Formula 1 Search System

Diogo Nunes
up201808546@up.pt
M.EIC, FEUP

Jéssica Nascimento
up201806723@up.pt
M.EIC, FEUP

João Vítor Fernandes
up201806724@up.pt
M.EIC, FEUP

Abstract

Nowadays, as the amount of available data on the Internet, the indexing and search process might be a problematic issue. In this project, we used a subset of a Kaggle dataset collection about Formula 1 (Races, Drivers, Constructors, Circuits and Seasons) and information retrieved from Wikipedia. After preparing/analysing the data and once we gathered a good number of information needs, Solr was used to retrieve and index information. When experimenting with evaluation formulas, we noticed that our search system is highly dependent on the way that the queries are constructed and the way we use the tools that Solr has to offer.

Keywords: datasets, search system, formula 1, scraping, crawling, wikipedia, solr, tokenizers, filters, information needs, queries, schema, configuration, relevance, evaluation, improvements, web interface

Introduction

There is much information on the Internet about Formula 1. However, as it keeps growing, information sources also increase and some details might be in one source and not in another. In this project, we want to keep a solid database with every detail about Formula 1 since the very beginning. We will create a Search System around that information, letting users search about any details of some Formula 1 topics, such as races and drivers.

Formula 1 is a rich topic in terms of information. We can talk about drivers, constructors (teams), circuits, each race, each championship, even about the regulations that are constantly changing and their history. In this project, we will try to construct a consistent database that joins all these topics described, so that everyone can search for some keywords and have results ordered by their relevance.

This report is divided in three parts: Data Preparation, Information Retrieval and System Improvements. In the first one, we start by collecting, cleaning and analysing our data, in part two we focus only on the races file to understand how to index and improve the default schema of Solr[3] and after all that analysis, in the third part, we join the rest of the data provided by Kaggle[1] and Wikipedia[2] and compare the use of our schema with a schemaless version and with the use of boosting on our schema.

Part I
Data Preparation

The preparation and cleaning of the data was done using Python[5], OpenRefine[6], BeautifulSoup4[7], and NLTK[8]. For data analysis, we used Python scripts and Draw.io[9] for the Data Processing Pipeline and Data Conceptual Model.

1 Data Collection

1.1 Datasets

The dataset, which was extracted from Kaggle[1], contains information about almost every detail about Formula 1 since the very beginning of the sport (1950). It was compiled from Ergast Developer API [10]. The dataset is composed of 13 CSV files: circuits, drivers, constructors, seasons, qualifying, races, status, results, lap_times, driver_standings, constructor_standings, constructor_results, pit_stops. However, we believe we have already enough information using only 5 of them: races, seasons, drivers, constructors, and circuits. We also added a CSV file which was the result of gathering some important aspects of Formula 1. This was an added extra that wasn't in the dataset we got from Kaggle[1]: we scrapped Wikipedia[2] pages about regulations, awards, federations, car components and history.

These are the files we used and their respective attributes:

races.csv: raceId, name, year, round, circuitId, date, time, url.

drivers.csv: driverId, number, firstName, lastName, birth, nationality, url.

constructors.csv: constructorId, name, nationality, url.

circuits.csv: circuitId, circuitRef, name, location, country, lat, lng, alt, url.

seasons.csv: year, url.

1.2 Wikipedia:

Since every file has URLs to Wikipedia[2] pages, we extracted data from them to enrich our database. The scraping process was done using Python and the BeautifulSoup4[7] package, in order to get large textual fields of information. Thus, we scrapped most of the paragraphs existing on each page.

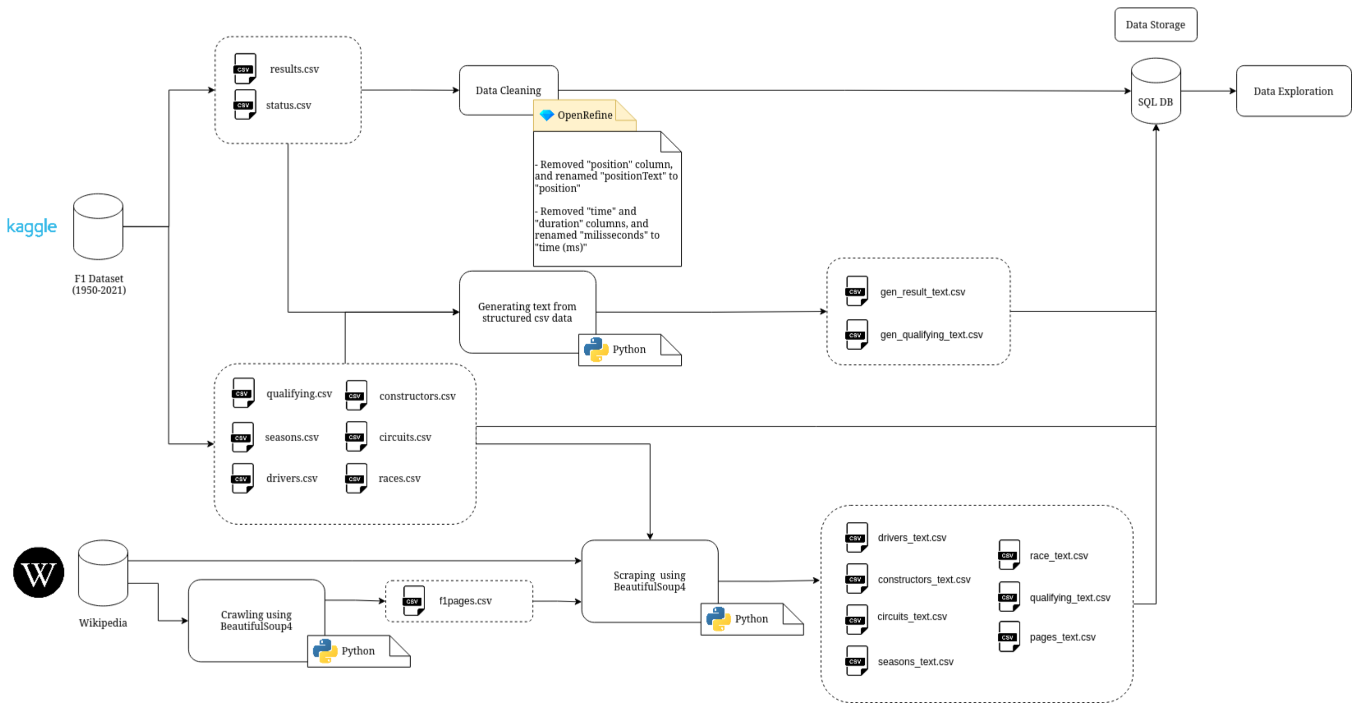


Figure 1. Data Processing Pipeline

2 Conceptual Model

In Figure 2, we can further understand the relationships between our data. A circuit represents the race track where a race occurred. There are many races every season (yearly) which has drivers participating in them. These drivers can change their constructor (team) but not in the middle of a season. All these classes have a textual field, scraped from different sources.

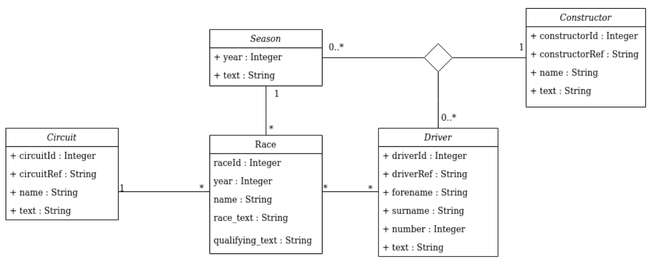


Figure 2. Data Conceptual Model

3 Data Characterization

We started by merging races.csv (content described above) with qualifying_text.csv and race_text.csv into a file named races_joined.csv. This file has all the paragraphs extracted from each race related to their qualifying and/or race sections. This means that, at this point, one document is a single race, described by its raceId, year, round (number of the race on that year), circuitId, name, date, time, URL, qualifying_text (column with all qualifying paragraphs of that race) and race_text (column with all race paragraphs of that race). A similar process was done for all the other files, but on these ones we only have 1 text column (called "text").

To understand what information we have, we created a Python[5] program to obtain the number of documents, the average number of words and the most common words in each file:

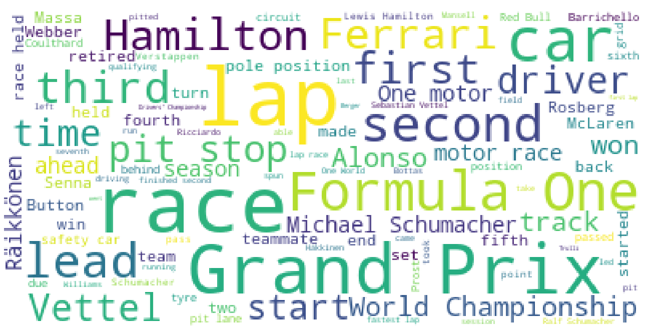


Figure 3. Most common words in paragraphs about F1 races

File	Number of documents
Races	1058
Drivers	853
Constructors	209
Circuits	77
Seasons	72
Pages	19
Total	2288

Table 1. Number of documents of each file

Category	Field name	Average word number
Races	race_text	606
Races	qualifying_text	147
Drivers	text	996
Constructors	text	1796
Circuits	text	1777
Seasons	text	3251
Pages	text	2924

Table 2. Average number of words for every text field

4 Information Needs (IN)

With the datasets provided, our goal is to be able to obtain relevant results for the most trivial searches on Formula 1 as well as much more complex ones. Below we have examples of searches that we want to achieve:

- IN1: Youngest driver to win a race or a championship (at their time)
- IN2: Incidents / Crashes / Contacts involving a certain driver
- IN3: Overtakes of a certain driver
- IN4: Teammates of a certain driver
- IN5: Races where a certain driver started on pole (1st)
- IN6: Fastest Lap award description and winners
- Many others that search mainly on the factual data instead of text columns: Drivers from a certain nationality, Constructors (teams) that a certain driver represented, countries, that hosted races on different cities, fastest lap award description and winners, ...

Part II Information Retrieval

In this part, we start our indexation, retrieval and, evaluation of information focusing only on the races file.

1 Tool Selection

The information retrieving tool used was Solr 8.10 [3]. We can index documents in different formats (such as CSV, JSON and XML), we can use schemas with tokenizers and filters or not (schemaless) and we can even use filters in queries. The documentation had a dense format, meaning that although there was a lot of information on the tools and features Solr had to offer, practical examples of those tools were sparse. Due to the size of the documentation, we didn't have time to explore everything in it. However we tried out a considerably large amount of those tools, and picked the ones that functioned as desired.

2 Collections and Documents

We started by importing the CSV files with all the text in one single column in a string format. However, we realized it makes more sense to have a list of phrases instead of a single string, so that we can easily have better results (matches in queries). We split the text and then we started to import JSON files instead of CSV files.

Additionally, we only used 1 core, even having different types of documents there. Each document is identified by an additionally column "category", that specifies the document type (e.g. "Race" for races, "Driver" for drivers, etc).

3 Indexing Process

A schema of the indexed documents was defined on a schema.json file, which defines how the documents are indexed into Solr. This schema defines how documents are processed during indexing and querying. This is done through creating and assigning field-types, as well as using the built-in tokenizers and filters.

We named the field-type that processes our scraped textual information as "text". This is the case for fields such as:

- race_text
- qualifying_text
- name

The default tokenizer was used to break the streams of text into tokens, since this satisfied our needs. White-space is treated as a delimiter, which was the main desired behavior.

When it came to filtering results, as the first step, all characters are converted to their ASCII equivalents, this defines the number of different characters that are taken into account, avoiding that rarer or non-standard characters disrupt the searching process. Then tokens are then turned into their respective lower-case and singular form, this prevents having to case match or provide the exact grammatical number for a noun, in order for relevant documents to be shown.

This refinement is applied both on indexing and on querying time, with the help of the following Solr's built-in filter elements:

- ASCIIFoldingFilterFactory [11]: converts alphabetic, numeric and symbolic Unicode characters which are not in the Basic Latin Unicode block (the first 127 ASCII characters) to their ASCII equivalents, if one exists;
- LowerCaseFilterFactory [12]: converts any uppercase letters in a token to the equivalent lowercase token; all other characters are left unchanged;
- EnglishMinimalStemFilterFactory [13]: stems plural English words to their singular form;
- ManagedSynonymGraphFilterFactory [14]: maps single or multi-token synonyms, producing a fully correct graph output;
- FlattenGraphFilterFactory [15]: must be included on index-time analyzer specifications that include at least one graph-aware filter;
- StopFilterFactory [16]: discards tokens that are on the given stop words list.

Information collected was joined into a single file named races_joined.csv, combining the gathered reports from the qualifying section and the final race. The resulting documents have the following attributes:

- raceId
- year
- round
- circuitId
- name
- date
- time
- url
- qualifying_text
- race_text

A result document could, for example, return the following information:

Attributes	Values
raceId	1054
year	2021
name	Portuguese Grand Prix
qualifying_text	Qualifying started at 16:00 local time (...)
race_text	The 2021 Portuguese Grand Prix (...)

Table 3. Example of a result document about races

4 Retrieval Process

For this section, we had to choose a query parser between the 3 that Solr offers:

- Lucene (standard)
- DisMax (Maximum Disjunction)
- eDisMax (Extended DisMax)

We quickly noticed that eDisMax was the best one, because it offers more fields and is an improved version of DisMax.

In this task, we used the following fields:

- *q:* field to input the query of the user;
- *q.op:* AND or OR;
- *fq:* filter query;
- *fl:* field list;
- *qf:* query fields with optional boosts;
- *pf:* phrase boosted fields (gives boost based on proximity of searched words)
- *ps:* phrase slop (maximum amount of tokens that a search result might have between searched words).

We made a JSON file with a default query parser configuration for each core (schemaless, schema and schema with boosts), so that we can easily just fill the query field and automatically use the eDisMax query parser with the configurations we made.

Configuration file for both Schemaless and Schema cores:

- fl: *, score
- defType: edismax
- qf: race_text qualifying_text driver_text constructor_text circuit_text season_text page_text name nationality firstName lastName year location
- pf: race_text qualifying_text driver_text constructor_text circuit_text season_text page_text name nationality firstName lastName year location
- ps: 10

Configuration file for schema with boost core:

- fl: *, score
- defType: edismax
- qf: race_text^10 qualifying_text driver_text constructor_text circuit_text season_text page_text name^100 nationality^50 firstName^25 lastName^50 year^25 location^25
- pf: race_text^10 qualifying_text driver_text constructor_text circuit_text season_text page_text name^100 nationality^50 firstName^25 lastName^50 year^25 location^25
- ps: 10

5 Evaluation

In this section, we evaluate the results obtained, so that we can have an objective picture of the performance of our system. Basically, we present Average Precision, Precision at 10 and Recall at 10 to evaluate if our queries are returning the relevant results we wanted to achieve. Since we use the races file, the following queries search on race_text and qualifying_text. Finally, to understand if our schema was better than the Solr's default, we calculated the three metrics for the default schema (schemaless), our schema and our schema with the use of boosts.

Query 1:

This query focuses on the information need:
Youngest drivers to ever win a race or a championship. Our objective here is to search about the youngest drivers that, over the years, win races or championships, so, in theory, we don't have only 1 relevant result (there are many others).

q: youngest driver win
q.op: AND
fq: category:race

When running this query, we obtained the results in following order (R means Relevant, N means not relevant):

- Schemaless: RRRRNNRNRN
- Schema: RRRRNRNNRN
- Schema + Boost: RRRRNRNNRN

Metric	Schemaless	Schema
Average Precision	0.90	0.92
Precision at 10 (P@10)	0.6	0.6
Recall at 10 (R@10)	1.0	1.0

Table 4. Evaluation Results for query 1 (Schemaless vs Schema)

Metric	Schema	Schema + Boost
Average Precision	0.92	0.92
Precision at 10 (P@10)	0.6	0.6
Recall at 10 (R@10)	1.0	1.0

Table 5. Evaluation Results for query 1 (Schema vs Schema+Boost)

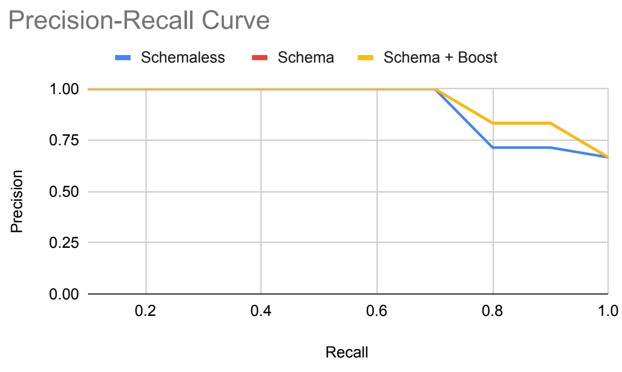


Figure 4. Precision-Recall Curve of Query 1

Query 2:

This query focuses on the information need:
Incidents involving a certain driver. Here we want to search any incident, accident/crash or just a contact that involves a certain driver, either it's fault or not. We chose the driver Max Verstappen because he's an aggressive driver, so that we can have a bigger amount of document results. The use of the proximity operator was used so that these pairs of words appear relatively close from each other.

q: "incident Verstappen"∼10 "crash Verstappen"∼10 "accident Verstappen"∼10 "collision Verstappen"∼10 "contact with Verstappen"∼10
q.op: OR
fq: category:race

When running this query, we obtained the results in following order (R means Relevant, N means not relevant):

- Schemaless: RRRRNRRRNN
- Schema: RRRNRRNRRR
- Schema + Boost: RRRRRRNRRN

Metric	Schemaless	Schema
Average Precision	0.94	0.87
Precision at 10 (P@10)	0.7	0.8
Recall at 10 (R@10)	1.0	1.0

Table 6. Evaluation Results for query 2 (Schemaless vs Schema)

Metric	Schema	Schema + Boost
Average Precision	0.87	0.97
Precision at 10 (P@10)	0.8	0.8
Recall at 10 (R@10)	1.0	1.0

Table 7. Evaluation Results for query 2 (Schema vs Schema+Boost)

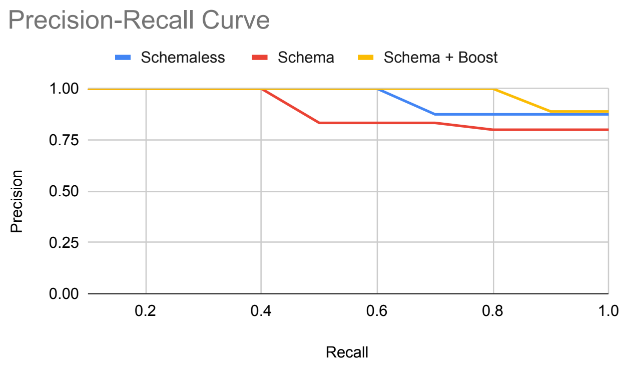


Figure 5. Precision-Recall Curve of Query 2

Query 3:

This query focuses on the information need:
Races where a certain driver made overtakes. We decided to filter by year because this driver was 4x champion between these years, so the probability of having more overtakes on this period is bigger. The use of the proximity operator was used for the same objective of the query 2: reduce documents retrieved so that the name of the driver and the word "overtake" appears relatively close from each other.

q: "Vettel overtake"∼10 "Vettel pass"∼10
q.op: AND
fq: category:race
fq: year:[2010 TO 2013]

When running this query, we obtained the results in following order (R means Relevant, N means not relevant):

- Schemaless: RNNNRNNRNN
- Schema: RRNNRRNRRN
- Schema + Boost: RRNRRRRRRN

Metric	Schemaless	Schema
Average Precision	0.59	0.76
Precision at 10 (P@10)	0.3	0.6
Recall at 10 (R@10)	1.0	1.0

Table 8. Evaluation Results for query 3 (Schemaless vs Schema)

Metric	Schema	Schema + Boost
Average Precision	0.76	0.88
Precision at 10 (P@10)	0.6	0.8
Recall at 10 (R@10)	1.0	1.0

Table 9. Evaluation Results for query 3 (Schema vs Schema+Boost)

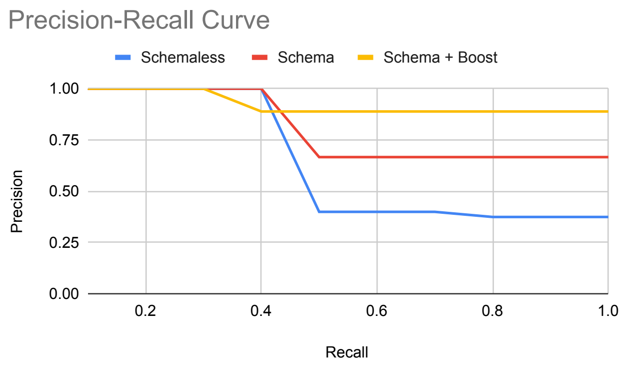


Figure 6. Precision-Recall Curve of Query 3

Part III System Improvements

In Part II, we started to evaluate our system only with the file about races, with the schema we created, and with no comparison between different schemas. Then, to have a clearer understanding of the impact (positive or negative) our schema had on the results, in Part III we compared the default schema with 2 other schemas we created: one with boosting and other without it.

Once we gathered all Formula 1 files into each schema, we compared those three aforementioned schemas. This part shows how the results changed depending on which schema was used, utilizing the same queries seen above. Here we tested new queries about different topics.

We also made changes to the paper structure, by making it follow the requirements more accurately. Additionally, we made changes on subsections 3 (Data Characterization) and 4 (Information Needs) of Part I and on subsections 1 (Tool Selection), 2 (Collections and Documents), 4 (Retrieval Process) and 5 (Evaluation) of Part II. We did such changes because on the end of Part II we didn't accomplish what we wanted to, so we thought it was a good idea to do it for the last paper.

1 Evaluation of queries

The first three queries are the same as seen above, but this time the result might not be only races, but all type of documents. So, for the next queries, we removed the **fq** field. This field filters the type of document we want to have on the results.

1.1 Query Recapitulation

- Query 1: Youngest drivers to ever win a race or a championship.
- Query 2: Incidents involving a certain driver.
- Query 3: Races where a certain driver made overtakes.

Query 1

For this information need, we add *query: youngest driver win*.

By not limiting the type of documents retrieved, we are now able to have results of not only races where that happen, but also the drivers that made such conquers. In addiction, a Season document where the champion driver was the youngest to win (at that time) can also be retrieved.

When running this query, we obtained the results in following order (R means Relevant, N means not relevant):

- Schemaless: RRRRRRRNRN
- Schema: RRRRRNRRRN
- Schema + Boost: NRRRRNRNRR

Metric	Schemaless	Schema
Average Precision	0.99	0.95
Precision at 10 (P@10)	0.8	0.8
Recall at 10 (R@10)	1.0	1.0

Table 10. Evaluation Results for query 1 (Schemaless vs Schema)

Metric	Schema	Schema + Boost
Average Precision	0.95	0.69
Precision at 10 (P@10)	0.8	0.7
Recall at 10 (R@10)	1.0	1.0

Table 11. Evaluation Results for query 1 (Schema vs Schema+Boost)

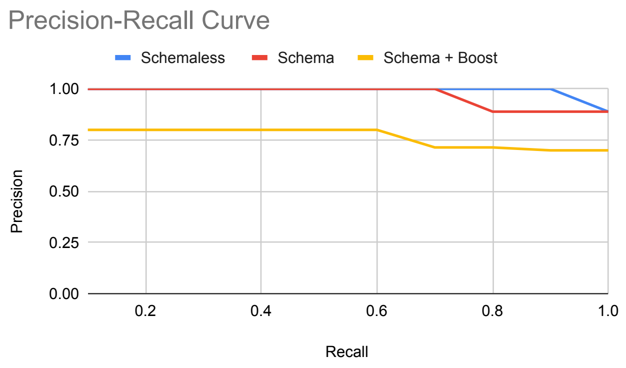


Figure 7. Precision-Recall Curve of Query 1

Query 2

For this information need, we add *query: "incident Verstappen" ∼10 "crash Verstappen"∼10 "accident Verstappen"∼10 "collision Verstappen"∼10 "contact with Verstappen"∼10.*

Being able to search on every type of document here, we can retrieve documents from not only races, but also from other aggressive drivers that might have collisions with Verstappen. Just like Query 1, seasons documents can also be retrieved.

When running this query, we obtained the results in following order (R means Relevant, N means not relevant):

- Schemaless: RRNRRRRRRR
- Schema: RRRRRNRRRR
- Schema + Boost: RRRNRRNRRN

Metric	Schemaless	Schema
Average Precision	0.88	0.95
Precision at 10 (P@10)	0.9	0.9
Recall at 10 (R@10)	1.0	1.0

Table 12. Evaluation Results for query 2 (Schemaless vs Schema)

Metric	Schema	Schema + Boost
Average Precision	0.95	0.88
Precision at 10 (P@10)	0.9	0.7
Recall at 10 (R@10)	1.0	1.0

Table 13. Evaluation Results for query 2 (Schema vs Schema+Boost)

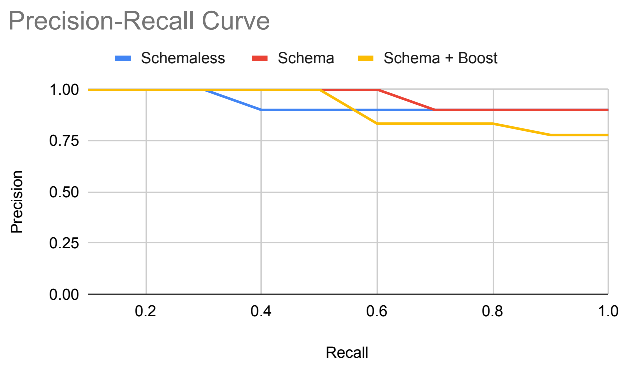


Figure 8. Precision-Recall Curve of Query 2

Query 3

For this information need, we add *query: "Vettel overtake"∼10 "Vettel pass"∼10.*

In this query, just like the two above, races and drivers documents are abundant, but also seasons and even circuits (where a certain great overtake was done by Vettel) can be retrieved here.

When running this query, we obtained the results in following order (R means Relevant, N means not relevant):

- Schemaless: NRRRNNNNNN
- Schema: RRRNRNRNNR
- Schema + Boost: RRRRNNRRNN

Metric	Schemaless	Schema
Average Precision	0.64	0.85
Precision at 10 (P@10)	0.3	0.6
Recall at 10 (R@10)	1.0	1.0

Table 14. Evaluation Results for query 3 (Schemaless vs Schema)

Metric	Schema	Schema + Boost
Average Precision	0.85	0.91
Precision at 10 (P@10)	0.6	0.6
Recall at 10 (R@10)	1.0	1.0

Table 15. Evaluation Results for query 3 (Schema vs Schema+Boost)

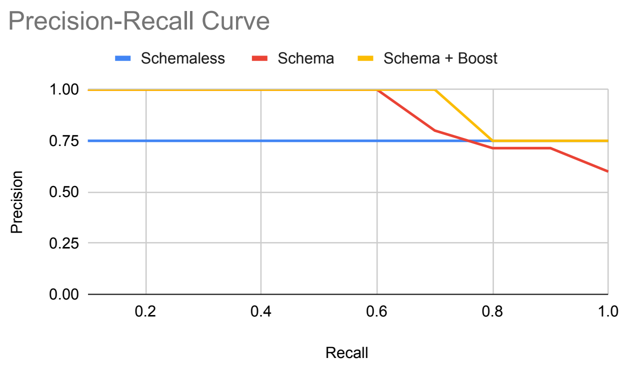


Figure 9. Precision-Recall Curve of Query 3

Query 4:

This query focuses on the information need:
Teammates of a certain driver

We restrict this query to drivers, because it only makes sense that way. By doing that, our retrieved results will be only drivers that were teammates of Raikkonen in this case (relevant), or, drivers that, in a certain moment, had a highlight moment with a Raikkonen teammate (not relevant).

q: "Raikkonen teammate"∼10
q.op: AND
fq: category:driver

When running this query, we obtained the results in following order (R means Relevant, N means not relevant):

- Schemaless: NNNNNNNNNN
- Schema: RNRRRRNRNR
- Schema + Boost: RNRRRRNRNR

Metric	Schemaless	Schema
Average Precision	0.0	0.79
Precision at 10 (P@10)	0.0	0.7
Recall at 10 (R@10)	0.0	1.0

Table 16. Evaluation Results for query 4 (Schemaless vs Schema)

Metric	Schema	Schema + Boost
Average Precision	0.79	0.79
Precision at 10 (P@10)	0.7	0.7
Recall at 10 (R@10)	1.0	1.0

Table 17. Evaluation Results for query 4 (Schema vs Schema+Boost)

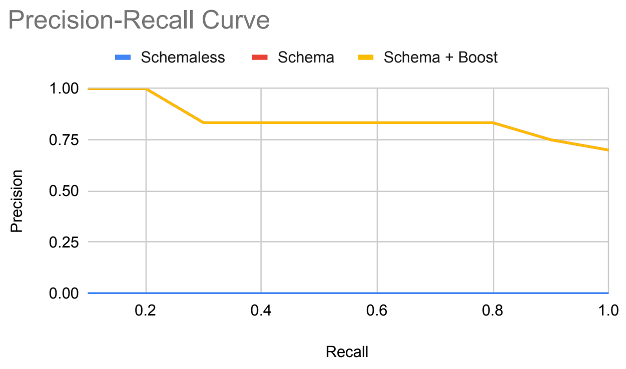


Figure 10. Precision-Recall Curve of Query 4

2 Web interface

The idea behind Figure 17, was to create an interface using React with a connection to our solr schema. A user could input in the search box the information needed. The interface would show the first ten results solr returned. Only the category, text and Wikipedia link of each result document would be displayed.

Due to lack of time, we only did the prototype of the web interface implementation, as can be seen on Figure 17 on Annexes.

Conclusion

We constructed a well designed Search System capable of being asked/queried about every single race realized until today. Throughout this paper we show its potential. Our dataset is capable of answering any question about this topic with relatively good precision. By looking back at the comparative analysis of schemas, made to improve our search system, we conclude that the majority of search results are relevant. Although our schema and boosts sometimes doesn't improve the search results, we believe it's still a decently good schema that will, in general, improve our Search System performance.

Future Work

There are some features that our Search System may incorporate later, such as:

- A web interface connected to Solr;
- Using strategies to find related documents inside other documents, so we can easily read about topics that might appear, and that the user wants to clarify;
- Propose new ranking signals using the existing information (e.g. PageRank signal bases on citation data).

References

[1] https://www.kaggle.com/
[2] https://en.wikipedia.org/wiki/Main_Page
[3] https://solr.apache.org/guide/8_10/
[4] https://www.kaggle.com/rohanrao/formula-1-world-championship-1950-2020
[5] https://www.python.org/
[6] https://openrefine.org/
[7] https://beautiful-soup-4.readthedocs.io
[8] https://www.nltk.org/
[9] https://app.diagrams.net
[10] http://ergast.com/mrd/
[11] https://solr.apache.org/guide/8_10/filter-descriptions.html#ascii-folding-filter
[12] https://solr.apache.org/guide/8_10/filter-descriptions.html#lower-case-filter
[13] https://solr.apache.org/guide/8_10/filter-descriptions.html#english-minimal-stem-filter
[14] https://solr.apache.org/guide/8_10/filter-descriptions.html#managed-synonym-graph-filter
[15] https://solr.apache.org/guide/8_10/filter-descriptions.html#flatten-graph-filter
[16] https://solr.apache.org/guide/6_6/filter-descriptions.htmlFilterDescriptions-StopFilter

Annexes

Data Characterization

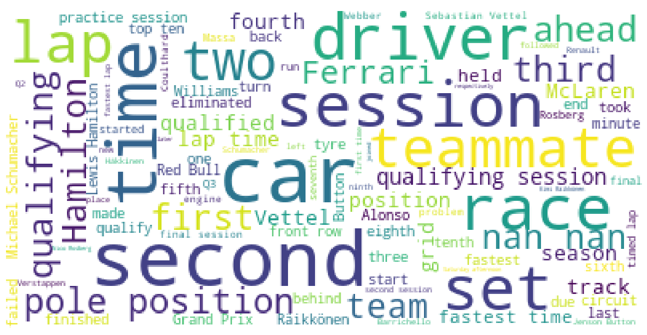


Figure 11. Most common words in paragraphs about the F1 qualifying sessions

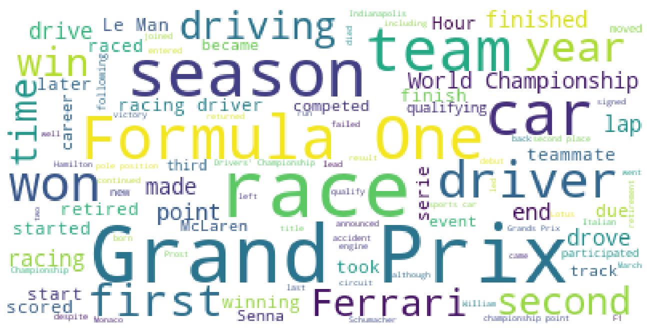


Figure 12. Most common words in paragraphs about F1 drivers

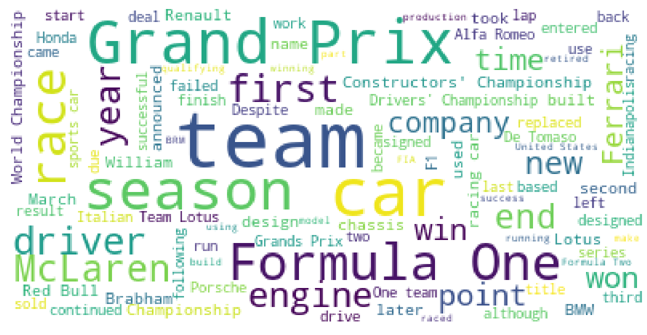


Figure 13. Most common words in paragraphs about F1 constructors

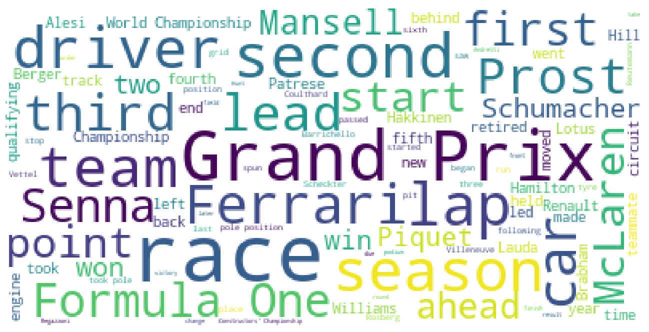


Figure 14. Most common words in paragraphs about F1 seasons

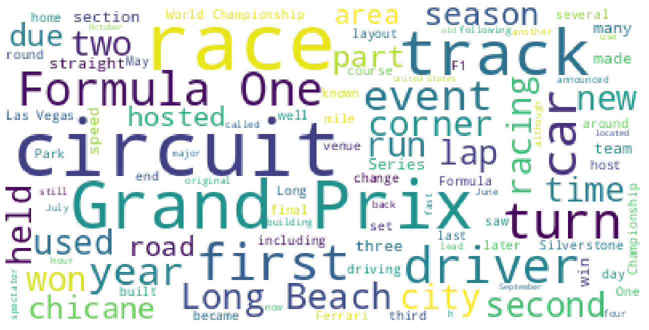


Figure 15. Most common words in paragraphs about F1 circuits

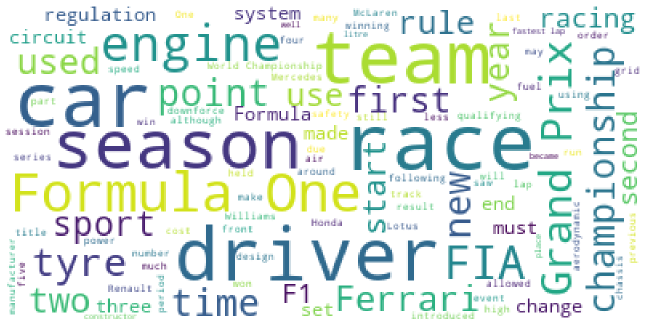
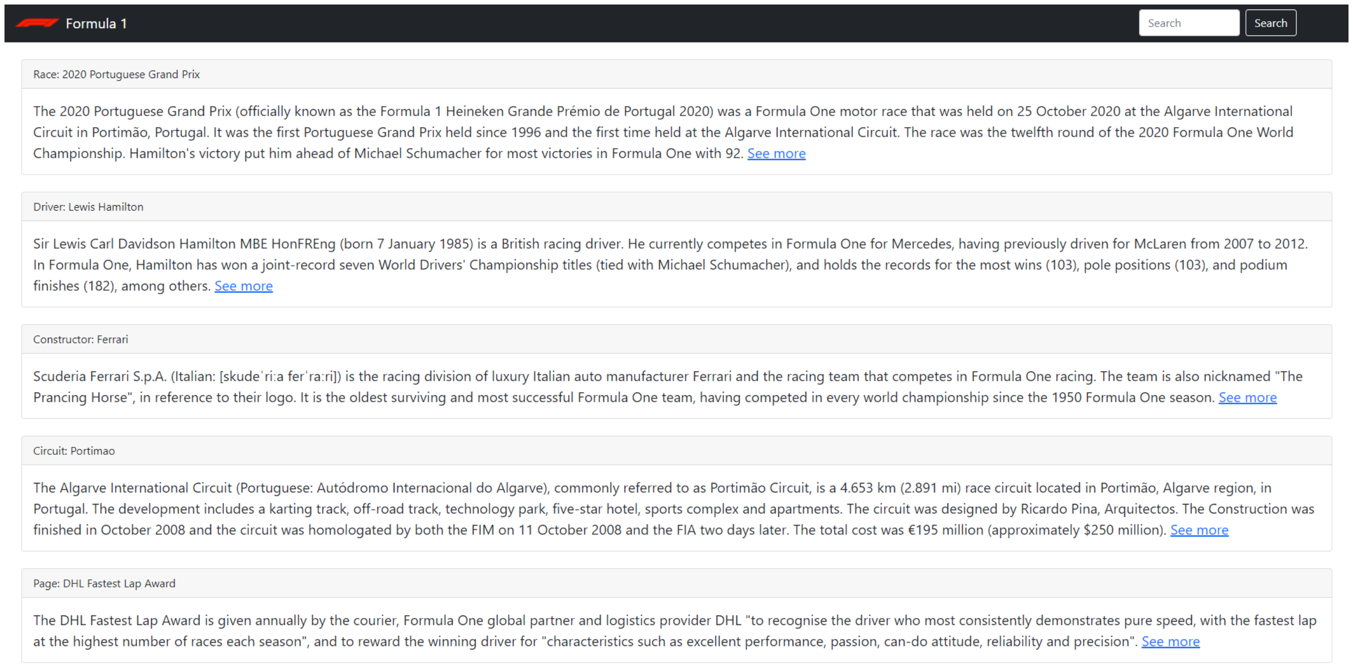


Figure 16. Most common words in paragraphs about F1 pages

Web Interface



Race: 2020 Portuguese Grand Prix

The 2020 Portuguese Grand Prix (officially known as the Formula 1 Heineken Grande Prémio de Portugal 2020) was a Formula One motor race that was held on 25 October 2020 at the Algarve International Circuit in Portimão, Portugal. It was the first Portuguese Grand Prix held since 1996 and the first time held at the Algarve International Circuit. The race was the twelfth round of the 2020 Formula One World Championship. Hamilton's victory put him ahead of Michael Schumacher for most victories in Formula One with 92. See more

Driver: Lewis Hamilton

Sir Lewis Carl Davidson Hamilton MBE HonFREng (born 7 January 1985) is a British racing driver. He currently competes in Formula One for Mercedes, having previously driven for McLaren from 2007 to 2012. In Formula One, Hamilton has won a joint-record seven World Drivers' Championship titles (tied with Michael Schumacher), and holds the records for the most wins (103), pole positions (103), and podium finishes (182), among others. See more

Constructor: Ferrari

Scuderia Ferrari S.p.A. (Italian: [skudeˈriːa ferˈraːri]) is the racing division of luxury Italian auto manufacturer Ferrari and the racing team that competes in Formula One racing. The team is also nicknamed "The Prancing Horse", in reference to their logo. It is the oldest surviving and most successful Formula One team, having competed in every world championship since the 1950 Formula One season. See more

Circuit: Portimao

The Algarve International Circuit (Portuguese: Autódromo Internacional do Algarve), commonly referred to as Portimão Circuit, is a 4.653 km (2.891 mi) race circuit located in Portimão, Algarve region, in Portugal. The development includes a karting track, off-road track, technology park, five-star hotel, sports complex and apartments. The circuit was designed by Ricardo Pina, Arquitectos. The Construction was finished in October 2008 and the circuit was homologated by both the FIM on 11 October 2008 and the FIA two days later. The total cost was €195 million (approximately $250 million). See more

Page: DHL Fastest Lap Award

The DHL Fastest Lap Award is given annually by the courier, Formula One global partner and logistics provider DHL "to recognise the driver who most consistently demonstrates pure speed, with the fastest lap at the highest number of races each season", and to reward the winning driver for "characteristics such as excellent performance, passion, can-do attitude, reliability and precision". See more

Figure 17. Web app's interface