Project Title	High-performance data-centric stack for big data applications and operations
Project Acronym	BigDataStack
Grant Agreement No	779747
Instrument	Research and Innovation action
Call	Information and Communication Technologies Call (H2020-ICT-2016-2017)
Start Date of Project	01/01/2018
Duration of Project	36 months
Project Website	http://bigdatastack.eu/

D6.2 – Use case description and implementation Y3

Work Package	WP6 – Use case description and implementation – M34
Lead Author (Org)	Maurizio Megliola (GFT)
Contributing Author(s) (Org)	Stathis Plitsos (DANAOS), Bernat Quesada Navidad (ATOS Worldline), Antoni Munar (GFT Spain), Richard McCreadie (University of Glasgow), Anestis Sidiropoulos (ATC), Dimitris Poulopoulos (UPRC)
Due Date	31.10.2020
Date	12.11.2020
Version	1.0

Dissemination Level

X	PU: Public (*on-line platform)
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

The work described in this document has been conducted within the project BigDataStack. This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 779747. This document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

Versioning and contribution history

Version	Date	Author	Notes
0.1	14.10.2020	Maurizio Megliola (GFT)	ToC and initial version from D6.1
0.2	29.10.2020	Bernat Quesada Navidad (ATOS)	Contribution to the Connected Consumer use case
0.3	29.10.2020	Stathis Plitsos (DANAOS)	Contribution to the Shipping use case
0.4	30.10.2020	Richard McCreadie (University of Glasgow)	Contribution to the Connected Consumer use case
0.5	30.10.2020	Maurizio Megliola (GFT), Antoni Munar (GFT Spain)	Contribution to the Insurance use case
0.6	30.10.2020	Richard McCreadie (University of Glasgow)	Contribution to the Insurance use case
0.7	30.10.2020	Maurizio Megliola (GFT), Anestis Sidiropoulos (ATC)	Contribution to the Insurance use case and integration of the contributions
0.8	31.10.2020	Bernat Quesada Navidad (ATOS)	Contribution to the Connected Consumer use case
0.9	04.11.2020	Antoni Munar (GFT Spain), Dimitris Poulopoulos (UPRC)	Contribution to the Insurance use case
0.10	04.11.2020	Maurizio Megliola (GFT)	Version integrated and ready for the internal review
0.11	07.11.2020	Amaryllis Raouzaiou	Internal review
0.12	10/11/2020	Stathis Plitsos (DANAOS)	Addressing UC#1 comments
1.0	10/11/2020	Maurizio Megliola (GFT)	Addressing UC#2 and #3 comments. Final version ready for the submission

Table of Contents

List of tables

List of figures

1. Executive Summary

BigDataStack delivers a complete high-performant stack of technologies addressing the needs of data operations and applications. The BigDataStack project was conceived as a data centric platform, integrating approaches for Data as a Service. Approaches for data cleaning, data layout and efficient storage, combined with seamless data analytics will be realised holistically across multiple data stores and locations. BigDataStack holistic solution incorporates approaches for data-focused application analysis and dimensioning, and process modelling towards increased performance, agility and efficiency. A toolkit allowing the specification of analytics tasks in a declarative way, their integration in the data path, as well as an adaptive visualization environment, realize BigDataStack's vision of openness and extensibility.
This deliverable includes the description of the three BigDataStack use cases together with their implementation on top of the above mentioned architecture components and aims at showcasing the added-value and innovations of the BigDataStack technical solutions in different real-world cases. These use cases are the following:

- Real-time Shipping Management (RSM)
- Connected Consumer (CC)
- Smart Insurance (SI).

The RSM use case utilizes the BigDataStack architecture for big data management (emphasis on the data as a service key offering), its analytics and methods for scheduling of orders, preventive maintenance, visualization of the current state and final results. By incorporating these aspects through the DANAOS platform, BigDataStack allows to shipping companies to cherish their data and use them in a difficult decision-making process, such as the supply management of a fleet.

The CC use case utilizes the BigDataStack environment to implement and provide a recommender system for the grocery market. All of the data that are used for training the analytic algorithms of the use case is corporate data provided by EROSKI, one of the top food retailers companies in Spain.

The SI use case uses BigDataStack to implement smart recommendation systems for the insurance market. The datasets that is used within the process of the use case is corporate data provided by an insurance company, based in Italy, selected from the GFT customers' portfolio.

Thus, the current deliverable presents the description for the three BigDataStack use cases, the context, the goal, the datasets and the main scenarios implemented.

2. Introduction

2.1. Relation to other deliverables

The current deliverable, the final BigDataStack deliverable concerning **Use Cases** is related to several other BigDataStack deliverables in a direct or indirect way. *D2.1 (State of the art and Requirements analysis - I), D2.2 (State of the art and Requirements analysis - II) and D2.3 (State of the art and Requirements analysis - III* identify and specify the technical requirements for BigDataStack both through use case (UC) providers and technology providers, while *D2.4 (Conceptual model and Reference architecture - I), D2.5 (Conceptual model and Reference architecture - II) and D2.6 (Conceptual model and Reference architecture – III)* provide information about the key functionalities of the overall architecture, the interactions between the main building blocks and their components, along with the internals of these components regarding research approaches. In addition, the final technical deliverables from WP3, WP4, WP5 (respectively, D3.3 WP3 Scientific Report and Prototype description – Y3, D4.3 WP4 Scientific Report and Prototype description – Y3, D5.3 WP 5 Scientific Report and Prototype Description – Y3) are the deliverables which present the final technical status (dealing with **Data-driven Infrastructure Management**, **Data as a service** and **Dimensioning, Modelling and Interaction Services** respectively) of BigDataStack project.

2.2. Document structure

The document is structured as follows:

- Section 2 provides an introduction to the deliverable and a description of the document structure.
- Section 3 introduces and describes the Real-time Ship Management use case by DANAOS, providing a description and information about goals, datasets, scenarios and evaluation.
- Section 4 introduces and describes the Connected consumer use case by ATOS WorldLine, providing a description and information about goal, product recommendation models, datasets, key performance indicators, grocery recommendation performance and scenario.
- Section 5 introduces and describes the Smart Insurance use case by GFT, providing a description and information about goal, insurance dataset, insurance data analysis, insurance recommendation and churn prediction, scenarios and key performance indicators.
- Finally, in Section 6, conclusions are reported.

3. Real-time Ship Management (RSM)

This section provides a description of the Real-time Ship Management use case scenario from DANAOS.

3.1. Overview and goal

The Real-time Ship Management (RSM) use case exploits the BigDataStack environment with an emphasis on the data as a service offering for big data management, its analytics and methods for scheduling of orders, preventive maintenance, visualization of the current state and final results.

DANAOS Monitoring platform is an integrated web-based service that combines data from sensors onboard, operational data from a shipping company's database and weather data in order to assist the technical and operations department of a shipping company to monitor its vessels. Its practical use is to provide an overview of a vessel's performance and alert the users when policy-based rules are violated. By incorporating these aspects through the DANAOS Monitoring platform, BigDataStack allows shipping companies to utilize their data and use them in a difficult decision-making process, such as the monitoring of the vessel, the identification of potential failures and the supply management of a fleet.

DANAOS Routing platform is an integrated web-based service that combines data from the DANAOS Monitoring platform along with routing information and algorithms and allows to the Operations department of a shipping company to produce valid routes for vessels with respect to the ETA to the next port and weather conditions.

What is the scenario's goal?

The scenario's goal is to:

- Monitor the main engine of a vessel.
- Identify malfunction patterns and notify accordingly the Technical, the Supplies and the Operations departments.
- Adjust the vessel of the route once a predictive maintenance alert is generated.
- Minimize the overall maintenance cost.
- Avoid off-hire seasons due to machinery failures and unexpected but compulsory maintenance.

What is the business objective?

The main business objectives are:

- Advanced monitoring of key components in the engine room and at office.
- Better organisation of the Technical, the Supplies and the Operations department.
- Minimization of machinery failures that cause the ship to go off-hire.
- Minimization of consumed fuel.
- Minimization of emitted CO2.
- Reduction of operating costs, by optimising the requisition process of new spare parts.

Who are the actors in the use case?

Table 1 - Real-time Ship Management Actors

Id	Name	Description
ROL-01	Data Owner	BigDataStack offers a unified Gateway to obtain both streaming and stored data from data owners and record them in its underlying storage infrastructure that supports SQL and NoSQL data stores.
ROL-02	Data Scientist	BigDataStack offers the Data Toolkit to enable data scientists both to easily ingest their analytics tasks and to specify their preferences and constraints to be exploited during the dimensioning phase regarding the data services that will be used (for example preferences for the data cleaning service).
ROL-03	Business Analysts	BigDataStack offers the Process Modelling Framework allowing business users to define their functionality -based business processes and optimize them based on the outcomes of process analytics that will be triggered by BigDataStack. Mapping to specific process analytics tasks will be performed in an automated way.
ROL-04	Application Engineers and Application Service Owners	BigDataStack offers the Application Dimensioning Workbench to enable application owners and engineers to experiment with their applications and dimension it in terms of its data needs and data-related properties.
ROL-05	Technical Department	BigDataStack offers the Data and Services Monitoring component to monitor the produced streams of data. Results of the analytics algorithm for preventive maintenance are alerts that the engineer of the Technical department wishes to further investigate through the respective data.
ROL-06	Operations Department	BigDataStack offers the Deployment Engine Component through which custom services can be deployed. In this case the deployed services allow to an employee of the Operations department to adjust accordingly the vessel's route, if a spare part is delivered to a specific port.
ROL-07	Supply Department	BigDataStack offers the Deployment Engine Component through which custom services can be deployed. In this case, an employee of the Supply department wishes to order a specific spare part, when a malfunction occurs.

What are the key activities these actors are involved in?

The RSM use case consists of the following main activities:

1. **Acquire data**. The Data Owner wishes to set a data source from which BigDataStack obtains data to store them in its infrastructure

2. **Select attributes.** The Data scientist wishes to select a set of attributes depending on custom criteria by utilizing the Data Toolkit of BigDataSTack.
3. **Monitor the selected attributes**. The Data Scientist wishes to monitor the data through the Visualization Environment of BigDataStack.
4. **Deploy the application services.** The application engineers and application service owners wish to deploy their application services on BigDataStack.
5. **Use an analytics algorithm or inject a new one**. The Data Scientist wishes to use an algorithm from a list of existing algorithms or inject a new one into BigDataStack.
6. **Produce alerts depending on monitoring**. An engineer of the Technical department wishes to receive predictive maintenance alerts.
7. **Adjust the route of the vessel with respect to speed, main engine load and weather conditions**. An employee from the Operations department wishes to know and confirm the adjusted route once a spare part order is placed from the Supply department..

3.2. Pilot description

A vessel has to complete its route within a specific time-frame. When a part of the main engine fails unexpectedly, the ship risks staying off-hire. This can be very damaging to a shipping company, as chartering revenues decrease, while replacing a spare part immediately increases cost. Thus, identification of potential failure allows timely ordering, or even replacement of spare parts before failure. Furthermore, we wish to identify malfunctions on sensors that cause data-loss. Last, we wish to be alerted when policies of the shipping company are violated.

Regarding the main engine, in this project we focus particularly on a specific malfunction of a component of the main engine, i.e., the cross-head bearings. This is a damage that is not sudden, it evolves gradually and can only be detected with an on-site inspection. To the best of our knowledge, there is no correlation of the main engine data that pinpoint this malfunction. Its main cause is bad lubrication due to slow steaming, a policy imposed by charterers to ship-owning companies in order to reduce fuel consumption.

BigDataStack can assist with its architecture and the provided functionalities. In order to integrate DANAOS Monitoring and Routing platforms into BigDataStack and cherish its flexibility and functionality, all components of the architecture should be utilized. However, there is a set of components which are of major importance. The following figure describes these components and their interconnection.

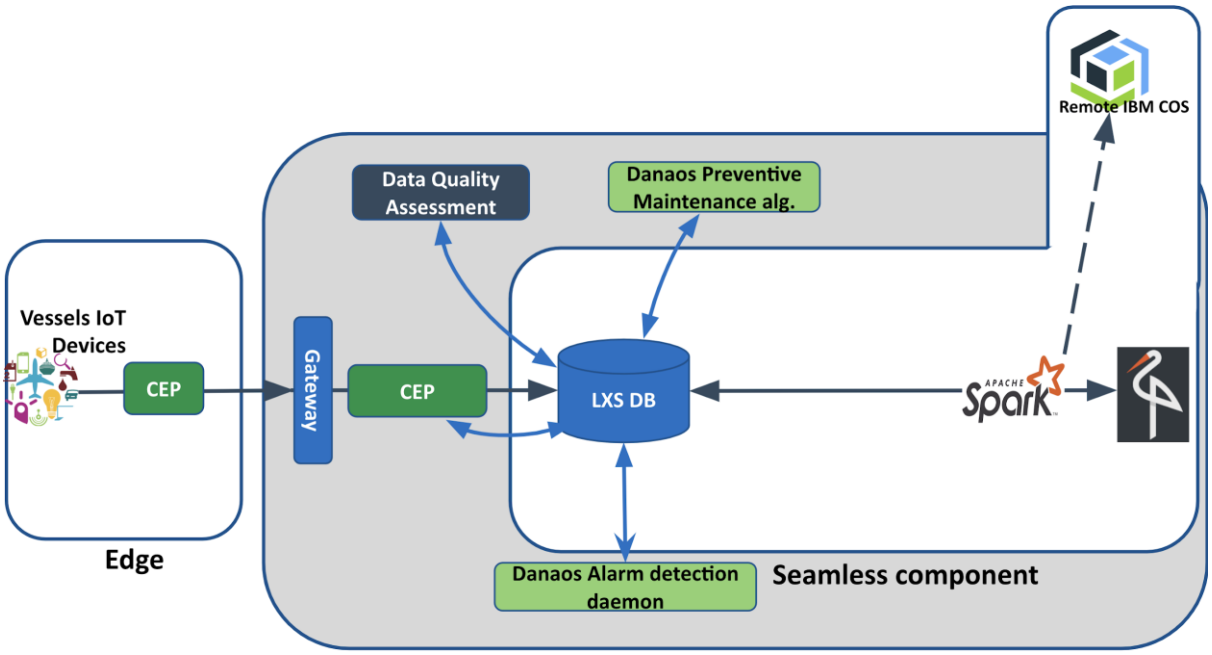


Figure 1 - RSM Pilot's architectural overview

The Edge of this architecture comprises of the IoT devices onboard along with the CEP component. At this point, CEP is used to monitor sensors and identify malfunctions on sensors that cause data-loss. That is, the identification of consecutive null values from a source or erratic values. The latter is identified by complex domain-specific rules. Once such a pattern is identified, an alert is produced.

Past the edge, raw data and alerts flow from the Gateway again to another CEP component. This is used to monitor data and produce alerts whenever the policies of the shipping company are violated. The rules of these policies are stored in LXS DB. In this case, we identify the violation of the Charter Party Agreement, i.e., a deal between the shipping company and the charterer which guarantees a set of thresholds for the consumption of the vessel under specific weather conditions. Once data are processed from the CEP component, if necessary, an alert is produced and data are stored into LXS DB.

The seamless component comprises of the LXS DB, data-skipping and the IBM object storage. We use LXS DB to store fresh data and IBM object storage for historical data. The seamless component allows querying over both data-stores without worrying about data location.

On top of this, the DANAOS preventive maintenance algorithm, uses the seamless component to fetch data and identify malfunction patterns on the provided series of data. If a malfunction pattern is identified an alert is produced and stored into LXS DB. Note here, that this algorithm uses filtered/cleaned data, not raw data. The cleansing is performed by the Data Quality Assessment tool.

Finally, the DANAOS Monitoring platform, uses a daemon that queries LXS DB for alerts and visualizes the produced alerts, whether it is a CEP or a preventive maintenance alert, to the end user. These alerts allow the user to trigger the route adjustment via the DANAOS Routing platform. The route adjustment, changes the speed of the vessel to the next port with respect to the imposed load on the main engine and the weather conditions. We take into account a flexible ETA that can delay at maximum 3 hours the arrival of the vessel to the next port.

Status before the first review of the project

The status up to the first review of the project included the following major completed tasks:

- The use case services have all been deployed on BigDataStack infrastructure and the corporate datasets have been stored in the BigDataStack storage engine.
- Integration of DANAOS platform services with the BigDataStack storage engine and more specifically with the LeanXcale database to store and retrieve the corresponding datasets.
- Containerization of DANAOS platform services using Docker and Docker-compose.
- Deployment of containerized DANAOS Monitoring platform services in BigDataStack infrastructure.
- Implementation and testing of a new predictive maintenance algorithm that efficiently detects a specific malfunction of the main engine of a vessel.
- Integration of the preventive maintenance algorithm in BigDataStack infrastructure so that it can be deployed and managed by the BigDataStack infrastructure management offering.
- Integration of CEP and analytics algorithm with DANAOS platform services. The CEP use is twofold: first we use it to identify data-loss onboard and secondly to produce alerts whenever a policy of the shipping company is violated.
- Inclusion of IBM Object Storage data-skipping technology in the Seamless component that enhances query speed, as demonstrated in a joint presentation of IBM Israel and Danaos Shipping Co. Ltd. in the THINK conference.

Current status and progress

The current progress status includes the following major completed tasks:

- Altering the routing algorithm to consider:
  - Weather conditions
  - Limited main engine load
  - Flexible ETA to the next port (delays up to 3 hours)
- Thorough testing of the routing results with the Operations department
- Integration of the new algorithm in the routing platform
- Containerization of the DANAOS routing platform
- Deployment of the containerized DANAOS routing platform in the BigDataStack infrastructure
- End to end integration testing.

Risks

The main engine, posing the highest risk, consists of various spare parts depending on many parameters. Thus, it is difficult to accurately predict failures. If false alarms occur, the operating costs increase, as ordering of unnecessary parts is not optimal.

3.3. Datasets

DANAOS Shipping Co. Ltd. possesses a complete dataset generated from sensors installed onboard along with operational data and historical data of malfunctions.

In more detail the used datasets in this project include the aforementioned categories for 10 vessels, i.e.,

- Operational data, i.e., telegrams (38K records) produced every 12 hours or upon arrival/departure of a vessel. Rarely updated, e.g., when an error is identified.
- General purpose sensor data (21 different sensors, 29M records, per minute basis).
- Main engine sensor data (100 different sensors, 29Mrecords, per-minute basis).
- Damage reports (25 different cross-head bearing damage incidents on the provided vessels).

Given the data schemas described below, we want to state that the DANAOS datasets do not have any GDPR-related aspect.

TELEGRAMS table structure (14 attributes)

id: Telegram id,

vessel_code: The id of the vessel,

telegram_date: Telegram timestamp (UTC),

type: Telegram type: D:Departure, A:Arrival, N:Noon-telegram,

total_teus: Total Twenty-foot Equivalent Unit (TEU) (# of containers)

total_feus: Total Fourty-foot Equivalent Unit (FEU) (# of containers)

cons_ifo_static_counter: sensor-based measurement TEUs

cons_ifo_static1_counter: sensor-based measurement of FEUs,

draft_aft: Vessel draft at stern (m),

draft_fore: Vessel draft at fore (m),

sea_temperature: Sea temperature (°C),

port_name: Current port name,

next_port: The name of the next port,

eta_next_port: ETA to the next port

VESSEL_DATA table structure (23 attributes)

vessel_code: Vessel id,

datetime: Timestamp of the measurement (UTC),

power: Consumed power (kW),

apparent_wind_speed: Wind-speed (kn),

speed_overground: GPS speed (kn),

stw_long double precision: Speed through water – longitudinal (kn),

stw_trans double precision: Speed through water – transverse (kn),

rpm: rotations per minute of the main shaft,

apparent_wind_angle: Wind angle (0-359.99 degrees),

total_teus: Total Twenty-foot Equivalent Unit (TEU) (# of containers),

total_feus: Total Fourty-foot Equivalent Unit (FEU) (# of containers),

cons_ifo_static_counter: Low-sulfur fuel oil consumption (metric tones),

cons_ifo_static1_counter: High-sulfur fuel oil consumption (metric tones),

port_mid_draft: Vessel draft at port-side (left-side looking to the fore) (m),

stbd_mid_draft: Vessel draft at starboard-side (right-side looking to the fore) (m),

draft_aft: Vessel draft at stern (m),

draft_fore: Vessel draft at fore (m),

stw: Speed through water – calculated by stw_trans and stw_lon (kn),

equivalent_teus: Total number of containers,

mid_draft: Vessel draft at mid-line (m),

trim: The trim of the vessel, calculated by draft_aft and draft_fore,

latitude: The latitude of the vessel's position,

longitude: The longitude of the vessel's position,

MAIN_ENGINE_DATA table structure (102 attributes)

vessel_code: The id of the vessel,

datetime: Timestamp of measurement in UTC,

airCoolerCWInLETPress: Air Cooler Cooling Water Inlet Pressure (Pa)

scavAirFireDetTempNo1: Cyllinder #1 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo2: Cyllinder #2 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo3: Cyllinder #3 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo4: Cyllinder #4 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo5: Cyllinder #5 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo6: Cyllinder #6 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo7: Cyllinder #7 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo8: Cyllinder #8 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo9: Cyllinder #9 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo10: Cyllinder #10 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo11: Cyllinder #11 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo12: Cyllinder #12 Scavenge Air Fire Detection Temperature (°C),

coolerCWinTemp: Air Cooler Cooling Water Inlet Temperature (°C)

cfWInPress: Cooling Fresh Water Inlet Pressure (Pa),

controlAirPress: Control Air Pressure (Pa),

cylLoTemp: Cylinder Lube Oil Temperature (°C)

exhVVSpringAirInPress: Exhaust Valve Spring Air Inlet Pressure (Pa)

foFlow: Fuel Oil Flowrate (lt),

foInPress: Fuel Oil Inlet Pressure (Pa),

foInTemp: Fuel Oil Inlet Temperature (°C),

hfoViscocityHighLow: Heavey Fuel Oil Viscosity High Low (mm2/s)

hpsBearingTemp: HPS Bearing Temperature (°C),

jcfWInTempLow: Jacket Cooling Fresh Water Inlet Temperature Low (°C)

cylExhGasOutTempNo1: Cyllinder #1 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo2: Cyllinder #2 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo3: Cyllinder #3 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo4: Cyllinder #4 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo5: Cyllinder #5 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo6: Cyllinder #6 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo7: Cyllinder #7 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo8: Cyllinder #8 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo9: Cyllinder #9 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo10: Cyllinder #10 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo11: Cyllinder #11 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo12: Cyllinder #12 Exhaust Gas Out Temperature (°C),

cylJCFWOutTempNo1: Cyllinder #1 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo2: Cyllinder #2 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo3: Cyllinder #3 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo4: Cyllinder #4 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo5: Cyllinder #5 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo6: Cyllinder #6 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo7: Cyllinder #7 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo8: Cyllinder #8 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo9: Cyllinder #9 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo10: Cyllinder #10 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo11: Cyllinder #11 Jacket Cooling Fresh Water Outlet Temperature (°C),
cylJCFWOutTempNo12: Cyllinder #12 Jacket Cooling Fresh Water Outlet Temperature (°C),
cylPistonCOOutTempNo1: Cyllinder #1 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo2: Cyllinder #2 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo3: Cyllinder #3 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo4: Cyllinder #4 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo5: Cyllinder #5 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo6: Cyllinder #6 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo7: Cyllinder #7 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo8: Cyllinder #8 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo9: Cyllinder #9 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo10: Cyllinder #10 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo11: Cyllinder #11 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo12: Cyllinder #12 Piston Cooling Outlet Temperature (°C),
tcExhGasInTempNo1: Turbo-Charger #1 Exhaust Gas Inlet Temperature (°C)
tcExhGasInTempNo2: Turbo-Charger #2 Exhaust Gas Inlet Temperature (°C),
tcExhGasInTempNo3: Turbo-Charger #3 Exhaust Gas Inlet Temperature (°C),
tcExhGasInTempNo4: Turbo-Charger #4 Exhaust Gas Inlet Temperature (°C),
tcExhGasOutTempNo1: Turbo-Charger #1 Exhaust Gas Outlet Temperature (°C),
tcExhGasOutTempNo2: Turbo-Charger #2 Exhaust Gas Outlet Temperature (°C),
tcExhGasOutTempNo3: : Turbo-Charger #3 Exhaust Gas Outlet Temperature (°C)
tcExhGasOutTempNo4: Turbo-Charger #4 Exhaust Gas Outlet Temperature (°C)
tcLOInLETPressNo1: Turbo-Charger #1 Lube Oil Inlet Pressure (Pa),
tcLOInLETPressNo2: Turbo-Charger #2 Lube Oil Inlet Pressure (Pa),
tcLOInLETPressNo3: Turbo-Charger #3 Lube Oil Inlet Pressure (Pa),
tcLOInLETPressNo4: Turbo-Charger #4 Lube Oil Inlet Pressure (Pa),
tcLOOutLETTempNo1: Turbo-Charger #1 Lube Oil Outlet Pressure (Pa),
tcLOOutLETTempNo2: Turbo-Charger #2 Lube Oil Outlet Pressure (Pa),
tcLOOutLETTempNo3: Turbo-Charger #3 Lube Oil Outlet Pressure (Pa),
tcLOOutLETTempNo4: Turbo-Charger #4 Lube Oil Outlet Pressure (Pa),
tcRPMNo1: Turbo-Charger #1 RPMs,
tcRPMNo2: Turbo-Charger #2 RPMs,
tcRPMNo3: Turbo-Charger #3 RPMs,

tcRPMNo4: Turbo-Charger #4 RPMs,

orderRPMBridgeLeverer: Order RPM (Bridge Lever)

rpm: Rotations per minute of the main shaft

scavAirInLetPress: Scavenge Air Inlet Pressure (Pa),

scavAirReceiverTemp: Scavenge Air Receiver Temperature (°C),

startAirPress: Starting Air Pressure (Pa),

thrustPadTemp: Thrust Pad Temperature (°C),

mainLOInLetPress: Main Lube Oil Inlet Pressure (Pa),

mainLOInTemp: Main Lube Oil Inlet Temperature (°C)

foTemperature: Fuel Oil Temperature (°C)

foTotVolume: Fuel Oil Total Volume (lt)

power: Consumed power (kW),

scavengeAirPressure: Scavenge Air Pressure (Pa)

torque: Torque of the main shaft (N/m),

coolingWOutLETTempNo1: Turbo-Charger #1 Air Cooler Cooling Water Outlet Temperature (°C),

coolingWOutLETTempNo2: Turbo-Charger #2 Air Cooler Cooling Water Outlet Temperature (°C),

coolingWOutLETTempNo3: Turbo-Charger #3 Air Cooler Cooling Water Outlet Temperature (°C),

coolingWOutLETTempNo4: Turbo-Charger #4 Air Cooler Cooling Water Outlet Temperature (°C),

foVolConsumption: Fuel Oil Consumption (lt/min)

VESSEL_DAMAGES table structure (5 attributes)

vessel_code: The id of the vessel,

defect_type: Type of damage (Main Bearing, Crosshead Bearing, Crankpin Bearing)

defect_details: Short description of damage

date_of_damage: Date of damage

cause_of_damage: Short description for cause of damage

3.4. Use Case Scenarios

3.4.1. Activity 1: Acquire data

The first activity is to define the data sources and acquire the required data. The Data Owner wishes to define the data sources and access protocols. The process is described in the following table.

Step	Description
RSM-A1-01	The Data Owner picks the data source type from a drop down list, e.g., database, FTP server etc. If the selected type is database then the RDBMS should be defined, again from a drop-down list, e.g., postgres, SQL-server etc.
RSM-A1-02	The Data Owner enters in plain text format the required connection attributes, i.e., IP address, username, password, etc.
RSM-A1-03	The system validates the connection.
RSM-A1-04	The Data Owner defines what particular resource is to be accessed. For example, if the selected datasource type is database, selects from a drop down list the preferred tales. If the source is an FTP server, the exact path should be defined.
RSM-A1-05	The system informs the user that the requested data sources are successfully defined.

3.4.2. Activity 2: Select attributes

Another activity is to select a set of attributes from the defined data sources. The process is described in the following table.

Step	Description
RSM-A2-01	The Data Analyst picks from the selected data-sources the contained attributes. For example, if the source is a table in a database, a list of columns is displayed where the Data Analyst picks the preferred columns.
RSM-A2-02	The system informs the user that the requested attributes are successfully defined.

3.4.3. Activity 3: Monitor the selected attributes

The Data Toolkit of BigDataStack provides the ability to monitor the selected attributes. The Data Analyst wishes to activate the monitoring component for the selected set, or sub-set of defined attributes.

Step	Description
RSM-A3-01	The Data Analyst picks from the defined attributes which are to be monitored.
RSM-A3-02	The system informs the user that the requested attributes are successfully defined.
RSM-A3-03	The Data Analyst sets the CEP rules for each variable or combination of variables.
RSM-A3-04	The Data Analyst confirms the rules for CEP monitoring.

RSM-A3-05	The system prompts the user to set the output source where results from CEP monitoring will be put.
RSM-A3-06	The system informs the user that the CEP monitoring is successfully configured.

3.4.4. Activity 4: Deploy the application services

BigDataStack offers a set of functionalities to the Application Engineer in order to deploy existing services into its infrastructure.

Step	Description
RSM-A4-01	The Application Engineer imports a Docker file or a list of files with the related application
RSM-A4-02	The Application Engineer defines environment variables, e.g., database URLs, external APIs, etc.
RSM-A4-03	The Application Engineer can benchmark these custom applications using BigDataStack's Benchmarking tool.
RSM-A4-04	The system informs the user if the deployment was successful or not.

3.4.5. Activity 5: Use an analytics algorithm or deploy a new one

BigDataStack offers a set of well-established analytics algorithms from which a Data Analyst can pick and deploy. Alternatively, the Data Analyst has the ability to inject a custom algorithm that suits the analytical task accordingly.

Step	Description
RSM-A5-01	The Data Analyst picks from a drop-down list the preferred algorithm.
RSM-A5-02.1.1	If the selected algorithm is an existing one, the system prompts the user to define the preferred attributes.
RSM-A5-02.1.2	The Data Analyst picks the preferred attributes.
RSM-A5-02.1.3	The system prompts the user to define the source where the output of the algorithm will be put.
RSM-A5-02.1.4	The Data Analyst sets the output source, e.g., database and table name etc..
RSM-A5-02.2.1	If the selected algorithm is a custom one, the system prompts the user to define the algorithm. That is, programming language, source code etc.
RSM-A5-02.2.2	The Data Analyst sets the requested information.
RSM-A5-02.2.3	The system prompts the user to define the preferred attributes.
RSM-A5-02.2.4	The Data Analyst picks the preferred attributes.
RSM-A5-02.2.5	The system prompts the user to define the source where the output of the algorithm will be put.
RSM-A5-02.2.6	The Data Analyst sets the output source, e.g., database and table name etc..
RSM-A5-03	The system informs the user that the analytics algorithm definition is successfully completed.

3.4.6. Activity 6: Produce alerts based on real-time monitoring information

Once the monitoring framework of BigDataStack is configured and the application services are successfully deployed, an engineer of the Technical department of the shipping company wishes to be informed whenever an alert from CEP monitoring is produced.

Step	Description
RSM-A6-01	The system stores an alert in the defined output source of the CEP.
RSM-A6-02	The alert is accessible from the application services, thus the application prompts the user that an alert is produced.

3.4.7. Activity 7: Adjust the route of the vessel with respect to speed, main engine load and weather conditions.

This activity includes the required steps to adjust the route of a vessel with respect to speed, main engine load and weather conditions. This step takes place after the generation of an alert from the predictive maintenance algorithm. That is, once the Technical, the Supplies and the Operations department are notified by the produced alert of Activity 6, the route adjustment takes place.

Step	Description
RSM-A7-01	The Technical, Supplies and Operations department is notified that a predictive maintenance alert is produced.
RSM-A7-02	The Operations department uses Danaos Routing Service to adjust the route of the vessel with respect to speed, main engine load and the weather conditions.
RSM-A7-03	The system calculates the optimal or close-to optimal route.
RSM-A7-04	The Operations department is notified about the new route.

3.5. Evaluation

In this section we describe the use-case execution scenario, the Criteria Success Factors (CSFs) and the KPIs for each CSF.

3.5.1. Use-case execution scenario

We divide the use-case scenario execution in the following steps denoted by the RSM-S.xx convention:

- RSM-S.1 Identify sensor malfunctions and notify the end-user
- RSM-S.2 Identify business rules violations and notify the end-user
- RSM-S.3 Identify malfunction patterns in the main engine and notify the end-user (preventive maintenance alerts)
- RSM-S.4 Once a preventive maintenance is produced, the user wishes to adjust the route of the vessel.

Below, is the table where each step is matched with the respective Activity as described in the previous section.

User Activity – Use-case execution scenario mapping	
Activity 1,2,3,4,5	RSM-S-1
Activity 1,2,3,4,5	RSM-S-2
Activity 1,2,3,4,5	RSM-S.3
Activity 7	RSM-S-4

For each step we present screenshots from the involved DANAOS services, i.e., DANAOS Monitoring and Routing platforms.

RSM-S.1: Identify sensor malfunctions

In the screens below are available the generated alerts from CEP where a malfunction on the sensor is identified.

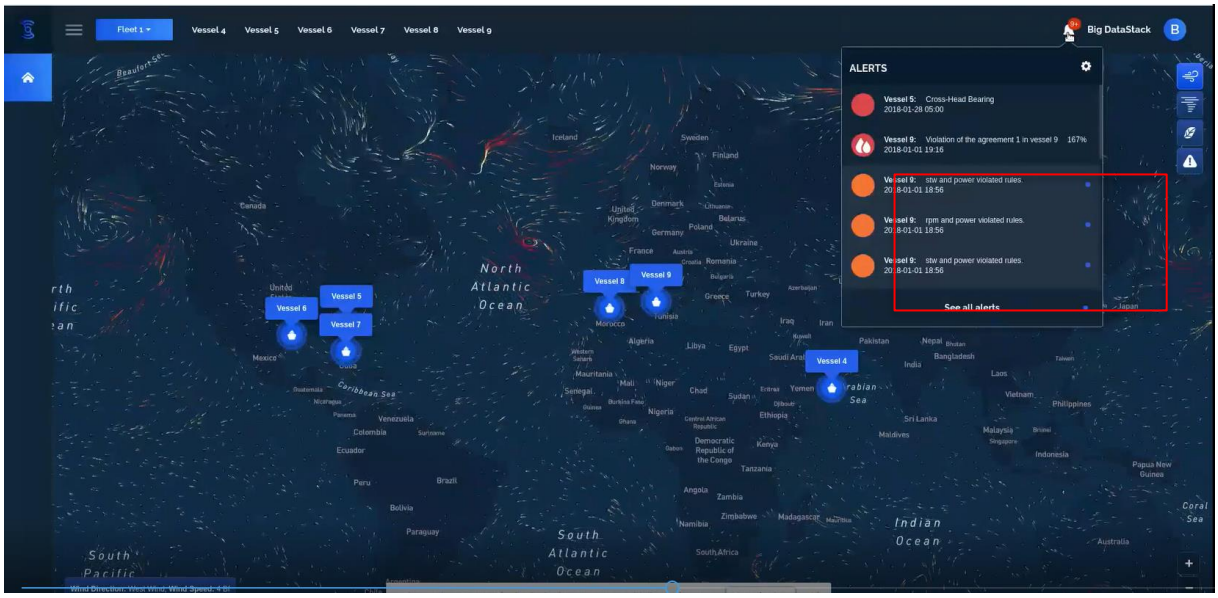


Figure 2 - Produced Sensor Alerts @ Danaos Monitoring Platform

RSM-S.2: Identify business rules violations

In the screens below are available the generated alerts from CEP where a business rule with respect to the fuel consumption (Charter Party Agreement) is violated.

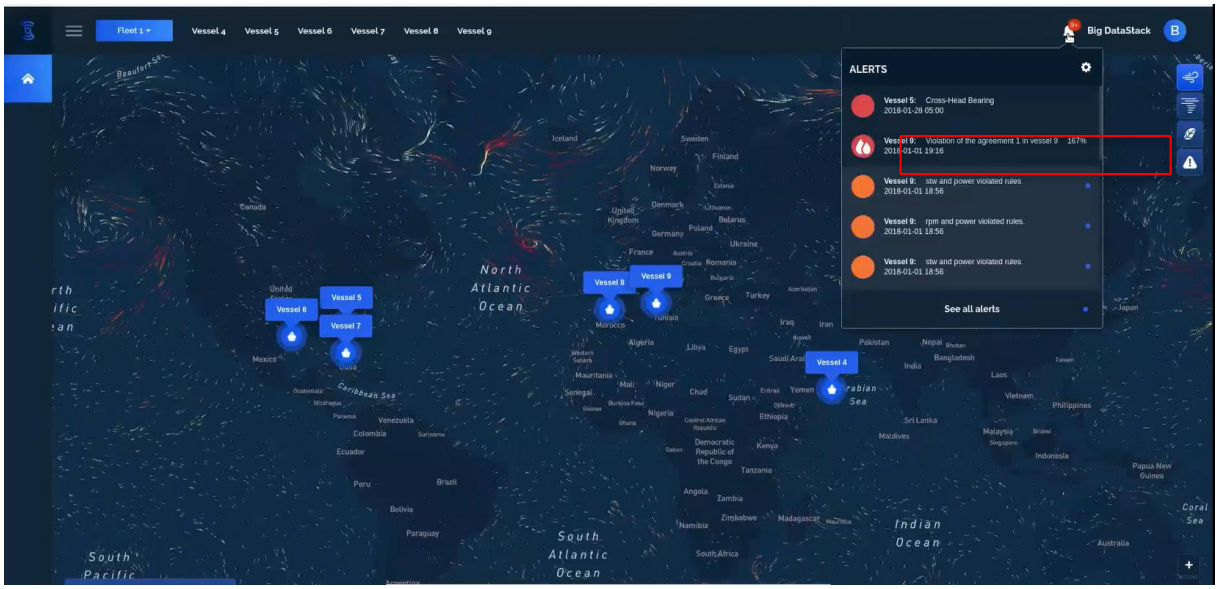


Figure 3 - Produced Business Violation Alerts @ Danaos Monitoring Platform

RSM-S.3: Identify malfunction patterns in the main engine

In the screens below are available the generated alerts from the preventive maintenance algorithm.

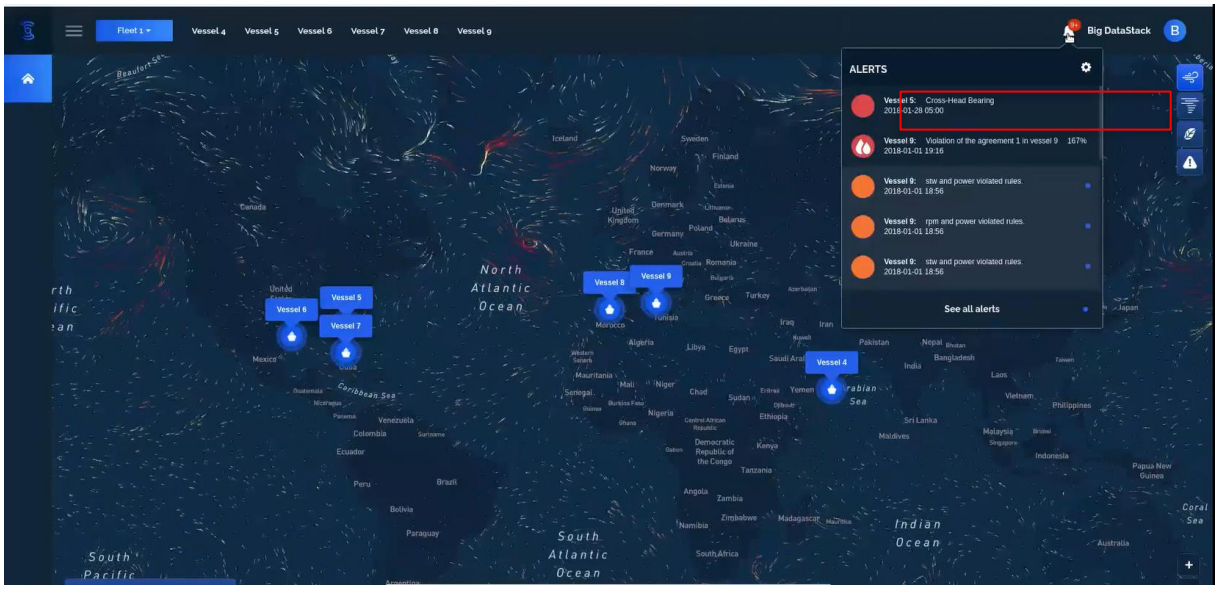


Figure 4 - Produced Preventive Maintenance Alerts @ Danaos Monitoring Platform

Below is a screenshot summing up all the produced alerts for RSM-S.1,2,3.

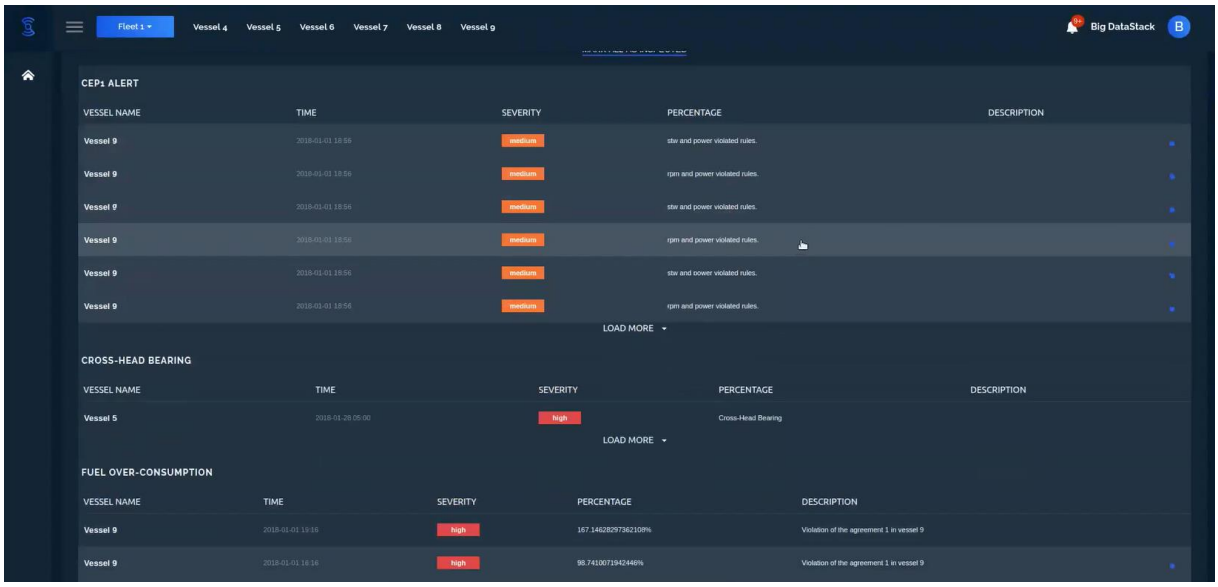


Figure 5 - Overview of Produced Alerts @ Danaos Monitoring Platform

RSM-S.4: Adjust the route of the vessel

Once a preventive maintenance alert is generated, the end-ser wishes to adjust the route of the vessel. In this case, from RSM-S.3 we have a preventive maintenance alert that is produced for Vessel 5 at 2018-01-28 05:00. In this step we will use the DANAOS Routing service to adjust the route of the vessel with respect to the load on the main engine, the weather conditions and a flexible ETA that will delay the vessel at maximum 3 hours.

First we pick the vessel of interest, in this case vessel 5

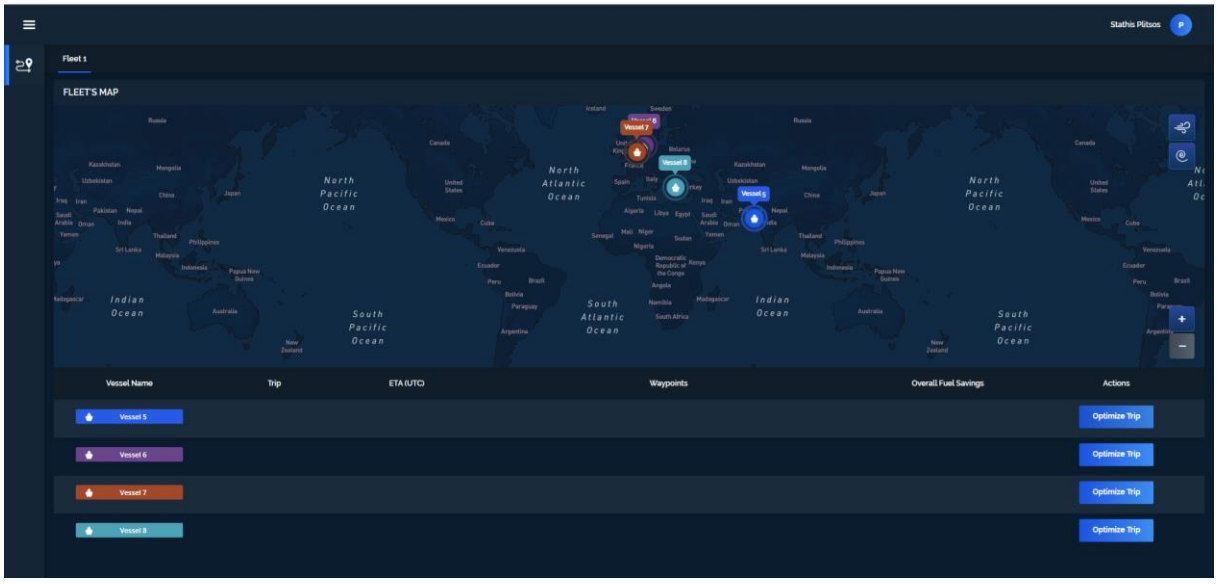


Figure 6 - Overview of Vessels @ Danaos Routing Platform

.

Then we add the required information to start the optimization of the route.

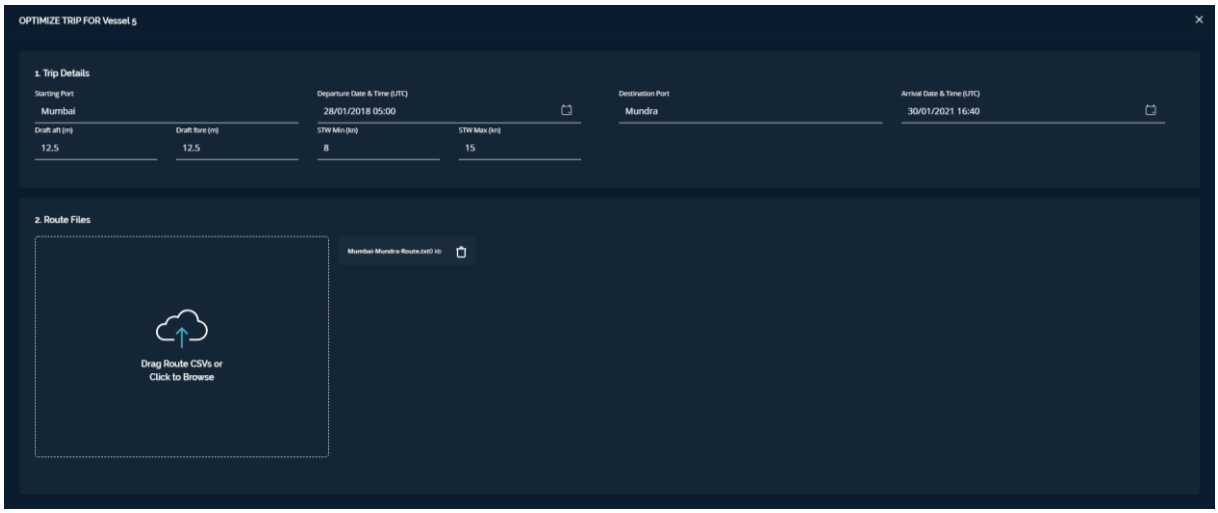


Figure 7 - Route Adjustment Wizard @ Danaos Routing Platform

Then we validate that the provided path is the desirable.

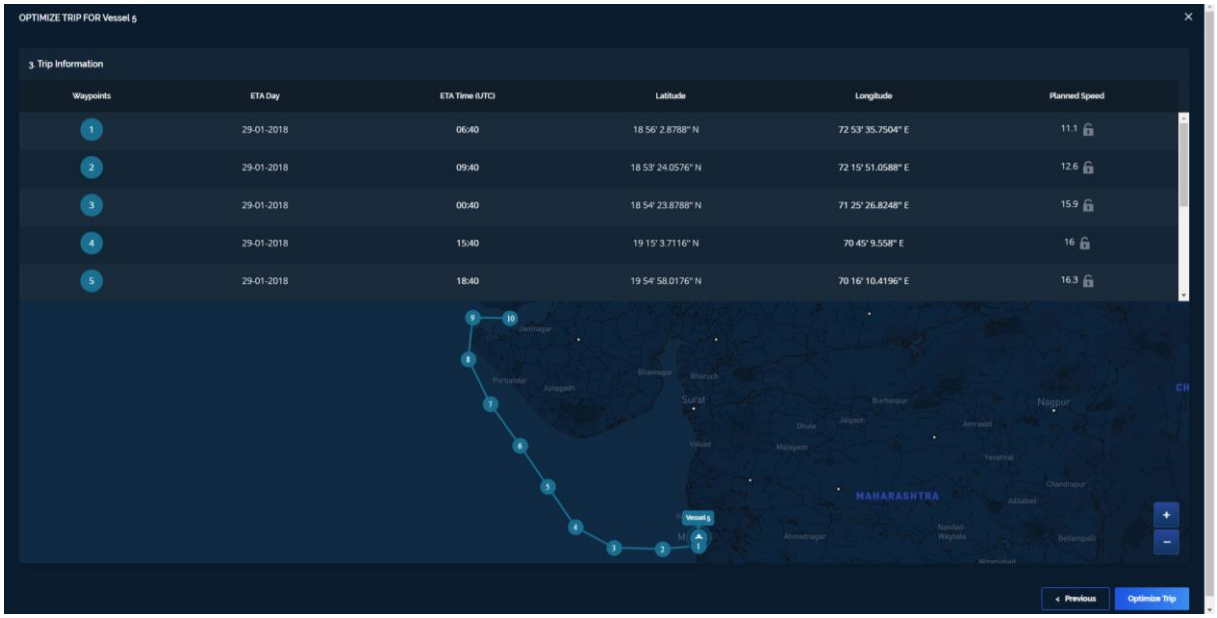


Figure 8 - Imported Route Overview @ Danaos Routing Platform

Once the optimization algorithm completes, we see the results and compare the speeds that the captain of the vessel used with the reduced speeds provided by the algorithm. Last, the expected savings are available. In this case, the expected savings in fuel go up to 5%.

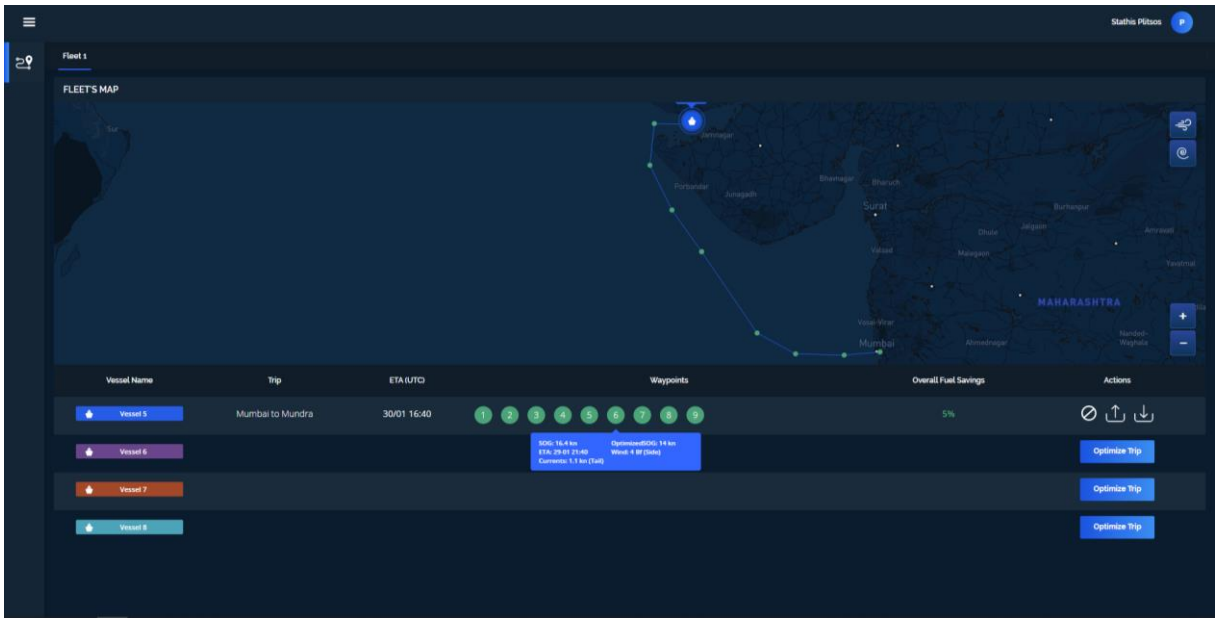


Figure 9 - Optimized Route Overview @ Danaos Routing Platform

3.5.2. Criteria Success Factors & KPIs

Criteria Success Factors (CSFs) for Real-time Ship Management Use Case	
CSF RSM.1	The production of accurate CEP alerts with respect to sensor malfunctions
CSF RSM.2	The production of accurate CEP alerts with respect to business rules violations
CSF RSM.3	The production of accurate Preventive Maintenance alerts
CSF RSM.4	Accuracy of Routing algorithm with main engine load limitations and flexible ETAs
CSF RSM.5	Reduction of consumed fuel
CSF RSM.6	Reduction of emitted CO2
CSF RSM.7	Reduction of imposed load on the main engine

KPI RSM.1	
Title	Sensor Malfunction Alert Accuracy
Objective / CSF Reference	CSF RSM.1
Measures	Number of accurate alerts over total number of produced alerts.
Success Criteria	The ratio of accurate alerts over total alerts exceeds 90%. The results are derived from actual CEP alerts and sensor malfunction reports
Activity Reference	Activity 1,2,3,6

KPI RSM.2	
Title	Business Rule Violation Alert Accuracy
Objective / CSF Reference	CSF RSM.2
Measures	Number of accurate alerts over total number of produced alerts.
Success Criteria	The ratio of accurate alerts over total alerts exceeds 75%. The results are derived from actual CEP alerts, actual vessel data and Charter Party Agreements.
Activity Reference	Activity 1,2,3,6

KPI RSM.3	
Title	Predictive Maintenance Alert Accuracy
Objective / CSF Reference	CSF RSM.3
Measures	Number of accurate alerts over total number of produced alerts.
Success Criteria	The ratio of accurate alerts over total alerts exceeds 95%. The results are derived from predictive maintenance alerts and historical actual malfunctions.
Activity Reference	Activity 1,2,3,4,6

KPI RSM.4	
Title	Routing Algorithm Accuracy
Objective / CSF Reference	CSF RSM.4
Measures	Times that a vessel is estimated to reach the next port with lower load on the main engine and within a 3-hour window of delay, over the total number of cases where such a route adjustment is imposed.
Success Criteria	This ratio goes up to 100%. The results have been derived from executions of the algorithm for each incident of malfunction.
Activity Reference	Activity 7

KPI RSM.5	
Title	Reduced Fuel Consumption
Objective / CSF Reference	CSF RSM.5
Measures	The difference of actual and estimated over the actual fuel consumption.
Success Criteria	This ratio goes up to 6.7%. The results have been derived from executions of the algorithm for each incident of malfunction where the estimated consumption is produced and from actual measurements of consumed fuel on-board.
Activity Reference	Activity 7

KPI RSM.6	
Title	Reduced CO2 Emissions
Objective / CSF Reference	CSF RSM.6
Measures	The difference of actual and estimated over the actual emitted CO2 quantity.
Success Criteria	This ratio goes up to 6.7%. The results have been derived from executions of the algorithm for each incident of malfunction where the estimated consumption is produced and from actual measurements of consumed fuel on-board. By converting the consumed fuel to CO2 using the appropriate factors for each type of fuel, we calculate the emitted CO2.
Activity Reference	Activity 7

KPI RSM.6	
Title	Reduced Load on the Main Engine
Objective / CSF Reference	CSF RSM.7
Measures	The difference of the average actual load over the estimated load given the speed across a voyage over the actual load.
Success Criteria	This ration goes up to 9%. The results have been derived from executions of the algorithm for each incident of malfunction where the estimated speed and load are produced and from the actual main engine load as measured from sensors on-board.
Activity Reference	Activity 7

4. Connected Consumer (CC)

This section provides a description of the Connected Consumer use case scenario from ATOS Worldline.

4.1. Overview and goal

The Connected Consumer use case utilizes the BigDataStack environment to implement and offer a recommender system for the grocery market. All of the data that are used for training the analytic algorithms of the use case are corporate data provided by EROSKI, one of the top food retailers companies in Spain.

What is the scenario's goal?
The final goal of the scenario is to take advantage of the capabilities of BigDataStack to produce recommendations to the customers of the on-line applications of EROSKI.

What is the business objective?
The business objective of this scenario is to create a collaborative-filtering recommender system that produces recommendations of new products to the customers of EROSKI. Thus helping EROSKI to improve the user experience of some of their current applications (e-commerce site, loyalty app, …) and at the same time to improve their customers' loyalty.

Who are the actors in the use case?
The actors involved in the first scenario of the CC scenario have been extracted from D2.1. Concretely, these are summarized in the following table:

Table 2 - Connected Consumer Actors

Id	Name	Description
ROL-01	Data Owner	BigDataStack offers a unified Gateway to obtain both streaming and stored data from data owners and store them in its underlying storage infrastructure that supports SQL and NoSQL data stores. In the CC the Data Owner is EROSKI.
ROL-02	Data Scientist	BigDataStack offers the Data Toolkit to enable data scientists both to easily ingest their analytics tasks by utilizing a declarative paradigm, and to specify their preferences and constraints to be exploited during the dimensioning phase regarding the data services that will be used (for example preferences for the data cleaning service)
ROL-03	Business Analysts	BigDataStack offers the Process Modelling Framework allowing business users to define their functionality-based business processes (through declaratively-defined models) and optimize them based on the outcomes of process analytics that will be triggered by BigDataStack.

ROL-04	Application Engineers and Application Service Owners	BigDataStack offers the Application Dimensioning Workbench to enable application owners and engineers to experiment with their applications and dimension it in terms of its data needs and data-related properties
ROL-CC-1	Grocery Consumers	BigDataStack offers a performant environment enabling the achievement of the goals set by the use case in terms of resource provisioning and execution of analytics based on given service level objectives. Based on that, the customers of EROSKI, are end-users of any of the applications of EROSKI (e.g. someone who is browsing in the e-commerce site of EROSKI) and will obtain the recommendation results following the successful execution of the analytics pipelines on BigDataStack infrastructure.

What are the key activities these actors are involved in?
The CC use case can be broken down into the following main activities:

1. **Definition of the analytics for the recommender.** This activity aims at defining the business processes and the main analytical tasks that the recommender needs. The main actors in this activity are the Business Analyst and the Data Scientist.

2. **Deployment of the application services.** This activity aims at testing the capability to deploy on top of BigDataStack the application services of the recommender system. This activity applies to several actors: the business analyst, the data scientist and the application engineer. The former sets the time constraint for the whole process of recommendations. The data scientist calculates the service level objectives (SLOs) needed for each of the application services that compose the application. The application engineer tests several configurations before proceeding to do the final deployment.

3. **Re-Deployment of the application services.** This activity demonstrates the capability of BigDataStack to make run-time adaptations in the configuration of the application services due to changes in the incoming data. The activity applies to the application engineer and the data owner.

4. **Visualize recommendations.** The Data scientist wants to know which recommendations the system will propose for a given user. Main actors in this activity are the data scientist and the business analyst.

5. **Provide recommendations.** The External Applications need to provide recommendations to their users. E.g. a new App oriented to strengthen the loyalty of the EROSKI end-users wants to have the feature of suggesting new products to its users. Main actor in this activity is an end user of the client applications of the recommender.

6. **Collect feedback.** The Data scientist wants to have further information about the success or failure of the recommendations provided by the recommender. For this reason, the system is prepared for collecting feed-back from the external systems about the usage of the recommendations. Main actors in this activity are the data-owner and the data scientist. This activity aims at testing the capabilities of BigDataStack to ingest data through the Gateway.

4.2. Pilot description

The pilot that has been implemented relies on the LeanXcale data store of BigDataStack. Before running the pilot, 2 years of history have been loaded into LeanXcale.

The pilot that has been implemented contains 3 different sets of application and data services:

i. web services access layer
ii. calculate recommendations – batch model
iii. model update – real-time analytics

The first set of application services, Web Services access layer, is the entry-door for the external applications to the recommender system.

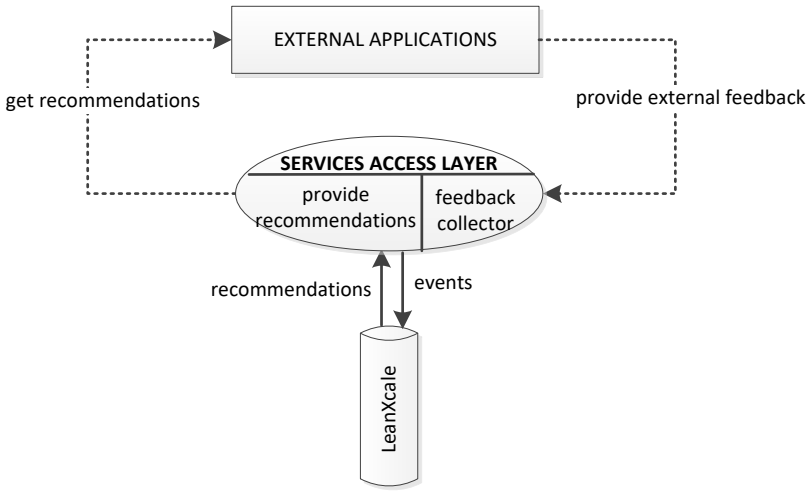


Figure 10 - Web Services access layer

It consists of two different services:

- *Feedback collector service*, external applications provide information about the interactions between a user and a product by means of this service.
- *Provide recommendations service*, external applications retrieve the recommendations calculated by the system with this service. Consumers of these services will be the client applications that need to show recommended products to its users. Multi-channel is provided by the system.

The second set of application services, named 'calculate recommendations', aims at running the analytic flow to calculate the products that will be recommended to the user. These set of processes calculate the Recommendations matrix for each customer segment in batch.

Three different application services have been implemented in the analytic flow:

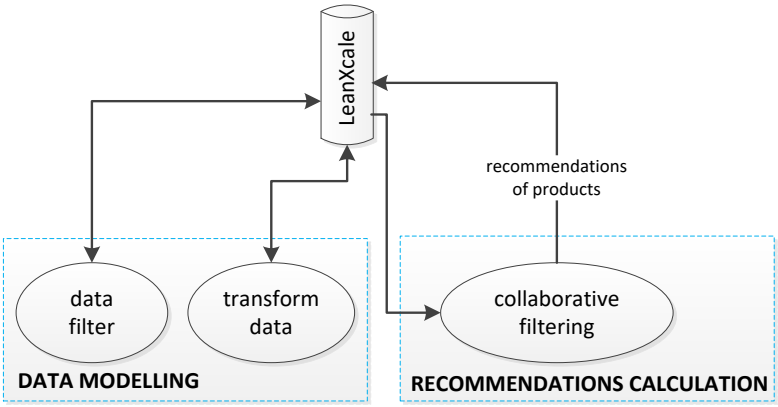


Figure 11 - Analytic flow

Concretely,

- Data filter. In charge of selecting the necessary data to run the model.
- Transform data. In charge of calculating the input matrix needed by the model.
- Collaborative filtering. Execution of the algorithm.

The third set of application services has been called 'model update'. The goal is to update the batch-model produced periodically in real time.

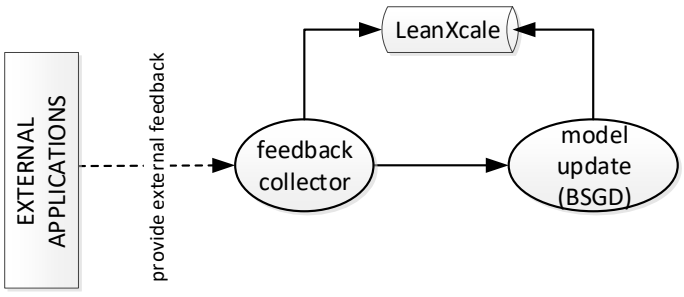


Figure 12 - Model update process

It is composed by 2 processes:

- Feedback Collector service (described above within the first set of application services)
- Model update, specialised in updating the recommendations matrix (BSGD optimization method has been implemented in this part)

Many retailers nowadays are offering recommendations of products and promotions to their customers. However, these recommendations are sometimes calculated weeks in advance to a certain promotional campaign. To make matters worse, many times the calculation of these recommendations is done using outdated customer segments. In this context, the challenge is to be able to speed up the process of calculation of recommendations so that users can be offered personalized suggestions of products. In order to fulfil this objective, the system implemented aims at providing the capability to make in-store consumer-tailored predictions to its users. As soon as an interaction between a user and a product arrives to the system, the analytic flow needed to calculate the recommendations for a certain user is triggered. In this way, the system is offering fresh recommendations to its users.

4.3. Product Recommendation Models

At the core of this use-case is the recommendation calculation. In effect, for a user (with a given purchase history), the goal is to produce a ranked list of items (products) to recommend to them, where the top-ranked items are the most suitable for them personally. More precisely, grocery recommendation is a special case of the more general field of item recommendation, where products are not bought alone but instead bought in groups (i.e. shopping baskets). There are a range of different approaches that can be used to generate such recommendations for a user, with varying degrees of effectiveness. In this section we provide a brief overview of the literature in grocery recommendation, introduce a new open source framework for grocery recommendation we developed within BigDataStack called BetaRecsys (https://beta-recsys.readthedocs.io/en/latest/), and then summarize the different recommendation models that we experiment within the context of this use-case. Much of this section is adapted from published work from BigDataStack, we include the source papers in Appendices B-G.

4.3.1. Related Works in Grocery Recommendation

Recommender Systems: Recommender systems have attracted much interest in both academia and industry. Most of the commonly used techniques in recommender systems can be classified into two main categories, i.e. the classical collaborative filtering methods and the neural network (NN)-based methods. Early work on recommender systems mainly focused on modelling explicit feedback about items (i.e. rating scores) from users based on the matrix factorization (MF) algorithms [**Error! Reference source not found.**],[**Error! Reference source not found.**],[**Error! Reference source not found.**]. A MF algorithm encodes the user-item explicit feedback as a rating matrix and predicts the rating scores of unseen items for users by completing the matrix. Many sophisticated matrix factorization techniques, such as time Singular Value Decomposition [**Error! Reference source not found.**], implicit factorization machines (FM) [**Error! Reference source not found.**] and context-aware FM [**Error! Reference source not found.**], have been proposed to address both classical item recommendation as well as more advanced scenarios, such as time-aware [**Error! Reference source not found.**], implicit feedback [**Error! Reference source not found.**] and context-aware recommendation [**Error! Reference source not found.**]. However, these classical methods suffer from the

matrix sparsity problem and cannot capture non-linear relationships between users and items.
To address these issues, recent deep neural network-based recommendation methods [**Error! Reference source not found.**], [**Error! Reference source not found.**], [**Error! Reference source not found.**], [**Error! Reference source not found.**], [**Error! Reference source not found.**] were proposed to learn the embeddings of users and items by neural networks from user-item interactions, as well as from various types of contextual information, resulting in significant performance gains. For instance, the Neural Collaborative Filtering [**Error! Reference source not found.**] model is a general framework that integrates deep neural networks into matrix factorization approaches using implicit feedback (e.g. purchase history and click behaviour). Meanwhile, Li et al. proposed a collaborative variational auto-encoder [**Error! Reference source not found.**] that learns deep latent representations from content data in an unsupervised manner and also learns implicit relationships between items and users from both content and ratings.

Grocery Recommendation: From our use-case perspective, grocery product recommendation can be formulated as a prediction task, where the aim is to predict the items that a user will purchase next, given a sequence of shopping baskets (each containing multiple items) that they have previously bought. Some variants of the early MF-based methods can also be applied to this task, e.g. Factorizing Personalized Markov Chains (FPMC) [**Error! Reference source not found.**] can model sequential behaviours between every two adjacent baskets by conducting a tensor factorization over the transition cube. Recurrent Neural Networks (RNNs) are also another category of methods that can capture global sequential features among baskets [**Error! Reference source not found.**]. However, RNNs suffer from the problem of high computational costs [**Error! Reference source not found.**]. On the other hand, skip-gram-based methods have been shown to be both effective and reasonably efficient at addressing the grocery recommendation task [**Error! Reference source not found.**] [**Error! Reference source not found.**]. For example, the Triple2vec [**Error! Reference source not found.**] model is both an effective and scalable solution for grocery recommendation. It first samples a large set of triples from the historical baskets, then uses the Skip-gram model to predict the occurrence probability of each triple. However, this method is a point estimate solution that only represents entities as deterministic embedding vectors and it cannot integrate the rich intrinsic feature of products, leaving much space for improving the expressive ability of the embeddings.

Integrating Side Information within Recommender Systems: With the increased availability of side information for a range of recommendation tasks, integrating such information into recommender systems has been widely studied [**Error! Reference source not found.**] [**Error! Reference source not found.**] [**Error! Reference source not found.**]. Indeed, there are variants of the MF methods, such as hierarchical Bayesian matrix factorization [**Error! Reference source not found.**], which incorporates side information into traditional MF approaches. For example, the HIRE [**Error! Reference source not found.**] model is a recommendation model that uses weighted matrix factorization to obtain users' and items' representations that encode both the flat and hierarchical side information, thereby improving the recommendation performance. Other works have examined the integration of side information for deep neural networks, such as the stacked denoising auto-encoder [**Error! Reference source not found.**] and the marginalized denoising auto-encoder [**Error!**

Reference source not found.]. More recently, many recommendation models have explored using the Variational auto-encoders (VAEs) [**Error! Reference source not found.**], [**Error! Reference source not found.**], [**Error! Reference source not found.**], [**Error! Reference source not found.**] to jointly encode user ratings and side information when training, to overcome the (often) high-dimensionality of side information. However, most of these methods only consider a single type of item feature, for instance, only using a bag-of-words/one-hot categorical feature vector [**Error! Reference source not found.**], [**Error! Reference source not found.**] or a product category [**Error! Reference source not found.**], [**Error! Reference source not found.**].

4.3.2. BetaRecsys

As can be seen from the above literature review, there are a wide range of approaches for product recommendation that could be used as the basis for the use-case. We reported early work in implementing and evaluating different approaches for grocery recommendation previously in D3.2 Section 10. However, during Y3, we built-on this initial work to produce a new open source framework, called BetaRecsys (https://beta-recsys.readthedocs.io/en/latest) that aggregates the different grocery recommendation algorithms we implemented and developed during BigDataStack for use by academia and industry. We summarize the motivation and functionality provided by BetaRecsys in the remainder of this section, however further details can be found on the website and associated paper provided in Appendix D.

Motivation and Technology Gap: As discussed above, rapid progress has been made over the last 5 years, significantly advancing the state-of-the-art in recommendation tasks. However, partially as a result of this intensive and rapid progress, there are now many barriers-to-entry in the recommendation field for new researchers and industry practitioners, as well as increasing challenges when comparing different works from the literature. In particular, current challenges include:

1) Algorithm implementations are fragmented across different code repositories and often do not function out-of-the-box.
2) Reported performances across works are often not comparable even when reported on the same dataset and with the same metrics, due to different data pre-processing and splitting techniques being employed.
3) The usage of different evaluation toolkits is problematic as subtle differences in metric implementations lead to different reported performances.
4) There is highly inconsistent use of model tuning strategies (most notably the omission of baseline tuning), leading to unfair comparisons.

Indeed, recent works [**Error! Reference source not found.**] [**Error! Reference source not found.**], [**Error! Reference source not found.**], [**Error! Reference source not found.**] have demonstrated that, without properly tuning model hyper-parameters, existing state-of-the-art deep learning baselines cannot even consistently outperform non-neural linear ranking models. Hence, there is a clear need for platforms that provide a common recommendation methodology and pipeline, enabling model comparison across datasets, metrics, and baselines in a sound and fair manner. To date, a range of platforms have been proposed and developed, including

Spotlight[1], Microsoft-Recommenders[2], DeepRec [**Error! Reference source not found.**], OpenRec [**Error! Reference source not found.**] and Cornac[3]. However, while these works have greatly aided in making the field more accessible, they are also to blame for some of the challenges summarized earlier, as these platforms are not strongly opinionated about the recommendation methodology and pipeline. For instance, the popular Microsoft-Recommenders platform provides no common framework, resulting in each contained model implementing their own data preparation process. By leaving these aspects at the user's discretion and providing little in the way of guidance/best practices to follow, means that poor and/or inconsistent methodological choices are sadly commonplace. Hence, we proposed a new platform named BetaRecsys which is a unified platform for building, evaluating and tuning automatic recommender systems.

BetaRecsys Architecture: BetaRecsys provides an end-to-end workflow for researchers and practitioners to build their new models or use the built-in models, extending the popular Pytorch format. It also provides a standardized way to configure model training and evaluate the resultant models under a unified framework that is fully compatible with BigDataStack. Figure 13 provides an overview of the platform architecture, which highlights the major components of the platform. We summarize each of the four main components of the framework (Prepare Data, Build Models, Tune & Train Models, and Evaluate Models) below.

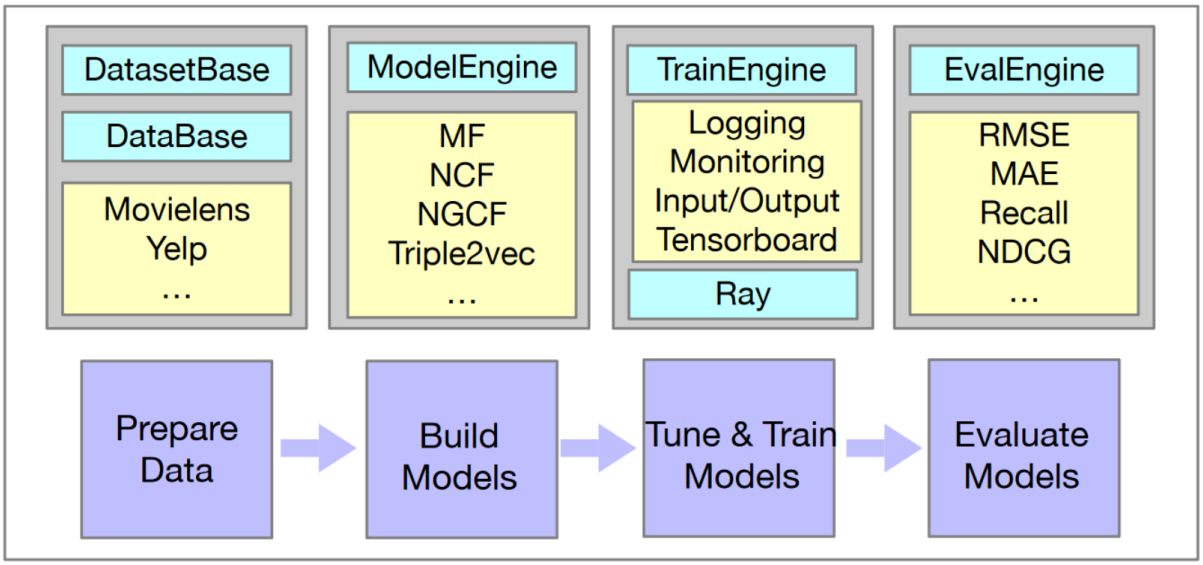


Figure 13 - BetaRecsys Architecture

Prepare Data: To make the workflow efficient, we implement two key reusable components for preparing training data for different recommender models. The DatasetBase component provides unified interfaces for processing the raw dataset into interactions and splitting it using common strategies (e.g. leave-one-out, random split or temporal split) into

[1] https://github.com/maciejkula/spotlight
[2] https://github.com/microsoft/recommenders
[3] https://github.com/PreferredAI/cornac

training/validation/testing sets. Meanwhile the DataBase provides the tools to further convert the resultant data sets into usable data structures (e.g. tensors with <user, item, rating> or <user, positive_item, negative_item(s)> structures), dependant on the requirements/supported features of the target model. Out-of-the-box we support a number of commonly used recommendation datasets, including those used to evaluate this use-case as described later in Section 4.4.

Build Models: Our framework provides a model engine (i.e. represented by the class ModelEngine) for conveniently building a Pytorch recommender model in a unified manner. In particular, it provides a unified implementation for saving and loading models, specifying the compute device, optimizer and loss (e.g. BPR loss [**Error! Reference source not found.**] or BCE loss [**Error! Reference source not found.**]) to use during training. Out-of-the-box, 9 recommendation models are provided, including classic baselines like MF [**Error! Reference source not found.**], as well as more advanced neural models such as NCF [**Error! Reference source not found.**], NGCF [**Error! Reference source not found.**] and Triple2vec [**Error! Reference source not found.**].

Train & Tune Models: The TrainEngine component provides unified mechanisms to manage the end-to-end training process. This encompasses: loading the configuration; loading the data; training each epoch; calculating validation performance; checkpointing models; testing early stopping criteria; and calculating the final test performance. The TrainEngine also supports monitoring/visualizing the training progress in real time, including resource consumption and training metrics (such as the training loss and evaluation performance on both the validation and testing sets) of a deployed model via Tensorboard. It can also expose these real-time metrics to a Prometheus time-series data store via an in-built Prometheus exporter, enabling programmatic access to the training state by the Realization Engine within BigDataStack (see D3.3 Section 6). To support easier and faster hyperparameter tuning for each model, we also integrate the Ray framework[4], which is a Python library for model training at scale. This enables the distribution of model training/tuning across multiple gpus and/or compute nodes.

Evaluate Performance: Three categories of commonly used evaluation metrics for recommender systems are included in this platform, namely rating metrics, ranking metrics and classification metrics. For rating metrics, we use Root Mean Square Error (RMSE), R Squared (R2) and Mean Average Error (MAE) to measure the effectiveness. For ranking metrics, we include Recall, Precision, Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP) to measure performance of ranking lists. Model evaluation using built-in classification metrics like Area-Under-Curve (AUC) and Logistic loss are also supported. To accelerate the evaluation process, the metric implementations are multi-threaded.

In summary, BetaRecsys is a new open source framework for performing grocery recommendation (and other recommendation tasks) that was developed within BigDataStack to support this use case. The framework has been released as open source and is actively being built upon in other projects (including the Flagship EC H2020 Infinitech project).

- https://github.com/beta-team/beta-recsys

[4] https://github.com/ray-project/ray

4.3.3. Recommender Models Used

To evaluate recommendation performance for this use-case specifically, we experiment with a range or recommendation models either included within BetaRecsys and/or built by the use-case partner. We list these recommendation models below:

- **TopPop**: A naive method that recommends the top-k frequent items in the training set to each user in the test set.
- **BPR** [**Error! Reference source not found.**]: The Bayesian Personalized Ranking (BPR) method is a classical method for learning personalized ranking from implicit feedback.
- **NeuCF** [**Error! Reference source not found.**]: NeuCF is a deep neural network-based collaborative filtering model that uses a multi-layer perceptron internally to learn non-linear user-item interactions.
- **NGCF** [**Error! Reference source not found.**]: This is a new recommendation framework that uses the graph neural network to integrate the user-item interactions into the embedding process.
- **Triple2vec** [**Error! Reference source not found.**]: A representation model that learns the embeddings of users and items to recover product co-occurrences both within a basket and across baskets from the same user based on the Skip-gram model.
- **cVAE** [**Error! Reference source not found.**]: The cVAE model is a recommendation model that uses two Variational Autoencoders to simultaneously recover the user ratings and side information of items.
- **HIRE** [**Error! Reference source not found.**]: The HIRE model is a recommendation model that uses the flat and hierarchical side information to improve the performance of recommendation. In this paper, for fair comparison, we only use its variant that eliminates the contribution of the hierarchical side information of users and items.
- **VBCAR**: This is a new grocery recommendation model developed within BigDataStack, referred to as the Variational Bayesian Context-Aware Representation model for grocery recommendation, which is a novel variational Bayesian model that learns distributional representations of users and items by leveraging basket context information from historical interactions. See Appendix B for the associated research paper.
- **VBCAR-S**: This is an extension of VBCAR that integrates side information to enhance recommendation performance. See Appendix C for the associated research paper.

It is worth noting that a further two models have been developed (T-VBR and GCN) that aim to more effectively capture temporal dynamics and enhance the user/item embeddings, respectively. However, at the time of writing this deliverable these works are still under review at top-tier recommendation conferences and journals, and hence we do not report the associated results here. We do however include the associated papers under review for reference later in Appendices E and F.

4.4. Datasets

For this use-case the primary dataset used for evaluation is real shopping data provided by the end-user EROSKI, which is described in Section 4.4.1. However, as grocery recommendation has been previously studied in the literature, there are a number of publicly available datasets that can also be used for evaluation, which we summarize in Section 4.4.2.

4.4.1. EROSKI Grocery Transactions Dataset

The primary dataset that is used for training the analytic algorithms of the use case is comprised of corporate data provided by EROSKI, one of the top food retailer companies in Spain. The dataset contains information about EROSKI clients. However, GDPR requirements have been taken into account. First, the full version of this dataset is only available for access within and is restricted to processing upon data secure Openshift testbeds that have ISO270001 certification. Furthermore, the data has been subject to the following anonymisation process before being made available to the consortium:

- The only data that could be used to uniquely identify a person is the field "ID_CLIENTE". ID_CLIENTE is an internal identifier of the database of EROSKI that is not known by the customers, i.e. only a person with access to the database of EROSKI could identify the customer from ID_CLIENTE.
- ID_CLIENTE has been encrypted by EROSKI with an SHA-1 algorithm. Encryption has been done before providing the dataset to the BigDataStack consortium. A SHA-1 (168 bits) algorithm has been used for encryption of ID_CLIENTE.
- For each ID_CLIENTE, SHA-1 has been applied to "string_1"+ID_CLIENTE+"string_2". String_1 and string_2 are alphanumeric that contain capital and non-capital letters, numbers and special characters. These 2 values are only known by EROSKI.

In terms of the dataset itself, the raw data is comprised of four main tables that together provide information about grocery purchases made to the EROSKI platform. This is comprised of 1.8 million categorized grocery products. Customer data for around 5 million EROSKI users is stored in a clients table. Purchases are then stored as 'tickets', where a ticket specifies a customer and one or more products bought together in a basket. There are a total of around 3.6 trillion tickets. Tickets are also connected to a place of purchase (centre/shop/store). Details of the data structures for products, clients, tickets and centers can be found in Appendix A.

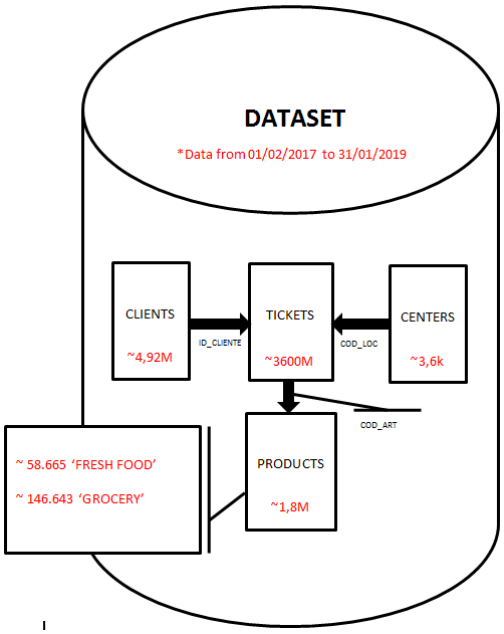


Figure 14 - EROSKI Dataset Structure

There are two versions of this dataset that are used for evaluation, referred to as EROSKI-Full and EROSKI-Sample. EROSKI-Full is, as the name

suggests, the entire EROSKI dataset described above. Meanwhile, EROSKI-Sample is a smaller subset of the dataset that is more heavily anonymized and filtered, such that it can be used on non-ISO270001 certified testbeds. This version contains no client information (beyond the anonymized identifier discussed above), limited information about the purchased products (broad product category information) and a random sample of tickets for the clients contained. EROSKI-Sample covers around 5,000 customers, 10,000 items and 125,000 tickets.

4.4.2. Secondary Public Datasets

In addition to the above private dataset, there are also several other grocery transaction datasets that can be used to evaluate the effectiveness of grocery recommendation systems. These datasets differ from the primary EROSKI dataset in two important ways. First, the main disadvantage of these datasets are they are smaller and provide significantly less contextual metadata about the users and items, as they are designed for public release and hence have been heavily sanitized to avoid releasing personal data or secrets about the inner workings of the associated platforms. Second, on the other hand, these datasets have the significant advantage that they do not carry the same security requirements, and can hence be deployed on any testbed, which will be important later when we discuss the training of the recommender models later in . In particular, we identified two main alternative datasets that we use later for evaluation. Each of these datasets provides information about purchase transactions connecting a user identifier to a set or sequence of products they bought together, and may include some additional (side) information about the products:

- **Instacart**[5]: The Instacart dataset is comprised of files that describe customers' orders made to the Instacart platform over time. The dataset contains over 3 million grocery orders (transactions) from more than 200,000 customers. For each customer, between 4 and 100 orders are included, along with sequence and date information. This dataset contains three columns of product side information, where two types (`aisle_id' and `department_id') are categorical data and one (`product name') is textual data.
- **Dunnhumby**[6]: This dataset contains transactions made by households over a two-year period for 2,500 households. These households are those who frequently purchase goods from Dunnhumby. This dataset provides four types of product side information, where two types (Manufacturer and Department) are regarded as categorical data and two types (Commodity Description and Sub Commodity Description) are textual data.

The statistics of these datasets are shown in Table 3 below. Items are the products in each dataset, while baskets represent the groups of items purchased together (these are equivalent to the tickets in the EROSKI data). For our later experiments, we will divide these datasets into training, validation and test sets, comprised of individual item purchases, referred to as interactions.

[5] https://www.kaggle.com/c/instacart-market-basket-analysis/data
[6] http://www.dunnhumby.com/careers/engineering/sourcefiles

Table 3 - Secondary Grocery Recommendation Dataset Statistics

Dataset	#Users	#Items	#Baskets	#Interactions (train/valid/test)
Instacart (25%)	46,850	9,174	527,431	1,429,811/2,969,208/2,968,120
Instacart (100%)	204,784	49,550	2,007,649	20,293,801/21,998,255/22,005,746
Dunnhumby	2,497	61,600	113, 831	715,007/359,776/385,778

4.5. Key Performance Indicators

Connected Consumer takes advantage of the platform BigDataStack to achieve its objectives and improve the user experience at the EROSKI ecosystem of applications. The features offered by BigDataStack facilitate the success of the recommender. The evaluation of the system has been made through the definition of two different types of objectives. In particular, a group of Key Performance Indicators (KPIs) have been identified to evaluate the degree of success or failure of the system in a more quantitatively. A KPI is defined by one or more metrics. In addition, success criteria are defined by a target or threshold value for the respective KPI value.

The evaluation of the Connected Consumer use case and calculations of the KPIs described below has taken place on an ISO-27001 compliant environment, as required by EROSKI.

KPI CC.1	
Title	Production of accurate recommendations
Measures	Mean Average Precision, Recall at k and Normalize Discounted Cumulative Gain measure how many of the first k recommended items are in the set of true relevant items averaged across all users. *
Success Criteria	. MAP: 0,05 Recall at k: 0,08 NDCG: 0,3 The result has been derived from experimentation with real data provided by EROSKI.
*MAP and NDCG take into account the order of the first recommended products meaning that it can penalize if incorrect recommendations are at the top k. Recall at k does not take into account the order.	

KPI CC.2	
Title	Reduction in time required for data analytics

Measures	Reduction in time taken to refresh the recommendation for a given user (given an interaction of the user with the system)
Success Criteria	Migrating from a daily batch model to a real time one has derived into that time has decreased in more than 99%. Derived from experimentation with real data provided by EROSKI.

KPI CC.3	
Title	Increase the number of consumers receiving personalized recommendations
Measures	Percentage of Increase in the number customers that receive personalized recommendations
Success Criteria	Having analyzed the EROSKI dataset of EROSKI, the number of users that will receive personalized promotions will be over 50% of the customers. Derived from experimentation with real data provided by EROSKI.

Notably, KPI CC.2 and CC.3 are evaluated through acceptance testing and quantification of usage by the end-user. However, KPI CC.1 can also be evaluated experimentally using the datasets described previously in Section 4.4. As such, in the next section we report recommendation performance results over these datasets using the models summarized in Section 4.3.

4.6. Grocery Recommendation Performance

In this section we report the performance of the 9 grocery recommendation models listed in Section 4.3.3 for both the EROSKI-Sample and secondary datasets (Instacart and Dunnhumby). The goal is to demonstrate grocery recommendation performance that meets KPI CC.1. In particular, we summarize the setup for training these models in Section 4.6.1, and then report the performances achieved in Section 4.6.2.

4.6.1. Experimental Setup

Data Splitting: For model evaluation, different from the majority of prior works [**Error! Reference source not found.**] [**Error! Reference source not found.**] that use the leave-one-out split strategy, we split the dataset using the temporal split strategy, where all the baskets of the datasets are split into training (80%) and test (20%) sets according to the temporal order of the baskets, and the last 20% of baskets in the training set are also used as the validation set for tuning the hyper-parameters and iterations. We use this strategy as it is more realistic [**Error! Reference source not found.**] [**Error! Reference source not found.**] than the more commonly reported leave-one-out data splitting strategy, as evidence from the future can `leak' into the learned model under leave-one-out. We treat product recommendation as an item ranking task (for each user). We report four main metrics, namely NDCG@k, Recall@k and MAP@k, which are all standard ranking evaluation metrics and are

widely used for evaluating recommender systems [**Error! Reference source not found.**] [**Error! Reference source not found.**] [**Error! Reference source not found.**].

Training Environment: The majority of the models tested here are deep neural models that require a GPU to train in a realistic amount of time. For security reasons, the EROSKI-Full dataset can only be deployed on ISO-27001 compliant environments, and the only compliant testbed available to BigDataStack does not have GPU support. Hence, we only experiment with the EROSKI-Sample, Instacart, and Dunnhumby datasets here. Furthermore, we use only the 25% sample of the large Instacart dataset to reduce the training time for the wide range of models being tested and tuned. These models were trained in containerized form on hyper-compute nodes within the GLA testbed. Each node has 2x Intel Xeon Gold 6244 Processors, 512Gb of DDR4 memory and 8x Nvidia Titan RTX graphics cards. Each model was provided a single graphics card and sufficient CPU and memory not to bottleneck the process.

Parameter Tuning: The baseline methods are trained using the implementations included within BetaRecsys and the hyper-parameters (e.g. the factor number of NCF [**Error! Reference source not found.**]) are tuned based on the validation sets. To fairly compare the performances of all the models, we train/validate/test all of them using the same training/validation/testing sets, and the embedding sizes are set to 64. To reduce the time it takes to calculate the performance per user (which can be prohibitive for tens of thousands of users), we employ the standard practice of randomly sampling 100 negative items for each user in both the test sets and validation sets [**Error! Reference source not found.**] [**Error! Reference source not found.**]. For all the compared methods, the number of iterations and hyperparameters are chosen based on the validation sets, and all the reported performance results are based on the test set. The Triple2vec, VBCAR and VBCAR-S models are trained with a RMSprop optimizer based on the same amount of sampled triples (1 million). For reference, our VBCAR and VBCAR-S models use the following hyperparameters: 512 (dimension of hidden layer in encoder), 0.001 (learning rate), and 512 (batch size).

4.6.2. Performance Results

We report grocery recommendation performance of the 9 models listed in Section 4.3.3 over the three datasets in Table 4. From Table 4 we observe the following. First, as a sanity check, we see that all of the supervised approaches are more effective than the unsupervised TopPop baseline (that simply returns popular items), showing the importance of personalization for this task. Second, we generally see a performance uplift for the neural models such as Triple2Vec and VBCAR over the non-neural models like BPR. We also see that the new VBCAR-S model that we developed for BigDataStack is the most effective of the classical and state-of-the-art models tested. Finally, as we can see, models such as VBCAR can indeed meet KPI CC.1 with NDCG performances above 0.6, indicating very strong recommendation performance in this scenario.

Table 4 - Grocery Recommendation Performance Statistics

Model	Side Info	Instacart 25%			Dunnhumby			EROSKI-Sample		
		NDCG	Recall	MAP	NDCG	Recall	MAP	NDCG	Recall	MAP
TopPop	✗	0.1658	0.1147	0.0737	0.1581	0.009	0.009	0.3007	0.0822	0.0505
BPR	✗	0.4071	0.1363	0.1427	0.579	0.0728	0.0537	0.2995	0.082	0.0502
NeuCF	✗	0.4646	0.2753	0.1893	0.633	0.071	0.0537	0.4271	0.1334	0.0775
NGCF	✗	0.4709	0.2805	0.2131	0.6576	0.0743	0.0521	0.4404	0.1425	0.0854
Triple2vec	✗	0.4678	0.2871	0.2012	0.6195	0.0874	0.0659	0.5697	0.1601	0.1154
VBCAR	✗	0.4792	0.2913	0.2185	0.6722	0.0883	0.0741	0.5904	0.1755	0.1287
cVAE	✓	0.4154	0.1962	0.1826	0.5705	0.0727	0.0596	0.3344	0.0952	0.0565
HIRE	✓	0.322	0.1262	0.1032	0.3721	0.0625	0.0312	0.2995	0.082	0.0502
VBCAR-S	✓	**0.4832***	**0.2937***	**0.2211 ***	**0.6835***	**0.0912***	**0.0766***	**0.6028***	**0.1809***	**0.134***

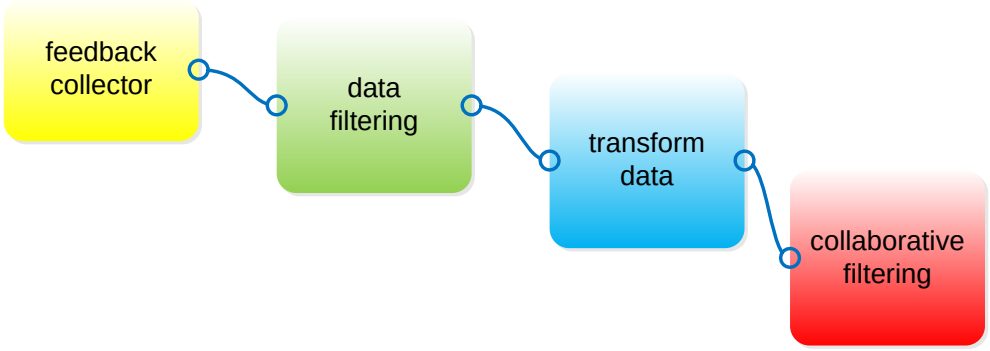
4.7. Use Case Scenario

Having shown that the recommender system deployed for this use-case is fit-for purpose in terms of performance, we now summarize how the final EROSKI recommendation system as described in Section 4.2 can be defined and deployed using the different components of the BigDataStack platform as an end-to-end scenario, comprised of different activities.

4.7.1. Activity 1: Definition of the analytics for the recommender

The first activity is related to the definition of the analytic flow of the recommender system in BigDataStack. In this activity the main actors are the business analyst who defines the high-level analytic tasks needed for the recommender and the data scientist who has to map the high-level tasks defined by the business expert to concrete analytic tasks. The following characters, who represent different end-users, take part in the activity:

- **Fernando** is the business analyst who is in charge of declaring the business process at a high level. He has also defined some business constraints for the system.
- **Martí** is a data scientist who has been tuning the different parameters for the analytic algorithm and testing the different application services.

Step	Description
CC-A1-01	From the Process Modelling Framework, Fernando builds the graph that represents the application. Concretely, Fernando adds the nodes feedback collector, data filter, transform data and collaborative filtering and links them together. 

	Fernando is also in charge of defining the global business goals of the system. One of them is Completion time < 1 minute (i.e. when the recommender receives feedback of an interaction <user, product>, the refreshment of the recommendations for this user should take less than a minute) Once the graph is ready and saved the work made by Fernando in the Process Modelling Framework is available for Martí in the Data Toolkit.
CC-A1-02	From the Data Toolkit, Martí opens the graph deployed by Fernando. He concretizes the analytic tasks defined by Fernando in the business graph. Martí makes the following actions in the Data Toolkit: • Links the high-level analytic tasks defined by Fernando to concrete executables. • Sets the SLOs for each of the executables. Concretely, the needed throughput for each of the application services of the analytic flow. To make his calculations, Martí takes into account the global business goal defined by Fernando in the previous step. • Sets values for parameters of recommendation algorithm such as implicitPrefers, maxIter (maximum number of iterations to be run).

4.7.2. Activity 2: Deployment of the application services

This activity is related to the deployment of the application specific services of the recommender system. The user will have two possibilities to deploy his application in the Application Dimensioning Workbench (ADW): assisted and manual. This activity is focused on the manual deployment. In this type of deployment, the user selects the QoS and the resources that are expected to be needed for each of the application services. Based on these input parameters, BigDataStack suggests to the user the deployment configuration that suits best to the application.

In this activity, the main actors are the application engineer, the data scientist (who has to define the application service level objectives (SLOs) for each of the application services of the recommender) and the business analyst. We are working with the following characters that represent the different end-users:

- **Fernando** is the business analyst who is in charge of declaring the business process at a high level. He has also defined some business constraints for the system.
- **Martí** is a data scientist who has been tuning the different parameters for the analytic algorithm and testing the different application services.
- **Matteo** is the application engineer who has been developing the different application components of the recommender system

Step	Description
CC-A2-01	Fernando sets the business goals of the system. One of them says 'once the recommender receives feedback of an interaction <user, product> the refreshment of the recommendations for this user shouldn't take more than one minute.
CC-A2-02	Martí, needs to deploy the application services of the recommender system he has been experimenting with. Martí calculates the SLOs for the application services of the recommender so that the whole process takes less than one minute for the calculation of the

	recommendations of a user. Then provides the numbers to Matteo with the help of the data toolkit.
CC-A2-03	Matteo makes an estimation of the resources (CPUs, Memory and number of replicas) needed for each application service and then registers the SLO's and the estimation in ADW. BigDataStack provides Matteo the a priori best deployment pattern. Matteo confirms the first deployment of the application in the ADW of BigDataStack.

4.7.3. Activity 3: Re-deployment of the application services

Once the Application Services have been deployed in BigDataStack, changes in the incoming data may happen and the fulfilment of the SLO's could be in danger. This activity aims at demonstrating the capability of BigDataStack to identify that the SLO's defined for the application services are not being fulfilled as well as to adapt the underlying needed resources so that the SLO's are fulfilled again. In this activity, the main actors are the application engineer who has to confirm or not the redeployment of the application services and the data owner who has to accept the increment of economic costs of the application due to the new resources needed.

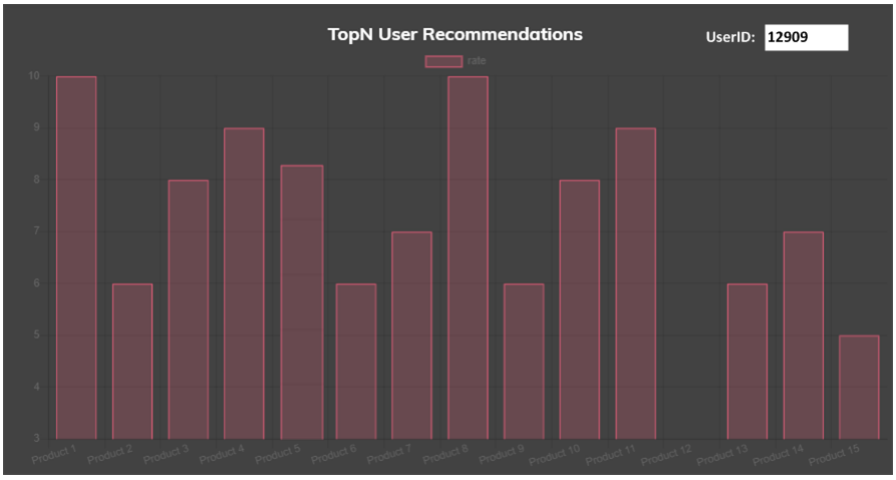
- **Matteo** is the application engineer who has been developing the different application components of the recommender system. He was the one making the first deployment of the application services of the recommender system with the Application Dimensioning Workbench.
- **Rosanna** is the data owner. She is also the owner of the budget for the recommender system.

Step	Description
CC-A3-01	Matteo receives a warning from ADW that alerts him that there has been detected a quality of service failure and recommending him to alter the application deployment. In ADW, Matteo visualizes the different new alternatives that have been calculated by BigDataStack. He can see that all of them are more expensive than the current one.
CC-A3-02	Matteo transmits to Rosanna the different possibilities in terms of costs and performance. Having analysed the different possibilities, Rosanna accepts one of the new configurations without altering the SLO's.
CC-A3-03	Matteo selects the configuration accepted by Rosanna and confirms the re-deployment of the application in the ADW of BigDataStack.

4.7.4. Activity 4: Visualize recommendations

This activity is related to the retrieval and visualization of the recommendations calculated by the system. It demonstrates the capability of the adaptable visualizations component of BigDataStack to display the outcomes produced by the recommender system. In this activity, the main actor is the data scientist, who wants to visualize the recommendations calculated by the system for a given customer. We are working with the following character that represents the end-user:

- **Martí** is a data scientist who is testing and analyzing the results of the application services layer offered by the system.

Step	Description
CC-A4-1	Martí, is experimenting with the recommender system and needs to display the outcomes calculated for the system for a given user. He connects to the adaptable visualization component and enters the user whose recommendations wants to analyse. BigDataStack provides him with information about the recommendations calculated for the user – the top products calculated for the user as well as the ranking given for each of these products. Find below a prototype of the screen he will be displaying: 

4.7.5. Activity 5: Provide recommendations

This activity relates to the capability of the recommender system to provide the recommendations calculated by the system to external systems. The data storage used for the recommender system has been the LeanXcale SQL data store provided by BigDataStack. It demonstrates the capability of BigDataStack to retrieve data from one of its data stores. In this activity, the main actor is an end user of the ecommerce site of EROSKI, who is browsing in the e-commerce site of EROSKI. We are working with the following character that represents the EROSKI end-user,

- **Manuel** is a shopper who is making a purchase in the e-commerce site of EROSKI.

Step	Description
CC-A5-01	Manuel is navigating in the e-commerce site of EROSKI in order to make his weekly purchase. He accesses the section of the store named beers. The e-commerce site requests to the recommender system the recommendations to provide for the user Manuel.
CC-A5-02	The recommender retrieves the data from the BigDataStack data store to provide the e-commerce the recommendations calculated.

CC-A5-03	The e-commerce site selects those products that are available in the store on which Manuel is currently buying: Then displays them to Manuel somewhere in the screen of the section beers.

4.7.6. Activity 6: Collect feedback

This activity relates to the capability of the recommender system to collect feedback from the external systems that are using the recommendations calculated by the system. It demonstrates the capability of BigDataStack to store valuable data in one of its data stores, concretely, in LeanXcale. In this activity, the main actor is an end user of the ecommerce site of EROSKI, who is browsing in the e-commerce site of EROSKI. We are working with the following character that represents the EROSKI end-user:

- **Manuel** is a shopper who is making a purchase in the e-commerce site of EROSKI.

Step	Description
CC-A6-01	Manuel is navigating in the e-commerce site of EROSKI in order to make his weekly purchase. He is located in the section of the store named beers. The e-commerce has recommended to him to buy 'Feta cheese' because the collaborative algorithm has calculated to be likely he would like this product.
CC-A6-02	Manuel has heard of Feta cheese before but never tasted it. He clicks on the product in order to see the detailed product cart.
CC-A6-03	The e-commerce site sends an event to the recommender system warning it that Manuel may be interested in the item.
CC-A6-04	The recommender system stores the event 'Product visualized' in the LeanXcale data store and launches the calculation of the recommendations for Manuel.

5. Smart Insurance (SI)

This section provides a description of the Smart Insurance (SI) use case scenario from GFT.

5.1. Overview and goal

The BigDataStack financial use case is built upon the BigDataStack platform to implement smart recommendation systems for the insurance market. The datasets used within the process of the use case is corporate data provided by an insurance company, based in Italy, selected from the GFT customers' portfolio, as well as open data including statistics about hydrogeological risk data, earthquake risk data, crime data.

What is the scenario's goal?
The overall goal of the BigDataStack insurance demonstrator is to exploit the BigDataStack platform big data technologies in order to access and analyse information coming from diverse and heterogeneous data sources including the in-house data (e.g. customers location, customers portfolio), in order to provide recommendations for Insurance Companies allowing them a better customers' management (e.g. strategies for cross-selling and up-selling).

What is the business objective?
The business objective of this scenario is to allow insurance companies to improve the management of their customers' portfolio, by providing smart services that provide personalized products to their clients, through machine learning algorithms and cross-selling/up-selling strategies. Also, the aim is to help the company to improve their customers' propensity (i.e. their will to continue using the insurance's services) and on the other hand, predict the customers' churn rate. Summarizing, the expected benefits of the BigDataStack adoption by Insurance companies, is to improve the customer's satisfaction through personalised offering and support. This is made possible by rapid processing and analysis of huge volumes of cross domain data.

Who are the actors in the use case?
The actors involved in the first scenario of the SI pilot have been extracted from D2.1, as reported in the following table.

Table 5 - Smart Insurance Actors

Id	Name	Description
ROL-01	Data Owner	BigDataStack offers a unified Gateway to obtain both streaming and stored data from data owners and store them in its underlying storage infrastructure that supports SQL and NoSQL data stores. In the SI the Data Owner is the Insurance company.
ROL-02	Data Scientist	BigDataStack offers the Data Toolkit to enable data scientists both to easily ingest their analytics tasks by utilizing a declarative paradigm, and to specify their preferences and constraints to be exploited during the dimensioning phase regarding the data services that will

		be used (for example preferences for the data cleaning service)
ROL-03	Business Analysts	BigDataStack offers the Process Modelling Framework allowing business users to define their functionality-based business processes (through declaratively-defined models) and optimize them based on the outcomes of process analytics that will be triggered by BigDataStack.
ROL-04	Application Engineers and Application Service Owners	BigDataStack offers the Application Dimensioning Workbench to enable application owners and engineers to experiment with their applications and dimension it in terms of its data needs and data-related properties

What are the key activities these actors are involved in?

The SI use case can be broken down into the following main activities:

1. Data acquisition. This activity aims to set up a data source from which BigDataStack obtains data to store in its infrastructure. At this stage, personal information is removed from the datasets from the data owners. Thus, the main actor in this step is the Data Owner.
2. Analytics definition. This activity aims at defining the business processes and the main analytical tasks that the recommender system needs. The main actors in this activity are the Business Analyst and the Data Scientist.
3. Deployment of the application services. This activity aims at testing the capability to deploy on top of BigDataStack the application services of the recommender system. This activity mainly applies to the application engineers.
4. Display recommendations. The Data scientist wants to know which recommendations the system will propose for a given user. The main actor in this activity is the data scientist.

5.2. Pilot Description

Data analytics in the insurance industry is transforming the way insurance businesses operate. Data and analysis have always been the basis of the insurance industry, although in the last years the evolution of ICT solutions in data analysis reflected, as reported by Accenture[7], in an increase in the number and value of investments. In fact, 80% of insurers currently invest moderately or significantly in new digital technologies and 61% are expecting to increase their investment soon. On the other hand, the Chartered Institute of Loss Adjusted[8] stated that 82% of industry professionals believe organizations which do not utilize big data will likely become uncompetitive. New technological support has dramatically impacted on the insurance sector as well as in the other fields, and insurers are leveraging data to attract,

[7] See: https://www.accenture.com/_acnmedia/PDF-79/Accenture-Technology-Vision-Insurance-2018.pdf [Accessed on June 18, 2019].

[8] See: https://www.the-digital-insurer.com/wp-content/uploads/2015/03/478-3-ASNY-Presentation-Predictive-Analytics-Final.pdf [Accessed on June 18, 2019].

retain and service clients and producers, develop new products, assess and mitigate risks, set rates, process claims and manage financial performance.

The BigDataStack platform is the perfect candidate as backbone for the full management of the data collected by insurance companies: BigDataStack has the right tools to make the data, from different sources and of different formats, meaningful, and to manage them in the whole stack of related applications. By combining data with analytics, insurers can generate insights that help transform the business, create closer relationships with customers, gain competitive advantages or perhaps generate entirely new business models.

The SI pilot comprises two main scenarios:

1. Product Recommendation:
   Based on the similarities between customers (from corporate datasets) and the geographic location where the customers are present (from corporate + open datasets), the service will recommend policies to both existing or new customers.
2. Churn Prediction:
   Based on the above information and their relationship with cancelled policies, the service will predict the likelihood that the customer is going to cancel a particular policy.

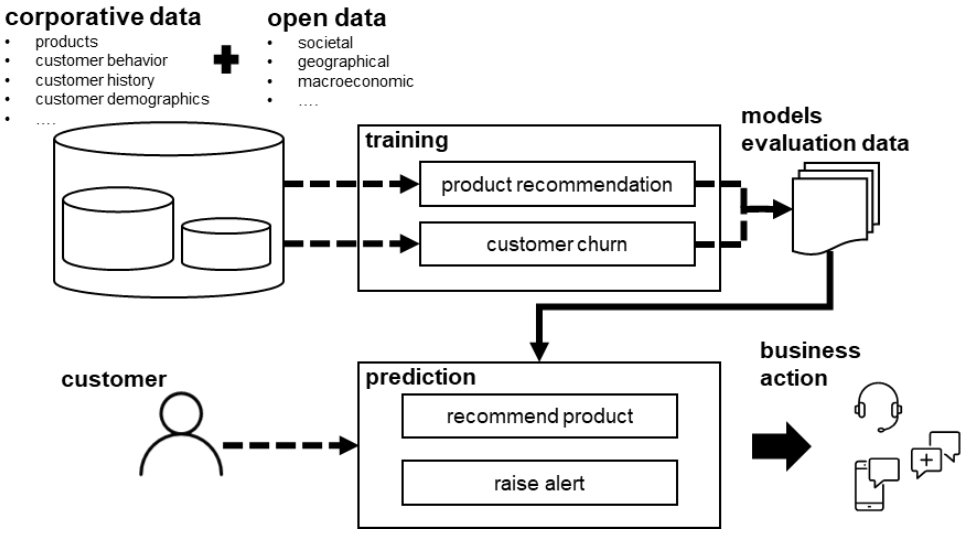


Figure 15 - Functional and analytic flow.

Hence, from an engineering perspective, the use-case centres around the development of BigDataStack-compatible applications the enable these two scenarios. Once developed, these will be configured, deployed and orchestrated via the BigDataStack platform.

In the next section we introduce the dataset used to support this use-case, while in Section 5.4 we describe the outcomes of an investigative summary into this dataset and its properties. Meanwhile, in Section **Error! Reference source not found.** we describe each of these two scenarios and their implementation in more detail.

5.3. Insurance Dataset

In order to understand the scenarios, we provide a brief description of the structure of the data provided by the insurance company end-user. Here the actual corporate data is distributed in more than 20 tables. A summary of the primary tables and the information they contain can be found in Appendix H. An extensive EDA was performed to identify the most relevant variables for the use cases. Three of these tables contained most of the information needed for both scenarios:

- **Insurance Policies**: The table **PTF** contains information of 8M+ insurance policies (idpolizza) with the product acquired (prodotto), including the status and the province of the policy. The policies are distributed into four different business areas (ramo). The number of products by business areas are (active policies only): *Rami Elementari=90+, Vita=80+, Auto=4, and Cauzioni=1.* The *ramo Auto* contains the products *Auto (97%), Natanti (boats), Porsche, and Mio Camper*, which are too specific to be used as recommended products.
- **Policy Holders**: The table **ANA_PTF** contains information about all the users (codice_fiscale) related to each policy (idpolizza) and their role. We are only interested in the owner of the policy (role=CONTRAENTE) as the actual customer.
- **Purchased Policies**: The table **ANACONTATORI** contains information about the users (codice_fiscale) and their portfolio, the number of policies by business area

The rest of the corporate tables have additional information about the users, policies and claims. Finally, from the open data tables available, a master table by province was generated integrating basic & crime statistics data for each of the 107 Italian provinces.

Anonymisation: Following the GDPR directive, all sensitive information of the datasets have been anonymized. For the encryption, we used a cryptographic hash function, the MD5 algorithm. It is a unidirectional function different from coding and encryption because it is irreversible. The spread of this encryption algorithm is still widespread (just think that the most frequent integrity check on file is based on MD5). This function takes as input an arbitrary length string and outputs another 128-bit output. The process happens very quickly and the output (also known as "MD5 Checksum" or "MD5 Hash") returned is such that it is highly unlikely to obtain the same hash value in output with two different input strings. We have modelled the length of the encrypted string, based on the length of the field to be encrypted. For example, for the tax code the encrypted string is 16 characters, while for the license plate it is 8 characters. This eliminates the possibility of tracing back to the initial value. We have performed several decrypting tests present on numerous online sites and no one has been able to decrypt the string entered. Furthermore, we have carried out a univocal check of all the encrypted keys, so that the possibility of two different string yielding identical encrypted strings is excluded.

5.4. Insurance Data Analysis

The insurance dataset described above is interesting from a data analytics perspective, as the temporal dynamics of the data (how policies and claims interact over time) strongly influence the effectiveness of any developed solution for the use-case scenarios. Hence, we first

performed an analysis of the dataset and its properties to better inform subsequent developments, which we summarize in this section.

The first aspect of the dataset that is of note is that it contains a large number of heterogeneous policies. In particular, the full dataset contains over half a million policyholders and over a hundred non-life insurance policies (e.g. car, home, health insurance, etc.). In this short study, we focus on a single policy type group, i.e. car insurance policies. This sub-set comprises 123,906 customers (45.7% male, 28.2% female and 26.1% unspecified). We provide a illustrative visual overview of how the data is structured in Figure 16 below. The unfilled blue rectangles represent policies purchased by a customer over time, while green boxes indicate insurance claims made by those customers over time.

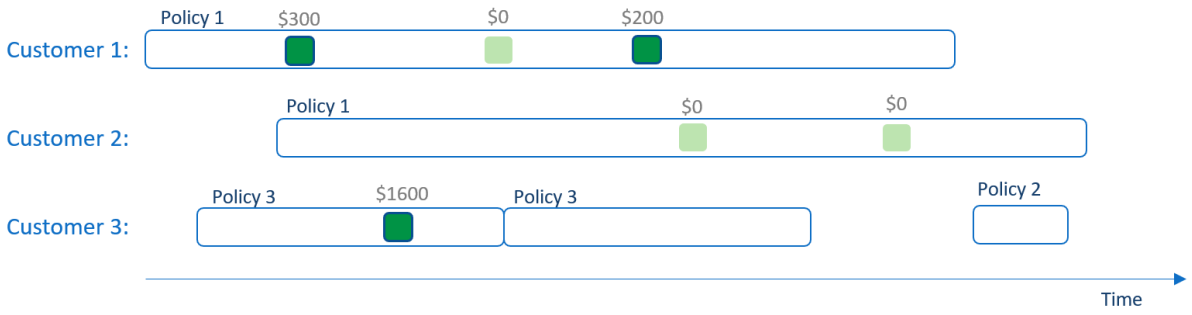


Figure 16 - Insurance Data Structure

As we can see from Figure 16, each customer can have one or more insurance policies over time. For an insurance policy, a customer may make zero or more claims. Furthermore, it is of note that a claim may result in a $0 pay-out, i.e. the claim was rejected. If we analyse the claim distribution we obtain a mu of $283.2 and sigma of $1686.6, indicating that claim values are strongly imbalanced. In fact, there are only around 10,000 claims with payout values greater than zero, while 113,907 claims (i.e., roughly 92%) were rejected, i.e. most claims fail. This will likely negatively affect the training models when using the claim-based data as there are few purchased policies that actually pay out to the customers.

Furthermore, it is of note that each policyholder in the dataset is associated with one or more claims registered at different times during the insurance coverage period. The area-based visualisation in Figure 17 illustrates the proportion of customers that have made one claim, two claims, three claims, etc. In this case, the largest area (X) represents customers that make only one claim (around 120,000 customers), while the smallest block represents the 243 customers that filed ten claims. This shows that per-customer, even when claims are made, there is often only a single claim per policy.

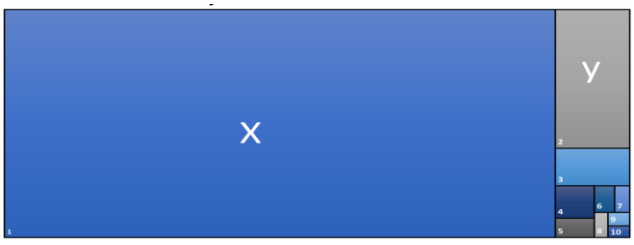


Figure 17 - Tree-Map Visualisation of Number of the Claims per Customer

Finally, considering data modelling, since each claim occurs at a particular date, the insurance data can be represented as a series of points (one for each claim) indexed by date. However, unlike other time series data, observations are sequences taken at successive not-equally spaced points. Moreover, instances in time exist only in the event of a claim resolution, and as noted above, customers with multiple claims are rare.

This above analysis is an extract from a larger examination of the dataset and its utility for modelling insurance claims performed in BigDataStack. The full report can be found in Appendix I.

5.5. Insurance Recommendation and Churn Prediction

In this section we summarize the current development efforts associated to each of the two use-case scenarios.

5.5.1. Policy Recommendation Scenario

The high-level goal of the policy recommendation scenario is, for a given user, to produce a set of recommended policies that the user may select (where suitable policies are ranked more highly).

Dataset Preparation: There are some characteristics that make this use case difficult to solve with a simple recommender, especially the high unbalanced distributions of products by business area. Another limiting aspect of this use case is the limited usable data available for modelling (see the previous section); the recommender is modelled with active policies only (1.3M+); the cancelled policies were rejected here for two reasons: i) many products from the cancelled policies were not found in the active policies (assumed obsolete), ii) they are prone to being cancelled again. As a recommendation system, the product and the user must be clearly identified. Crossing the above tables **PTF X ANA_PTF** results in 700k+ active policies with identified owners (400k+ different users). This corporate policy data is completed with relevant variables found in other tables, such as the number of policies of the user by business area (portfolio) and the user entity (person or company). This meaningful corporate policy dataset is enriched with the table by province resulting in a full master table of 700k active policies with corporate & open data features. From those, only 250k policies belong to the business areas (ramo) with different products than can be recommended: *Rami Elementari* and *Vita*. Finally, the samples are further reduced for modelling to 44k-176k depending on the selected features & the data preparation filters for the ML algorithm (e.g. duplicates removal). Based on this processed data, several recommenders based on collaborative filtering and supervised machine learning (ML) were developed and evaluated.

Collaborative Filtering Models: Although some acceptable results were obtained, the evaluation of recommenders based on collaborative filtering revealed two limitations with our data:

- The average number of items per user is very low (1.16), thus making any user-item matrix factorization technique difficult to work with. Only reduced subsets of users with more than one item could be successfully modelled.

- The impossibility of introducing policy-content features (type of user, location ..), as collaborative filtering methods are only based on users, products & ratings.

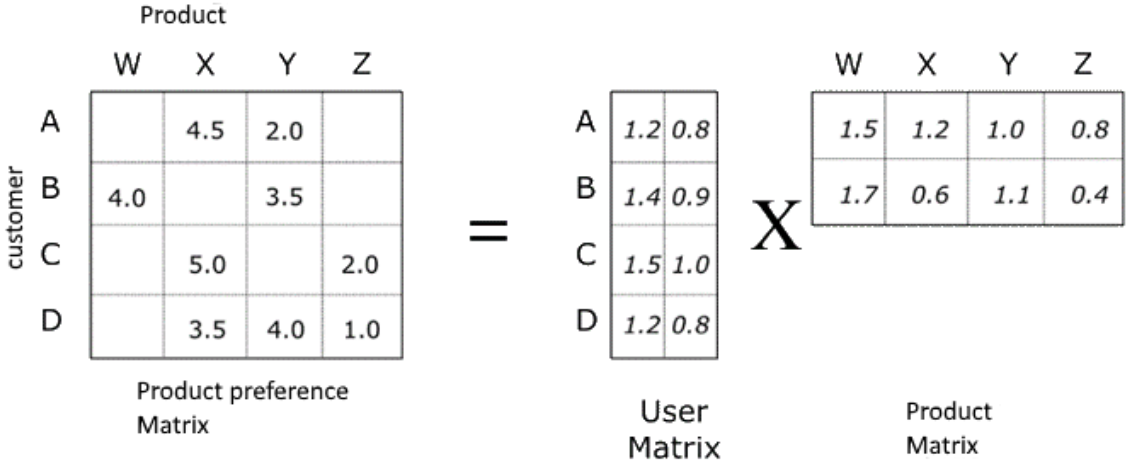


Figure 18 - Collaborative filtering schema. For this insurance case this typical recommendation algorithm has showed to be less performant than other approaches.

Supervised Machine Learned Models: A ML classifier implemented as a product recommender was finally chosen to address the complete recommender with corporate & open data, allowing 30+ useful variables. Here the target is the product or products which are more likely to be acquired by an existing or new customer (or groups of customers). This solution was selected over hybrid recommenders (user clustering + collaborative filtering) and other advanced methods since the variables related with the location are specific to the policy, not the user nor the product (a user can have multiple policies in several provinces). The best results were obtained with tree-based classifiers (Random Forest / XGBoost) with accuracies between 19% (WCS) - 36% (duplicated samples allowed) for 120+ classes (products). The drawback of not modelling the interactions between users and items (though the average items per user is only 1.16), can be compensated with a multiclass multioutput classifier. Finally, a postprocess stage for the predictive model is performed to rank the products and avoid recommending repeated ones. Other advantages of the selected solution are:

- As products can be recommended to new customers, potential areas or specific combinations of features of new customers can be identified for commercial expansion.
- Tree-based ML classifiers are compatible with modern interpretability techniques such as Shapley values. The model and each of the individual recommendations can be explained by returning which features contributed most to the specific recommendation and how.
- The model is flexible enough to be continuously improved with feature engineering and new corporate and open data variables (e.g. subprovince-level socio-economic features)
- Besides cross-selling (implicit in the ML model), up-selling could be later implemented at the postprocess stage given a detailed description of the different products, their priorities & values (an existent numerical variable *premium* can be used here)

A complementary collaborative filtering *ramo recommender* (business area) was developed directly from the ANACONTATORI table (portfolio of users). Here the model tries to learn the interaction between the customer (user) and the *ramo* (item) based on the amount of policies per *ramo* of the customers (rating). The goal is to recommend *ramos* (products aggregated by business area) to existing customers. The advantages of this recommender include the large number of samples (3.3M) and users (845k) modelled, the quality of the numerical rating, from 0 (no product) to 10 (10+ products in this ramo), and the performance obtained: RMSE<1 (target 0-10). The main drawback is that it is a business area recommender with only four items that should be complemented with a full product-level recommender.

5.5.2. Churn Prediction Scenario

Customer retention is one of the main goals of customer relationship management systems given the business relevance of such problem: the cost for customer acquisition far exceeds the cost of customer retention. According to some studies can achieve up to a factor of $\times 20$. Churn models aim to identify early churn signals of customers with n increased likelihood to leave voluntarily. Here the goal is for a given user, to calculate the probability that the user will not renew an existing policy. Since in the insurance industry policies are normally renewed at given periods of time, a policy-level churn prediction is modelled with data from both active & cancelled insurance policies. Most of the EDA & ETL is reused from the product recommendation scenario, e.g. we are only interested in the owner of the policy as the actual customer. Popular in the machine learning research community model for the churning prediction problem comprise Neural Networks, Decision Trees, Regression and Logistic Regression, Support Vector Machine and Naïve Bayes. The model developed here is a ML binary based in Naïve Bayes and Random Forest classifiers, which includes all the above advantages of the recommender based on supervised ML.

5.5.3. Implementation Details

Environment: All the work has been implemented and evaluated in a BigDataStack's OKD pod with one Linux container in Python 3.6 and Spark 3.0.1. The full EDA and ETL stages (mostly cleaning, normalization, and crossing) of the corporate & open data files are performed on Python/Pandas. The amount of data available along with the need of columns aggregation requires flexibility. Python/Pandas was selected for outperforming PySpark in the ETLs stages with the data available. The processed data such as the master tables generated are saved in .parquet format for compatibility with Spark.

Libraries Used: The Collaborative Filtering models for both ramo and product recommenders are implemented and evaluated with PySpark ALS. The final Supervised ML models were evaluated for both PySpark and Python ML Stack (scikit-learn, xgboost, shap, pysurvival). The Python ML solution was finally selected due to the small amount of data available for modelling (44k to 176k samples), with training times <2s (>2m in PySpark). Python ML also provides more flexibility and advanced libraries for explainability such as shap. The Churn Prediction model is also implemented with Supervised ML in Python for the same reasons as the recommender.

Deployment: A Flask web server app in a Python container with the training and inferences services is used to deploy the solution (Figure 15). One for product recommendation and one for churn prediction. The training service of the app receives a web request with the desired parameters of the data preparation and ML modelling (e.g. hyperparameters for training the model), generates and saves the ML model, and returns the ML performance obtained and additional information. Here all the pipeline, including the model training, can be executed in Python /Pandas/Scikit-learn within a few seconds. On the other hand, the inference service performs predictions over the trained model to get the recommended products to existing or new customers (a post process Stage is added to recommend a proper product) or the churn prediction of existing policies. Additionally, this service could return the prediction explained.

5.6. Use Case Scenarios

Having described the two use-case scenarios for this use-case we now summarize how the resulting system can be defined and deployed using the different components of the BigDataStack platform as an end-to-end scenario, comprised of different activities.

5.6.1. Activity 1: Data acquisition

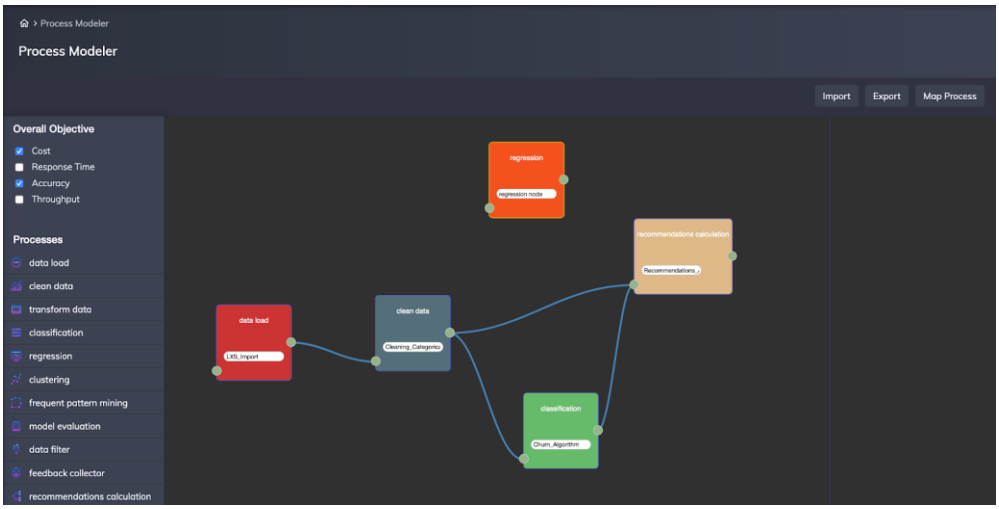
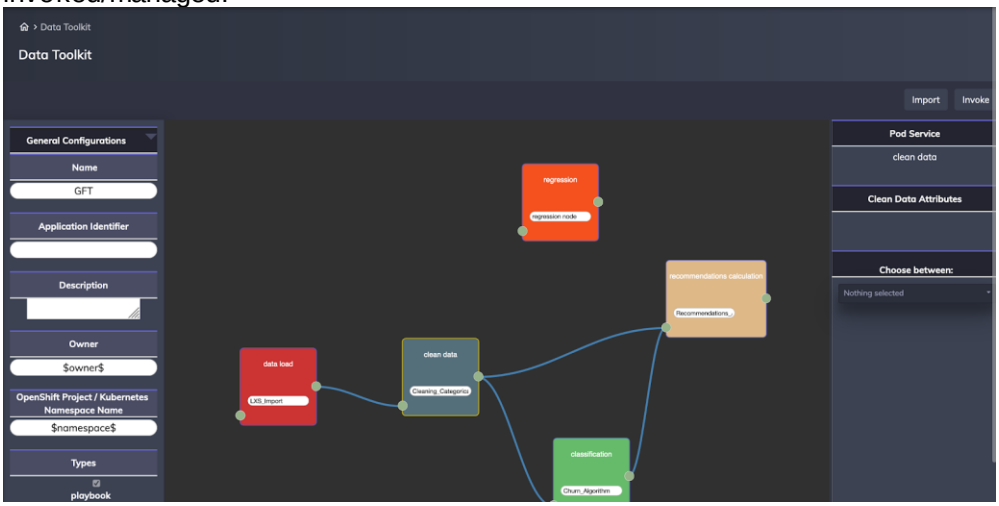
The first activity is to define the data source in order to obtain the required datasets. The process is described in the following table.

Step	Description
SI-A1-01	The Data Owner picks the data source type from a drop down list, e.g., database, FTP server etc. If the selected type is database then the RDBMS should be defined, again from a drop-down list, e.g., postgres, SQL-server etc.
SI-A1-02	The Data Owner enters in plain text format the required connection attributes, i.e., IP address, username, password, etc.
SI-A1-03	The system validates the connection.
SI-A1-04	The Data Owner defines what particular resource is to be accessed. For example, if the selected datasource type is database, selects from a drop down list the preferred tables. If the source is an FTP server, the exact path should be defined.
SI-A1-05	The system informs the user that the requested data sources are successfully defined.

5.6.2. Activity 2: Analytics definition

This activity concerns the definition of the analytic flow of the recommender system in BigDataStack. The main actors are the business analyst (BA) who defines the high-level analytic tasks needed for the recommender and the data scientist (DS) who will map the high-level tasks to concrete analytic tasks.

Step	Description
SI-A2-01	This use case is triggered by the Business Analyst through the creation of a high level abstract graph (Process Modeler component). The Business Analyst can define also the Overall Objectives of her application. Subsequently, the use case

	demonstrates the propagation of this graph to the Process Mapping component where in some processes/services (wherever it is applicable) the selected Machine Learning algorithm will further be assessed based on the selected Overall Objective. This step is optional. Once the graph is ready and the business goals are saved, the work made by the BA in the Process Modelling Framework is available for the DA in the Data Toolkit. 
SI-A2-02	As a second step the Data Analyst consumes this graph proposal by the Process Mapping in the Data Toolkit component and concretizes the analytic tasks, by linking the high-level analytic tasks defined by the BA to concrete executables. There, specific values (regarding the Overall Objectives also coupled with SLOs supported by the pods at the infrastructure level) are fine-tuned and the result is the generation of the playbook (i.e. YAML file). The playbook later on is directly posted to the Realization Engine where deployment of the application can be invoked/managed. 

5.6.3. Activity 3: Deployment of the application services

This activity is related to the deployment of the application specific services of the recommender system. The user will deploy his application in the Application Dimensioning Workbench (ADW), by selecting the QoS and the resources that are expected to be needed

for each of the application services. Based on these input parameters, BigDataStack suggests to the user the deployment configuration that suits best to the application.
The main actors are the application engineer (AE), the data scientist (DS) and the business analyst (BA).
From within the Realization Engine, the application owner can view the different components of their application. For this use-case, this comprises the insurance recommendation service, the churn predictor, along with supporting services such as containers for training/updating the underlying machine learned models for those components. These components can be either manually deployed by the user through the Realization Engine, or otherwise managed automatically through a container responsible for the application business logic that can issue actions through the Realization Engine. For example, such a container could be used to perform hyper-parameter search for the insurance recommender service with the aim of enhancing model performance.

Step	Description
SI-A3-01	The BA sets the business goals of the system.
SI-A3-02	The DS needs to deploy the application services of the recommender system he has been experimenting with. The DS calculates the SLOs for the application services and then provides the numbers to the AE with the help of the data toolkit.
SI-A3-03	The AE estimates the resources (CPUs, Memory and number of replicas) needed for each application service and then registers the SLO's and the estimation in ADW. BigDataStack provides the AE the a priori best deployment pattern. The AE confirms the deployment of the application in the ADW of BigDataStack.

5.6.4. Activity 4: Display recommendations

This activity is related to the retrieval and displaying of the recommendations calculated by the system. It demonstrates the capability of the adaptable visualizations component of BigDataStack to display the outcomes produced by the recommender system. The main actor is the data scientist (DS).

Step	Description
SI-A4-1	The DS is experimenting with the recommender system and needs to display the outcomes calculated for the system for one or more users. She/he connects to the adaptable visualization component and enters the user whose recommendations wants to analyse. BigDataStack provides her/him with information about the recommendations calculated for the users.

5.6.5. Activity 5: Provide recommendations

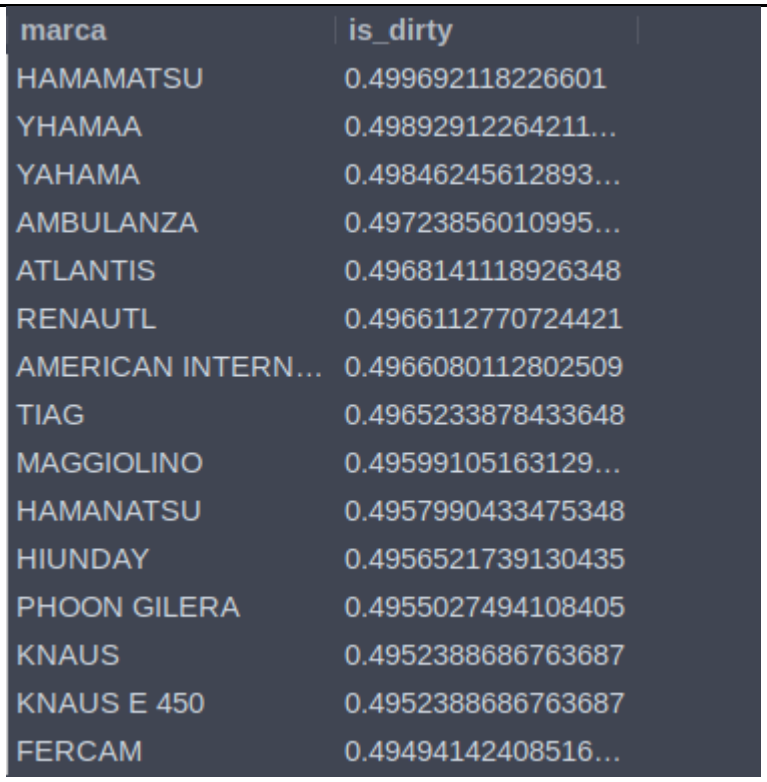
This activity concerns the capability of the recommender system to provide the recommendations calculated by the system to external systems. The data storage for the recommender system is the LeanXcale SQL data store provided by BigDataStack. The system features a web REST API that could interact with the insurance company, as a web service, per user, etc.

Step	Description
SI-A5-01	The recommender and/or churn predictor retrieves the data from the BigDataStack data store to provide the web REST API the calculated recommendations/results.
SI-A5-03	An external web application can access the BigDataStack web REST API to select the recommended offerings and other results.

5.6.6. Activity 6 Data Quality Assessment

This activity aims at evaluating the quality of the data prior to any analysis on them, to ensure that analytics outcomes are based on datasets of high quality. The component incorporates a set of algorithms to enable domain-agnostic error detection, in a tabular dataset, detecting several types of errors (e.g., format errors, spelling mistakes, etc.) within a single column. The actor is the Data Scientist.

Step	Description
SI-A6-01	The DQA component retrieves the data stored in the database at periodic intervals
SI-A6-02	The DQA component fills the last column in the database (i.e., *is_dirty*) with a probability indicating that the row is valid or not.

	
SI-A6-03	The user can view the state of its dataset from the main BigDataStack dashboard. 
SI-A6-04	The Data Scientist can find rows that have high probability of being corrupted and exclude them from downstream analytical tasks.

5.7. Key Performance Indicators

Our smart insurance company uses the BigDataStack platform to deploy the product recommender together with the churn prediction system to improve the user experience and satisfaction and enhance the business results in a diversity of scenarios, like:

1. Provide on-line product recommendations at any contact point with the customer, whether is at the insurance renewal or when the customer contacts the insurance company by any reason or any channel (mostly seeking help or for information

purposes, by phone or from the web).
2. Use product recommendations on marketing campaigns. This will mostly performed in batch mode for selected groups of customers.
3. On-line, on-demand churn detection at the customer interaction point with the insurance company for quickly reaction or take advantage of the contact situation to try to reverse the customer drift to leave the company.
4. Periodically or in reaction to external factors, like competition campaigns for increased customer retention.

To evaluate the degree of success or failure of the system with regard to these scenarios, different Key Performance Indicators (KPIs) have been identified that can provide timely, relevant and accurate performance criteria. A KPI is defined by one or more metrics, together with threshold values.

KPI SI.1	
Title	Recommendation and churn lift curve
Measures	For a given product or customer, compute the churn or purchase probability as predicted by your model binned by different probability values. At each bin, overlay the real customer churn or product acquisition probability.
Success Criteria	Smooth almost exponentially increasing distribution towards 1 when the probability in the x-axis goes to 1 gives an spectral overview of a well behaved model.

KPI SI.2	
Title	Recommendation and churn gain curve
Measures	For different product groups or customer groups measure the gain with the recommendation / prediction churn system on place against the historical or control group, asses the gain in purchases or customers retained.
Success Criteria	Compare gains against cost of retention / promotion action will yield an overall effectiveness of the system, beyond the mere prediction effectiveness

KPI SI.3	
Title	Model Precision & Recall

Measures	ROC curve, precision and recall of model recommendations (purchase / no purchase, churn / no churn) per customer / product.
Success Criteria	This KPI allows for an optimal selection of the machine learning algorithm. Any success that is above random recommendations and with gains above the retention / promotion costs can be considered a success.

KPI SI.1 and KPI SI.2 can be evaluated by the end-user in real operating conditions. However, KPI SI.3 can be evaluated at algorithm training and design, and as such we briefly summarize the results in the next section.

5.7.1. Performance Evaluation

For machine learning models as the developed for the recommendations and churn use case, due to its statistical nature, to main aspects must be tested: the ability of the model to capture the fundamental traits of the phenomena at hand (fit, underfit or overfit) and the ability to generalize to unseen data, where the model has to demonstrate its ability. The standard methodology is to split the dataset in *train* and *test*. As in any process where the time dimension (*i.e.* the temporal order of the predictions) can play a significant influence, the *train* and *test* datasets are also ordered in time, so any data point in the *train* dataset occurs before any data point in the test set [1], [18]. This allows to prevent the so called *data leakage*, where information generated in a future is during training to produce the prediction. Data with such characteristics obviously will not available in real conditions, and therefore can report a misguiding generalization ability of the model.

Due to the intrinsic nature of the insurance sector, where the basket of possible products is relatively reduced with respect to other use cases, we use standard metrics like the accuracy of predicting the churn of a given customer or, when applicable, that a given customer will acquire a given product from a basket. Accuracy can be a misleading metric, especially in the case of heavily imbalanced datasets, where the number of customers buying different products, or the number of customers churning / not churning is very different. Usually, more reliable metrics like precision and recall or f1 are used. However, in our case due to the inherent diversity of the problem, the values of such metrics *per se* can yield deceptive low values. What is most interesting from the business point of view, is the *lift* from a random generated (based in the most common class) recommendation or churn prediction. Therefore, our chosen metric will be the accuracy compared with respect a random model based that recommends a product or predicts churn with a probability based in the relative frequencies of the purchased products or churn rate.

Product Recommender Results

For a training dataset of 40858 customers, characterized by 135 different features where the target is to predict if the customer will buy a product from a basket of 120 different products, Table 6 shows the model accuracy, f1 macro and f1 weighted for the random, naïve Bayes

and random forest model, measured on a test set of 35299 customers, respectively. The random forest model achieves a significant lift with compared with the random model.

Model	*Accuracy*	*F1 macro*	*F1 weighted*
Random	0.14	0.00	0.04
Naïve Bayes	0.09	0.01	0.05
Random Forest	0.36	0.05	0.25

Table 6 - Product Recommender metrics

Churn prediction

We consider here churn prediction as a binary classification problem. With a training dataset of 930,712 customers characterized by 15 different features, Table XX shows the model accuracy, f1 score and ROC AUC score measured on a test set of 232,679 customers for the random, naïve Bayes and random forest models, respectively. Here we observe that the accuracy lift of the random forest model is not significant with respect to the random model, although the ROC AUC it is. This is an indication that the decision thresholds of the random forest model must be properly calibrated, as the ROC curve in Figure XX shows. The trade-off between true positive rate / false positive rate must be chosen on the basis of business considerations like customer retention cost, negative incentives for the customer in the false positive rate, etc.

Model	*Accuracy*	*F1 score*	*ROC AUC*
Random	0.71	0.83	0.50
Naïve Bayes	0.71	0.83	0.50
Random Forest	0.75	0.84	0.76

Table 7 - Churn prediction metrics

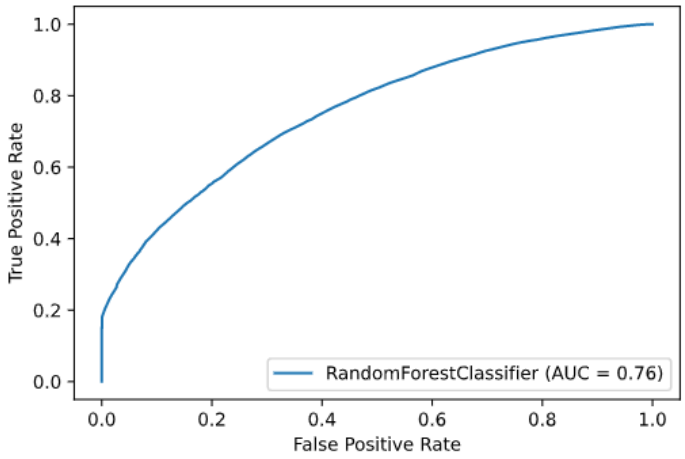


Figure 19 - Churn prediction ROC curve

6. Conclusions

This document presents the three use cases of the BigDataStack project: Real-time Shipping Management, Connected Consumer and Smart Insurance. All use cases have been deployed on BigDataStack infrastructure and the corresponding datasets have been ported to the BigDataStack storage engine (LeanXcale database and Object store). The applications and their data / analytics services are using BigDataStack offerings to achieve the corresponding identified business and technical goals for each use case.

Moreover, for each of the three pilots, this deliverable has provided information on their description, scenarios and the full implementation as well as the integration with the BigDataStack platform.

Bibliography

[1] Pedro G Campos, Fernando Déz, and Manuel Sánchez-Montañés. Towards a more realistic evaluation: testing the ability to predict future tastes of matrix factorization-based recommenders. In *RecSys*, pages 309–312, 2011.

[2] Yifan Chen and Maarten de Rijke. A collective variational autoencoder for top-n recommendation with side information. In *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, pages 3–9, 2018.

[3] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *RecSys*, pages 101–109, 2019.

[4] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1809–1818, 2015.

[5] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182, 2017.

[6] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008.

[7] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

[8] Sheng Li, Jaya Kawale, and Yun Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 811–820, 2015.

[9] Xiaopeng Li and James She. Collaborative variational autoencoder for recommender systems. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 305–314, 2017.

[10] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 689–698, 2018.

[11] Tianqiao Liu, Zhiwei Wang, Jiliang Tang, Songfan Yang, Gale Yan Huang, and Zitao Liu. Recommender systems with heterogeneous side information. In *Proceedings of the World Wide Web Conference*, pages 3027–3033, 2019.

[12] Jarana Manotumruksa, Craig Macdonald, and Iadh Ounis. A deep recurrent collaborative filtering framework for venue recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1429–1438, 2017.

[13] Jarana Manotumruksa, Craig Macdonald, and Iadh Ounis. A contextual attention recurrent architecture for context-aware venue recommendation. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 555–564, 2018.

[14] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, pages 1257–1264, 2008.

[15] Xia Ning and George Karypis. Sparse linear methods with side information for top-n recommendations. In *Proceedings of the 6th ACM conference on Recommender systems*, pages 155–162, 2012.
[16] Bo Pang, Min Yang, and Chongjun Wang. A novel top-n recommendation approach based on conditional variational auto-encoder. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 357–368, 2019.
[17] Sunho Park, Yong-Deok Kim, and Seungjin Choi. Hierarchical bayesian matrix factorization with side information. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2013.
[18] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. *ACM Computing Surveys*, 51(4):66, 2018.
[19] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th conference on uncertainty in artificial intelligence*, pages 452–461, 2009.
[20] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 811–820, 2010.
[21] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644, 2011.
[22] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. *arXiv preprint arXiv:2005.09683*, 2020.
[23] Steffen Rendle, Li Zhang, and Yehuda Koren. On the difficulty of evaluating baselines: A study on recommender systems. *arXiv preprint arXiv:1905.01395*, 2019.
[24] Noveen Sachdeva, Giuseppe Manco, Ettore Ritacco, and Vikram Pudi. Sequential variational autoencoders for collaborative filtering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 600–608, 2019.
[25] Daniel Valcarce, Alejandro BellogÃ-n, Javier Parapar, and Pablo Castells. Assessing ranking metrics in top-n recommendation. *Information Retrieval Journal*, 23:411â€“448, 2020.
[26] Mengting Wan, Di Wang, Jie Liu, Paul Bennett, and Julian McAuley. Representing and recommending shopping baskets with complementarity, compatibility and loyalty. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1133–1142, 2018.
[27] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Relational stacked denoising autoencoder for tag recommendation. In *Proceedings of the 29th AAAI conference on artificial intelligence*, 2015.
[28] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019.
[29] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *SIGIR*, pages 165–174, 2019.
[30] Yaxiong Wu, Craig Macdonald, and Iadh Ounis. A hybrid conditional variational autoencoder model for personalised top-n recommendation. In *Proceedings of ICTIR*, 2020.

[31] Teng Xiao, Shangsong Liang, Weizhou Shen, and Zaiqiao Meng. Bayesian deep collaborative matrix factorization. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, 2019.
[32] Longqi Yang, Eugene Bagdasaryan, Joshua Gruenstein, Cheng-Kang Hsieh, and Deborah Estrin. Openrec: A modular framework for extensible and adaptable recommendation algorithms. In *WSDM*, pages 664–672, 2018.
[33] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. A dynamic recurrent model for next basket recommendation. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 729–732, 2016.
[34] Shuai Zhang, Yi Tay, Lina Yao, Bin Wu, and Aixin Sun. Deeprec: An open-source toolkit for deep learning based recommendation. *arXiv preprint arXiv:1905.10536*, 2019.

Appendix A : EROSKI Dataset Tables and Fields

CLIENTS table structure (21 attributes)

- ID_CLIENTE: Client id,
- TIPO_CLIENTE_ORO: Type of gold client
- FLG_CLIENTE_APP: Flag if the client is an app client or not,
- FLG_CLIENTE_WEB: Flag if the client is a web client or not,
- FLG_CLIENTE_NUTRICIONAL: Customer shows interest in healthy products
- FRANJA_GASTO_ORO_INICIAL: Initial Range of expenditure
- POSIBLE_VALOR_ORO: Percentage indicating the discount given to the customer for being a gold customer
- CLIENTE_1000_ORO: Flag indicating whether the client is 1000 Oro or not
- FRANJA_GASTO_ORO_ACTUAL: Current Range of expenditure
- TIPO_MADUREZ: Type of maturity of the client
- DESC_SEG_C_CLIENTE: Description of the type of maturity of the client
- DESC_SEG_G_FIDELIDAD: Segmentation of the customer according to his loyalty
- DESC_INTERES_AHORRO: Segmentation of the customer according to his interest in promotions
- DESC_INTERES_FRESCOS: Segmentation of the customer according to his interest in fresh food
- DESC_INTERES_LOCAL: Segmentation of the customer according to his interest in local food
- DESC_INTERES_SALUD: Segmentation of the customer according to his interest in healthy food
- DESC_INTERES_SALUD_DETALLE: additional detail on which type of healthy food the customer is interested in
- DESC_MISION_COMPRA: description of the purchase mission of the customer
- DESC_SEG_SEC: segment description
- DESC_SEG_SOCIODEMO: Socio-demographic segment of the client.
- COD_LOC: preferred store

TICKETS (36 attributes)

- ID_CLIENTE: Client id,
- COD_LOC: Store's localization id,
- DIA: Day,
- COD_CAJA: Till id,
- NUM_TICKET: Ticket number (id),
- NUM_LINEA: Line number (id),
- COD_TIPO_MOVIM: Movement type,
- HORA_EMISION: Timestamp of tickets emission,
- COD_TIPOMARCA_HIST: Type of brand of the product
- COD_F_PAGO_DET -> M_FORMA_PAGO: Type of payment procedure,
- UNID_VENTA_TARIFA: Total amount of items sold in tariff's type,
- UNID _VENTA_OFERTA: Total amount of items sold in offer's type,
- UNID _VENTA_COMPETE: Total amount of items sold in competence's type,
- UNID _VENTA_LIQUID: Total amount of items sold in liquidation's type,
- UNID _VENTA_CAMPANA: Total amount of items sold in campaign's type,
- IMP_VENTA_TARIFA: Total economic amount of the items sold by tariff's type,
- IMP_VENTA_OFERTA: Total economic amount of the items sold by offer's type,
- IMP_VENTA_COMPETE: Total economic amount of the items sold by competence's type,
- IMP_VENTA_LIQUID: Total economic amount of the items sold by liquidation's type,
- IMP_VENTA_CAMPANA: Total economic amount of the items sold by campaign's type,
- IMP_DTO_CONSUMER: Discount amount applied for using VISA Eroski,
- IMP_DTO_TRAVEL: Discount amount applied for using loyalty card Travel Club,

- IMP_DTO_COUPON: Discount amount applied for the usage of coupons,
- IMP_DTO_CUOTA: Discount amount applied for being member of EROSKI Club,
- IMP_DTO_ONSITE: Discount amount applied after redemption of loyalty Travel points,
- IMP_DTO_OTROS: Other discounts,
- IMP_DTO_VALE: Amount of discounts coming from the redemption of a supplier coupon,
- IMP_CONSUMO_RAP: Special discount applied in the shop,
- COD_ART: Article's id,
- FLG_TECLA: information about whether the product has been sold by a direct key or not
- ANO_OFERTA: year of the offers applied to the order
- COD_OFERTA: offer code
- COD_TIPO_CENTRO: type of shop (primary/secundary)
- FLG_SCANNER: has the product been scanned during the purchase (Y/N)
- IMP_PVP_TARIFA: amount of the order if all of the items had been charged to the customer with catalogue prices

CENTERS structure (55 attributes)

- COD_LOC: Store's localization id,
- COD_PROVIN: Province id,
- DESC_LOC: Center's description,
- DESC_PROVIN: Province's name,
- FLG_PLATAF : Indicator of distribution platform,
- FEC_MODIF: Date of last modification,
- COD_ZONA: Zone id,
- DESC_ZONA: Zone description,
- COD_REGION: Region id,
- DESC_REGION: Region description,
- COD_AREA: Area id,
- DESC_AREA: Area's description,
- COD_ENSENA: Type of center id,
- DESC_ENSENA: Type of center description (Eroski City, Eroski Center…),
- COD_NEGOCIO: Store's id,
- DESC_NEGOCIO: Store's type,
- COD_SOCIEDAD: Type of company,
- DESC_SOCIEDAD: Company's description,
- COD_GAMA_OBLIG: Code of mandatory catalogue,
- COD_FINANZIA: financing code,
- DESC_DIRECCION: address,
- DESC_POBLACION: location,
- FLAG_CUOTA: quota flag,
- FEC_INI_LOC: opening date,
- FEC_FIN_LOC: closing date,
- NUM_CAJAS: number of boxes,
- NUM_M2: squared meters of the store,
- NUM_M_LINEA: linear meters,
- COD_LOC_AME: store code in AME system,
- COD_TP_LOC: type of location,
- DESC_TP_LOC: description of the type of location,
- COD_LOC_PADRE: father location code,
- COD_MUNICIPIO: location code,
- COD_TP_POTENCIAL: type of potential code,
- FEC_ULT_APERTURA : last opening date,
- COD_POSTAL: zip code,

- COD_AGR_IMP: grouping code,
- FLG_CECO_MODELO_COSTES: cost model flag,
- LATITUD: latitude,
- LONGITUD: longitude,
- COD_ISLA: ISLA code,
- FLG_LEAN: lean flag,
- FLG_TRANSFORMADO: transformed flag,
- FLG_PUESTA_PUNTO_PLUS: tunning flag,
- COD_NIVEL_ESTR_LOC: code of local structure of sales of the center,
- COD_N1: code of the level 1 of the structure of sales of the center,
- DES_N1: description of the level 1 of the structure of sales of the center,
- COD_N2: code of the level 2 of the structure of sales of the center,
- DES_N2: description of the level 2 of the structure of sales of the center,
- COD_N3: code of the level 3 of the structure of sales of the center,
- DES_N3: description of the level 3 of the structure of sales of the center,
- COD_N4: code of the level 4 of the structure of sales of the center,
- DES_N4: description of the level 4 of the structure of sales of the center,
- COD_N5: code of the level 5 of the structure of sales of the center,
- DES_N5: description of the level 5 of the structure of sales of the center,

PRODUCTS structure (79 attributes)

- COD_ART: product id,
- DESC_ART: product description,
- FLG_TECLA: exists a direct key to sell the product or not,
- COD_TIPOMARCA: type of brand code,
- DESC_TIPOMARCA: description of the type of brand code,
- COD_N1_PPAL: Area's id,
- DESC_N1: Area's description,
- COD_N2_PPAL: Section's id,
- DESC_N2: Section's description,
- COD_N3_PPAL: Category's id,
- DESC_N3: Category's description,
- COD_N4_PPAL: Subcategory's id,
- DESC_N4: Subcategory's description,
- COD_N5_PPAL: Segment's id,
- DESC_N5: Segment's description,
- FEC_INI_ART: Article start time,
- FEC_FIN_ART: Article finishes time,
- COD_FORMATO: Format id (KG, Gr, Unities…),
- COD_MARCA: Brand's id,
- COD_EAN: EAN code,
- COD_TALLA: Size code,
- DESC_TALLA: Size code description,
- COD_COLOR: Colour code,
- DESC_COLOR: Colour code description,
- COD_PACK : Number of items per pack,
- COD_BLOQUEO: has the product blocked for the sales?,
- COD_ENS_EROSKI: commercial codification in the Hypermarket,
- COD_ENS_CONSUM: commercial codification in the SUPERmarket,
- COD_TIPO_FORMATO: unit of measurement (related to COD_FORMATO),
- COD_ART_PRIM: father product code,
- COD_TIPO_MARCA2: code of EROSKI Brand (only for products belonging to a EROSKI brand)),

- DESC_TIPO_MARCA2: description of EROSKI Brand (only for products belonging to a EROSKI brand)),
- FEC_ULT_BLOQ: date on which the product was blocked for the sales,
- COD_PORCI_CONS: product has info for the consumer related to the number of portions,
- DESC_PORCI_CONS: indicator about whether the product has a description for the portions,
- CC_CAPRABO: Comercial code of CAPRABO,
- COD_CATEGORI_HIP: Category code hypermarket,
- DESC_CATEGORI_HIP: Description of the Hypermarket Category,
- COD_CATEGORI_SUP: Category code supermarket,
- DESC_CATEGORI_SUP: Description of the supermarket Category,
- COD_SENSIBI_HIP: SENSIBI code hypermarket,
- DESC_ SENSIBI HIP: Description of the SENSIBIcode of the hypermarket,
- COD_ SENSIBI SUP: Category code supermarket,
- DESC_ SENSIBI SUP: Description of the SENSIBIcode of the supermarket,
- FLG_COMPRA: indicator about whether the product is for purchasing,
- FLG_VENTA: indicator about whether the product is for sales,
- COD_FAMILIA: family of the product,
- DESC_FAMILIA: description of the family of the product,
- COD_AMBITO_EROSKI: Scope code of the product in the hypermarkets,
- DESC_AMBITO_EROSKI: Description of the scope of the product in the hypermarkets,
- COD_AMBITO_CONSUM: Scope code of the product in the supermarkets,
- DESC_AMBITO_CONSUM: Description of the scope of the product in the supermarkets,
- COD_CODMARCA: brand code (related to COD_MARCA)
- FLG_MMPP: Does the product belong to a EROSKI brand?,
- COD_POSICION_MARCA: Maker brand / EROSKI Brand code,
- DESC_POSICION_MARCA: Description of the code of maker Brand / EROSKI Brand code,
- FLG_SALUD_BIENESTAR: health indicator,
- FLG_INNOVACION: innovation indicator,
- FLG_GAMA_TURISTICA: tourism product,
- FLG_PODER_ADQUISITIVO: indicator about product for customer with a high purchasing power,
- FLG_BLOQ_DEFINITIVO: Product definitely blocked,
- COD_SUBMARCA: sub-brand code,
- DESC_SUBMARCA: sub-brand description
- FLG_GAMA_LOCAL: local product,
- FLG_GAMA_REGIONAL: regional product,
- FLG_PESO_SGA: flag product by weight,
- FLG_LIQUIDABLE: flag payable,
- FLG_EXDEPRECIACION: depreciation flag,
- COD_TP_ART: product type,
- DESC TIPO_ARTICULO: description of the product type,
- CANTIDAD: number of ítems per lot,
- FEC_LANZAM: launch date,
- PORC_IVA: VAT rate,
- COD_PROVR_GEN: code of generic supplier,
- COD_PROVR_TRABAJO: code of work supplier,
- NOMBRE: name of the work supplier,
- PESO: weight (in grams),
- PESO_NETO: net weight (in grams),
- VOLUMEN: volume (in cm3)

Appendix B: VBCAR Paper

Variational Bayesian Context-aware Representation for Grocery Recommendation

Zaiqiao Meng, Richard McCreadie, Craig Macdonald and Iadh Ounis
University of Glasgow, Scotland, UK
(firstname.lastname)@glasgow.ac.uk

ABSTRACT

Grocery recommendation is an important recommendation use-case, which aims to predict which items a user might choose to buy in the future, based on their shopping history. However, existing methods only represent each user and item by single deterministic points in a low-dimensional continuous space. In addition, most of these methods are trained by maximizing the co-occurrence likelihood with a simple Skip-gram-based formulation, which limits the expressive ability of their embeddings and the resulting recommendation performance. In this paper, we propose the Variational Bayesian Context-Aware Representation (VBCAR) model for grocery recommendation, which is a novel variational Bayesian model that learns the user and item latent vectors by leveraging basket context information from past user-item interactions. We train our VBCAR model based on the Bayesian Skip-gram framework coupled with the amortized variational inference, so that it can learn more expressive latent representations that integrate both the non-linearity and Bayesian behaviour. Experiments conducted on a large real-world grocery recommendation dataset show that our proposed VBCAR model can significantly outperform existing state-of-the-art grocery recommendation methods.

KEYWORDS

Context-Aware, Recommender Systems, Variational Bayesian, Skip-gram, Grocery Recommendation

ACM Reference Format:
Zaiqiao Meng, Richard McCreadie, Craig Macdonald and Iadh Ounis. 2019. Variational Bayesian Context-aware Representation for Grocery Recommendation. In „ ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/xxxxxxx.xxxxxxx

1 INTRODUCTION

Recommender systems that use historical customer-product interactions to provide customers with useful suggestions have been of interest to both academia and industry for many years. Various matrix completion-based methods [4, 12, 13] have been proposed to predict the rating scores of products (or items) for customers (or users). Recently, many grocery recommendation models [3, 15, 16] were proposed that target grocery shopping use-cases. In real grocery shopping platforms, such as Amazon and Instacart, users' interactions with items are sequential, personalized and more complex than those represented by a single rating score matrix. Thus effective recommendation models for this use-case are designed to learn representations of users and items so that contextual information, such as basket context [16] and time context [9], are captured within the learned representations, which results in increased recommendation performance. In the grocery shopping domain, prod2vec [3] and triple2vec [16] are two state-of-the-art models that learn latent representations capturing the basket context, based on the Skip-gram model for grocery recommendation. In these models, both the user's general interest (which items the user likes) and the personalized dependencies between items (what items the user commonly includes in the same basket) are encoded by the embeddings of users and items. Furthermore, when combined with negative sampling approaches [11], these Skip-gram-based models are able to scale to very large shopping datasets. Meanwhile, through the incorporation of basket contextual information during representation learning, significant improvements in grocery recommendation have been observed [3, 16].

However, these representation models still have several defects: (1) they represent each user and item by single deterministic points in a low-dimensional continuous space, which limits the expressive ability of their embeddings and recommendation performances; (2) their models are simply trained by maximizing the likelihood of recovering the purchase history, which is a point estimate solution that is more sensitive to outliers when training [1].

To alleviate the aforementioned problems, we propose a *Variational Bayesian Context-Aware Representation* model, abbreviated as *VBCAR*, which extends the existing Skip-gram based representation models for grocery recommendation in two directions. First, it jointly models the representation of users and items in a Bayesian manner, which represents users and items as (Gaussian) distributions and ensures that these probabilistic representations are similar to their prior distributions (using the variational auto-encoder framework [5]). Second, the model is optimized according to the amortized inference network that learns an efficient mapping from samples to variational distributions [14], which is a method for efficiently approximating maximum likelihood training. Having inferred the representation vectors of users and items, we can calculate the preference scores of items for each user based on these two types of Gaussian embeddings to make recommendations. Our contributions can be summarized as follows [1]:

(1) We propose a variational Bayesian context-aware representation model for grocery recommendation that jointly learns probabilistic user and item representations while the item-user-item triples in the shopping baskets can be reconstructed.
(2) We use the amortized inference neural network to infer the embeddings of both users and items, which can learn more

[1]The code of our VBCAR model is publicly available from: https://github.com/mengxiqiao/VBCAR

ACM RecSys, Workshop on Context-Aware Recommender Systems

Appendix C: VBCAR-S Paper

Noname manuscript No.
(will be inserted by the editor)

Variational Bayesian Representation Learning for Grocery Recommendation

Zaiqiao Meng · Richard McCreadie · Craig Macdonald · Iadh Ounis

Received: date / Accepted: date

Abstract Representation learning has been widely applied in real-world recommender systems for capturing features of both users and items. Existing grocery recommender methods only represent each user and item by single deterministic points in a low-dimensional continuous space, which limit the expressive ability of their embeddings, resulting in recommendation performance bottlenecks. In addition, existing representation learning methods for grocery recommendation only consider the items (products) as independent entities, neglecting their other valuable side information, such as the textual descriptions and categorical data of items. In this paper, we propose the Variational Bayesian Context-Aware Representation (VBCAR) model for grocery recommendation, which is a novel variational Bayesian model that learns distributional representations of users and items by leveraging basket context information from historical interactions. Our VBCAR model is also extendable to leverage side information by encoding contextual features into representations based on the inference encoder. We conduct extensive experiments on three real-world grocery datasets to assess the effectiveness of our model as well as the impact of different construction strategies of item side information. Our results show that our VBCAR model outperforms the current state-of-the-art

Zaiqiao Meng
School of Computing Science, University of Glasgow, Scotland, UK
E-mail: Zaiqiao.Meng@glasgow.ac.uk

Richard McCreadie
School of Computing Science, University of Glasgow, Scotland, UK
E-mail: Richard.McCreadie@glasgow.ac.uk

Craig Macdonald
School of Computing Science, University of Glasgow, Scotland, UK
E-mail: Craig.Macdonald@glasgow.ac.uk

Iadh Ounis
School of Computing Science, University of Glasgow, Scotland, UK
E-mail: Iadh.Ounis@glasgow.ac.uk

Appendix D: BetaRecsys Paper

BETA-Rec: Build, Evaluate and Tune Automated Recommender Systems

Zaiqiao Meng
Richard McCreadie
Craig Macdonald
Iadh Ounis
Siwei Liu
Yaxiong Wu
Xi Wang
zaiqiao.meng@gmail.com
University of Glasgow

Shangsong Liang
Yucheng Liang
Guangtao Zeng
Junhua Liang
Sun Yat-sen Univeristy

Qiang Zhang
University College London

ABSTRACT

The field of recommender systems has rapidly evolved over the last few years, with significant advances made due to the in-flux of deep learning techniques. However, as a result of this rapid progress, escalating barriers-to-entry for new researchers is emerging. In particular, state-of-the-art approaches have fragmented into a large number of code-bases, often requiring different input formats, pre-processing stages and evaluating with different metric packages. Hence, it is time-consuming for new researchers to reach the point of having both an effective baseline set and a sound comparative environment. As a step towards elevating this problem, we have developed BETA-Rec, an open source project for Building, Evaluating and Tuning Automated Recommender Systems. BETA-Rec aims to provide a practical data toolkit for building end-to-end recommendation systems in a standardized way. It provides means for dataset preparation and splitting using common strategies, a generalized model engine for implementing recommender models using Pytorch with 9 models available out-of-the-box, as well as a unified training, validation, tuning and testing pipeline. Furthermore, BETA-Rec is designed to be both modular and extensible, enabling new models to be quickly added to the framework. It is deployable in a wide range of environments via pre-built docker containers and supports distributed parameter tuning using Ray. In this demo, we will illustrate the deployment and use of BETA-Rec for researchers and practitioners on a number of standard recommendation datasets. The source code of the project is available at github: https://github.com/beta-team/beta-recsys.

CCS CONCEPTS

• **Information systems** → **Collaborative filtering**.

KEYWORDS

Recommender Systems, Framework, Open-source, Toolkit

ACM Reference Format:
Zaiqiao Meng, Richard McCreadie, Craig Macdonald, Iadh Ounis, Siwei Liu, Yaxiong Wu, Xi Wang, Shangsong Liang, Yucheng Liang, Guangtao Zeng, Junhua Liang, and Qiang Zhang. 2020. BETA-Rec: Build, Evaluate and Tune Automated Recommender Systems. In *Fourteenth ACM Conference on Recommender Systems (RecSys '20), September 22–26, 2020, Virtual Event, Brazil.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3383313.3411524

1 INTRODUCTION

Recommender systems that suggest items of interest to users based on available information such as purchases and interactions histories have been the subject of intensive research by both industry and academia in recent years [3–5, 12]. In particular, deep learning techniques have achieved tremendous success in recommender systems, leading to large and significant improvements in performance being reported [3, 4, 12].

However, partially as a result of this intensive and rapid progress, there are now many barriers-to-entry in the recommendation field for new researchers, as well as increasing challenges when comparing different works from the literature. In particular, current challenges include: 1) the implementations are fragmented across different code repositories and often do not function out-of-the-box; 2) the reported performances across works are often not comparable even when reported on the same dataset and with the same metrics;[1] 3) the usage of different evaluation toolkits is problematic as subtle differences in metric implementations lead to different reported performances; 4) the inconsistency among various tuning strategies of models (notably the omission of baseline tuning) leads to unfair comparisons. Indeed, recent works [1, 6, 7, 11] have demonstrated that, without properly tuning model hyperparemeters, existing state-of-the-art deep learning baselines cannot even consistently outperform a non-neural linear ranking models. Hence, there is a clear need for platforms that provide a common recommendation methodology and pipeline, enabling model comparison across datasets, metrics, and baselines in a sound and fair manner.

[1] Since data pre-processing and splitting techniques differ each other.

Appendix E: T-VBR Paper

Learning Temporal Representations for Grocery Recommendation

Anonymous Author(s)

ABSTRACT

This paper focuses on the recommendation of items in a grocery shopping scenario, where users purchase multiple products in sequential baskets. It is commonly acknowledged that both users' interests and products' popularities vary over time. However, few prior grocery recommendation methods account for such temporal patterns, instead, only presenting each user and product using static low dimensional vectors. In contrast, we propose a new model: Temporal Variational Bayesian Representation (T-VBR) for grocery recommendation, which is able to encode temporal patterns to improve effectiveness. T-VBR is designed under the temporal variational Bayesian framework, and it learns temporal Gaussian representations for users and items by encoding information from: 1) the basket context; 2) item side information; and 3) the temporal context from past user-item interactions. T-VBR is trained using sampled triples of users with two items bought together in baskets during different time windows, via a Bayesian Skip-gram model based on a temporal variational auto-encoder. Experiments conducted on four public grocery shopping datasets show that our proposed T-VBR model can significantly outperform the existing state-of-the-art grocery recommendation methods, and can learn more expressive representations that effectively capture the temporal information.[1]

1 INTRODUCTION

Recommender systems that learn from the customers' historical interactions with items (e.g. movies and news) with the goal of providing more relevant item suggestions have been of interest to both academia and industry for many years. Various collaborative filtering-based methods [12, 31, 36, 44] have been proposed to make automatic predictions about the interests of users by learning preferences or tastes from their past interactions. Among these, latent variable models (a.k.a. latent representation learning models) [12, 27, 43, 44] have gained substantial attention, due to their wide applicability in capturing various useful contextual information from the training datasets, as well as their state-of-the-art effectiveness when modelling the hidden preferences or tastes of users.

Our paper focuses on the task of grocery recommendation, which is typically addressed by learning low-dimensional latent representations of users and items based on the purchase history, such that these representations can then be used to predict which products a user might choose to buy in the future. With the increasing volumes of purchase history data, such as the transaction logs and product metadata that are collected from a number of retailers (e.g. Walmart and Tesco) and online grocery shopping platforms (e.g. Amazon[2] and Instacart[3]), there is a growing body of data and research on grocery recommendation [10, 28, 43, 44, 49]. Among the grocery shopping datasets, each user's interactions are commonly stored in timestamped logs, indicating when items were purchased. Moreover, item purchases are grouped into distinct shopping baskets containing multiple items that were bought together by a given user. Capturing basket dependencies between each user and multiple items have been proven to be a crucial modelling dimension for advancing the representation learning models for grocery recommendation [10, 21, 28, 44]. For example, the Triple2vec [44] model encapsulates the basket context via sampling triples (i.e. (user, item, item) triplets) from the purchase history to capture the personalized dependencies between items within the same basket.

It is commonly acknowledged that user preferences and product popularities drift over time. However, most of the existing representation learning methods for recommender systems only represent each user and product by static deterministic points in a low-dimensional continuous space [10, 21, 44], which is unable to capture either changes in the user's preferences over time or the varying popularity of items over time. This is because all user interactions are considered equal when training, meaning that old (and potentially out-dated) interactions from previous years are typically considered just as relevant as much more recent interactions. We argue that these temporal factors influence purchase decisions and hence should be accounted for within the recommendation model. Moreover, most of these existing methods cannot embed the item side information (e.g. product name and category information) into the representations when learning, meaning that the intrinsic similarities between different products cannot be captured, which are known to be important knowledge about the products [6, 32].

To alleviate these issues, we propose a new *Temporal Variational Bayesian Representation* model, denoted as *T-VBR*, which is a novel variational Bayesian model that learns the temporal representations of users and items for grocery recommendation. Our model infers the Gaussian embeddings of both users and items and represents them by means of Gaussian distributions. Given the natural properties of Gaussian distributions, T-VBR innately represents their uncertainties with the corresponding variances of the inferred embeddings. In particular, we explicitly model the evolution of users and products over time through a *smoothing neural network*, such that the users' preferences over time, as well as the items' popularity patterns can be effectively captured. T-VBR is also extensible, and hence is able to leverage item side information to capture relationships between items, leading to increased effectiveness.

The contributions of this paper can be summarized as follows:
(1) We extend the existing state-of-the-art Skip-gram based representation models for grocery recommendation into the temporal variational Bayesian auto-encoder framework.
(2) We propose the temporal representation learning model, T-VBR, which is a Bayesian model that captures dynamic purchase behaviour, learns temporal Gaussian latent representations for both users and items, and supports the addition of side information.

[1] Our source code is available at https://github.com/anonymcodes/t-vbr
[2] www.amazon.com
[3] www.instacart.com

Appendix F: GCN Paper

Graph Neural Pre-training for Recommendation with Side Information

Anonymous Author(s)

ABSTRACT

Leveraging the side information associated with entities (i.e. users and items) to enhance the performance of recommendation systems has been widely recognized as an important modelling dimension. While many existing approaches focus on the *integration scheme* to incorporate entity side information – by combining the recommendation loss function with an extra context-aware loss utilizing the side information – in this paper, we propose instead a novel *pre-training scheme* for leveraging the side information. In particular, we first pre-train a representation model using the side information of the entities, and then fine-tune it using an existing general representation-based recommendation model. Specifically, we propose two models, i.e. **GCN-P** and **COM-P**, by considering the entities and their relations constructed from side information as two different types of graphs respectively, to pre-train entity embeddings using graph neural networks. For the **GCN-P** model, two single-relational graphs are constructed from all the users' and items' side information respectively, to pre-train entity representations by using the Graph Convolutional Networks. For the **COM-P** model, two multi-relational graphs are constructed to pre-train the entity representations by using the Composition-based Graph Convolutional Networks. Experimental results of our pre-training models fine-tuned under four representative recommender models, i.e. MF, NCF, NGCF and LightGCN, show that effectively pre-training embeddings with both user's and item's side information can significantly improve these original models in terms of both the effectiveness and stability.

ACM Reference Format:
Anonymous Author(s). 2021. Graph Neural Pre-training for Recommendation with Side Information. In *WWW '21: The Web Conference 2021, April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The goal of recommender systems is to assist users in filtering out non-relevant information and selecting a personalized set of interesting items to maximize the users' satisfaction. Modern recommendation models achieve this goal by learning representation vectors (i.e. embeddings) of the two entities (i.e. users and items) that capture the users' interests and items' attractiveness [52, 53], so that the learned embeddings can be used to accurately predict which items a user might choose in the future, e.g., by computing the dot product or a multilayer perceptron (MLP) [31] of the users' and items' embeddings. Typically, the users' interests and items' attractiveness are reflected through ratings, clicking or other user-item interactive behaviours. However, in real-world data, the user-item interactive behaviours are in general highly sparse. Therefore, a number of context-aware recommendation models [3, 18, 19, 25, 36] have been designed to address this issue by integrating rich side information from users and items, such as the users' age groups and the items' textual descriptions, demonstrating promising performance improvements over models solely using interactive behaviour data.

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

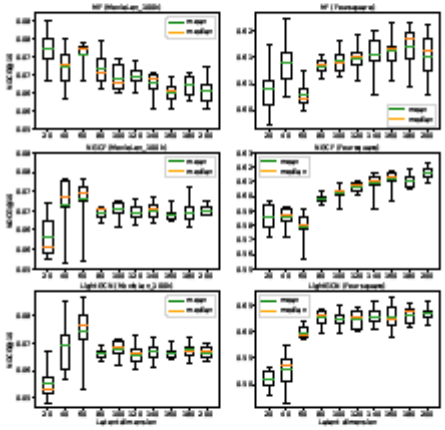


Figure 1: Box-and-whisker diagrams for the NDCG performances of the MF [31], NGCF [41] and LightGCN [8] models on the Movielens_100k and Foursquare datasets. Large variances can be observed over 50 runs with different random seeds, demonstrating the instabilities of these three models.

To leverage the side information associated with users and items, many approaches have been proposed, most of which mainly focus on encoding the side information during the training process with user-item interactions [3, 19, 25, 28]. Most approaches follow the conventional *integration scheme* that optimizes a loss function consisting of two components, i.e. the recommendation loss and an additional context-aware loss leveraging the side information [3, 19, 21, 40]. However, such integration scheme may fail in scenarios where the two loss components strongly disagree during training, resulting in reduced effectiveness. Moreover, although there have been many proposed powerful neural network-based

Appendix G: Data Splitting Strategies Analysis Paper

Exploring Data Splitting Strategies for the Evaluation of Recommendation Models

Zaiqiao Meng
University of Glasgow
zaiqiao.meng@glasgow.ac.uk

Richard McCreadie
University of Glasgow
richard.mccreadie@glasgow.ac.uk

Craig Macdonald
University of Glasgow
craig.macdonald@glasgow.ac.uk

Iadh Ounis
University of Glasgow
iadh.ounis@glasgow.ac.uk

ABSTRACT

Effective methodologies for evaluating recommender systems are critical, so that different systems can be compared in a sound manner. A commonly overlooked aspect of evaluating recommender systems is the selection of the data splitting strategy. In this paper, we both show that there is no standard splitting strategy and that the selection of splitting strategy can have a strong impact on the ranking of recommender systems during evaluation. In particular, we perform experiments comparing three common data splitting strategies, examining their impact over seven state-of-the-art recommendation models on two datasets. Our results demonstrate that the splitting strategy employed is an important confounding variable that can markedly alter the ranking of recommender systems, making much of the currently published literature non-comparable, even when the same datasets and metrics are used.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Test collections.*

KEYWORDS

Recommender Systems, Spliting Strategy, Model Evaluation, Leave-one-out, Temporal Split

ACM Reference Format:
Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2020. Exploring Data Splitting Strategies for the Evaluation of Recommendation Models. In *Fourteenth ACM Conference on Recommender Systems (RecSys '20), September 22–26, 2020, Virtual Event, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3383313.3418479

1 INTRODUCTION

Recommender systems (RecSys) have been subject to extensive research examining how to most effectively find items of interest that a user would like to buy or consume within large datasets. Recommendation spans a range of domain-specific sub-tasks (such as grocery recommendation [24] and venue recommendation [11]) and different scenarios (such as session-based recommendation [29] and sequential recommendation [14]). Many approaches have been proposed to solve these tasks over the last two decades, among which neural network-based recommendation models are currently very popular, due to their high effectiveness and adaptability to different sub-tasks and scenarios [29]. As the recommender systems field matures, advances in performance naturally become more incremental, leading to smaller increases in model effectiveness. This places more strain on the evaluation methodology's ability to distinguish between systems with similar performance, as researchers and practitioners chase ever smaller performance gains.

With the current influx of very similar neural network-based recommendation models being published, there needs to be increased emphasis placed on eliminating confounding factors that can lead to uncertainty during evaluation, otherwise it will be impossible to confidently determine whether gains are truly being made. In the Information Retrieval (IR) domain, standardization efforts such as TREC, and other evaluation initiatives like NTCIR, CLEF and FIRE laid down guidelines on what constitutes a sound evaluation methodology in that domain. However, standardization efforts in the recommender systems domain appear to have been less successful, with most current research papers reporting a wide-range of distinct combinations of datasets, metrics, baselines and data splitting strategies, which makes it difficult to measure progress in the field [4, 17, 29].

Standardization of datasets and baselines within the RecSys community is an on-going process. In particular, while recent works [4, 17, 18] tend to share similar baseline models (e.g. some variant of BPR [15]) and in some cases may share datasets, there are no commonly agreed-upon standards for important aspects that can impact performance such as data preparation. Indeed, a recent study [18] found that suitably tuned baselines could in some cases match or out-perform state-of-the-art approaches, highlighting the importance of hyper-parameter tuning and standardized benchmarks [3, 4, 17, 18] to enable fair comparisons and reproducibility. However, beyond these known issues, one factor that is often overlooked (and typically is not detailed sufficiently in prior works to be reproducible) is the *data splitting strategy* employed. This is how a recommendation dataset is split into training, validation and testing sets. In the IR domain, this split is usually explicitly defined by the test collection (i.e. training and test query sets). However, there is often no equivalent guidance in RecSys scenarios, leading to a wide range of strategies for dividing any particular dataset being employed and reported [14, 20, 21, 29]. Hence, it is natural to ask "does the data splitting strategy matter?", because if it does, much of the recently published work is not comparable, even when performances are

ACM ISBN 978-1-4503-7583-2/20/09.
https://doi.org/10.1145/3383313.3418479

Appendix H: Insurance Dataset Tables and Fields

ana

id_univoco_anagrafica	string	Flow unique identifier: REGISTRY
id_univoco_master	string	Subject unique identifier
codice_fiscale	string	Registry type (P = person, N = company)
tipo_anagrafica	string	Surname / company name
cognome	string	Gender (M=male, F=female, N=company)
nome	string	Public Administration (YES/NO)
sesso		
pubblica_amministrazione		

Ana_ptf

codice_fiscale	string	Subject unique identifier
idpolizza	string	Policy unique identifier
ruolo	string	Subject role
cognome	string	Surname / company name
nome	string	Name

ana_sin

id_univoco_anagrafica	string	Flow unique identifier: REGISTRY
id_univoco_master	string	
codice_fiscale	string	Subject unique identifier
idsinistro	string	Claim unique identifier
ruolo	string	Subject role
cognome	string	Surname / company name
nome	string	Name

ana_vei

codice_fiscale	string	Subject unique identifier

targa		License plate
cognome		Surname / company name
nome		

anaage

codice_fiscale	string	Subject unique identifier
agenzia	string	Agency ID
descrizione	String	Description

anaaia

codice_fiscale	string	Subject unique identifier
codice_anomalia	string	Anomaly identifier

anabds

codice_fiscale	string	Subject unique identifier
bds	bigint	
p1	bigint	
p2	bigint	
p3	bigint	
p4	bigint	
p5	bigint	
p6	bigint	

anacci

codice_fiscale	string	Subject unique identifier
tipo_assicurazione	string	Insurance type
ente_comunicante	string	Communicating entity
data_infortunio	string	Accident date
luogo_infortunio	string	Accident place

lesione_1	string	Injury nr 1
lesione_2	string	Injury nr 2
lesione_3	string	Injury nr 3
lesioni_ulteriori	string	Other Injuries
percentuale_inabilita	double	Disability percentage
data_decesso	string	Date of death

anacnt

codice_fiscale	string	Subject unique identifier
tipo_contatto	string	Contact type
contatto	string	Contact

anacontatori

codice_fiscale	string	Subject unique identifier
portafoglio	bigint	Total insurance policies number
portafoglio_auto	bigint	Auto insurance policies number
portafoglio_re	bigint	Elementary branches insurance policies number
portafoglio_vita	bigint	Life insurance policies number
portafoglio_cauzioni	bigint	Deposits policies number
sinistri_aperti	bigint	Open claims number
veicoli_attivi	bigint	Insured vehicles number

anafid

codice_fiscale	string	Subject unique identifier
tipo_soggetto	string	Subject type

anaind

codice_fiscale	string	Subject unique identifier
comune	string	Subject main address, city
provincia	string	Subject main address, province
nazione	string	Subject main address, country

flag_principale	string	

analnkcnt

tipo_contatto	string	Contact type
contatto	string	Contact
codice_fiscale_a	string	Subject unique identifier a
codice_fiscale_b	string	Subject unique identifier b

crvdlnk

partita_iva	string	VAT number
codice_fiscale	string	Subject unique identifier
denominazione	string	Subject / company name
cognome	string	Surname / company name
nome	string	

crvdsem

codice_fiscale	string	Subject unique identifier
semaforo	string	Traffic light

ptf

idpolizza	string	Policy unique identifier
agenzia	string	Agency ID
descrizione_agenzia	string	Agency description
provincia_agenzia	string	Province of the agency
ramo	string	Policy branch
tipo_polizza	string	Policy type (Individual / Collective)
stato_polizza	string	Policy state (Active/ Canceled / Suspended)
stato_coass	string	No coinsurance / Our delegation / Delegation
codice_prodotto	string	Product Code-Product Description
prodotto	string	Product

data_effetto	string	Policy effective date
data_scadenza	string	Policy effective deadline
premio	double	Policy premium

sin

idsinistro	string	Claim unique identifier
idpolizza	string	Policy unique identifier
data_sinistro	string	Claim occurrence date (Format: YYYY-MM-DD)
ora_sinistro	string	Claim occurrence time (Format: HH: MM)
tipo_sinistro	string	Accident type (RCA / ARD / RE)
tipo_danno	string	Damage reported type (1 = THINGS / 2 = PEOPLE / 3 = MIXED)
tipo_gestione	string	Claim management type
flag_autorita_presenti	string	Authority flag present (S - Yes, N - No)
stato_sinistro	string	Accident status
data_definizione_sinistro	string	Claim closing date (Format: YYYY-MM-DD)
numero_veicoli	bigint	Vehicles involved number
comune	string	Claim occurrence address, city
provincia	string	Claim occurrence address, province
pagato	double	Paid
riservato	double	Reserved
data_denuncia	string	Claim complaint date (YYYY-MM-DD)

sinantifrode

idsinistro	string	Claim unique identifier
semaforo	string	Traffic light
verifica	string	Verification
note_verifica	string	Verification notes
approfondimento	string	Deepening
note_approfondimento	string	Deepening notes
antifrode	string	Anti fraud

sinantifrodectl

idsinistro	string	Claim unique identifier
controllo	string	Check

vei

targa	string	License plate
marca	string	Vehicle brand
modello	string	vehicle model
tipo_veicolo	string	Vehicle type
tipo_targa	string	License plate type
data_immatricolazione	string	Matriculation date

vei_ptf

targa	string	Vehicle identifier
idpolizza	string	Policy unique identifier

vei_sin

Targa	string	Vehicle identifier
Idsinistro	string	Claim unique identifier

Open datasets

OD_Caratt_Geo_Morf_Comuni (OD Geo Morphological Municipalities Features)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
comune	nvarchar	city
aree a pericolosità idraulica bassa (kmq)	float	low hydraulic hazard areas
aree a pericolosità idraulica media (kmq)	float	medium hydraulic hazard areas
aree a pericolosità idraulica elevata (kmq)	float	high hydraulic hazard areas
area di attenzione pai - aa (kmq)	float	attention area PAI

area a pericolosità da frana pai moderata - p1 (kmq)	float	moderate landslide hazard area PAI - p1
area a pericolosità da frana pai media - p2 (kmq)	float	medium landslide hazard area PAI - p2
area a pericolosità da frana pai elevata - p3 (kmq)	float	high landslide hazard area PAI - p3
area a pericolosità da frana pai molto elevata - p4 (kmq)	float	very high landslide hazard area PAI - p4
provincia	nchar	province

OD_Classificazione_Sismica_Province (OD Province Seismic Classification)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
Regione	nvarchar	region
Province	nvarchar	province
CodiceIstat	float	Istat code
Denominazione	nvarchar	name
Classificazione2015	int	classification 2015

OD_Codifica_Comuni_Province (OD Coding Municipalities Provinces)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
comune	nvarchar	city
provincia	nvarchar	province
sigla	nvarchar	acronym

OD_Elenco_Cod_Province (OD Province Codes List)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
codreg	float	region code
regione	nvarchar	region
codice	float	code
sigla	nvarchar	acronym
provincia	nvarchar	province

OD_Immatricolazioni_Auto (OD Car registrations)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
Provincia	varchar	province
Regione	varchar	region
Classe di cilindrata	varchar	displacement class
Numero veicoli	varchar	number of vehicles

OD_Inail_Dati_Infortuni (OD Inail Accident Data)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
DataRilevazione	varchar	Detection Date
DataProtocollo	varchar	Protocol date
DataAccadimento	varchar	Occurrence date
DataDefinizione	varchar	Definition date
DataMorte	varchar	Death date
LuogoAccadimento	varchar	Occurrence place
IdentificativoInfortunato	varchar	Injured ID
Genere	varchar	Gender
Eta	varchar	Age
LuogoNascita	varchar	Place of birth
ModalitaAccadimento	varchar	Happening mode
ConSenzaMezzoTrasporto	varchar	With Without Means Of Transport
IdentificativoCaso	varchar	Case ID
DefinizioneAmministrativa	varchar	Administrative Definition
DefinizioneAmministrativaEsitoMortale	varchar	Administrative Definition Mortal Outcome
Indennizzo	varchar	Compensation
DecisioneIstruttoriaEsitoMortale	varchar	Investigative Decision Mortal Outcome
GradoMenomazione	varchar	Degree of impairment
GiorniIndennizzati	varchar	Compensation Days
IdentificativoDatoreLavoro	varchar	Employer ID
PosizioneAssicurativaTerritoriale	varchar	Territorial Insurance Position
SettoreAttivitaEconomica	varchar	Economic Activity Sector
Gestione	varchar	Management
GestioneTariffaria	varchar	Tariff management
GrandeGruppoTariffario	varchar	Large Tariff Group

OD_Istat_Dati_Incidenti_Stradali (OD Istat Road Accident Data)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
Provincia	nvarchar	province
conducente morto	float	dead driver
passeggero morto	float	dead passenger
pedone morto	float	dead pedestrian
totale morto	float	total dead
conducente ferito	float	injured driver
passeggero ferito	float	injured passenger
pedone ferito	float	injured pedestrian

totale ferito	float	total injured
conducente totale	float	total driver
passeggero totale	float	total passenger
pedone totale	float	total pedestrian
totale totale	float	total total

OD_Precipitazioni_Medie_Ultimi_10_anni (OD Average Precipitation Last 10 years)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
Provincia	varchar	province
2009	varchar	year 2009
2010	varchar	year 2010
2011	varchar	year 2011
2012	varchar	year 2012
2013	varchar	year 2013
2014	varchar	year 2014
2015	varchar	year 2015
2016	varchar	year 2016
2017	varchar	year 2017
2018	varchar	year 2018

OD_Reati (OD Crimes)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_altri_delitti (OD other crimes)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_Associazione_per_delinquere (OD Criminal association)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

OD_Reati_Associazione_tipo_mafioso (OD Mafia type association)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

OD_Reati_Furti (OD theft crimes)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

OD_Reati_Furti_Autovetture (OD car theft offenses)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

OD_Reati_Furti_Con_Strappo (OD Theft Crimes With Tear)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION

POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

OD_Reati_Furti_in_abitazioni (OD Thefts in homes)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

OD_Reati_Furti_in_esercizi_commerciali (OD Thefts in commercial establishments)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

OD_Reati_Furto_con_destrezza (OD Theft With Dexterity)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

OD_Reati_Incendi (OD fire crimes)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province

NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_omicidi (OD homicidal offenses)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	Float	pos#
PROVINCIA	Nvarchar	province
NUMERO REATI	Float	number of crimes
OGNI 100mila ABITANTI	Float	every 100000 inhabitants
VAR# % ANNUA	Float	annual percentage change

OD_Reati_omicidi_consumati (OD homicidal crimes committed)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_rapine (OD robbery crimes)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_Reciclaggio_impiego_denaro (OD Money laundering crimes)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants

VAR# % ANNUA	float	annual percentage change

OD_Reati_Stupefacenti (OD Narcotic offenses)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_Tentati_Omicidi (OD Murder Attempted Offenses)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_Truffe_frodi_informatiche (OD Computer fraud scams)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_Usura (OD Wear offenses)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR# % ANNUA	float	annual percentage change

OD_Reati_Violenze_sessuali (OD Sexual Violence Offenses)

****

COLUMN_NAME	DATA_TYPE	DESCRIPTION
POS#	float	pos#
PROVINCIA	nvarchar	province
NUMERO REATI	float	number of crimes
OGNI 100mila ABITANTI	float	every 100000 inhabitants
VAR#% ANNUA	float	annual percentage change

Appendix I: Report on Insurance Loss Prediction using the Insurance Dataset

Insurance Claim Prediction: Regression Models for Modeling Claim Severity

Giorgio Rofflo, Richard McCreadie, Craig Macdonald and Iadh Ounis
University of Glasgow

ABSTRACT

In recent years, there has been a growing interest in machine learning methods to determine the expected cost of complaints lodged by policyholders. Machine learning and deep learning models can equip insurers with a better understanding of the claim costs, thereby enhancing the ability to predict the final cost of claims and the accuracy of estimating the claims reserve. In this paper, we provide a comparison between different regression approaches for modelling the cost of non-life insurance claims, and hence severity. In particular, we focus on Linear Regression, Random Forest Regression, Support Vector Regression and deep feed forward Artificial Neural Networks. Experiments have been performed on *Allstate Claim Severity* - a publicly available insurance dataset - and a proprietary dataset provided by the *GFT Group*. Results show that the Random Forest regressor outperforms all the other alternative methods on the GFT dataset in terms of mean absolute error (MAE). On the Allstate dataset, the 6-layer ANN predicts the claim severity 36.2% better than chance.

ACM Reference Format:
Giorgio Rofflo, Richard McCreadie, Craig Macdonald and Iadh Ounis. 2020. Insurance Claim Prediction: Regression Models for Modeling Claim Severity. In *Proceedings of Not Published.* ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

Leading insurance carriers have access to more data than ever and with traditional methods of analysis, only 15% of the data they have access to is used [15]. This means that they have difficulties in unlocking the value from this data and many valuable insights hidden in their processing are overlooked [2].

One major benefit of machine learning is that it can be effectively applied on insurance data and leveraged for different insurance business areas such as customer behaviour, fraud identification, risk management and loss prevention [9, 16]. However, traditional models for claim prediction are *policy-based* [3]. In fact, these models consider groups of policies with similar characteristics such that historical losses to make future predictions. As a consequence, one of the shortcomings of these traditional methods is that they do not take into account the characteristics of the policyholders and therefore the predictions are not *user-centric* but they are rather based on the type of policy. This problem introduces a high variance in the final prediction (e.g., when policyholders do not follow the cluster trend and their claims can change drastically). In the case of loss prevention, the current modelling does not allow an insurer to accurately estimate the claims reserve to cover future payments.

Hence, this paper is a first step in addressing this gap by quantifying how effective current machine/deep learning approaches are for insurance prediction where few prior interactions (claims) are available. In particular, we cast this as a single-instance regression problem, where we aim to predict the severity of claims (using continuous and categorical features), i.e., to determine the expected cost of claims filed by policyholders. Therefore, we propose a comparison between different regression approaches such as Linear Regression, Random Forest Regression (RF), Support Vector Regression (SVR) and Artificial Neural Networks (ANN) [7, 10, 19, 20] that shows how machine learning and deep learning models work effectively for this type of data in a user-centric manner.

Our results on two real-world datasets are encouraging, showing that Random Forest and ANN outperform other regression methods (by a statistically significant margin) in terms of mean absolute error, which translates into a better predictability and a greater stability of claim modeling. On the other hand, this is clearly a difficult problem, with mean prediction errors ranging between $1,000 to $1,500 USD per claim depending on dataset, motivating further research into this area.

The contributions of this paper are three-fold: i) we provide a comparison of different regression methods on the insurance loss prediction task with sparse claim data, ii) we demonstrate the application of machine learning models in a user-centric way, i.e. by integrating users characteristics (e.g. gender[1] and location) iii) we provide a comparative analysis of a new large *GFT* insurance dataset and consequent possible strategies for future works.

The remainder of the paper is organised as follows: Sec.2 illustrates some related literature, mostly focusing on the comparative approaches we consider in this study. A description of the datasets used in this paper can be found in Sec.3. Experiments and results are reported in Sec.4. Finally, in Sec.5, conclusions are given, and future perspectives are envisaged.

2 RELATED WORK

In recent years there has been an increase in interest in the automobile insurance industry [9, 16]. In the literature of auto insurance claim prediction, prior works have experimented with a range of models, including linear regression [8], ensembles of randomised regression trees [6] and Artificial Neural Networks (ANN) [20].

In [5], the authors considered the set of algorithms mentioned above to predict Bodily Injury Liability Insurance claim payments based on the characteristics of the insured customer's vehicle. However, the task is simplified to a binary classification problem, and therefore, to predict whether instances are associated with a claim amount equal to zero or not. In [8], the authors proposed the application of regression methods to Allstate Insurance Claims Severity [17] (a publicly available insurance dataset that we also use in our experiments). Their results show that random forests and gradient boosted trees had the best performances out of all of the models by a substantial margin. The accuracy of the random forest model in the case study of claim severity prediction for car insurance has been further studied in [6]. In the paper, the authors used the principal component analysis to perform dimensionality reduction

[1] We note that practice features like gender may not be usable in some regions that stipulate reserved categories.