Live Data Sampling for Analytics

Public Version

Company: Amadeus IT Group

Department: Search, Shopping and Pricing – Data mining Development and Studies

Location: 485 Route du Pin Montard – BP 69, 06902, SOPHIA ANTIPOLIS cedex, France

Student: Miguel ZABALLA PARDO

Program studies: Master in Data Science – EIT Digital

Current date: 15/05/2018

Internship dates: 05/03/2018 – 31/07/2018

Company supervisor: Guillaume LE GRAND- Manager, research engineer

University supervisor: Rodrigo CABRAL FARIAS

Short description: Study, propose and implement prototypes and strategies to predict the sampling ratio at query time that can optimize the aggregations times that render visualizations for business intelligence given large amounts of data.

Table of Contents

1. Introduction

In this report we are going to describe the work proposed and accomplished for the student's internship at Amadeus IT Group. The structure of the report will follow the chronological order in which the tasks have been performed. Thus, firstly we introduce the subject and context of the project. Later on, we describe the work to be done, including the planning. We will continue depicting the work done. Finally, we enumerate some conclusions that can be drawn from the project.

1.1. Context

In Business Intelligence, storing and aggregating this vast amount of data for visualizations purposes has a high cost, in terms of money and time. Although, big data tools and techniques such as parallel processing and distributed storage systems like Elasticsearch are used to allow acceptable response times for the visualizations this might not be enough. For this reason, another technique for reducing response time is data sampling. There exist different approaches for data sampling, in this project we will tackle the response time issue through a subset of data sampling strategies at query time.

1.2. Learning Objectives

Apart from the technical goals described in section *2. Description of Work* it is noteworthy some transversal goals for the student's education as a data scientist:

- Understanding and contributing to a complex big data ecosystem.
- Getting experience on software development activities and tools.
- Produce complete analyses and conclusion from data.
- Research on machine learning techniques.
- Propose ways to take a machine learning solution to production.

The context described above, and the student's background and interest on the topic guarantee that the objective can be achieved.

2. Description of Proposed Work

2.1. Problem Description

A business intelligence platform, that enables access and analysis of information to improve and optimize decisions in the industry, consists of a set of dashboards. One of the dashboards allows the exploration through graphical visualizations. For instance, we can see a plot of the top entries for each of the values of a certain field.

However, when the amount of days comprised is high the amount of data to be aggregated increases, therefore the query response time increases as well. This is due to the design of the data storage (Elasticsearch), since data from one specific day is stored in only one Elasticsearch index –an index is like a database in SQL. So, if the query needs to aggregate many indices the response time will be high.

Although, the number of indices is not the only variable having an impact on the response time of the query.

A plot has been extracted from a previous response time performance study done by another team member and clearly shows that response times are above 5 seconds.

The problem seems clear: response times over 5 seconds must be reduced. Thus, a possible solution consists on sampling, i.e. not fetching all data and fetching only a fraction (a sample) of it. However, as sampling is made randomly on the data this method has a drawback, we could be taking a sample which is not representative, and we will measure this by the sampling error. So, applying sampling to decrease the query response time could entail low data quality.

There are four possible solutions to tackle this problem:

- **Pre-aggregation:** the idea consists on having precomputed and stored some interesting measurements about the data, so that when the user makes the query the values to be displayed are ready to be shown
- **Usage restrictions:** another approach could be to limit the possibilities of the user through user interface. For instance, constraining the number of days for the query…
- **Increasing hardware:** it is also possible to increase the capacity of the machines or increase the number of executors to reduce the response times. But apart from the economic costs that this implies it means only postponing the problem, not actually solving it, as we expect the dataset to continue growing.
- **Live data sampling:** A reasonable solution is to apply sampling at query time. This means that the sampling ratio applied to the data to be aggregated is chosen accordingly to the nature of the current query. The aim is to find the highest sampling ratio (for a low sampling error) and the lowest response time. Studying and prototyping this solution is the main goal of the project.

2.2. Technical goals

The problem outlined above in the context can be tackled in several ways. However, considering the live data sampling the best approach involves the following functional/ technical goals:

- Understand the Business Intelligence platform and the data storage (Elasticsearch) workflows and structures.
- Building a generator of queries using Python DSL for Elasticsearch.
- Visualize, analyse and conclude from the generated data.
- Consider and assess different strategies for live data sampling.
- Propose and implement at least one model prototype for data sampling.

Thus, the goal of the internship is to **propose and implement prototypes and strategies to predict the sampling ratio that can optimize visualizations within a large business intelligence system.**

2.3. Relative and prior work

In this section we are going to talk about the fundamentals of Elasticsearch, and why doing live sampling is feasible. Besides, we will talk about different sampling methods and some research work done on this topic.

2.3.1. Elasticsearch

Elasticsearch is both a search engine and data store. In this section we will briefly describe some basic concepts about ES and how it has been deployed on the cluster. The way it has been deployed has big impact on the search performance and quality of the system, while the optimization of the infrastructure boosts scalability and availability.

Every instance of Elasticsearch is called a node. And the nodes are grouped in a cluster. The main data container is the index. An index is similar to database in SQL. In an index the data is grouped in fields, defined by the mapping (a JSON object), and the index has several documents (or records).

In our dataset one index corresponds to one day of searches. Currently around 650 indices or days of traffic have already been ingested. Each of them having from 40GB to 60GB. To process huge volumes of records the indices are split into multiple shards, a shard is an Elasticsearch partition (64 in our case). Since every record is stored only in one shard if we target only a certain number of shards we can reduce the response time. The way this is done is through the sharding algorithm.

The sharding or routing algorithm can be defined during the setup, in our case, it has been set up based on a sampling factor. The sampling factor is also a field defined in the mapping and needs to be inserted in an Elasticsearch query. Thus, if the sampling factor is low the number of documents hit will be also low and more importantly the number of targeted shards will also be low. The way for computing which shards to target is given by the following formula:

$$H(r)\ \%\ \#\ of\ shards = shard\_ids$$

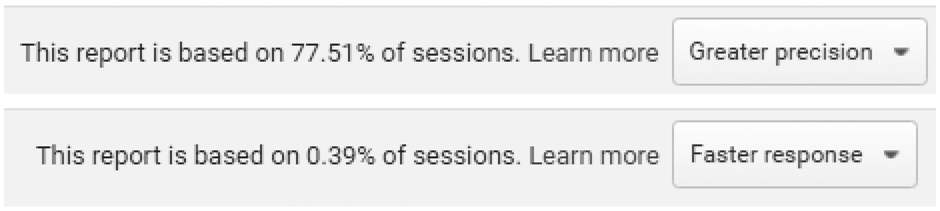
Where the function *H()* is a hash function (Murmur 3), *r* is the sampling factor chosen for the query, the number of shards is 64 an the domain of the sampling factor is 0 to 99999.

For instance, if the sampling factor of a query is 50000 the number of hits will be approximately half of the authentic number of hits, and the number of shards will be half, thus 32 in this case. However, since all data is spread among all shards we get a representative slice of the data. This is the fundamental idea that allows sampling at query time.

2.3.2. Similar approaches

Google Analytics

When looking into literature, our first task was to research about the solution that others have given to a similar problem. Google as one of the pioneers in big data solutions has also faced this problem. Although, we could not find technical details of how the tackled the problem, they do provide solution based on live data sampling for their Google analytics platform. Concretely, for their ad-hoc report documentations, they state [11]: *The sampling algorithm uses a sample of the complete data that is proportional to the daily distribution of sessions for the property for the date range you're using. For example, if over a 5-day period sessions were sampled at 25%, then the sample would include 25% of each day's sessions. […]. The sampling rate varies query to query depending on the number of sessions during a date range for a given view...* They even have plugin that allows to setup the time versus quality trade-off:


This report is based on 77.51% of sessions. Learn more

Greater precision

This report is based on 0.39% of sessions. Learn more

Faster response

Figure 1. Screenshot Google Analytics

Predicting the Response Time

The second important relative work is with regards the viability of predicting the response. In this topic we found two main works that inspire the solution taken in the project:

- The first one is a work done also within Amadeus to predict the response time of a flight request for a certain database. He considers this problem as very tricky as the variables do not only rely on the request but also on random variables such as the content of the cache. So, he finally proposes using classifying algorithms to return a probability [12]. AS we will see later, the key idea we extract from his work is to convert the regression prediction problem to classification prediction problem. Basically, creating bins for the distribution of the response time.
- To provide feasibility for predicting the response time we found an interesting paper on predicting website response time using support vector machines the authors claim to provide a 96% accuracy. Although the dataset and network are completely different from ours, we can claim we will do our utmost [16].

2.4. Approach

Following a typical cross-industry standard data mining process (CRISP-DM) we identify:

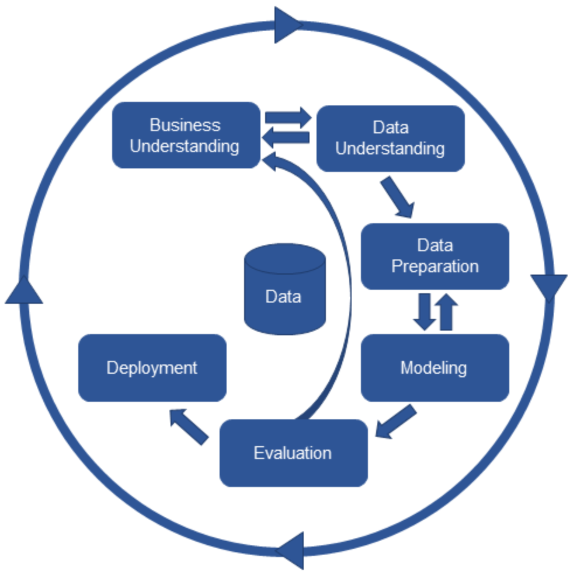


Figure 2. CRISP-DM schema

1. **Business understanding**: this step comprises the full comprehension of the end goal, in this case the optimization of business intelligence (BI) platforms through live data sampling. Reading literature, exploring the tools and similar datasets.
2. **Data understanding**: understand and research about the different queries a user can make on the BI platform to decide the final outlook of the dataset to obtain. Besides, this is a hands-on-stage and requires the knowledge of Elasticsearch.
3. **Data preparation**: Determine, collect and visualize KPIs that contribute to proposals of prediction strategies. This stage will be the longest one as implies building the generation of queries.
4. **Modelling**: prototype and assess strategies for predicting the sampling ratio.
5. **Evaluation**: define metrics for accuracy and precision of the models. And asses those metrics to propose in the final strategy.
6. **Deployment**: once the performance and feasibility of the strategy has been validated we could think of ways to implement the model in current existing platforms. The deployment of the strategy is not included in the scope of this project.

We expect to do several iterations over the stages, concretely in the data preparation phase, because getting a representative queries dataset is going to be a hard task.

2.5. Planning

We now share the initial tasks plan.

	Task Name	Start Date	End Date	Duration (Days)
Business understanding	Kick off meetings, reading documentation…	05/03/2018	19/03/2018	14
	Define scope of the project	12/03/2018	19/03/2018	7
Data understanding	Getting familiar with ES	12/03/2018	24/03/2018	12
	Environment setup	19/03/2018	26/03/2018	7
	Shoot queries to DB	20/03/2018	03/04/2018	14
Data preparation	Run/update querying scripts	25/03/2018	08/04/2018	14
	Parse/transform/clean datasets	08/04/2018	15/04/2018	7
	Full vs small bench comparison	15/04/2018	24/04/2018	9
	Visualize KPIs	24/04/2018	03/05/2018	9
Modelling	Generate prediction strategies	03/05/2018	17/05/2018	14
	Set up metrics (Response time vs error)	07/05/2018	11/05/2018	4

Evaluate	Test on reduced bench	11/05/2018	16/05/2018	5
	Test on full bench	16/05/2018	21/05/2018	5
	Caching benchmark bias test	21/05/2018	26/05/2018	5
	Display results	26/05/2018	31/05/2018	5
	Objectives achieved? -> Start over again!	31/05/2018	01/06/2018	1

Table 1. Task planning

2.6. Tools and other remarks

The main coding language has been Python. The main libraries have been:

- Python DSL for Elasticsearch for data collection.
- Numpy and Pandas for data manipulation.
- Matplotlib and Seaborn for data visualization.
- Sckit learn for machine learning.

Besides for the modelling and data mining Weka and SPSS. Other tools used are:

- Kibana - Elasticsearch UI.
- SublimeText as text editor.
- Jupyter Notebooks for the Python scripts including visualizations.
- Git repository for version control.

Regarding the implemented code it guarantees repeatability. This means that it contains all files to be easily executed and modified. The annex contains the repository Readme file.

3. Description of Accomplished Work

The work done so far can be split up into 3 sections: Query Generation, Visualizations and solutions.

3.1. Queries generator

The first big task to be done is the creation of scripts that generate representative queries with regards the queries a potential user might make to the “Travel Intelligence” dashboard. These queries are executed against Elasticsearch and must have the same schema as the queries shot by the web service for the visualization. The purpose of this task is to obtain a dataset containing a row per query and its features. Thus, we create a set of queries and later we shoot them. We now will outline some technical properties of the generation of the queries.

3.1.1. Filters and aggregations

Conceptually and technically, a query can be divided into two parts: **filters** and **aggregations**.

Filters are defined with the following properties:

- They can only reduce the number of hits of the query.
- The number of filters per query is randomly selected from 2 to 4.
- Each filter contains one parameter. For every filter the selected parameter is randomly chosen among all parameters.
- Each parameter contains at least one value. For every parameter the selected values are randomly chosen from a list of possible values.
- The parameters can be divided into four groups:
    - Term parameters.
    - Range parameters.
    - Mandatory: sampling ratio.
    - Indices: *date range.* Technically for Elasticsearch it is not a filter, but the way is generated is the same as if it was a filter.

Once the filters are executed the output data is aggregated. The properties that define the aggregations are:

- There is only one aggregation per query.
- Conceptually the operation is the same as a classic “group by”.
- There are two kinds of aggregations, each containing 3 values that help define it:
    - Top - groups by and sorts the data with an output maximum size.
        - Parameters to group by.
        - Parameters to sort by.
        - The possible maximum sizes are: 5, 1000, 5000, 10000.
    - Date - groups by date interval and outputs a metric for each group:
        - Kind of dates
        - Interval can be: day, week, month, quarter, or year.
        - Parameters to count by.
- The type of aggregation for each query is randomly chosen.
- The 3 values the parameter can take are randomly chosen.

3.1.2. Generation of queries Schema

The scripts for shooting the queries follow this schema:

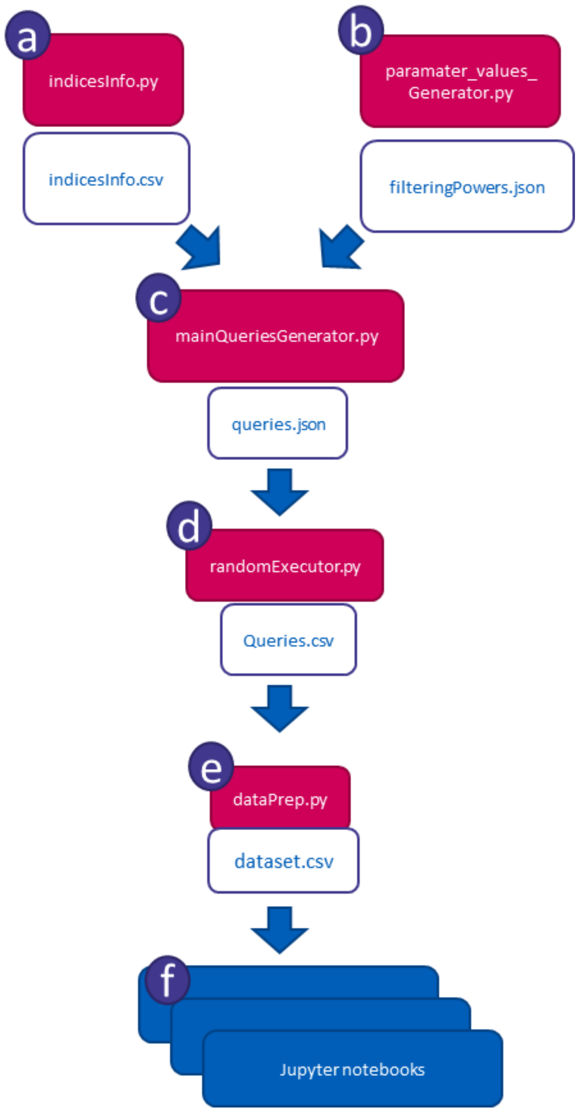


Figure 3. Queries Generator Schema

This first simple schema contains five scripts:

a) indicesInfo.py - computes the number of documents and the size in Gigabytes for each index. It sends this info to the main queries generator. Like the parameter_values_generator.py script it is not run every time we want a set of new queries, but it will need to be updated periodically.
b) paramater_values_Generator.py - outputs the whole lists of the possible values that the parameters can take in the filter. Besides, for each possible value it saves the *filtering power*, which is an estimation of the probability of this value to be part of the query. The filtering power is computed by dividing the number of occurrences of the value over the total number occurrences of all values in the list.
c) mainQueryGenerator.py – takes as input the lists of possible values, the sizes of the indices, the comprised dates, the sampling ratios, the number of times the same query is shot and the number of queries to be executed. And outputs these queries in JSON format.
d) randomExecutor.py - randomly runs and shoots to ElasticSearch all the stored queries one after the other. The reason to execute the queries randomly is to avoid that caching reduces the

response time in case every query is executed more than once with different sampling ratios. It outputs a CSV file (timErrorAnalysis.csv) containing the information of one query per line. The attributes are shown in table XX.

e) **dataPrep.py** - joins the filtering power for each query filter and computes the main *filtering power*, which is the product of the filtering powers for each query. It also adds computation as the mean, maximum and minimum when each query is shot more than once.

f) **Jupyter notebooks** – these are set of scripts for visualization and analysis purposes. Its content and conclusion are described in section *3.4 Visualizations and 3.5 The Strategies*. Besides the readme file attached in the annex of this report contains a detailed explanation of each note-book.

3.1.3. Datasets

The basic structure of the ideal dataset follows this schema. We have as many queries as possible. But we consider that 3000 queries per sampling ratio is a fair quantity. There are have been two important bottlenecks when producing this this dataset:

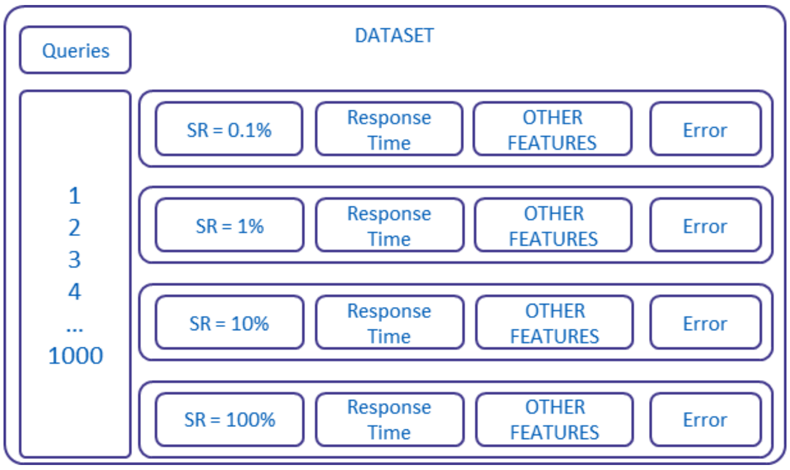


Figure 4. Ideal dataset

- As we will show later, the average time for a non-sampled (SR= 100%) query is 5 minutes. Thus, producing these 3000 queries can take around 15000 minutes, or 250 hours, or 10 days…
- We find out that non-sampled data is necessary to compute the error. Thus, computing the error will also take some time.

In order to accelerate the analysis and have some results with a portion of the data, we decided to split the dataset in three parts. That can require less time to be computed and can still provide us useful information. These datasets are:

1. **Only sampled dataset.** We omit the error. Besides, for this dataset we run each query 5 times. This allows us to get a more representative value for the response time (the median of the 5 values) and to assess the variance of the response time.
2. **Small non-sampled dataset**. Containing only 1000 queries.
3. **Reduced version of ideal dataset**. Only 200 queries per sampling ratio.

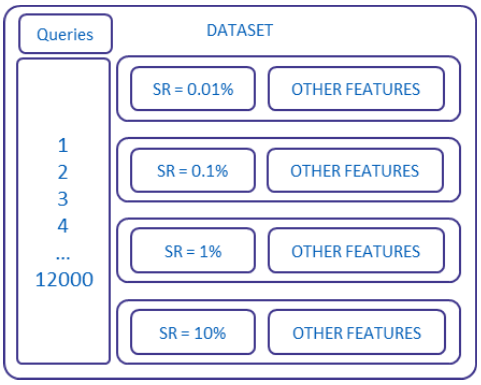


Figure 5. Only sampled dataset

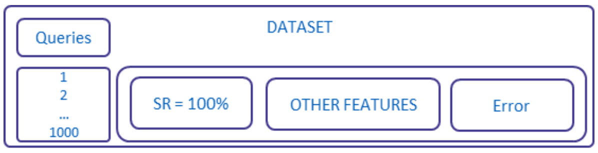
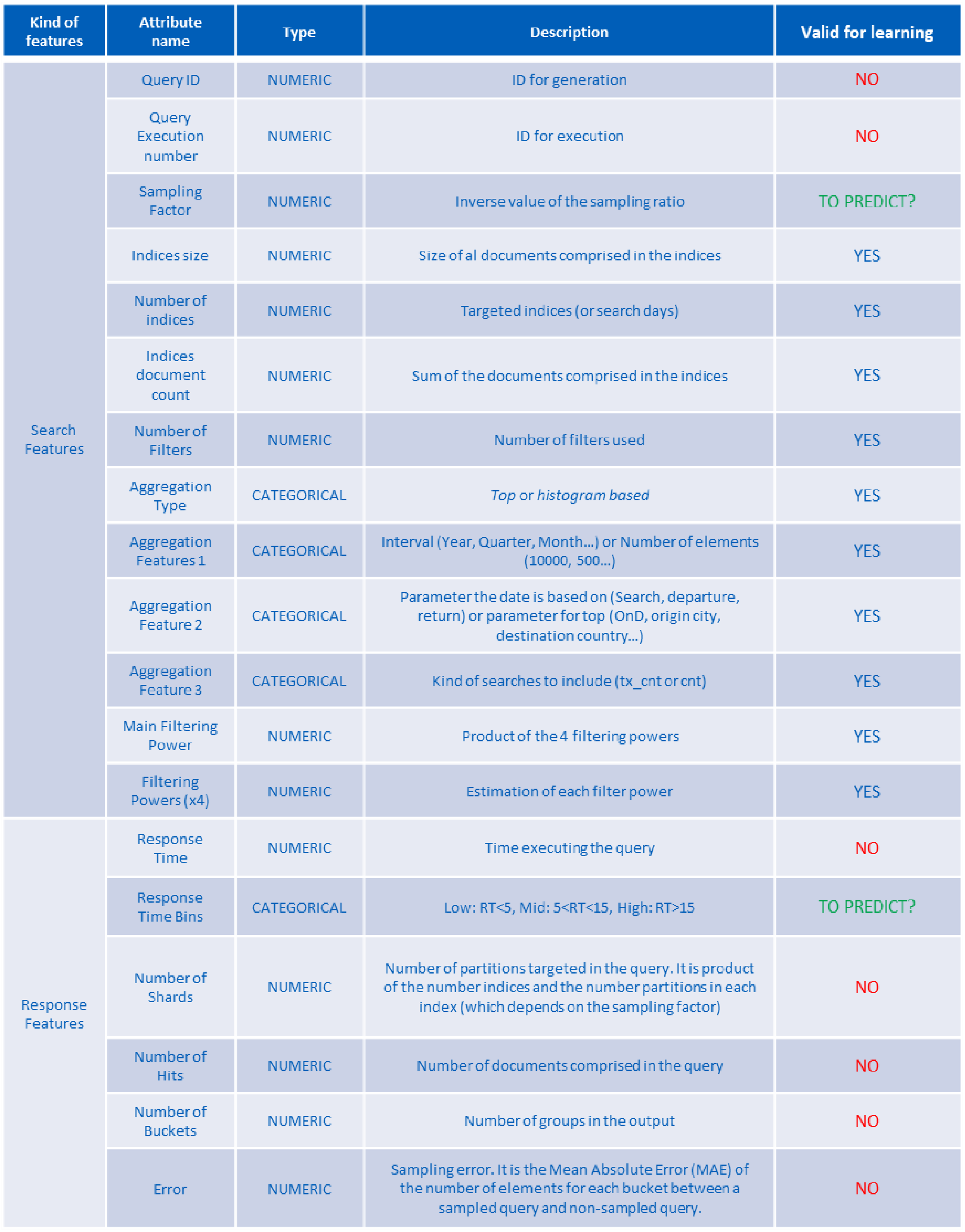


Figure 6. Small non-sample dataset.

Despite this punctual approach all results, and visualizations have been done with the ideal dataset.

3.1.4. Features

The following table describes all attributes included in this dataset:



Kind of features	Attribute name	Type	Description	Valid for learning
Search Features	Query ID	NUMERIC	ID for generation	NO
	Query Execution number	NUMERIC	ID for execution	NO
	Sampling Factor	NUMERIC	Inverse value of the sampling ratio	TO PREDICT?
	Indices size	NUMERIC	Size of al documents comprised in the indices	YES
	Number of indices	NUMERIC	Targeted indices (or search days)	YES
	Indices document count	NUMERIC	Sum of the documents comprised in the indices	YES
	Number of Filters	NUMERIC	Number of filters used	YES
	Aggregation Type	CATEGORICAL	*Top* or *histogram based*	YES
	Aggregation Features 1	CATEGORICAL	Interval (Year, Quarter, Month…) or Number of elements (10000, 500…)	YES
	Aggregation Feature 2	CATEGORICAL	Parameter the date is based on (Search, departure, return) or parameter for top (OnD, origin city, destination country…)	YES
	Aggregation Feature 3	CATEGORICAL	Kind of searches to include (tx_cnt or cnt)	YES
	Main Filtering Power	NUMERIC	Product of the 4 filtering powers	YES
	Filtering Powers (x4)	NUMERIC	Estimation of each filter power	YES
Response Features	Response Time	NUMERIC	Time executing the query	NO
	Response Time Bins	CATEGORICAL	Low: RT<5, Mid: 5<RT<15, High: RT>15	TO PREDICT?
	Number of Shards	NUMERIC	Number of partitions targeted in the query. It is product of the number indices and the number partitions in each index (which depends on the sampling factor)	NO
	Number of Hits	NUMERIC	Number of documents comprised in the query	NO
	Number of Buckets	NUMERIC	Number of groups in the output	NO
	Error	NUMERIC	Sampling error. It is the Mean Absolute Error (MAE) of the number of elements for each bucket between a sampled query and non-sampled query.	NO

Figure 7. Features table.

The number of shards can be computed using search features with the following formula for every query:

$$\#Targeted\ Shards = \#Targeted\ Indices \times \frac{Sampling\ Factor}{\#Sampling\ Factors} \times \#Total\ Shards$$

Formula 1. Number of targeted shards

This implies that the relation of the number of targeted shards and the sampling ratio is given by the following table

Number of indices	1	1	1	1	2	2	2	2	3	3	3	3		650	650	650	650
Sampling Ratio	100%	50%	10%	1%	100%	50%	10%	1%	100%	50%	10%	1%		100%	50%	10%	1%
Shards involved	64	32	6	1	64	32	6	1	64	32	6	1		64	32	6	1
Targeted shards	64	32	6	1	128	64	12	2	384	192	96	3		64	32	6	1

Figure 8. Shards-sampling ratio relation

This formula comes from the routing algorithm explained in 2.3.1 Elasticsearch.

The way the error is computed took some time to decide, as it is metric very specific to the problem. When a query is computed the output is a set of buckets/groups and for each bucket there is result value. This value, when there is no sampling, is the real value. However, when there is sampling, this value can vary as it is computed with only a fraction of the data and later multiplied by the sampling factor.

So, the way we compute the error is the Mean Absolute Error of the two arrays after normalization. The normalization is necessary so that both arrays have the same order magnitude. Y' and X' are the arrays containing the values of the buckets with sampling and with no sampling. Y and X are the normalized (or weight) vectors. In the cases that a bucket did not have value for the sampling array we inserted a 0.

$$y_i = \frac{y'_i}{\sum_i^n |y'_i|} \quad and \quad x_i = \frac{x'_i}{\sum_i^n |x'_i|}$$

$$MAE = \frac{\sum_i^n |x_i - y_i|}{n}$$

Figure 9. Normalization formulae

Obviously, when the data is not sampled, i.e. the sampling ratio is 100% the error is 0.

3.1.5. Fine tuning parameters for query generation

Now, we are going to assess the representativeness of our queries. Ideally, we would compare our dataset with the query log database of the platform. However, this database does not exist yet. Thus, we will have to make some assumptions and compare them. We can modify the representativeness of our queries by modifying two *meta-parameters*:

- **Number of filters range**: this meta-parameter defines the bounds for the number of filters. The number of filters of a query has an impact on the kind of query we are creating. So, if we apply many filters, the number of hits will be smaller. The question is whether this will also lessen the response time. Although, each query has a random number of filters, the range of values for the number of filters can be modified. The issue arises as we do not know if a user will typically set up between 1 and 3 filters or between 4 and 8 filters…
- A similar issue occurs with the range of possible values for every parameter in a filter. Each parameter has a set of possible values. For instance, the parameter *origin city* can take 9876 values, as this is the number of cities. When we select a random subset from the 9876 elements, the size of the random subset has to be determined. If the size of the subset varies randomly from 0 to 9876 when it is set close to 9876 the filter will have almost no effect. So, we want to limit the maximum size of the subset. For this purpose, we have created a *meta-parameter* called **denominator** that defines this maximum size. In our example if the denominator is 4. The selected random subset can have up to 2469 (9876/4) elements. Yet, the values for the parameter are chosen uniformly from 1 to 2469. Thus, the higher the denominator the tighter the filter.

We suspect that the modifications of these two *meta-parameters* (number of filters range and denominator) have a high impact on the number of hits of the queries and most likely on the response time. To show this, we have run the same queries with different values in four scenarios and compare number of hits and response time.

1. Low denominator (1) and low number of filters range (2).
2. High denominator (4) and low number of filters range (2).
3. Low denominator (1) and high number of filters range (8).
4. High denominator (4) and high number of filters (8).

We show the results for the four scenarios containing one index in the following table:

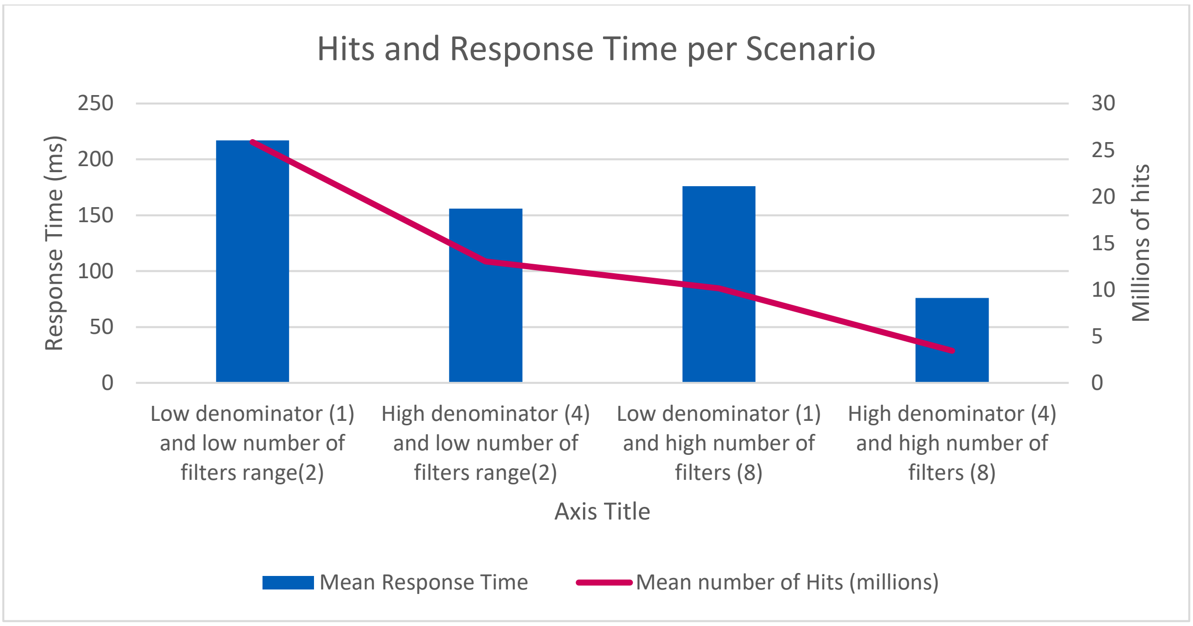


Figure 10. Hits VS Response Time analysis

We can already withdraw some insights from these plots:

- Low denominator has higher number of hits and higher response time than high denominator.
- Similarly, low number of filters range has higher number of hits and higher mean response time than high number of filters.
- High number of filters range and low denominator has lower number of hits than a low number of filters and high denominator having both similar mean response time.

This last insight suggests that possibly there is a small overhead per filter, which makes the mean response time be a bit higher when the number of filters grows. In addition, as it seems reasonable that users typically would perform queries with not more than 3 or 4 different filters number of filters we decided to fix the number of filters range from 2 to 4.

We finally decided to keep the denominator meta-parameter as 4, since the relation that hits and response time have when the denominator is high is more linear than when the denominator is low and we modify the number of filters.

3.2. Visualizations

This section is an exploratory analysis of the datasets created. Some of the questions we want to answer with this analysis are:

- How are the main features distributed?
- Are there clear relationships between some of the features?
- How do the features vary with different sampling values?
- How does the error behave?
- How does the filtering power relate to the main KPIs?

We will start by exploring non-sampled data, then will add to the analysis the sampled data.

3.2.1. Distributions

Our first goal is to see how Response Time distributes. We show the probability density function for our biggest dataset with non-sampled data.

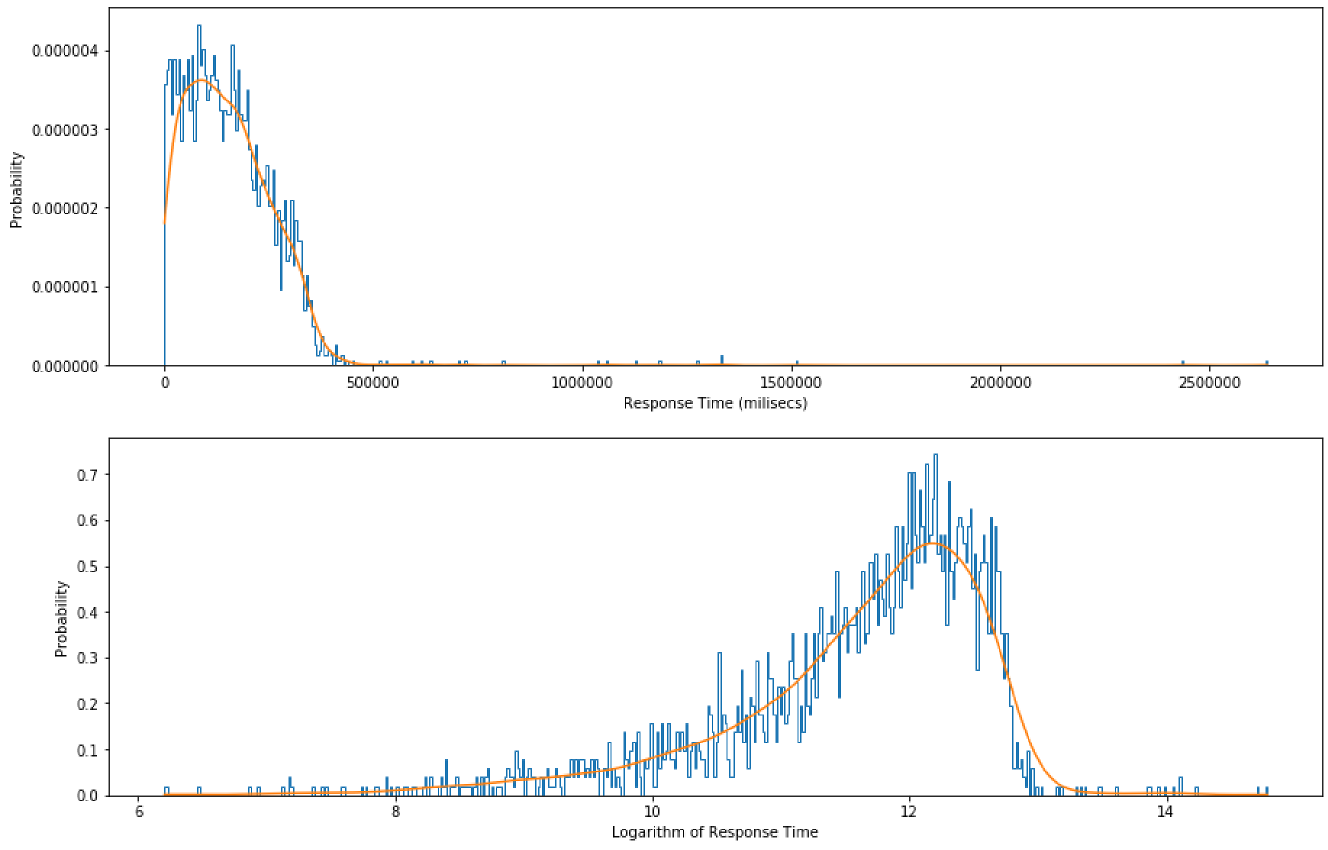


Figure 11. Response Time distribution

We conclude that the time is distributed exponentially, the maximum Response Time is around 40 minutes (although it can be considered an outlier). 90% of the queries are below 300 seconds (5 minutes). We also see that that 98% of the queries are above our 5 seconds threshold. Keeping in mind that the Response Time must be below 5 seconds we already foresee the high response times as a big issue. We also plot the logarithmic probability density function to make sure our data exponentially distributed. We find out our data is almost log normally distributed and negative skewed, to the right.

We can also plot the distribution of the Hits – Hits are the number of documents that match a query. Again, our distribution is exponential, but this time having very long neck and tail. However, in this case there are queries which have 0 hits which is problem for the applied logarithmic transformation. We have removed these queries as the amount was less than 5% and it doesn't have effect on the visualization. The logarithmic plot show that the values are spread quite uniformly and can go up to 56 billion hits. This might a good way to measure the representativeness of our queries.

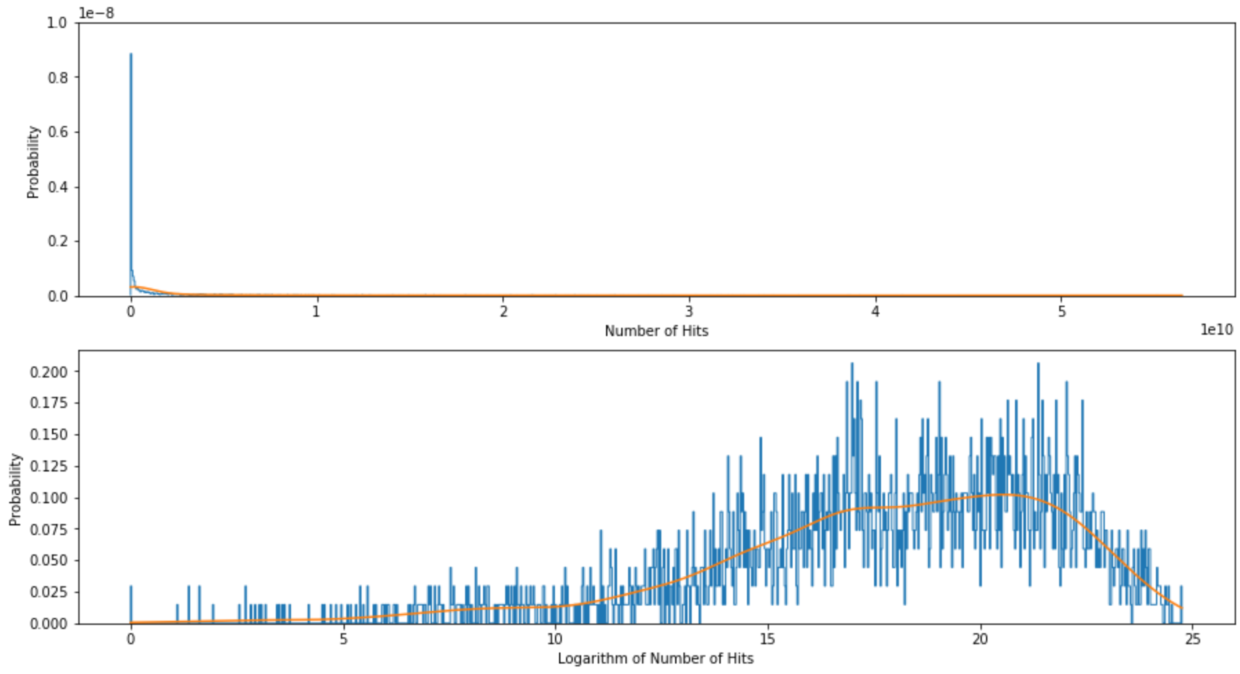


Figure 12. Hits Distribution

3.2.2. Key Relationships

Due to the architecture of Elasticsearch, we bear in mind two important hypothesis related to the Response Time distribution.

1. Hits could have an impact on Response Time: the more documents to process the longer the time.
2. Number of indices are linearly related to the Response Time: previous studies showed non user-friendly response times for more than indices including more than a year data.
3. The number of buckets might be related with kind of aggregation we make.

Hits VS Response Time

We plot a scatter plot for both variables comprising only one index (on the left-hand side) and including several indices. Note that both are non-sampled data queries. Since both variables are exponentially distributed we plot their logarithm.

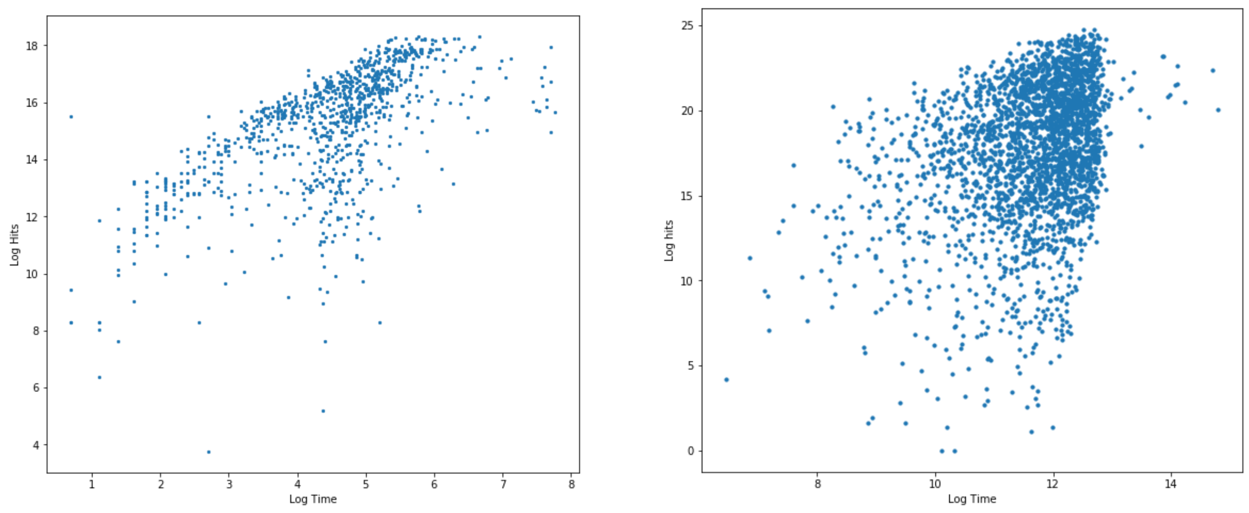


Figure 13. Hits vs Response Time

It is clear that when there is more indices the linear relationship disappears. This is an effect of the Simpson's paradox. The rough idea of this paradox is that a trend appears in several different groups of data but disappears or reverses when these groups are combined. In our case, it means that when several indices are combined the linear relationship between Hits and Response Time disappears. For this reason, we believe that although the linear relation is not visible the number of hits can still be a good predictor for the response time.

Number of indices VS Response Time

We first check this out visually for time and logarithmic time. The linear relationship is obvious! The greater number of indices the higher the response time. We are most likely in front of a very good predictor. Still the bubble of dots is quite large, so will need more predictors for the response time.

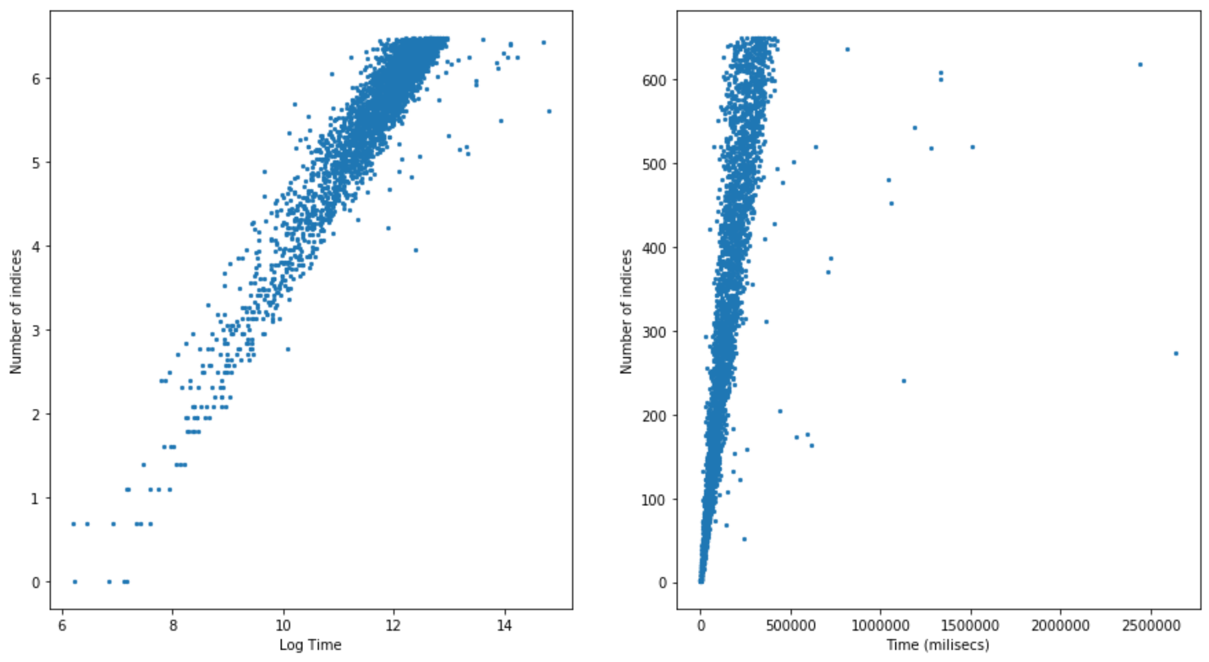


Figure 14. Number of indices vs Logarithmic Response Time

Number of buckets - Type of Aggregation VS Response Time

Our hypothesis was that the number of buckets is somehow related to the type of aggregation we are doing. And indeed, in the following scatter plot we identify some clusters. Just as a reminder the type of aggregation can be of two kinds:

- Based on histogram: then the subtypes will be either day, week, quarter, month or year.
- Based on top: for a certain parameter (*OnD*, *origin*, *destination*…) we get the top X elements. X can be 5, 1000, 5000 or 10000.

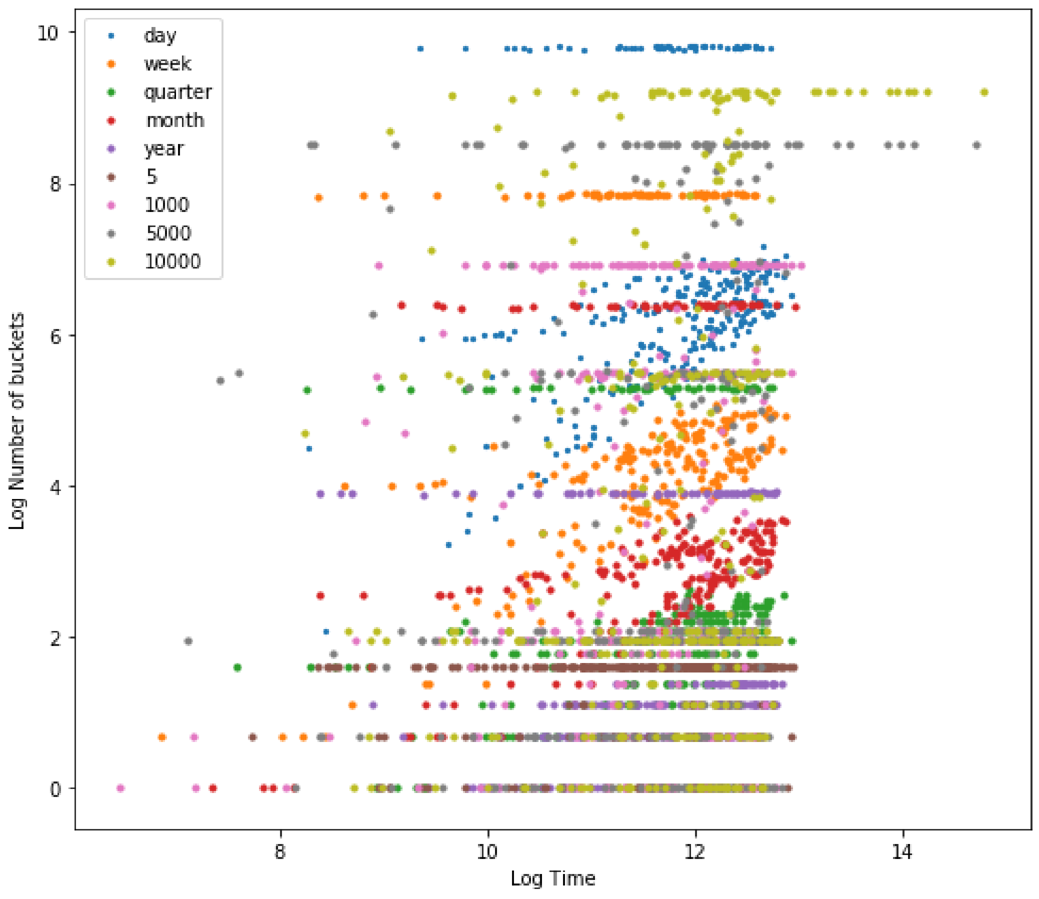


Figure 15. Buckets vs Response time

We do not observe a clear relation between the logarithmic response time and the number of buckets, except for some specific classes like day and week (blue and orange) in the plot that seem to be somehow linearly related. Besides we observe that some 5000 and 10000 Top queries have larger response time than the rest. These queries are Top *OnDs* queries. As the number of possible *OnDs* is very big they take longer to execute.

Sampled data

We can now introduce the sampling ratios. As explained in the features table this is the inverse of sampling factor. Thus it can go from 0.001% to 100%. The sampling ratio is just another input parameter for the query. One important question to solve is to which sampling ratios to choose. It seems intuitive that we want this value to be closer to 0 than to 100 as sampling 80% of data would not have a big impact on Response Time neither on Hits. For this reason, we have chosen a logarithmic scale for the sampling ratios so that we can assess how they behave in values close to 0. The following plots help us to find this out. For the sake of clarity in the following plots we have only considered 5 sampling ratios the ones used in the first iteration: 0.01%, 0.1%, 1%, 10% and 100%.

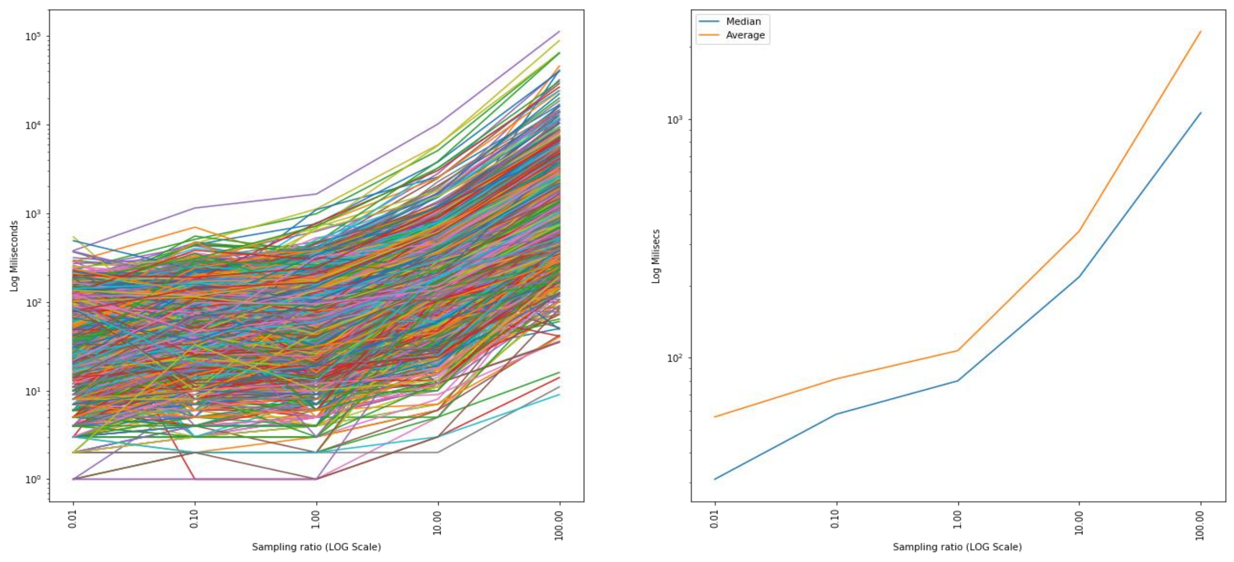


Figure 16. Response Time per Sampling Ratio

The main insight is that there is not much Response Time decrease when less than 1% of data is sampled as we see in the elbow-shaped plot on the right. Besides the maximum response time for 1% is around 3 seconds. This implies that 1% could be our minimum sampling ratio.

Another key indicator is the Sampling Error. This is a metric and cannot be considered in a hypothetical learning or predictive strategy. However, it must be considered especially for the selection of the Sampling Ratio. The way it is computed has been explained in Section *3.1.4 Features.*

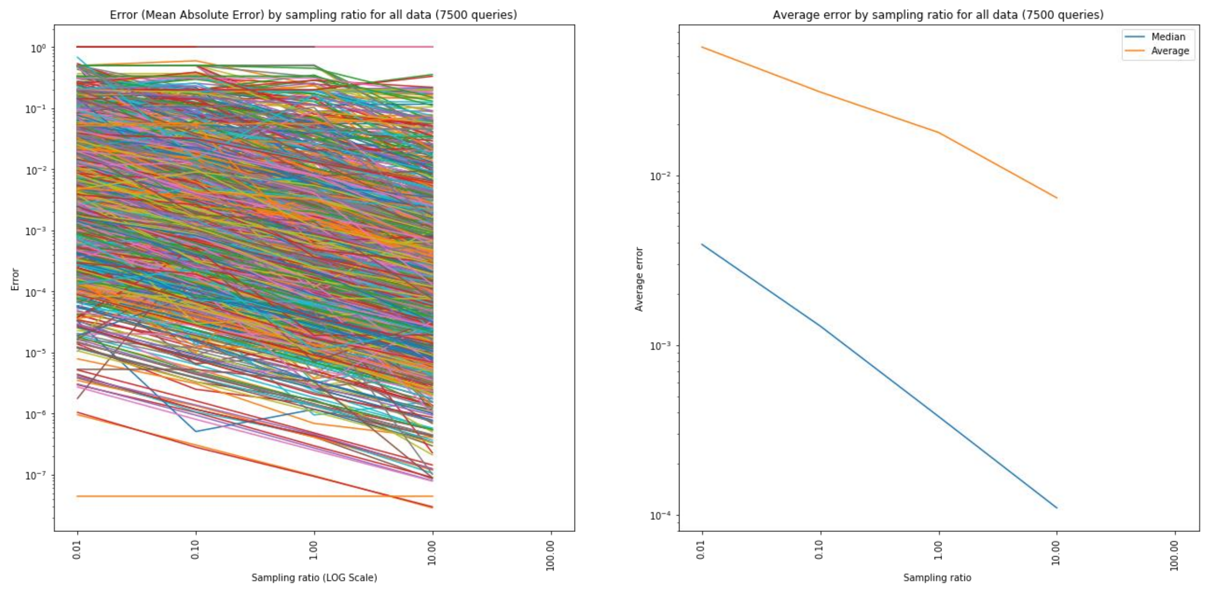


Figure 17. Error per Sampling Error

We should consider that the sampling is done randomly thus we can always find cases where the error is quite high and they error should typically decrease with a lower sampling factor, however this for a specific query the error could increase. For these reasons the median is a more representative metric than the average.

On the top of the left side plot there is a set of queries which always have error 1. These are queries with a low number of buckets and when we sample although sampling a 10% we get a null result, so the error will be 1. Trying to sample correctly those queries will be one of the main points of the prediction.

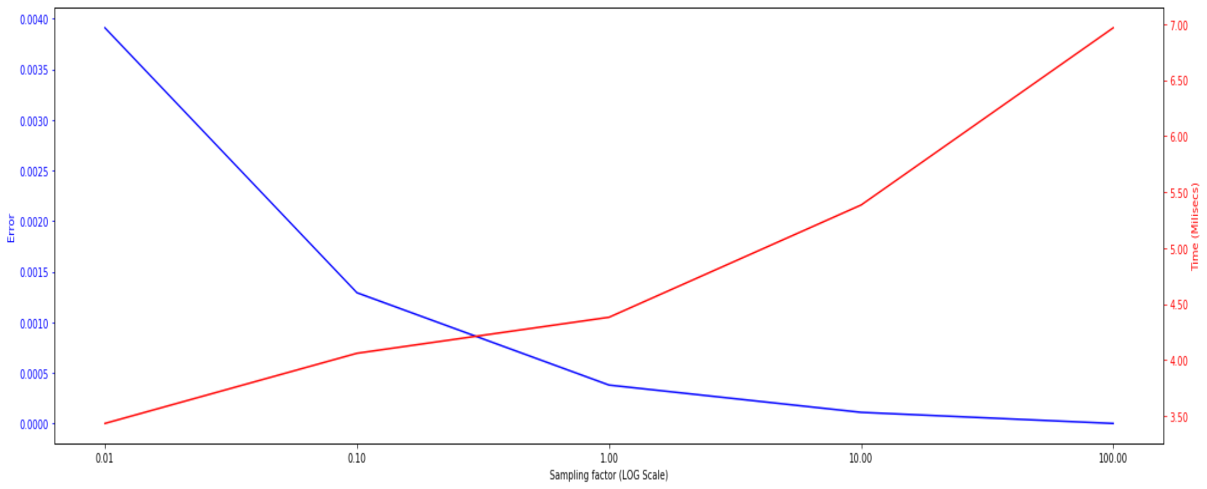


Figure 18. Logarithmic Medians Response Time vs Error

There is an interesting trade-off between the Sampling Error and Response Time over the different Sampling Ratios. The interesting point is the intersection point between both lines. As that the optimal point to for the sampling ratio. A good and plausible solution would have been to find a point at which they always intersect, however they idea is more conceptual than practical as this point varies depending and the amount of days considered, if take into account the logarithmic error or the log response time.

Filtering powers

We now want to show that the filtering powers actually gives some meaningful information. First, we want to note that we are aware that the filtering powers are an estimation of the number of hits. However, as the number of hits would not a plausible predictor for production we need somehow to estimate its value.

They are not an accurate solution since it is not true that the concatenation of two filters will result as the multiplication of both *power filters*. The reason is that the two filters are not statistically independent.

For instance, suppose that 10% of all searches have origin Nice and 20% of searches have destination Spain. Then, *origin city* Nice power filter equals 0.1 and destination country Spain power Filter is 0.2. Then, our method would output a main power filter of 0.02 for the concatenation (multiplication) of both. However, it is not necessarily true that a 2% from all searches are from Nice to Spain.

A precise solution would imply to number each search Nice-Spain and divide by the total number of searches. But, this is not feasible as the number of possible combinations is very large. Note that the number of filters can be up to 4, the number of parameters is a round 20 and the values that these parameters can take can vary from 2 to 9000.

It is very import for us to know whether this estimation is good. We plot the relation of the Filtering Power vs the number of Hits for each query of our dataset.

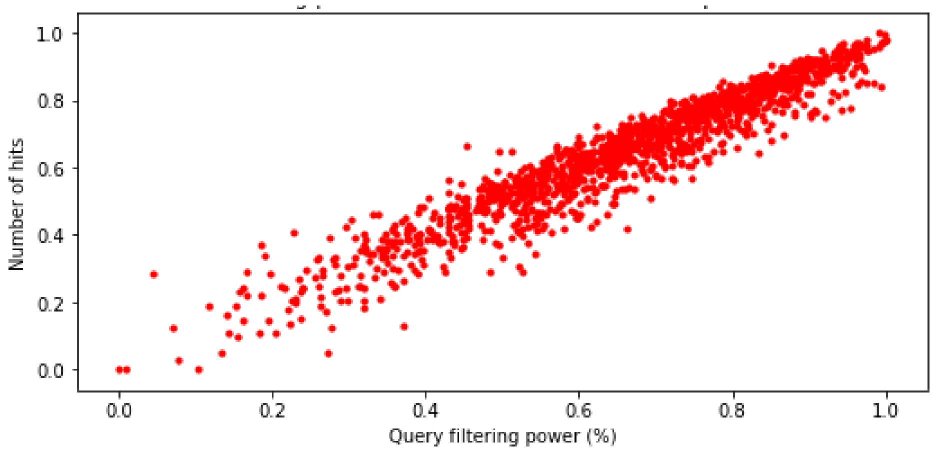


Figure 19. Filtering Power vs Hits

Indeed, the linear relationship is clear. However, we cannot see this relation in the filtering power versus Response Time plot, as these relations was not very sharp for Hits versus Response time when many indices were included, now that there is bit more noise is even more fuzzy.

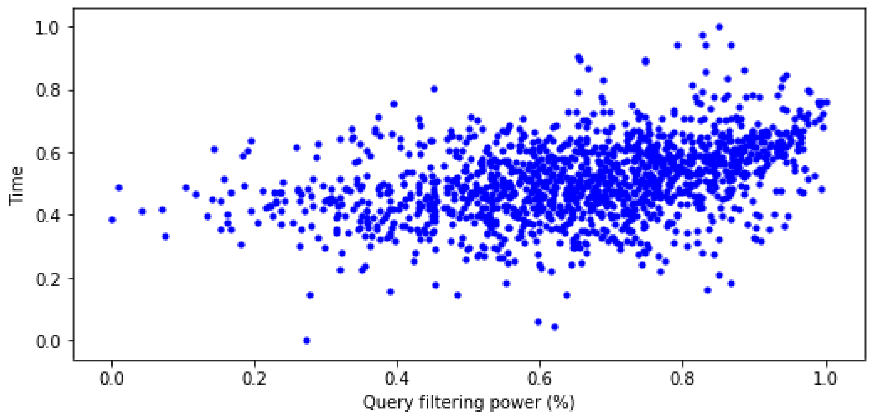


Figure 20. Filtering Power vs Time (Normalized)

3.2.1. Features dependencies

Our first task for selecting the most important features in our dataset is plotting the correlation matrix. The correlation is a statistical measure that gives the degree of dependence of two random variables. The dataset for the following plot contains 3 sampling ratios and 3000 queries for each sampling ratio. We have only considered those features that could be considered to be in production.

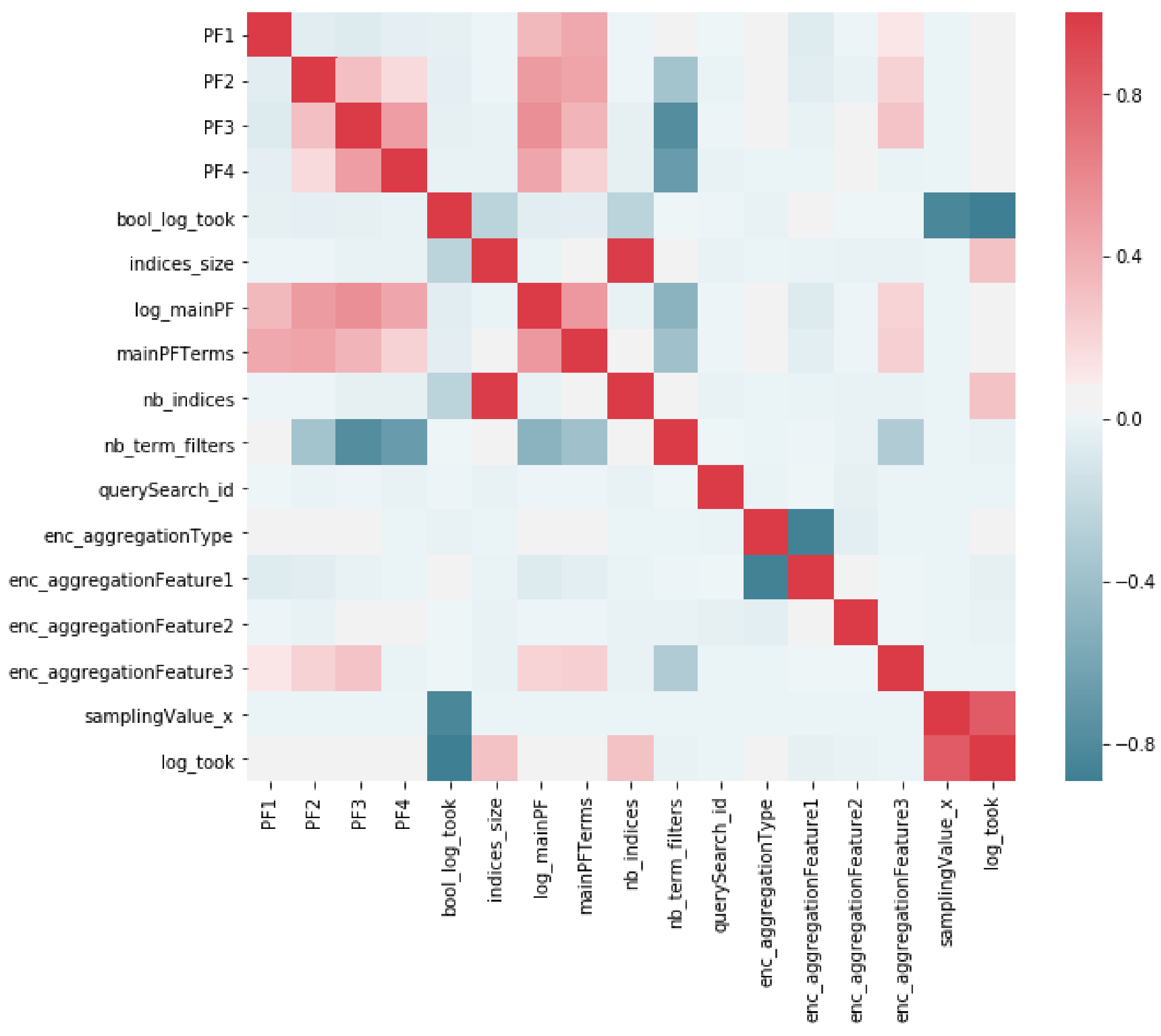


Figure 21. Correlation Matrix

We take some insights in the form of bullet points:

- Sampling Ratio and Response Time are highly correlated.
- *Bool_log_took* is the Response Time Bins attribute and is also highly correlated to the sampling value and of course the response time.
- Indices size and number of indices are highly correlated, most likely only one of them would be necessary. And they are also correlated to the Response Time bins.
- It seems that all power filters are related among them, this is because the values that they usually take lie within the same range.

3.2.2. Variance of Response Time

The idea behind this section is to understand the variability of the response time for one specific query. Due to the latency of the network, one query does not have the exact same response time if executed twice. Although we expect the values to remain within the same range. Furthermore, when testing this by shooting the same query consecutively, we come across with the cache problem. This means that the queries after the first might have lower response time.

For this purpose, it is essential that we consider how the cache works and behaves in Elasticsearch. There are different kinds of caching strategies in ElasticSearch, which this project does not cover. However, on the one hand, previous studies have shown that ElasticSearch cache works very well, thus disabling completely the cache to have solution that forgets about caching would be counterproductive and far from reality. On the other hand, if we repeatedly call the same query the sored response will be biased towards a decrease. The solution is to fill in the cache with intermediate queries.

We have done two parallel analysis:

- Using sampled data: XXX queries for each of the 3 sampling ratios and
- Using non-sampled data

In conclusion, we run 3000 different queries with the 5 sampling ratios and we repeat this process 5 times. We expect that 15000 queries are enough to fill in the cache completely. We now plot the first 50 queries for our largest sampling ratio (10%). Each line represents one of the 5 times the query is shot.

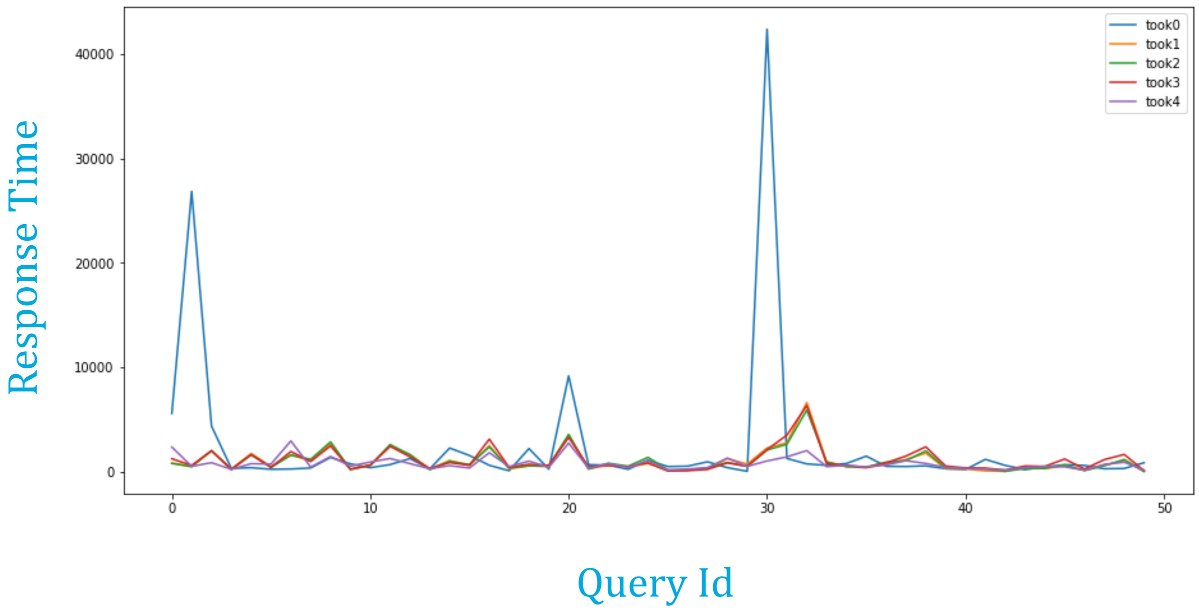


Figure 22. Query variance analysis

We see that although this variability exists the response time of the query lies within a range this is more evident when the sampling ratio is higher as the response times are also higher. We also observe some outliers, most likely timeouts. However, this can be controlled when in production. So we decided to remove them for the following table.

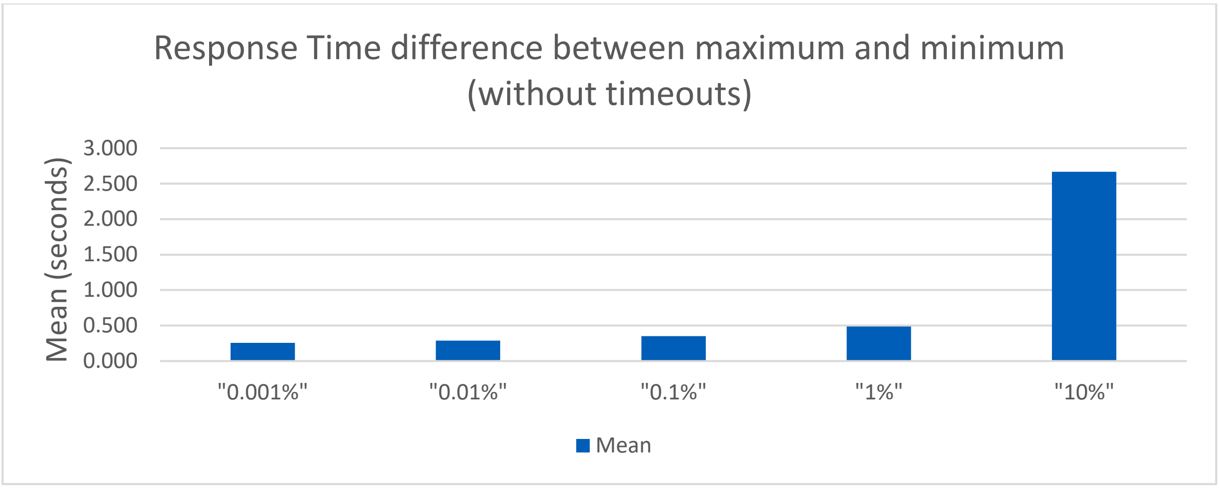


Figure 23. Response time differences

The response time without timeouts gives almost always response times below 5 seconds and a 1% sampling are enough to guarantee being below 5 seconds. We must note that although sampling ratios below 1% are targeting the same number of shards the response is reduced as the number of hits is diminished.

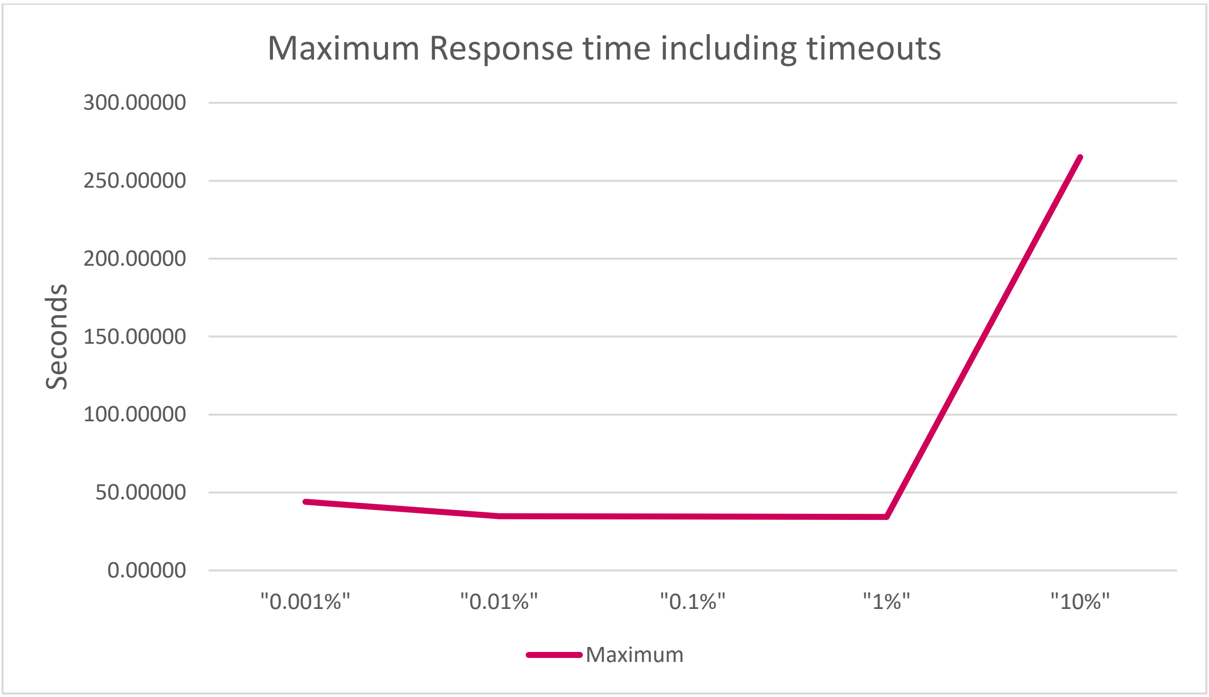


Figure 24. Maximum Response times

We observe that the maximum response time are around 35 seconds even for very low sampling ratios.

3.2.1. Shards vs Indices

This section wants to inspect if there is any effect or impact on the response by increasing the number of targeted shards in two datasets, each of the in a different way:

1. Increase the number of indices and sampling values constant.
2. Increase the number of involved shards and number of indices constant

For each one, we run two datasets having 64 kind of queries and 10 queries for each kind of queries. These are spread the following way:

	Number of indices	Sampling Values	Targeted Shards
Dataset 1	64	1562, 3124, … , 99968	64, 128, … , 4096
Dataset 2	1, … , 64	99968	64, 128, … , 4096

Figure 25. Shards vs Indices comparison table

We now plot the response times per number of targeted for the median of the type of query:

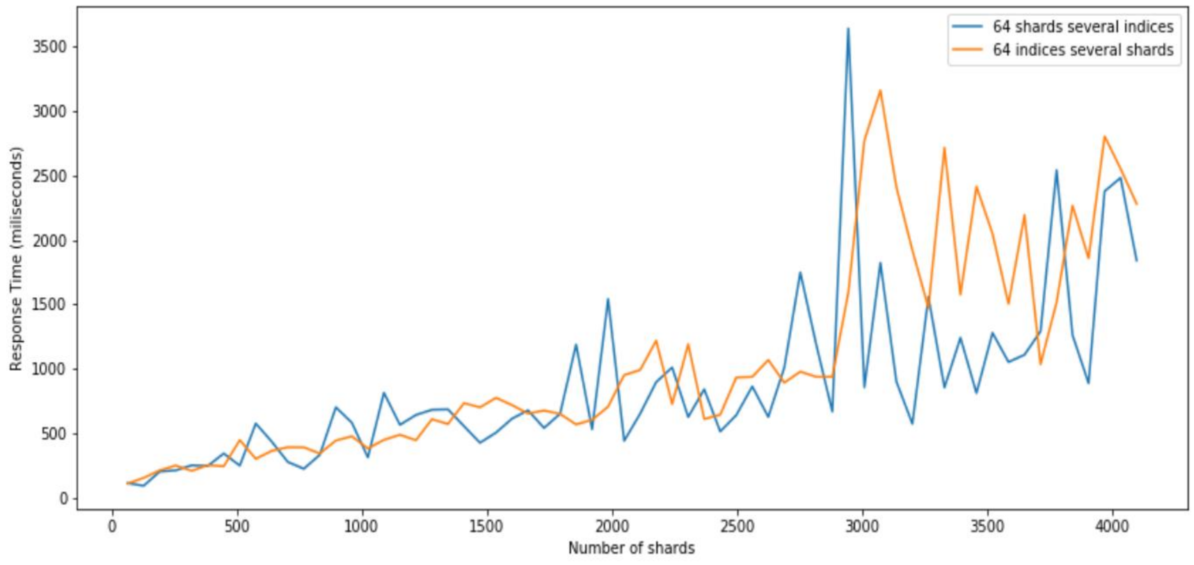


Figure 26. Response time Indices vs Shards increment

We see no great difference in the response time. Although the variance when the indexes are fixed seem to be lower. Although this study does not bring immediate consequence to our problem, we must note that according to this if the number of indices continues growing, and it definitely will, the response time will also increase.

3.3. Proposing Solutions

Now can frame our problem in one sentence:

"For each query choose a sampling factor such that the response time is below 5 seconds and the sampling error is minimized."

Apart from the visualizations and conclusions and insights showed above we want to keep in mind that the ultimate goal is to take this project into production. This implies the following:

- As there is no user generated data yet. The training data for the model has to be artificial, at least in the beginning. This implies that training data can be as large and general as possible.
- The extraction of too many features from different sources could difficult the put into production stage.
- The model must be understandable. A too complex solution could not be correctly understood and produce delays. A simple model can always be refined with time.
- The model can be trained before put being put in production for a long time. However, when in production the overhead the prediction may cause should be minimal.

3.3.1. The Three Approaches

As one of the main goals of the project is to evaluate different strategies for live data sampling we want to go through the three initial strategies (and its varieties) we had in mind.

Predicting the Response Time

The first approach is more intuitive. It consists on switching the variable to predict (sampling ratio) with a predictor variable the response time. Our training dataset will contain one query per row, each row should have query attributes, response time and a sampling ratio. Then, we want to predict the response time of a query given the query features and a sampling ratio. When in production we will predict several sampling ratios for the same query, and we will choose our sampling ratio the one that has a response time below 5 seconds. The model would be looks as follows:

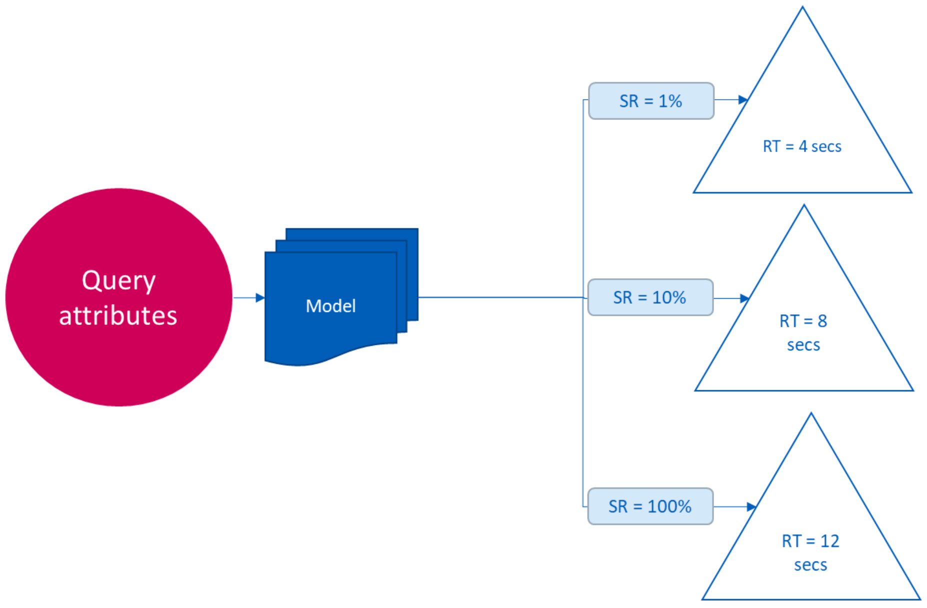


Figure 27. Predicting the Response Time schema

This is the conceptual idea, still we would need to define how many and which are the best sampling ratios. Besides, as we will show we will categorize the response time.

Predicting the Sampling Ratio

The second approach consists on switching Sampling Ratio and Response Time. So, our dataset would be built similarly. However, when predicting in production we would make a set of predictions, each containing a response time, and the output would be the sampling ratio. The model will be a function as follows:

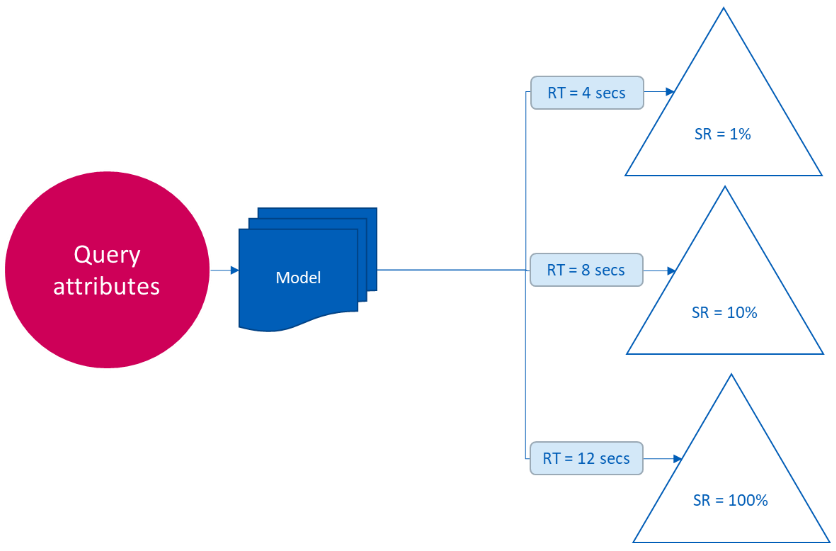


Figure 28. Predicting the sampling ratio schema

This approach has two main drawbacks:

- A) The range of values that the response time can take is not known. Thus, we could be having a large error if we do not choose smartly the values for which we want to predict, especially the maximum, as if the maximum response time is much below the actual response time, the errors would be large. So, the solution is to study beforehand which are the possible values that the response time can take and choose a representative set.
- B) The variable number of shards cannot be included in our dataset. The number of shards is computed as the number of indices times the sampling ratio. Now, we want to predict the sampling ratio so the number of shards cannot be included as a predictor variable.

Again, the set response times to predict would need to be defined.

Making a sampled query before predicting

The last approach we consider is the based on enriching the dataset with more variables. Concretely, by making a sampled (thus, very short) query that allow us to estimate the attributes related to the response of the query, such as: number of hits, number of shards, and number of buckets... Then, we would predict more accurately the response time or the sampling ratio. Finally, we would make a second query with an appropriate/predicted Sampling Ratio. This graph simplifies the idea:

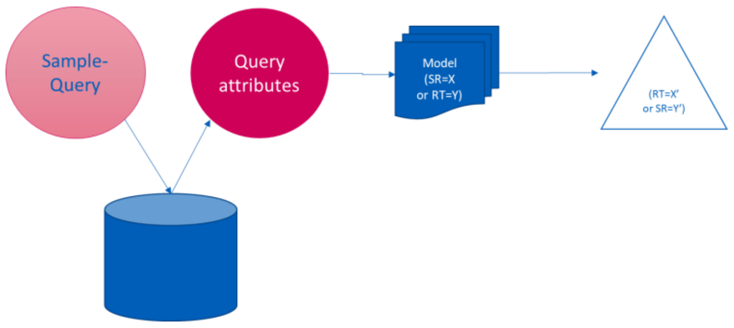


Figure 29. Sample query approach schema

The main risk of this strategy implies is that the sample query takes too long and has an impact on the total response time for the user.

From the Section 3.2.3 Variance of the Response we can also withdraw some conclusions to decide whether shooting a previous query is a good strategy. The idea is to see what the response times are with lowest sampling values. We observe that without considering timeouts and having between 1 and 4 filters which is quite optimistic, the response times are:

Sampling Ratio	Mean (no timeouts)	Maximum (no timeouts)	Maximum (including timeouts)
0.001%	**0.258 secs**	**4.410 secs**	**44.02 secs**
0.01%"	**0.287 secs**	**5.034 secs**	**34.73 secs**

Figure 30. Response Time for low Sampling ratios

Although mean values seem acceptable we consider the maximum to be above user-friendly values and thus we want to assess other strategies before taking risk of having response times over 30 seconds. Although they could be controlled when in production.

3.3.2. Assessment

To choose one of the approaches explained above we are going to compare the accuracy of predicting the response time versus predicting the sampling ratio. Later

, we will compare different models and finally we will show the possible scenarios.

3.3.2.1. Choose the predicting variable

Choosing between the two approaches can be a complex decision. That is why we are going to simplify to the maximum our problem. In the end, we want to know if it is better to choose M1 or M2 where they are defined as follows:

$$(SR, X) \rightarrow M1(X)(SR) = RTB$$
$$(RTB, X') \rightarrow M2(X')(RTB) = SR'$$

For this comparison we have defined the following:

- RTB are the Response Time Bins. There will be 3 categories:

- RT<5 seconds: LOW
- 5<RT<15 seconds: MID
- RT>15 seconds: HIGH

_ SR. the set of sampling ratios will be

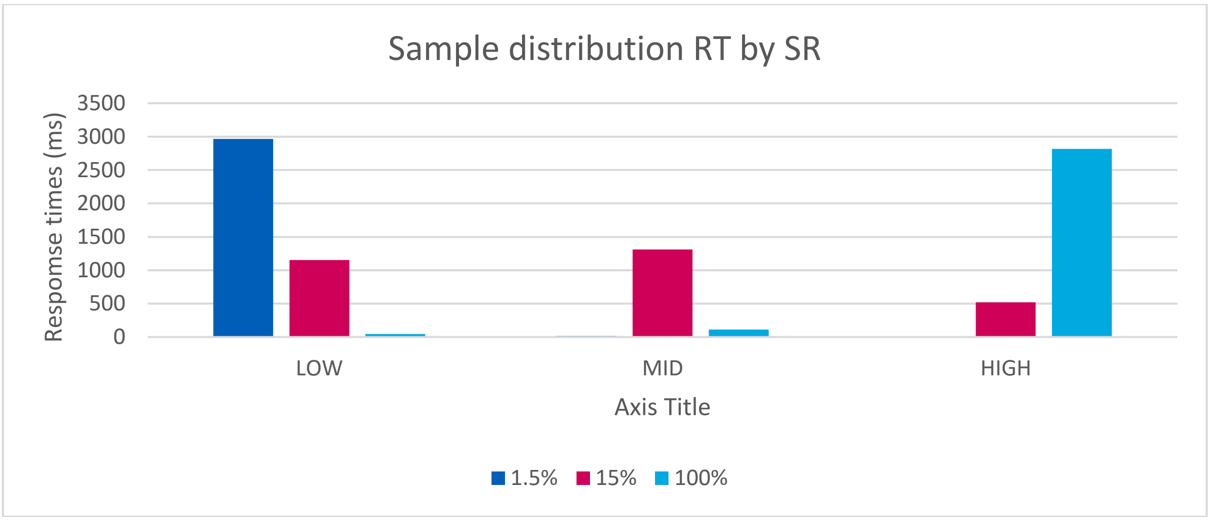
- 100%, i.e. non-sampled data or sampling factor = 100000.
- 15%: sampling factor = 15620.
- 1.5%: sampling factor = 1562.

_ X and X'. These are the attributes that can be used in one or the other strategy. We show the comparison in the table XX. The only difference is that predicting variables are switched and in both cases with the strategy of predicting all the classes in production they can be used for learning.

_ Models. We will use the Scikit-Learn implementation Logistic Regression with default values. No parameter fine tuning for simplification.

_ Metric. Our KPI of interest will be the precision of the RT<5 class for M1 and the recall of the 100% class for M2. Anyways, we will show the full confusion matrix.

_ Dataset. A summary of our dataset shows the following:



_ Results: The confusion matrix of our models show the following:

Predicting the Response Time (M1)		**Predicted Value**					**KPIs**			
		HIGH	**MID**	**LOW**	Total			Precision	Recall	Support
Real Value	**HIGH**	**1059**	**68**	**6**	1133		**HIGH**	0.91	0.93	1133
	MID	**80**	**280**	**139**	499		**MID**	0.76	0.56	499
	LOW	**19**	**19**	**1411**	1499		**LOW**	0.91	0.97	1499
	Total	1158	367	1556	3081		Average	0.89	0.89	936

Figure 31. Predicting RT (M1) table

Predicting the	**Predicted Value**		**KPIs**

Sampling ratio (M2)		**1.5%**	**15%**	**100%**	Total			Precision	Recall	Support
Real Value	**1.5%**	**873**	**154**	**0**	1027		**1.5%**	0.77	0.85	1027
	15%	**261**	**639**	**127**	1027		**15%**	0.75	0.62	1027
	100%	**0**	**63**	**964**	1027		**100%**	0.88	0.94	1027
	Total	1134	856	1096	3081		Average	0.80	0.80	3081

Figure 32. Predicting SR (M2) table

Both results give similar measurements. It seems that predicting the sampling ratio model outperforms the predicting the response time model. However, this small difference can be due to the unbalanced classes in the first model, this is observable by looking at the support.

It is a difficult decision to decide among both models. But, we believe that we should go forward by **predicting the sampling ratio**. For two reasons:

1. We can more easily obtain balanced classes. This does not imply that the unbalanced problem will necessarily incur in a worse performance. However, considering that we have the opportunity to use a dataset with natural balanced classes we believe it is best option. To justify that unbalanced classes might perform worse we transcribe the explanation given in the article "Why it is important to work with a balanced classification dataset" *[15]*:

   > *[…] Machine learning classifiers such as Random Forests fail to cope with imbalanced training datasets as they are sensitive to the proportions of the different classes. As a consequence, these algorithms tend to favour the class with the largest proportion of observations (known as majority class), which may lead to misleading accuracies. This may be particularly problematic when we are interested in the correct classification of a "rare" class (also known as minority class) but we find high accuracies which are actually the product of the correct classification of the majority class (i.e., are the reflection of the underlying class distribution). […]*

   Note that although the model's comparison has been done using Logistic Regression and not with a Random Forest the majority voting technique is still used in this algorithm for multinomial classification.

2. It seems more intuitive to predict the sampling ratio, as it is the value that it will immediately be applied. As a consequence, more simple for production.

However, we cannot not affirm that a strategy based on predicting the response time would not work.

Features Importance Comparison

First, we want to motivate the reason why not having a particular section for feature engineering, because although it is critical in a machine learning implementation we consider that the metrics give enough good results and removing attributes would be counterproductive in future when more days/ indices are included and when the dataset is updated with real user queries.

Nonetheless we want to give an insight on how the features behave in our models. We show the features importance compute from the *Extra Trees Classifier* implemented in python.

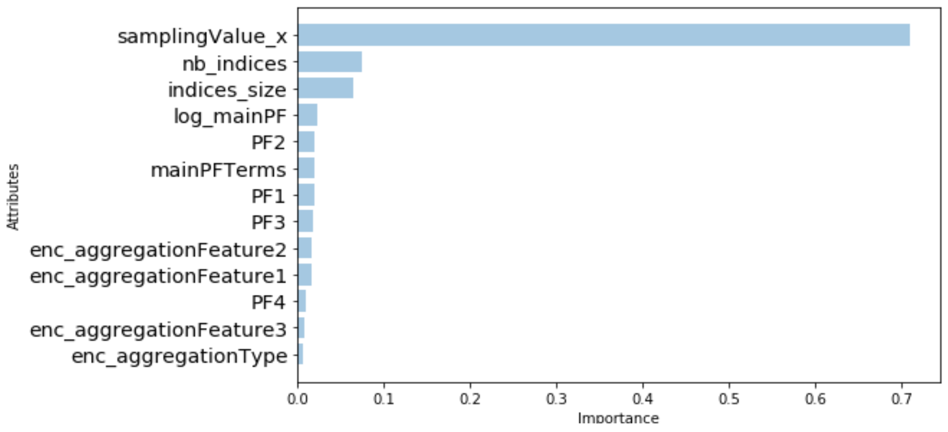


Figure 33. Predicting Response Time Bins

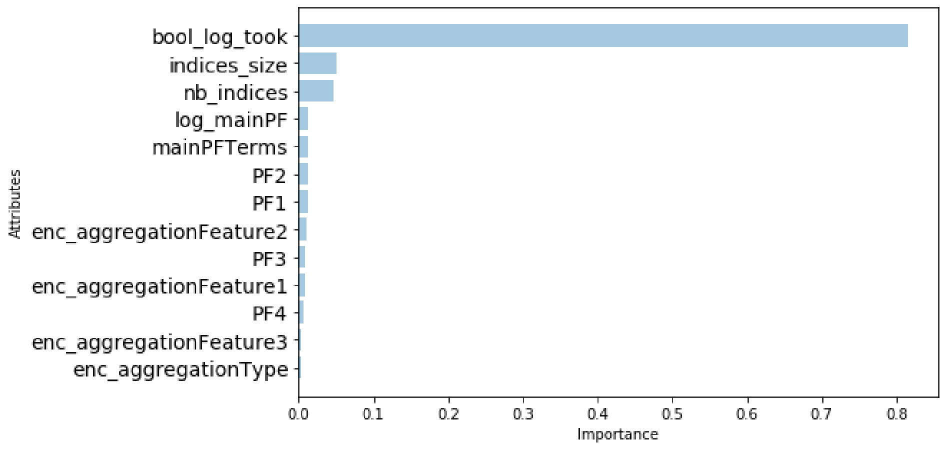


Figure 34. Predicting Sampling Ratio

*bool_log_took = Response Time Bins

Some insights from the plots:

- Response Time bins and Sampling Ratio are by far the most relevant features, respectively.
- The indices' size and number are also important features. Their order of importance is switched.
- The main power filter- the multiplication of the power filters is also an important feature. The estimation is again not bad.
- PF2 stands above PF1. As the number of filters can vary from 1 to 4. The first filter loses a bit of importance.

3.3.2.2. Classifier Selection

Although in the previous section we have already used a Logistic Regression model. It is important that we make sure that other models, especially those that have high interpretability, are not outperforming the logistic regression. For this reason, we show a comparison of logistic regression, decision tree and random forest.

We observe that there is no real gain in any of the model. All of them have similar performance metrics. All three classify quite well the un-sampled queries, while it has more difficulties in distinguishing between the 1% and 10%, our guess is that the most important feature for learning is the response time and thus the response for low sampling ratios are more similar, thus fuzzier for the algorithm.

Since the data does not provide enough reason to chooses one model or another. We will have to base our decision on the essence of each of the three algorithms:

1. Logistic regression. It is one of most basic learning algorithms, it is based on multiple linear regression and uses to the logistic function to make the final classification, which allows to classify either as 1 or 0. Since in our case is multinomial, the strategy implemented used to combine the logistic functions is one-vs-the-rest.

Predicting the Sampling ratio LOGISTIC REGRESSION		**Predicted Value**					**KPIs**		
		1.5%	**15%**	**100%**	Total		Precision	Recall	Support
Real Value	**1.5%**	**873**	**154**	**0**	**1027**	**1.5%**	0.77	0.85	1027
	15%	**261**	**639**	**127**	**1027**	**15%**	0.75	0.62	1027
	100%	**0**	**63**	**964**	**1027**	**100%**	0.88	0.94	1027
	Total	1134	856	1096	3081	Average	0.89	0.80	936

Figure 35. Logistic Regression results table

2. Decision trees. It creates splits based on the variable with higher Gini coefficient, this is a metric for the dispersion of the values of the variable. The issue is to decide the depth of the tree as if it is too deep, it can be overfitting, which means the generalization of the model is not good. On the other hand, if the tree is too shallow then it can predict incorrectly quite often – underfitting.

Predicting the Sampling ratio DECISION TREE		**Predicted Value**					**KPIs**		
		1%	**10%**	**100%**	Total		Precision	Recall	Support
Real Value	**1.5%**	**1021**	**6**	**0**	**1027**	**1.5%**	0.70	0.99	**1027**
	15%	**412**	**553**	**62**	**1027**	**15%**	0.73	0.54	**1027**
	100%	**19**	**194**	**814**	**1027**	**100%**	0.93	0.79	**1027**
	Total	1452	753	876	**3081**	Average	0.79	0.97	**3081**

Figure 36. Decision tree result table

3. Random Forest. This is an ensemble method of decision trees, which means that produces several decision trees, each having a different subset of attributes. Then, it outputs the result by getting the majority vote from all created trees. Not to be confused with an ensemble of extra trees classifiers where several trees are combined but using different subsets of the instances. The purpose of the random forest is to obtain a better generalization and robustness as the trees will have different depths.

Predicting the Sampling ratio RANDOM FOREST		**Predicted Value**					**KPIs**			
		1%	**10%**	**100%**	Total			Precision	Recall	Support
Real Value	**1.5%**	**886**	**139**	**2**	1027		**1.5%**	0.75	0.86	1027
	15%	**284**	**600**	**142**	1027		**15%**	0.77	0.58	1027
	100%	**10**	**38**	**980**	1027		**100%**	0.87	0.95	1027
	Total	1180	777	1124	3081		Average	0.80	0.80	3081

Figure 37. Random Forest results table

It is commonly believed that linear regression performs better for continuous variable as the transformation from categorical values to dummy variables might ignore valuable information:

> *DTs can work with both numerical and categorical data directly, which is not the case for numerical classifiers such as linear classifiers or neural networks, as these methods require the data to be real-valued (and ordinal). For example, if a categorical feature can take three values such as (i) red, (ii) blue, or, (iii) yellow, it is often represented by a group of three binary features such that one of these features takes the value 1 while the other two are 0. A numerical classifier would treat this group of three features independently where any combination of 0/1 values are possible - ignoring the valuable information that only three values for the triplet are possible.* (Günlük, Kalagnanam, Menickelly, Scheinberg, 2018)[13]

This is the main reason to choose Decision trees or Random Forests over Logistic Regression.

The choice between Random Forest and Decision trees apart from being based on our metric results can be based on previous research comparisons. Literature says that Random Forests outperform Decision Trees, as usually ensemble methods do [14]. The only reason we would opt for the decision trees is that Decision are easier to understand, i.e. they are more interpretable. However, we consider that the exploratory analysis done already gives meaning to the data.

3.3.2.3. Number of classifiers

One of the key questions to answer to validate our strategy is, since when we are doing 3 predictions in production for each of the response times bins: do we need one classifier per Response Time Bin? Or is one classifier including all Response Times bins is enough? In terms of simplicity it will be much easier to consider only one: less parameter fine tuning and more interpretability. However, if the results show that there is big gain in model performance we would have to consider creating several classifiers.

In order to answer this question, we are going to compute an approach with 3 different Random Forest classifiers, one for each sampling ratio and compare results with the table XX that shows the result from the model using one Random Forest classifier.

Predicting the Sampling ratio RANDOM FOREST		**Predicted Value**			
		1%	**10%**	**100%**	Total
Real Value	**1.5%**	**714**	**294**	**6**	1014
	15%	**364**	**494**	**168**	1026
	100%	**6**	**156**	**879**	1041
	Total	1084	944	1053	3081

KPIs			
	Precision	Recall	Support
1.5%	0.66	0.70	1014
15%	0.52	0.48	1026
100%	0.83	0.84	1041
Average	0.80	0.68	3081

Figure 38. Random Forest several classifiers results table

The results show that there is not gain in this approach, 0.84 Recall for the non-sampled class versus 0.95 in the random forest above.

It is noteworthy that the metrics 1 classifier versus are not totally comparable as the support of each class varies a bit. The reason for this is the following: when we separate the training set and test set in the previous models (1 classifier) we do it by the query id. One query id is repeated 3 times, one per sampling ratio. So, we guarantee that the three repetitions of the query are all either in the test or training set. However, when have three classifiers, each repetition of the query goes to a different classifier, once a classifier has been initiated the train and test split occurs, and the random state of the splitter changes, so in the following classifier the split is different. In conclusion, we cannot guarantee that the 3 repetitions are all in test or train set, consequently the support of each class for the test set varies. Although we have tried to maintain the overall support value.

3.3.3. Number of Sampling Values

Now that we found a model that predicts fairly we can think of ways to reduce the sampling error. One possibility is to add more or different sampling values. Instead of having only 3 (100%, 15% and 1.5%) we can have more in between values. The idea is that those queries which are being sample to 1.5% might be taking less than a second and it would still be user friendly to run the same query in two seconds if the sampling error is diminished.

To make sure this strategy works, we need to make sure that the sampling error is decreasing and that the classifier's error is not dropping. For this purpose, we compare three models:

Model	Number of sampling values	Sampling Ratios	Sampling factors	Number of shards per index
1	3	1.5% 15% 100%	1562 15620 100000	1 10 64
2	5	1.5% 6.2% 15% 25% 100%	1562 6248 15620 24992 100000	1 4 10 16 64

3	7	1.5%	1562	1
		3%	3124	2
		6.2%	6248	4
		15%	15620	10
		25%	24992	16
		50%	49984	32
		100%	100000	64

Figure 39. Number of Sampling values comparison table

To make this comparison will use several metrics:

1. To measure the sampling error:
    1.1. **Median sampling error** of several queries as explained above (MAE). The issue with this metrics is that it has very small values since many queries have 0 sampling error.
    1.2. **Percentage of queries** with an error higher than 0.001.
2. To measure the prediction error, i.e. the classifier's error:
    2.1. **Recall** of the non-sampled class

In the three cases we used the same attributes and a Random Forest. The results are in the following table:

Model	Median Sampling Error	% of queries with an error higher than 0.001	Recall
1	3.05975E-05	11.3%	0.95
2	2.37638E-05	9.5%	0.75
3	2.27072E-05	9.6%	0.51

Figure 40. Number of sampling Values results comparison table

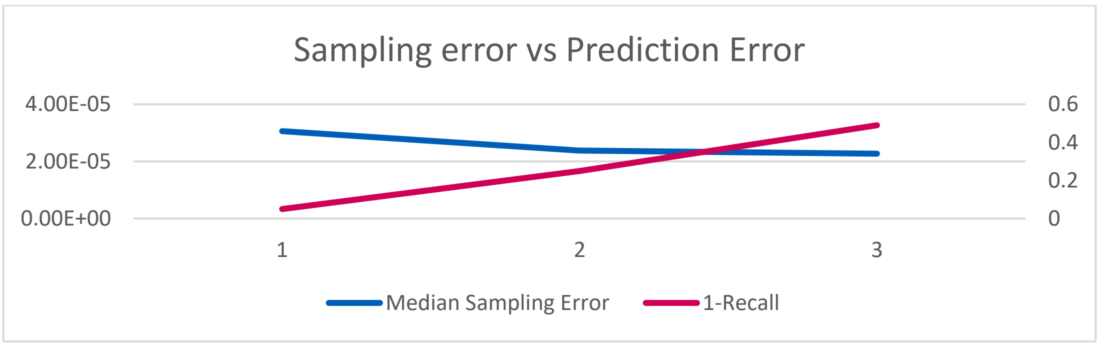


Figure 41. Sampling vs prediction error

In the plot and table, we observe the trade—between the sampling error and the prediction error. The idea behind this plot is that if we increase the number of sampling values the sampling error will decrease as we know that we can sample less and still being in a low Response Time bin. However, our random classifier will do worse in that case; because the Response Time Bin is the attribute with more importance in the algorithm when many sampling ratios are applied the algorithm cannot distinguish properly. In the following plots we show the probability density distributions the response for different sampling ratios.

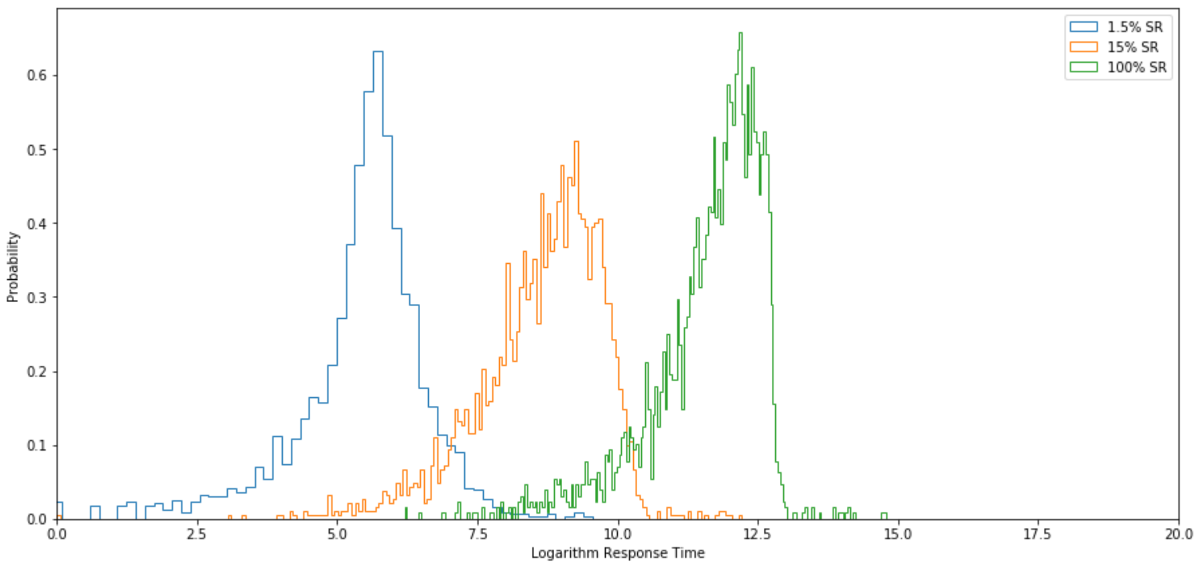


Figure 42. Logarithmic RT distribution for 3 sampling ratios

For the 3 sampling ratios the response times do not overlap. Although the response time cannot be used in production and we use the response time bucket, it is enough to see why the model is performing well.

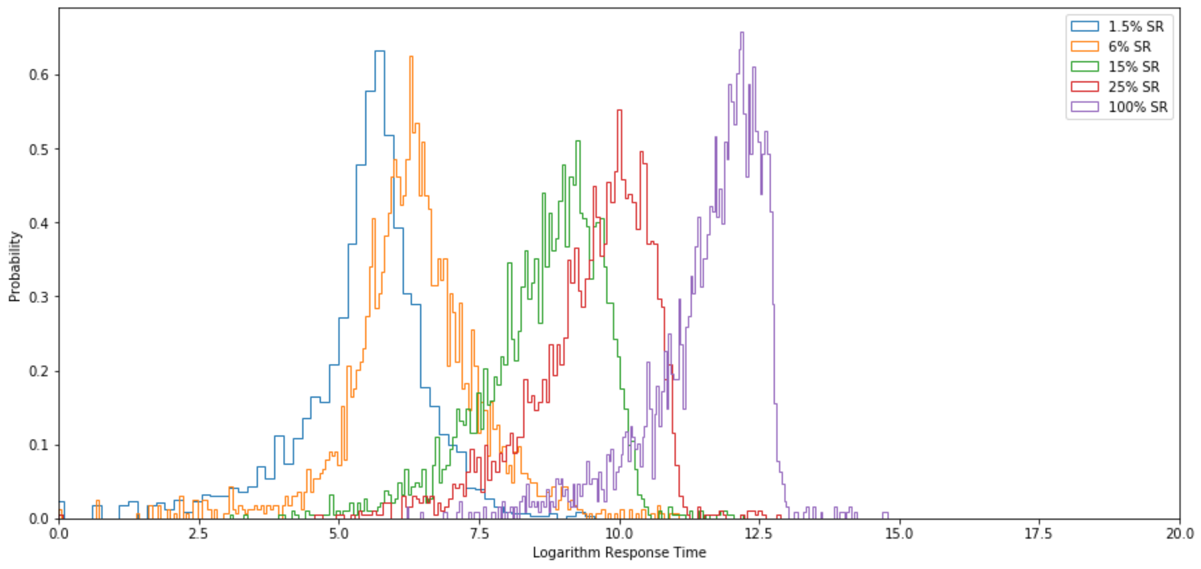


Figure 43. Logarithmic RT distribution for 5 sampling values

Now there is bit more overlap, thus the classifier will have more problems predicting correctly.

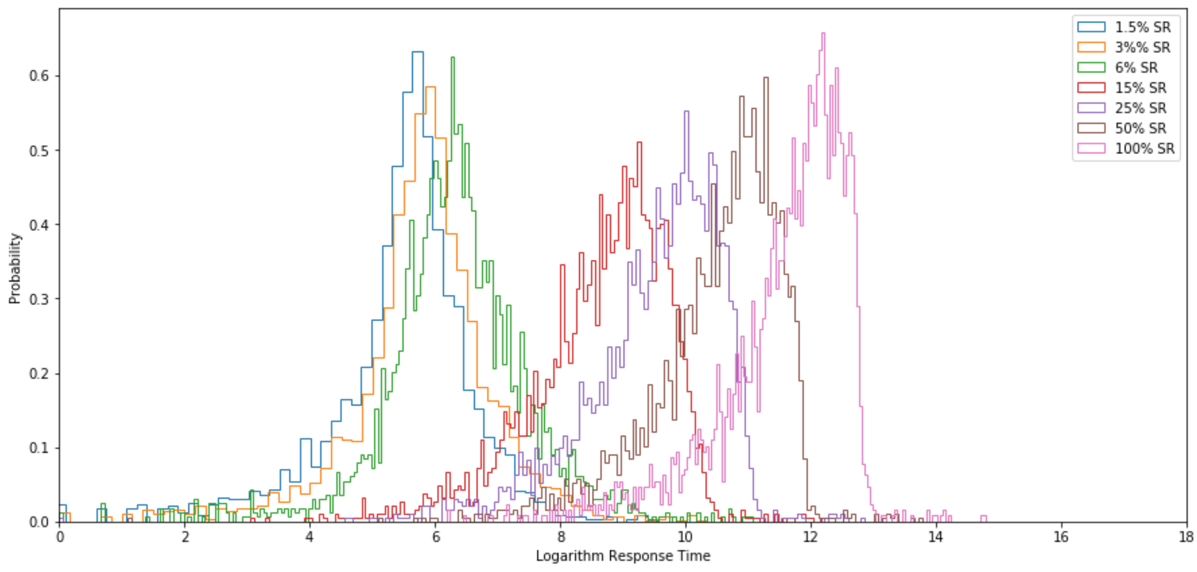


Figure 44.Logarithmic RT distribution for 7 sampling values

This plot clearly shows the reason why the classifier is performing worse with 7 sampling values.

3.3.4. Selecting the Sampling Ratio

Now, let's consider that we have already taken a decision for one of the proposed models in the section just above. For instance, 7 sampling values. The goal now is how we choose which sampling ratio to apply for each query. Bear in mind from *Figure 30. Predicting the sampling Ratio* that we predict 3 instances, one for each Response Time Bin (LOW, MID, and HIGH). Do we always want to apply the sampling ratio from the LOW Response Time Bin? Do we might want to choose the MID Response Time bin?

To have different scenarios that give us freedom to solve this question we introduce a new query attribute, we call it TAG. This is a numerical value that can be computed heuristically from the value of the classifier predicted sampling ratio and the Response Time Bin. The following table shows the tag values in case of 3 sampling values.

The "predicting the sampling ratio" schema has been updated in the following figure:

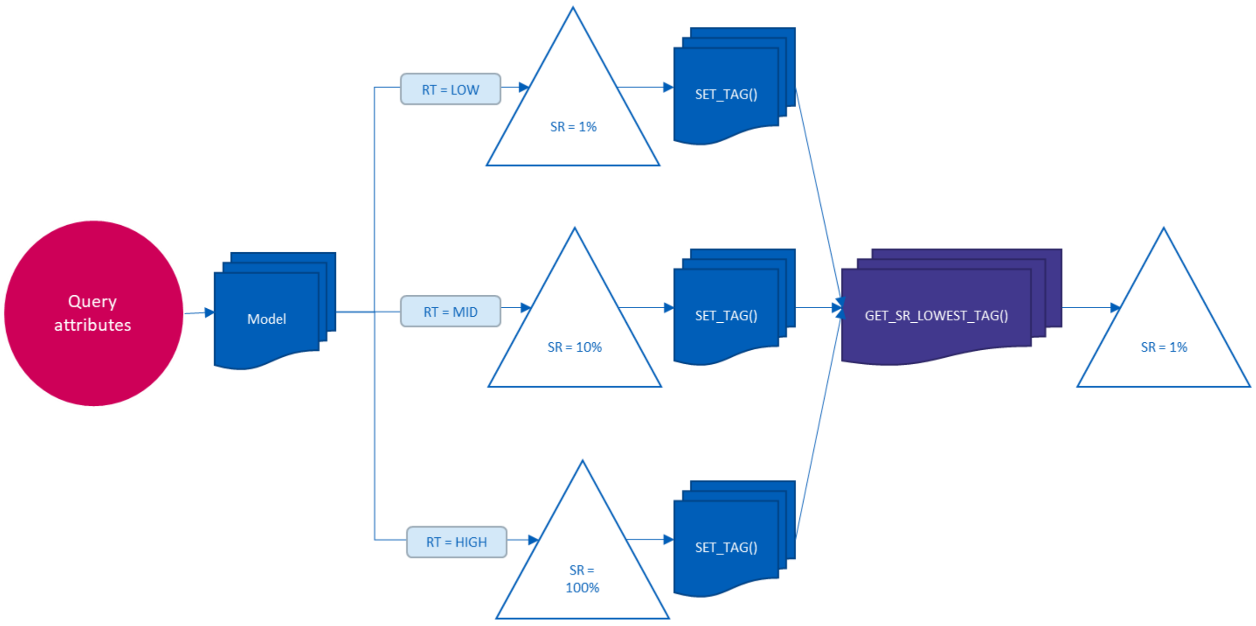


Figure 45. Updated predicted the sampling ratio schema

The "Set_tag" is an heuristic function defined as follows:

Standard set tag function		
PREDICTED SAMPLING RATIO	RESPONSE TIME BIN	TAG
100%	LOW	1
15%	LOW	2
1.5%	LOW	3
100%	MID	4
15%	MID	5
1.5%	MID	6
100%	HIGH	7
15%	HIGH	8
1.5%	HIGH	9

Figure 46. Standard *"set tag"* function example

Then, we can just the get the query repetition with the lowest TAG, so that the Sampling Ratio for our query is chosen. Now by modifying the *"set tag"* function we can switch from a conservative strategy where we prefer a low response time before a low sampling ratio. However, we can modify the function to go for a scenario where the low sampling error is preferred. We show two modifications of the function in the tables below:

Non-sample MID&LOW priority set tag function		
PREDICTED SAMPLING RATIO	RESPONSE TIME BIN	TAG
100%	LOW	1
100%	MID	2
15%	LOW	3
1.5%	LOW	4
15%	MID	5
1.5%	MID	6
100%	HIGH	7
15%	HIGH	8
1.5%	HIGH	9

Non-sample&15% MID&LOW priority set tag function		
PREDICTED SAMPLING RATIO	RESPONSE TIME BIN	TAG
100%	LOW	1
100%	MID	2
15%	MID	3
15%	LOW	4
1.5%	LOW	5
1.5%	MID	6
100%	HIGH	7
15%	HIGH	8
1.5%	HIGH	9

Figure 47. "Set tag" function alternatives.

When comparing these two scenarios, we need to add two key metrics:

- The percentage of queries that have a HIGH Response Time.
- The percentage of queries that have a MID Response Time.

The following table shows the comparison of the 3 scenarios, using the "*set tag*" function from the tables above for 7 sampling values:

MODEL	%HIGH	%MID	Median Error	Percentage of queries with error > 0.001
Standard	0	0.49%	2.27072E-05	9.60%
Non-sample MID&LOW priority	0	35.90%	2.10377E-05	9.40%
Non-sample&15% MID&LOW priority	0	60.60%	1.77697E-05	8.80%

Figure 48.Set tag functions results comparison table

Logically, the sampling error (Media Error) is reduced at the cost of several queries with response times between 5 and 15 seconds. We expected the percentage of queries with an error above 0.001 to be reduced more significantly.

3.3.5. Final Scenarios

To wrap up all the knowledge from generated the work done we are going to propose different scenarios that could be used for live data sampling. The aim of this section is that this report serves as tool the product owners and managers to decide the final strategy to implement. We firstly want to show the boxplots of the sampling errors for each sampling value and for each of the strategies proposed:

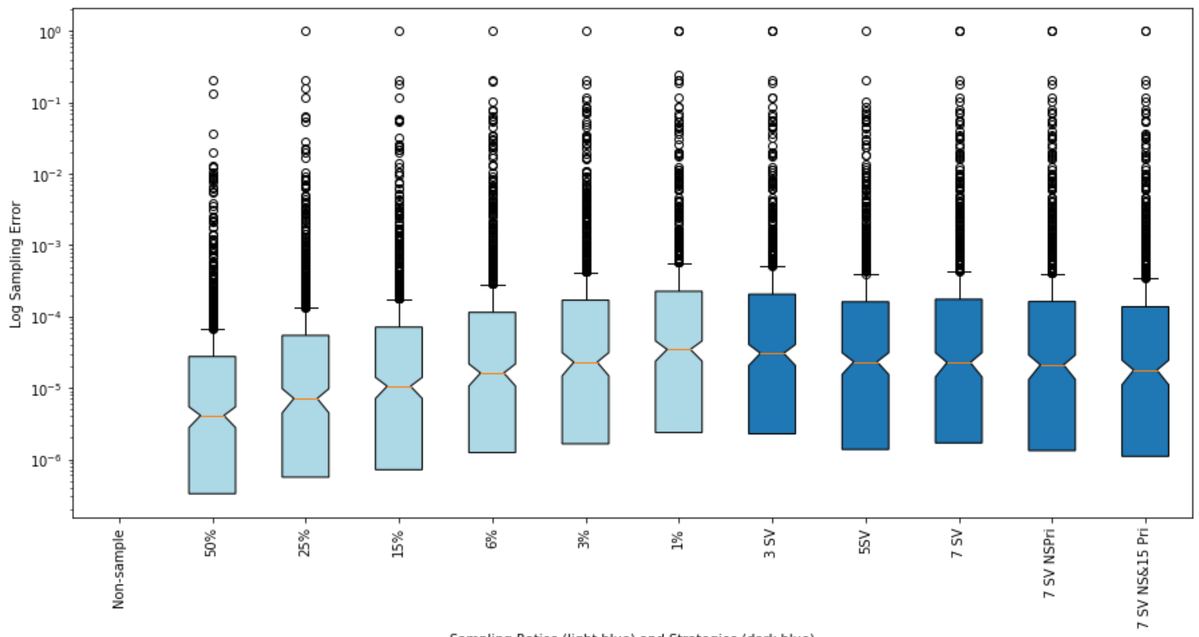


Figure 49. Box plot error model comparison

First of all, we observe the increase of the sampling error proportional to the increase of the sampling ratio. Then we see that, that the strategy also makes the sampling error decrease however the decrease is not very substantial. Finally, the comparison table shows:

Scenario	Definition	HIGH	MID	MedSaErr	%e>0.001	100%REC	SUPP
1	No sampling	**94.50%**	**3.80%**	0	0	1	2978
2	Sample to 15%	**17%**	**44.00%**	1.02001E-05	6.90%	1	2978
3	Sample to 1.5%	**0.00%**	**0.05%**	3.31088E-05	12.00%	1	2978
4	3 sampling values	**0**	**0.58%**	3.05975E-05	11.30%	0.95	1027
5	5 sampling values	**0**	**0.48%**	2.32764E-05	9.50%	0.75	1027
6	7 sampling values	**0**	**0.49%**	2.27072E-05	9.60%	0.51	1027
7	7 sampling values (set tag 1)	**0**	**35.90%**	2.10377E-05	9.40%	0.51	1027
8	7 sampling values (set tag 2)	**0**	**60.60%**	1.77697E-05	8.80%	0.51	1027

Figure 50. All models comparison table

As a reminder the key metrics for each scenario are:

- Percentage of queries between 5 and 15 seconds (MID)
- Percentage of queries above 15 seconds (HIGH)
- Median Sampling error (MedSaErr)
- % of queries above 0.01 error (%e>0.001)
- Model Recall of the non-sampled class – model metric (100%REC)
- Support: number of different queries comprised (SUPP)

We want to finish with some comments on the final scenarios. At first glance, it seems that the strategies based on prediction do not outperform very much a simple strategy, scenario 3 for instance, and it is true, however, a strategy based on prediction has one key advantage. With the growth of the number indices targeted the response times will also grow. Thus, the sampling error will grow too. So, at some point it will be necessary to diminish this error. And this exactly what we have done in this project. We have shown that **it is possible to reduce the sampling error by choosing the sampling ratio at query time**, which is crucial for a scalable solution.

4. Further work

4.1. Deployment

We also propose that the implementation of this project in production is done in 3 steps:

- The main bias of this project is probably on the query generation. It is very difficult to foresee what the users' queries will be. So, the goal is to have some functional as soon as soon as possible. And store the user queries. This must be done **applying a low constant sampling ratio** so that users do not have to wait too long for the query. Of course, users should notice about the quality of their data and have the possibility of not sampling and having large response time.
  A sampling ratio of 1.5% would give enough data to compare results.

- Once the dataset is built and we know the kind of queries users are making we could **introduce a heuristic function** to decide whether sampling is necessary for each query or it is not. For instance, sample only if the number of indices is more than 30. As the maximum response time we could get is around 20 seconds.

- If user queries show similar results to our generated queries a **prediction strategy** can be implemented to keep the sampling error low. We consider that the training set of the prediction strategy can use the artificial data used in this project, however the validation and test set must use user generated data.

In the code of the project a prototype of how the prediction would work in production has been implemented.

4.2. Other ideas to consider

Extract Index performance

To improve the performance of the model and considering that the number of indices is one of the main predictors. It might be interesting to extract to information from Kibana about the performance of the index, such as if this index as performed well recently or if its shards are being hit at the moment.

One index per Week

An alternative to a prediction-based strategy would be to re-index the data storage, by having indices including more than one day, for instance one week. If we do so the number of targeted shards would decrease for aggregations comprising 1 or 2 years. Consequently, the response time would also decrease. Note that it would still let us make aggregations for short date intervals. Obviously, this approach would have to be explored with more detail. One of the main drawbacks would be the high time cost of re-indexing, which we know is very in ElasticSearch.

Search by Departure

Still to do a similar research on the Search by departure. The research done on this project is only based on queries by user which are searching by search date. This means that for a specific query the indices comprised are only those in the search date range. But other visualization would need to allow to do a search by departure, in this case the indices comprised in the query would be determined by the departure date.

5. Conclusions

We now can withdraw some conclusions from the work done:

- Building the query generator could have been a never-ending task. But at some point, we must set a limit. It might be possible that the queries are not fully representative of users' queries, however there is no other way to prototype than making assumptions, besides as mentioned before they can be very useful for a training set.
- Creating the visualizations has been a key step to get to know the data. Moreover, as the predictive strategies do not fully reduce the sampling error, applying a more manual solution can bear on these visualizations.
- Relying on visual techniques for variable relationships can sometimes be misleading… For instance:
  - The variance of a query using t-tests shows that response time of a query are significantly however this difference was not appreciable through visualizations.
  - Sometimes it seemed that there was no relation between some variables, however the use of logarithmic transformation made the relationship happen.
- When implementing a machine learning model, it is necessary to explain why you have those results. Black box models do not work. It is not possible to put machine learning model in production if you do not fully understand why your model is learning well in some cases and why not in others. Interpretability is key.
- Putting in production a model fully based on artificially created data is dangerous. This has been the main limitation of this project.

Besides we can also enumerate the main takeaways for the student:

- Individual work requires discipline and methodology. The student considers this research project a great experience for developing rigour, reflection and critical thinking skills.
- The student considers that the freedom of work given by his supervisors has, on the one the side, allow him to make mistakes and learn from them. And other hand to be able to learn about what the student wanted and thought it was interested to do. Still, the guidelines given by the supervisor and colleagues have maintained under control the naivety and inexperience of the student.
- Doing an auto-criticism exercise the student consider he has fall in two main mistakes during the project

- Showing results too soon in an intent to show something that can prove there has been work done. Such as providing visualizations not meaningful or results with no double check.
- Relying too much on individual work. Although, it is one of the requirements of research, explaining the concerns or simply how the project is progressing more frequently could have been beneficial for the project.

- Finally, the student considers that his learning expectations have been met and considers this experience has one of the greatest (if not the greatest) experiences in his professional career.

6. References

[1]: Amadeus. 2016. Internal documents. Sampling_in_PB.pptx

[2]: Amadeus. 2016. Internal documents. Sampling.pptx

[3]: Amadeus. 2016. Internal documents. Search_AnalysisPOC.pptx

[4]: Amadeus. 2016. Internal documents. FareSearchCurriculum.pptx

[5]: Amadeus. 2016. Internal documents. AmadeusTI_for_TA.pptx

[6]: Amadeus. 2016. Internal documents. SA_new_architecture.pptx

[7]: Nguyen T., Song I. 2016. Centrality Clustering-Based Sampling for Big Data Visualization.

[8]: ElasticSearch GitHub. [elasticsearch-dsl-py/elasticsearch_dsl at master elastic/elasticsearch-dsl-py · GitHub]

[9]: ElasticSearch DSL Documentation. [Elasticsearch DSL — Elasticsearch DSL 5.4.0 documentation]

[10]: https://www.sv-europe.com/crisp-dm-methodology/

[11]: Google analytics documentation: https://support.google.com/analytics/answer/2637192?hl=en

[12]: Predicting Response Time. Thomas Lawson.

https://rndwww.nce.amadeus.net/confluence/display/CNV/Comparison+with+random+classifying

[13]: O. Günlük, J. Kalagnanam, M. Menickelly, K. Scheinberg. Optimal Decision Trees for Categorical Data via Integer Programming.

[14]: J. Ali, R.Khan, N. Ahmad, I. Maqsood. September 2012. Random Forests and Decision Trees. International Journal of Computer Science Issues, Vol. 9, Issue 5, No 3.

[15]: Why it is important to work with a balanced classification dataset. http://amsantac.co/blog/en/2016/09/20/balanced-image-classification-r.html

[16]: X. Zhang, C. Feng, G. Wang. Prediction of Website Response Time based on Support Vector Machine. The 2014 7th International Congress on Image and Signal Processing https://ieeexplore.ieee.org/document/7003908