


Tailored IoT & BigData Sandboxes and Testbeds for Smart,
Autonomous and Personalized Services in the European
Finance and Insurance Services Ecosystem



D5.8 – Library of ML/DL Algorithms-II

Revision Number	3.0
Task Reference	T5.4
Lead Beneficiary	FBK
Responsible	Bruno Lepri
Partners	AGRO, ATOS, ATOS-IT, BOUN, CP, CTAG, FBK, FTS, GFT, GLA, JRC, JSI, NUIG, ORT, PRIVE, RB, UBI, UPRC, WEA
Deliverable Type	Report (R)
Dissemination Level	Public (PU)
Due Date	2021-07-31
Delivered Date	2021-11-03
Internal Reviewers	ENG, HPE
Quality Assurance	INNOV
Acceptance	WP Leader Accepted and/or Coordinator Accepted
EC Project Officer	Beatrice Plazzotta
Programme	HORIZON 2020 - ICT-11-2018
	This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632

Contributing Partners

Partner Acronym	Role ¹	Author(s) ²
FBK	Lead Beneficiary	
ABILAB	Contributor	
AGRO	Contributor	
ATOS	Contributor	
BOUN	Contributor	
CXB	Contributor	
ENG	Contributor	
INNOV	Contributor	
ISPRINT	Contributor	
JSI	Contributor	
NBG	Contributor	
PRIVE	Contributor	
UBI	Contributor	
UPRC	Contributor	
WEA	Contributor	
ENG	Internal Reviewer	
HPE	Internal Reviewer	

Revision History

Version	Date	Partner(s)	Description
0.1	2021-07-15	FBK	ToC Version
0.2	2021-07-21	FBK, UBI	Contributions on Section 2
0.3	2021-08-20	INNOV	Contribution on Section 3.1
0.4	2021-08-26	BOUN, NBG	Contributions on Section 3.7, 3.4
0.5	2021-08-31	JSI	Contribution on Section 3.6
0.6	2021-09-01	UPRC	Contribution on Section 3.3
0.7	2021-09-05	FBK, FTS, CXB	contribution on Section 3.5
0.8	2021-09-06	ENG, FBK	Contribution on Sections 3.8, 1
0.9	2021-09-07	ISPRINT	Contributions on Section 3.10
0.10	2021-09-14	PRIVE	Contribution on Section 3.1
0.11	2021-09-17	AGRO, ATOS	Contribution on Sections 3.9, 3.12
0.12	2021-09-30	FBK	Contribution on Section 4
0.13	2021-10-07	ABILAB	Contribution on Section 3.13
0.14	2021-10-11	WEA	Contribution on Section 3.11
1.0	2021-10-12	FBK	First Version for Internal Review
1.1	2021-10-20	HPE	Internal Review

¹ Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

² Can be left void

1.2	2021-10-22	ENG	Internal Review
2.0	2021-10-22	FBK	Version for Quality Assurance
3.0	2021-11-02	FBK	Version for Submission

Executive Summary

The deliverable D5.8 is the second of a series of three deliverables programmed within the scope of task T5.4. The general objectives of the task are to first identify state-of-the-art and innovative machine learning solutions to practical challenges within the sphere of the FinTech and InsuranceTech revolution. As a second step, the task aims at implementing and integrating, as INFINITECH services, ready-to-use solutions of potential interest for a wide spectrum of financial institutions and companies.

Given these objectives, Section 2 of deliverable D5.8 describes the development of a framework for the definition and instrumentation of standardized ML/DL pipelines on top of the MLflow open-source platform. This framework allows to produce ready-to-use ML and DL models as well as to deploy and publish these models as microservices. This common interface enables and supports the ML/DL needs and activities of the INFINITECH partners as well as the needs of banks, insurance companies, and other potential adopters and users of the INFINITECH platform. Finally, the developed framework integrates the INFINITECH Open API Gateway which is developed within task T5.5.

Section 3 of the deliverable documents the progress of the ML/DL tasks of each INFINITECH pilot, reporting the type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms. Specifically, we document the ML/DL tasks' definitions for each pilot, the innovative or state-of-the-art ML/DL algorithms implemented and evaluated by the pilots, their initial performances, and the type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms.

This deliverable is additionally intended to be a guide for pilot's developers, providing a step-by-step introduction to the end-to-end framework for deploying algorithms at the core of the INFINITECH library of ML/DL algorithms. At this step of the project, the deliverable will help pilot's harmonization and final preparation of a perfectly orchestrated service framework.

We conclude the deliverable providing some conclusions and discussing the next steps toward the third and final deliverable of the task.

Table of Contents

1	Introduction	10
1.1	Objective of the Deliverable	10
1.2	Insights from other Tasks and Deliverables	11
1.3	Structure	11
2	INFINITECH end-to-end framework for the deployment of ML/DL-based microservices	12
2.1	MLflow	13
2.2	Minikube and the local Kubernetes cluster	15
2.3	Consul Service Registry	16
2.4	Deploy MLflow model on Minikube	20
2.5	INFINITECH Open API Gateway	22
3	INFINITECH pilots' ML/DL tasks: Progress and preliminary results	27
3.1	Pilot 2 - Real-time risk assessment in investment banking	28
3.2	Pilot 4 - Personalized portfolio management - mechanisms for an AI-based portfolio construction	29
3.3	Pilot 5B Financial Management (BFM) tools delivering smart business advice	33
3.4	Pilot 6 - Personalized closed-loop investment portfolio management for retail customers	37
3.5	Pilot 7 - Avoiding financial crime	40
3.6	Pilot 8 - Platform for Anti Money Laundering (AML) supervision	41
3.7	Pilot 9 - Analyzing Blockchain transaction graphs for fraudulent activities	51
3.8	Pilot 10 - Real-time cyber-security analytics on financial transactions' Big Data	53
3.9	Pilot 11 - Personalized insurance products based on IoT connected vehicles	55
3.10	Pilot 12 - Real-world data for novel health insurance products	56
3.11	Pilot 13 - Alternative/automated insurance risk selection - product recommendation for Small and Medium Enterprises (SMEs)	57
3.12	Pilot 14 - Big Data and IoT for the agricultural insurance industry	58
3.13	Pilot 15 - Open Inter-Banking	62
4	Conclusions	63
5	BIBLIOGRAPHY	64

List of Figures

Figure 1 – Schematic of the end-to-end framework for the definition and instrumentation of standardized ML/DL pipelines.	13
Figure 2 – New Nginx configuration file with the custom modification highlighted in yellow.	16
Figure 3 – New Dockerfile to build the new image.	16
Figure 4 – Minikube starts the node.	16
Figure 5 – Using the “minikube dashboard” command to access the dashboard and monitor the cluster.	17
Figure 6 – Kubernetes dashboard panel.	17
Figure 7 - Fetching the Helm Chart provided by Hashicorp and adding it to the Helm local repository.	18
Figure 8 - Exposing the UI through a Kubernetes NodePort service.	18
Figure 9 - Enabling secure communication between pods.	18
Figure 10 - Annotating the Kubernetes service to be discovered in the deployment file.	18
Figure 11 - Helm-values YAML file.	19
Figure 12 - Configuration file stored in the k8 subfolder.	19
Figure 13 - Name of the Consul deployment on Kubernetes and of the fetched Helm chart.	20
Figure 14 - Minikube service list command.	20
Figure 15 - List of deployed services.	21
Figure 16 - MLflow deployment.	22
Figure 17 - Injecting the pod with “sidecar” Consul proxy.	22
Figure 18 - Synchronizing the Kubernetes services with the Consul Service Registry.	22
Figure 19 - Additional annotations for the INFINITECH Open API Gateway.	23
Figure 20 - Configuration file to deploy the INFINITECH Open API Gateway on Kubernetes.	24
Figure 21- LoadBalancer service.	25
Figure 22 - Docker-compose file to set the UI properties.	25
Figure 23 - Browser to navigate the UI.	26
Figure 24 - Invocation endpoint.	26
Figure 25 - Calling the endpoint with a CURL POST.	27
Figure 26 - Schematics of portfolio selection based on fitness score.	32
Figure 27 - Hybrid classification model flowchart.	35
Figure 28 - Rule-based step approach categorization results.	36
Figure 29 - SHAP values of each feature included in the Catboost model.	37
Figure 30 - Proposed one step forward validation scheme.	38
Figure 31 - Structure of the configuration file	43
Figure 32 - Format of the message read from Kafka topic.	44
Figure 33 - Shape of the JSON consumer file.	44
Figure 34 - Shape of the JSON Kafka consumer file.	45
Figure 35 - The format of the outputted object.	50
Figure 36 - The shape of the JSON file containing a single field data whose value is an array of objects.	50
Figure 37: StreamStory pipeline.	52
Figure 38 - Nested resampling for parameter tuning with 3-fold Cross Validation in the outer and 4-fold Cross Validation in the inner loop.	61

List of Tables

Table 1 – INFINITECH Library: overview list of ML/DL algorithms	28
Table 2 - Example of multi-factor aggregation table for fitness computation of customer portfolio.	31
Table 3 - Test cases showing portfolio score of generated portfolios through optimization.	33
Table 4 - Initial results of the cash flow prediction mode.	38
Table 5- Initial performances obtained on a small dataset of Ethereum transactions having 14544 addresses.	53
Table 6 - Data types.	57
Table 7 - Summary of EO data features used for the calibration/prediction of drought and hailstorm services.	60
Table 8 - Initial performances obtained on a small dataset of Spain drought cases.	63
Table 9 - T5.4 objectives and D5.8 achievements.	66
Table 10 - T5.4 KPIs and D5.8 achievements.	67

Abbreviations/Acronyms

ADWIN	Adaptive Windowing
AFAC	Allocation Factor
AI	Artificial Intelligence
AML	Anti-Money Laundering
API	Application Programming Interface
AUC	Area Under Curve
BDA	Big Data Analytics
BFM	Business Financial Management
BOC	Bank Of Cyprus
BSD	Berkeley Software Development
CFT	Combating the Financing of Terrorism
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRM	Customer Relationship Management
CTAG	Galician Automotive Technology Centre
CUDA	Compute Unified Device Architecture
CV	Computer Vision
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DL	Deep Learning
DNNs	Dense Neural Networks
DPI	Dots Per Inch
EO	Earth Observation
ES	Expected Shortfall
FAIR	Facebook Artificial Intelligence Research
FAPAR	Fraction of Absorbed Photosynthetically Active Radiation
FATF	Financial Action Task Force
FBK	Fondazione Bruno Kessler
FinTech	Financial Technology
FIU	Financial Intelligence Unit
Forex	Foreign Exchange
FVC	Fractional Vegetation Cover
GDPR	General Data Protection Regulation
GFT	Gesellschaft für Technologietransfer
GLA	University of Glasgow
GNDVI	Green Normalized Difference Vegetation Index
GPU	Graphics Processing Unit
INRIA	National Institute for Research in Digital Science and Technology
InsuranceTech	Insurance Technology
IoT	Internet Of Things
JSON	JavaScript Object Notation
k8s	Kubernetes
KG	Knowledge Graph

kNN	K-nearest Neighbor
KPIs	Key Performance Indicators
KYC	Know Your Customer
LAI	Leaf Area Index
LIME	Local Interpretable Model-agnostic Explanation
LLM	Logic Learning Machine
LMR	Learning for Mismatch Removal
LOOCV	Leave One Out Cross Validation
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAP	Mean Average Precision
MCARI	Modified Chlorophyll Absorption Ratio Index
Mifid	Markets in Financial Instruments Directive
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NBG	National Bank of Greece
NDCG	Normalized Discounted Cumulative Gain
NDVI	Normalized Difference Vegetation Index
NLP	Natural Language Processing
NPL	Non-Performing Loan
OCR	Optical Character Recognition
PCA	Principal Component Analysis
PT	Process Tree
R2	R Squared
RBF	Radial Basis Function
REIP	Red Edge Inflection Point
ReST	Representational State Transfer
RMSE	Root Mean Squared Error
ROC AUC	Area Under the Receiver Operating Characteristic Curve
RWD	Real Wheel Drive
SHAP	SHapley Additive exPlanation
SME	Small and Medium Enterprise
SMOTE	Synthetic Minority Oversampling Technique
SNAP	Stanford Network Analysis Project
SRF	Sharpe Ratio Factor
SUMO	Simulation of Urban MObility
SVM	Support Vector Machine
TB	Terabyte
Tf-idf	Term frequency-inverse document frequency
tps	transactions per second
TPU	Tensor Processing Unit
TSVD	Truncated Singular Value Decomposition
UI	User Interface

URL	Uniform Resource Locator
VaR	Value at Risk
VASP	Virtual Asset Service Provider
VAT	Value Added Tax
XAI	eXplainable Artificial Intelligence
XML	eXtensible Markup Language

Introduction

The current deliverable, D5.8, is the second one of a series of three deliverables whose aim is to describe the activities conducted in task T5.4. The main objectives of this task are the identification, implementation and integration, as INFINITECH services, of state-of-the-art as well as of innovative Machine Learning (ML) and Deep Learning (DL) solutions. In addition, task T5.4 aims at providing technical tools to support the ML and DL needs and planned solutions of the INFINITECH pilots.

Thus, the first part of the deliverable introduces and describes in detail a key outcome of task T5.4, namely **a framework for the definition and production of ML/DL microservices and pipelines** (see Section 2 for the detailed description of its implementation and functioning). More specifically, the developed framework allows for the *definition and instrumentation of such ML/DL pipelines* in a standardized manner, on top of the MLflow open-source platform³, in order to produce *ready-to-use ML and DL models* as well as to *deploy and publish these models as microservices*.

After the presentation of the implementation of this framework, the second part of the deliverable (see Section 3) is dedicated to a detailed summary of the progress of the ML/DL tasks of each INFINITECH pilot, of their preliminary results (e.g., novel ML/DL algorithms implemented within the pilot, state-of-the-art ML/DL algorithms evaluated within the pilot, initial performances obtained, etc.) as well as on the type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms (e.g., pre-built/pre-trained ML/DL models, ML/DL frameworks with data to perform the training step, only ML/DL frameworks and results obtained on proprietary data, etc.).

1.1 Objective of the Deliverable

As already mentioned in the previous deliverable, the main goal of task T5.4 (“Library of ML/DL Algorithms for Financial/Insurance Services”) is to collect, describe, and make available as a library a diverse set of ready-to-use and ready-to-deploy ML/DL algorithms that are suitable and potentially transformative for a large number of applications in the finance and insurance domains. This library of algorithms will include both conventional state-of-the-art ML/DL algorithms and innovative ones developed by the technical partners and by the pilots during the INFINITECH project. This overall goal encompasses, in this second deliverable D5.8, the following specific objectives:

- **Develop tools to support the easy definition and deployment of ML/DL pipelines.** One of the most important goals of this task is the development of tools to support the ML/DL needs and activities of the INFINITECH partners (for example, the ML/DL tasks defined within the various INFINITECH pilots) as well as the needs of banks, insurance companies, and other potential adopters and users of the INFINITECH platform and of its services. This objective is pursued here by implementing a dedicated INFINITECH framework for the definition and production of ML/DL microservices and pipelines. As previously said, this framework allows the definition of ready-to-use ML and DL models and their deployment as microservices.
- **Document and support the ML/DL needs and planned solutions of the pilots.** This is another very important goal of the task T5.4. Indeed, the development activities of the task, and more in general of the INFINITECH project, have their main focus on the pilots. The first deliverable of T5.4 was dedicated to document the planned ML/DL solutions devised by each pilot and to report in detail the needed ML/DL tools and libraries. In this second deliverable, we document the progress of the ML/DL tasks of each INFINITECH pilot as well as we report the type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms (e.g., pre-built/pre-trained ML/DL models, ML/DL frameworks with data to perform the training step, etc.)

³ <https://mlflow.org/>

1.2 Insights from other Tasks and Deliverables

The deliverable D5.8 is released within the scope of WP5 “Data Analytics Enablers for Financial and Insurance Services” activities and documents the second round of outcomes of the work performed within the context of T5.4 “Library of ML/DL Algorithms for Financial/Insurance Services”. As also highlighted in the previous deliverable, the task T5.4 is connected with the outcomes of WP2 “Vision and Specifications for Autonomous, Intelligent and Personalized Services” in which the overall requirements of the entire INFINITECH platform are defined. Furthermore, the work reported in this deliverable D5.8 is connected tightly and done in collaboration with the one performed within the context of task T5.5 of WP5. Indeed, the implemented end-to-end framework for the *standardized definition and instrumentation of ML/DL pipelines*, described in detail in Section 2, integrates the INFINITECH Open API Gateway which is developed within task T5.5.

1.3 Structure

We organized the structure of D5.8 to be easily associated with the two deliverable objectives described in Section 1.1:

- Section 2 presents in detail an end-to-end framework developed for the *definition and instrumentation of ML/DL pipelines* in a standardized manner. More specifically, we describe in detail the four different software components composing the framework, namely (i) MLflow, (ii) Kubernetes⁴, (iii) Consul Service Registry⁵, and (iv) the INFINITECH Open API Gateway developed within the Task 5.5 (see the associated deliverables of T5.5). These four components are integrated in an end-to-end scenario of ML/DL service deployment. In particular, we have implemented a first use case for our framework using the ML/DL-driven recommendation algorithms included in BetaRecsys, a novel open-source framework, developed by the University of Glasgow (GLA) and adopted by Pilot 6.
- Section 3 maps and describes in detail the ML/DL tasks of the INFINITECH pilots. There we focus our attention to document the ML/DL tasks’ definitions for each pilot, the innovative or state-of-the-art ML/DL algorithms implemented and evaluated by the pilots, the initial performances, and type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms (e.g., pre-built/pre-trained ML/DL models, ML/DL frameworks with data to perform the training step, etc.).
- Section 4 draws some conclusions and discusses the next steps toward the third and final deliverable of task T5.4.

⁴ <https://kubernetes.io/>

⁵ <https://www.consul.io/>

2 INFINITECH end-to-end framework for the deployment of ML/DL-based microservices

In this section we present in detail an end-to-end framework developed for the *definition and instrumentation of ML/DL pipelines* in a standardized manner. This framework is built on top of the MLflow platform, and it has the objective of producing *ready-to-use ML and DL models* as well as *to deploy and publish these models as microservices*.

More specifically, the framework consists of the integration of four different software components: (i) MLflow, (ii) Kubernetes, (iii) Consul Service Registry, and (iv) the INFINITECH Open API Gateway developed within task T5.5. These four components are integrated in an end-to-end scenario of ML/DL service deployment. Currently, the framework integrates the ML/DL-driven recommendation algorithms implemented in BetaRecsys, a novel open-source framework, developed by the GLA and in use in Pilot 6. For this reason, Pilot 6 is the first to benefit from the developed technological framework but the plan is to extend it to the majority of pilots and to make this framework the core of the library of ML/DL algorithms within the INFINITECH platform.

More in detail, the service deployed is a ML/DL model, trained, tracked and wrapped in a Docker⁶ image with the MLflow API. The Docker image thus constructed is integrated with Swagger documentation that describes, according to Open API specifications, the endpoints of the container. The docker is then deployed on a Kubernetes cluster through its configuration file. Consul Service Registry is installed on the same cluster with the main functionality of discovering the service and tracing it in its own service registry. The last component, the INFINITECH Open API Gateway, developed within task T5.5, acts as an abstraction layer for the client for the discovery of services, leveraging the Consul Service Registry's offerings, and as an advanced reverse proxy. This component is placed in front of the architecture thus defined. In the following subsections, we describe in detail the four software components of the proposed framework, while a schematic illustration of their integration is depicted in Figure 1.

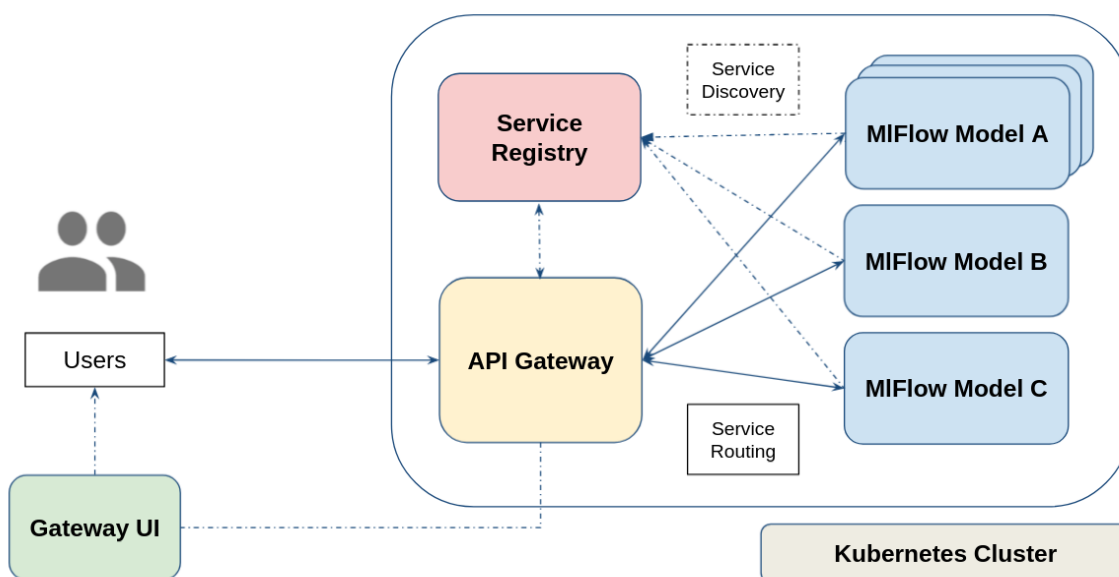


Figure 1 – Schematic of the end-to-end framework for the definition and instrumentation of standardized ML/DL pipelines.

⁶ <https://www.docker.com/>

2.1 Mlflow

Mlflow is an open-source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry.

The framework is library-agnostic. All its functions are accessible through REST API and command-line interface (CLI). Mlflow offers Python, R, and Java APIs, and supports many ML frameworks and libraries, like PyTorch⁷ (Lightning), TensorFlow⁸ and Keras⁹, Scikit-learn¹⁰, etc. It is also integrated with various technologies such as Conda, Docker and Kubernetes, Databricks, Azure/S3/Google Cloud Storage, etc.

Mlflow currently has four components:

- Tracking;
- Projects;
- Models;
- Models Registry.

The Mlflow *Tracking* component is an API for logging parameters, code version, metrics, and artefacts. It exposes a user interface (UI) to graphically view and handle the results of the ML processes. An Mlflow *Project* is a format for packaging data science code in a reusable and reproducible way, based primarily on conventions defined by the library. The *Project* component includes an API and command-line tools for running projects into workflows on different platforms, including Kubernetes. An Mlflow *Model* is, instead, a standard format for packaging ML models that can be used in a variety of downstream tools. The format defines a convention that lets a user save a model in different “flavours” that can be understood by different downstream tools. The models can be packaged in a Docker image. Finally, the Mlflow *Model Registry* component is a centralized model store, set of APIs, and UI, built to collaboratively manage the full lifecycle of an Mlflow *Model*. It provides model lineage (for example, it registers which Mlflow experiment and run have produced the model), model versioning, information on stage transitioning, and annotations.

2.1.1 Mlflow model as a service

For our purpose, the Mlflow *Model* component is the most relevant among the four ones. As mentioned, Mlflow offers the possibility to wrap the trained model into a docker image. The library provides a blueprint for the construction of such an image. Inside the docker, the model is exposed on a Gunicorn¹¹ server with an Nginx¹² instance in front of it. The container exposes an invocations’ API on one port (8080). The API can receive data and send predictions back. The port and the API exposed are the same for every Mlflow Docker Model.

As per the requirements of the INFINITECH API Gateway, a service must provide a Swagger JSON document that describes the application endpoints. Due to the standardization of the services provided by Mlflow, the Swagger document is easily standardizable.

⁷ <https://pytorch.org/>

⁸ <https://www.tensorflow.org/>

⁹ <https://keras.io/>

¹⁰ <https://scikit-learn.org/>

¹¹ <https://gunicorn.org/>

¹² <https://www.nginx.com/>

Once we have created the JSON documentation file, we must include it in a new docker image built on top of the one created by Mlflow. As we mentioned, the model is served on an internal server. The incoming requests to the server are handled by an Nginx proxy. Mlflow provides a basic configuration for the proxy server. To make the Swagger file accessible, the basic Nginx configuration file must be modified to tell the service where the JSON is and to send it back if required.

The following example presents the new Nginx configuration file with the custom modification highlighted in yellow.

```
Worker_processes 1;
daemon off; # Prevent forking

pid /tmp/nginx.pid;
error_log /var/log/nginx/error.log;

events {
    # defaults
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    access_log /var/log/nginx/access.log combined;

    upstream gunicorn {
        server 127.0.0.1:8000;
    }

    server {
        listen 8080 deferred;
        client_max_body_size 5m;

        keepalive_timeout 5;

        location ~ ^/(ping|invocations) {
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Host $http_host;
            proxy_redirect off;
            proxy_pass http://gunicorn;
        }

        location ~ ^/swagger {
```

```

    add_header Content-Type    application/json;
    alias /var/www/html/swagger_mlflow-model.json;
}

location / {
    return 404 "{}";
}
}
}

```

Figure 2 – New Nginx configuration file with the custom modification highlighted in yellow.

In addition, the following example illustrates the dockerfile to build the new image.

```

FROM smartcommunitylab/mlflow-model

COPY swagger_mlflow-model.json /var/www/html/
COPY nginx.conf
/miniconda/lib/python3.8/site-packages/mlflow/models/container/scoring_server/nginx.conf

```

Figure 3 – New Dockerfile to build the new image.

2.2 Minikube and the local Kubernetes cluster

It is possible to deploy a single infrastructure composed of Mlflow, Consul Service Registry, and Kubernetes using Minikube¹³. In particular, by means of Minikube we can deploy a single-node Kubernetes cluster locally. Indeed, most of the functionalities provided by the docker orchestrator are accessible with Minikube. More specifically, we can interact with the cluster with kubectl. First of all, Minikube must start the node.

```

File Modifica Visualizza Cerca Terminale Aiuto
mmartini@mmartini:~$ minikube start
🐻 minikube v1.17.1 on Ubuntu 18.04
🔗 minikube 1.18.1 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.18.1
💡 To disable this notice, run: 'minikube config set WantUpdateNotification false'
🌟 Using the docker driver based on existing profile
👍 Starting control plane node minikube in cluster minikube
🔄 Restarting existing docker container for "minikube" ...
📦 Preparing Kubernetes v1.20.2 on Docker 20.10.2 ...
🔍 Verifying Kubernetes components...
🌟 Enabled addons: default-storageclass, storage-provisioner, dashboard
🎉 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
mmartini@mmartini:~$

```

¹³ <https://minikube.sigs.k8s.io/>

Figure 4 – Minikube starts the node.

Then, we can access the dashboard using “minikube dashboard” to monitor the cluster.

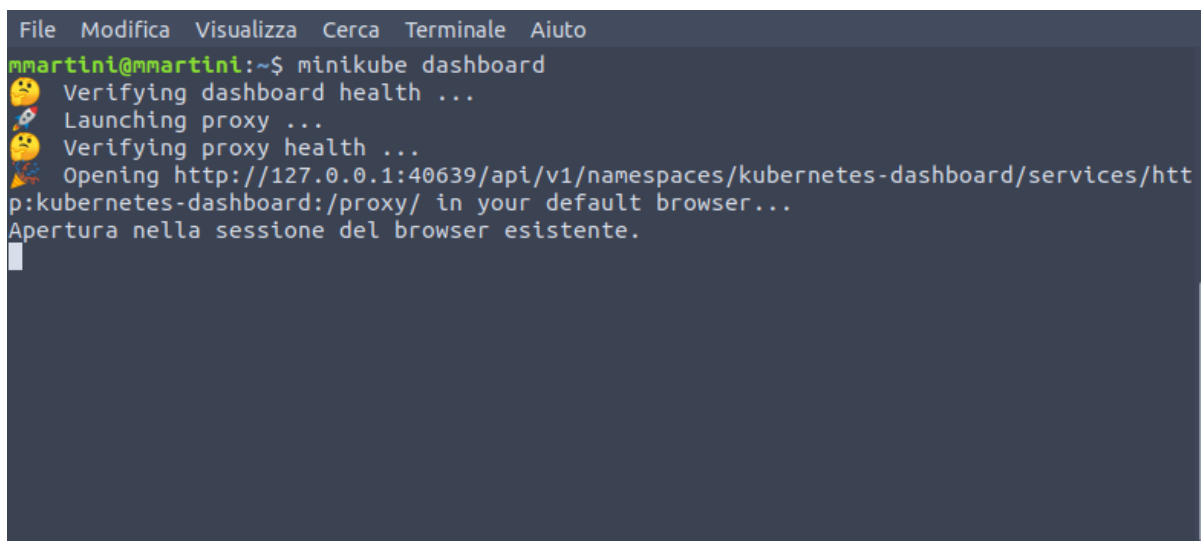


Figure 5 – Using the “minikube dashboard” command to access the dashboard and monitor the cluster.

Minikube should automatically open the predefined browser on the dashboard panel. From the dashboard, we can monitor the status of the pods, services, deployments, etc.

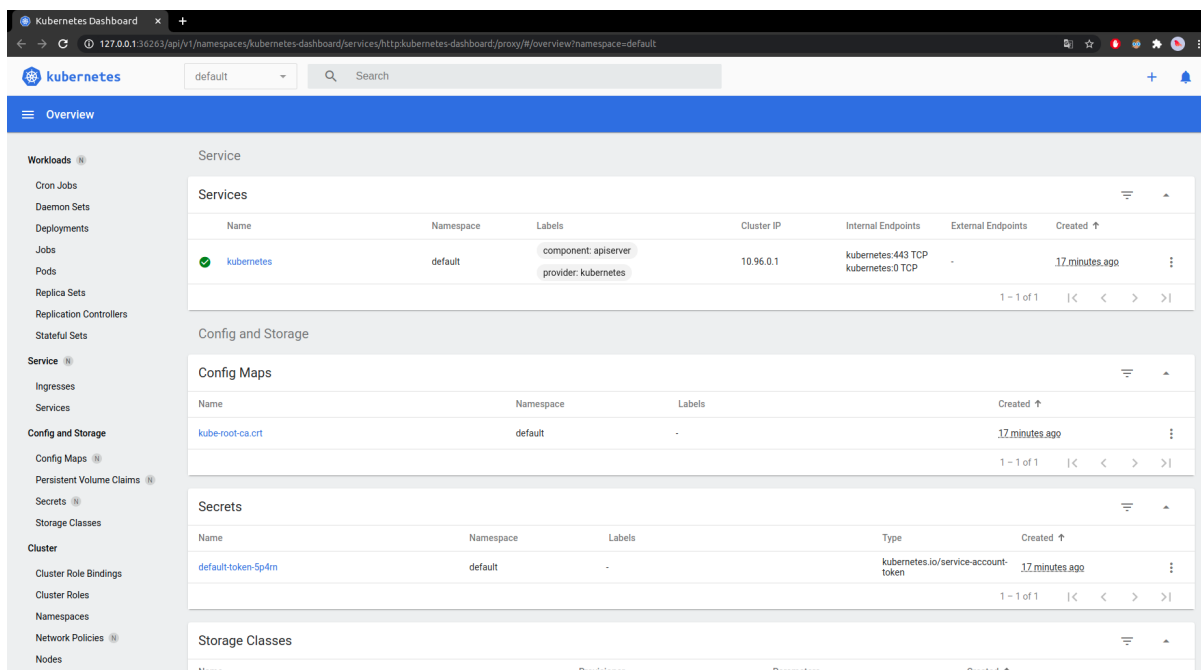


Figure 6 – Kubernetes dashboard panel.

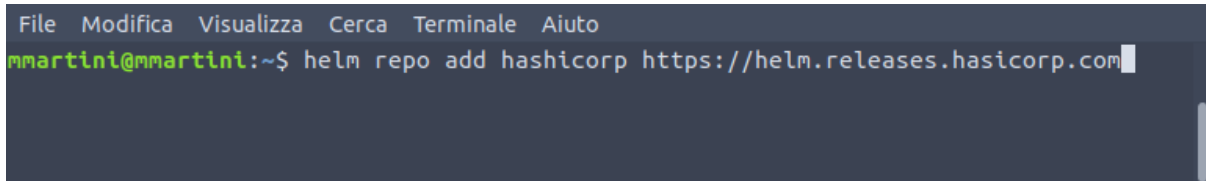
2.3 Consul Service Registry

Consul is a service networking solution to automate network configurations, discover services, and enable secure connectivity across any cloud or runtime.

For our end-to-end framework, the most relevant component of Consul Service Registry is the discovery service with its service registry. The idea is to register Kubernetes services on the Consul Service registry. The INFINITTECH Open API Gateway, developed within T5.5, points directly to the service registry and

enables users to discover services' existence, their health status and redirect traffic. A UI will finally display this information.

It is worth noting that it is possible to integrate Consul Service Registry with Kubernetes through Helm. To this end, we first have to fetch the Helm Chart provided by Hashicorp and add it to our Helm local repository. Alternatively, we can clone the Github repo.



```
File Modifica Visualizza Cerca Terminale Aiuto
mmartini@mmartini:~$ helm repo add hashicorp https://helm.releases.hashicorp.com
```

Figure 7 - Fetching the Helm Chart provided by Hashicorp and adding it to the Helm local repository.

In particular, we need to create a YAML Helm-values configuration file. In this file, we define some parameters. For example, we expose the UI through a Kubernetes NodePort service.

```
ui:
  service:
    type: 'NodePort'
```

Figure 8 - Exposing the UI through a Kubernetes NodePort service.

Then, we enable secure communication between pods.

```
connectInject:
  enabled: true
```

Figure 9 - Enabling secure communication between pods.

Given that we need to register the model service on the Consul Service Registry, we have added the “syncCatalog” option to the configuration file. The synchronization is bidirectional: indeed, Kubernetes services are discovered by Consul and vice-versa. However, Kubernetes services are not discovered by default. The Kubernetes service we want to be discovered must be appropriately annotated in the deployment file.

```
syncCatalog:
  enabled: true
  default: false
  toConsul: true
  toK8S: true
```

Figure 10 - Annotating the Kubernetes service to be discovered in the deployment file.

Below, we show the full Helm-values YAML file.

```
# Choose an optional name for the datacenter
global:
  datacenter: minidc

# Enable the Consul Web UI via a NodePort
ui:
  service:
    type: 'NodePort'

# Enable Connect for secure communication between nodes
connectInject:
  enabled: true

# Enable CRD Controller
controller:
  enabled: true

client:
  enabled: true

# Use only one Consul server for local development
server:
  replicas: 1
  bootstrapExpect: 1
  disruptionBudget:
    enabled: true
    maxUnavailable: 0

# Sync catalog
syncCatalog:
  enabled: true
  default: false
  toConsul: true
  toK8S: true
```

Figure 11 - Helm-values YAML file.

Consul can now be installed on the Minikube node. It is worth to note that `helm-consul-values.yml` is the configuration file stored in the `k8` subfolder, `consul` is the name we gave to the Consul deployment on Kubernetes, and `hashicorp/consul` is the name of the fetched Helm chart.

```
File Modifica Visualizza Cerca Terminale Aiuto
mmartini@mmartini:~$ helm install -f ./k8/helm-consul-values.yml consul hashicorp/consul
```

Figure 12 - Configuration file stored in the `k8` subfolder.

```
File Modifica Visualizza Cerca Terminale Aiuto
mmartini@mmartini:~$ helm status consul
NAME: consul
LAST DEPLOYED: Fri Mar 26 12:14:54 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
Thank you for installing HashiCorp Consul!

Now that you have deployed Consul, you should look over the docs on using
Consul with Kubernetes available here:

https://www.consul.io/docs/platform/k8s/index.html

Your release is named consul.

To learn more about the release, run:

$ helm status consul
$ helm get all consul
mmartini@mmartini:~$
```

Figure 13 - Name of the Consul deployment on Kubernetes and of the fetched Helm chart.

The URL where Consul UI is exposed can be found through the `minikube service list` command.

```
File Modifica Visualizza Cerca Terminale Aiuto
mmartini@mmartini:~$ minikube service list
-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|-----|-----|-----|
| default | consul | No node port | |
| default | consul-consul-connect-injector-svc | No node port | |
| default | consul-consul-controller-webhook | No node port | |
| default | consul-consul-dns | No node port | |
| default | consul-consul-server | No node port | |
| default | consul-consul-ui | http/80 | http://192.168.49.2:32626 |
| default | kubernetes | No node port | |
| kube-system | kube-dns | No node port | |
| kubernetes-dashboard | dashboard-metrics-scraper | No node port | |
| kubernetes-dashboard | kubernetes-dashboard | No node port | |
|-----|-----|-----|-----|
mmartini@mmartini:~$
```

Figure 14 - Minikube service list command.

Because we haven't deployed any service yet, the only one listed is the Consul one.

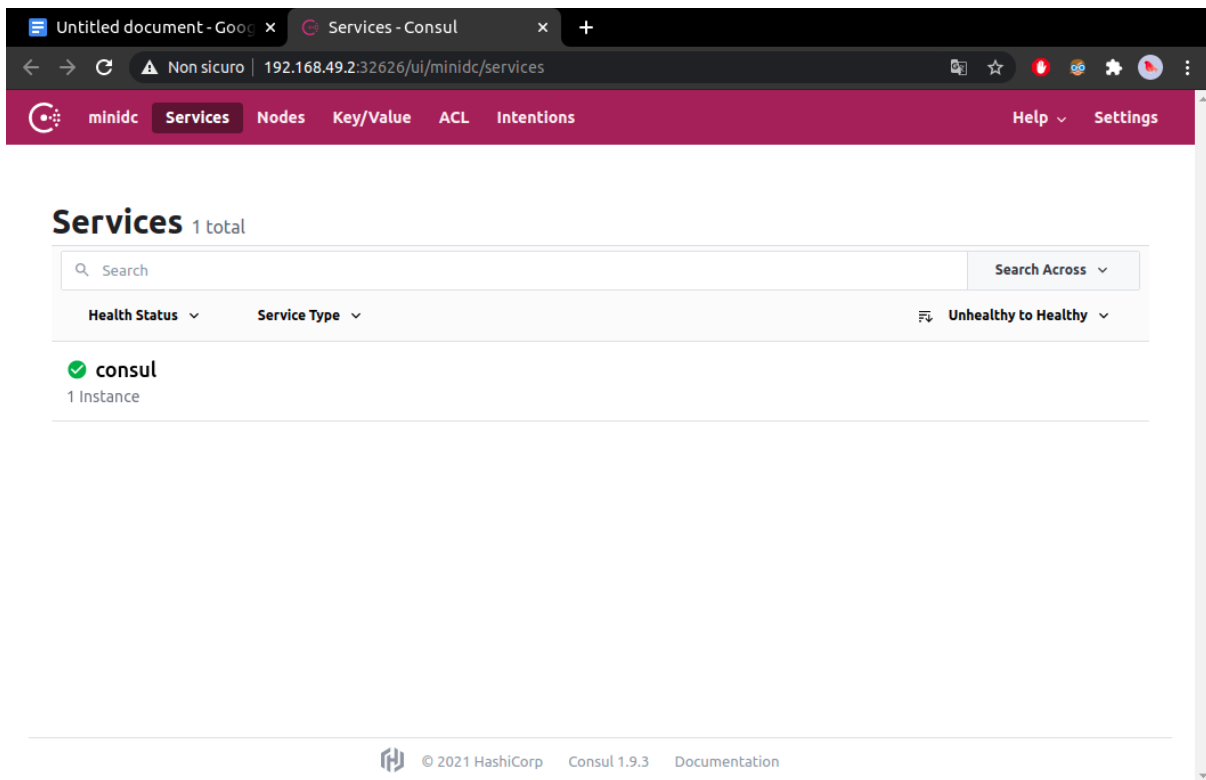


Figure 15 - List of deployed services.

2.4 Deploy MLflow model on Minikube

Our MLflow deployment consists of one application, the MLflow model, and a service that maps the pod's port. Below, we report the details.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mlflow-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mlflow-model
  template:
    metadata:
      labels:
        app: mlflow-model
    annotations:
      'consul.hashicorp.com/connect-inject': 'true'
```

```

spec:
  containers:
  - name: mlflow-model
    image: smartcommunitylab/mlflow-model:swagger
    imagePullPolicy: Never
    ports:
    - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: mlflow-service
  annotations:
    'consul.hashicorp.com/service-sync': 'true'
    'consul.hashicorp.com/service-meta-authentication': 'false'
    'consul.hashicorp.com/service-meta-authorization': 'false'
    'consul.hashicorp.com/service-meta-contextPath': ''
    'consul.hashicorp.com/service-meta-swaggerPath': '/swagger'
spec:
  selector:
    app: mlflow-model
  ports:
  - port: 8080
    targetPort: 8080

```

Figure 16 - MLflow deployment.

To enable communications between Consul Service Registry and Kubernetes, the standard way is to annotate deployments/services' metadata with a specific Consul's tag.

In particular, using the following annotation we have injected the pod with a "sidecar" Consul proxy.

```

annotations:
  'consul.hashicorp.com/connect-inject': 'true'

```

Figure 17 - Injecting the pod with "sidecar" Consul proxy.

This move allows encrypted communications between pods and external domains (Consul adopts TLS communication protocol to encrypt the data).

To synchronize the Kubernetes services with the Consul Service Registry, it is necessary to provide some further annotations to the configuration file.

```

'consul.hashicorp.com/service-sync': 'true'

```

Figure 18 - Synchronizing the Kubernetes services with the Consul Service Registry.

Doing this, we tell Consul to enable the synchronization of the “mlflow-service” service on its registry. Because we also must provide the INFINITECH Open API Gateway with some additional metadata, we include some more annotations under the service metadata. This allows the gateway application to get the metadata it needs.

```
'consul.hashicorp.com/service-meta-authentication': 'false'
'consul.hashicorp.com/service-meta-authorization': 'false'
'consul.hashicorp.com/service-meta-contextPath': /
'consul.hashicorp.com/service-meta-swaggerPath': /swagger
```

Figure 19 - Additional annotations for the INFINITECH Open API Gateway.

2.5 INFINITECH Open API Gateway

The INFINITECH Open API Gateway, developed within task T5.5, is a sophisticated gateway based on the Open API specifications. It provides a single point of entry for all ML/DL microservices, insulates the clients from the problem of determining the locations of the dynamically deployed microservice instances and it hides the exposure of internal concerns (i.e., service discovery and Open APIs documentation) to external Client Apps. The Gateway is deployed on the Kubernetes cluster with its own configuration file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway-depoyment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: api-gateway
  template:
    metadata:
      labels:
        app: api-gateway
    spec:
      containers:
        - args:
            - bash
            - -c
            - sleep 15 && java -Xms256M -Xmx1024M -jar api-gateway.jar
          env:
            - name: CIRCUIT_BREAKER_FAILURE_RATE_THRESHOLD
              value: "50"
```

```

- name: CIRCUIT_BREAKER_MINIMUM_NUMBER_OF_CALLS
  value: "10"
- name: CIRCUIT_BREAKER_PERMITTED_NUMBER_CALLS_IN_HALF_OPEN_STATE
  value: "10"
- name: CIRCUIT_BREAKER_SLIDING_WINDOW_SIZE
  value: "10"
- name: CIRCUIT_BREAKER_SLIDING_WINDOW_TYPE
  value: COUNT_BASED
- name: CIRCUIT_BREAKER_SLOW_CALL_DURATION_THRESHOLD
  value: 5m
- name: CIRCUIT_BREAKER_SLOW_CALL_RATE_THRESHOLD
  value: "50"
- name: CIRCUIT_BREAKER_WAIT_DURATION_IN_OPEN_STATE
  value: 30S
- name: CONSUL_HOST
  value: consul-consul-server
- name: CONSUL_PORT
  value: "8500"
- name: SPRING_PROFILES_ACTIVE
  value: prod
- name: TIMELIMITER_TIMEOUT_DURATION
  value: 10m
- name: TZ
  value: Etc/UTC
image: harbor.infinitech-h2020.eu/interface/api-gateway:0.2.2
name: api-gateway
ports:
  - containerPort: 8080
resources:
  {}
---
apiVersion: v1
kind: Service
metadata:
  name: api-gateway-service
spec:
  ports:
    - port: 50000
      targetPort: 8080
  type: LoadBalancer

```



```
selector:
  app: api-gateway
```

Figure 20 - Configuration file to deploy the INFINITECH Open API Gateway on Kubernetes.

A LoadBalancer service is put in front of the Open API Gateway pod to expose it outside the cluster. Using MiniKube CLI we can find the address where the service is listening. The Gateway can be reached on <http://192.168.49.2:30063>.

```
File Modifica Visualizza Cerca Terminale Aiuto
mmartini@mmartini:~/Virtualenvs/infinitech/demo/deployments$ minikube service list
-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|-----|-----|-----|
| default | api-gateway-service | 50000 | http://192.168.49.2:30063 |
| default | consul | No node port | |
| default | consul-consul-connect-injector-svc | No node port | |
| default | consul-consul-controller-webhook | No node port | |
| default | consul-consul-dns | No node port | |
| default | consul-consul-server | No node port | |
| default | consul-consul-ui | http/80 | http://192.168.49.2:30334 |
| default | kubernetes | No node port | |
| default | mlflow-model | No node port | |
| default | mlflow-model-sidecar-proxy | No node port | |
| default | mlflow-service | No node port | |
| kube-system | ingress-nginx-controller-admission | No node port | |
| kube-system | kube-dns | No node port | |
| kubernetes-dashboard | dashboard-metrics-scraper | No node port | |
| kubernetes-dashboard | kubernetes-dashboard | No node port | |
|-----|-----|-----|-----|
mmartini@mmartini:~/Virtualenvs/infinitech/demo/deployments$
```

Figure 21- LoadBalancer service.

Once the gateway is ready, outside the cluster we can deploy its UI. We used a docker-compose file to set the UI properties. Below, we can see the YAML.

```
version: "3.8"

services:
  api-gateway-ui:
    image: harbor.infinitech-h2020.eu/interface/api-gateway-ui:0.3.0
    container_name: api_gateway_ui
    ports:
      - "30000:80"
    environment:
      - GATEWAY_URL=http://192.168.49.2:30063
      - IMPLEMENTATION=INFINITECH
    logging:
      options:
```

```
max-size: "10MB"  
max-file: "10"
```

Figure 22 - Docker-compose file to set the UI properties.

The UI connects to the Gateway address. With a browser we can navigate the UI.

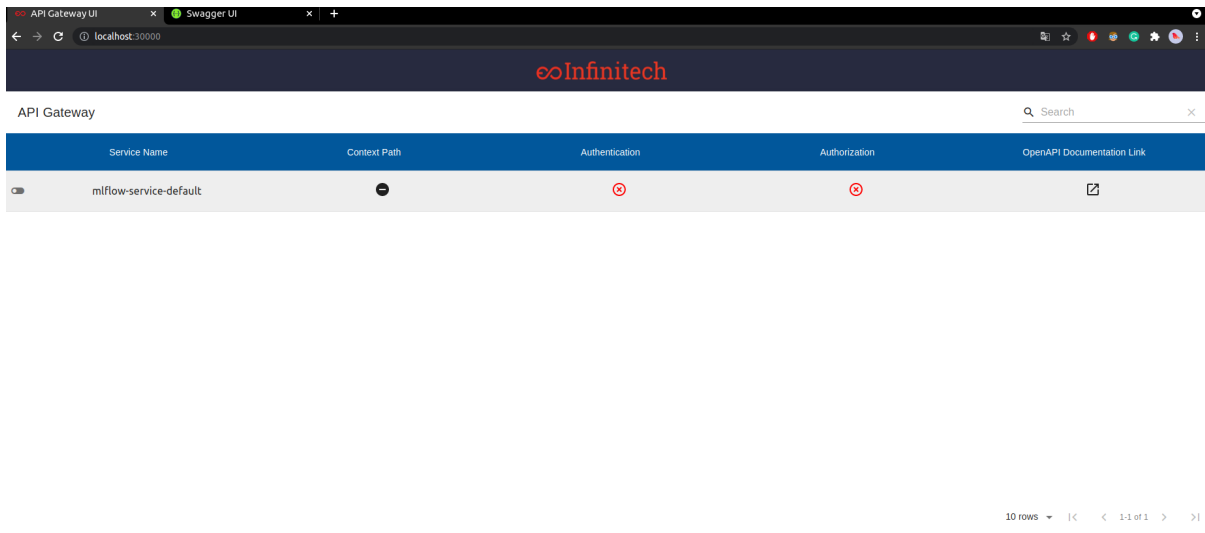


Figure 23 - Browser to navigate the UI.

As we can see, our MLflow microservice is listed. We can expand the Swagger documentation with the flag on the left or open it through the link on the right. As mentioned before, an MLflow docker model exposes only one invocation endpoint.

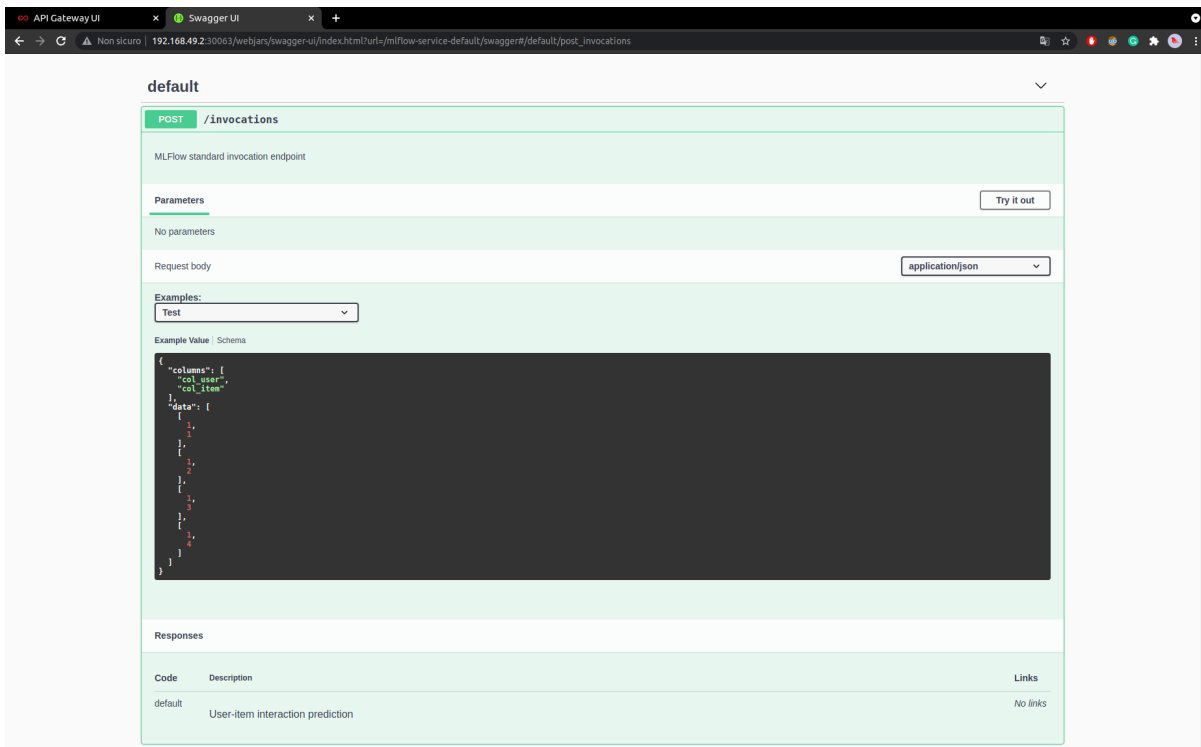


Figure 24 - Invocation endpoint.

In this demo, we used a Neural Collaborative Filtering [1] model developed by the University of Glasgow (GLA). This model expects as input data a list of users' and items' ids, and it tries to predict the likelihood that one user interacts with a given item.

We can now send data to the model through the Gateway endpoint. We can call the endpoint and perform a POST request with CURL, a command-line tool for getting and sending data using URL syntax.

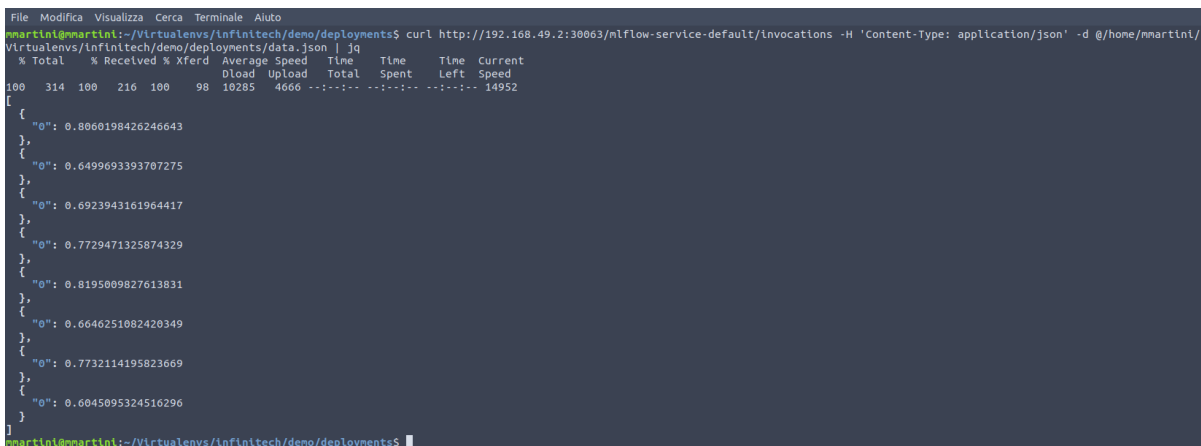


Figure 25 - Calling the endpoint with a CURL POST.

3 INFINITECH pilots' ML/DL tasks: Progress and preliminary results

In this section, we describe in detail the progress of the ML/DL tasks that the INFINITECH pilots have designed. In particular, we focus ourselves on the preliminary results (e.g., novel ML/DL algorithms implemented within the pilot, state-of-the-art ML/DL algorithms evaluated within the pilot, initial performances obtained, etc.) as well as on the type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms (e.g., pre-built/pre-trained ML/DL models, ML/DL frameworks with data to perform the training step, only ML/DL frameworks and results obtained on proprietary data, etc.). For an overview list of the types of pilots' tasks and of the ML/DL algorithms pilots plan or have implemented see Table 1.

It is worth noting that a couple of pilots (i.e., Pilot 1 and Pilot 3) are currently in a reorganization phase due to the change of some partners and thus they will not be discussed in the current deliverable.

Table 1 – INFINITECH Library: overview list of ML/DL algorithms

Pilot	ML/DL algorithm	Finance problem	Tags/ characterization	ML/DL Framework
2	Homegrown/Standard	Risk assessment	DL, Monte Carlo simulations	GluonTS, MxNet
4	Homegrown	Portfolio Management	Genetic algorithms	Not specified
5B	Homegrown	Financial Management	ML, DL	Scikit-learn, Statsmodels, Keras, Pytorch, Tensorflow, GluonTS
6	Homegrown/Standard	Portfolio Management	ML	imblearn, Scikit-learn
7	Standard	Financial Crime	ML	Scikit-learn
8	Standard	Anti Money Laundering	ML, DL	Scikit-learn, other libraries not specified
9	Homegrown/Standard	Blockchain fraudulent activities	Graphs, ML	imblearn, Scikit-learn
10	Standard	Cyber security analytics	ML	Spark-ML, Scikit-learn, Keras
11	Standard	Personalized insurance products	ML	Scikit-learn, TensorFlow, Keras
12	Standard	Real world data for health insurance products	ML/DL	Scikit-learn, TensorFlow, Keras

13	Standard	Insurance product recommendation	ML/DL, Reinforcement Learning	Not specified
14	Standard	Big Data and IoT for agricultural insurance	ML	Orfeo Toolbox, ML libraries in R
15	Standard	Open Inter-Banking	DL	Not specified

3.1 Pilot 2 - Real-time risk assessment in investment banking

This pilot implements a real-time risk assessment and monitoring procedure for two standard risk metrics – *Value-at-Risk* (VaR) [2] and *Expected Shortfall* (ES) [3]. Both the metrics can be applied for measuring various types of risk, such as the market risk of portfolios of assets. The pilot implements both risk metrics for estimating market risk and it allows updates with changing market prices and/or changes in the bank's portfolio in (near) real-time. To this end, the pilot leverages real-time signals of assets that comprise a Forex (Foreign Exchange) portfolio. In addition, it implements the evaluation of what-if-scenarios allowing pre-trade analysis, i.e. estimating changes in risk measures before a new trading position is entered. Finally, the pilot will also provide sentiment analysis, leveraging Natural Language Processing (NLP) techniques, on financial news feeds as an extra risk indicator.

The innovation of the pilot is twofold: first of all, it adopts real-time analytics over Forex portfolios to empower traders to make accurate decisions on-line. Second, it illustrates different ways for calculating mainstream risk parameters (i.e., VaR), including methods based on scientific/statistical computing (e.g., Monte Carlo simulations) and machine learning approaches (i.e., DeepVaR).

The pilot uses price data for the most liquid Forex, Stocks, Stock Indices and Derivatives. The initial focus of the pilot is on Forex assets, yet its extension to other types of portfolios and securities will be straightforward. Additional data (i.e., financial news) will be considered at the late stages of pilot deployment in order to incorporate the sentiment analysis feature.

To calculate risk metrics (such as VaR) and to conduct pre-trade analysis, the pilot leverages two different approaches:

- **Scientific/Statistical Computing Approach:** This approach is based on purely statistical methods for computing the VaR, namely, parametric, non-parametric, and semi-parametric methods [4]. Specifically, the utilized statistical methods for VaR calculation are:
 - *Variance-Covariance:* assuming that the portfolio returns follow normal distribution, VaR is obtained after calculating the returns' variance-covariance matrix.
 - *Historical simulation:* VaR calculation is based on the historical performance of the given portfolio.
 - *Monte Carlo simulations:* Based on some assumptions, this procedure randomly draws the portfolio returns' distribution, which is used to calculate the VaR.
- **Machine Learning Approach:** Pilot 2 introduces a novel semi-parametric VaR model, the so-called DeepVaR, combining deep neural networks with Monte Carlo simulations. In this approach, the parameters of the returns' distribution are initially estimated by a Recurrent Neural Network (RNN) based model, with the network output being used to predict all possible future returns in Monte Carlo fashion. The RNN-model in question follows the methodology proposed by [5] and is provided by GluonTS Python library for deep learning-based time series modeling [6] as DeepAR estimator. This model is able to capture various nonlinear dependencies of the input time-series, such as seasonality,

resulting in consistent quantile estimates. Moreover, DeepAR could be fed with several input time-series simultaneously, enabling cross-learning from their historical behaviour jointly. As a result, changes in the dynamics of one time-series may affect the predicted distributions of the other time-series.

Moreover, the pilot will be based on state-of-the-art ML algorithms and technologies (e.g., CNNs, Transformers, Transfer Learning) for NLP tasks to offer sentiment analysis functionality. However, this component is still under development and the exact ML algorithm that will be utilized is under investigation, as the current pilot prototype is based only on the market prices, and not on textual data, to offer risk assessment.

At the current stage, within Pilot 2 INNOV has developed a microservice which do the following steps:

- 1) it takes as input the trading positions on some financial assets;
- 2) it reads the historical prices of these assets from a database (this step is done by LXS);
- 3) it returns an estimation of VaR and ES.

The performance consists of more than 1% of daily VaR/ES violations per year, estimated with a confidence level of 99% (mean value between the portfolios).

The python libraries used in Pilot 2 are:

- **Docker**: A platform for docker-container building/running/distributing.
- **Dash**: Python framework for building web analytic applications.
- **GluonTS**: Probabilistic time-series modelling framework for time-series forecasting.
- **MxNet**: Deep Learning library.

Type of contribution to provide to the INFINITECH library of ML/DL algorithms

The contributions planned by Pilot 2 are the following:

- **DeepVaR**: develop a ML/DL framework with results obtained on proprietary data.
 - *Input*: Historical asset prices, Portfolio positions on each asset, confidence probability of VaR/ES.
 - *Output*: VaR/ES estimation.
- **Tool for sentiment analysis** on financial news using a pre-trained model
 - *Input*: Text (e.g., title of FX news).
 - *Output*: Sentiment Score (negative, neutral, positive).

3.2 Pilot 4 - Personalized portfolio management - mechanisms for an AI-based portfolio construction

The main goal of this pilot is to develop and adapt within a wealth management platform, called “Privé Managers”, an *optimization algorithm* (further on called “Privé Optimizer AIGO”), as well as

improving and expanding its capabilities as an AI engine to aid investment propositions for retail clients.

The AI-based portfolio construction will enable advisors and/or end-customers to use the existing wealth management platform and make use of its risk-profiling and investment proposal capabilities, starting from customers’ risk awareness. AIGO allows for a variety of use cases which cater to the needs of financial advisors, end-clients and financial firms alike.

The innovative AIGO genetic algorithm can be used for proposing investments, and evaluating them given an easy-to-use, personalizable set of criteria, in the form of “fitness factors”. These “fitness factors” will be used to generate “health” scores, which are used to define the “fittest” investments.

Both quantitative and qualitative historical data will be leveraged for personalizing the portfolio construction and investment proposals, rendering the journey clear from a regulatory perspective.

The highlights of this methodology lay in four different characteristics:

- **Flexibility in the fitness factors used to define the objectives of the optimization process.** Users can design and assign weights to those factors that are important to investors. It gives a place to the users to integrate their market views and also a feature “frozen” to keep the holdings an investor wants to stay untouched. With the flexibility of this methodology, the optimizer can provide a highly customized solution depending on the clients’ needs.
- **Convergence speed to reach a recommended portfolio.** Given that there are multiple angles that need to be optimized and there is a huge product universe to select from, the optimizer can provide a solution with an average 5.4 second response time.
- **Compatibility with other optimization methods and other strategies.** For example, a traditional mean-variance optimization can be defined by a Sharpe Ratio Factor (SRF) [30], by which the optimizer will recommend a portfolio with the highest sharpe ratio, or a strategy such as a product diversification can be defined by a product diversification allocation factor (AFAC), by which the optimizer can recommend a portfolio according to the product diversification strategy.
- **Multi-asset support.** This methodology can work with different asset classes and product types including stocks, bonds, funds and structured products.

The typical steps of using *Privè Optimizer* are:

1. provide current portfolio holdings details;
2. specify their preference including account currency, preferred currency(ies), fitness factors, weights, etc.;
3. specify product universe (a product universe is a pool of assets which the optimizer can select from);
4. send the request through *Privè Optimizer* API or UI;
5. *Privè Optimizer* generates and returns the recommended portfolio through the developed algorithm.

Below we provide the details of the *patented optimization algorithm*:

1. after receiving the inputs, the 1st generation of portfolios is generated;
 - 1.1. there will be 256 portfolios in every generation;
 - 1.2. a random generation process is used;
 - 1.3. the first portfolio of the first generation will be the input portfolio (from the data input);
2. the assets and their weights within the portfolio are randomly assigned;
3. the Fitness Score, which is the weighted aggregate of different factors, is calculated for all portfolios. The portfolios are then ranked according to their scores;
4. portfolios with fitness score in the top 50% range will be selected, the bottom 50% will be disposed of;
5. the surviving 50% (128) portfolios will be treated as parents to generate 128 offspring; each child is generated by two adjacent parents, for example, the 1st portfolio will be combined with the 2nd one, the 2nd one with the 3rd one and so on. The 128-th portfolio will pair with a randomly created new portfolio;
6. mutation is performed by introducing assets, not from existing portfolios;
7. step 2 to step 6 will be repeated for 100 times;
8. the portfolio with the highest fitness score is selected as the final output.

Table 2 - Example of multi-factor aggregation table for fitness computation of customer portfolio.

	Weighted Fitness Factor 1	Weighted Fitness Factor 2	Weighted Fitness Factor 3	...	Fitness-Score (Weighted Sum of Factors)
Portfolio 1	0.13	0.29	0.14		0.88
Portfolio 2	0.10	0.07	0.03		0.24
Portfolio 3	0.34	0.16	0.21		0.90
...					
Portfolio 254	0.01	0.01	0.00		0.05
Portfolio 255	0.03	0.06	0.05		0.20
Portfolio 256	0.29	0.18	0.31		0.83

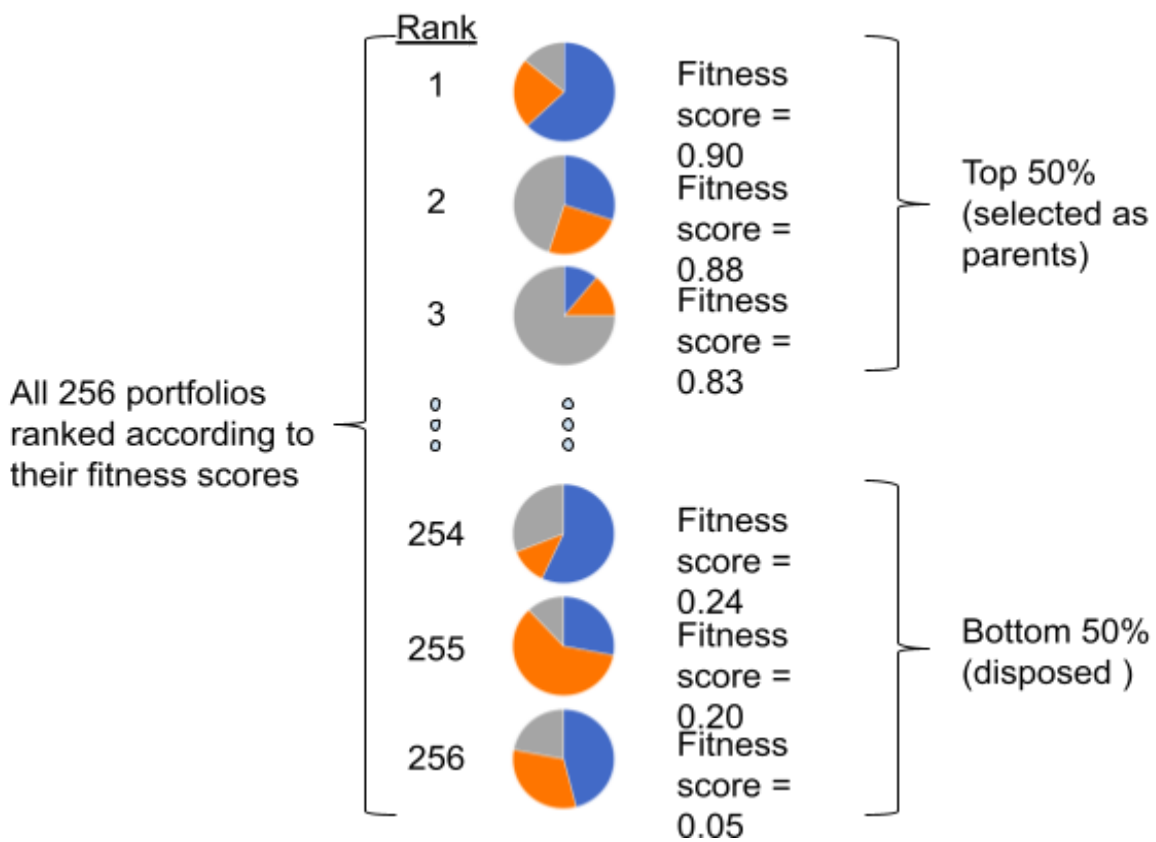


Figure 26 - Schematics of portfolio selection based on fitness score.

Testing of the algorithm to generate the best optimized portfolio.

By optimizing a given asset universe given a set of fitness factors and their respective weights, we are able to generate a total portfolio fitness score which represents how closely the optimized portfolio has aligned with our target preferences.

From the results tabulated in Table 3, we can see that the optimized portfolio score is maximized. In the results we run a high weight on the volatility target, thus we can see that for extremely small volatility targets, the portfolio depends highly on the number of low volatility assets used in the asset universe for optimization.

Table 3 - Test cases showing portfolio score of generated portfolios through optimization.

Volatility Target	Factor 1 weight	Factor 2 weight	Factor 3 weight	Factor 4 weight	Factor 5 weight	Total fitness score
0.03	0.55	0.075	0	0.06	0.315	25.19%
0.03	0.55	0.075	0	0.06	0.315	31.42%
0.03	0.55	0.075	0	0	0.375	32.35%
0.03	0.55	0.075	0	0	0.375	49.68%
0.03	0.55	0.075	0.075	0.06	0.24	51.13%
0.03	0.55	0.075	0.075	0.06	0.24	48.07%
0.03	0.55	0.075	0.075	0	0.3	48.29%
0.03	0.55	0.075	0.075	0	0.3	51.39%
0.03	0.55	0.075	0	0.06	0.315	51.28%
0.07	0.55	0.075	0	0.06	0.315	70.26%
0.07	0.55	0.075	0	0	0.375	77.39%
0.07	0.55	0.075	0	0	0.375	73.20%
0.07	0.55	0	0.075	0.06	0.315	68.73%
0.07	0.55	0	0.075	0.06	0.315	75.23%
0.07	0.55	0	0.075	0	0.375	71.44%
0.07	0.55	0	0.075	0	0.375	75.33%
0.07	0.55	0	0	0.06	0.39	76.84%
0.07	0.55	0	0	0.06	0.39	73.65%
0.1	0.55	0	0	0	0.45	73.99%
0.1	0.55	0	0	0	0.45	80.65%

0.1	0.55	0	0.075	0.06	0.315	79.33%
0.1	0.55	0	0.075	0.06	0.315	76.10%
0.1	0.55	0	0.075	0	0.375	81.68%
0.1	0.55	0	0.075	0	0.375	79.42%
0.1	0.55	0	0.075	0.06	0.09	82.58%
0.1	0.55	0	0.075	0.06	0.09	87.93%

3.3 Pilot 5B Financial Management (BFM) tools delivering smart business advice

This pilot aims at building AI-powered Business Financial Management (BFM) tools that assist Small and Medium Enterprises (SMEs) clients of the Bank of Cyprus (BOC) in managing their financial health by providing assistance with activities in the area of cash flow management, continuous spending/cost analysis, budgeting, revenue review, or Value Added Tax (VAT) provisioning. BFM tools unlock the potential of AI and harness the power of data to generate personalized actionable business insights. More precisely, the BFM tools aim at driving the SME digital adoption rate as well as paving the ground for reducing credit risk, lowering the amount of Non-Performing Loans (NPLs) and for vital needed improved/streamlined SME lending.

The system will integrate multiple BFM tools, such as cash flows, health status, benchmarking, budgeting, into a global recommender unit. The *recommendation system* will be based on data from three main pillar sources: (i) *Market data*, (ii) *Bank data* (e.g., customer data), and (iii) *SMEs data* (e.g., SMEs taxation data).

This stream of data will be then used as input of the ML algorithms at the core of the pilot analytics. In particular, the ML concept constitutes one of the innovative aspects of the pilot's approach. It leverages multiple data sources to generate actionable insights on key BFM aspects. The tasks on which ML will be used to construct the smart virtual advisor are the following ones: (i) categorization engine, (ii) cash flow prediction, (iii) budget prediction, (iv) key performance indicators (KPIs) monitoring, (v) taxation monitoring and categorization, (vi) invoices processing, and (viii) benchmarking.

The smart virtual advisor will merge and process insights to transform them into “recommended actions”, which will construct the output of the coupled data-analytics system.

Many of the components above will be interconnected to provide holistic business information, alerts and recommendations. Here we summarize the technologies, tools and libraries used to develop the data analytics' and ML components:

- **Docker:** A platform for docker-container building/running/distributing.
- **Statsmodel:** Time-series analysis and forecasting
- **Scikit-learn:** Development of ML models and model testing.
- **Keras, PyTorch, TensorFlow:** DL platforms utilized for the recommender engine.
- **Apache Kafka:** Open-source stream-processing software platform.
- **Apache Spark:** Open-source distributed general-purpose cluster-computing framework.
- **GluonTS:** Probabilistic time-series modelling framework for cash flow forecasting

ML algorithms and technologies will be utilized for most of the pilot's analytical components. In more detail, the *Transaction Categorization Engine*, a key component for the development of the rest of the tools, follows a hybrid approach combining rule-based categorization to label a high percentage of SME

transactions. The transactions which are not categorized through the rule-based approach are then fed into an ML model, as depicted in Figure 27.

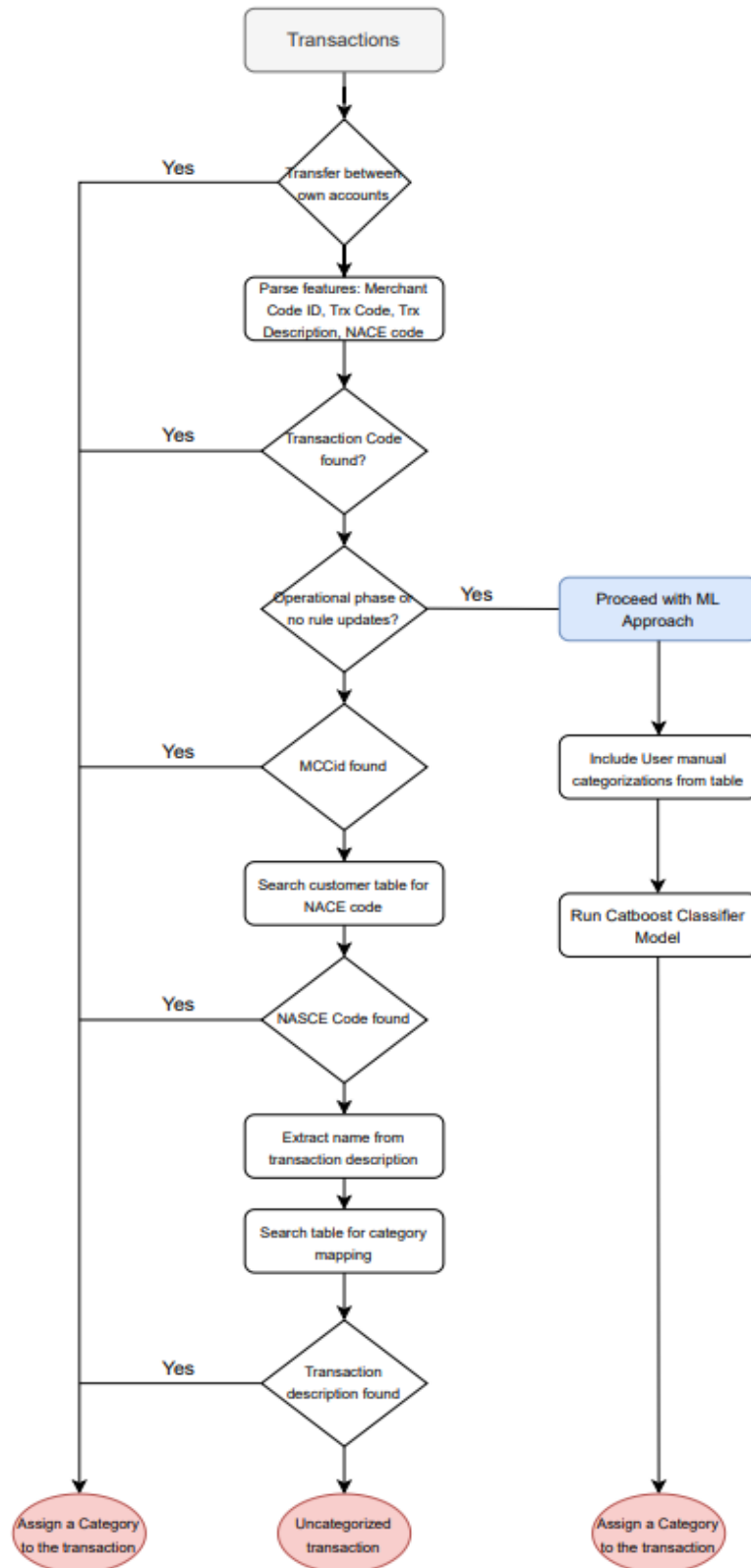


Figure 27 - Hybrid classification model flowchart.

Rule-based model: the labels of the dataset were based on the outputs of a rule-based model. For this case the evaluation is referring to the coverage of the dataset that can be accomplished based on the hand labels assigned by the experts and it reaches the 95% approximately. Apart from the coverage of each step, it is of high interest the distribution of the transactions among the provided categories. The final results in terms of percentage are depicted in Figure 28.

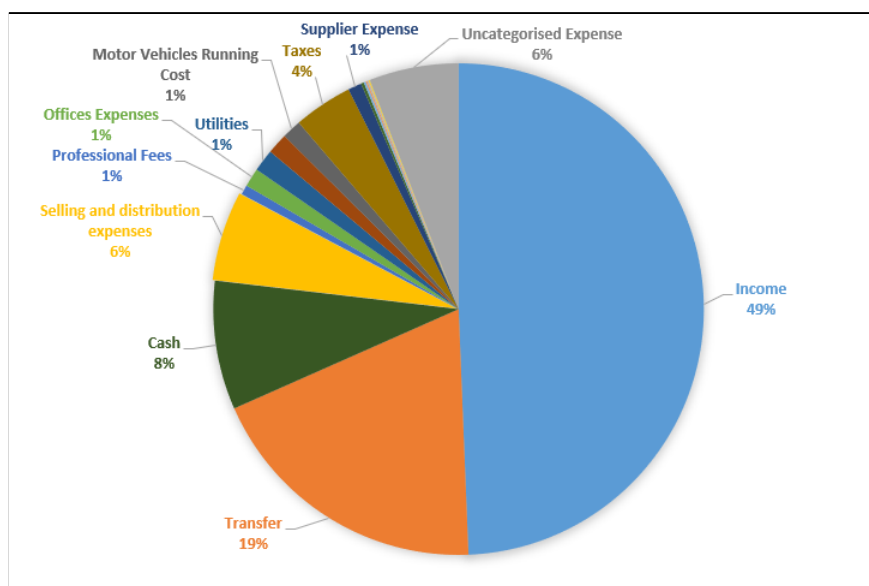


Figure 28 - Rule-based step approach categorization results.

Catboost classifier: in order for this approach to be effective and at the same time to alleviate the bias inserted by the rule-based model, it is crucial to give the capability to the users to change a transaction category. To this end, a Catboost [7] model was trained in a way of adopting the changes made by the users. To be more effective, and given the nature of the different steps utilized in the rule-based model, a hybrid-model was preferred instead of a fully-ML one, as a result of the confidence provided by the categorization based on the first 2 steps of the rule-based model.

It is evident that the scope of the utility of this model is to mimic (learn) the outputs of the last 3 steps of the rule-based model. The number of the legit labels (i.e., master categories) that can be produced in these steps are 16, so the evaluation of multi-class tasks in terms of various metrics was applied.

In the context of this pilot, the interpretation of the results is more important than the results themselves in order to obtain insights useful both in a technical and business perspective. In general, explainable methods can be categorized in 2 general classes: (i) ML models’ built-in feature importance, and (ii) post-hoc feature importance utilizing models such as LIME (Local Interpretable Model-agnostic Explanation) [8] and SHAP (SHapley Additive exPlanations) [9]. In the examined case, both LIME and SHAP techniques were leveraged as a qualitative evaluation of the results. The SHAP values denoting the importance of each feature included in the Catboost model are depicted in Figure 28. It is evident that the model learned the rules based on MCCCodeID and Recipient NACE code as the two most important features. Moreover, skAccKey is the third most important feature and thus it strengthens the proposed approach of a user-oriented updating

methodology. Finally, apart from the feature importance based on the SHAP analysis, it is of high importance to qualitatively check some of the outcomes based on the LIME, a recent technique capable of explaining the predictions of any classifier or regressor using local approximations (sample-based) of the original model with other interpretable models.

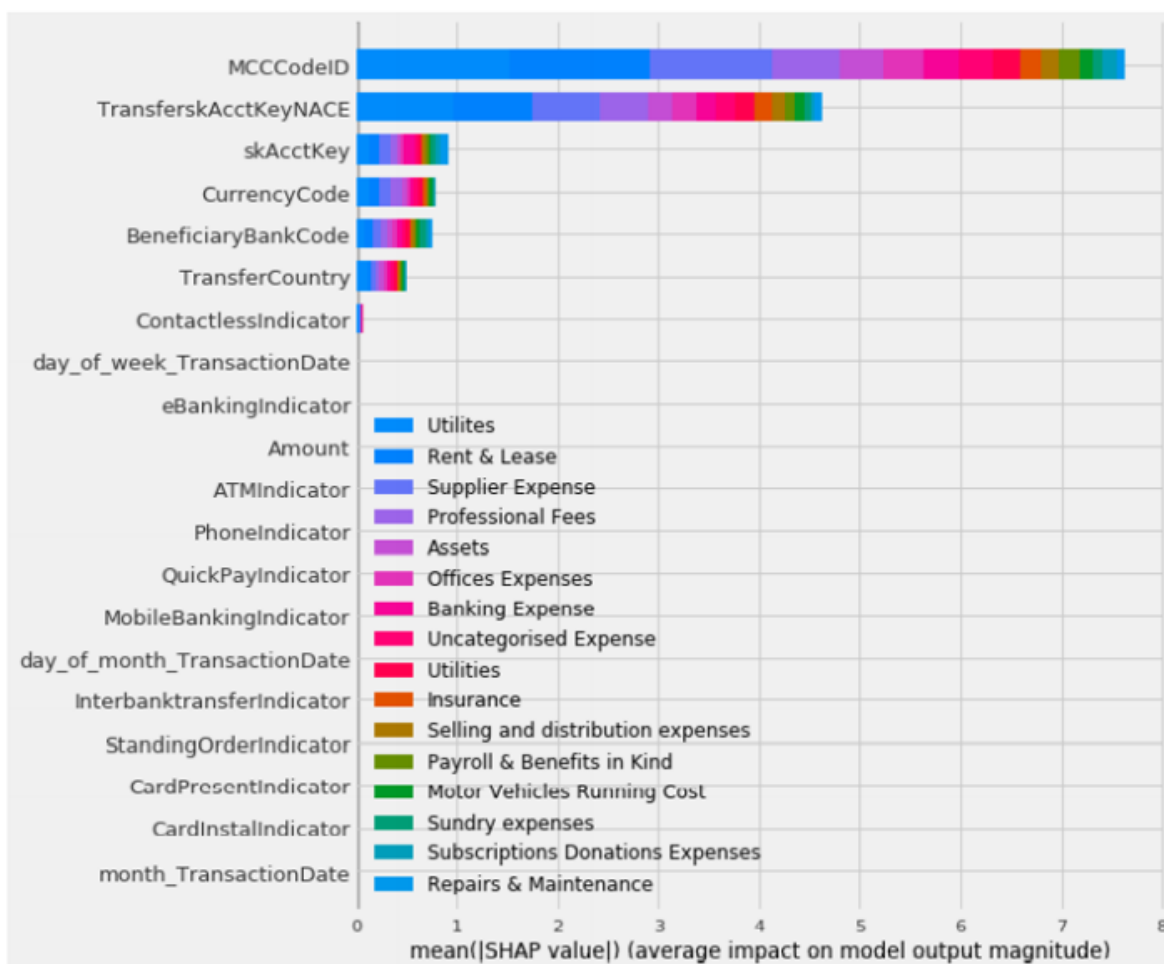


Figure 29 - SHAP values of each feature included in the Catboost model.

Regarding the second tool (i.e., SMEs’ cash-flow prediction), the way the task and the objective were modelled implied the data transformation in time-series representation. To this end, both resampling and aggregating the amount of the transactions based on each specific account and date have been applied.

As expected, various "traditional" challenges regarding time-series forecasting arose that should be tackled in the context of our research and development. These challenges can be summarized in (i) the "cold start" problem which refers here to the time-series that have small number of transactions, (ii) the stationarity-seasonality trade-off where it is assumed that in order to have predictable time-series they have to be stationary (i.e., without trend and seasonality factors present), (iii) the presence of noisy data or even many outliers, and finally (iv) the adequate length of the dataset in order to apply ML/DL techniques.

To this end many measures were taken such as setting a minimum number of executed transactions during the examined historical period as a threshold, and only the time series that have more transactions than this threshold were utilized. Furthermore, in order to leverage the DeepAR model we have re-sampled the

data in a weekly time frame getting the aggregation of the amount of the transactions made for a specific category.

Usually, ML cross-validation methods reflect a pitfall in a time series forecasting approach, as they may result in a significant overlap between train and test data. Thus, the optimum approach is to simulate models in a "walk-forward" sequence, periodically re-training the model to incorporate specific chunks of data available at that point in time. This procedure is depicted in Figure 30.

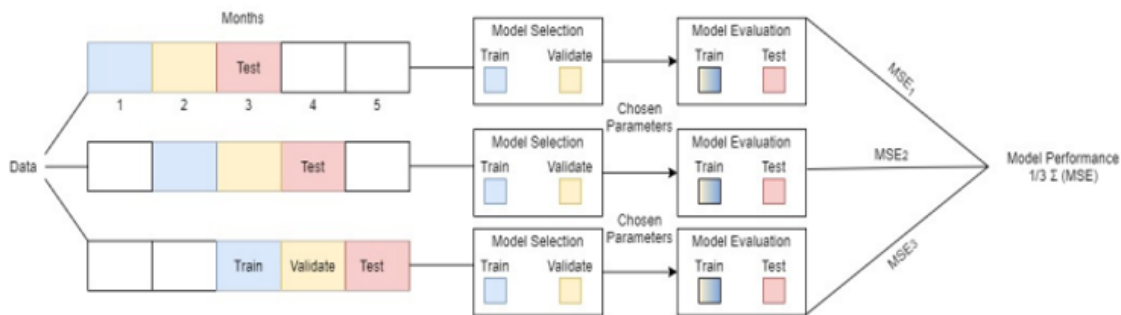


Figure 30 - Proposed one step forward validation scheme.

Table 4 shows the initial results obtained using the original data, the data obtained after applying the pre-processing steps described previously, and the enriched dataset obtained using surrogate data.

Table 4 - Initial results of the cash flow prediction mode.

category	seasonal_error	MASE	sMAPE	MSIS	QuantileLoss[0.5]	Coverage[0.5]	QuantileLoss[0.8]	Coverage[0.8]	QuantileLoss[0.95]	Coverage[0.95]
Assets	425.004451	0.955994	1.940701	26.064849	2401.091588	0.466529	2900.684391	0.722597	2367.565423	0.871434
Banking Expense	694.784455	72.957481	1.910183	2478.095219	3340.891748	0.477389	4437.331438	0.746194	4268.931029	0.891764
Cash	4065.432897	0.787062	1.658916	18.493877	25428.835599	0.515232	31496.964056	0.755126	26010.145812	0.882095
Insurance	156.242906	1.01829	1.921569	23.23016	964.912578	0.464005	1150.166498	0.724043	908.232181	0.872601
Motor Vehicles Running Cost	600.610689	3.625093	1.702401	50.167932	3599.007606	0.442301	3863.190312	0.730953	2646.93137	0.880811
Offices Expenses	249.59533	3.975575	1.829996	57.882584	2379.410203	0.470939	3007.474585	0.736683	2660.359016	0.879383
Payroll & Benefits in Kind	382.469903	1.972266	1.940531	46.08671	3326.830876	0.468127	4666.329313	0.730176	4582.028392	0.874585
Professional Fees	2117.501017	7.830937	1.818849	265.033923	16769.57321	0.48071	19810.913754	0.743384	15255.594114	0.884735
Rent & Lease	803.825233	0.938132	1.944683	15.177176	4093.789031	0.465478	4920.855602	0.709043	3708.97372	0.856373
Repairs & Maintenance	210.978409	7.209771	1.945956	54.962608	1815.282686	0.487616	2171.715104	0.740278	1693.385367	0.879977
Selling and distribution expenses	773.484602	5.174809	1.683581	110.461375	4951.311866	0.490133	5679.797721	0.751037	4288.930704	0.887629
Subscriptions Donations Expenses	74.533946	25.023398	1.920883	215.479253	916.44074	0.472896	1052.95497	0.731465	776.814175	0.880421
Supplier Expense	2107.319424	2.459285	1.849253	44.455763	18531.24653	0.463676	19777.1842	0.726911	14281.448688	0.872471
Taxes	2786.071249	0.403885	1.825333	6.073532	13210.548795	0.366667	12938.64653	0.571296	8515.301325	0.801852
Transfer	8942.757042	3.394084	1.577965	116.27894	57575.255998	0.494688	63056.221555	0.751048	43652.938185	0.888007
Uncategorised Expense	11734.93409	2.673312	1.634515	59.273003	72403.651755	0.483415	76042.949449	0.741045	49743.826605	0.881903
Utilities	291.783627	4.429108	1.848284	76.449565	1855.911632	0.46859	2126.565573	0.731943	1604.632641	0.878192

At the current stage, the developed analytical services work as ReST microservices, which means that both of them are triggered via a custom API, with a POST https request using a JSON format body. It should be highlighted that the models were re-trained every week and saved in the persistent volume of the BoC sandbox. The inference of the new validation data is provided automatically and internally in the sandbox. Then, the results are stored in the INFINISTORE and served when requested by the users, who are provided with the appropriate credentials, via the custom API.

3.4 Pilot 6 - Personalized closed-loop investment portfolio management for retail customers

Pilot 6 focuses on providing personalized investment recommendations for the retail customers of the National Bank of Greece (NBG). NBG leverages large customer datasets and large volumes of customer-related alternative data sources (e.g., social media, news feeds, on-line information) to make the process of providing investment recommendations to the retail customer more targeted, automated, effective, as well as context-aware (i.e., tailored to the state of the market).

More specifically, the pilot uses data related to investment products' clients from the NBG data warehouse:

- customer Relationship Management (CRM) data;
- deposit account transactions;
- card transactions;
- investment related transactions;
- Mifid (Markets in Financial Instruments Directive) questionnaire data.

These different sources of data are being used in order to feed a **ML-driven Customer Investment Risk Appetite Profile algorithm**. As an outcome, the algorithm divides all customers in specific profile clusters based on the investment and banking behaviour.

Based on the (i) *customer risk profile* clusters and (ii) the relative mapping of financial instruments vs customer risk profile data provided from the bank, we feed the **ML-driven Personalized Investment Recommendation algorithm**, in order to provide personalized recommendations for each customer on the most suitable and profitable financial instruments.

In order to enhance the results of the **Personalized Investment Recommendation algorithm** the recommendations will take into account data based on news and social media leveraging sentiment analysis, for each of the financial instruments that the algorithm selects and for each customer, thus also providing an expected suitability score.

In summary, the pilot aims at utilizing ML tools and algorithms for the development of two main components:

- a. **Customer Investment Risk Appetite Profiling;**
- b. **Personalized Investment Recommender.**

The final objective is to develop for NBG an application that will allow account officers to provide more personalized recommendations for specific portions of the clients that are interested in investing in various financial instruments available from the bank.

The pilot splits into a few main parts as described below.

The first service of the pilot is the generation of the **Customer Investment Risk Appetite Profile algorithm**, using mainly transactional data. This prediction is an important input information for the second service of the pilot which is the **Recommendation algorithm**.

The first service predicts whether any customer, either with or without investment activity, would belong to one of 4 possible risk appetite profiles. The 4 target classes for the first service are:

- conservative;
- income;
- balanced;

- aggressive;

In the case of significant class imbalance, e.g. the aggressive class being underrepresented, it is recommended to consolidate two classes autonomously, and the underpopulated class merged to its adjacent one, resulting in 3 final classes.

In general pilot's implementation uses for the ML part the following Python modules:

- Scikit-learn;
- Pandas;
- NumPy;
- Imblearn (for target class balancing).

For the first service, we start with a *data aggregation step*.

In our latest effort, the inner bank IT systems build and provide suitable Knowledge Performance Indicators (KPIs) that project/transform the transactional information into customer level.

Our KPIs include:

- basic aggregations (e.g., sum, counts, medians, standard deviations, etc.);
- time windowed aggregations (i.e., 3, 6, 9 and 12 months);
- aggregations per value of important nominal attributes, aggregations per user;
- percentages;
- quality attributes like salaries and pensions, spending ratios, etc.

The second step consists in *data cleaning* and more precisely in (i) handling extreme and missing values, and (ii) removing non-useful attributes.

Then, we have a *feature engineering* task consisting of the following steps:

- discretize semi-empty attributes so that all information is used;
- standardize all numeric features before use, so that relative magnitudes don't influence models;
- apply one-hot encoding (dummy values) on nominal values;
- test various feature transformation techniques provided by Scikit-learn, like PCA (Principal Component Analysis), TruncatedSVD, etc.

Finally, we have a *model inference and hyperparameter tuning* task consisting of the following steps:

- the evaluation process of the models uses both cross-validation and a simple Percentage-Split testing. In order to oversample the training data, the usage of Imblearn's SMOTE technique [10] is selected, so that to have balanced classes in the training set, as long as enough data to train the models on;
- testing various feature selection techniques provided by Scikit-learn such as SelectKBest & Chi2, SelectKBest & ANOVA, etc.;
- tuning, using grids, the parameters of various Scikit-learn models such as Logistic Regression, Random Forest, Decision Trees, AdaBoost, Neural Network Multilayer Perceptron (MLP);
- creating predictions on a big section of our experienced customers as well as our retail customers.

It is worth noticing that the first service has a multiclass problem to handle. However, many of Scikit-learn's popular techniques and functions do not handle the multiclass problem. During the inference stage, these techniques handled the multiclass problem either by using algorithms that handle this task (e.g., Logistic Regression and MLP), either by using `sklearn.multiclass.OneVsRestClassifier`, which is a meta-classifier that uses a simple classifier and applies the one-vs-all strategy, which consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. Each instance belongs to the predicted class by the classifier with the smallest distance value.

In order to evaluate our models, we look at the generic picture of and compare the following metrics on the test set:

- Precision and Recall per class;
- overall Accuracy;
- One-vs-One ROC AUC scores: macro and weighted by prevalence;
- One-vs-Rest ROC AUC scores: macro and weighted by prevalence;
- Cohen Kappa score.

We have created various experimentation machine learning cycles, mainly focusing on improving results through the creation of new innovative features. A final model has been chosen, while the predicted labels are indeed in agreement with the actual investment activity for the customers that are labeled and already have investment activity.

For the second service, the **Personalized Investment Recommender**, Pilot 6 fully leverages on the BetaRecSyS framework that has been developed by GLA for the INFINITECH purposes (open-source project for building, evaluating, and tuning automated recommender systems).

The two aforementioned services are being developed in parallel and as a next step the predicted customer risk profiles will be merged with the most suitable investment assets per customer.

Initial performances obtained

We provide an aggregated summary of results as they occurred on an inference set of about 15k customer data. In particular, we obtain an Accuracy value of 0.431, a Precision value of 0.421, a Recall value of 0.431, and an F1-Score value of 0.422.

NBG and Pilot 6's contribution to the INFINITECH library of ML/DL algorithms and to the INFINITECH marketplace consists in providing:

- ML model input data (NBG non-investment transactional 2018 anonymous data);
- ML model output data (risk appetite prediction labels on customers);
- a detailed explanation of each step of the adopted methodology.

Currently, Pilot 6 does not provide a live API, while NBG will make it available for everyone with testbed access in our ML Virtual Machine (VM) to use the developed framework. The latter contains fetching input data from the LXS database, the usage of Python scripts that produce predictions, and exporting the output predictions into the LXS database.

3.5 Pilot 7 - Avoiding financial crime

The goal of Pilot 7 is to provide an easy-to-use ML platform which can support bank agents in detecting and selecting anomalous fast-loan requests. Doing this, it will help scale both the requests processed as well as to positively refine an automated processing pipeline. More specifically, building a fast loan automated system could on one side help customers develop their projects while on the other will make it safer for the bank to approve positively evaluated grants. From a technical perspective, the advanced loan approval system will make agents work faster and more effectively.

Anomaly detection associated with potential fraud attempts, and, more in general, with potentially hazardous loan issuing are the core task of this pilot. Our approach makes use of Isolation Forests [11], a consolidated machine learning algorithm for the detection of anomalies that isolate data samples by the recurrent random splitting of randomly selected features, and associate to each data point a measure of "normality" based on the number of splitting required for them to be isolated.

Moreover, PCA is used to explore the results; while SHAP (SHapley Additive exPlanations) [9] are used to show how much each feature contributes to the anomaly detection process.

In a similar way, Random Forests [12] are optimized using as a target a hand-curated version of loan request anomalies.

The algorithm is developed in Python 3.6. For data handling and feature engineering, we make use of the Pandas Python library. For the learning process, we use the Scikit-learn Python library.

First views on Performance

Algorithmic performances are tested against a synthetically generated and hand-labelled ground-truth dataset provided by CAIXA bank. Synthetic data are analysed individually by bank agents to find hazardous loan requests and potential frauds. Such operations are labelled as anomalies. The algorithm takes these labels as a binary target variable and tries to optimize the Random Forest accordingly. The same algorithm is run over a large set of features; however, performances are tested also for models with feature-subsets to make the processing more understandable and reusable for different companies. Specifically, performances are computed in terms of the model F1-score. As output the pilot will produce:

- methods/approaches, the simple data set;
- data and results analysis, code pipeline;
- performance analysis and requirements to achieve the requested performance.

Planned contributions to the INFINITECH Library and community

Within Pilot 7 we take into account several assets, which may contribute to the INFINITECH library of ML/DL algorithms and more in general to the financial community. These are:

- algorithms or models including the description of the required input as well as of the resulting output data;
- microservices that provide the models as a service. However, partners involved in the pilot need further considerations on the general practicability of such a service and the related implementation requirements before deciding on the real contribution.

3.6 Pilot 8 - Platform for Anti Money Laundering (AML) supervision

This pilot has the ambition of developing a platform that will improve the effectiveness of the existing supervisory activities in the area of Anti-Money Laundering/Combating the Financing of Terrorism (AML/CFT) by processing large quantities of data owned by the Bank of Slovenia and other competent authorities (e.g., Financial Intelligence Unit (FIU)).

More precisely, this platform is based on three components: (i) an anomaly detection and prediction component, (ii) a pattern discovery and matching component, and (iii) a StreamStory component. Below, we provide details on these three components.

Anomaly detection and prediction component

Anomaly detection is a task often addressed in the field of ML, but usually not well-defined. An anomaly refers to a previously unseen or rare uncharacteristic event. In practice, an anomaly can refer to an event that has been seen previously, but is defined as undesirable within the system (e.g., fraud, money laundering, etc.). In general, we could define an anomaly as “an outlier” data point which does not follow the collective common pattern of the majority of the data points and hence can be easily separated or

distinguished from the rest of the data [13]. In the INFINITECH project we are dealing with data from the financial domain, mainly transactions of various types and interactions between different entities. Time series can describe behaviour of various entities on different levels, for example time series of transactions of particular entities which are more or less similar, at the level of one particular financial institution or on a set of similar entities. Within a time-series an anomaly can be defined as a data point that does not follow the expected trends or seasonality characteristics. To detect anomalies, one must first define what type of anomalies s/he has in mind. The biggest challenge in anomaly detection is the definition of what we regard as an anomaly within the selected INFINITECH use cases. Therefore, we will mainly be focused on unsupervised methodologies. Thus, in order to detect and identify anomalies we will test a range of various approaches:

- clustering based approaches: K-nearest neighbours algorithm (k-NN) [14], hierarchical clustering [15];
- statistical profiling (n-sigma/percentile filter);
- concept drift detection: Drift Detection Method [16], Adaptive Windowing (ADWIN) [17];
- predictive confidence based on:
 - baseline models;
 - Isolation Forest [11];
 - ML predictive models.

The architecture of the *anomaly detection and prediction component* is straightforward: the component would anticipate feature vectors relevant for a particular use-case and would yield anomaly alerts based on those data.

Internally, the *anomaly detection and prediction component* could implement several anomaly detection approaches. However, the findings on real INFINITECH use-case data will drive the choice of the more adequate approaches. As libraries and tools used or developed, the Pilot 8 is leveraging the following ones for this component:

- Qminer (NodeJS) [18];
- Scikit-learn (Python).

Architecture

The anomaly detection program consists of three main types of components:

- consumer component;
- anomaly detection and prediction component;
- output component.

Each component has many implementations that are interchangeable, and the ones used depend on the task the program is solving. Normalization is an additional optional component which replaces anomalous samples with normalized ones. Another optional component is the visualization component which does not affect the general workflow and is used for streaming data visualization.

Configuration file

The program is configured through a configuration file specified with the `-c` flag (located in the configuration folder). It is a JSON file with the following structure.

```

{
  ...
  consumer configuration
  ...
  "anomaly_detection_alg": ["list of anomaly detection algorithms"],
  "anomaly_detection_conf": [{
    ...
    anomaly detection configuration
    ...
    "input_vector_size": ...,
    "averages": [...],
    "shifts": [...],
    "time_features": [...],
    "normalization": "normalization component", # optional
    "normalization_conf": "normalization component configuration", # optional
    "output": ["list of output components"],
    "output_conf": ["list of output components configurations"],
    "visualization": "visualization component", # optional
    "visualization_conf": "visualization component configuration" # optional
  }]
}

```

Figure 31 - Structure of the configuration file

Consumer component

Consumer components differ in where the data is read from. Anomaly detection and prediction components can read data in three different ways: from Kafka consumer, from the file consumer, and from the file Kafka consumer.

- **Kafka consumer:** Data is read from one or more Kafka topics. Each topic corresponds to a separate anomaly detection algorithm from the field "anomaly_detection_alg" which has the configuration defined in a corresponding object in the list under the "anomaly_detection_conf" field. Input format: The format of the message read from Kafka topic must be of the following shape:

```

{
  "ftr_vector": array (a feature vector) of values,
  "timestamp": timestamp of the data in datastream in unix timestamp format
}

```

Figure 32 - Format of the message read from Kafka topic.

The configuration file must specify the following parameters:

- `bootstrap_servers`: Kafka server. Example: ["localhost:9092"];
- `auto_offset_reset`: where the consumer starts to read messages. If "latest", when the consumer is started, it will continue reading from the message where it last left off. If "earliest" it will read from the oldest available message onwards. Example: "latest";
- `enable_auto_commit`: if true, the consumer's offset (the point in the topic where messages are read) will be periodically committed in the background. Otherwise, offsets can be committed manually. Example: "True";
- `group_id`: the name of the consumer group. Example "my-group";

- `value_deserializer`: any callable that takes a raw message value and returns a deserialized value. Example `"lambda x: loads(x.decode('utf-8'))`;
- `topics`: a list of topics streaming the data. Example `["anomaly_detection"]`.

Kafka consumers also have the option to filter data stored in the specified topic. To read only the messages with timestamps in a specific range, we can specify the field `"filtering"`, with the target time and tolerance. Only the messages with time of day \pm tolerance will be inserted into the algorithm. The configuration file must specify:

- `filtering`: an array of filtering parameters for each topic - either `"None"` or array of shape `"[[target_time_hour, ...minute, second], [tolerance_hour, ...minute, ...second]]"`. Example: `"[[[1, 0, 0], [0, 0, 30]]]"` - will read only data with time of day 00:59:30 - 01:00:30.

- **File consumer**: data is read from a CSV or JSON file.

Input format:

- CSV: the CSV can have a `"timestamp"` (strings in Unix timestamp format) column. All other columns are considered values for detecting anomalies.
- JSON: the JSON file must be of the following shape:

```
{
  "ftr_vector": array (a feature vector) of values,
  "timestamp": timestamp of the data in datastream as strings in unix timestamp format
}
```

Figure 33 - Shape of the JSON consumer file.

File consumer also requires a list of anomaly detection algorithms, however only the first algorithm from a list is used for anomaly detection (similar thing applies for configuration). The configuration file must specify the following parameters:

`file_name`: the name (and location) of the file with the data. Example: `"sin.csv"`.

- **File Kafka consumer**: used when the first part of the data stream is written in a file and then it continues as a Kafka stream. Also, it can be used for model-less approaches as a way of "learning" from train data, so that the anomaly detection would work better on the actual Kafka input stream.

Input format:

- CSV: the CSV can have a `"timestamp"` (strings in Unix timestamp format) column. All other columns are considered values for detecting anomalies;
- JSON: the JSON file must be of the following shape.

```
{
  "ftr_vector": array (a feature vector) of values,
  "timestamp": timestamp of the data in datastream as strings in unix timestamp format
}
```

Figure 34 - Shape of the JSON Kafka consumer file.

File Kafka consumer also requires a list of anomaly detection algorithms, however only the first algorithm from a list is used for anomaly detection (similar thing applies for configuration). The configuration file must specify the following parameters:

- `file_name`: the name of the file with the data, located in `data/consumer/directory`. Example: `"sin.csv"`.

The following parameters are similar to ones in Kafka consumer:

- `bootstrap_servers`: Kafka server. Example: `["localhost:9092"]`;
- `auto_offset_reset`. Example: `"latest"`;
- `enable_auto_commit`. Example: `"True"`;
- `group_id`. Example `"my-group"`;
- `value_deserializer`: Example `"lambda x: loads(x.decode('utf-8'))"`;
- `topics`: A list of topics streaming the data. Example `["anomaly_detection"]`.

Anomaly detection and prediction component

This component receives data from a consumer component and sends output to output components. The following arguments are general for all components:

- `input_vector_size`: an integer representing the dimensionality of the inputted vector (number of features). Example: 2;
- `use_cols`: a list of integers representing the indexes of elements in the inputted vector to be used. Example: [0, 2];
- `output`: a list of output objects. Example: `"FileOutput()"`;
- `output_conf`: a list of output objects configurations;
- `normalization`: a normalization object. This is an optional component. Example: `"LastNAverage()"`;
- `normalization_conf`: the normalization's configuration;
- `visualization`: a visualization object. This is an optional component. Example: `"GraphVisualization()"`;
- `visualization_conf`: the visualization's configuration.

Feature construction arguments:

- **averages**: it specifies additional features to be constructed. In this case averages of the last *i* values of a feature are calculated and included in the feature vector. Example: `[[2, 3, 5], [2]]` -> this means that the first feature gets additional features: average over last 2 values, average over last 3 values, and average over last 5 values and the second feature gets average over last 2 values;
- **periodic_averages**: a list (for different features) of lists (for different periods) of length 2 where the first element is the period and the second is a list of N-s (number of samples of this periodic sequence from which we will calculate average). Example: `[[[5, [2, 3]], [2, [4, 5]]]` -> in this example we have one feature with two periods (5 and 2). The first one contains 2 and 3 elements of the sequence and the second one 4 and 5;

- **shifts**: it specifies additional features to be constructed. In this case shifted values of a feature are included in the feature vector. Example: `[[1, 2, 3], [4, 5]]` -> this means that the first feature gets additional features: value shifted for 1, value shifted for 2, and value shifted for 3 and the second feature gets value shifted for 4 and value shifted for 5;
- **"time_features"**: it specifies additional features to be constructed. In this case the following features can be constructed: day of month, month of year, weekday, hour of day. Note that the construction of these features requires a Unix timestamp format. If that is not the case it passes an empty array as a parameter for that field. Example: `["day", "month", "weekday", "hour"]`.

Functionalities:

- **Border check**: a simple component that checks if the `fr_vector` falls within the specified interval and also gives warnings if it is close to the border. It requires the following arguments in the configuration file:
 - `warning_stages`: A list of floats from interval `[0, 1]` which represent different stages of warnings (values above 1 are over the border). Example: `[0.7, 0.9]`;
 - `UL`: Upper limit of the specified interval. Example: 4;
 - `LL`: Lower limit of the specified interval. Example: 2.
- **EMA**: calculates the exponential moving average of test values. It receives data from a consumer component and sends output to output components. EMA is calculated using past EMA values and the newest test value, giving more weight to newer data. It is calculated in the following way: $EMA_{latest} = fr_vector \times smoothing + EMA_{last} \times (1 - smoothing)$. Warnings are issued if the EMA approaches UL or LL. It requires the following arguments in the configuration file:
 - `N`: parameter from which the smoothing is calculated - roughly translates to how many latest test values contribute to the EMA. Example: 5;
 - `warning_stages`: similar to border check - levels to identify EMA approaching limits. Example: `[0.7, 0.9]`.
- **Isolation Forest**: iForest algorithm. The basic principle is that an anomalous data point is easier to separate from others, than a normal datapoint. In the current implementation, one instance consists of N consecutive (or non-consecutive) test values. Instances are constructed via the "shifts" module. The algorithm can also be modified by adding other features. A pre-trained model can be loaded from the "models" folder, or a new model can be trained with the appropriate train data. Arguments in configuration file when a user is training a new model:
 - `train_data`: location of the train data. Example: `"data/Continental/0.txt"`. Data should be in CSV format, as it is described in Training files. This is an optional parameter. If it is not specified `load_model_from` is expected;
 - `max_features`: the number of features to draw from X to train each base estimator. Example: 4;
 - `max_samples`: the number of samples to draw from X to train each base estimator. Example: 100;
 - `contamination`: the proportion of outliers in the dataset. Example: 0.1 if we know we have 10% of outliers in our dataset. It can be set to "auto";
 - `model_name`: name of the model, which will be saved in "models/" folder;
 - `retrain_interval`: an integer representing the number of samples received by the anomaly detection component that triggers model retraining. If this optional parameter is not present, the model is not retrained. Every time the model is retrained the training set is saved to `IsolationForest_last_{samples_for_retrain}_samples.csv` file in data folder and configuration file is changed so that the next time the model is ran it will train on this dataset (e.g., in case of crash and rerun). Example: 100;
 - `samples_for_retrain`: an integer representing the number of most recent samples that are used to retrain the model. If it is not specified, it uses all samples (it may cause memory

- overflow). If a training dataset is specified those samples are also considered for retraining until they are overwritten by newer samples. Example: 2000;
- `retrain_file`: path and file name of the file in which retrain data will be stored (example: `./data/retrain/test.csv`). Example of model train configuration: `IsolationForestTrain.json`. Model train mode is activated if `"load_model_from"` is not in the configuration file, and `"train_data"` is defined. After training, the trained model will be used to continue with evaluation of data from consumers automatically. If we have a pre-trained model, we load it by specifying `load_model_from`;
 - `load_model_from`: location of a pre-trained iForest model. This is an optional parameter. If it is not provided, `train_data` must be and vice versa. If neither of them are provided, the retrain interval must be and the model will output undefined results until the first retrain. Example: `"models/IsolationForest"`.
- **PCA + Isolation forest**: PCA projects high dimensional data to a lower dimensional space. This is a very effective first preprocessing step, if we have a large number of features in an instance (even > 1000). First, PCA is applied to the input data followed by the Isolation Forest algorithm [11]. In addition to the Isolation Forest requirements, the following parameters must be specified in the config file:
 - `N_components`: dimensionality of the PCA output space (example: 5).
 - **Welford's algorithm**: very similar to border check and only in this case UL and LL are calculated with every new sample (with equations: $UL = \text{online_mean_consumption} + X \cdot \text{online_standard_deviation}$ and $LL = \text{online_mean_consumption} - X \cdot \text{online_standard_deviation}$). There are two modes for this method. The first one calculates mean and standard deviation of the previous N samples and the second one calculates them for all the data samples so far. It requires the following arguments in the configuration file:
 - `N`: an integer representing the number of samples used for calculating mean and standard deviation. If it is not specified all samples till some point will be used in calculation. Example: 5;
 - `warning_stages`: a list of floats from interval [0, 1] which represent different stages of warnings (values above 1 are over the border). Example: [0.7, 0.9];
 - `X`: a parameter in the equation for calculating LL and UL. The larger it is, the bigger the interval of normal values is (more false negatives). Example: 1.5;
 - **Filtering**: digital filtering is applied to the input data via a lowpass filter, smoothing the signal. The filter used is Butterworth filter of the order specified in the configuration file. This helps to identify trend changes in the data. Similarly, to Border check and EMA, warnings are issued if the filtered signal approaches UL or LL. The filtered signal helps to identify trend shifts. Mode 0: taking the difference between the actual value of each new data point from the current filtered signal value can help to identify sudden spikes. Mode 1: in mode 1 a normalised value is calculated: $\text{value} = \text{last change in value} / (UL - LL)$. The normalised value is classified according to warning stages or marked as an error if the value is >1 or <-1. It requires the following arguments in the configuration file:
 - `UL`: Upper limit of the specified interval. Example: 4;
 - `LL`: Lower limit of the specified interval. Example: 2;
 - `warning_stages`: similar to e.g. EMA. Example: [0.7, 0.9];
 - `filter_order`: order of the filter - how many latest points are used in the filter response. Most of the time, somewhere around order 3 is enough, as a higher order filter will cause a delay in the filtered signal compared to the real-time data. Example: 3;
 - `cutoff_frequency`: the frequency at which the filter response starts to drop off. Lower frequency components of the input data will be passed through, but higher frequency components are filtered out. This is not an actual physical frequency but rather a float between 0 and 1. Example: 0.1;
 - `mode`: either 0 or 1; 0 - output is the filtered signal, 1 - output is the difference between the test value and the filtered signal (this is for visualization purposes).

- **Hampel:** the Hampel filter is used to both detect outliers and replace them with "normal" values. The filter analyses a window of data including data points before and after the data point in question. The median value of these points is calculated, as well as the standard deviation. If the point in question is more than n_sigmas times the standard deviation away from the median it is marked as an outlier and replaced with the median value. Keep in mind that this algorithm can only analyse each data point, when there are W points available after that point. The algorithm thus works with a delay of W points. It requires the following arguments in the configuration file:
 - n_sigmas : tolerance of the algorithm. Example: 3;
 - W : window size - in the calculations W points before and W after the point in question are used. Example: 5;
 - K : a multiplier which is dependent on the distribution of input data. For Gaussian it is 1.4826.
- **GAN:** this model is built on the autoencoder principle. N dimensional data is given to the model which has a bottleneck of significantly fewer dimensions. The model is trained on "good" data, where it learns to reproduce the inputs very precisely. When the trained model is given an anomalous feature vector, it cannot reproduce it exactly. The measure of how anomalous the datapoint is, it's the reconstruction error, which is a mean squared error comparing the input and output feature vectors. Based on the reconstruction errors on the training set, a threshold is determined. When the model detects a high reconstruction error, the datapoint is labeled as an outlier. It requires the following arguments in the configuration file:
 - N_shifts : number of past data, which are used to construct the feature vector. Example: 9;
 - N_latent : dimensionality of the latent space. Example: 3;
 - K : coefficient to determine the threshold for the reconstruction error from max_err - largest error on the training set \rightarrow threshold = $K * max_err$. Example: 0.95;
 - $retrain_interval$: an integer representing the number of samples received by the anomaly detection component that triggers model retraining. If this optional parameter is not present, the model is not retrained. Every time the model is retrained the training set is saved to `IsolationForest_last_{samples_for_retrain}_samples.csv` file in data folder and configuration file is changed so that the next time the model is ran it will train on this dataset (e.g., in case of crash and rerun). Example: 100;
 - $samples_for_retrain$: an integer representing the number of most recent samples that are used to retrain the model. If it is not specified, it uses all samples (it may cause memory overflow). If a training dataset is specified, those samples are also considered for retraining until they are overwritten by newer samples. Example: 2000;
 - $retrain_file$: path and file name of the file in which retrain data will be stored. Example: `./data/retrain/test.csv`.
- **Clustering:** this model uses DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [19] clustering method (from Scikit-learn) to extract and save core samples. Every incoming sample is then compared to those core samples. If the Euclidean distance of the sample to every core sample is greater than the threshold then the sample is anomalous. It requires the following arguments in the configuration file:
 - eps : a float representing the maximum distance between two samples needed for one to be considered as in the neighbourhood of the other;
 - $min_samples$: an integer representing the number of samples (or total weight) in a neighbourhood for a point to be considered as a core point. This includes the point itself;
 - $threshold$: a float, representing the Euclidean distance that is required to determine anomalies (the larger the distance the less points are anomalous);
 - $retrain_interval$: an integer representing the number of samples received by the anomaly detection component that triggers model retraining. If this optional parameter is not present, the model is not retrained. Every time the model is retrained the training set is saved to `IsolationForest_last_{samples_for_retrain}_samples.csv` file in data folder and

configuration file is changed so that the next time the model is ran it will train on this dataset (e.g., in case of crash and rerun). Example: 100;

- `samples_for_retrain`: an integer representing the number of most recent samples that are used to retrain the model. If it is not specified, it uses all samples (it may cause memory overflow). If a training dataset is specified those samples are also considered for retraining until they are overwritten by newer samples. Example: 2000;
- `retrain_file`: path and file name of the file in which retrain data will be stored. Example: `"/data/retrain/test.csv"`.
- **Combination**: not really a stand-alone algorithm but rather a combination of arbitrary algorithms with the idea to obtain a single estimate about anomalousness of the sample. From each algorithm's estimate a final status is determined using a specific logic (that can be completely case-specific). It requires the following arguments in the configuration file:
 - `anomaly_algorithms`: a list of the anomaly detection algorithms used;
 - `anomaly_algorithms_configurations`: a list of the anomaly detection algorithms' configurations. These configurations have the same fields as stand-alone anomaly detection algorithms would have;
 - the output `status_determiner`: an object containing a logic to determine the final status. Additional objects for that purpose can be implemented.

The following status determiner methods are currently available:

- **AND()**: gives error if all statuses are error, warning if all statuses are warning (or error), otherwise OK. Undefined statuses are ignored;
- **OR()**: gives error if at least one status is error, warning if at least one status is warning, otherwise OK. Undefined statuses are ignored.

Training files

Training files for the algorithms that need training (GAN, Isolation Forest and PCA) must be provided in CSV format with two specified columns: `timestamp` and `ftv_vector`. The first field contains a timestamp (in Unix timestamp format) of the sample and the second one contains a string, representing the feature vector (of the same format as the one that would be received from the consumer). Same holds for retrain files (generated automatically by the algorithm).

Output component

Output component differs in where the data is outputted to. It is worth noting that more than one output component can be specified. It receives three arguments from the anomaly detection component: value (the last value of the stream), timestamp and status (whether data is anomalous). The following arguments of the configuration file are general for all outputs:

- `send_ok`: a boolean telling the component if OK samples are to be sent to output. This is an optional parameter. Default value is true;
- **Terminal output**: timestamp, value and status are outputted to the terminal. It does not require any parameter in the configuration file;
- **Kafka output**: value is outputted to separate Kafka topics.

Output format: the outputted object is of the following form:

```
{
  "algorithm": anomaly detection algorithm used,
  "value": the sample from the datastream,
  "status": result of the anomaly detection algorithm,
  "timestamp": timestamp of the data in datastream in unix timestamp format,
  "status_code": result of the anomaly detection algorithm (descriptive),
  "suggested_value": optional field, that can suggest a "normal" value if anomaly is detected
}
```

Figure 35 - The format of the outputted object.

Status codes are defined in the following way: OK: 1, warning: 0, error: -1, undefined: 2. It requires the following arguments in the configuration file: `node_id`: The anomalies will be sent to topic: `anomalies_[node_id]`. Example: 1.

- **File output:** data is outputted to a JSON CSV or txt file.

Output format:

- **CSV: the CSV file contains the following fields:**
 - "algorithm": anomaly detection algorithm used;
 - "value": the sample from the data stream;
 - "status": the result of the anomaly detection algorithm;
 - "timestamp": timestamp of the data in data stream in Unix timestamp format;
 - "status_code": the result of the anomaly detection algorithm (descriptive);
 - "suggested_value": optional field that can suggest a "normal" value if anomaly is detected.
- **JSON:** the JSON file contains a single field "data" whose value is an array of objects of the following shape:

```
{
  "algorithm": anomaly detection algorithm used,
  "value": the sample from the datastream,
  "status": result of the anomaly detection algorithm,
  "timestamp": timestamp of the data in datastream in unix timestamp format,
  "status_code": result of the anomaly detection algorithm (descriptive),
  "suggested_value": optional field, that can suggest a "normal" value if anomaly is detected
}
```

Figure 36 - The shape of the JSON file containing a single field data whose value is an array of objects.

The output in the txt file is the same as the terminal output. Status codes are defined in the following way: OK: 1, warning: 0, error: -1, undefined: 2. It requires the following arguments in the configuration file:

- `file_name`: the name of the file for the output located in the log/directory. Example: "output.csv";
- `mode`: whether we want to overwrite the file or just append data to it. Example: "a".

Pattern discovery and matching component:

Pattern analysis provides two main functionalities: (i) *pattern matching*, and (ii) *discovery*. The component provides support for detection of complex patterns in a graph data representation. A pattern is defined as a sequence of events with a given length or specific graph paths or sequences. In pattern matching analysis we focus on detecting relevant (already identified) patterns. In this analysis, specifically in the financial domain, there are two main challenges: either we need to reduce the data points from the initial time series or the data patterns are too sparse, hence we have the opposite problem. Usually, appropriate data preparation needs to be performed in the data preparation phase, but in case of sparse data, there is no easy solution. During the development of the pattern discovery and matching component, Pilot 8 will research and develop novel approaches to detect rare patterns in sparse data space. As libraries and tools, the pilot will leverage the following ones for this component:

- Qminer (NodeJS) [18];
- SNAP (Stanford Network Analysis Project)¹⁴.

The current version of the pattern discovery tool relies on a specific scenario based on pattern detection in large scale network transactions. It is currently able to detect and visualise specific scenarios described by domain experts and combine them as parameterized queries. The results are presented as transaction networks induced by accounts and connections produced by possible suspicious agents. The networks can be further analysed and explored manually. The tool also provides simple combinations of queries and explorations, thus further facilitating pattern detection across different scenarios. Current versions have successfully detected specific suspicious patterns in historical data and uncovered existing transaction networks of already known suspicious actors. Qualitative analysis has shown a much easier data exploration pipeline, faster visualization and easier interpretation of specific transaction typologies detected and matched by the parameterized queries and manual exploration.

Detected patterns can be exported and further analysed with standard ML methods. The screening tool component will provide an ML framework and docker images enabling pattern detection and matching through an API providing full transaction network data and enriched data. Endpoints will be provided to query the transaction networks corresponding to specific scenarios and detected patterns for analysis, visualization and reporting.

StreamStory Component:

StreamStory [20] is a component for the analysis of multivariate time series on multiple scales. It computes and visualizes a *hierarchical hidden Markov model* (HMMM) [21] which captures the qualitative behaviour of the systems' dynamics, where the system is described with a group of timeseries. StreamStory is mostly used for multivariate data visualization, time series visualization, cluster visualization and multi-scale visualization. It enables analyses on the intersection of two or more data categories. In each category, we focus on techniques able to visualize large datasets. Despite each of these categories offering a wide array of tools and techniques, the visualization of large, multivariate time series is still an understudied area. Let's assume that one system is described with a series of time-series data. StreamStory enables analysts to visualize and compute regularities between different time series and therefore enable analysts to create new knowledge about the system's behaviour.

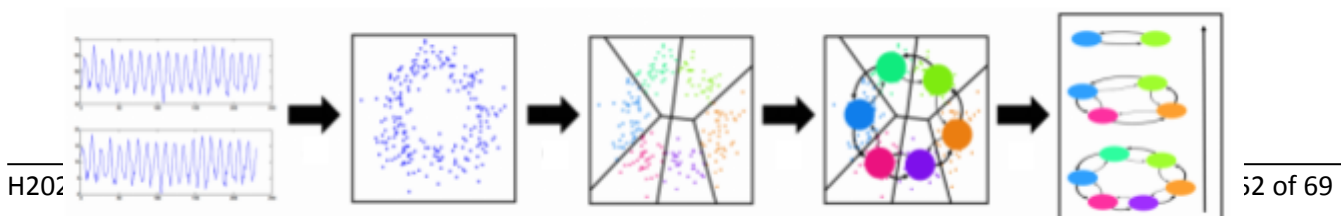


Figure 37: StreamStory pipeline.

Figure 37 depicts the StreamStory pipeline:

- first of all, the multivariate time series is represented as a point cloud. We show two noisy approximately periodic signals, mapping to 2D space;
- the space is then partitioned and the typical states of the system are identified using a clustering algorithm. System states correspond to intervals of the phase of the signal and represent partitions;
- each partition is then translated into a Markov chain model;
- the Markov chain model is simplified by aggregating states, giving a multi-scale view of the model.

In this manner a novel abstraction for multivariate time series is presented and it is based on multi-scale continuous time Markov chains extracted from data using ML techniques. The data is shown in a qualitative manner - via a hierarchy of directed graphs - allowing users to find suitable scales to interpret the data. The system requires minimal user input and, in addition to automatically constructing the representation, it provides numerous auxiliary tools to aid the interpretation. As libraries and tools, the pilot will leverage the following ones for this component:

- StreamStory [20];
- Qminer (NodeJS) [18].

The StreamStory component is currently in the process of code refactoring. It will be used as a standalone component, which could be used especially in the process of data discovery and problem formulation as well as a valuable explanation tool.

3.7 Pilot 9 - Analyzing Blockchain transaction graphs for fraudulent activities

This pilot aims at developing and deploying a scalable and high-performance system dedicated to the analysis of the Blockchain transaction graph. In particular, the system investigates whether customers' Blockchain account transactions can be traced to fraudulent activities or accounts. Indeed, Blockchain cryptocurrencies and tokenized assets, that are obtained fraudulently, can go through various transfers on the Blockchain and end up as stable coins (e.g., USD, EUR, TRY tokens) in different jurisdictions. As a result, a company accepting stable coins may be paid by using coins that can be traced to addresses involved in fraudulent activities. Holding stable coins that originated from fraudulent or sanctioned addresses, can be risky for the company. Hence, the construction of a massive Blockchain transaction graph and its analysis are necessary activities to detect frauds. Recently, the Financial Action Task Force (FATF) has also published a guidance [31] that states *“Virtual Asset Service Providers (VASPs) be regulated for anti-money laundering and combating the financing of terrorism (AML/CFT) purposes, licensed or registered, and subject to effective systems for monitoring or supervision.”*

To carry out the analyses, Pilot 9 is (i) preparing massive Bitcoin and Ethereum Blockchain transaction data as well as a database of blacklisted addresses, (ii) developing graph algorithms (both parallel and sequential) which can be used to extract graph-based features and perform analyses, and (iii) using ML algorithms for analysis. In the first period of the INFINITECH project, Pilot 9 focused on parts (i) and (ii). In the second period, we have started part (iii) and thus we are focusing on ML algorithms. The ML algorithms under investigation are parallel graph algorithms that were implemented and tested on the 625M Bitcoin and 766M Ethereum transaction datasets. More specifically, we are focusing on the following algorithms, metrics and analyses: (i) transaction graph construction, (ii) PageRank on transaction graph, (iii) degree distributions of transaction graph nodes, (iv) number of transfer transactions of 40 major ERC20 tokens on Ethereum, (v) connected component, (vi) Shortest Path based on blacklisted node trace forest, (vii)

extracting node features, and (viii) tracing subgraph of blacklisted addresses. Our recent progress on the initiated ML/DL algorithms are described below.

Task definition

Given a list of blockchain addresses, a list of blacklisted addresses and transfer transactions, we want to classify whether the addresses are fraudulent or not. Since we want to predict fraudulent addresses, the problem is posed as a supervised ML problem.

Developed and tested ML/DL algorithms

We investigated the scientific literature related to features that can be used in our study. These papers include “Anomaly detection in the Ethereum network” [22], “Detection of illicit accounts over the Ethereum blockchain” [23], “Detecting malicious accounts on the Ethereum Blockchain with supervised learning” [24], “Detecting malicious Ethereum entities via application of machine learning classification” [25].

Then, we have used the Scikit-learn and Imbalanced-learn libraries for our ML task. We extracted the following features from the transactions for each address: (i) number of sent/received transactions, (ii) number of unique addresses from which the address sent or received the transaction, (iii) total amount sent/received, (iv) average amount sent/received, (v) net account balance, (vi) first/last transaction timestamp, (vii) time difference between first and last transaction, (viii) last transaction direction, (ix) last transaction amount, (x) min/max amount sent/received, (xi) number of failed sent/received transactions, and (xii) average time difference between sent/received transactions. Since the number of blacklist addresses is small compared to all addresses, we used oversampling techniques.

Initial performances obtained

We have obtained initial results on a small dataset of Ethereum transactions having 14544 addresses (from blocks numbered 10.000.000-10.100.000) and 144 blacklisted addresses (scraped from the Internet). The initial performances obtained are given in Table 5.

Table 5- Initial performances obtained on a small dataset of Ethereum transactions having 14544 addresses.

	Decision Tree	Random Forest	Gradient Boosting	K-nearest Neighbour	Support Vector Machine	MLP	Gaussian Naive Bayes
Accuracy	0.941	0.918	0.956	0.880	0.857	0.918	0.956
Precision	0.520	0.520	0.533	0.512	0.510	0.523	0.533
Recall	0.589	0.650	0.621	0.627	0.626	0.669	0.621
F1	0.523	0.521	0.546	0.497	0.487	0.525	0.546

We plan to expand our dataset as well as our feature set in the future. Note that we have collected a larger size of Ethereum/Bitcoin blacklist addresses, namely 5846 Ethereum and 21060 Bitcoin addresses. However, they span a very large range of blocks which in return includes a massive number of addresses. This

introduces large computational requirements. Therefore, we applied our ML algorithms on smaller ranges of blocks in our initial tests.

Our initial ML analysis used features that have been utilized in the literature. These features are data local to addresses and can be computed or extracted easily. Since we have developed a scalable distributed graph system, we are able to compute features that require global computation. For example, our PageRank and connected components algorithms were implemented for this purpose. Recently, we have also developed a convex subgraph finding algorithm for Blockchain transaction graphs. A “peel chain” activity [26] is a transaction pattern where funds are moved on a directed path of many addresses in order to conceal traces to fraud related addresses. We generalized the directed path to a general convex subgraph and contributed an integer programming-based algorithm for it [27]. In the future, we plan to use the presence of convex subgraph information as well as the PageRank information as features in our ML algorithms.

Type of contribution to provide to the INFINITECH library of ML/DL algorithms

Currently, we are able to contribute the following:

- Ethereum blockchain transaction data available at the Zenodo link: <https://zenodo.org/record/4718440>,
- Computed feature data such as the Blockchain address PageRank that can be used in ML algorithms;
- Convex transaction subgraph extraction program. Presence of addresses on convex subgraphs can be used as a feature in ML algorithms. This feature can potentially be made use of in Blockchain transaction analysis as well as in non-Blockchain addresses such as IBAN number-based bank accounts.

Finally, all of our input/output is file based. Transactions are read from the files whose format is available at <https://zenodo.org/record/4718440>. Features are calculated and saved in files. These features are then input to the ML algorithms. The output classification result is saved to a file.

3.8 Pilot 10 - Real-time cyber-security analytics on financial transactions' Big Data

This pilot aims at significantly improving the detection rate of malicious events (i.e., fraud attempts) and enabling the identification of security-related anomalies while they are occurring by the analysis in real-time of the financial transactions of a home and mobile banking system. This approach thus allows proactive and prompt interventions on potential security threats.

More specifically, Pilot 10 is developing a tool based on ML techniques applied to real-time, financial transaction data-streams focused on adaptive detection for malicious transactions leveraging on established big-data analytics' practices. The analysis of vast amounts of data will help to define relevant cyber-risk rating metrics and allow us to implement adaptive security measures and controls, based on real cyber-security postures.

The final dataset will be a generated, realistic dataset that is consistent with the real data present in the data operations' environment. The *synthetic* dataset will be created by Poste Italiane starting from randomly generated personal data with no correlation with real people.

The use case involves a lambda architecture approach, consisting of a batch and streaming workflow. Clustering techniques such as K-means will first be used to group the different data points. Moreover, the data, before being processed by an unsupervised algorithm, will be properly prepared. Data preparation includes: (i) transformation of characteristics from categorical to numerical, (ii) data normalization, and (iii)

standardization. A study of the identified clusters will then be carried out by Poste Italiane through data visualization and exploration tools. This step will include representation of data on different charts and tables containing statistical information (e.g., count, mean, standard deviations, etc.) of data grouped by cluster.

Once the dataset has been labelled, a supervised algorithm such as Random Forest (Breiman, L., 2001) will be used to train a ML model which will then be used in a real-time context to label new incoming data. Depending on the result label, it will be possible to identify whether a given transaction is identifiable to a possible fraudulent action or not. The batch workflow can be scheduled in order to produce an updated ML model based on the data collected, thus updating the model used for the streaming workflow.

The clustering step is used as input for a classification algorithm that can apply all the classification metrics such as Accuracy, Precision, Recall, F1-score, etc. In this way, the performance of the supervised method can then serve as a surrogate for the performance of the unsupervised learner.

For this pilot, the Apache Spark¹⁵ framework will be used to process large amounts of data. Apache Spark provides a set of supervised and unsupervised algorithms across its scalable ML library and support for data preparation. Other libraries could be considered, such as Scikit-learn to perform data analysis and machine learning operations, and TensorFlow/Keras to implement artificial neural networks.

Spark supports different programming languages. PySpark, the Python API written in Python to support Apache Spark, will be used for the implementation of use case tasks.

Progress of the machine learning tasks

Concerning the batch layer:

- For the data preparation stage's goals two services were made available: (i) OneHot Encoder and (ii) StringIndexer;
- K-means service was used for the prepared data. However, currently studies are underway to improve clustering by following a hybrid approach that also takes into account bank customers' profiling;
- the Random Forest model training was already executed by considering the overall labelled data set. In this case, a possible improvement will occur in case a silhouette analysis will be applied to the dataset before the training in order to enhance the accuracy of the clustering approach;
- finally, data augmentation techniques are under investigation to balance the size of the different clusters obtained (fraudulent versus trusted ones).

Within the speed layer, a streaming Random Forest service was implemented.

Actual performance evaluations (in terms of Accuracy, Precision, Recall, regression metrics) were planned but they were not executed yet.

Type of contribution to provide to the INFINITECH library of ML/DL algorithms

Within Pilot 10 each ML service adopted is not a standalone service but it is closely related to the lifecycle of an ALIDA workflow. ALIDA is a micro-service based platform for composition, deployment, optimisation, execution and monitoring of pipelines of Big Data Analytics (BDA) services (see for more details deliverable D5.7). ALIDA is a result of previous research activities developed by ENG. Currently, it is a work in progress. ALIDA offers a catalogue of BDA services (ingestion, preparation, analysis, visualization): users design their own (stream/batch) pipeline by choosing the BDA services from it, indicate which Big Data set they want to process, launch and monitor the execution of the pipeline and personalize the results' visualization by choosing from a set of available graphs, all this without worrying about having software developer's skills or particular knowledge on big data technologies. Thus, each ML service is registered in the ALIDA catalogue as

¹⁵ <https://spark.apache.org/>

a containerized Docker application including the Python code and its dependencies. After implementing the algorithm using PySpark, creating the Dockerfile and pushing the new docker image inside a repository, this microservice is registered into the ALIDA catalogue through the GUI. See for more details: <https://home.alidalab.it/>.

Downstream of the execution of the entire bundle, the real-time display of suspicious transactions will take place using a tool that is under development within WP4. This tool, developed by ENG, will be part of the INFINITECH marketplace.

3.9 Pilot 11 - Personalized insurance products based on IoT connected vehicles

From the AI perspective, and as described in D7.12, the core of the pilot's expected outcomes relies on the development of a ML powered model to analyse the drivers' profiles and classify and identify drivers according to the way they drive within a given context. This way, Pilot 11 pursues a driving profiling tool that exploits connected vehicles' data and allows car insurance companies to customise the offered solutions to selected clients according to a more realistic model of their associated risk. The architecture and components that gather and homogenise vehicles' and context data is presented in D7.12. In terms of ML/DL technologies, the pilot develops 2 services:

- **Fraud detection:** intended to help on the proper identification of the actual driver involved in a traffic incident, the service develops a *Driver Profile* model that links with the client's insurance policy and assigns a specific profile to each authorised driver. By analysing the vehicle's technical data prior to the incident, it can infer which was the actual profile driving the car.
- **Pay as you Drive:** it processes the connected vehicles' technical data and correlates this with available context information (weather conditions, traffic alerts and roads information) to classify driver's performance. The obtained driver profile (from technical data) and classification (from context information) will support the prices and bonuses offered by the company to the insured client.

As already described in D5.7, two different machine learning approaches will be used to develop these services: i) the first, for the *Fraud Detection* service, will consist of a clustering algorithm, which allows differentiating multiple routes (a route is understood as all the technical data from a connected vehicle captured in a given time interval) taken by the same car and to infer if a different driver is using the vehicle. Techniques such as K-Means and Mean-Shift will be used to group different data points. ii) *Pay as you Drive*, will be based on classification algorithms. Models such as K-Nearest Neighbours or Support Vector Machine (SVM) are explored, allowing for drivers to be classified in an efficient and powerful way.

The data used for these purposes will mainly come from connected vehicles provided by CTAG (Galician Automotive Technology Centre)¹⁶. These vehicles can provide real-time data while being on-road, such as the current speed, the steering angle or the instant acceleration. Additional to this data, weather information from the city of Vigo and its surroundings will be collected as well, in order to match the weather conditions to the driving style. CTAG will also be able to provide alerts info from the Vigo surroundings, in order to infer driving incidences and consider them for the driver's profile. To collect great amounts of data for analysis and training, the SUMO (Simulation of Urban MObility) simulation package [28] will be used to create on-demand simulated routes using a predefined set of scenarios.

Table 6 - Data types.

¹⁶ <https://ctag.com/en/>

Data Type	Description	Table name	Provider	Nº of Registers	Volume of Data	Data rate
Vehicle	Technical data from connected vehicles	etvehicle	CTAG	13,5M	3GB	80k/day (reg) 18,2 MB/day
Alert	Traffic alerts	etalert	CTAG	860.453	125 MB	5,2k/day (reg) 0,75MB/day
Weather Observed	Weather information	etweather observed	AEMET (Atos)	8.110	2,1 MB	72/day (reg) 16kb/day

For this pilot, various AI and data science libraries will be used.

- **Scikit-learn:** This library gives developers the ability to perform data analysis and simple Machine Learning operations.
- **NumPy:** Provides functions to work on arrays and perform computations on them.
- **Pandas:** Allows performing data manipulations on the datasets.
- **TensorFlow/Keras:** High-level Artificial Intelligence library, provides methods to easily implement, train and serve Machine Learning models.
- **EASIER.AI:** This pilot will use the EASIER.AI platform to train and orchestrate its services.

3.10 Pilot 12 - Real-world data for novel health insurance products

The objective of this pilot is to demonstrate how real-world data can be utilized by insurance companies to develop novel products for the benefit of the insurer and of the customers. To achieve this goal, Healthentia, an e-Clinical platform developed by Innovation Sprint, is used as a real-world data capturing infrastructure to automatically collect measurements from IoT devices (activity trackers) and users' reports.

The collected data is utilised in two ways by the pilot. On the one hand, the data is ingested into the pilot's testbed via the INFINITECH's *Data Collection component* developed by UBI (see deliverable D5.13 for additional details), where it is anonymised by the anonymizer, stored in the LeanXcale database and used for offline model training. The process is manually initialised by a ML engineer, and it is orchestrated by the Data Protection Orchestrator. Trained models are exported back to Healthentia.

On the other hand, the data keeps flowing through Healthentia, where an online decision system offers the following two services to its end-users, namely the health insurance companies:

- **Risk assessment.** The lifestyle of each client is scored, allowing the end-users to offer personalised pricing for their insurance plan.
- **Fraud detection.** The decisions are analysed to extract feedback from the clients, persuading them to truthfully use the system. Also, the behaviour of clients is analysed to detect fraudulent usage of the activity trackers or untruthful replies to questionnaires.

For the pilot, different data sources will be used:

- Longitudinal data coming from the users of the Healthentia app (~750 kB/user/week);
- Synthetic longitudinal data from the Innovation Sprint's RWD (Rear Wheel Drive) Simulator (~750 kB/user/week);
- Static insurance company data (minor volume of quasi-static data).

Machine Learning (ML) approaches

The methodology for providing the risk assessment service is straightforward. Classifiers will be trained to yield if the lifestyle of the client is improving or worsening. Since the pilot does not expect an abundance of data, Random Forest classifiers will most probably be used. On the other hand, the simulated data will allow training of Neural Network classifiers as well. Should the RWD Simulator data be found to be realistic enough, it might also be used for training the final classifier.

The classifier decisions are accumulated across time to yield the risk assessment. They are also analysed using SHAP (SHapley Additive exPlanations) [9] to yield the most important factors trying to drive the decisions one way or another. These factors form the basis for feedback towards the clients in some virtual coaching context. The process of data collection, classifier training and decision analysis is detailed in Pnevmatikakis et al. [29].

Fraud detection will be based on the clustering of behaviours for outlier detection. Both traditional approaches will be investigated (K-means and Gaussian Mixture Models), but also deep learning ones using autoencoders.

The ML/DL libraries to be used to offer this functionality are Scikit-learn and TensorFlow/Keras. The numerical processing libraries are NumPy and Pandas.

3.11 Pilot 13 - Alternative/automated insurance risk selection - product recommendation for Small and Medium Enterprises (SMEs)

The main objective of Pilot 13 is to implement a data analysis platform applying ML technologies to better predict the insurance needs of Small and Medium Enterprises (SMEs). In this context, the platform will generate a risk map of the SMEs based on their daily activities and will predict how the risk will vary over time. Therefore, the pilot will design and implement a service that effectively monitors the current risks of SMEs, as well as their risk variance in the future, in order to improve the control of the accident rate, the renewal of insurance policies and offer personalised insurance coverage.

To this end, real data will be leveraged that will be obtained from a variety of sources such as the web, social media profiles, official registers, opinion platforms, business directories, e-commerce platforms, and more. The dataset will be composed of data originating from 50000 European SMEs (i.e., structured and unstructured data in image format and text format). The estimated data volume is expected to be around 1 TB of data.

The innovation in the project is threefold: (i) the use of ML techniques to predict the needs of SMEs concerning their insurance, (ii) the use of alternative data, traditionally not considered by insurance companies, which comes from open sources, and finally (iii) data with periodic extractions and not only at the time of contracting the product, as it is currently the case

As already mentioned, the use of the data is one of the differentiating factors since the pilot will leverage data from open and non-traditional online sources such as a dataset on SMEs geolocation information and characteristics, a dataset on reviews and opinions about SMEs, etc. Doing this, the pilot owners manage to accelerate processes, improve efficiency and be able to offer products to the SME clients of the insurance companies according to their risk profile without any previous interaction with the company.

To calculate metrics such as VaR [2] and to conduct pre-trade analysis, the pilot will explore two different approaches. More precisely, the output data will be classified into two groups:

- *Direct, raw data*, whose utility is basic for data cleaning and autocompletion in the subscription processes and in the automation of the same. In this type of data techniques semantic analysis, tagging and search for coincidences, and chain similarity algorithms will be applied;
- *Processed data* based on algorithms designed using ML to determine risk levels, propensity to purchase a given insurance, and insurance package recommendations.

To this end, the ML techniques planned to be used and already used by Wenalyze are the following ones.

- **Supervised learning: Linear regression**, namely a statistical technique used to study the relationship between variables. A small percentage of the data is extracted for subsequent verification of the results, to which the relationship obtained will be applied.

- **Unsupervised learning: Clustering** or grouping objects by similarity, in groups or sets so that members of the same group have similar characteristics.

- **Reinforcement Learning**: This is a subfield of ML that teaches an agent how to choose an action from its space of action, within a particular environment, to maximise rewards over time.

It is worth noticing that the exact ML algorithms that will be explored are still under investigation, as the initial pilot prototype leverages a pure statistical/scientific computing approach.

For this pilot, various AI and data science modules and libraries integrated will be used.

- **Auto Pilot**: It automatically creates, trains and adjusts the best ML models based on available data, while allowing the user to maintain complete control and visibility.
- **Data Wrangler**: To prepare and clean data for machine learning.
- **AWS Pipeline**: AWS Data Pipeline helps a user move, integrate, and process data across AWS compute and storage resources, as well as on-premises resources.
- **AWS Sagemaker**: High-level Artificial Intelligence library, that provides methods to easily implement, train and serve ML models.

The algorithms that are being developed are (i) Cyber Risk, (ii) Reputational Risk Level, (iii) Management List, (iv) Family Business, (v) Business Continuity Rating, (vi) Directors and Officers insurance needs, and (vii) General Liability insurance needs.

3.12 Pilot 14 - Big Data and IoT for the agricultural insurance industry

The objective of Pilot 14 is to define, structure and test specific services for the agricultural insurance sector in order to better protect agricultural assets by evaluating risks in a data-driven way and to improve the business process of agricultural insurers and clients (i.e., farmers). The services tested will be: (i) a mapping of risks related to agriculture in predefined markets, (ii) the prediction and assessment of weather and climate risk probability, and (iii) a damage assessment calculator for insurance companies.

Progress of the ML tasks

In particular, Pilot 14 aims at identifying areas where crop productivity and catastrophe probability is high based on intelligent risk mapping, at creating additional datasets with high predictive value for improving underwriting of agricultural risks concerning weather and climate risk probability, and finally at improving the damage assessment and claims handling procedures for the insurance industry to increase the efficiency of calculating indemnity pay-outs.

Specifically, the formulation of risk metrics and damage calculation are developed as two different services. One is focused on drought and the consequent prediction of the afflicted crop production (barley, wheat, and cereals), while the second is aimed at classifying the damage caused by hailstorms on wheat, maize, and soybean production. Consequently, the task is addressed by posing a supervised ML solution,

considering in situ data information about crop yield and hailstorm crop damage, and training regression/classification algorithms on these data, with Earth Observation (EO) layers used as predictor feature variables.

Thus, these services both leverage on an extensive use of EO data. More in detail, (i) the “drought service” makes use of the EO zonal statistics leveraging on the Normalized Difference Vegetation Index (NDVI) and its anomalies; while the “hailstorm service” works with Change Detection DPI (specifically, the difference percentage) of EO features. It involves both biophysical parameters such as the Leaf Area Index (LAI), as well as vegetation indices such as the Normalized Difference Vegetation Index (NDVI).

Table 7 - Summary of EO data features used for the calibration/prediction of drought and hailstorm services.

Satellite Platform	EO Feature	EO Change Detection Feature	Spatial Resolution (m)
Sentinel 2 MSI	NDVI	DPI	10
	LAI		20
Moderate Resolution Imaging Spectroradiometer (MODIS)	NDVI	NDVI Anomaly	250

Albeit the two services come as two different tasks and, potentially, as separate tools, the ML pipeline, adopted and implemented within the scopes of the pilot, is shared.

More in detail, the ML framework consists of five principal procedures.

Pre-Analysis

First of all, a preliminary analysis of the data which furnishes a first understanding of the relations in play was performed.

- Y-X descriptive statistics, histograms, box plots;
- correlation analysis: correlation matrix (Y-X).

Outlier Detection

Then, a detailed analysis focused on outlier events was planned. Specifically, this procedure makes use of PCA and Leave-One-Out Cross Validation (LOOCV) as well as residual and influence plots. Specifically, outlier analysis was performed through PCA and Leave-One-Out Cross Validation on the transformed Y-X feature space, and the application of the appropriate distance threshold values on F-residuals, as indicated by the residual and influence plots. The specific outlier detection method, applied in this case, involves the use of the following tools:

- Mahalanobis distance metric;
- F-residuals vs. Hotelling T2 plots.

Model Training

The third procedure consists of the main model training. Key elements of the training procedure are the internal steps of:

- hyperparameter tuning;
- feature selection.

ML algorithms have hyperparameters that need to be set in order to achieve optimal results. Often suitable parameter values are not obvious and it is preferable to tune the hyperparameters, namely automatically identifying values that lead to the best performance.

In the case of Drought service, the ML algorithm predicts the final crop yield, from a list of predictors that include time series of Vegetation Features (NDVI) and Change Detection Features (NDVI Anomaly). Some of

these features contribute a lot in the prediction while some others do not. The technique of extracting a subset of relevant features is called feature selection. Feature selection can enhance the interpretability of the model, speed up the learning process and improve the learner’s performance. This step is performed on different classes of features: all variables, only variables which are highly correlated with the predictor, highly and mildly correlated variables, selection by filter.

The abovementioned optimization steps are governed by the method that they are evaluated on, as far as prediction performance. The ML framework imposes a Nested Cross Validation approach. In order to obtain honest performance estimates for a learner all parts of the model building process, like pre-processing and model selection steps, should be included in the resampling.

For steps that themselves require resampling like parameter tuning or feature selection this results in two nested resampling loops. In the outer resampling loop, we have three pairs of training/test sets. On each of these outer training sets parameter tuning is done, thereby executing the inner resampling loop. This way, we get one set of selected hyperparameters for each outer training set. Then, the learner is fitted on each outer training set using the corresponding selected hyperparameters and its performance is evaluated on the outer test sets.

During the training of each model, the dataset is partitioned into training and prediction subsets (70-30% of the dataset). On the training subset the models are evaluated through Nested Cross Validation (due to the limited number of samples) with repeated 10-Fold Cross Validation on the inner loop and 4-Fold Cross Validation on the outer, which gives an unbiased performance hint on how the model would predict, over different hyperparameters and feature selection. Nested Cross Validation is done to get an estimate of a model’s performance.

- Train/predict is done to create the final predictions;

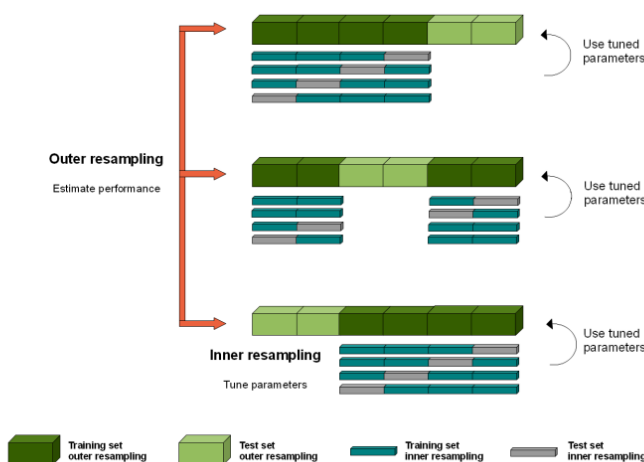


Figure 38 - Nested resampling for parameter tuning with 3-fold Cross Validation in the outer and 4-fold Cross Validation in the inner loop.

- The ML algorithms are chosen within the Support Vector Machine and Random Forest “family”:

Support Vector Machine (libSVM)

- *epsilon SVM algorithm: Regression;*
- *C- SVM algorithm: Classification;*
- Kernel Type: Radial Basis function (RBF)
- Hyperparameter Tuning: parameters grid search (k-fold Cross Validation, k=10)
 - Cost (C);

- Epsilon (ϵ);
 - Gamma (γ).
- Feature selection (correlation p-values, univariate filters).

Random Forest

- Feature selection is performed internally by tuning.
- Hyperparameter tuning: parameters grid search (k-fold Cross Validation, k=10):
 - max tree depth;
 - min_samples_leaf;
 - min_samples_split;
 - ntree: number of branches will grow after each time split.

Model Testing

In particular, the evaluation procedure is performed in the final step of the ML pipeline. Ultimate evaluation of the selected model is performed on the “unseen” prediction subset.

Regression Models Evaluation

The following statistics were used as evaluation metrics, for the developed regression models.

RMSEC	Root Mean Square Error of Calibration
R ² Cali	Coefficient of Determination for Calibration
RMSEV	Root Mean Square Error of Validation
R ² Vali	Coefficient of Determination for Validation
RPD ¹⁷	Residual Prediction Deviation
RPIQ ¹⁸	Residual Prediction Interquartile Range

Classification Models Evaluation

Overall Accuracy: the number of correctly classified pixels divided by the total number of pixels;

Precision: the proportion of true positives divided by the total number of pixels of this class;

Recall: the proportion of the true positives to the total number of pixels that class;

F1 score: Harmonic mean of precision and recall;

Cohen’s Kappa: essentially evaluates how well the classification performed as compared to just randomly assigning values.

Machine Learning Software

The pilot takes advantage of the Orfeo Toolbox and several ML libraries and tools within the R environment. In particular, the main R-libraries, among others, are ggplot2, caret, e1071, MLR, Metrics, MLmetrics.

Below we provide some initial performances obtained by training on a small dataset of Spain drought cases, predicting final yield (t/h) with NDVI Anomaly predictor features.

¹⁷ This statistic computes the Residual Prediction Deviation (RPD), which is defined as the standard deviation of observed values divided by the Root Mean Square Error or Prediction (RMSEP). The RDP takes both the prediction error and the variation of observed values into account, providing a metric of model validity that is more objective than the RMSEP and more easily comparable across model validation studies. The greater the RPD, the better the model's predictive capacity.

¹⁸ This function is more suitable than RPD, in highly skewed datasets, as it uses the IQR instead of the St. Dev.

Table 8 - Initial performances obtained on a small dataset of Spain drought cases.

SVM Regression						
Crop	RMSEC	R ² Cali	RMSEV	R ² Vali	RPD	RPIQ
Barley	314.68	0.79	509.99	0.42	1.3	1.02
Wheat	5.17	0.98	533.6	0.49	1	0.92
Random Forest Regression						
Crop	RMSEC	R ² Cali	RMSEV	R ² Vali	RPD	RPIQ
Barley	314.68	0.79	515.68	0.37	1.2	1.05
Wheat	5.17	0.98	536.95	0.46	1.1	0.9

Type of contribution to provide to the INFINITECH library of ML/DL algorithms

The contribution to the INFINITECH library of ML/DL algorithms will be only a ML framework, and the trained models obtained on proprietary data.

All the required inputs to the ML algorithms are provided by the Agroapps API¹⁹, a collection of RESTful endpoints that provide farm related data.

The output regression/classification layers are saved either to a geojson or a geotiff file.

3.13 Pilot 15 - Open Inter-Banking

As described in the previous deliverable D5.7, Pilot 15 aims to develop, integrate and deploy a data-intensive system to extract key concepts from internal operational documents, using ML and natural language processing (NLP) approaches.

Although general-purpose tools are easily available and neural techniques demonstrate very accurate language modeling and inference capabilities (e.g., BERT [32] or GPT-3 [33]), straightforward applications of such neural methods in business process mining scenarios are still limited. The pilot exploits thus a neural pre-training method optimized against legacy semantic resources able to minimize the training effort.

At the current stage, the pilot has investigated the application of a **zero-shot learning approach** based on the (re-)use of complex predictive NLP models, even in scenarios where manually labelled datasets are limited (or even absent). In particular, the pilot exploits a **semantic model**, e.g., a **knowledge graph (KG) in the form of a process tree (PT)**, as a linguistically augmented information source, where domain-specific relations between concepts are made explicit.

In the suggested augmentation, we map individual relations into short texts, able to make semantic evidence about the domain (e.g., denitions or hierarchical relations) available since the early phases of the training of a Transformer-based architecture.

The suggested approach allowed thus to pre-train a BERT-based model through a large-scale domain-specific training dataset (up to 1.2M controlled sentences), which can be easily extracted from the semantic resource, including concept definitions and ontological relations within a taxonomy (i.e., the PT).

As a result, the pre-trained model allows i) text classification by associating raw texts to (possibly multiple) nodes of the KG and ii) supporting more complex natural language inference tasks in the domain, e.g., by predicting properties of the process described in the PT taxonomy.

The experimental results over the pre-training stage and the automatic acquisition of process-related metadata suggest the feasibility of the approach at the industry level.

¹⁹ <https://api.agroapps.gr/>

4 Conclusions

In the current deliverable, we have described in detail the development of an end-to-end framework for the definition and instrumentation of standardized ML/DL pipelines on top of the MLflow open-source platform. This framework is the core component of the INFINITECH library of ML/DL algorithms, and it allows the production of ready-to-use ML and DL models as well as their deployment and publishing as microservices. In this way, the described framework enables and supports the ML/DL needs and activities of the INFINITECH partners and the needs of banks, insurance companies, and other potential adopters and users of the INFINITECH platform.

Additionally, in D5.8 we have also documented the progress of the ML/DL tasks of each INFINITECH pilot, reporting the tasks' definitions for each pilot, the innovative or state-of-the-art ML/DL algorithms implemented and evaluated by the pilots, the initial performances of these algorithms, and the type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms.

In the third and last version of the deliverable (D5.9) we plan to describe the extension of our end-to-end framework to some interested and selected INFINITECH pilots and its integration with the work conducted by HPE within the WP6 "Tailored Sandboxes and Testbeds for Experimentation and Validation". In particular, in WP6 HPE and other INFINITECH partners are planning to design and develop an approach that leverages machine learning operations (MLOps) based on Kubeflow. MLOps are a set of practices that have the objective of deploying and maintaining in production ML/DL models in an efficient and reliable way. To this end, next months will be dedicated to enforce the collaboration between the task T5.5 and the WP6, and in particular between FBK and HPE working on a joint approach.

The deliverable D5.9 will also document the final results obtained by each pilot in the ML/DL tasks and the ML/DL tools, frameworks and microservices provided to the library of ML/DL algorithms. We will also dedicate a specific section of the third and last deliverable to an ethical evaluation of the algorithmic approaches proposed by the pilots as well as to a description of specific measures and approaches aiming at improving the fairness and the explainability of the proposed ML/DL algorithms.

Table 9 – T5.4 objectives and D5.8 achievements.

Objectives	Comment
Develop tools to support the easy definition and deployment of ML/DL pipelines.	D5.8 describes in detail the development of an end-to-end framework for the definition and instrumentation of standardized ML/DL pipelines on top of the MLflow open-source platform. This framework allows the production of ready-to-use ML and DL models as well as their deployment and publishing as microservices, thus enabling the ML/DL activities of the INFINITECH partners and satisfying the needs of banks, insurance companies, and other potential adopters and users of the INFINITECH platform.
Document and support the ML/DL needs and planned solutions of the pilots.	D5.8 documents the progress of the ML/DL tasks of each INFINITECH pilot, reporting the tasks' definitions for each pilot, the innovative or state-of-the-art ML/DL algorithms implemented and evaluated by the pilots, the initial performances of these algorithms, and the type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms.

Table 10 – T5.4 KPIs and D5.8 achievements.

KPI	Comment
Reduction of effort for the development of ML/DL functionalities in the sector >= 50%	D5.8 describes in detail the development of an end-to-end framework that integrates Open APIs as well as a way to select ML/DL algorithms, define a pipeline, deploy and evaluate them. This framework significantly reduces the effort for the development and deployment of ML/DL algorithms in the financial and insurance sectors.
ML/DL algorithms in the INFINITECH library and open repositories (i.e. OpenML) >= 20	D5.8 describes the progress of the ML/DL tasks of each INFINITECH pilot, reporting the tasks' definitions for each pilot, the innovative or state-of-the-art ML/DL algorithms implemented and evaluated by the pilots, the initial performances of these algorithms, and the type of contribution each pilot is planning to provide to the INFINITECH library of ML/DL algorithms. More specifically, D5.8 already identifies and describes more than 20 ML/DL algorithms for the finance and insurance sectors. Some examples are the following ones: Random Forest, Support Vector Machine, Long Short Term Memory (LSTM) neural networks, K-means clustering, Logistic Classifier, Naive Bayes, Hierarchical Hidden Markov Models, Extreme Gradient Boosting, Neural Collaborative Filtering, Neural Graph Collaborative Filtering, Triple2Vec, etc.

5 BIBLIOGRAPHY

- [1] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 173-182.
- [2] Holton, G. A. (2014). *Value-at-Risk: Theory and Practice* (2nd ed.). London: Academic Press.
- [3] Acerbi, C., & Tasche, D. (2002). Expected Shortfall: a natural coherent alternative to Value at Risk. *Economic notes*, 31(2), 379-388.
- [4] Longerstaey, J., & Spencer, M. (1996). Riskmetrics — technical document. *Morgan Guaranty Trust Company of New York*: 51, 54
- [5] Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36(3), 1181-1191.
- [6] Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D.C., Rangapuram, S., Salinas, D., Schulz, J., Stella, L., Caner Turkmen, A., and Wang, Y, (2019). GluonTS: probabilistic time series models in Python. *arXiv preprint arXiv:1906.05264*.
- [7] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., and Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. In *Proceedings of Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*.
- [8] Ribeiro, M.T., Singh, S., and Guestrin, C. (2016). “Why should I trust you?”: Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135-1144.
- [9] Lundberg, S.M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, pp. 4768-4777
- [10] Chawla, N.V., Herrera, F., Garcia, S., & Fernandez, A. (2018) SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research*, 61, 863--905.
- [11] Liu, F.T., Ting, K.M., & Zhou, Z.-H. (2008). Isolation Forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422
- [12] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. Springer Link. doi.org/10.1023/A:1010933404324
- [13] Bhattacharya, A. (2020) Effective Approaches for Time Series Anomaly Detection. *towardsdatascience.com*.
- [14] Altman, N.S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3), 175-185
- [15] Rokach, L., & Maimon, O. (2005). Clustering methods. *Data mining and knowledge discovery handbook*. Springer US, 321-352.
- [16] Barros, R., & Santos, S. (2018). A large-scale comparison of concept drift detectors. *Information Sciences*, 451-452, 348-370
- [17] Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the Seventh SIAM International Conference on Data Mining, SDM '07, SIAM*, pp. 443–448
- [18] Fortuna, B., Rupnik, J., Brank, J., Fortuna, C., Jovanoski, V., Karlovcec, M., Kazic, B., Kenda, K., Leban, G., Muhic, A., Novak, B., Novljan, J., Papler, M., Rei, L., Sovdat, B., Stopar, L., Grobelnik, M., & Mladenic, D. (2014). QMiner: Data Analytics Platform for Processing Streams of Structured and Unstructured Data. In

Proceedings of the Software Engineering for Machine Learning Workshop at Neural Information Processing Systems (NIPS 2014)

- [19] Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226-231.
- [20] Stopar, L., Skraba, P., Grobelnik, M., & Mladenic, D. (2019). StreamStory: Exploring multivariate time series on multiple scales. *IEEE Transactions on Visualization and Computer Graphics*, 25(4), 1788-1802
- [21] Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32, 41-62
- [22] Singh, A. (2019). Anomaly detection in the Ethereum network. A thesis for the degree of Master of Technology/Indian Institute of Technology Kanpur.
- [23] Farrugia, S., Ellul, J., & Azzopardi, G. (2020). Detection of illicit accounts over the Ethereum blockchain. *Expert systems with applications* 150: 113318.
- [24] Kumar, N., Singh, A., Handa, A., & Shukla, S.K. (2020). Detecting malicious accounts on the Ethereum blockchain with supervised learning. In *Proceedings of International Symposium on Cyber Security Cryptography and Machine Learning*, pp. 94-109. Springer, Cham.
- [25] Poursafaei, F., Hamad, G.B., & Zilic, Z. (2020). Detecting malicious Ethereum entities via application of machine learning classification. In *Proceedings of the 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE.
- [26] Team, C. (2020). Indirect exposure: Why you need to look beyond direct counterparties to understand cryptocurrency address risk. Nov. 2020, <https://blog.chainalysis.com/reports/cryptocurrency-risk-blockchain-analysis-indirect-exposure>
- [27] Kılıç, B., Özturan, C., & Şen, A. (2021). Finding convex subgraphs in Blockchain transaction graphs, submitted to BCCA 2021 conference (under review).
- [28] Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y. P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., & Wiessner, E. (2018, 11). Microscopic Traffic Simulation using SUMO. *21st International Conference on Intelligent Transportation Systems (ITSC)*, 2575-2582. IEEE Xplore. [/doi.org/10.1109/ITSC.2018.8569938](https://doi.org/10.1109/ITSC.2018.8569938)
- [29] Pnevmatikakis, A., Kanavos, S., Matikas, G., Kostopoulou, K., Cesario, A., & Kyriazakos, S. (2021). Risk assessment for personalized health insurance based on real-world data. *Risks*. 9(3):46. <https://doi.org/10.3390/risks9030046>
- [30] Sharpe, W.F. (1994). The Sharpe Ratio. *The Journal of Portfolio Management* 21(1), 49-58.
- [31] FATF. (2019). Guidance for a risk-based approach, virtual assets and virtual asset service providers, FATF, 2019, Paris. <https://www.fatf-gafi.org/publications/fatfrecommendations/documents/guidance-rba-virtual-assets.html>
- [32] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- [33] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.