FEDERAL UNIVERSITY OF CEARÁ

TECHNOLOGY CENTER

ELECTRICAL ENGINEERING DEPARTMENT

CURSO DE GRADUAÇÃO EM ELECTRICAL ENGINEERING

NICOLAS DE FREITAS SOARES

OPTIMIZATION OF A FLIGHT RECOMMENDATION SYSTEM

FORTALEZA

2022

NICOLAS DE FREITAS SOARES

OPTIMIZATION OF A FLIGHT RECOMMENDATION SYSTEM

Undergraduate Final Paper presented at Electrical Engineering Graduation Course of the Technology Center of Federal University of Ceará, as a partial requirement for obtaining the degree of Bachelor's in Electrical Engineering.

Supervisor: Prof. Dr. Fabrício Nogueira Gonzalez.
Co-supervisor: Prof. Dr. José Antonio Macedo.

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S655o Soares, Nicolas de Freitas.
Optimization of a flight recommendation system / Nicolas de Freitas Soares. – 2021.
51 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2021.
Orientação: Prof. Dr. Fabrício Gonzalez Nogueira.
Coorientação: Prof. Dr. José Antônio Macedo.

1. Bookings . 2. Recommendations. 3. Subset. 4. Value . 5. Search. I. Título.

CDD 621.3

NICOLAS DE FREITAS SOARES

OPTIMIZATION OF A FLIGHT RECOMMENDATION SYSTEM

Undergraduate Final Paper presented at Electrical Engineering Graduation Course of the Technology Center of Federal University of Ceará, as a partial requirement for obtaining the degree of Bachelor's in Electrical Engineering.

Approved in: December 20th, 2022.

EXAMINATION COMIITTEE

Prof. Dr. Fabrício Gonzalez Nogueira (Supervisor)
Universidade Federal do Ceará (UFC)

Prof. Dr. José Antônio Macedo (Co-supervisor)
Universidade Federal do Ceará (UFC)

Prof. Dr. Luiz Henrique da Silva Barreto
Universidade Federal do Ceará (UFC)

Msc. Gabriel Freitas Machado
Universidade Federal do Ceará (UFC)

Aos meus pais, Jorge e Camilla, minha irmã Beatriz, e aos meus verdadeiros amigos.

AKNOWLEDGEMENTS

To my tutor and boss Guillaume Le Grand, who supported me in the process of this research during my internship at Amadeus I.T. Group in France, together with the SHM team, especially the manager Romain Meynard and the coworker Antonio Paladini. They have guided and instructed me throughout my experience in the company. I extend my thanks to all SSP and Amadeus employees that joined me along my journey.

I would like to thank Dr. Fabricio Gonzalez Nogueira at Universidade Federal do Ceará, who guided me on the aspects involved on finishing this work and to Dr. Luiz Henrique da Silva Barreto, general coordinator of my double degree program. Their invaluable support is greatly appreciated.

A special thanks to Dr. Anthony Kolar and Dr. Raul de Lacerda at CentraleSupélec – CS, in France. Both accompanied me throughout my entire trajectory in the exchange program. I am also thankful to Dr. Frédéric Boulanger, responsible of the Software Engineering path of CS.

I should also acknowledge Dr. José Macedo and all Insight Lab crew under his leadership at UFC. As my undergraduate supervisor during my first years at UFC, Dr. Macedo and his group have inspired and encouraged me to pursue new horizons in the data science field.

Finally, a very warm recognition to my family, who supports me in every step of my personal and professional career, always providing me support and encouragement, on defeats and victories. A special thanks to my father, Jorge Barbosa Soares, my role model, and my best friend. He masterfully permeates various spheres of my life, always with the best of intentions. I owe to him much of the man I am today.

"As invenções são, sobretudo, o resultado de um trabalho teimoso."

(Santos Dumont)

RESUMO

Quando um Site de Viagens solicita recomendações de um trajeto entre dois pontos em uma determinada data, há potencialmente milhares de itinerários possíveis. Apenas retornar os 100 voos mais baratos não é a resposta mais relevante, afinal as pessoas fazem suas escolhas considerando também outros aspectos da sua conveniência (Ex.: Quanto tempo? Quantas conexões? Qual é a hora de partida?). O problema aqui investigado não é selecionar o melhor voo, mas um subconjunto de voos que tenha uma elevada probabilidade de conter o voo selecionado. Motores de busca e sistemas de recomendação já são implementados no mercado, como Netflix, Google, Amazon etc. Por conseguinte, o desafio é aplicar esta abordagem à indústria de viagens. O grupo de engenharia de produtos Search, Shopping & Pricing (SSP) trabalha como principal órgão para o sistema de Distribuição da Amadeus, bem como o negócio de e-commerce de TI das companhias aéreas e os novos negócios, desde inteligência de viagens a TI em ferrovias. É responsável pelos produtos de Pesquisa e Compras de Viagens, que alimentam atualmente um grande número de websites de companhias aéreas e agências de viagens. Com base em um vasto histórico de reservas de voo, o objetivo do presente trabalho é analisar criticamente um dos principais produtos da SSP visando melhorar os seus algoritmos e otimizar o motor de busca da Amadeus, a fim de melhor retornar recomendações de viagem. A metodologia utilizada não é apenas para classificar as recomendações, mas também para encontrar o subconjunto de recomendações mais susceptível de contribuir para uma reserva. O trabalho introduz o conceito de um motor de busca (*value search*) e mostra como este sistema se baseia na probabilidade de reservas. Depois, ajustam-se alguns dos seus parâmetros, executam-se diferentes combinações, compara-se os seus resultados e escolhe-se o mais eficaz. Ao final, são obtidos resultados consideráveis que mostram o crescimento da performance do algoritmo, tornando os conceitos de seleção de subconjuntos simples e compreensíveis de implementar em um sistema de recomendação.

Palavras-chave: reservas, recomendações, motor de busca, subconjunto*, value search.*

ABSTRACT

When a Travel Website requests for travel recommendations from one point to another on a given date, there are potentially thousands of possible itineraries. Simply returning the 100 most unexpensive flights is not the most relevant answer. People make their choices considering additional aspects based on their convenience (e.g., How long? How many connections? What is the departure time?). The problem investigated herein is not to select the best flight, but an optimal subset that has a high probability of containing that flight. Search engines and recommendation systems are already implemented in the market, e.g., Netflix, Google, Amazon, etc. Therefore, a challenge is to apply this approach to the travel industry. The Search, Shopping & Pricing (SSP) product engineering group serves the core Distribution business of Amadeus as well as Airline IT e-commerce business and the Rail IT and Travel Intelligence new businesses. It is responsible for the travel Search and Shopping products, which power today a vast number of airline and travel agency websites. Based on past bookings, the main goal of the work presented in this final paper is to critically analyze one of SSP's core products with the aim of enhancing their algorithms and optimizing Amadeus' search engine to better return travel recommendations. The used methodology is not just to rank the recommendations, but to find the subset of recommendations that is the most likely to lead to a booking. It is introduced the concept of a value search engine and how this system is based on reservation probability. Then, it is adjusted some of its computing parameters, run different combinations, compared their results, and selected the most effective. At the end, the aim is to obtain considerable results that show the growth of the algorithm's performance, making the concepts of subset selection simple and understandable to implement in a recommendation system.

Keywords: bookings, recommendations, search engine, subset, value search.

LIST OF FIGURES

LIST OF TABLES

SUMMARY

1 INTRODUCTION

1.1 Initial Considerations

Created in October 1987 by an agreement from Air France, Lufthansa, Iberia and SAS, Amadeus' main purpose was to generate a neutral global distributed system to connect provider's content with travel agencies and customers in real time. The purpose was to bring together the entire air ticket sales process, regardless of airline companies.

Its field of activity originally revolved around products focused for airlines. Amadeus currently offers multiple services related to tourism, such as the hotel industry, the car rental market, and the railway market. Today, the company is a major IT provider for the global travel and tourism industry, internationally known for developing and optimizing the best IT solutions to allow clients access optimal commercial results. The company covered 3.2% of the world's travel and tourism GDP in 2018 and operates in 190 different countries worldwide, processing over 55,000 transactions per second at peak (WORLD TRAVEL & TOURISM COUNCIL, AMADEUS, 2019).

A great obstacle for companies such as Amadeus is to predict what travel recommendation(s) is(are) likely to be booked. There are many possible recommendations, and selecting the best one is a challenge for two main reasons: (i) it is hard to understand what the end users are likely to book, and (ii) there is a need to select a subset that is not common to machine learning – ML. For example, it is known that the most unexpensive recommendation is not always the one selected. There are other variables besides cost that impact consumers' choice, and they need to be considered.

1.2 The Problem

When a Travel Website requests for travels from one point to another on a given date, there are potentially thousands of possible itineraries. It is a challenge and a major task to select the most interesting travel recommendations among all possible combinations. A good travel recommendation would have a higher chance to lead to a booking.

Even if the recommended flight is the cheapest or the fastest, it does not guarantee that it is the most suitable for a specific request. Users have different needs for the same request, and moreover, the same user can change his or her mind depending on travel date and destination. Finally, the very recommendations offered at a moment to a user can influence

his/her decision-making process. This phenomenon is studied in what is called game theory (OWEN, 2013).

Problems like the ones mentioned above show why choosing a group of flights to recommend to a given user can be so challenging even for a Machine Learning algorithm. The problem of product recommendation systems has been extensively addressed in the industry, i.e., Netflix for movies, Amazon for products, or YouTube for videos. But, when it comes to the travel industry, we are faced with a much larger and more dynamic sample space than the referred well known products. Business Bureau reported that Netflix offers 2,926 movie titles and 950 series (and more than 28,000 episodes) in Brazil (COSSETTI, 2018). Whereas Amadeus operates worldwide and processes over 55,000 transactions per second at peak. This makes the approach used on Netflix impractical for our case (WORLD TRAVEL & TOURISM COUNCIL, AMADEUS, 2019).

1.3 General Concepts

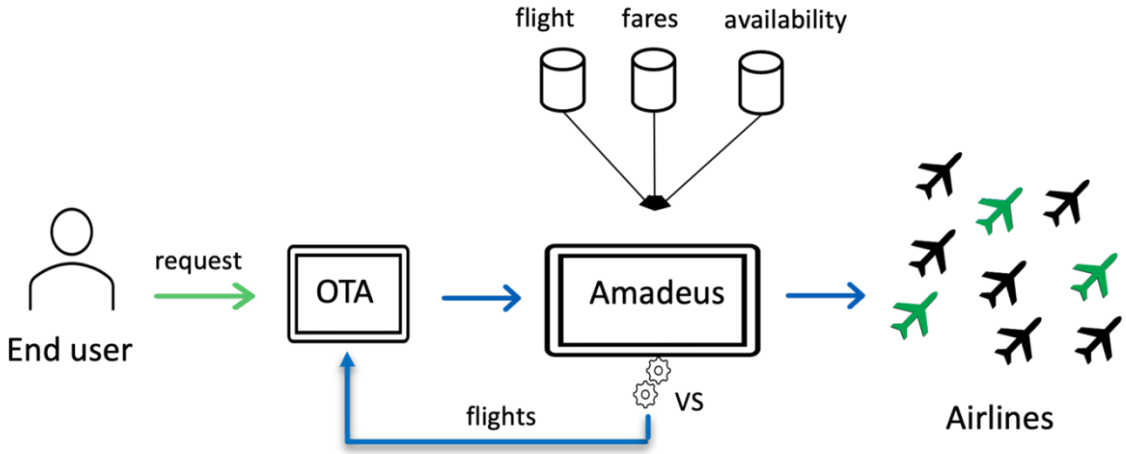
An engine was developed at Amadeus core to gather travel recommendations. Checking important criteria such as availability, price, and fares, it computes many possible flights combinations and returns the top recommendations subset for a single travel request. These computed results are given to Online Travel Agencies – OTAs. The referred engine is one of the main products of SSP's flight recommendation system. The algorithm used to optimize this engine is called Value Search (VS). This algorithm computes a single value for every recommendation to sort them by chances of leading to a booking; the smaller the value, the greater the chance.

When looking for a flight, there are some specific features implemented before starting the search at all, such as origin and destination pair – ONDs, travel date, number of passengers, etc. The assemble of these features compose a single flight demand, called query. A query can be made by a single passenger, an airline website, or an online travel agency – OTA. The latter can search in many airlines at the same time with tools called metasearch engines. Once the query is made, there are many possible itineraries, a combination of different flights that satisfy query's parameters. The booking is made by choosing the most suitable recommendation among all the given ones.

Once a flight ticket is bought, there are a big amount of information that comes with it. This information is called the trip information, or tripinfo, and it contains numbers such as departure time, number of connections, elapsed flight time (eft), i.e., the amount of

time of the entire trip with the connections, and of course, the price. Figure 1 shows a representative scheme of how a set of recommended flights is returned to an end user.

Figure 1 – Scheme of a search for travel recommendations



Source: Author (2022).

There are many possible combinations of flight recommendations. Selecting the best options is a challenge for two main reasons:

i. We need to understand what end users are likely to book, with the most unexpensive flight often not being the one selected. Most of the time, it is necessary to present a variety of recommendations, including the cheapest flight among them, even with this not being the best indicator of a good recommendation. That is why other flight criteria need to be analyzed.
ii. The selection of this subset is not intuitive to machine learning algorithms. We do not know if it is an efficient way to solve the problem. The work herein will not cover this approach. Instead, we have developed a study that gathered personal criteria and used an optimizer to create a client-driven product.

The focus is on a family of algorithms that assign a value to each recommendation based on its information, i.e., parameters, and sort them according to this value. Our challenge is to select the right parameters to correctly sort the recommendations, so the selected subset has a high chance of containing the booking.

1.4 Objective

The general purpose of this work is to analyze the flight recommendation system through the Value Search engine and to optimize the selections of subsets containing the booked recommendation. Note that what is necessary is not correctly select a value for each recommendation or sort them differently, but to select a subset of recommendations that has the higher chance of containing the booking.

Specific objectives of the work to achieve the general objective are as follows:

- To select a combination of optimal weights related to fixed recommendation criteria.
- To adjust the hyperparameters in the hyperopt optimizer, such as search space and number of iterations, necessary for a consistent statement.
- To create an intuitive process to read and write data in a distributed system using the Hadoop cluster on HDFS and implement the algorithm in python with the PySpark framework.
- To produce, with a certain margin of safety, a new algorithm that performs better than the algorithm used by the company to choose a subset of recommendations with the highest probability of containing booking.

1.5 Workflow

The work of this undergraduate final paper contains several steps, further detailed ahead. It can be generally divided in 3 main phases: (i) algorithm development, (ii) distributed data processing, and (iii) statistical analysis. Each of these phases is directly dependent on the result of the previous one. Table 1 presents the work organization in time, showing the phases that overlap each other, and those that only start when the previous has ended.

Table 1 – Workflow organization in 2022

May	June	July	August	September	October
Data Mining					
Algorithm Development					
Data Analysis					
	Criteria analysis				
		Distributed Data			
			Segmentation		
				Statistical Analysis	

Source: Author (2022).

The paper is divided in three main sections: (i) theoretical background and a few data science concepts, and machine learning practices; (ii) methodology, containing the used tools and techniques; (iii) results, which represent the most important part of this document. Noting that this work is a constant back and forth between observations and tunning of the parameters. In other words, this is how the optimizer developed seeks to improve the algorithm. Some statistical resources are used to assure the veracity of the observations. At the end, I intend to quantify, within a confidence interval, the improvement of the algorithm's performance.

2 THEORICAL BACKROUND

In data science, there are many methods used to analyze the behavior of a large dataset. In our case, a typical data set consists of hundreds of million travel searches that turn into billions of travel recommendations (i.e., Terabytes) every day.

But there are many questions regarding these data. Do the flight prices follow a normal distribution? Is the data time related? If it needs to be predicted, should we use machine learning methods as linear regression or more complex neural networks like LSTM? What are the fairest methods to perform predictions on client's intent to book? Which type of parameters are we trying to optimize? What is the reliability of the gathered results? To answer these questions, we need to understand how algorithms are used on recommendations systems.

2.1 Recommendation systems

Modern companies use recommendation systems that try to best fit the customer profiles. Feedback techniques are reported in the literature to implement a costumer driven approach based on a large dataset by filtering personalized information (DEBASHIS; SAHOO; DATTA, 2017). In Amadeus flight recommendation system such feedbacks are not that useful due the volatility and quantity of the data. Instead, we analyze the historical data of booked flights and train our selection algorithm to maximize the probability to find a good subset (AMDEUS, 2022).

Unlike the recommendation systems handled by product companies such as Netflix or Amazon, based on the experience of other users who have used the same product as the customer, value search algorithm will be based on the customer's information, trip information and available recommendations. Then, pre-determined weights are applied to calculate a specific value per recommendation, called VS value, then rank the recommendations by this value, and select the top subset (generally 50 recommendations). This means that no machine learning will be applied in the subset selection process.

The obstacle becomes finding a good subset among all recommendations. There are many subset selection methods but choosing the right one depends on the nature of the problem. Some works found in the literature address contents such as clusterization (DASZYKOWSKI; WALCZAK1; MASSART, 2002) and genetic algorithms (TOMINAGA, 1998) that help algorithms to group elements that have similar criteria. But at this part of the

paper, the focus is on a good representation of the historical data, i.e., weighting the most relevant criteria.

2.2 Relevant feature selection

The Value Search engine ranks recommendations based on the VS value. This value is calculated by a weighted average of the recommendation criteria. Then, the top 50 recommendations are selected. This subset has the most likelihood of containing a booking. The lower the value, the more chances a recommendation has of becoming a booking.

Other phenomena that are important to be aware is the bias of the dataset. If the data is too related, it can cause some bias on the training phase of the algorithm, which can cause overfitting. One possible solution found in the literature is filtering the data, reducing to only the minimum features sufficient to determine the good prediction (JOHN; KOHAVI; PFLEGER, 1994).

Flight criteria are the information considered by the customer when booking a flight, such as price, elapsed flying time (eft), number of connections, etc. For example, if the eft criteria have a value of $50€/h$ ($C_{eft} = 50$), it means that the traveler is willing to pay 50€ more on a flight ticket to avoid spending one more hour in the trip. But the criteria by itself do not accurately reflect the will of the user for booking. That is why it is applied a certain weight to each of the criteria to obtain VS value per recommendation. This value changes accordingly to the chosen weights. The formula that explains how the values are calculated is

$$VS_{value} = P + \sum C_i w_i \text{ , where:} \quad (1)$$

- VS: value search value (per single recommendation)
- P: price of the flight
- C: criteria
- w: weight
- i: index from zero to the number of criteria

One should note that no weight was added to the price criteria. That is due the fact that we want to state the price as the unit, and all the other criteria will depend on varieties of the price itself. After defining the formula, it is time to find an optimal solution. The goal is to

compute a value to rank the booked recommendation as good as possible in the recommendations' set. Table 2 illustrates how the recommendation criteria are distributed on the dataset and their respective types.

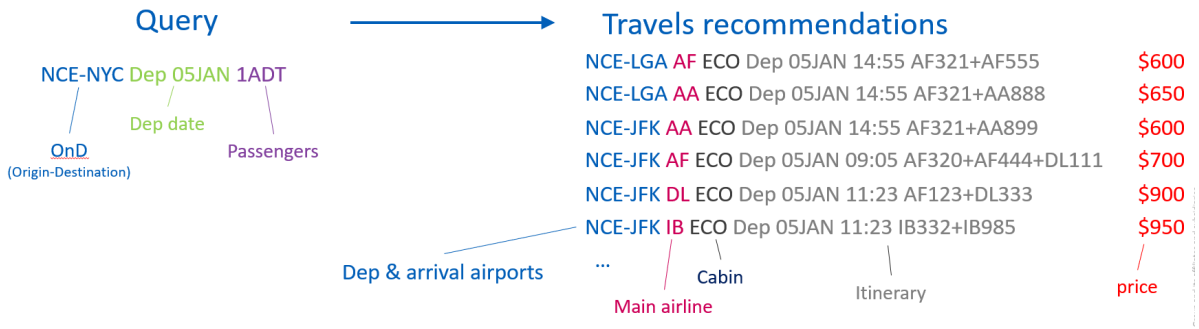
Table 2 – Recommendation criteria representation.

Recommendation attributes	**Type**
Elapsed flying time	Float
Change of airport	Integer
Connections	Integer
Ground time	Float
Has late arrivals	Boolean
Has early departs	Boolean

Source: Author (2022).

Imagine that each of these criteria matter when it comes to the decision of purchasing a flight. A specific weight, initially random, is assigned to each of these criteria. Then the VS value is calculated as a weighted average for a single recommendation, which means that for Table 2 it is calculated one single VS value. Figure 2 illustrates how travel recommendations look at the beginning and the corresponding information obtained after the analysis.

Figure 2 – Flight recommendation representation.



Source: Amadeus.

The Value Search formula was later changed for performance analysis, but for confidentiality reasons it is not published in this paper. The goal, however, was to keep the mathematical calculation linear. Linear programming is one of the simplest ways to perform

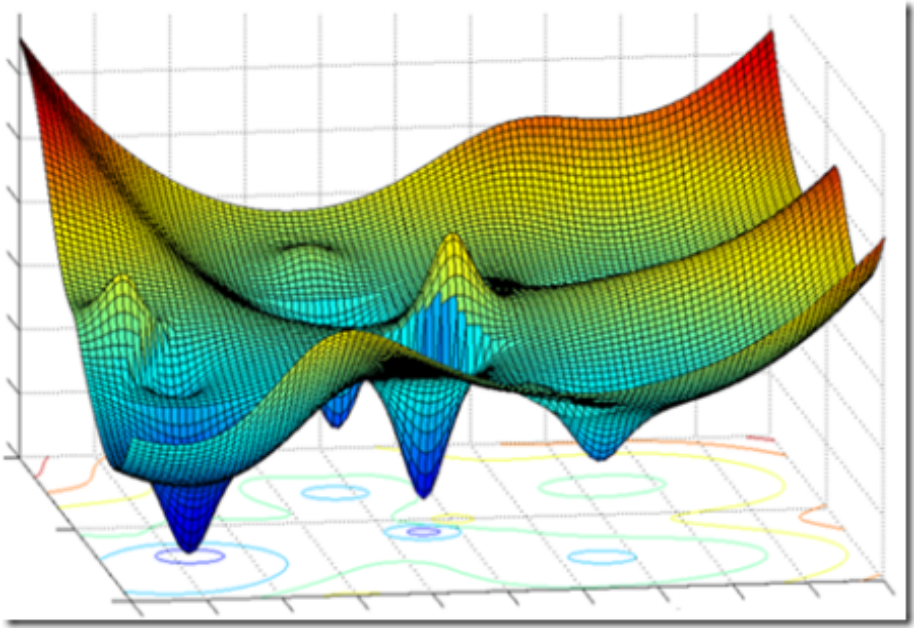
optimization. It helps solving some very complex linear programming problems and linear optimization problems by making a few simplifying assumptions (AVCONTENTTEAM, 2017).

2.3 Optimization using Hyperopt

We train the algorithm updating its weights in every iteration. Since weights and criteria are the parameters of VS, the adjustments of how the algorithm works are the so called hyperparameters (ex.: number of iterations, search space, etc.). In machine learning, hyperparameters are defined as the values that are used to control the learning process. And the tuning technique that attempts to compute the optimum values of hyperparameters is called grid-search (MALATO, 2021).

This used optimization technique consists in a repetitive attempt at different values of weights searched on a discrete space (called a grid) with the intention of minimizing a loss function. For this purpose, I have made use of a python package called hyperopt, an optimizer that uses Tree-based Parzen Esimtors - TPE to optimize a user-defined loss function (SINGH et al., 2019).

Figure 3 – 3D grid search representation



Source: Ippolito (2019).

It is interesting to have a certain diversity on our recommendation set, for simplicity purposes, called RecoSet. For example, having only flights with similar price, eft,

and connections, would not be a diverse RecoSet. The first option would not be that different from the 10th one. This behavior could lower the chances of a booking. Determining how good a RecoSet is or evaluating algorithms' performance is not a trivial task. The intention of this section is to contribute to a better understanding of how recommendations are selected and to suggest different score calculation formulas that return a better RecoSet.

Assuming a good understanding of the nature of the data and implementing the grid search method for tunning VS algorithm's hyperparameters, the problem is reduced to choosing the hyperparameters to be analyzed (JOHN; KOHAVI; PFLEGER, 1994). And for that, it is necessary to understand the performance of the algorithm, i.e., evaluate somehow its training. That is why I introduce on the next topic the validation methods.

2.4 Validation methods

The validation phase is extremely important in machine learning algorithms. Using proper validation techniques is useful to understand the model, but most importantly, estimate an unbiased generalization. The bias is a result caused by overfitting; a representation too close for a specific data set that causes problems to ML algorithms when trying to predict a model behavior. It can exist because the criteria used to choose the model was not the same one to judge the suitability of this model.

One of the validation methods used on the work herein is the random train/test split data. This allows the algorithm to shuffle the dataset before training again. With the correct hyperparameters, the algorithm is evaluated on the capacity of selecting a RecoSet containing the booked recommendation. After a few weeks, we were able to develop different formulas that could suggest a better performance of the "probability to book" a certain recommendation within a RecoSet, named as Booking Coverage Performance (BCP).

$$BCP = \frac{nb\ RecoSets\ containing\ booking}{total\ nb\ of\ RecoSets}. \quad (2)$$

The above created parameter (BCP) represents the percentage of right guessed RecoSets, and its results can vary according to some variables like the size of the selected subset, aka setsize, as shown on Table 3.

Table 3 – BCP results for a small dataset

Booking Coverage for Amadeus					
Algorithms		Setsize = 20			
		Train = 50%	Train = 60%	Train = 70%	Train = 80%
	1	72.02	71.3	70.79	73.86
	2	80.90	81.16	77.89	80.55
		Setsize = 70			
		Train = 50%	Train = 60%	Train = 70%	Train = 80%
	1	83.21	79.79	85.8	82.96
	2	86.01	85.71	87.83	86.32
		Setsize = 70			
		Train = 50%	Train = 60%	Train = 70%	Train = 80%
	1	86.13	84.95	85.19	86.32
	2	88.56	87.83	88.03	86.32

Source: Author (2022).

A problem found on these results is the reliability of the BCP. Every time the BCP of a same training set was computed, the results were slightly different, which means that there was a certain noise on the observations. The next phase of this research consists in distributed programming. By performing big data analysis and obtaining multiple results of the same algorithm, we were able to use statistical assumptions.

2.5 Statistical analysis

The distributions of BCP could be close to a Gaussian curve, which is intuitive to presume. Suppose our algorithm has a BCP = 0.5, i.e., guesses correctly 50% of the RecoSets in each training dataset. If we compute the same algorithm again, with the same parameters and same data, it is very unlikely that we obtain a BCP of 0.4 or 0.6. In other words, our standard deviation – std should not be bigger than 10%. Analysis in statistical variables such as cumulative mean, std, and variance will be key aspects of our analysis to determine how accurate our algorithm is performing and what is the reliability of the obtained results.

The major goal of this part will be to obtain the BCP mean of multiple observations of the same algorithm and training/testing dataset of two or more different formulas. Then, certify that the difference between these means is higher than the sum of std of both (BAKER; HARDYCK; PETRINOVICH, 1966). Only then we will be able to judge that one formula is performing better than another.

3 METHODS AND TOOLS

3.1 Nature of the Data

The dataset considered in the paper was built as following: for every booked recommendation, aka booking, we retrieved the 500 best recommendations (for convention designated as “recos”). Together we linked the booked recommendation to its related query and tripinfo. The raw dataset is composed by one booking per row. For every booking there are its related 500 recos. Together they compose our flight data. An illustration of the data is presented in Table 4.

Table 4 – Flight data representation

Column	Description
Tripinfo	A compact description of the discrepancy. Contains information about the client, booking date, origin/destination, stay of the trip
Query	Search date, quantity of passengers, origin/destination, etc.
Booking	Only the booked recommendation
Recos	500 Round Trip recommendations

Source: Author (2022).

The original data set provided by Amadeus makes more than 2Tb of information. As a first approach, it was analyzed a (40Gb) file from April 2022, and only considering round trips. Further, we will explain how the algorithm can get these 500 recos and select a 50 recos subset that best fits client’s needs. This subset of 50 recommendations is called RecoSet.

At the biggening, the analysis was made in a small data sample so we could train the algorithm to choose the best flight criteria without losing much time or memory process. Once the algorithm was created, it was needed to scale it to obtain more reliable results. Which means we started working with Big Data, using Spark framework and the Hadoop Distributed File System to compute distributed data analysis.

3.2 Apache Spark

Apache Spark is an open-source unified analytics engine generally used for big data processing. It uses multiple clusters for parallelism it scales the computation capacity of a normal GPU. Fortunately, python allows an interface possible to write Spark Applications using python APIs and provides a PySpark shell for interactive data analysis in a distributed environment. In other words, moving our work to Spark language allows us to scale analytics to large databases and run multiple processes in parallel (SHARMA, 2022).

Figure 4 – Spark framework to python logo

Source: https://aprendizadodemaquina.com/artigos/pyspark-entenda-a-engine-do-spark-para-rodar-python/

Apache Spark is a fast-computing system that has high integrity but also allows high-level tools like Spark SQL for SQL and MLib for machine learning (ROCCO, 2021). This engine allows processing large data sets with a distributed algorithm. Spark, unlike MapReduce, stores data in memory and uses different data storage models where data is fetched and joined from multiple sources. On the other hand, Hadoop uses replication to achieve default tolerance (SHARMA, 2022).

Table 5 – RDD is visualized on PySpark interface

flights	query	bookings	recos
"{""discrepancy_i...	UNB+IATB:1+FSITE+...	UNB+IATB:1+1ASIFQ...	UNB+IATB:1+1ASIFQ...
"{""discrepancy_i...	UNB+IATB:1+FSITE+...	UNB+IATB:1+1ASIFQ...	UNB+IATB:1+1ASIFQ...
"{""discrepancy_i...	UNB+IATB:1+FSITE+...	UNB+IATB:1+1ASIFQ...	UNB+IATB:1+1ASIFQ...
"{""discrepancy_i...	UNB+IATB:1+FSITE+...	UNB+IATB:1+1ASIFQ...	UNB+IATB:1+1ASIFQ...

only showing top 4 rows

Source: Author (2022).

Using the pre-processed data done on Jupyter Notebook with Spark framework, we compute the information in a way that is more convenient to store in cache. A first approach was on JSON format. JSON files are efficient for parallelization since they are open standard files that allow parsing and are reliable when dividing the information in different threads. For that we needed to adapt the code to PySpark. In this chapter, the second part of the transformation phase takes place. The final table is as Table 6 shown below.

Table 6 – Recommendation information on JSON format

search_date	booking_id	Json
2022-06-02	ee7de1d530654083a...	{"booking_id": "e...|
2022-06-02	2fb8814dbc0f457c9...	{"booking_id": "2...|
2022-06-02	ce57fa20a5e6403e8...	{"booking_id": "c...|
2022-06-02	52f2bd1fb88d420ab...	{"booking_id": "5...|
2022-06-02	556fb59519a34c578...	{"booking_id": "5...|
2022-06-02	495fa08fd1484055a...	{"booking_id": "4...|

only showing top 5 rows

Source: Author (2022).

3.3 Hadoop Distributed File System – HDFS

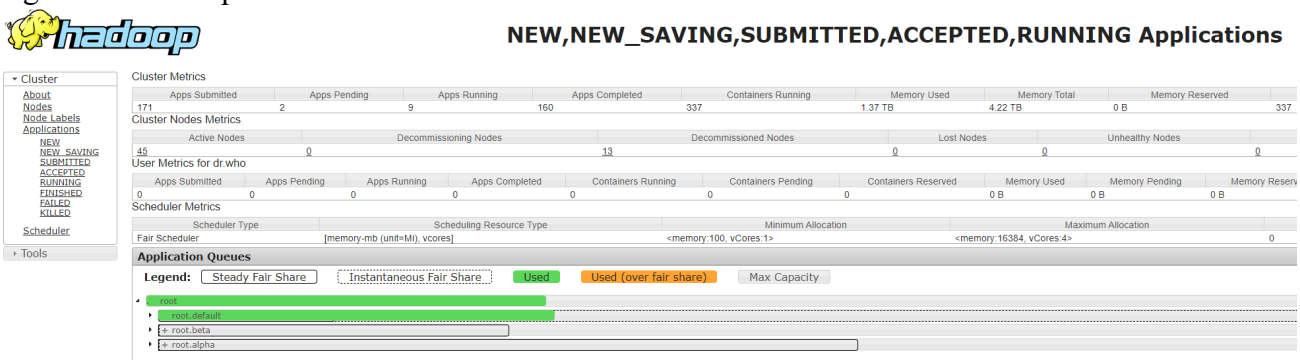
The Hadoop Distributed File System (HDFS) is a distributed system designed to run commodity hardware. Some of its advantages is that it is highly fault-tolerant, aka flexible, and is designed to be deployed on low-cost hardware. Due to its massive capacity and reliability, HDFS is a very suitable storage system for Big Data. In combination with YARN, it increases the data management capabilities of the Hadoop cluster and enables efficient Big Data processing (HADOOP APACHE, 2022).

Figure 5 – Hadoop system logo

Source: Hadoop Apache (2022).

There is now too much data to be stored centrally, especially due to cost and storage capacity constraints. Figure 6 shows an example of how jobs were monitored while running on HDFS user interface. More figures about tasks distributions and job information's can be found on the appendix of this paper.

Figure 6 – Hadoop user interface



Source: Hadoop Apache (2022).

It is also possible to check each job individually and to see the distribution of executors for every task. The interactive interface of Hadoop provides several other information, and examples are found on appendices of this final paper.

The problem is how to parallelize our loss function. It is important to say that the loss function code is the only parallel computation done at every hyperopt iteration. First, the JSON files were created to store all the information. But, at computation phase, the cluster memory was exploding its limits (15Gb/Job) when loading into cache. The solution was therefore to change one more time the preprocessing file structure to Parquet files. Parquet is optimized for the Write Once Read Many (WORM) paradigms, which perfectly suits our needs, since we preprocess only once, and compute multiple times.

3.4 Jupyter Notebook and Bash scripts

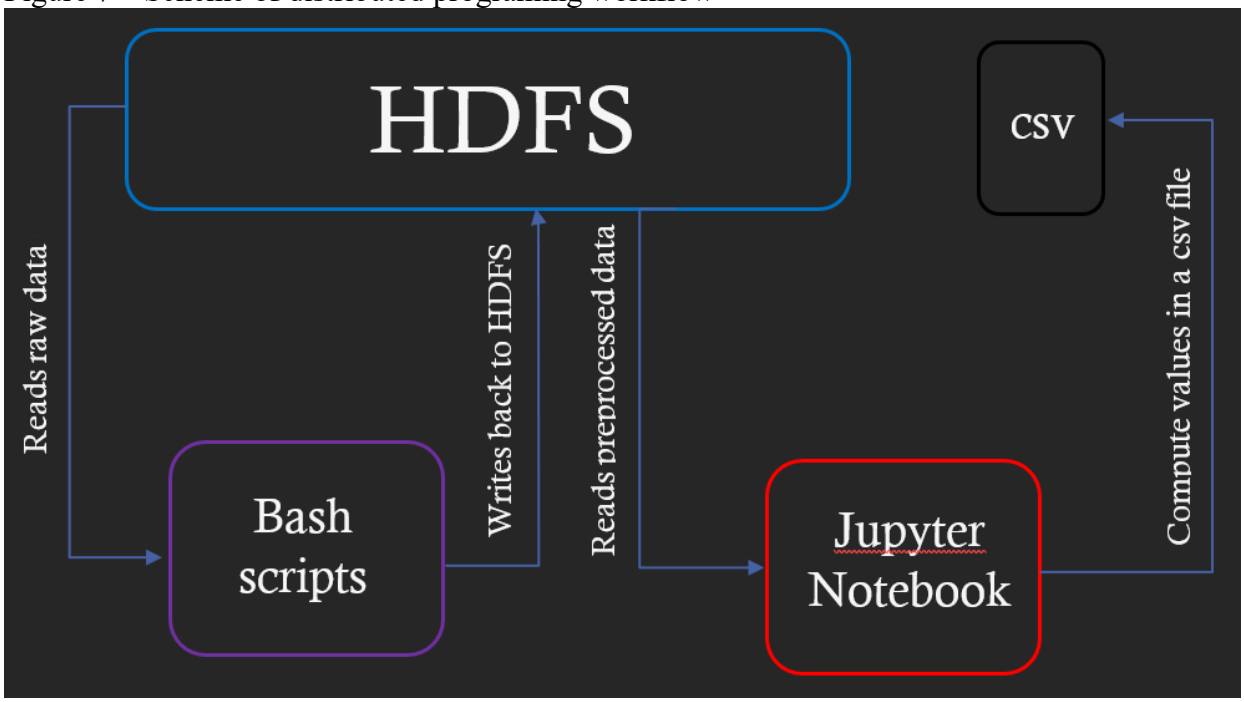
The Spark code was launched from a spark-submit command. The data frame is read from HDFS, preprocessed in parallel and written in parquet back in HDFS. The problem faced during this phase of the work was that there was no python's hyperopt library installed on the spark gateway. Since the code created a new Spark session, when calling hyperopt, spark errors were raised, and the job aborted.

Later, the code was tested on Jupyter Notebook, where we managed to install hyperopt packages. But since Jupyter Notebook uses a spark session already existing, the problem is that there was no control on the session's parameters. This means that we cannot set the allocated memory neither on driver nor on executors. Compacting the data back to parquet files and writing back to HDFS was exceeding spark session's memory limits.

These are problems attributed to developers, and they usually are not difficult to solve since they can be managed by infrastructure adjustments. But they are not related to the scope of this work.

The found solution was to divide the code into two main steps. Therefore, the preprocessing phase was done from spark submit scripts and the computation phase done in Jupyter Notebook with the Spark framework. Ii is also worth mentioning that the final parquet file was divided into smaller files and the writing process was repeated in a loop until there is no more data to be read. Figure 7 explains this parallelization-memory tradeoff.

Figure 7 – Scheme of distributed programing workflow



Source: Author (2022).

4 RESULTS

The Hyperopt algorithm was evaluated with different parameters for each run. The results were compared by BCP. The algorithm's performance of one single run translates by how many times the booking was located on its RecoSet. Table 7 exemplifies a scenario where the booking is inside 3 out of 5 selected RecoSets, this means that the algorithm performed with a BCP of 60%. We then define a loss function that behaves opposable to booking coverage. Maximizing BCP means minimizing our loss function.

Table 7 – Illustration of algorithms' RecoSet selection

Recommendations	Booking	Algorithm	RecoSets
500 recos	booking 1	Hyperopt	Set N ✓
500 recos	booking 2	Hyperopt	Set N ✗
500 recos	booking 3	Hyperopt	Set N ✓
500 recos	booking 4	Hyperopt	Set N ✓
500 recos	booking 5	Hyperopt	Set N ✗

Source: Author (2022).

$$booking_{coverage} = 60\%, \tag{3}$$

$$P_{argmax}\left(booking_{coverage}\right) = \min\left(loss_{function}\right). \tag{4}$$

Afterwards, we vary the size of the recommendation subset (*setsize*). We consider the recommendation sorting is a success if the booking is too be found in the first *setsize* recommendations. Of course, the larger the set size is, the better is the performance. Also, we modify the percentage of the total data that would be used to train the algorithm (*DF_Train*). What is interesting to see is that there is an optimal point of *DF_Train setsize*, because the bigger the training data is, the smaller is the testing data (*DF_Test*), decreasing the chances of finding a booking. Table 8 shows different BCPs for each setsize and training set percentage. Note it is intuitive to say that the bigger the setsize the better the BCP.

Table 8 – BCP for different training set sizes

Booking Coverage Performance						
Algorithm	**Setsize**	**No Train**	**Train = 50%**	**Train = 60%**	**Train = 70%**	**Train = 80%**
	20	52.17%	56.52%	62.16%	53.57%	55.56%
	50	77.17%	76.09%	83.78%	85.71%	83.33%
Hyperopt	70	83.70%	86.96%	94.59%	89.29%	83.33%

Source: Author (2022).

The table represents the behavior of our algorithm for different train/test size and setsize. We can see that a split data of 60% for training and 40% for testing returns the best BCP.

The loss function calculus is based in two main variables: *bk_found* and *bk_rank*. The *bk_found* is a value that returns a Boolean (1 or 0) showing whether or not the booking was found within the RecoSet. While the *bk_rank* is the bookings' position among the recommendations sorted by VS value. The better the booking positioned is on the rank, the lower is the error, and the lower is the loss function's output.

It is worth mentioning that the final evaluation method is based on whether or not the booking was found. It turns out that for training the algorithm, this type of feedback was resulting in big discontinuities. A booking can be ranked 1st or 50th, and it will be rewarded the same ($setsize \geq 50$). And even though we will keep the booking coverage as a final measure of success, we believed that we could help the optimization algorithm with a different loss function, adding up a part that tells us when we are getting closer to have the booking among the top 50. This change will boost the algorithm when the booking is on the 49th position compared to the 50th one.

The new loss function adds up the ranking criteria and finding criteria to compute one single score. We have named this function **loss function score**. Other loss function types were also tested but did not perform as well as the referred one. The formulas are as follows:

$$\mathrm{bk}_{\mathrm{found}} = \mathit{True\ if\ booking\ in\ RecoSet\ else\ False}, \tag{5}$$

$$\mathrm{bk}_{\mathrm{rank}} = \frac{\mathit{number\ of\ recommendations} - \mathit{booking\ position}}{\mathit{number\ of\ recommendations}}. \tag{6}$$

Therefore, we can represent our loss function by

$$\mathrm{loss}_{\mathrm{function\ score}} = -\left(bk_{found} + bk_{rank}\right), \text{ which is the most performant,} \tag{7}$$

$$\mathrm{loss}_{\mathrm{function}_{\mathrm{found}}} = -\,bk_{found}, \tag{8}$$

$$\mathrm{loss}_{\mathrm{function}_{\mathrm{rank}}} = -\,bk_{rank}, \tag{9}$$

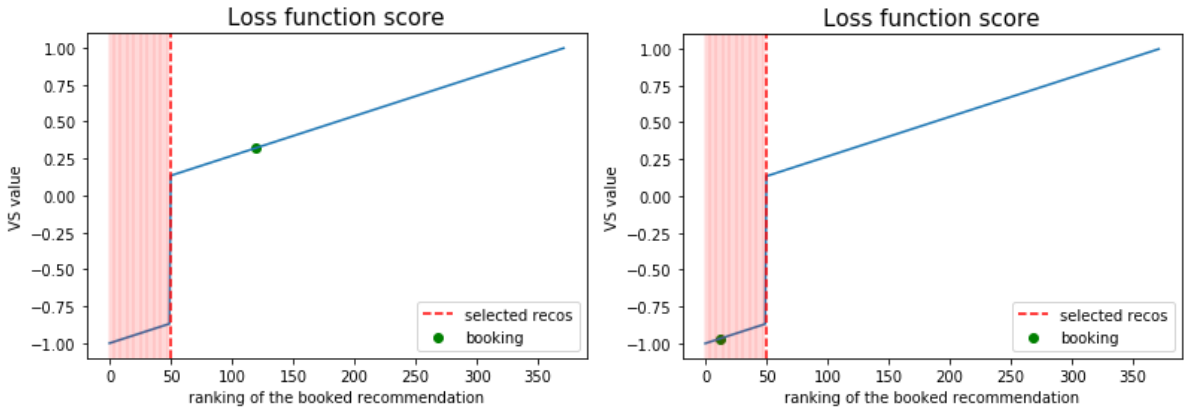
$$\mathrm{loss}_{\mathrm{function}_{\mathrm{rankfound}}} = bk_{rank}(not\ in\ bk_{found}), \text{ or even} \tag{10}$$

$$\mathrm{loss}_{\mathrm{function}_{\mathrm{2div}}} = \frac{len(recommendations)*2-(bk_{found}+bk_{rank})}{len(recommendations)*2}. \tag{11}$$

The representativeness of the best performant loss function is shown below. Nevertheless, it is important to mention a limitation. For machine learning algorithms it is known that the lost function needs to obey certain requirements: it needs to be continuous,

derivable, and convex. In the case of the implemented $\text{loss}_{\text{function score}}$ this is not the case since there is a discontinuity when $\text{loss}_{\text{function score}} = 0$. However, it fits the purpose of this work. The first one rates badly the booking outside the RecoSet, while the second one correctly rates the booking. The decision can be influenced by the computed weights on the grid search phase or by the chosen setsize. As an example, a setsize of 500 would pick all recommendations having an accuracy of 100%.

Figure 8 – Loss function with a right-guessed RecoSet and a wrong-guessed RecoSet

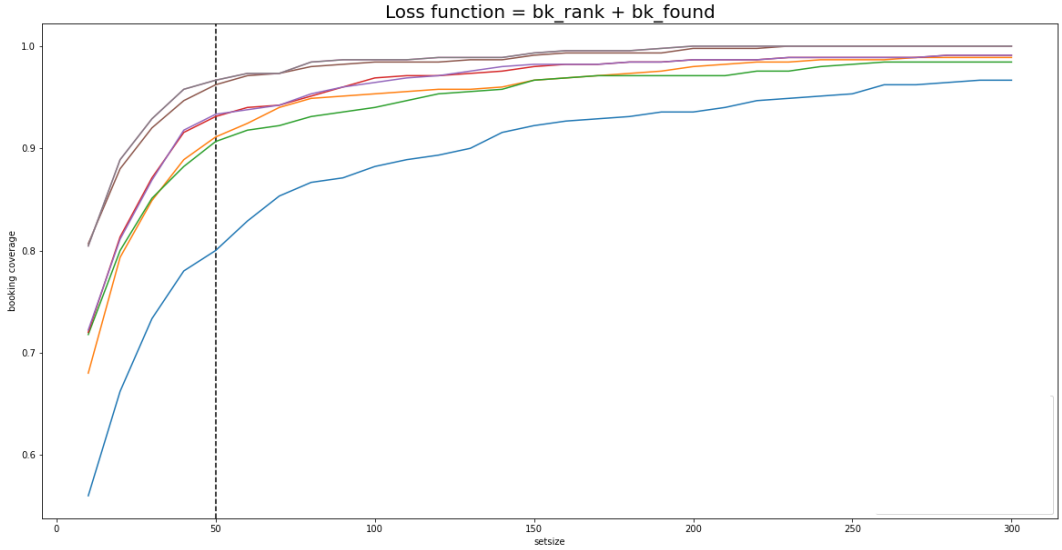


Source: Author (2022).

The results may change accordingly to the setsize, loss function, weights (which are computed by our hyperopt algorithm), and even to the train/test split data. The same process was done for every row of the dataset, and the VS was computed for every RecoSet, obtaining an assemble of good-rated bookings and bad-rated bookings, or in other words, right-guessed and wrong-guessed RecoSets.

Moreover, other calculations were suggested to increase hyperopt's performance, but they are hidden for confidentiality purposes. They will be called Value_S, Value_P, Value_NS. The challenge translates in adjusting the VS calculus (as previously explained) to get a closer representation of the representativeness of a recommendation taking all the parameters (criteria and weights) into account. Figure 9 shows the variation of the BCP for our different formulas. Every line on the graph represents a different way of calculating VS.

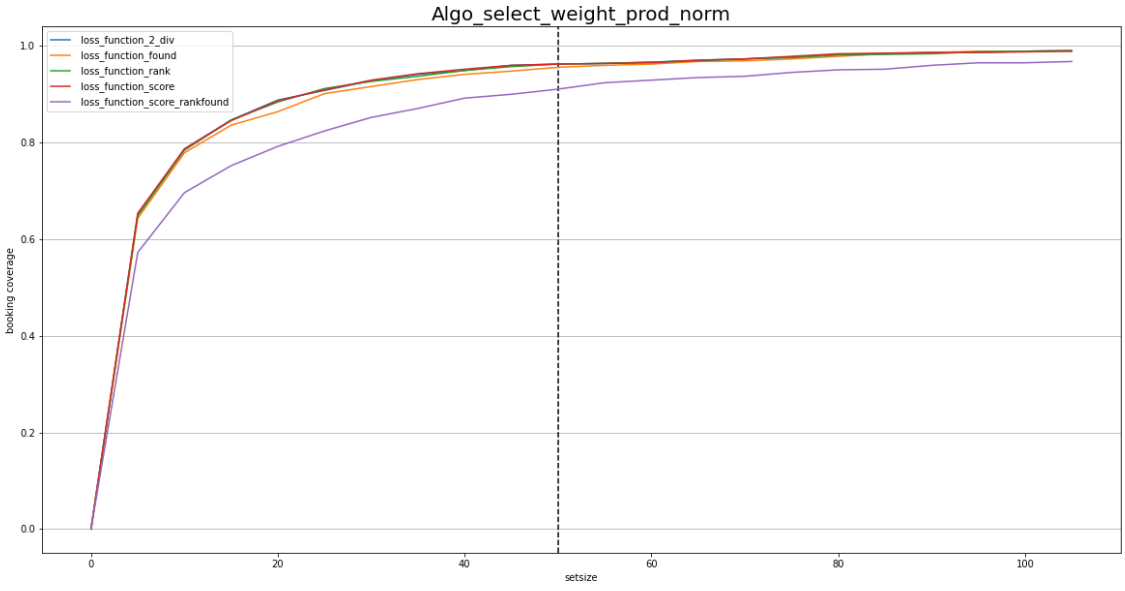
Figure 9 – BCP of different Value Search formulas throughout different setsize



Source: Author (2022).

We train the algorithm recursively by combining formulas and loss functions for computing the booking coverage with the same hyperparameters. Afterwards, to assure the effectiveness of the results, the compilation was produced in different datasets. We created a table that combines different formulas with loss functions, using 70 hyperopt iterations and 40% for test data. You can find all the data details on the appendices. Finally, the two combinations that better performed were chosen: loss function score and Value_P, and loss function score and Value_NS. Figure 10 shows the variation of performance by *setsize* of Value_NP for different loss functions.

Figure 10 – BCP of different loss functions throughout different setsize



Source: Author (2022).

There are other formulas that show higher BCP. This could allow us to perform a more refined analysis on the variation of weights for the different criteria at each iteration of hyperopt. Our challenge is then to select the right criteria to compute the value.

It is worth noting that other analyses have been done varying the **search space** of the optimizer, but these modifications do not have such a consistent effect as the select type and the loss function.

So, to move forward it is necessary to fix some settings:

1. Loss function: either loss function rank or loss function score
2. Formulas: either Value_P or Value_NS
3. Space: Hyperopt Uniform from -50 to 50.
4. *DF_train/DF_Test* percentage: 60% for train and 40% for test

4.1 Distributed Analysis

Sometimes, when analyzing BCP and comparing our algorithm's formulas, we obtained quite different results, or it was difficult to compute larger datasets. To optimize data analysis, we need to introduce a new field of Data Science: Big Data is used when datasets start getting too large to deal by traditional data-processing applications software. Therefore, we need to migrate to a Distributed processing approach, and for that we will use PySpark on Hadoop cluster. Spark can parallelize the computation and transform the original Data Frame (DF) in a Resilient Distributed Dataset (RDD).

In this phase, we read the preprocessed data stored on HDFS explained at the methodology chapter. The computation process consists in three main steps: scheme creation, grid search and evaluation. Then, we run different scenarios and log the results in a CSV format.

4.1.1 Scheme creation

Data indexation in spark demands a specific scheme that describes the structure of the Data Frame. Together they create a data structure in Spark SQL. This facilitates data type definitions, speeds up parallel computation and reduces process time. To create the full scheme (representative of the whole dataset), it was needed to read at least the first row, since all the rows have the same format.

4.1.2 Grid search

Once the scheme is created, it is time to read the data. Spark only executes assigned tasks when needed. This means that some functions schedule tasks on an executer but really do not consume computational time. The structure of Spark only performs actions to compute the data when this information is demanded. This strategy is called lazy execution.

We split the data at 60% train and 40% test. At training phase, just like before, the hyperopt algorithm runs multiples iterations and updates the weights to find the loss function's local minimum. At the end of the computation, we obtain the optimal weights.

4.1.3 Evaluation

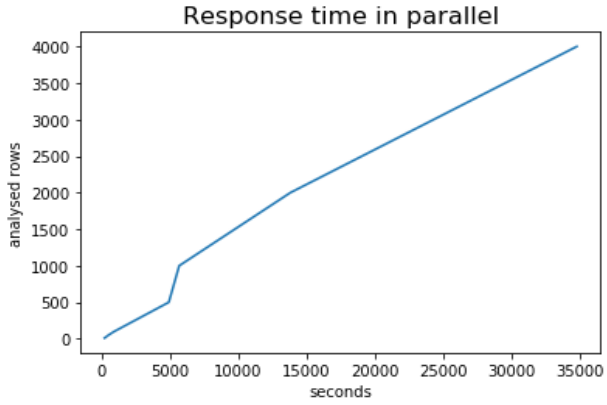
The recommendations are then sorted according to their computed VS and the top 50 is chosen. The evaluation is done only on the test dataset. The Hyperopt analysis is again done for different formulas and the BCP is computed for every one of these guesses. By the end of the evaluation process, we obtain BCPs for fix hyperparameters: number of hyperopt iterations and number of dataset rows.

4.2 Response Time

As shown further ahead, we faced two main problems to correctly parallelize the Hyperopt Algorithm using Spark's framework: (i) a long-time response, and (ii) the inconsistency of booking coverage performance.

When analyzing the algorithm's response time, we noticed a linear correlation with the size of the dataset. This was not the expected behavior. The code was shooting one job at a time and waiting one task to be finished to send the next one. The processing time was increasing tremendously, as indicated in Figure 11.

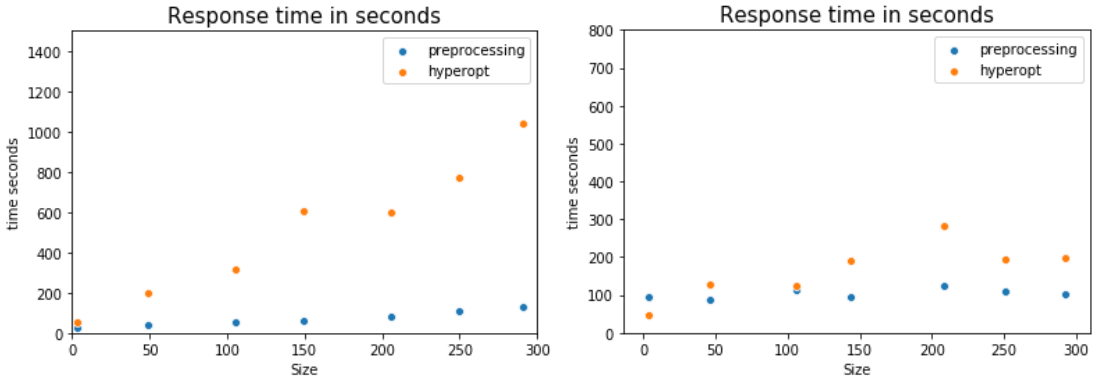
Figure 11 – Response time in seconds for different number of rows



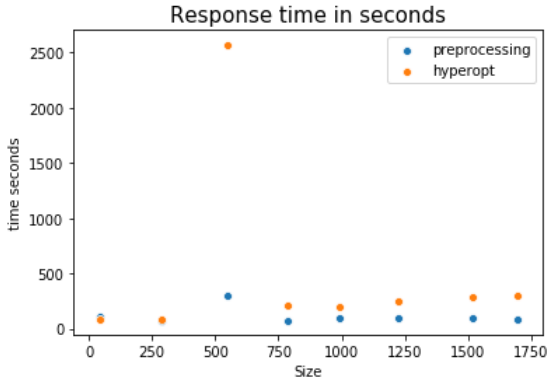
Source: Author (2022).

On the first image we observe that the time of Hyperopt processing increases linearly with the size of processed rows. The goal is to flat this curve. The solution was to force the spark framework to distribute the data into partitions. The bigger the quantity of partitions, the easier it is for spark to distribute tasks into its workers. This means that small parts of the data are being processed at the same time in different executors. The next graphs are the representations of the same analysis with different number of partitions. Figure 12 illustrates this scenario.

Figure 12 – Comparison of response time for different number of partitions



1 partition 50 partitions

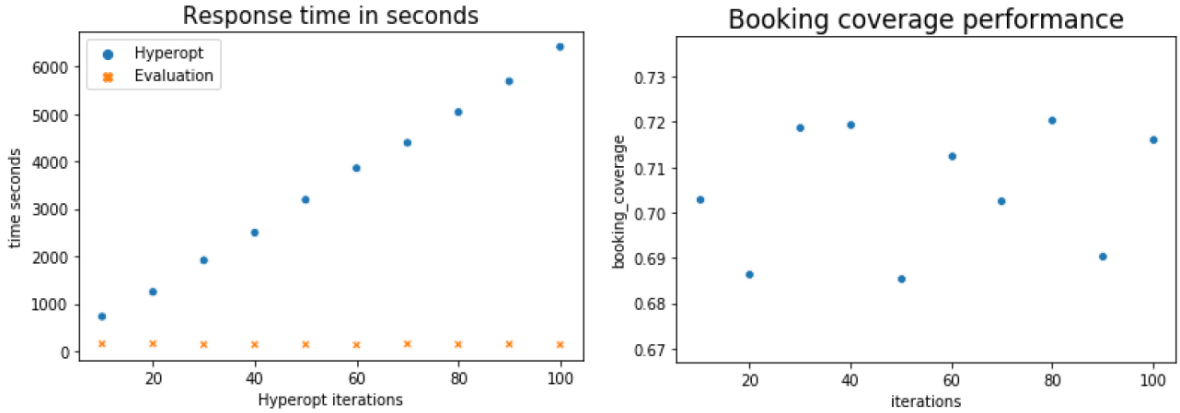


200 partitions

Source: Author (2022).

Once we solved the partition time consuming problem, is time to vary the number of hyperopt iterations to see how much time they take and how much representative they are when calculating loss function's local minimum. Figure 13 shows the number of times in seconds taken by hyperopt to process 5,000 rows with different iterations number as hyperparameter.

Figure 13 – Comparison of response time for different hyperopt iterations

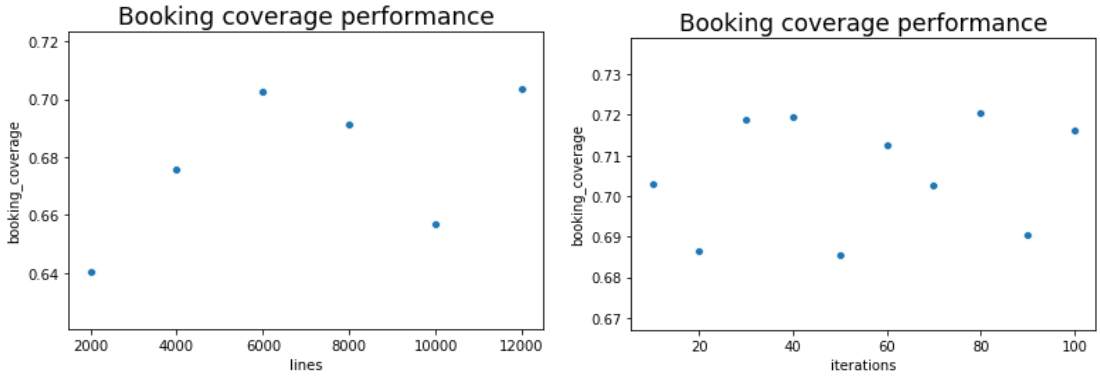


Source: Author (2022).

When comparing BCP results, we conclude that the number of hyperopt iterations does not change more than 2% the algorithm's performance for a 5,000 rows dataset. This means that the results between the models will only be valid with a confidence interval greater than 2% in this case.

From Figure 14 we see the inconsistency of BCP when varying the number of dataset rows for a fixed number of iterations (left) or varying the number of hyperopt iterations for a fixed number of dataset rows (right). Since the dataset is being randomly split, every point has a different training/testing dataset and thus, provides a different result.

Figure 14 – BCP variation for different dataset sizes and hyperopt iterations



Source: Author (2022).

The solution was to run the same evaluation multiple times with the same hyperparameters to see if we get the exact same result. If not, we can get an approximate representation of the algorithm's performance. Then, we compute the mean, median and standard deviation for the BCP, and we can validate the reliability of the results, i.e., provide confidence intervals.

4.3 Parallelization-Memory tradeoff

In this section, we compute the big data and obtain all the values we need to start the statistical evaluation. As previously mentioned, we have detected a certain variance in our BCP according to changes in parameters and hyperparameters. Then, the idea is to repeatedly simulate the same computation with fixed parameters to understand its distributions. To every run we called a round where we do the analyses and save the results.

The main.py could be exemplified as follows:

Frame 1 – Main Function

Main function

1. Read dataset from HDFS
2. For r in rounds:
    a. Train test split on dataset
    b. Grid search computation using hyperopt optimizer on train data
    c. Evaluate the obtained weights on test data
    d. Write results on a csv file

Source: Author (2022).

The results were computed approximately 50 rounds for each combination of different hyperparameters, aka hyperopt iterations and number of lines. Later, we added other parameters like number of partitions and setsize. Finally, the results were also compared with new evaluations using dummy algorithms as select functions, like a simple sort by price or sort by eft. Everything was added in the same file at each round. The final csv file looks like the content in Table 9.

Table 9 – Comparison of BCP for different formulas with the same hyperparameters

Rounds	Function	Setsize	Iterations	Limit	Repartitions	BCP	Dummy_1	Dummy_2	Dummy_3
1	Value_P	50	10	100	300	66.67%	22.22%	61.11%	55.56%
2	Value_P	50	10	100	300	63.89%	41.67%	63,89%	63.89%
3	Value_P	50	10	100	300	52.78%	36.11%	50.00%	55.56%

Source: Author (2022).

After preprocessing the data, the goal is to store it in a sample file format that could be easily loaded for further analysis. On the HDFS-Spark phase, load process was much more important, when at first it was not a problem over a small quantity of data.

Once everything stored in HDFS, the readable data frame was given by four main information: (i) search: relative to query information's such as search date and Origin and Destination (ONDs) pairs, (ii) booking: the booked recommendation information only, (iii) result: information about all the RecoSets available and the list of recommendations itself. Moreover, every Row was identified by a unique; (iv) booking id, allowing the information to by correctly indexed, as shown on Table 10.

Table 10 – Top 5 rows of RDD

booking_id	search	booking	result
621afc246e2f43af9...	[116, 2022-09-25,...	null	[559.9, 30.083334...
c1f4b782489a4633b...	[109, 2022-09-18,...	null	[188.5, 4.5, 4.5,...
37bacc903c3e4d79a...	[15, 2022-06-16, ...	null	[101.97, 3.566666...
4742483cf1104e70a...	[1, 2022-06-02, T...	[2.25, 225.8,, 79...	[155.8, 2.0, 2.0,...
18d65e94dde84eceb...	[2, 2022-06-03, J...	null	[342.4, 3.25, 3.2...

Source: Author (2022).

Once filtering data from a million bookings, we end up with only approximately half (517,586 bookings). After preprocessing it, the actual amount of computable data reduces to 300,000 bookings, which we need to consider as a fair amount to make representative assumptions.

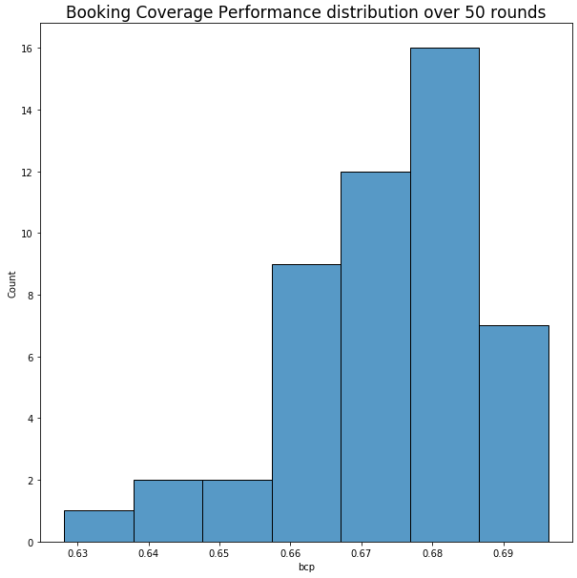
When we trained the same algorithm several times, we find that the results were varying significantly from one run to another. This can be due the randomness train/test split or because of hyperopt grid search inconsistency, or even both. Comparing algorithms thus requires estimating this noise to make sure an increase of booking coverage from one algorithm to another is significant. It requires classic statistical analysis.

To overcome this problem, we need to repeat the results and estimate an error margin. Moreover, we need then to adjust our parameters, i.e., determine a data length and several (hyperopt) iterations so that we can reduce this margin. The question is how much noise we should allow between evaluations in such a way that we can affirm with a certain accuracy that one formula is better than another.

4.4 Statistical Analysis

To start our analysis, we have assumed that our noise is normally distributed. The graph in Figure 15 presents a histogram with the frequency of BCPs obtained over 50 rounds.

Figure 15 – Distribution of BCP within 50 rounds



Source: Author (2022).

Regarding the Empirical Rule for a normal distribution, we know that:

~68% of the data is within one standard deviation of the mean

~95% of the data is within 1.96 standard deviations of the mean

~99.7% of the data is within 3 standard deviations of the mean

Assumptions underlying the following empirical rule:

- The mean estimation error is zero
- The distribution of the errors in the estimates s normal

Moreover, according to a math corollary:

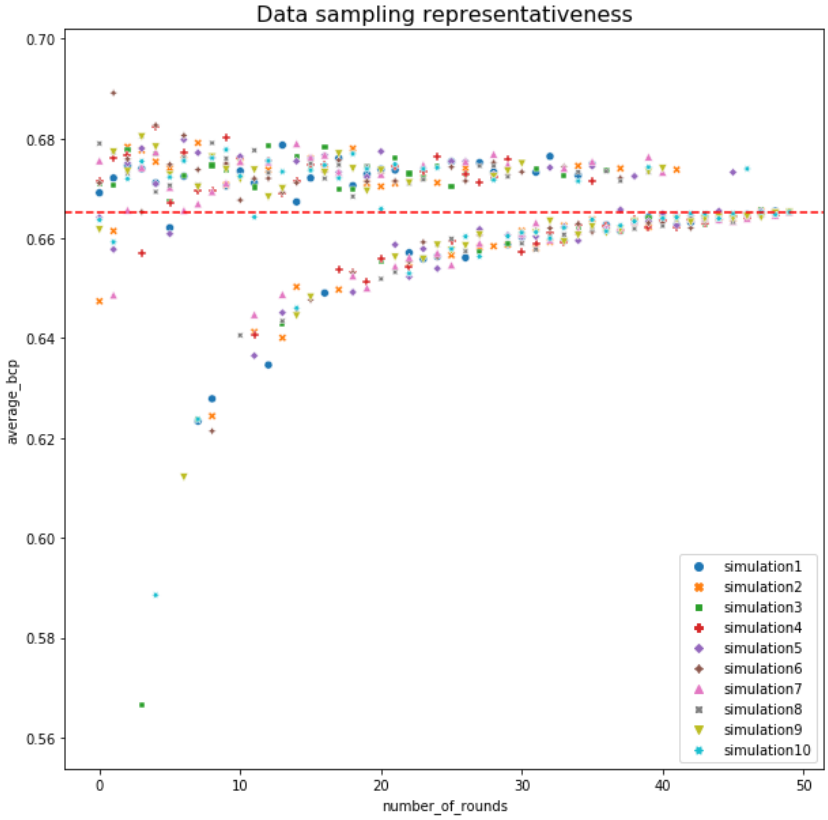
Let $Y_1, Y_2, \ldots, Y_n$ be random sample of size n from a normal distribution with mean μ and variance σ^2. Then the sample mean, $Y' = \frac{1}{n}\sum_i^n Y_i$, is also normally distributed with mean μ but variance $\frac{\sigma^2}{n}$. Which implies that $\frac{Y'-\mu}{\sigma/\sqrt{n}}$ is a standard nromal random variable Z.

Which means that if the BCP is normally distributed, the average of 10 simulations of BCP would also be normally distributed.

Finally, with these values, we can estimate the acceptable noise $\mathcal{N}(\mu, \sigma^2)$. Furthermore, it is intuitive that the bigger the number of iterations (itr) and the number of bookings analysed (lim) are, the better is the representativeness of our BCP average. Then I can start with a small number of itr/lim and if R is valid for these hyperparameters, it is valid for any larger set.

The graph in Figure 16 shows the calculus of the sample mean of R rounds. The data is sampled 10 times, every time being a different simulation. The first simulations are means of 1 round, aka the BCP itself. As we increase the number of rounds, the means converges to a certain value, which intuitively is the mean of the entire sample. $Hyperparameters: lim = 5000,\ itr = 10$.

Figure 16 – Distribution of the mean of 10 simulations of BCPs throughout the number of rounds (from 1 to 50)



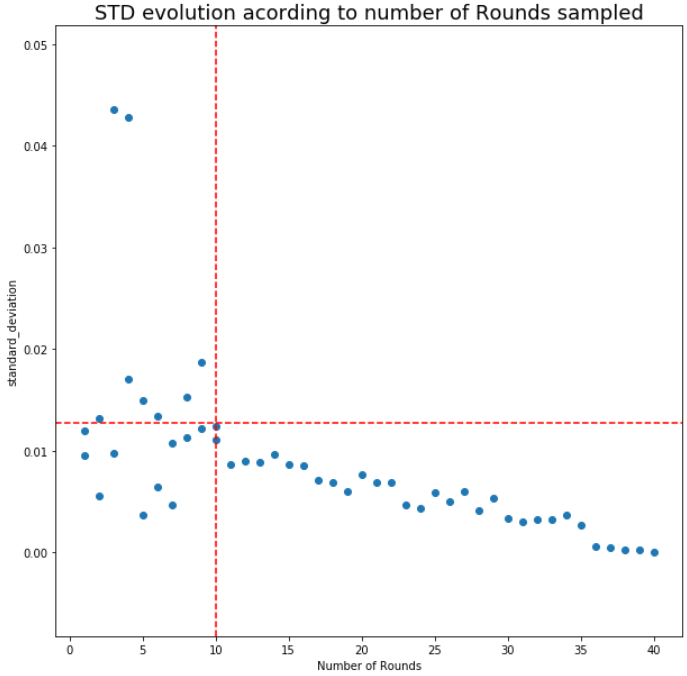
Source: Author (2022).

Now, considering some assumptions:

- An acceptable error margin would be 1%. This means $\mu \pm 0.005$.
- The BCP average for 10 rounds is approximately 65% and its std is 0.78% or approximately 0.008.
- To estimate a good formula over another we need a $\sigma \leq 0.005$.

The next step is to find our std ($\sigma = 0.005$) on the graph. According to the Law of Large Numbers we admit that after a certain point, the average will converge into a value and the variance tends to zero (00.00) on the infinite, according to Figure 17.

Figure 17 – STD evolution according to number of Rounds sampled



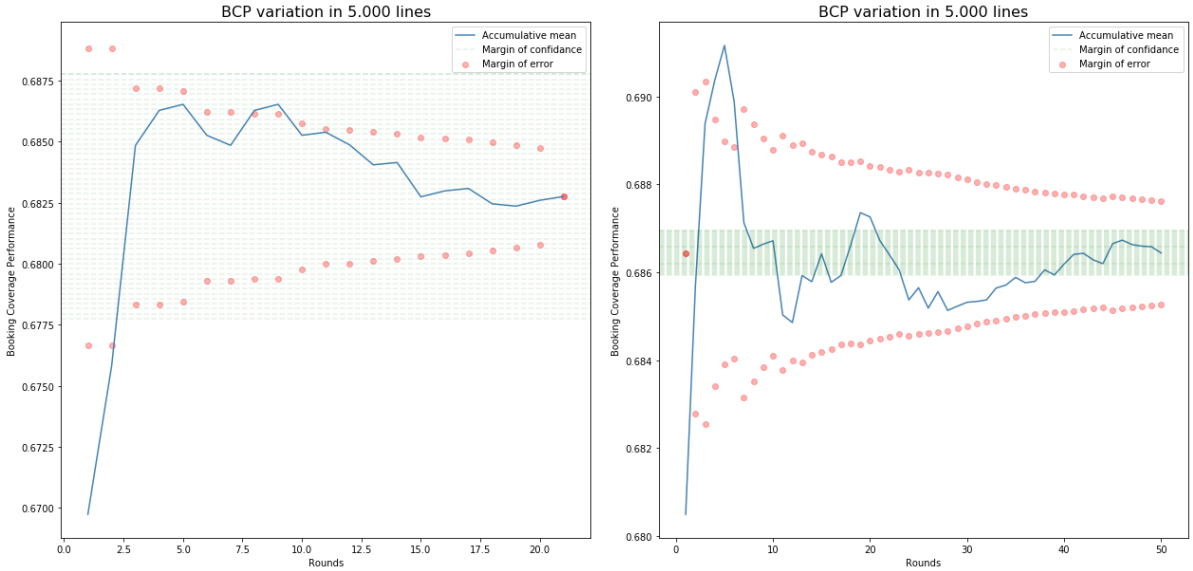
Source: Author (2022).

As we can see on the 4th quarter of the plot above, the std only decreases with the increase of R. Therefore, we could think that 10 rounds could allow a fair noise $\mathcal{N}(\mu, 0.005^2)$.

Figure 18 shows the cumulative mean in blue and its related standard deviation in red, with the variation of the green zone being the margin of confidence representing $\sigma < 0.005$. On the other hand, when analyzing just around 5,000 rows, we face another problem. When estimating $\sigma = 0.005$ (left), we see that three rounds would be enough to accept the BCP results, which could give us an uncertainty of the representativeness of the data. On the flip side, increasing $\sigma = 0.001$ (right), we get that not even 50 rounds are enough to reach a

good margin of confidence, and therefore such more rounds would demand an enormous computational power.

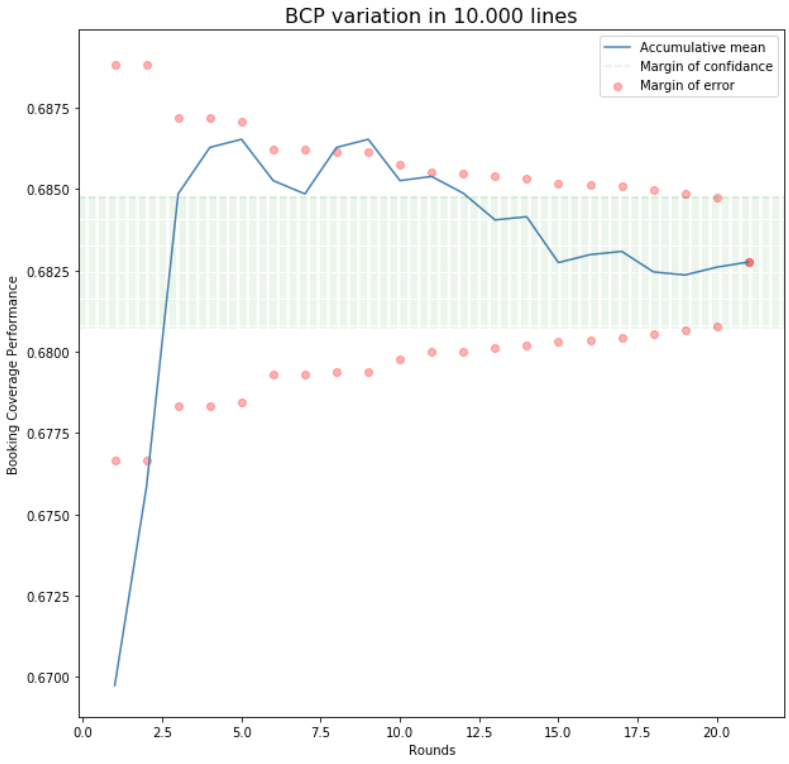
Figure 18 – Comparison over BCP variation for different standard deviations



Source: Author (2022).

The only way of overcoming this problem is by increasing the size of the data. When analyzing over 10,000 rows as shown in Figure 19, we can have a good guess that with $\sigma = 0.002$ (20 rounds) we would be within a good confidence interval.

Figure 19 – BCP variation in a large dataset

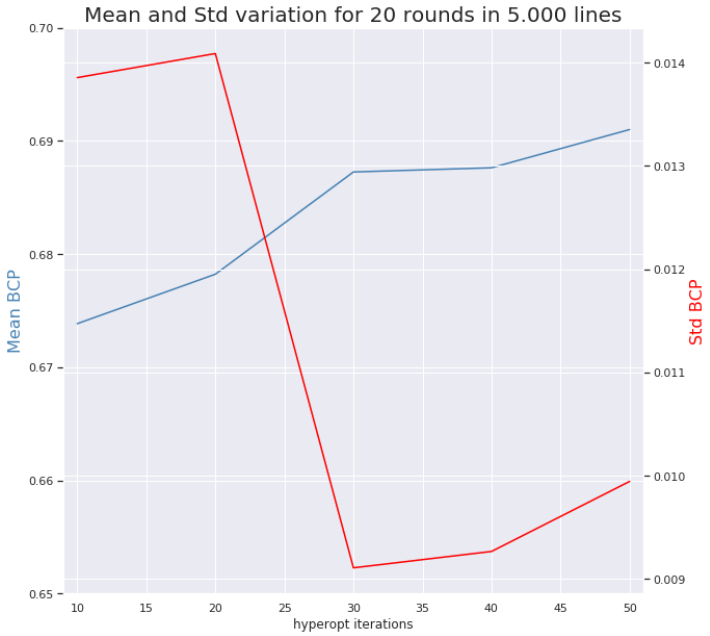


Source: Author (2022).

Now that we have fixed 20 Rounds, we need to adjust the hyperparameters, starting by hyperopt iterations at grid-search phase, illustrated at Figure 20.

This parameter defines the number of times the weights will be updated. As predicted, the more iterations we implement, the better will be the weights and bigger will be the BCP. What is an interesting thought is to see that the std of BCP increases considerably after 50 iterations. That could be explained due to overfitting, meaning that the weight dictionary is a well representative to the train data. And even though BCP has a good performance, it considerably changes its value at every random train test split. Then I believe an acceptable value would be $itr = 30$.

Figure 20 – Mean and std variation for 20 rounds

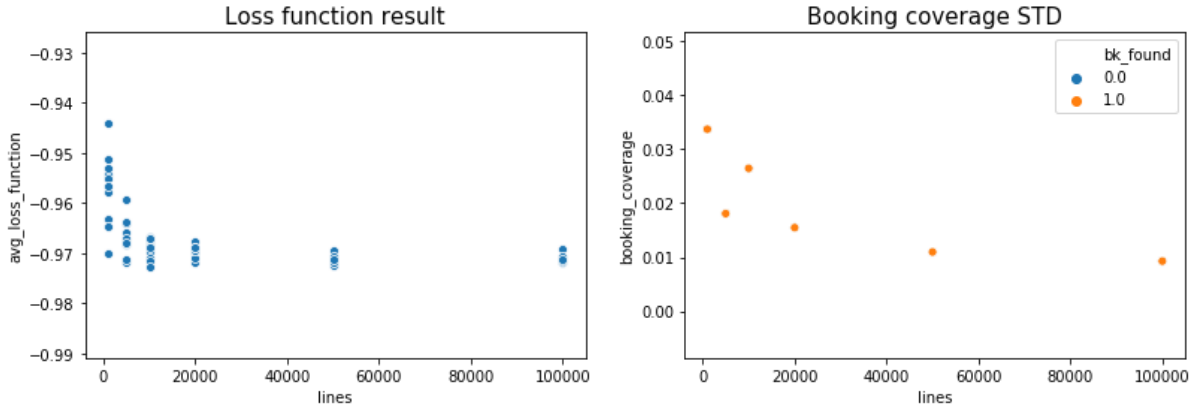


Source: Author (2022).

The number of rows is equivalent to the number of bookings analyzed, because there is one booking per line. This number is defined at the filtering phase as the limit of rows read from the data frame. The bigger this number is, the better is the representation of the data, but also the bigger is the processing time.

The graphs in Figure 21 show that for a fixed number of hyperopt iterations on grid search phase, the calculated loss function, and the variance of BCP decay when increasing the number of lines. We can see that after a certain point, the reduction is not that considerable, meaning that we can stop the evaluation after a certain limit allowing a small error. But which limit exactly?

Figure 21 – loss function and BCP std evolution throughout the number of dataset rows.



Source: Author (2022).

Table 11 shows the mean and standard deviation over 20 rounds for different data sizes. We can see that the $BCP_{1000} > BCP_{1000}$ over 0.01, which is a reliable affirmation since its $\sigma = 0.008$. We could then assume a 10,000 data sample to do our analysis ($lim = 10{,}000$).

Table 11 – BCP statistical results

BCP	mean	Std	Rounds
100	0.681944	0.080344	20
1000	0.671040	0.026634	20
10000	0.682749	0.008413	20
50000	0.684284	0	1
300000	0.690408	0	1

Source: Author (2022).

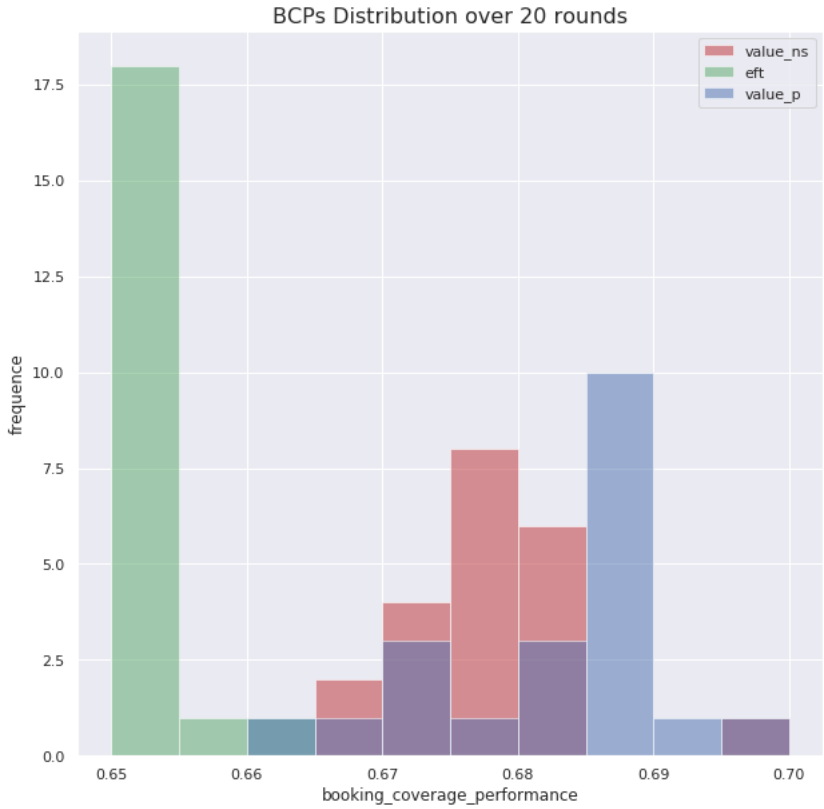
The highest dataset size that was successfully computed by main.py was 50 thousand rows. The 300 thousand rows dataset was also computed but with only 10 hyperopt iterations (instead of 30). A greater number of iterations was too much computational memory to Jupyter's spark session.

The graph on Figure 22 illustrates the distribution of BCP in three different formulas of using hyperopt optimizer. It is possible to see that although with less variance, the eft results have a smaller average BCP than the two following ones. Also, Value_NS is more consistent than Value_P in terms of standard deviation.

Finally, Table 12 compares multiple combinations of hyperparameters. If we assume that the mean of Value_P is bigger than the mean of Value_NS, we need to check of their standard deviation doesn't scape the 95% confidence interval. When analyzing for 100

and 5,000 rows, this statement true, i.e., it stays on the confidence interval (in greed). However, when analyzing 10,000 rows it is not the case (in red).

Figure 22 – BCP distribution for 20 rounds over 10000 rows



Source: Author (2022).

According to Table 12, we can see that Value_P algorithm performs 3.69% better than Value_NS for 5,000 rows (bookings) and 0.49% on 10000 rows. Although, only on the first case we achieve the confidence interval of 95% because its intervals of 1.96 standard deviations are lower than Value_NS' mean on the first case. On the second example, more tests or different tunning methods should be improved to make a correct statement.

It is important to mention that an improvement of 1% in our algorithm's BCP represents over 6 million well recommended bookings per year, what could translate into an income over 5.5 million euros per year for Amadeus Air Business.

Table 12 – BCP for different formulas and dataset sizes

	Rounds	Value_P	Price	Eft	Value_NS	Rows
Mean	10	68,00%	39,00%	66,00%	65,00%	100
	20	68,00%	39,00%	66,00%	65,00%	100
	10	67,67%	38,68%	65,67%	60,80%	5000
	20	68,45%	38,82%	65,80%	64,76%	5000
	10	68,53%	38,38%	65,04%	67,82%	10000
	20	68,27%	38,19%	65,05%	67,78%	10000
Std	10	00,09%	00,01%	00,01%	00,12%	100
	20	00,09%	00,01%	00,01%	00,12%	100
	10	02,33%	02,49%	03,61%	01,32%	5000
	20	01,33%	00,81%	00,72%	03,53%	5000
	10	00,87%	01,38%	00,63%	00,84%	10000
	20	00,84%	00,94%	00,42%	00,64%	10000

Source: Author (2022).

5 CONCLUSIONS

As part of the conclusions of the work presented in this final paper, recommendation systems are not trivial to model, and they involve multiple factors that change the nature of the problem. Regarding the algorithm developed and presented, the creation of a new formula, new loss function and the validation method were key to an improvement on Amadeus' value search engine.

With the related work we could find good estimators of weights for recommendation criteria by grid search method, and used hyperopt optimizer, to tune or hyperparameters. On the distributed programming part of the work, the scheme on Figure 7 illustrates the different tasks using HDFS for the data read and write and Jupyter Notebook with Spark framework for the computation. Finally, statistical tests prove that the Value_P algorithm performs 1% better than Value_NS with a 95% of confidence interval.

As a main conclusion, with the correct techniques and adjustments showed on this work, Amadeus' value search engine can archive better booking coverage when selecting recommendations subsets. This factor could have a positive impact on Amadeus' recommendation system.

As additional conclusions, I believe that research in machine learning and deep learning algorithms are practical for understanding how to optimize subset selection, even if not fully used in real world scenarios. New technologies such as cloud computing that will soon be implemented in the Amadeus' product system will change the way we store and analyze data. By initiating the code migration to Azure cloud from Microsoft, we can use MLib library in Databricks or even other machine learning methods, such as the Learn to Rank method or the Transformers technique on dynamic processes.

As far as recommendations for future works, I suggest using Support Vector Machines – SVMs for classification and prediction analysis. Regarding past bookings, it would be interesting in analyzing price, eft, and number of connections behaviors separately and combining these criteria with other parameters not yet studied. Furthermore, one could do segmentation and clusterizations of queries by users' profile, different time periods, flight criteria, etc. A possibility is also expanding the Database to OW+RT.

As a final note, the research work reported in this paper is far from being finished. But it has brought a tangible result for Amadeus, high in the maturity technology level, so that the company can use the knowledge developed to improve its flight recommendation products.

6 REFERENCES

AMDEUS, Support & Knowledge. **Como adicionar observações ao PNR (Críptico).** 2022. Disponível em: https://servicehub.amadeus.com/c/portal/view-solution/938255/como-adicionar-observacoes-ao-pnr-criptico-. Acesso em: 17 Dez. 2022.

AVCONTENTTEAM. **Introductory guide on Linear Programming for (aspiring) data scientists.** Analytics Vidhya 2017. Disponível em: https://www.analyticsvidhya.com/blog/2017/02/lintroductory-guide-on-linear-programming-explained-in-simple-english/. Acesso em: 15 Dez. 2022.

BAKER, B. O.; HARDYCK, C. D.; PETRINOVICH, L. F. Weak Measurements vs. Strong Statistics: An Empirical Critique of S. S. Stevens' Proscriptions nn Statistics. **Education and Psichological Measurement**, v. 26, n. 2, 1966.

COSSETI, M. C. **Quantos filmes têm na Netflix?** Tecnoblog. 2018. Disponível em: https://tecnoblog.net/responde/quantos-filmes-tem-na-netflix/. Acesso em: 22 Nov. 2022.

DAS, D.; SAHOO, L.; DATTA, S. A Survey on Recommendation System. **International Journal of Computer Applications,** v. 160, n. 7, 2017.

DASZYKOWSKI, M.; WALCZAK, B.; MASSART, D. L. Representative subset selection. **Analytica Chimica Acta.** v. 468, n.1, 2002.

HADOOP APACHE. **HDFS Architecture Guide**. 2022. Disponível em: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Acesso em: 15 Abr. 2022.

IPPOLITO, P. P. **Hyperparameters Optimization**. 2019. Disponível em: https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d. Acesso em: 27 Mar. 2022.

JOHN, R. KOHAVI, K. PFLEGER, B. et al. Irrelevant features and the subset selection problem. In: **Proceedings of the eleventh international conference on machine learning**, v.129, 1994.

MALATO, G. **Hyperparameter tuning.** Grid search and random search. YOUR DATA TEACHER, 2021. Disponível em: https://www.yourdatateacher.com/2021/05/19/hyperparameter-tuning-grid-search-and-random search/#:~:text=Grid%20search%20is%20the%20simplest,performance%20metrics%20using%20cross%2Dvalidation. Acesso em: 27 Mar. 2022.

OWEN, G. **Game Theory**. Japan: Emerald, 2013. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=yeVbAAAAQBAJ&oi=fnd&pg=PP2&dq=G.+OWEN,+2013,+GAME+THEORY&ots=Yo7UQBRIkg&sig=vpQM85ZBAkDNqPntyCckiws8ooQ#v=onepage&q=G.%20OWEN%2C%202013%2C%20GAME%20THEORY&f=true. Acesso em: 16 Dez. 2022.

ROCCO, D. **Bayesian Hyperparameter Optimization with MLflow**. phData, 2021. Disponível em: https://www.phdata.io/blog/bayesian-hyperparameter-optimization-with-mlflow/. Acesso em: 15 Ago. 2022.

SHARMA, S. **Spark & MapReduce:** Introduction, Differences & Use Case. K21Academy, 2022. Disponível em: https://k21academy.com/BIG-DATA-HADOOP-DEV/SPARK-VS-MAPREDUCE/. Acesso em: 05 Ago. 2022.

TOMINAGA, Y. Representative subset selection using genetic algorithms. **Chemometrics and Intelligent Laboratory Systems**, v.43, 1998.

WORLD TRAVEL & TOURISM COUNCIL. **Powering better journeys through travel technology.** 2019. Disponível em: https://amadeus.com/en. Acesso em: 16 Dez. 2023.

APPENDICE A – GIT AND BASH ORGANIZATION

At the start, the local part of the work was merged into the master branch of our Bitbucket Repository. But since we kept developing the code to distributed analysis, some major changes were necessary. Therefore, a release branch was created to save the POO code, and the master received the merge of the Spark code branch. A following step-by-step scheme illustrates the new branch organization.

1. Created branch release/hyperopt_object_oriented from the master branch to save the code developed at master so far.

2. Merge the code into feat/hyperopt_remote (developing branch).

3. Updates to change the code into Spark framework.

4. Pull request from feat/hyoperopt_remote to master branch

From this point on, the git commits were not only made from our local machines, but also from our user accounts on the Hadoop cluster. On every major modification on the cluster, we needed to import the code at local. Using Linux terminal, from one side we have a logged account connected on the cluster and the other one from my local machine. The code transmission was made using git commands. The image below illustrates one simple operation.

```
nico@nico-VirtualBox:~/contextualization$ git branch
* feat/hyperopt_remote
  master
  release/hyperopt_object_oriented
nico@nico-VirtualBox:~/contextualization$ git pull
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 9 (delta 7), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), 85.92 KiB | 18.00 KiB/s, done.
From ssh://git-ssp.cicd.rnd.amadeus.net:7999/sadam/contextualization
   ea91704..2c5c96e  feat/hyperopt_remote -> origin/feat/hyperopt_remote
Updating ea91704..2c5c96e
Fast-forward
 bash/write.sh                              |   8 +-
 pyspark_with_dependencies/dependencies.zip | Bin 112235828 -> 112235828 bytes
 src/writeonhdfs.py                         |  22 +---
 src/writeonhdfs_loop.py                    | 353 ++++++++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 363 insertions(+), 20 deletions(-)
 create mode 100644 src/writeonhdfs_loop.py
nico@nico-VirtualBox:~/contextualization$
```

```
Last login: Thu Sep 29 07:04:13 2022 from 10.67.121.173
CentOS Core 7.9

################################################################
#                                                              #
#                           WARNING                            #
#                                                              #
# THIS SERVER IS ONLY TO START APPS ON INR/SSP HADOOP CLUSTER! #
# DO NOT RUN ANY HEAVY LOCAL TASKS!                            #
#                                                              #
################################################################

[ndefreitassoares@nceorilnx-v44 ~]$ cd contextualization/
[ndefreitassoares@nceorilnx-v44 contextualization]$ git pull
Already up-to-date.
[ndefreitassoares@nceorilnx-v44 contextualization]$ git add -all
error: did you mean `--all` (with two dashes ?)
[ndefreitassoares@nceorilnx-v44 contextualization]$ git add --all
git commit -[ndefreitassoares@nceorilnx-v44 contextualization]$ git commit -m "all 20220527 in HDFS"
[feat/hyperopt_remote 2c5c96e] all 20220527 in HDFS
 Committer: Nicolas DE FREITAS SOARES <ndefreitassoares@nceorilnx-v44.nce.amadeus.net>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 4 files changed, 363 insertions(+), 20 deletions(-)
 create mode 100644 src/writeonhdfs_loop.py
[ndefreitassoares@nceorilnx-v44 contextualization]$ git push
Counting objects: 16, done.
Delta compression using up to 48 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 85.94 KiB | 0 bytes/s, done.
Total 9 (delta 7), reused 0 (delta 0)
remote:
remote: View pull request for feat/hyperopt_remote => master:
remote:   https://git-ssp.cicd.rnd.amadeus.net/git/projects/SADAM/repos/contextualization/pull-reques
ts/13
remote:
To ssh://git@git-ssp.cicd.rnd.amadeus.net:7999/sadam/contextualization.git
   ea91704..2c5c96e  feat/hyperopt_remote -> feat/hyperopt_remote
[ndefreitassoares@nceorilnx-v44 contextualization]$
```

From the cluster side, a gateway allowed us to access Hadoop storage and read and load large data files. As mentioned on section 4.3, after the pre-processing phase and writing the data on HDFS, we could check stored files divided by search date, as illustrated below.

```
[ndefreitassoares@nceorilnx-v44 contextualization]$ hdfs dfs -ls json_files
Found 10 items
-rw-r--r--   2 ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/_SUCCESS
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-05-31
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-06-01
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-06-02
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-06-03
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-06-04
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-06-05
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-06-06
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-06-07
drwxr-xr-x   - ndefreitassoares ndefreitassoares          0 2022-09-13 15:51 json_files/search_date=2022-06-08
[ndefreitassoares@nceorilnx-v44 contextualization]$
```

All the process of creation of a spark-submit environment consists in three steps:

1. Virtual environment creation
2. Install python requirements
3. Build bandle.sh file

Table Results

1A from 13/03/2022					
	LF_2_div	**LF_found**	**LF_score**	**LF_rank**	**LF_rankfound**
Cheapest	78.67%	78.67%	78.67%	78.67%	78.67%
Fastest	90.00%	90.00%	90.00%	90.00%	90.00%
Value_S	94.17%	94.83%	95.00%	94.83%	95.00%
Value_NS	90.33%	90.33%	90.33%	90.50%	90.50%
Value_NSMV	90.17%	90.67%	90.17%	90.50%	90.33%
Value_P	95.67%	95.50%	95.67%	95.50%	95.17%
Value_NP	95.17%	95.50%	95.67%	95.50%	95.17%
Value_NPMV	95.17%	95.17%	95.17%	95.17%	95.17%

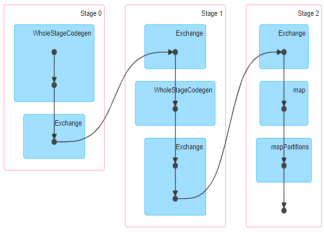
1A from 27/05/2022					
	LF_2_div	**LF_found**	**LF_score**	**LF_rank**	**LF_rankfound**
cheapest	76.72%	76.72%	76.72%	76.72%	76.72%
fastest	91.07%	91.07%	91.07%	91.07%	91.07%
Value_S	94.00%	93.12%	93.56%	93.56%	93.56%
Value_NS	90.92%	91.51%	91.36%	90.92%	91.36%
Value_NSMV	91.65%	91.51%	90.78%	91.36%	91.22%
Value_P	93.70%	93.70%	92.83%	92.97%	93.70%
Value_NP	92.97%	92.68%	92.97%	92.97%	93.12%
Value_NPMV	93.12%	93.12%	93.12%	93.12%	93.12%

Running Jobs

Details for Job 0

Status: RUNNING
Active Stages: 1
Pending Stages: 1
Completed Stages: 1

▸ Event Timeline
▾ DAG Visualization



▾ Active Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	javaToPython at NativeMethodAccessorImpl.java:0 +details (kill)	2022/09/13 14:07:25	19 s	0/1 (1 running)			982.2 MB	

▾ Pending Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	runJob at PythonRDD.scala:153 +details	Unknown	Unknown	0/1				

▾ Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	javaToPython at NativeMethodAccessorImpl.java:0 +details	2022/09/13 14:05:41	1.6 min	33/33	4.0 GB			10.3 GB