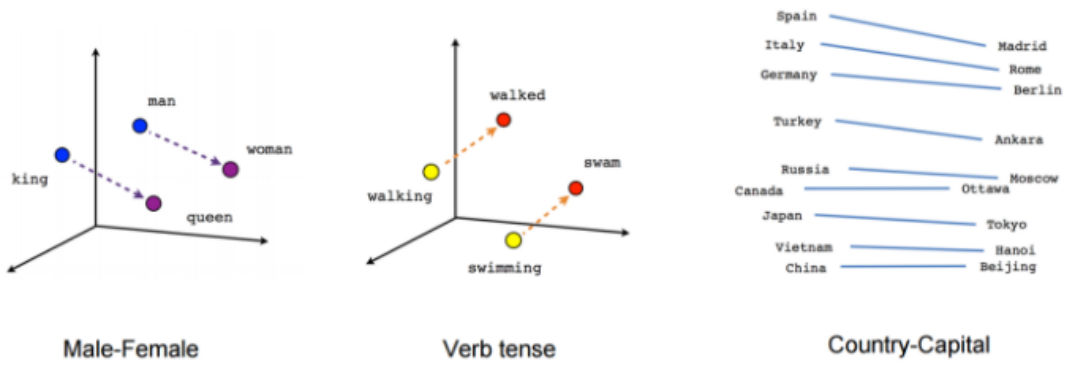



Figure 2.2: Example of word embedding, image by [29].

## 2.4 Word Embeddings

On the next chapter (Chapter 3) we will talk about *Word embeddings*, let's describe in detail what they are. *Word Embedding* is one of the most popular representation of document vocabulary. It is able to capture context of a word in a document, semantic and syntactic similarity, relation with other words, etc. This term indicates a set of modeling techniques in which words or phrases in a vocabulary are mapped into vectors or real numbers. Samples of textual data, also called *corpora*, are converted into numerical vectors though a two step process: the first is *tokenization* where words (or *n-grams*) will constitute the vocabulary of the dataset. The second is *vectorization* where the characteristics (of the tokens) will be assigned to a numerical measure. Vector representations of tokens, called *Word Embeddings*, are learned from NLP models, defined as *vector space* models which are based on neural network architectures. *Word2Vec* is one of the most popular technique to learn word embeddings using neural network, it was developed by *Tomas Mikolov* at Google. Word2Vec is a set of models used to produce *word embeddings*, in which the first release was in *C* but later also created for *Python* and *Java*. Word2Vec is a simple neural network with two levels and created to process natural language: the algorithm requires a corpus as input and returns a set of vectors representing the semantic distribution of words in the text.

## 2.5 BM25

In information retrieval **Okapi BM25** is a ranking function used by search engines to estimate the relevance of a document to a given search query. BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document. Given a query $Q$, containing keywords $q_1$, $q_2$, ..., $q_n$, the BM25 score of a document $D$ is [11]:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D)(k_1+1)}{f(q_i,D) + k_1(1 - b + b\frac{|D|}{avgdl})} \tag{2.2}$$

where:

- $f(q_i, D)$ is $q_i$'s term frequency in the document $D$;
- $|D|$ is the length of the document D;
- $avgdl$ is the average document length in the text collection from which documents are drawn;
- $k_1$ is a free parameter;
- $b$ is a free parameter;
- $IDF(q_i)$ is the **IDF** *(inverse document frequency)* weight of the query term $q_i$.

There are other similarities that you can use but, on this work, we will focus on **BM25**.

## 2.6 Performance measures

An important aspect in the development of a Conversational System (or Information Retrieval System) is to check the performances of the runs. On this section we will describe some measures used on this work, the same evaluations are used in *TREC CAsT* (Conversational Assistance Track). On *TREC 2021* [3] the following measures were used to evaluate each run:

- NDCG@500
- RECALL@500
- PRECISION@500
- MRR
- NDCG@3 (the one we tried to maximize at each run)

### 2.6.1 NDCG

To understand NDCG *(Normalized Discounted Cumulative Gain)*, we need to understand its predecessors: Cumulative Gain (CG) and Discounted Cumulative Gain (DCG). **Cumulative Gain** is the sum of all the relevance scores in a recommendation set (the set of retrieved documents):

$$CG = \sum_{i=1}^{n} rel_i \tag{2.3}$$

where $rel_i$ defines the graded relevance of the result at position $i$.
Let's see an example: suppose that an information retrieval system, for a specific query, returns six relevant documents:

$$D_1, D_2, D_3, D_4, D_5, D_6$$

and you know their relevance scores:

```
3,2,3,0,1,2
```

the Cumulative Gain will be:

$$CG_6 = \sum_{i=1}^{n} rel_i = 3 + 2 + 3 + 0 + 1 + 2 = 11 \tag{2.4}$$

The value computed with CG is unaffected by changes in the ordering of search results. In fact, moving a highly relevant document $d_i$ above a less relevant document $d_j$ the result is exactly the same. Based on this assumption DCG is preferred since it takes into account the position of the document on the ranked list, where the traditional formula of Discontinued Cumulative Gain is:

$$DCG_n = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)} \tag{2.5}$$

By using previous example, we find:

$$CG_6 = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)} = 3 + 1.262 + 1.5 + 0 + 0.387 + 0.712 = 6.861 \tag{2.6}$$

To normalize DCG in [0,1], you need to compute the ideal run (e.g. the run sorted in descending order of relevance) which rapresent the best retrieval possible and the maximum value of DCG. The **Normalized DCG** or $n$DCG will be:

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{2.7}$$

where $IDCG_p$ defines the maximum (ideal) value of $DCG_p$.

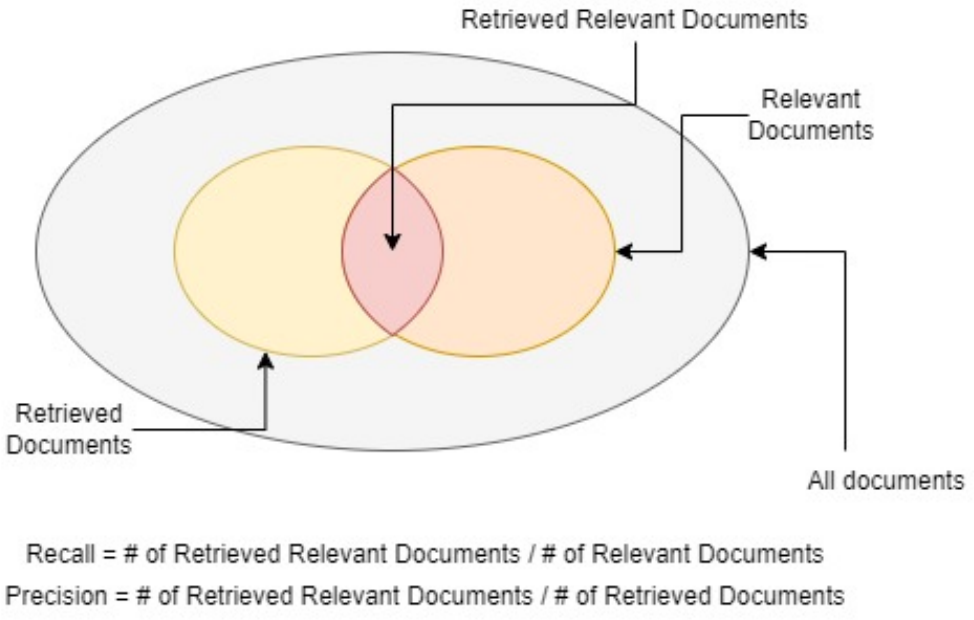



Figure 2.3: Precision and Recall measures

### 2.6.2 Precision and Recall measures

Let (observe Figure 2.3):

- **A** be the set of relevant documents;
- **B** be the set of retrieved documents;
- **A** ∩ **B** the set of retrieved relevant documents;

Precision is the fraction of the documents retrieved that are relevant to the users information need. Recall is the fraction of the documents that are relevant to the query that are successfully retrieved. Then:

$$Precision = \frac{|A \cap B|}{|B|} \tag{2.8}$$

$$Recall = \frac{|A \cap B|}{|A|} \tag{2.9}$$

### 2.6.3 MAP

Precision (is the fraction of the documents retrieved that are relevant to the user's information need) and Recall (is the fraction of the documents that are relevant to the query that are successfully retrieved) are single-value metrics based on the whole list of documents returned by the system. Let:

- $R$ be the set of the rank positions of relevant retrieved documents;
- $rr = |\mathrm{R}|$ be the total number of relevant retrieved documents;
- $N$ be the total number of retrieved documents;

the *Average Precision* is defined as:

$$AP = \frac{rr}{RB} \cdot \frac{1}{rr} \cdot \sum_{k \in \mathbf{R}} P(k) = \frac{1}{RB} \cdot \sum_{k \in \mathbf{R}} P(k) \tag{2.10}$$

where:

- $\frac{rr}{RB}$ is the Recall;
- $\frac{1}{rr} \cdot \sum_{k \in \mathbf{R}} P(k)$ is the arithmetic mean of $P(k)$

The **Mean Average Precision (MAP)** is the mean of AP over a set of topics.

### 2.6.4 MRR

The Mean Reciprocal Rank is a statistic measure for evaluating any process that produce a list of possible responses to a sample of queries. The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer: 1 for first place, $\frac{1}{2}$ for the second place, $\frac{1}{3}$ for the third place and so on. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries Q:

$$MRR = \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{2.11}$$

## 2.7 Background to analysis

Now we will describe two techniques used in statistics to prove whether the difference between two means is significant or due to chance alone. These methodologies will be used at the end of this work to show the difference in results between the various runs.

## 2.8 Student's T Test

The Student's t test is used to determine if there is a statistically significant difference between the means of two independent groups .
In the *student's t test*, two hypotheses are used:

- $H_0$ (or *null hypothesis*): the means of the two groups in the population are equal to each other (or the difference between the means is equal to zero);
- $H_1$ (or *alternative hypothesis*): the means of the two groups in the population aren't equal to each other (or the difference between the means isn't equal to zero).

The null hypothesis can never be rejected with absolute certainty. This statistical technique allows you to estimate the probability of obtaining a difference between the values of the two means, this probability is called **p-value**.

Usually the following criterion is used:

- *p-value* >= $\alpha$ : we cannot reject the *null hypothesis*, the difference observed between the means of the two groups **is not** statistically significant;
- *p-value* << $\alpha$ : we can reject the *null hypothesis*, the difference observed between the means of the two groups **is** statistically significant;

Usually $\alpha$ is set to 0.05, the same was done on this work.

## 2.9 Anova

ANOVA *(ANalysis Of VAriance)* is a generalization of the *Student's T test*. In fact, both techniques are used for the comparison of mean values. One difference is that: the *Student's T test* allows you to compare only two groups, otherwise ANOVA allows to compare any number of groups.

There are several ANOVA techniques (e.g. One-Way, Two-Way, MANOVA), on this paper we will use the *One-Way Anova* test. As for the *Student's T test*, *ANOVA* is also based on a null hypothesis and an alternative hypothesis:

- $H_0$ (or *null hypothesis*): the means of the groups in the population are equal to each other

$$H_0 : \mu_1 = \mu_2 = ... = \mu_n \tag{2.12}$$

- $H_1$ (or *alternative hypothesis*): the means of the two groups in the population aren't equal to each other (there is at least one different mean).

The results of the analysis of variance are typically presented using a table and it includes:

- *Origin*: the origins of the variance, including the factor under consideration (in our case the lot), errors and totals;
- *DF*: degrees of freedom for each origin of the variance;
- *Deviance (SS)*: sum of squares for each origin of variance, together with the total of all origins;
- *Quadratic mean*: the sum of the squares divided by the relative degrees of freedom;
- *Statistic F*: the square mean of the factor (lot) divided by the square mean of the error;

- *Prob*> *F*: the p-value.

The p-value is used to test the validity of the null hypothesis and it is used in the same way as described for the *Student's t test*.

# 3

# Experimental setting and methodological tools

To build a conversational system we have to decide what types of libraries we want to use and analyze how we can retrieve relevant documents from a corpus. Once the system works, documents are retrieved in sorted order according to a score computing using the document representation, the query and a ranking algorithm. The difficult part in a conversational system is that the queries can be referred to previous topics (or previous responses) and the current utterance is written in a conversational way. This implies that these systems must resolve coreferences (pronouns for example) to construct a correct query. Before going into the details of the methodologies, in this section we will see a presentation of the datasets used on this work and some libraries that we used to find well results in *TREC CAsT* 2021. For each tool, neural model (e.g. QA models) and dataset, a detailed introduction and an explanatory example will be shown (in order to discuss concepts that will be treated in the next chapter, Chapter 4).

## 3.1 Datasets in TREC 2021/2022

On this section we will see a presentation of the datasets used on this work: we will see the structure and an example. For *TREC CAsT* 2021 and 2022 three collections was used:

- **MS MARCO** [13];
- **KILT** [8];
- **WAPO** [28];

Before introducing the collections we talk about the difference between *document ranking task* and *passage ranking task*, this is a fundamental argument since the datasets are different depending on the task to do. In *document ranking task* we retrieve a list of relevant documents according to a query, otherwise in *passage ranking task* we retrieve relevant passages (inside documents) which are relevant to the query. Basically these datasets (MSMARCO, KILT and WAPO) are already divided, there are datasets that contain the entire documents (for *document ranking task*) and datasets that contain the passages of the documents already divided and ready to use. On this work we use the first one since the main task of TREC CAsT 2021 (and 2022) is to retrieve relevant documents.

### 3.1.1 MS MARCO

*MS MARCO (Microsoft Machine Reading Comprehension)* is a large scale dataset focused on machine reading comprehension, question answering, passage ranking, keyphrase extraction and conversational search studies. First released at **NIPS 2016** [18], the current dataset has 1,010,916 unique real queries that were generated by sampling and anonymizing *Bing* usage logs. The dataset started off focusing on QnA but has since evolved to focus on any problem related to search [14]. Now the *MS MARCO* dataset consists of six major components [18]:

- **Questions**: these are a set of anonymized queries from Bing's search logs, where the user is looking for a specific answer;
- **Passages**: For each question, on average the dataset includes a set of 10 passages which may contain the answer to the question. These passages are extracted from relevant web documents;
- **Answer**: For each question, the dataset contains zero, or more answers composed manually by the human editors;
- **Well-formed Answers**: For some question-answer pairs, the data also contains one or more answers that are generated by a post-hoc review-and-rewrite process;
- **Document**: For each of the documents from which the passages were originally extracted from, the dataset includes: URL, body text and title;
- **Question type**: Each question is further automatically annotated using a machine learned classifier with one of the following segment labels:
  - numeric;
  - entity;
  - location;
  - person;
  - description;

There are two versions of MS MARCO: V1 and V2, the second one is the most up-to-date and it is the one that we used on this work. On this paper we used the document dataset (for *document ranking task)* [1], there was a difference in structure between the 2021 and 2022 collections (the 2022 collection also contained many more documents), on this paper we will see the first one .

[1] https://msmarco.blob.core.windows.net/msmarcoranking/msmarco_v2_doc.tar

It contains 11.9 million of documents taken from *Bing* searches [13] and it is structured as follows (on Code 3.1 an example):

- **documentId**
- **documentLink**
- **documentTitle**
- **documentText**

```
D59865 # documentId
http://www.medicalnewstoday.com/articles/37136.php # documentLink
What you need to know about breast cancer # documentTitle
Newsletter MNT - Hourly Medical News Since 2003Search Log in
    Newsletter MNT - Hourly Medical News Since 2003Search Login What
    you need to know about breast cancer Last updated Mon 27 November
    2017By Christian Nordqvist Reviewed by Christina Chun, MPHSymptoms
     Stages Causes Types Diagnosis Treatment Outlook Breast cancer is
    the most common invasive cancer in women, and the second main
    cause of cancer death in women, after lung cancer. # documentBody
```

Code 3.1: Example of MS MARCO document

### 3.1.2 KILT DATASET

*KILT* is a resource for training, evaluating and analyzing NLP models on Knowledge Intensive Language Tasks. KILT has been built from 11 datasets representing 5 types of tasks [8] [19]:

- Fact-checking (FEVER): the task aims to verify a claim against a collection of evidence. It requires deep knowledge about the claim and reasoning over multiple documents.;
- Entity Linking (AIDA CoNLL-YAGO, WNED-WIKI, WNED-CWEB): the task aims to assign a unique Wikipedia page to entities mentioned in text;
- Slot filling (T-Rex, Zero Shot RE): the goal of the task is to collect information on certain relations (or slots) of entities (e.g., subject entity Albert

Einstein and relation educated_at) from large collections of natural language texts;

- Open domain QA (Natural Questions, HotpotQA, TriviaQA, ELI5): the task aims to produce the correct answer for a question, without a predefined location for the answer;
- Dialog generation (Wizard of Wikipedia): the task aims to develop an engaging chatbot that can discuss a wide array of topics with a user.

KILT includes the test set for all datasets considered. On this paper we used the KILT knowledge source[2], it is based on the 2019/08/01 Wikipedia snapshot and contains 5.9M articles.

The dataset is structured as follows (on Code 3.2 an example):

- **id** $\Longrightarrow$ `int`
- **wikipedia_id** $\Longrightarrow$ `int`
- **wikipedia_title** $\Longrightarrow$ `String`
- **text** $\Longrightarrow$ `List<String>`

```
{
    {
        "id": "290",
        "wikipedia_id": "290",
        "wikipedia_title": "An example of title",
        "text": ["A\n", "is the first letter..",".."]"
    }
}
```

Code 3.2: Example of KILT document

[2] http://dl.fbaipublicfiles.com/KILT/kilt_knowledgesource.json

### 3.1.3 WaPo DATASET

*WaPo* is a dataset consisting of 728 thousand of news articles from the Washington Post [3] published between 2012-2020 [28].
The dataset is structured in this way (on Code 3.3 an example):

- **id** $\Longrightarrow$ `String`
- **articleUrl** $\Longrightarrow$ `String`
- **title** $\Longrightarrow$ `String`
- **contents** $\Longrightarrow$ `List<WapoContent>` where WapoContent is an object that contains the real text content and other informations.

```
{
    {
        "id": "b2e89334-33f9-11e1-825f-dabc29fd7071",
        "article_url": "https://www.washi...",
        "title": "Danny Coale, Jarrett Boykin ...,
        "contents": [{"content": "Colleges", "mime": "text ...
    }
}
```

Code 3.3: Example of KILT document

## 3.2 LUCENE

*Lucene* is an open source Java library providing powerful indexing and search features implemented by Doug Cutting. It is supported by Apache Software Foundation and is released under the Apache Software License. There are a lot

[3] The Washington Post is an American daily newspaper

of wrappers of the original library for *C#*, *C++*, *Python* and *PHP* and it has an extensive documentation. Alternatives to Lucene can be:

- **Java**
  - Terrier (http://terrier.org)
  - Galago (https://sourgeforge.net/p/lemur/wiki/Galago)
- **C++**
  - JassV2 (https://github.com/andrewtrotman/JASSv2)
  - XAPIAN (https://xapian.org)

While developing an IR system in Lucene, we may have to index the documents: indexes are data structures designed to support search by avoiding linear scans and giving us results in small time. This step is fundamental in order to guarantee an efficient and effective Information Retrieval system. Given its importance, different setups were tested in order to decide which one was the best. To index the documents we used a library (GSON) that allowed to parse the text and to create a specific object in Java. These objects will then be used by the Lucene *IndexWriter* to create the index data structure.
For each document in **MS MARCO**, we stored the document *id*, *title* and *text* by converting the GSON MSMARCO object to a Lucene document with the following function:

```
public static Document getLuceneDocumentFromMsMarco(MSMarcoDocument doc, String id)
{
    Document d = new Document();
    d.add(new StringField(Constants.ID, doc.documentId, Field.Store.YES));
    d.add(new StringField(Constants.DATASET, "MARCO", Field.Store.YES));
    d.add(new StringField(Constants.LUCENE_ID, id, Field.Store.YES));
    d.add(new BodyField(document.documentTitle));
    d.add(new BodyField(document.documentText));
    return d;
}
```

Code 3.4: From MS MARCO to Lucene document

For each document in **KILT**, we stored the document *id* and *text* by converting the GSON KILT object to a Lucene document with the following function:

```
public static Document getLuceneDocumentFromKilt(KiltBase document, String luceneID)
{
    Document d = new Document();
    d.add(new StringField(Constants.ID, document._id, Field.Store.YES));
    d.add(new StringField(Constants.DATASET, "KILT", Field.Store.YES));
    d.add(new BodyField(String.join(" ",document.text))); //store body
    //Note that document.text is an array of strings
    return d;
}
```

Code 3.5: From KILT to Lucene document

For each document in **WAPO**, we stored the document *id* and *text* by converting the GSON WAPO object to a Lucene document with the following function:

```
public static Document getLuceneDocumentFromWapo(WapoDocument document, String ID)
{
    Document d = new Document();
    d.add(new StringField(Constants.ID, document.id, Field.Store.YES));
    d.add(new StringField(Constants.DATASET, "WAPO", Field.Store.YES));
    d.add(new StringField(Constants.TITLE, document.title, Field.Store.YES));
    d.add(new StringField(Constants.LUCENE_ID, ID, Field.Store.YES));
    StringBuilder body = new StringBuilder();

    for(var content : document.contents) {
        if (content != null) {
            body.append(
                    String.format("%s %s %s\n",
                        content.content,
                        content.fullcaption,
                        content.blurb)
            );
        }
    }

    d.add(new BodyField(body.toString())); // store body

    return d;
}
```

Code 3.6: From WAPO to Lucene document

The structure was slightly modified for the 2022 competition, as the structure of the datasets was slightly different. In 2022, each document was split into 250 token-length passages as required in the guidelines, as the goal was to return document passages to the user. There was a tool provided by the organizers that split the documents in passages.
We used a *Custom Analyzer* to index the documents with the *StandardTokenizerFactory* and the following *TokenFilters* [23]:

- **LowerCaseFilterFactory**: it replaces all uppercase characters in their corresponding lowercase, if the token is "*Happy*" the result is "*happy*" (in lowercase);
- **ApostropheFilterFactory**: it splits the word into two or more tokens when there is an apostrophe, if the token is "*O'Reilly's*" the result is two tokens *"O"* and *"Reilly"*;
- **EnglishPossessiveFilterFactory**: it removes singular possessives from words;
- **WordDelimiterGraphFilterFactory**: if a word contains special characters like "*Hot-Spot*" the word is separated in two or more words without any special character;
- **StopFilterFactory**: it removes the stopwords, we used the *Snawball* format.

To index all the datasets with the *CustomAnalyzer* a machine with 32 Gb of RAM and an i5 11600K took 2 hour and 15 minutes for a total of 65.5 Gb of data to store (the index size was 11.1 Gb). For TREC 2022 we used the Blade Cluster [4] of the University of Padua for time and hardware reason . Once indexing is done, we move on the search phase. Searching inherits the work done in the indexing phase, in fact each topic enters the same pipeline followed by documents.

[4] https://clusterdeiguide.readthedocs.io/en/latest/Overview.html

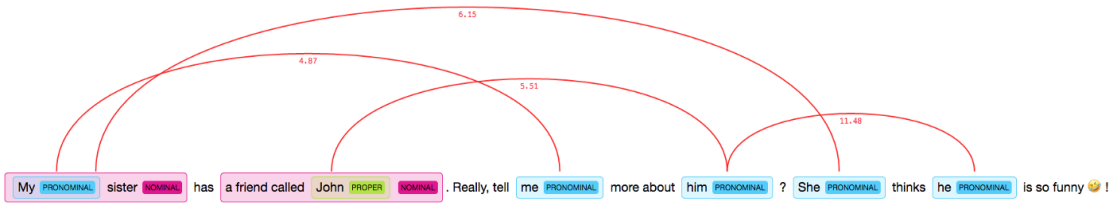



Figure 3.1: Neural coref resolution of the chat. Image from [17].

## 3.3 Neural Coref

*Neural Coref* [16] is a pipeline extension for *spaCy* and it is a free and open source library for **Natural Language Processing** *(NLP)* in *Python*. It allows to annotate and resolve coreference clusters using neural network. Let's see an example of possible coreference, let's have a look at *Bob* (green messages) who is talking with an *AI* system (grey message).

My sister has a friend called John

Really, tell me more about him

**She** thinks **he** is so funny

There are several implicit references in the last message from *Bob*

- **"she"** refers to the same entity as **"My sister"**: *Bob's sister*;
- **"he"** refers to the same entity as **"a friend called John"**: *Bob's sister's friend*;

Humans naturally associate these references together, but for a intelligent dialog system it is much more difficult.

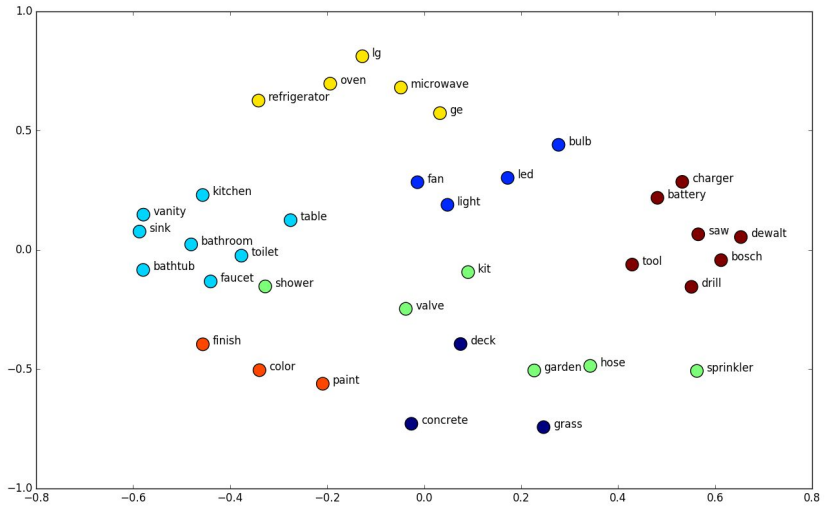



Figure 3.2: Example of word embedding. Image by *Shane Lynn* [29]

A typical coreference resolution algorithm goes like this:

- the algorithm extracts a series of mentions - word that are potentially referring to real world entities;
- for each mentions and each pair of mentions, we compute a set of features (e.g. Word2Vec). On this step we talk about *word embedding*, we already seen the concept on the previous chapter;
- then, we find the most likely antecedent for each mention based on this set of features. This last step is called *pairwise* ranking.

We decided to use this library because in *TREC CAsT*'s topics there are many coreferences that need to be resolved in order to build a correct query for the system. Each utterance in TREC is expressed in a conversational way and, during the conversation, there are a lot of pronouns that refer to previous entities. The idea was to use *NeuralCoref* to reformulate the utturences based on the previous responses and questions. Generally, the library automatically replaces pronouns or automatically resolves references but, sometimes, it leaves the sentences as they are because it is unable to establish which, of the available entities, is mostly

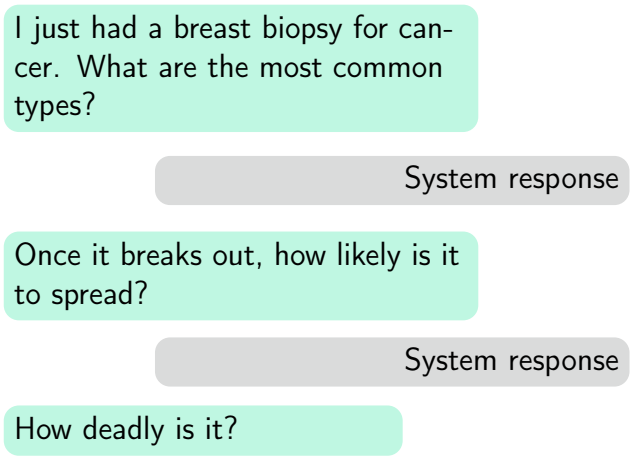



Figure 3.3: Topic 106, TREC CaST 2021

correct. In this case we created an ad-hoc script that replaces each pronoun with the entity that obtained the highest score from NeuralCoref (obviously checking that the entity is not the pronoun itself). On Figure 3.3 you can find an example in which NeuralCoref can be used to replace the various pronouns.
An alternative to Neural Coref can be *AllenNLP* [5], a complete platform for solving natural language processing tasks in PyTorch.

## 3.4 BERT

*BERT (Bidirectional Encoder Representation from Transformers)* is a Machine Learning model based on transformers, it was developed by Jacob Devlin and colleagues from Google for NLP applications. The BERT architecture is composed by several Transformer encoders stacked together where each encoder is composed of two sub-layers: a feed-forward layer and a self-attention layer. BERT uses of a *Transformer* that learns contextual relations between words in a sentence/text. This model was used by many teams in last TREC CAsT competitions giving extraordinary results on performance measure like Recall, NDCG@3 and

[5] https://allenai.org/allennlp/software/allennlp-library

Precision. The idea was: although Lucene returns very good results we can improve the first results of the ranked list by re-ranking the top $k$ relevant documents by using the BERT Classification. Using this classifier it is therefore possible to say with certainty if a document is relevant or not with respect to utterance by using a neural network .

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

model = AutoModelForSequenceClassification.from_pretrained('cross-encoder/ms-marco-TinyBERT-L-2-v2')
tokenizer = AutoTokenizer.from_pretrained('cross-encoder/ms-marco-TinyBERT-L-2-v2')

features = tokenizer(['How many people live in Berlin?', 'How many people live in Berlin?'], ['Berlin has a population of 3,520,031 registered inhabitants in an area of 891.82 square kilometers.', 'New York City is famous for the Metropolitan Museum of Art.'], padding=True, truncation=True, return_tensors="pt")

model.eval()
with torch.no_grad():
    scores = model(**features).logits
    print(scores)
```

Code 3.7: Example of BERT text classification

On the Code 3.7 [2] there is a question *"How many people live in Berlin?"* proposed to two text:

- *"Berlin has a population of 3,520,031 registered inhabitants in an area of 891.82 square kilometers";*
- *"New York is famous for the Metropolitan Museum of Art".*

The second text has nothing to do with that question, humans naturally associate the first question as more relevant for it but, for an *AI* system, it is much more difficult. What BERT does is classify texts in the same way as a human person (by using a neural network).

By running the script 3.7 BERT will give you the following results:

- 7.2358 for the first text
- -11.5623 for the second text

BERT assigns a score that goes from -20 to +20, higher scores indicate that the text is more relevant to the question.
On this work we used a pre-trained model: *cross-encoder/ms-marco-TinyBERT-L-2-v2* that is very fast. In Table 3.1 we reported others models that can be found in *HuggingFace* with the various performances (models are trained on MSMARCO passages dataset).

| **Model-Name** | **NDCG@10 (TREC DL 19)** | **MRR@10 (MS Marco dev)** | **docs/sec** |
|---|---|---|---|
| *cross-encoder/ms-marco-TinyBERT-L-2-v2* | 69.84 | 32.56 | 9000 |
| *cross-encoder/ms-marco-MiniLM-L-2-v2* | 71.01 | 34.85 | 4100 |
| *cross-encoder/ms-marco-MiniLM-L-4-v2* | 73.04 | 37.70 | 2500 |
| *cross-encoder/ms-marco-MiniLM-L-6-v2* | 74.30 | 39.01 | 1800 |
| *cross-encoder/ms-marco-MiniLM-L-12-v2* | 74.31 | 39.02 | 960 |

Table 3.1: Pre-trained Cross-Encoders with performance on the TREC DL 2019 and the MS Marco Passage Reranking dataset [2].

On the project we tried different models but, at the end, we decided to use *cross-encoder/ms-marco-TinyBERT-L-2-v2* because it is faster and performance were comparable to others.

One important aspect, which we will see on the next chapter (Chapter 4), is that the results can be influenced by the length of the text passed to the classifier:

- **short text** contains not enough information and BERT can classify the text as relevant even if it is not
- **long text** contains too much information and BERT can classify the text as not relevant even if it is

## 3.5 Transformers for question answering

Question answering models can retrieve the answer to a question from a given text by using a neural network. This type of operation can be useful for searching on our documents and answers to our utterances. Since in TREC, in each topic, the first utterance is a question, we decided to use the *AutoModelForQuestionAnswering* [21] module of *transformers*. This module is able to answer to questions on a given text. Generally, the answer to the first question and the first utterance itself (on each topic) define the main argument of the conversation, we can use that answer in Lucene to increase the matches on the index or we can use that response in NeuralCoref to solve the various coreferences. An example of QA script based on transformers and hugging face is the following:

```
from transformers import pipeline

qa_model = pipeline("deepset/roberta-base-squad2")
question = "Where do I live?"
context = "My name is Merve and I live in stanbul."
qa_model(question = question, context = context)
## result is {'answer': 'stanbul', 'end': 39, 'score': 0.953, 'start': 31}
```

Code 3.8: Example of QA (HuggingFace)

On the project we used a pre-trained model: "*deepset/roberta-base-squad2*", it was originally trained on another dataset which we never mentioned: *SQuAD*

(Stanford Question Answering Dataset).

SQuAD [24] is a reading comprehension dataset consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text (or *span*) from the corresponding reading passage, or the question might be unanswerable.

Although the model was not trained in TREC CAsT's datasets (MSMARCO, WAPO or KILT) the results were good in our work, this model was useful for solving coreferences with NeuralCoref: the answer of each utterance can be used to solve some pronouns.

## 3.6 NLTK: Sentence Tokenizer

Tokenization is the process in which a large quantity of text is diveded into smaller parts called tokens. We already saw a tokenizer (Section 3.2) on the previous section for Lucene. Natural Language Toolkit, also known as **NTLK** [15], is a suite of libraries and programs (written in Python) for symbolic and statistical analysis in the field of natural language processing. It provides easy-to-use interfaces with over 50 corpora and lexical resources as WordNet, along with a suite text processing libraries for classification, tokenization, stemming, tagging and parsing.

To use BERT correctly (e.g for re-ranking or for QA models) we have to split the document into passages. This part is very important and we will see on the next chapter that the length of a passage can greatly effect the results in BERT. Passing the whole document to the classifier is not a good idea because it contains a lot of information that can be relevant or not to the utterance, this solution may give wrong results specially because an utterance refer to small pieces of a document. Therefore, to split the document into passages, we used a module called *sent_tokenize* of the NTLK library. This module allows to you to split long texts into small passages or sentences, text is usually split when

characters like: ".", "!" or "?" are found (we used a regex to remove special characters like "{", "}", etc., BERT with special characters doesn't work well). Once we split the document into passages we can obtain a score of each passage (for a given query) by using BERT and then returning the maximum score. This score can be used to re-rank documents (since it defines if there exist a passage in the document that is highly relevant for the query). What we do is re-rank the documents based on the passages that have obtained a higher score on the current query.

```
from nltk.tokenize import sent_tokenize
text = "God is Great! I won a lottery."
print(sent_tokenize(text))

#Output: ['God is Great!', 'I won a lottery ']
```

Code 3.9: Example of sentence tokenization

We remember that we didn't store the whole document on the index. To obtain a document, given the *ID* (stored in the Lucene's index), we created a dictionary that contains all the offsets (of each document) in each file; in this way we can obtain a document in fast time without reading the whole file. This dictionary was saved on a file so that it can be used when needed [20].

## 3.7 Lucene OpenNLP

Since topics are expressed in a conversational way, to improve the performance on the searching phase of Lucene, we decided to use the *OpenNLP* analyzer. The Apache OpenNLP library is a machine learning based toolkit for processing of natural language text, it supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging and coreference resolution [1]. These task are usually required to build more advanced text processing services.

```
1  I           PRON    PRP     pronoun, personal
2  like        VERB    VBP     verb, non-3rd person singular present
3  to          PART    TO      infinitival to
4  play        VERB    VB      verb, base form
5  football    NOUN    NN      noun, singular or mass
6  .           PUNCT   .       punctuation mark, sentence closer
7  I           PRON    PRP     pronoun, personal
8  hated       VERB    VBD     verb, past tense
9  it          PRON    PRP     pronoun, personal
10 in          ADP     IN      conjunction, subordinating or preposition
11 my          ADJ     PRP$    pronoun, possessive
12 childhood   NOUN    NN      noun, singular or mass
13 though      ADP     IN      conjunction, subordinating or preposition
```

Code 3.10: Example of Part-of-Speech with Spacy (Python)

## 3.8 Spacy

Spacy is an open-source software library for advanced natural language processing, written for *Python*. Unlike **NLTK**, which is widely used for teaching and research, Spacy supports deep learning workflows that allow connecting statistical models trained by popular machine learning libraries like *TensorFlow* and *PyTorch*.

During the evolution of the project, we used the *WordNetLemmatizer* of Spacy in order to help BERT to resolve difficult words on embedding. Let's consider an example. During the project we saw that the word "*deadliness*" (TREC CAsT, topic 106, turn number 4) caused incorrect re-rankings, which didn't happen if the word "*deadly*" was used. In order to solve this problem we decided to help BERT by passing a query where a lemmatization procedure was already applied.

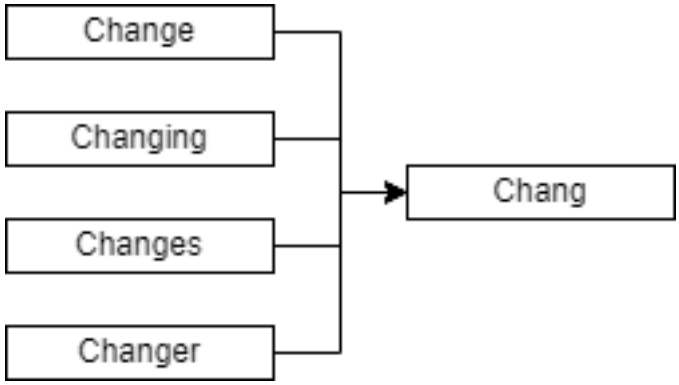



Figure 3.4: Stemming procedure

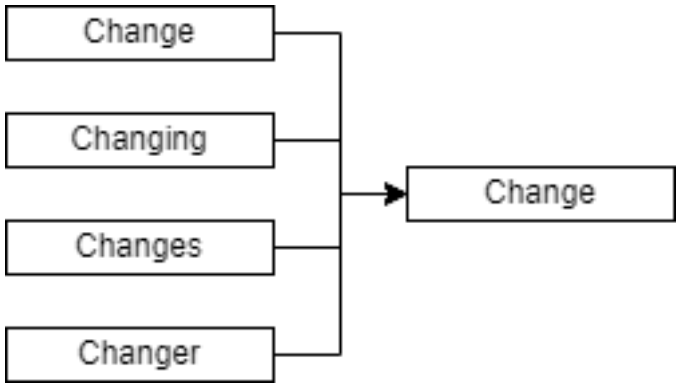



Figure 3.5: Lemmatization procedure

*WordNetLemmatizer* already contains a list of of possible substitution for a word but, in order to have better results, we extended this list with the "*lemmatization-en.txt*" file [6]. This procedure allowed us to improve the final scores.

Another Spacy's module used on the project was the *Part-of-speech tagging* (POS) model. We decided to use this module to obtains all the nouns of the first utterance of a conversation. This nouns can be passed to BERT in order to give the general context (usually contained on the first utterance) of the conversation.

[6] https://github.com/michmech/lemmatization-lists/blob/master/lemmatization-en.txt

Let's see an example (first utterance of topic number 106, *TREC CAsT* 2021) [25]:

*I just had a breast biopsy for cancer. What are the most common types?* (3.1)

The general context of the topic 106 (TREC CAsT 2021) [25] refers to *Breast Cancer*. With Spacy, we can obtain from the previous utterance (3.1) all the nouns (*breast*, *biopsy*, *cancer* and *types*) and passing this information on each of the next queries passed to BERT. This can be useful when the general information is missing on the utterance. An example of POS tagging can be found on the table 3.10.

### 3.8.1 Trec Eval

On the previous chapter (Chapter 2) we talked about measures. All measures can be computed by using `trec_eval` [26]: a standard tool used by the TREC community for evaluating an ad hoc retrieval run, given the results file and a standard set of judged results. The evaluation is based on two files:

- *qrels*: relevance judgements for each query;
- *run*: it contains the ranking of documents returned by an IR/Conversational system.

When evaluate measures with Trec Eval, you can obtain all the scores by passing the *all_trec* parameter. To obtain a specific score you can use the name of the measure you want as parameter, below and example to obtain NDCG@3.

```
./trec_eval -q -m ndcg_cut.3 results/qrels.txt results/run.txt
```

# 4

# Techniques

On this section we will analyze the various techniques adopted to solve the *Conversational Assistance Track* of TREC 2021, the same techniques will be then used to produce a run for TREC 2022. In this chapter we will start from the most basic solution (using *state of art*) to a little more complex solutions. For each proposed technique you will find details and comments, the analysis will be done on the last chapter.

The study started from some papers of TREC competition of the last year in which, one in particular [12], caught our attention. On this paper there were some techniques that the authors tried to use in previous competitions of TREC CAsT; the idea was to use this methods and, in the same time, try to improve the performances. One in particular was very interesting because it could solve many problems on NeuralCoref. Since in a conversational dialog an utterance can refer to the previous answers or to previous interactions, the idea of the authors was to classify each utterance with three labels:

- **SE**: classification label for utterances that are *Self Explanatory*;
- **FT**: classification label for utterances referring to the *First Topic*;
- **PT**: classification label for utterances referring to *Previous Topic*.

They trained a neural network in order to classify each utterance with the corresponding label. This classification method allows to the system to decide the way to rewrite an utterance during the conversation. Given an utterance classified as self-explanatory (SE): the rewriting is not necessary since all information are already contained on the query but, when the utterance is classified as First Topic (FT) or Previous Topic (PT), one can decide to rewrite the utterance: by concatenating the strings or by using NeuralCoref (or *AllenNLP* as the authors). Starting from this consideration the authors developed several method including: *First topic* and *ContextQuery*.

## 4.1 First query technique

The *First Topic* method is very simple to implement, it doesn't require to build any sophisticated approach, given a conversation: the current query $q_i$ is expanded with the first turn utterance.
Equation 4.1 defines the method used to obtain the new query:

$$u_i = q_0 + q_i \tag{4.1}$$

where:

- $u_i$ is the new query that you can pass to any information retrieval system (e.g. Lucene);
- $q_0$ is the first utterance of the conversation;
- $q_i$ is the current utterance.

This method, although it may seem of simple implementation, demonstrates the importance of using the first utterance on query rewriting, it contains important information that can be useful to match documents. As we have already seen on previous chapters, the first utterance (and answer) define the main topic of a conversation. The performance measures of this run are reported on Table 4.1.

| **Technique** | **NDCG@500** | **Recall@500** | **MAP@500** | **NDCG@3** |
|---|---|---|---|---|
| *First topic* | 0.2688 | 0.4366 | 0.0917 | 0.1608 |

Table 4.1: Results of First Query technique

## 4.2 Context query technique

In the second methodology, we included the antecedent utterance to the First Topic technique. Given a conversation, the current query $q_i$ is expanded with the first utterance of the conversation and the previous one:

$$u_i = q_0 + q_{i-1} + q_i \tag{4.2}$$

where:

- $u_i$ is the new query that you can pass to any information retrieval system (e.g. Lucene);
- $q_0$ is the first utterance of the conversation;
- $q_{i-1}$ is the previous utterance.
- $q_i$ is the current utterance.

This method considers the previous utterance in order to obtain documents which are more relevant to the query (the previous utterance usually contains information that is also relevant to the current utterance). On Table 4.2 we can find the performance measures of this run.

| **Technique** | **NDCG@500** | **Recall@500** | **MAP@500** | **NDCG@3** |
|---|---|---|---|---|
| *Context query* | 0.2954 | 0.4627 | 0.1051 | 0.1858 |

Table 4.2: Results of ContextQuery technique

## 4.3 Technique with BERT and NEURALCOREF

We have already seen on the previous chapters what is NeuralCoref, now we will see the configuration, the various problems and results (the entire pipeline is shown on the figure 4.3). Our project is based on two languages: *Python* and *Java*, in which:

- **Python** is used for BERT re-ranking, QA and NeuralCoref resolutions;
- **Java** is the main program: it uses Lucene to obtain a ranked list by using BM25 similarity, it parses the topics and it calls the python scripts when necessary;

Our program starts by initializing Lucene: the *Searcher* and *Indexer* classes.
The Indexer class contains all the code that allow Lucene to parse each document and to analyze them using the *CustomAnalyzer* described in Section 3.2.
The Searcher class contains all the code that allow to Lucene to search in the index data structure and then to return a ranked list of relevant documents (for the query) based on BM25 similarity. On this class we tried to use a *CustomAnalyzer* and an *OpenNLPAnalyzer* but, at the end, we decided to use the *OpenAnalyzer* because it returned better results. The entire process of adding **NeuralCoref** and **BERT** is done on the Searcher class and, in order to give you more details, you can find an example on Code 4.1 (below).

```
1  searcher = new Searcher() //initialize search class of Lucene
2  indexer = new Indexer() //initialize index class of Lucene
3
4  //custom analyzer used in the indexing phase
5  CustomAnalyzer.Builder indexAnalyzer = CustomAnalyzer.builder()
6                      .withTokenizer(StandardTokenizerFactory.class)
7                      .addTokenFilter(LowerCaseFilterFactory.class)
8                      .addTokenFilter(ApostropheFilterFactory.class)
9                      .addTokenFilter(EnglishPossessiveFilterFactory.class)
10                     .addTokenFilter(WordDelimiterGraphFilterFactory.class)
11                     .addTokenFilter(StopFilterFactory.class, "snawball");
12
```

```
13 //analyzer used in the searching phase
14 OpenNLPAnalyzer openNLPAnalyzer = new OpenNLPAnalyzer();
15
16 //load topics with GSON library
17 List<Topic> topics = parseTopic();
18
19 for(Topic topic : topics){
20     boolean isFirst = true; //to save the first utterance (query expansion)
21     String first_utt = ""; //first utterance
22     String first_resp = ""; //first response (using QA models)
23
24     String prev_utt = ""; //previous utterance
25     String curr_utt = ""; //current utterance (or current query)
26     String curr_resp = ""; //current response (using QA models)
27
28     for(Turn turn : topic.turn){
29         curr_utt = turn.utterance;
30
31         //call NeuralCoref (python script)
32         //try to obtain a new utterance with resolved coreferences
33         currentTopic = getResolvedTopic(prev_utt, curr_resp, curr_utt);
34
35         //call Lucene Searcher (BM25 similarity, OpenNLPAnalyzer)
36         //use computed currentTopic on the searching phase
37         searcher.search(currentTopic,openNLPAnalyzer)
38
39         //this method returns a ranked list on a file (temporary_run.txt)
40
41         //re-rank of the first 200 documents using BERT
42         curr_resp = bertReRanking(currentTopic)
43
44         //curr_resp = answer to currentTopic by using QA models
45
46         //update temporary variables
47         if(isFirst){
48             first_utt = currentTopic;
49             first_resp = curr_resp;
50             isFirst = false;
51         }
52         //now we update variables
53         prev_utt = currentTopic;
54     }
55 }
```

Code 4.1: Procedure to rewrite an utterance

On the Code 4.1 we can observe that we passed a computed *currentQuery* to the Lucene Searcher, this variable can be used to produce different techniques (depending on how we populate this variable):

- **FirstQuery** with Neural Coref and BERT solution;
- **ContextQuery** with Neural Coref and BERT solution.

When *Lucene* returns the result (a ranked list saved in a file), BERT is applied in order to re-rank the top $k$ documents of the list (we re-rank only the first 75 documents). We tried different values of $k = 30, 50, 75, 100$ and, at the end, we decided the best one comparing the results.

### 4.3.1 NeuralCoref

On the method *getResolvedTopic* (Code 4.1), three temporary variables are passed to the python script (that generates the current query passed to Lucene):

- *previous utterance*;
- *previous response*;
- *current utterance*.

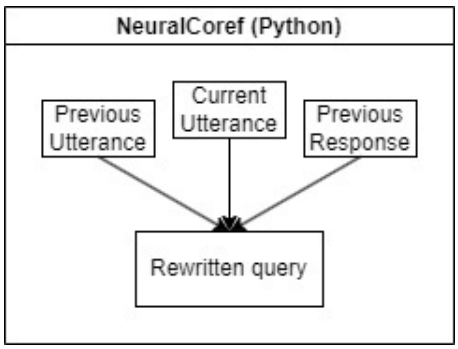



Figure 4.1: NeuralCoref query rewriting

This three variables are used by NeuralCoref to produce a new query on this way:

1. first of all, the strings are concatenated to produce one big query:

$$\texttt{query} = \texttt{previous utterance} + \texttt{previous response} + \texttt{current utterance}$$

2. *SpaCy* and *NeuralCoref* are used to produce a POS tagging and a cluster of terms/sentences;

3. now, the coreference resolution is done on the last part of the query (on the *current utterance* substring);

4. now, on the query, the *previous utterance* and *current response* are removed;

5. if the query contains other pronouns: we automatically substitute each pronoun with the most probable sentence/term (we used the scores returned by NeuralCoref);

6. if the query doesn't end with *"?"*: append the previous utterance to the query (the current utterance can be a clarification of the previous query)

7. return the computed query.

The complicated part on NeuralCoref was to choose the correct term/sentence when the library doesn't resolve the query automatically.
In this case we applied a greedy algorithm: always choose the best option with the highest score (that isn't a pronoun or the sentence/term itself) by using the dictionary returned by *NeuralCoref*. On Code 4.2 we can find an example of this algorithm.

```
1
2  #Given the query:
3  input = "I just had a breast biopsy for cancer. What are the most common types?"
4
5  #We want to find the best coreference for
6  coreferenceToFind = "the most common types"
7
8  result = neuralCoref.findSubstitute(input,coreferenceToFind)
9  print(result)
10
11 out = {the most common types: 1.8015064001083374, a breast biopsy for cancer:
       −1.557297945022583, cancer: −2.2499332427978516}
12
13 newQuery = neuralCoref.computeQuery(input,result)
14 print(newQuery)
15
16 out = What are the most common types a breast biopsy for cancer?
```

Code 4.2: Example of NeuralCoref

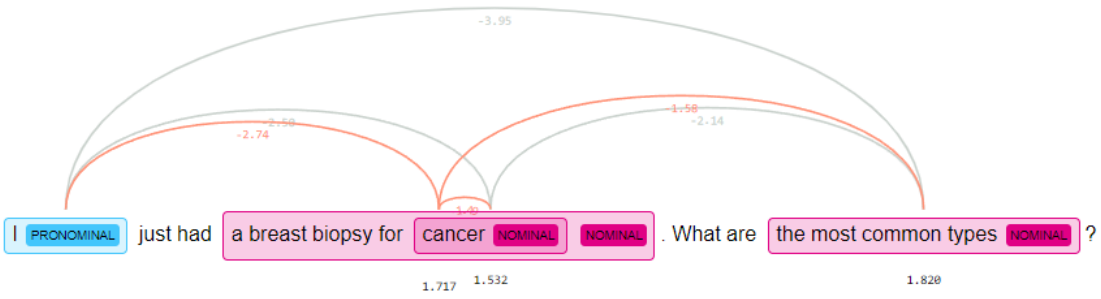



Figure 4.2: NeuralCoref graph of the Code 4.2

```
import spacy

# Load English tokenizer, tagger, parser and NER
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasnt "
        "worth talking to, said Thrun, in an interview with Recode earlier "
        "this week.")
doc = nlp(text)

# Analyze syntax
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

# Find named entities, phrases and concepts
for entity in doc.ents:
    print(entity.text, entity.label_)



OUT
# Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'the
    company', 'him', 'I', 'you', 'very senior CEOs', 'major American car companies', '
    my hand', 'I', 'Thrun', 'an interview', 'Recode']
# Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'talk', 'say']
# Sebastian Thrun PERSON
# 2007 DATE
# American NORP
# Thrun GPE
# Recode ORG
# earlier this week DATE
```

Code 4.3: Example of POS tagging with SpaCy

### 4.3.2 BERT

On Code 4.1, once the method *getResolvedTopic* is terminated, the computed query is passed to the Lucene searcher.
Here we tried several ways:

- **First query**: $q_0 + q_i$
- **Context Query**: $q_0 + q_{i-1} + q_i$
- **Context Query (with first answer)**: $q_0 + r_0 + q_{i-1} + q_i$
- **Context Query (with previous answer)**: $q_0 + r_{i-1} + q_{i-1} + q_i$

where:

- $\forall i \in \{1, 2, 3..., n\} \implies q_i$ is obtained by using NeuralCoref;
- $r_0$ is the answer to the first utterance of the turn
- $r_i$ is the answer to the previous utterance of the turn

When the Lucene searcher returns a ranked list, we re-rank the documents by using the BERT (text) classifier. The current query (as it is, without any other expansion except some nouns of the first utterance as specified above) is used to re-rank documents as follows:

1. we read the the ranked list (file) produced by Lucene and, for each document in the list, we obtain the corpus (this will be used to classify the text);
2. The corpus is split into sentences of length 56 tokens: we initialize chose this size because the MS MARCO passage dataset, used to train the model (see table 3.1), contained passages with this average length. Another fundamental aspect that led to choose this length was because QA models doesn't work well with short and long texts. Zhiguo Wang and other

colleagues of Amazon [27] proved that by splitting long texts into passages of medium length (50-100 words) improves the performance of QA models.

This length gave good results, but we will see that this length is not optimal. We have already seen on the previous chapter that this length can greatly affect the results. This operation (split document into passages) was not carried out in TREC CAsT 2022: the documents were already divided into passages.

3. Re-rank the first 75 documents of the run by using BERT classifier (we already seen that several parameters were tested and, at the end, we decided to use 75).

4. For each topic: we answer to the first utterance (or all, depending on the method used) of the conversation by using QA models (the response can be used on NeuralCoref for coreference resolution).

Now, we show that this procedure allows to increase the position of relevant documents, thus allowing to increase some import scores as **NDCG@3**.

Let's consider the query (TREC 2021, topic 106) [25]:

*How deadly is it?* `(it refers to "breast cancer")`

Before the re-ranking phase with BERT, Lucene returns the following ranked list:
*(the relevant document is at position 67)*:

| Topic_ID | Document_ID | Rank (pos) | Relevant |
|---|---|---|---|
| 106_3 | MARCO_D2757478 | 65 | False |
| 106_3 | MARCO_D1861932 | 66 | False |
| 106_3 | **MARCO_D3307814** | **67** | **True** |
| 106_3 | MARCO_D238615 | 68 | False |

Table 4.3: Results before BERT

When we apply BERT in order to re-rank the first 75 documents of the previous list, we obtained the following results:
*(the previous relevant document, MARCO_D3307814, has increased 13 positions)*.

| Topic_ID | Document_ID | Rank (pos) | Relevant |
|---|---|---|---|
| 106_3 | KILT_22096998 | 52 | False |
| 106_3 | MARCO_D3065945 | 53 | False |
| 106_3 | **MARCO_D3307814** | **54** | **True** |
| 106_3 | MARCO_D620300 | 55 | False |

Table 4.4: Results before BERT

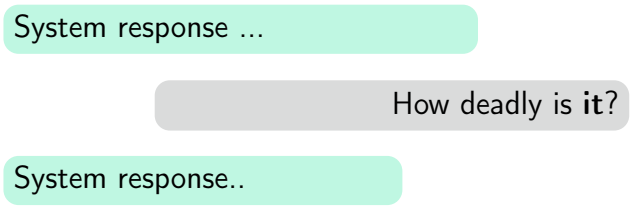
Through this methodology, we were able to significantly increase the scores, especially for the **NDCG@3**. As described in Section 4.3.2, different methods were tested using this algorithm and at the end we chose the best one: *Context Query (with previous answer)* (Section 4.3.2), the performance measures are reported on Table 4.5.

| **Technique** | **NDCG@500** | **Recall@500** | **MAP@500** | **NDCG@3** |
|---|---|---|---|---|
| NC, BERT, QA | 0.3715 | 0.5290 | 0.1652 | 0.2926 |

Table 4.5: NeuralCoref + BERT + QA performance measures

## 4.4 Technique with automatic resolved utterance

The query passed to BERT must be written in a correct way: it must contain all the necessary information and it must be written with a logical sense: it is not obvious when you use libraries like *NeuralCoref* to solve pronouns coreferences. As we will see in the next example, the same utterance written in two different ways causes the results to change dramatically.
Let's consider the topic 106 of TREC CAsT 2021 (turn number 3) [25]:




the pronoun "*it*" can refer to the response of the previous utterance or to the main argument of the previous utterance. Suppose that NeuralCoref replace the pronoun *"it"* with *"breast cancer"* (the topic of the previous utterance), in this case we obtain the Table 4.4 shown previously.
If the pronoun *"it"* is replaced with *"Lobular Carcinoma"* (response of the previous utterance) the result will be different, as it refers to two quite different contexts. In order to avoid this problem (if the algorithm fails the substitution: the resulting query is wrong) and increase the performance, we used the *automatic_resolved_utterance* provided by TREC CaST. This utterance is exactly the same, but written in a better way: it contains much more information or clarifications of the original. Starting from this concept, we decided to use a combination between *NeuralCoref* and the *automatic_resolved_utterance*: we decided to use this utterance as query when NeuralCoref gave us bad results (e.g. the resolved query was equal to the original utterance). During coreferences resolution with NeuralCoref we also try to use the *automatic_resolved_utterance* to solve various pronouns.
The code (described in Section 4.3) was then modified integrating this new feature in order to produce a new run, the performance measures are reported on Table 4.6.

| **Technique** | **NDCG@500** | **Recall@500** | **MAP@500** | **NDCG@3** |
|---|---|---|---|---|
| NC, BERT, QA (improved) | 0.4013 | 0.5574 | 0.1878 | 0.3291 |

Table 4.6: Run with automatic resolved utterance: performance measures

### 4.4.1 Last run: Technique without ContextQuery

During the developing of the project we noticed that there were a lot of utterances with scores (*NDCG@3*) 0 affecting negatively the total mean (see Table 4.7).
In a small set of this utterances the problem was simple: there were some misspelled apostrophes (wrong character) and BERT didn't recognize them. To solve this problem we replaced the characters with the correct ones.

| Topic_ID | NDCG@3 |
|---|---|
| 113_1 | 0.1480 |
| 113_2 | **0.0000** |
| 113_3 | 0.3827 |
| 113_4 | **0.0000** |
| 113_5 | **0.0000** |
| 113_6 | 0.4693 |
| 113_7 | 0.2346 |
| 113_8 | **0.0000** |

Table 4.7: Topic 113, a lot of utterances have score 0

During the developing of the context query technique we noticed that many results were relevant to the previous utterance but not to the current one. Since this problem came from Lucene, we decided to change the weight of the *previous utterance* in the *BooleanQuery* [1]. By giving less weight, we were able to give more importance to the current utterance and, in this way, at the top positions we only had documents relevant to the current one.
Another change we did was to use the the nouns of the first utterance of the conversation to give a general context to BERT when it was missing and, the same, was done for Lucene. Before we talked about the length of the text passed to BERT, we had previously used a length of 56 tokens but that length led to a bad classification: the text contained too little information to be classified

[1] https://lucene.apache.org/core/8_1_1/core/org/apache/lucene/search/BooleanQuery.html

correctly. We decided to increase this value to 250 tokens (as it was done in the tool provided by TREC CAsT 2022) noticing a big difference in the results. These changes to the code were also made to produce another run, on Table 4.8 the performance measures.

| **Technique** | **NDCG@500** | **Recall@500** | **MAP@500** | **NDCG@3** |
|---|---|---|---|---|
| NC, BERT, QA (no CQ) | 0.4105 | 0.5658 | 0.1922 | 0.3558 |

Table 4.8: Last technique without Context-Query

## 4.5 TREC 2022: SUBMITTED RUNS

Having participated in the 2022 competition, the only way to understand which runs to submit to TREC CAsT 2022 were based on the results obtained in the 2021 competition. After an analysis made in the results obtained and described in this work, we decided to submit to TREC CAsT 2022 the runs obtained with the following techniques:

- **NC, BERT, QA**: described on Section 4.3;
- **NC, BERT, QA (improved)**: described on Section 4.4;
- **NC, BERT, QA (no CQ)**: described on Section 4.4.1;
- **NC, BERT, QA (no CQ, LMDirichlet)**: is the same technique described in Section 4.4.1 in which we used the *LMDirichtlet* [10] similarity instead of BM25. This choice was made to change the initial ranking method before BERT. On 2021 the performance measures were comparable to the solution obtained with BM25 but we decided to submit anyway this run: it could give important results that can be analyzed in the next competitions.

On Figure 4.3 you can find a scheme that aims to give a more general concept of how the whole system worked, the same which then led to produce the runs for the TREC CAsT 2022 competition.

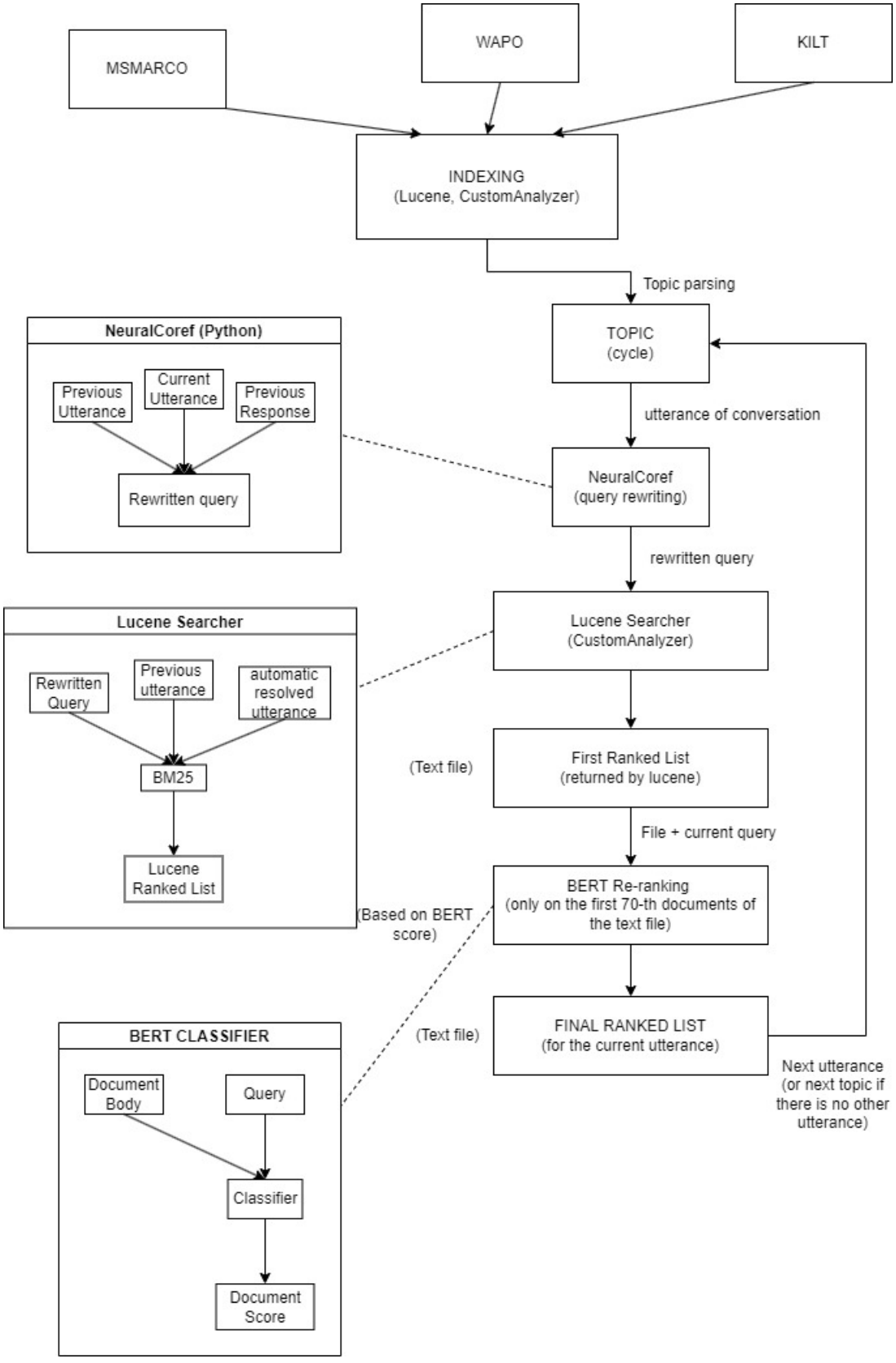



Figure 4.3: Pipeline

# 5

# Analysis

As described on the previous chapters we used different approaches to solve *TREC CAsT* 2021 and we compared them to get the best result.

| **Technique** | **NDCG@500** | **Recall@500** | **MAP@500** | **NDCG@3** |
|---|---|---|---|---|
| *Original query* | 0.1695 | 0.2667 | 0.0608 | 0.1240 |
| *First topic* | 0.2688 | 0.4366 | 0.0917 | 0.1608 |
| *Context query* | 0.2954 | 0.4627 | 0.1051 | 0.1858 |
| *NC, BERT, QA* | 0.3715 | 0.5290 | 0.1652 | 0.2926 |
| *NC, BERT, QA (improved)* | 0.4013 | 0.5574 | 0.1878 | 0.3291 |
| *NC, BERT, QA (no CQ)* | 0.4105 | 0.5658 | 0.1922 | 0.3558 |

Table 5.1: Run results

Looking at the Table 5.1 we can see that every solution adopted has brought improvements in every measure (this can also be seen in the figure 5.1). The most important measure, the one we have tried to maximize in each solution, was *NDCG@3* as it was the one that determined the final classification in the competition. With the last result obtained on this paper, we have reached values very close to the average results obtained in the 2021 competition (Table 5.2). On Table 5.1 there is a run called *Original query*: this one was produced using utterances as they are, without any automatic modification, expansion or replacement. This run was introduced in this work only to be able to define a

starting point of comparison.

Looking at the table 5.1 we can make the following considerations:

- the introduction of the first utterance of the conversation into the current query improves the result. We have already seen that the first utterance of each conversation contains the main topic of the discussion: by inserting this topic in next queries we can make more matches in the datasets. In Table 5.1 it is in fact possible to see a clear improvement from *Original query* to *First topic* technique;

- the introduction of the previous utterance in the first topic technique, as described in the previous chapter (Chapter 4, Section 4.2), has slightly improved the solution;

- our algorithm, which has *Context Query* as its starting point but with the use of *BERT*, *Neural Coref* and *QA models* has greatly improved the solution in terms of *NDCG@3* (we jumped from 0.1858 to 0.2926);

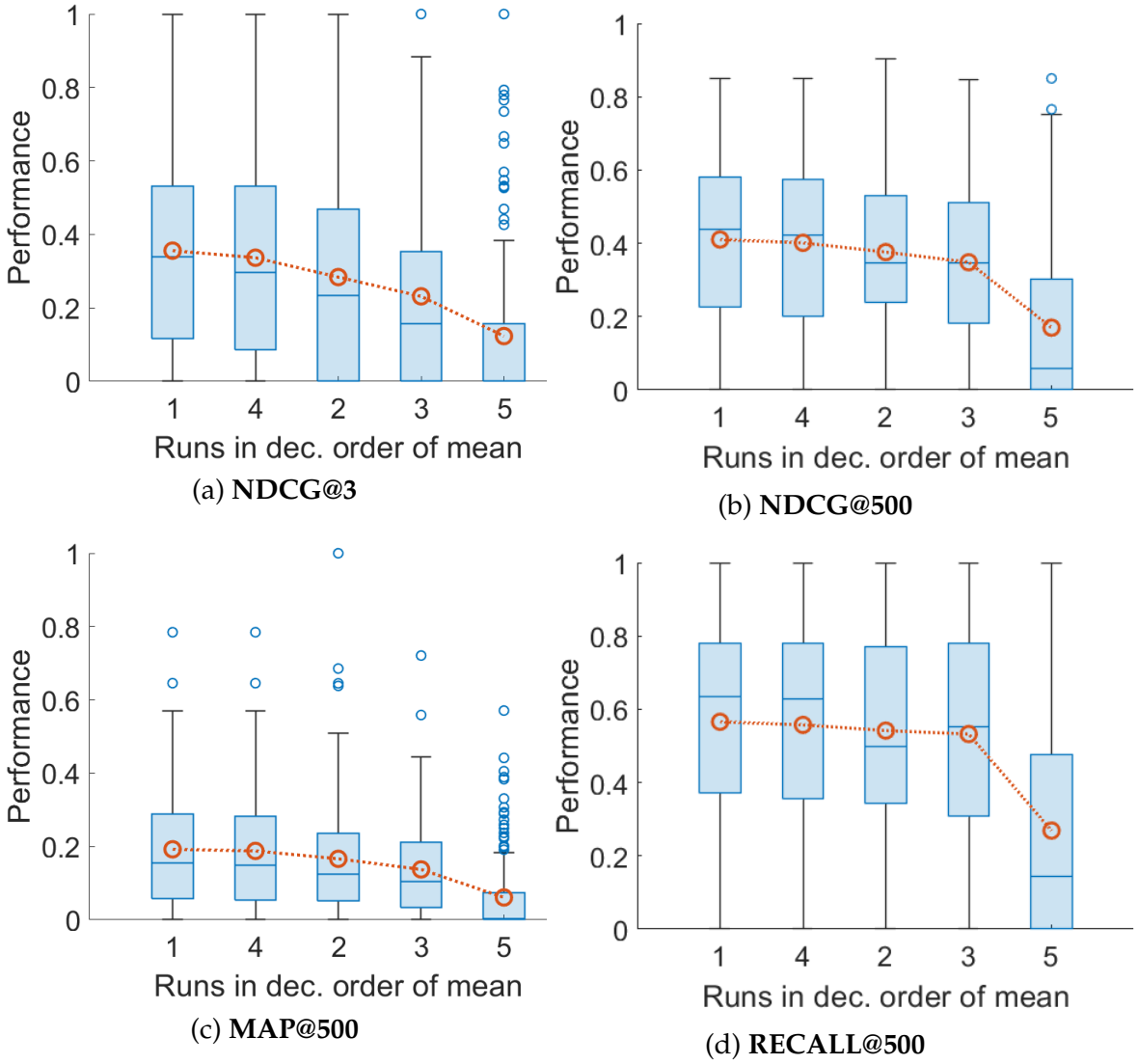
- the introduction of the *automatic_resolved_utterance* has clearly improved the solution: these utterances in fact contained more information than the originals, so they helped to increase performance;

- the removal of the previous utterance from the query, as described on the previous chapter (Chapter 4, Section 4.4.1), and the latest changes improved the performances. This improvement will be statistically proved on the following pages.

| **Group** | **Run_Id** | **Recall@500** | **MAP@500** | **MRR** | **NDCG@500** | **NDCG@3** |
|---|---|---|---|---|---|---|
| h2oloo | mono-duo-rerank | 0.850 | 0.376 | 0.679 | 0.636 | 0.526 |
| WaterlooClarke | clarke-cc | 0.869 | 0.362 | 0.684 | 0.640 | 0.514 |
| h2oloo | cqe-t5 | 0.846 | 0.342 | 0.644 | 0.618 | 0.488 |
| HBKU | HBKU_CQR_TC | 0.696 | 0.310 | 0.632 | 0.540 | 0.477 |
| HBKU | HBKU_CQRHC_BM25 | 0.598 | 0.287 | 0.622 | 0.490 | 0.471 |
| HBKU | HBKU_CQR_POS | 0.588 | 0.283 | 0.616 | 0.487 | 0.451 |
| CFDA_CLIP | CFDA_CLIP_ARUN1 | 0.697 | 0.308 | 0.613 | 0.539 | 0.444 |
| CFDA_CLIP | CFDA_CLIP_ARUN2 | 0.652 | 0.301 | 0.608 | 0.518 | 0.439 |
| h2oloo | cqe | 0.791 | 0.289 | 0.603 | 0.557 | 0.438 |
| MLIA-LIP6 | t5_doc2query | 0.761 | 0.290 | 0.585 | 0.548 | 0.436 |
|  | org_auto_bm25_t5 | 0.636 | 0.291 | 0.607 | 0.504 | 0.436 |
| UMD | umd2021_run3rrf | 0.723 | 0.298 | 0.611 | 0.539 | 0.425 |
|  | org_convdr_bert | 0.426 | 0.236 | 0.607 | 0.398 | 0.423 |
| uogTr | uogTrADT | 0.661 | 0.278 | 0.581 | 0.501 | 0.417 |
| UMD | umd2021_run2doc | 0.613 | 0.262 | 0.558 | 0.478 | 0.399 |
| HBKU | HBKU_CQR-HC | 0.531 | 0.236 | 0.531 | 0.422 | 0.392 |
| UMD | umd2021_run1 | 0.613 | 0.250 | 0.544 | 0.464 | 0.389 |
| MLIA-LIP6 | t5_monot5 | 0.360 | 0.190 | 0.571 | 0.337 | 0.388 |
| IITD-DBAI | IITD-RAW_U_T5_2 | 0.327 | 0.175 | 0.515 | 0.316 | 0.380 |
| h2oloo | t5 | 0.364 | 0.176 | 0.534 | 0.336 | 0.377 |
| UMD | umd2021_run4den | 0.735 | 0.265 | 0.521 | 0.512 | 0.377 |
| WaterlooClarke | clarke-auto | 0.721 | 0.260 | 0.524 | 0.487 | 0.375 |
| IITD-DBAI | IITD-RAW_U_T5_1 | 0.312 | 0.166 | 0.509 | 0.303 | 0.371 |
| MLIA-LIP6 | Rewritt5_monot5 | 0.361 | 0.184 | 0.549 | 0.332 | 0.370 |
| CMU-LTI | LTI-rewriter-g | 0.465 | 0.209 | 0.521 | 0.386 | 0.369 |
| CMU-LTI | LTI-rewriter-tc | 0.465 | 0.211 | 0.528 | 0.387 | 0.367 |
|  | org_convdr | 0.426 | 0.197 | 0.505 | 0.372 | 0.361 |
| CNR | CNR-run3 | 0.190 | 0.123 | 0.472 | 0.222 | 0.349 |
| CNR | CNR-run4 | 0.187 | 0.116 | 0.477 | 0.220 | 0.333 |
| uogTr | uogTrTDT | 0.557 | 0.216 | 0.491 | 0.408 | 0.332 |
| uogTr | uogTrTCT | 0.562 | 0.214 | 0.473 | 0.414 | 0.323 |
| TKB48 | bm25_automatic | 0.623 | 0.173 | 0.474 | 0.405 | 0.317 |
| CNR | CNR-run2 | 0.167 | 0.107 | 0.444 | 0.202 | 0.304 |
| CNR | CNR-run1 | 0.164 | 0.101 | 0.406 | 0.196 | 0.298 |
| CMU-LTI | LTI-rewriter-5q | 0.392 | 0.158 | 0.428 | 0.319 | 0.296 |
| UAmsterdam | astypalaia256 | 0.453 | 0.120 | 0.364 | 0.304 | 0.236 |
| V-Ryerson | DPH-auto-rye | 0.624 | 0.145 | 0.367 | 0.360 | 0.232 |
| UAmsterdam | historyonlyKILT | 0.288 | 0.084 | 0.314 | 0.214 | 0.196 |
| UAmsterdam | historyonly | 0.252 | 0.077 | 0.317 | 0.198 | 0.195 |
| MLIA-LIP6 | t5colbert | 0.589 | 0.076 | 0.270 | 0.314 | 0.154 |
| MEAN AND MEDIAN | | | | | | |
| **Mean** |  | 0.5283 | 0.2185 | 0.5224 | 0.4105 | 0.3716 |
| **Median** |  | 0.7210 | 0.2600 | 0.5240 | 0.4870 | 0.3750 |

Table 5.2: Scores TREC CAsT 2021 [3]

| Id | Technique |
|---|---|
| 1 | *NC, BERT, QA (no CQ)* |
| 2 | *NC, BERT, QA* |
| 3 | *Context query* |
| 4 | *NC, BERT, QA (improved)* |
| 5 | *Original query* |




(a) **NDCG@3**

(b) **NDCG@500**

(c) **MAP@500**

(d) **RECALL@500**

Figure 5.1: Box-plots

Looking at the box-plot 5.1d we can notice that the recall is high, this value can lead us to make the following considerations:

- in the first 500 retrieved documents there are many relevant documents;
- we can still improve the *NDCG@3* (there are many relevant documents which can be brought to the first positions with BERT).

Looking at the box-plot 5.1a we can notice an important improvement that we discussed on the previous chapter: the removal of the context query technique and the improvements on BERT statistically improved the performance; now we will prove it.
Comparing the two runs (Id 1 and 2) by using *NDCG@3* with a *student's t test*, we obtain the following *p-value*:

$$p - value = 0.000292 \tag{5.1}$$

Since this value is very low we can conclude that the two runs differ a lot and the result is statistically significant.

## 5.1 Comparing runs with Anova

*Anova test* is a generalization of the *Student's T test*:

- *Student's T test* allows to compare only 2 groups;
- *Anova* allows to compare more than two groups.

On this paper we used the *One-Way Anova Test*. The one-way analysis of variance (ANOVA) is used to determine whether there are any statistically significant differences between the means of three or more independent (unrelated) groups. Specifically, it tests the null hypothesis:

$$H_0 : \mu_1 = \mu_2 = ... = \mu_n \tag{5.2}$$

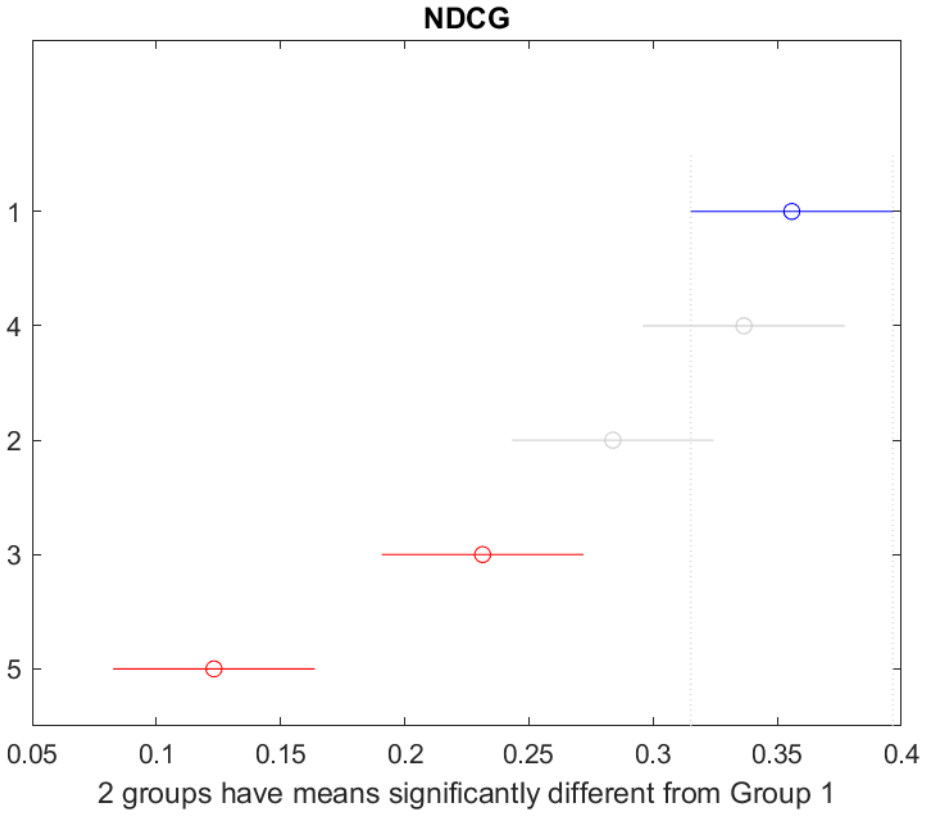



Figure 5.2: Comparing runs with Anova (on *NDCG@3*)

If one-way ANOVA reports a *p-value* of <0.05, you reject the null hypothesis. Let's consider now the Anova table which you can find below, since the *p-value* (= `1.7844e-15`) is very low we can conclude that the result of the test is statistically significant, that is, the means differ significantly.

| **Source** | **SS** | **df** | **MS** | **F** | **Prob > F** |
|---|---|---|---|---|---|
| **Columns** | 5.5238 | 4 | 1.3810 | 19.7111 | 1.7844e-15 |
| **Error** | 54.9977 | 785 | 0.0701 | - | - |
| **Total** | 60.5205 | 789 | - | - | - |

Table 5.3: Anova table

In particular, as shown on the figure 5.3, there are two runs that are significantly different from the most performing run in terms of *NDCG@3*.

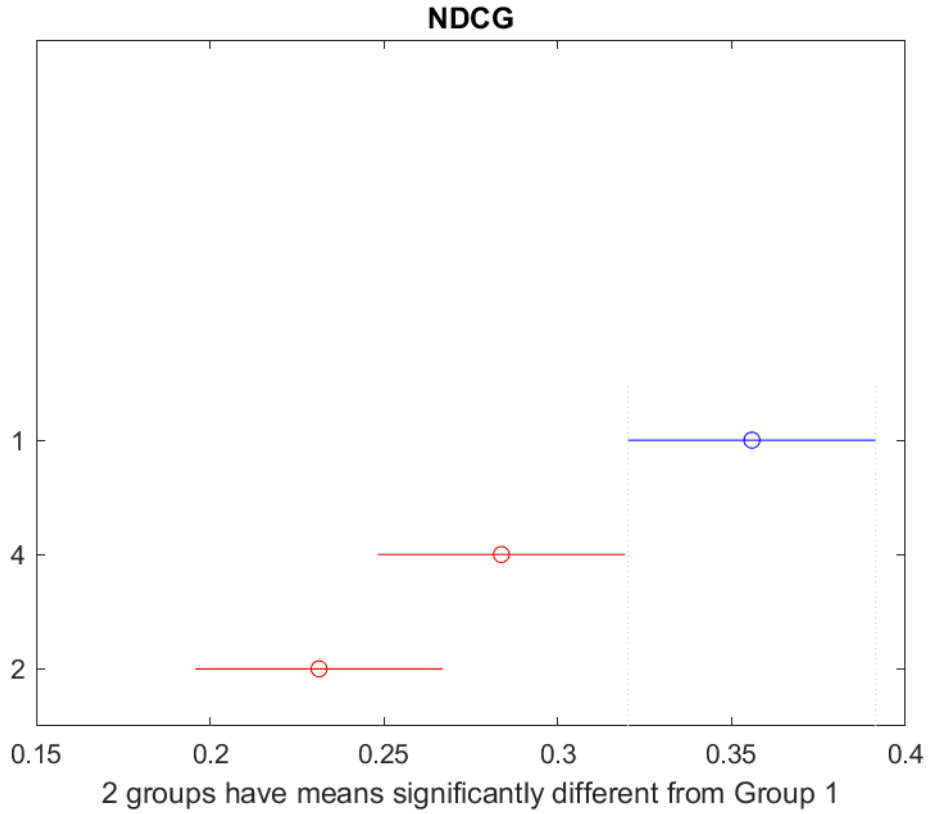



Figure 5.3: Comparing last runs (IDs: 1,4 and 2) with Anova (on *NDCG@3*)

Clearly, as shown graphically, the runs differ from each other. The introduction of BERT and systems for *Natural Language Processing* allowed a significant improvement in results.

# 6

# Conclusions and Future Works

The work of this project was the development of a conversational system that aimed in producing runs to be subscribed to TREC CaST 2022. During the lecture, we saw a lot of libraries and methodologies that can be applied to solve the TREC CAsT 2021 competition, the same were applied for TREC CaST 2022. The same techniques can be applied to build any conversational assistance like *Amazon Alexa*, *Google Assistant*, *Apple Siri* and many others. On this paper we saw methodologies which allow to solve Neural Language Processing (*Spacy* and *NLTK*), POS tagging and coreferences resolution (*NeuralCoref*) that allowed to rewrite the utterances during the conversation. Another key aspect we saw was BERT, a neural system which allowed to increase the final scores.

As future works we plan to improve the query rewriting system using *NeuralCoref*: this is the most difficult part when you develop a conversational system. The one described on this document can be improved, it replaces the pronouns by using only the *previous utterance* or the *previous response*. A possible approach could be merging the solution proposed in [12] (utterance labeling, discussed on Chapter 4) with the current one in order to create a better rewrite system. BERT could be improved considerably, for example by making a merge between the score of Lucene and the one obtained by BERT. Information passed to BERT can

also be greatly improved, clearly the more the information is clear and expanded in the correct way, the better the system works. The Lucene part of the project could be improved, the library is well supported and there are a lot of factors that can be tested, for example we could try other filters or searching methods.

# References


[3] Jeffrey Dalton, Chenyan Xiong, and Jamie Callan. "TREC CAsT 2021: The Conversational Assistance Track Overview". In: (2021). URL: https://trec.nist.gov/pubs/trec30/papers/Overview-CAsT.pdf.

[4] Jeffrey Dalton, Chenyan Xiong, and Jamie Callan. "TREC CAsT Track Year 4 2022 Guidelines". In: (2022). URL: https://www.treccast.ai/.

[5] Jianfeng Gao, Chenyan Xiong, Paul Bennett, and Nick Craswell. "Neural Approaches to Conversational Information Retrieval". In: *CoRR* abs/2201.05176 (2022). arXiv: 2201.05176. URL: https://arxiv.org/abs/2201.05176.

[6] Jianfeng Gao, Chenyan Xiong, Paul Bennett, and Nick Craswell. "Neural Approaches to Conversational Information Retrieval". In: *CoRR* abs/2201.05176 (2022). arXiv: 2201.05176. URL: https://arxiv.org/abs/2201.05176.

[7] Donna K. Harman. *The First Text REtrieval Conference*. 1993.

[9] Arash Habibi Lashkari, Fereshteh Mahdavi, and Vahid Ghomi. "A Boolean Model in Information Retrieval for Search Engines". In: *2009 International Conference on Information Management and Engineering*. 2009, pp. 385–389. DOI: 10.1109/ICIME.2009.101.

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. 2009. URL: https://nlp.stanford.edu/IR-book/information-retrieval-book.html.

[12] Ida Mele, Cristina Ioana Muntean, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, and Ophir Frieder. "Adaptive utterance rewriting for conversational search". In: *Information Processing, Management* 58 (2021).

[18] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. "MS MARCO: A Human Generated MAchine Reading COmprehension Dataset". In: *CoRR* abs/1611.09268 (2016). arXiv: 1611.09268. URL: http://arxiv.org/abs/1611.09268.

[19] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick S. H. Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. "KILT: a Benchmark for Knowledge Intensive Language Tasks". In: *CoRR* abs/2009.02252 (2020). arXiv: 2009.02252. URL: https://arxiv.org/abs/2009.02252.

[22] Ian Soboroff. "Overview of TREC 2021". In: (2021). URL: https://trec.nist.gov/pubs/trec30/papers/Overview-2021.pdf.

[27] Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. "Multi-passage BERT: A Globally Normalized BERT Model for Open-domain Question Answering". In: *CoRR* abs/1908.08167 (2019). arXiv: 1908.08167. URL: http://arxiv.org/abs/1908.08167.

[30] Hamed Zamani, Johanne R. Trippas, Jeff Dalton, and Filip Radlinski. "Conversational Information Seeking". In: *CoRR* abs/2201.08808 (2022). arXiv: 2201.08808. URL: https://arxiv.org/abs/2201.08808.

# Sitology


[1] *Apache OpenNLP*. URL: https://opennlp.apache.org/news/release-200.html.

[2] *Cross Encoder for MSMARCO*. URL: https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-12-v2.

[8] *Kilt Benchmarking*. URL: https://ai.facebook.com/tools/kilt/.

[10] *LMDirichlet*. URL: https://lucene.apache.org/core/8_0_0/core/org/apache/lucene/search/similarities/LMDirichletSimilarity.html.

[13] *MSMARCO*. URL: https://microsoft.github.io/msmarco/.

[14] *MSMARCO*. URL: https://microsoft.github.io/MSMARCO-Conversational-Search/.

[15] *Natural Language Toolkit*. URL: https://www.nltk.org/.

[16] *NeuralCoref*. URL: https://github.com/huggingface/neuralcoref.

[17] *NeuralCoref HuggingFace*. URL: https://huggingface.co/coref/.

[20] *Picke documentation*. URL: https://docs.python.org/3/library/pickle.html.

[21] *Question Answering*. URL: https://huggingface.co/tasks/question-answering.

[23] *Solar: filters for Lucene*. URL: https://solr.apache.org/guide/6_6/filter-descriptions.html.

[24] *The Stanford Question Answering Dataset*. URL: https://rajpurkar.github.io/SQuAD-explorer/.

[25] *TREC CAsT 2021 TOPICS*. URL: https://github.com/daltonj/treccastweb/blob/master/2021/2021_automatic_evaluation_topics_v1.0.json.

[26] *TrecEval*. URL: https://github.com/usnistgov/trec_eval.

[28] *Washington Post - ir dataset*. URL: https://trec.nist.gov/data/wapost/.

[29] *Word embedding*. URL: https://www.shanelynn.ie/get%5C-busy%5C-with%5C-word%5C-embeddings%5C-introduction/.

# Acknowledgments

I would like to express my special thanks of gratitude to my Professor *Nicola Ferro* who followed me during the project and to *Guglielmo Faggioli* which was a fundamental pillar between suggestions and techniques that were applied on this paper. A special thanks also goes to my girlfriend *Elena Pattaro* and my family for the support and the essential help during my University career.