AIDERS

Deliverable 3.3 Real-time algorithms for machine learning

Coordinator Name: **Christos Panayiotou**

Coordinator Email: **christosp@ucy.ac.cy**

Project Name: **Real-time Artificial Intelligence for DEcision support via RPAS data analyticS**

Acronym: **AIDERS**

Grant Agreement: **873240**

Project Website: http://www.kios.ucy.ac.cy/aiders/

Version: **1.0**

Submission Date: **12/05/2021**

Dissemination Level: **Public**

The project has received funding from the European Union Civil Protection Call for proposals UCPM-2019-PP-AG for prevention and preparedness projects in the field of civil protection and marine pollution under grant agreement – 873240– AIDERS.

Funded by European Union Civil Protection

Contents

Executive Summary

Machine learning algorithms typically rely on training data to learn high-level representations and carry out a given identification, classification, recommendation or prediction task. This deliverable focuses on identifying machine learning algorithms and data that can be used to train those algorithms with the goal to achieve increased situational awareness in emergency response. In particular, we a) identify the requirements for gaining situational awareness based on the requirements provided by the end-users, b) associate the user requirements with machine learning algorithms, c) provide an overview of state-of-the-art approaches, d) define the appropriate pre-processing techniques for each data type, e) provide available datasets and present a data collection approach, f) provide an overview of AI applications focused on processing multi-sensor data in emergency response and categorize them into the three scenarios of focus of the AIDERS project being: i) fires, ii) earthquakes and iii) flooding.

1. Introduction

Data and *Artificial Intelligence* (AI)—and *machine learning* (ML) algorithms in particular—can be instrumental in assisting first responders in handling emergency situations. For example, in the event of a fire, firefighters can gain better situational awareness and make better decisions by having access to a visualization of the propagation of a fire, receive live video from the scene or be provide with an estimation of the number of buildings or people in danger. The availability of all this information in a single platform has the potential to make their emergency response operations faster and more effective. Most importantly, the availability of these data enables the implementation of machine learning algorithms that can make predictions regarding the emergency situation. Such algorithms may include the detection of smoke or fire and estimations regarding its propagation, the altitude and spread of fire, the identification of the vegetation type. The implementation of such ML algorithms can provide first responders with critical insights which would be otherwise hard to get and help them in their operations faster and more effective.

An important and challenging part of the implementation of ML algorithms, is the collection and automated processing of multi-sensor data—*i.e.*, data collected from various sensors. The main challenges include *a)* the identification of the data required to cover the user-needs and that can be used to train algorithms, *b)* the definition of the data collection process, that is the definition of the methodology for collecting the appropriate data and *c)* the definition of the process for handling and processing the data collected – this is required since the data collected from sensors can be noisy and inaccurate. This deliverable addresses the above challenges by exploring the middleware layers required to support the deployment and execution of real-time ML algorithms. In particular, the real-time dimension of this challenge brings several constraints with regards to performances and scalability of the algorithms. This includes continuously storing and processing huge volume and diversity of input raw data to deliver an instantaneous feedback to the stakeholders to take appropriate decisions along mission-critical scenarios.

This deliverable is structured as follows. First, the real-time machine learning algorithms required for gaining situational awareness are covered (cf. Section 2). Then, an overview of existing state-of-the-art of data representations for spatiotemporal information is reviewed (cf. Section 3). In particular, we consider 15 candidate data management solutions that can be adapted, extended or improved to be potentially included in the AIDERS AI toolkit, thus aiming to significantly improve the situational awareness of first responders in real-time.

2. Real-time Machine Learning Algorithms

Along the past decade, multiple ML algorithms have developed by the research community and quickly found applications in various fields, from leisure to more critical systems, like autonomous vehicles and mission-critical systems. In the latter case, the employed ML algorithms are expected to process continuous information signals (incl. data streams like video or sound) to produce insightful, and accurate, feedbacks to first-responders in real-time. However, achieving such a tight real-time support raises several challenges for ML algorithms and the supporting infrastructure.

In this section, we therefore explore the requirements for deploying different classes of ML algorithms in the context of mission-critical scenarios to produce real-time feedbacks. To do so, we consider different popular classes of ML algorithms that are intended to work with live data streams and raise the challenges that such a support imposes to the underlying software infrastructure. In particular, given the important volume of data to be processed by such ML algorithms, we carefully look at the issues related to the storage, indexation, and processing of spatiotemporal data streams, which are central to the AIDERS project. As AIDERS aims to deliver a modular and extensible AI toolkit to assist first-responders in the field, we do not focus on a restricted set of ML algorithms supporting some specific feature, but rather consider families of algorithms and the requirements they impose on the data management layers, thus keeping in mind that the list of ML algorithms may be extended in the future without questioning the relevance of the data management layer.

By exploring the potential of existing ML algorithms to deliver real-time insights to first-responders, we intend to identify the underlying data representation that is required to store spatiotemporal data streams, as produced by drones and other equipment, and make them available to ML algorithms. Furthermore, to cope with the expected performance constraints, we also consider the tight integration of ML algorithms with the foreseen data management layer as a key to a successful implementation of real-time ML algorithms.

Supervised *vs*. unsupervised learning

Supervised learning is a machine learning approach that builds on labeled datasets. These datasets are designed to train algorithms into classifying data or predicting outcomes accurately. Using labeled inputs and outputs, the model can measure its accuracy and learn over time. Supervised learning can be separated into two types of problems when data mining: classification and regression:

- **Classification** problems use supervised algorithm to accurately assign input data into identified categories, such as classifying spam in a separate folder from your inbox. Linear classifiers, support vector machines, decision trees and random forest are all common types of classification algorithms;
- **Regression** is another type of supervised learning method that understands the relationship between dependent and independent variables. Regression models are helpful for predicting numerical values based on different input data points, such as sales revenue projections for a given business. Some popular regression algorithms are linear regression, logistic regression and

polynomial regression. Using previously recorded data like temperature, humidity, air quality, it is possible to forecast weather.

Unsupervised learning uses ML algorithms to analyze and cluster unlabeled data sets. These algorithms are intended to discover hidden data patterns with no human intervention. Unsupervised learning models are used for three main tasks: clustering, association, and dimensionality reduction:

- **Clustering** is a data mining technique for grouping unlabeled data based on their similarities or differences. For example, *K*-means clustering algorithms assign similar data points into groups, where the *K* value represents the size of the grouping and granularity. This algorithm can collect data from sensors attached to a drone and process them to separate an area into small clusters, for example, to discriminate between burned and healthy vegetation.
- **Association** is another type of unsupervised learning method that uses different rules to find relationships between variables in a given dataset. These methods are frequently used for market basket analysis and recommendation engines, along the lines of "*Customers Who Bought This Item Also Bought*" recommendations.
- **Dimensionality reduction** is a learning technique used when the number of features in a given dataset is too high. It reduces the number of data dimensions to a manageable size while also preserving the data integrity. Often, this technique is used in the preprocessing data stage, such as when auto-encoders remove noise from visual data to improve picture quality.

The main distinction between the two approaches is the use of labeled datasets. Supervised learning uses labeled input and output data, while an unsupervised learning algorithm does not. In supervised learning, the algorithm "*learns*" from a training dataset by iteratively making predictions on the data and adjusting for the correct answer. For example, a supervised learning algorithm can be trained with forest fire image samples to learn how to detect smoke and fire from a standard camera. While supervised learning models tend to be more accurate than unsupervised learning models, they require upfront human intervention to label the data appropriately. For example, a supervised learning model can predict how long your commute will be based on the time of day, weather conditions and so on. But first, you have to train it to know that rainy weather extends the driving time. For AIDERS project, it can train a neural network for aerial image understanding, using a dataset of appropriately labeled pictures where multiple disaster events like fire, floods, rubble, traffic accidents and also normal situations. Unsupervised learning models, in contrast, work on their own to discover the inherent structure of unlabeled data. Note that they still require some human intervention for validating output variables. For example, an unsupervised learning model can identify that online visitors often purchase groups of products at the same time. In addition to that, we also expect incorporate more recent AI algorithms based on deep learning, like *Convolutional Neural Networks* (CNN), which have demonstrated their capability to detect and localize specific hazards with accuracy (cf. Deliverable 3.1).

In both cases, input data are required to be stored and processed by the infrastructure in order to support the predictions—and eventually the training phase in the case of supervised machine learning. The volume of data streams produced by drones imposed by mission-critical scenarios challenges the data layers to scale and keep enriching the feedbacks proposed to first-responders. Beyond the traditional

supervised and unsupervised ML algorithms, several improvements have been proposed in the area to deal with the scarcity of resources imposed by the field deployment (both in terms of storage capacity and computing resources).

Incremental learning

Incremental learning is a method of machine learning in which input data is continuously used to extend the existing model's knowledge—*i.e.*, to further train the model. It represents a dynamic technique of supervised and unsupervised learning that can be applied when training data becomes gradually available over time (like data streams) or when its size reaches out of system memory limits (because of hardware constraints). Algorithms that can facilitate incremental learning are known as *incremental machine learning algorithms*.

Many traditional machine learning algorithms inherently support incremental learning. Other algorithms can be adapted to facilitate incremental learning. Examples of incremental algorithms include decision trees, decision rules, artificial neural networks or the incremental SVM. The aim of incremental learning is for the learning model to adapt to new data without forgetting its existing knowledge. Some incremental learners have built-in some parameter or assumption that controls the relevancy of old data, while others, called *stable incremental machine learning* algorithms, learn representations of the training data that are not even partially forgotten over time. Fuzzy ART and Topo ART are two examples for this second approach.

Incremental algorithms are frequently applied to data streams or big data, addressing issues in data availability and resource scarcity, respectively. Stock trend prediction and user profiling are some examples of data streams where new data becomes continuously available. Applying incremental learning to big data aims to produce faster classification or forecasting times.

Online machine learning

Online machine learning is a method of machine learning in which data becomes available in a sequential order and is used to update the best predictor for future data at each step, as opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once. Online learning is a common technique used in areas of machine learning where it is computationally infeasible to train over the entire dataset, requiring the need of out-of-core algorithms.

It is also used in situations where it is necessary for the algorithm to dynamically adapt to new patterns in the data, or when the data itself is generated as a function of time—*e.g.*, stock price prediction. Online learning algorithms may be prone to catastrophic interference, a problem that can be addressed by incremental learning approaches.

Reinforcement learning

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. RL is one of

three basic machine learning paradigms, alongside supervised learning and unsupervised learning. RL differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead, the focus is on finding a balance between exploration and exploitation phases. The environment is typically stated in the form of a *Markov decision process* (MDP), because many RL algorithms for this context use dynamic programming techniques. The main difference between the classical dynamic programming methods and RL algorithms is that the latter does not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

This approach is particularly interesting as the MDP offers a more compact representation of the captured knowledge, while RL offers a sequential decision process that builds upon previous experience and can thus cope with the characteristics of spatiotemporal data streams.

Deep reinforcement learning

Deep reinforcement learning (deep RL) is a subfield of machine learning that combines *reinforcement learning* (RL) and *deep learning*. RL considers the problem of a computational agent learning to make decisions by trial and error. Deep RL incorporates deep learning into the solution, allowing agents to make decisions from unstructured input data without manual engineering of the state space. Deep RL algorithms are able to take in very large inputs (*e.g.*, every pixel rendered to the screen in a video game) and decide what actions to perform to optimize an objective (*e.g.*, maximizing the game score). Deep reinforcement learning has been used for a diverse set of applications including but not limited to robotics, video games, natural language processing, computer vision, education, transportation, finance and healthcare, where they have produced results comparable to and in some cases surpassing human expert performance. For instance, it can be used to detect specific objects in images and videos, like people, buildings, cars, smoke or vegetation, and track them. The use of deep reinforcement learning in the context of AIDERS can pave the way to assist first-responders in rescue operations by offering automated pilot modes to track persons or hazards by following the evolution of a situation (monitoring the front of a forest fire), thus taking control of the drone by sending commands based on information produced by integrated deep learning algorithms.

The combination of RL with deep learning faces nonetheless the challenge of storing and processing large volume of data to reach the expected accuracy.

Synthesis

As a summary of the overview of ML techniques that can be considered in the context of AIDERS (cf. Deliverable 3.1 for a detailed description of concrete and appropriate ML algorithms), we can observe that the delivery of real-time ML algorithms requires to carefully consider the resource requirements in the context of mission-critical systems with limited capacities (both in processing and storage). While ML algorithms may provide some advanced techniques to support the incremental improvement of learned models, the traceability and explainability of decisions may still require to keep track of as much information as possible. This challenge therefore calls for the selection of an appropriate data

management layer that can expose spatiotemporal data streams to a wide diversity of ML algorithms, while offering a performant and compact storage of acquired information.

In the following section, we therefore review existing data management solutions that can be adopted to store and process spatiotemporal data streams while keeping their exploitation by ML algorithms. Our objective is to identify a candidate solution that can be embedded in the AI toolkit to be delivered to first-responders by AIDERS.

3. Spatiotemporal Data Representations

Our review of data management solutions for spatiotemporal data representations considers four categories of solutions:

1. Data management solutions that provide a native support for spatial data, but do not necessarily consider time as a first-class entity,
2. Data management solutions that provide a native support for time series, but may miss appropriate constructions for spatial dimensions and/or complex data representations,
3. Data management solutions that enclose a native spatiotemporal support,
4. Other data management solutions that offer a close integration of the data management layer with ML algorithms.

For each of these categories, we describe the solutions commonly adopted in this domain and consider their relevance with regards to the requirements of AIDERS, which can be summarized as:

a. Providing a flexible data model to cope with the diversity of sensors and missions,
b. Providing a support for temporal data streams,
c. Providing a support for geospatial data streams,
d. Providing a support for large and complex objects (*e.g.*, video),
e. Proposing a tight integration of the storage and processing layers (in particular ML processing).

The remainder of this section therefore lists the solution of relevance for AIDERS, before delivering a synthesis and a comparison of existing solution to guide the choice of the most suitable candidate for the implementation of the AI toolkit.

Data management solutions with spatial support

Spatial databases are databases optimized for storing and querying geometric data, like lines and polygons. They use spatial indexes to optimize spatial queries using specific datatypes, such as geohash, grid, and various kinds of trees (quadtree/octree, R-tree etc.).

PostGIS

PostGIS (https://postgis.net) is an extension of PostgreSQL database to deal with spatial datatypes. In particular, it adds support for geographic objects allowing location queries to be run in SQL:

```
SELECT superhero.name
FROM city, superhero
WHERE ST_Contains(city.geom, superhero.geom)
AND city.name = 'Gotham';
```

To do so, PostGIS adds extra types (geometry, geography, raster and others) to the PostgreSQL database. It also adds functions, operators, and index enhancements that apply to these spatial types. These additional functions (*e.g.*, the above function `ST_Contains`), operators, index bindings and types augment the power of the core PostgreSQL DBMS, making it a fast, feature-plenty, and robust spatial database management system. PostGIS follows the Open Geospatial Consortium's "*Simple Features for SQL Specification*"[1] and has been certified as compliant with the "*Types and Functions*" profile. PostGIS is open-source software, released under the GNU General Public License.

Spatial index is one of the three key features of a spatial database. Indexes make using a spatial database for large data sets possible. Without indexing, any search for a feature would require a "*sequential scan*" of every record in the database. Indexing speeds up searching by organizing the data into a search tree which can be quickly traversed to find a particular record. Standard database indexes create a hierarchical tree based on the values of the column being indexed. Spatial indexes are a little different – they are unable to index the geometric features themselves and instead index the bounding boxes of the features. Instead of comparing two tables of 10,000 records with each other, thus requiring 100,000,000 comparisons, spatial indexes reduce this cost down to 20,000 comparisons.

SpatiaLite

SpatiaLite (https://www.gaia-gis.it/fossil/libspatialite/index) is an open-source library intended to extend the SQLite core to support fully-fledged spatial SQL capabilities. It is similar to PostGIS, although SpatiaLite is not based on a client-server architecture: it adopts a simpler architecture, being directly embedded within the application itself. A complete database simply is an ordinary file which can be freely copied and transferred from one computer/OS to a different one without any special precaution. SpatiaLite extends SQLite's existing spatial support to cover the OGC's SFS specification (like PostGIS). It is not necessary to use SpatiaLite to manage spatial data in SQLite, which has its own implementation of R-tree indexes and geometry types. However, SpatiaLite is needed for advanced spatial queries and to support multiple map projections.

Elasticsearch

While Elasticsearch is best known for its full-text search capabilities (https://www.elastic.co), it also features full geospatial support. Elasticsearch offers two ways to represent geodata: *i)* latitude-longitude pairs using `geo_point` field type,[2] and *ii)* complex shapes defined in GeoJSON using `geo_shape` field type.[3] We can also provide geo point data in the form of geo hash. Similar to `geo_shape` type,

[1] https://www.ogc.org/standards/sfs
[2] https://www.elastic.co/guide/en/elasticsearch/reference/current/geo-point.html
[3] https://www.elastic.co/guide/en/elasticsearch/reference/current/geo-shape.html

`geo_shape` uses GeoJSON structure to query documents. Below is a sample query to fetch all documents that fall within given top-left and bottom-right coordinates:

```
{
  "query": {
    "bool": {
      "must": {
        "match_all": {}
      },
      "filter": {
        "geo_shape": {
          "region": {
            "shape": {
              "type": "envelope",
              "coordinates": [[75.00, 25.0], [80.1, 30.2]]
            },
            "relation": "within"
          }
        }
      }
    }
  }
}
```

Here, relation determines spatial relation operators used at search time. The list of operators supported by Elasticsearch are typically:

- `INTERSECTS` – (default) returns all documents whose `geo_shape` field intersects the query geometry
- `DISJOINT` – retrieves all documents whose `geo_shape` field has nothing in common with the query geometry
- `WITHIN` – gets all documents whose `geo_shape` field is within the query geometry
- `CONTAINS` – returns all documents whose `geo_shape` field contains the query geometry

GeoShape types are indexed by decomposing the shape into a triangular mesh and indexing each triangle as a 7-dimensions point in a k-dimensional B-tree.[4] This provides near perfect spatial resolution (down to $1e^{-7}$ decimal degree precision) since all spatial relations are computed using an encoded vector representation of the original shape instead of a raster-grid representation as used by the Prefix trees indexing approach. Performance of the tessellator primarily depends on the number of vertices that define the polygon/multi-polygon. To efficiently represent shapes in an inverted index, shapes are converted into a series of hashes representing grid squares (commonly referred to as "rasters") using implementations of a PrefixTree. The tree notion comes from the fact that the PrefixTree uses multiple grid layers, each with an increasing level of precision to represent the Earth. This can be thought of as increasing the level of detail of a map or image at higher zoom levels. Since this approach causes precision issues with indexed shape, it has been depreciated in favor of a vector indexing approach that indexes the shapes as a triangular mesh. Multiple PrefixTree implementations are provided:

[4] https://en.wikipedia.org/wiki/K-D-B-tree

- *GeohashPrefixTree* - Uses geohashes for grid squares. Geohashes are base32 encoded strings of the bits of the latitude and longitude interleaved. So, the longer the hash, the more precise it is. Each character added to the geohash represents another tree level and adds 5 bits of precision to the geohash. A geohash represents a rectangular area and has 32 sub rectangles. The maximum number of levels in Elasticsearch is 24; the default is 9;
- *QuadPrefixTree* - Uses a quadtree for grid squares. Similar to geohash, quad trees interleave the bits of the latitude and longitude the resulting hash is a bit set. A tree level in a quad tree represents 2 bits in this bit set, one for each coordinate. The maximum number of levels for the quad trees in Elasticsearch is 29; the default is 21.

The indexing implementation selected relies on a spatial strategy for choosing how to decompose the shapes (either as grid squares or a tessellated triangular mesh). Each strategy answers the following:

- What type of Shapes can be indexed?
- What types of Query Operations and Shapes can be used?
- Does it support more than one Shape per field?

Geocouch

GeoCouch (https://github.com/couchbase/geocouch) is a spatial extension for Couchbase and Apache CouchDB, which are two open-source document-oriented NoSQL databases. They store data as "documents", as one or more field/value pairs expressed as JSON. Field values can be simple things like strings, numbers, or dates; but ordered lists and associative arrays can also be used. CouchDB supports special documents within databases known as “design documents”. These documents, mostly driven by JavaScript you write, are used to build indexes, validate document updates, format query results, and filter replications. All documents are exposed through HTTP.

GeoCouch follows the GeoJSON format and brings support for point, linestring and polygon datatypes to CouchDB. One can for instance push a design document inside the database with a spatial function:

```
{"spatial":
  {"points": "function(doc) {
    if (doc.loc) {
      emit([{
          type: \"Point\",
          coordinates: [doc.loc[0], doc.loc[1]]
        }],
        [doc._id, doc.loc]
      );
    }};"
  }
}
```

Then, it can query all the data whose coordinates are inside an area with the following URL:
`http://example:5984/places/_design/main/_spatial/points?start_range=[0,0]&end_range=[180,90]`

It is worth noting that the development of GeoCouch does not seem active since the last update on the official repository dates from May 2018.

In the case of Couchbase, geospatial data are now natively supported. Data must be indexed on geospatial, one or more points that can be defined by their latitude/longitude or a Geohash. Then we can query the data whose coordinates are located within a radius, a rectangular or polygonal area.

ArangoDB

ArangoDB (https://www.arangodb.com/) is a "multi-model" database, meaning that it can be used as a document database, storing records inside JSON documents, or as a graph database, using these documents to represent nodes and edges, with two attributes to index the edge's vertices. ArangoDB includes the new GeoJSON functionalities based on Google's S2 geospatial index. It supports indexing on a subset of the GeoJSON standard, as well as simple latitude-longitude pairs (Non-GeoJSON mode). As such, we can define points, lines, polygonal chains and polygons. It also provides the following functions for requests:

- `GEO_DISTANCE()` returns the distance between two locations;
- `GEO_CONTAINS()` checks whether a location is inside another one;
- `GEO_INTERSECTS()` checks if two locations intersect each other;
- `GEO_EQUALS()`: checks whether GeoJSON objects are equal or not.

It only manages basic datatypes, like numbers, strings, booleans, and arrays or JSON-like documents.

Redis

Redis is an in-memory database, meaning the dataset resides on the main memory during operation, while it is periodically dumped to disk for persistence. Beyond the standard datatypes, Redis natively deals with spatial data by providing the following commands:

- `GEOADD` adds a geospatial item, defined by its latitude and longitude, to a set in the database. Data is stored into the key as a sorted set, in a way that makes it possible to query the items with the `GEOSEARCH` command. Redis represents positions of elements using a variation of the Geohash technique where positions are encoded using 52-bit integers. The encoding is also different compared to the standard because the initial min and max coordinates used during the encoding and decoding process are different;
- `GEODIST` returns the distance between two members in the geospatial index represented by the sorted set;
- `GEOHASH` returns the standard Geohash strings of the requested geospatial items;
- `GEOPOS` returns the latitude and longitude of the requested geospatial items;
- `GEOSEARCH` returns the members of a sorted set populated with geospatial information using `GEOADD`, which are within the borders of the area specified by a given shape. The query runs in $O(N + log(M))$ where N is the number of elements in the grid-aligned bounding box area around the shape provided as the filter and M is the number of items inside the shape;
- `GEOSEARCHSTORE` does the same as `GEOSEARCH`, but stores the result in destination key.

Geospatial indexing is implemented in Redis using sorted sets as the underlying data structure, but with on-the-fly encoding and decoding of location data and new APIs.[5] This means that location-specific indexing, searching, and sorting can all be offloaded to Redis, with very few lines of code and very little effort, using built-in commands, like `GEOADD`, `GEODIST`, `GEORADIUS`, and `GEORADIUSBYMEMBER`.

Tile38

Tile38 (https://github.com/tidwall/tile38) is an open source (MIT licensed), in-memory geolocation data store, spatial index, and real-time geofence. It supports many geospatial datatypes including:

- Points defined by latitude and longitude;
- Bounding boxes;
- XYZ tiles: rectangle bounding area on earth that is represented by an X, Y coordinate and a Z (zoom) level;
- QuadKey: the same coordinate system as an XYZ tile except that the string representation is a string characters composed of 0, 1, 2, or 3;
- Geohashes;
- GeoJSON format for representing a variety of object types including a point, multipoint, linestring, multilinestring, polygon, multipolygon, geometrycollection, feature, and featurecollection.

For all these datatypes, Tile38 supports the following search criteria:

- `WITHIN`: searches a collection for objects that are fully contained inside a specified bounding area;
- `INTERSECTS`: searches a collection for objects that intersect a specified bounding area;
- `NEARBY`: searches a collection for objects that intersect a specified radius.

However, it cannot deal with even primitive datatypes, such as numbers or booleans. Tile38 is exclusively designed for geospatial data.

Data management solutions with time support

Time-series databases are databases optimized for the storage and querying of time-series data, which are simply data indexed on time.

InfluxDB

InfluxDB (https://www.influxdata.com/) is a time-series database designed for fast-ingestion, and querying of timestamped data, such as events and metrics. However, it can only deal with primitive datatypes like booleans, numbers, strings and timestamps. InfluxDB records automatically include a *_time* field representing the timestamp of the record, and tag fields which can be set freely to index and speed up queries relying on these tags. It does not support ML processing internally but can be connected

[5] https://redislabs.com/docs/redis-for-geospatial-data/

to LoudML which is a new deep learning API that makes it simple to prepare, train, and deploy machine learning models for predictive analytics with InfluxDB.

OpenTSDB

OpenTSDB (https://github.com/OpenTSDB/opentsdb) consists of a *Time Series Daemon* (TSD) that rely on instances of Apache HBase, which is an open-source non-relational distributed database, suited for hosting very large datasets (up to several billions of records per table).
In OpenTSDB, a time series data point consists of:

- A metric name for the time-series;
- A UNIX timestamp (seconds or milliseconds since Epoch);
- A value (64-bit integer or single-precision floating point value);
- A set of tags (key-value pairs) that describe the time series the point belongs to.

It can then efficiently perform aggregations of time-series (computations of n^{th} percentile, average, maximal and minimal values etc.) and interpolate the possible missing values.

Data management solutions with spatio-temporal support

MobilityDB

MobilityDB (https://github.com/MobilityDB/MobilityDB) is a database management system for moving object geospatial trajectories, such as GPS traces. It adds support for temporal and spatio-temporal objects to the PostgreSQL database and its spatial extension PostGIS. It supports the "*Moving Features*" standards from the *Open Geospatial Consortium* (OGC). It allows temporal and spatio-temporal objects to be stored in the database, that is, objects whose attribute values and/or location evolves in time. MobilityDB includes functions for analysis and processing of temporal and spatio-temporal objects and provides support for GiST and SP-GiST indexes.

TimescaleDB

TimescaleDB (https://www.timescale.com) is an open-source, time-series database based on PostgreSQL. As an extension of PostgreSQL, it supports all of its properties, including support for binary data. The primary point of interaction with the data is a *hypertable*, the abstraction of a single continuous table across all space and time intervals, such that one can query it via standard SQL. Virtually all user interactions with TimescaleDB are with hypertables. Creating tables and indexes, altering tables, inserting data, selecting data, etc. can (and should) all be executed on the hypertable.

Internally, TimescaleDB automatically splits each hypertable into chunks, with each chunk corresponding to a specific time interval and a region of the partition key's space (using hashing). The good news is that TimescaleDB is compatible with all other PostgreSQL extensions and, for geospatial data, we'll use PostGIS.

Druid

Druid (https://druid.apache.org) is a real-time time-series analytics database. Internally, data are partitioned by time and optionally by other attributes, then further divided into smaller range of time. Druid natively supports five basic column types: "long" (64-bit signed int), "float" (32-bit float), "double" (64-bit float) "string" (UTF-8 encoded strings and string arrays), Timestamps are treated by Druid as longs. Users can query Druid by sending JSON-formatted documents over HTTP.

Druid allows to retrieve the first and last timestamp of a time-series, and perform aggregations on them, like counting, adding up the records, retrieve the min or max value for a field. It also supports basic geospatial data. For instance, one can filter elements that are contained inside a rectangular, polygonal or circular area:

```
"filter": {
  "type": "spatial",
  "dimension": "spatialDim",
  "bound": {
    "type": "rectangular",
    "minCoords": [10.0, 20.0],
    "maxCoords": [30.0, 40.0]
  }
}
```

GeoWave

GeoWave (https://github.com/locationtech/geowave) provides geospatial and temporal indexing capability to key/value stores (currently Apache Accumulo, Apache HBase, Apache Cassandra, Amazon DynamoDB, Cloud Bigtable, Redis, RocksDB, and Apache Kudu, as well as direct FileSystem support). It includes implementations that support OGC spatial types (up to 3 dimensions), and both bounded and unbounded temporal values. Both single and ranged values are also supported in all dimensions. GeoWave's geospatial support is built on top of the GeoTools project extensibility model. This means that it can integrate natively with any GeoTools-compatible project, such as GeoServer and UDig, and can ingest GeoTools compatible data sources. Basically, GeoWave is working to bridge geospatial software with distributed computing systems and attempting to do for distributed key/value stores what PostGIS does for PostgreSQL.

GreyCat

GreyCat (https://greycat.ai) is a temporal many-world graph database. Data are organized in nodes with relationships between them, and these data can have multiple values through time. After managing the time, GreyCat allows you to run simulations over the temporal graph. Inspired from the many-world interpretation in physics, it can fork the current database in order to simulate "what-if" scenarios.

For example, this technique allows to simulate for example what will happen if in a hypothetical action was taken without corrupting the current state of the graph. Greycat supports many datatypes:

numbers, strings, date/time, maps, arrays, and points, polygons, and circles as geospatial data. Greycat also provides math primitives and tensor operations through a custom API to implement and speed up ML algorithms.

Other data management solutions of relevance

MLDB

MLDB (https://mldb.ai) is an open-source database designed for machine learning. Users can execute procedures and functions through REST endpoints to load or save datasets, and train machine learning models or apply them. Datasets are schema-less, append-only named sets of *data points*, which are contained in *cells*, which sit at the intersection of *rows* and *columns*. There are several types of datasets available, based on MongoDB, SQLite, CSV format, and can store numbers, strings, date/time and binary data. Data points are composed of a *value* and a *timestamp*. Each data point can thus be represented as a (row, column, timestamp, value) tuple, and datasets can be thought of as sparse 3-dimensional matrices. The *timestamp* field allow MLDB to keep different state of the same, as long as its modification times. Its last community-edition release dates from 2017 so we can assume that the database is no longer maintained.

Synthesis

This section summarizes the strengths and weaknesses of existing spatio-temporal data management solutions that can be considered to support real-time machine learning. They key criterions to consider in order to handle real-time machine learning in the context of mission-critical systems, like AIDERS, relates to:

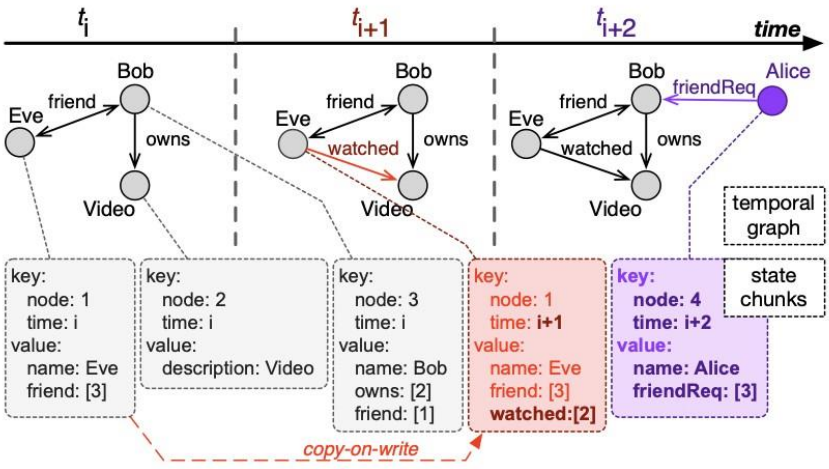
1. Proposing a **flexible data model** to store and organize the information in memory. This includes the support for complex data structures with relationships, but also the support for flexibility by accommodating a wide variety of data types and structural organizations, which can typically change from one mission to another;
2. Proposing a native support for the **temporal dimension** of events to reason upon latest periods of time. This aims to cope with the volatile nature of information to be processed by first responders and thus the necessity to reason upon the latest changes observed in the environment without including outdated information that may mislead the command center.
3. Proposing a native support for the **spatial dimension** of events to process localized events observed in the field in order to report with the maximum accuracy upon the evolution of a critical situation in a given area of interest;
4. Proposing a native support for the **BLOB objects** that cover the variety of multimedia streams that can be captured by the drones and their onboard sensors;
5. Proposing a symbiotic integration with **machine learning algorithms** in order to support instantaneous reaction to the occurrence of new events. This criterion aims to evaluate a tight integration of a data management solution with machine learning algorithms consuming the stored data and potentially producing data insights to be stored persistently;
6. Buidling upon an **active community** of developers to ensure the long-term support of the developed tools and methodologies.

The following table therefore provides a detailed evaluation of the above criterions for all the data management solutions introduced in the previous section:

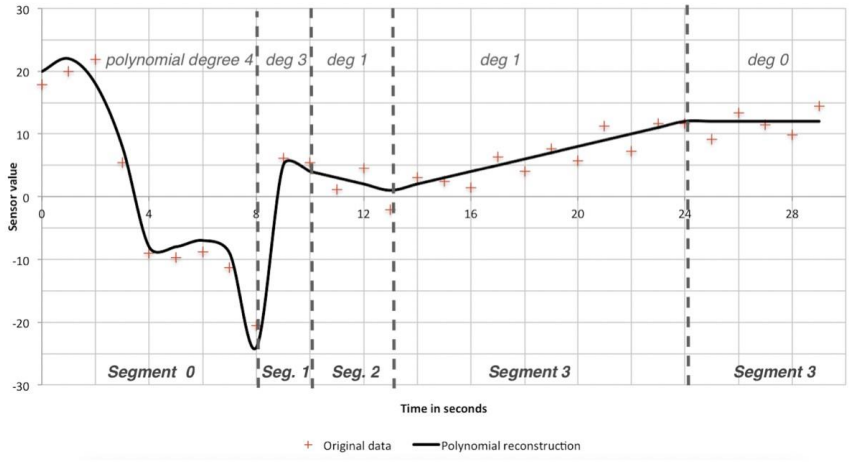
Solution	Model	Time	Spatial	BLOB	ML	Status
PostGIS	relational	no	advanced	yes	no	active
SpatiaLite	relational	no	advanced	yes	no	active
ElasticSearch	document	no	advanced	yes	no	active
GeoCouch	document	no	advanced	yes	no	maintenance
ArangoDB	document/ graph	no	basic	yes	no	active
Redis	key/value	no	basic	yes	no	active
Tile38	key/value	no	basic	yes	no	active
InfluxDB	key/value	yes	no	no	indirectly	active
OpenTSDB	key/value	yes	no	no	no	active
Druid	key/value	yes	basic	yes	no	active
MLDB	key/value	no	no	yes	yes	stalled
TimescaleDB	relational	yes	advanced	yes	no	active
MobilityDB	relational	yes	advanced	yes	no	active
GeoWave	key/value	yes	advanced	yes	no	active
GreyCat	**graph**	**yes**	**advanced**	**yes**	**yes**	**active**

Given the above evaluation of all solutions, we recommend to consider the GreyCat data management solution to support the deployment of real-time machine learning algorithms in the context of mission-critical applications, like first-responders assistance. This recommendation is motivated by the facts that GreyCat provides a flexible data model that supports complex and flexible information structures, organized as spatiotemporal graphs, that can embed machine learning algorithms to continuously evaluate and update insights according to the evolution of the situation.

In GreyCat, data are natively structured and stored as *spatiotemporal graphs*, which are *nodes* with relationships between them (*edges*), whose enclosed *attributes* can have multiple values through time, so called *streams*. In addition to attributes, nodes and edges can be indexed along time and spatial dimensions. As illustrated below, GreyCat graphs can therefore change along with time and these changes will be recorded as *state chunks*, which enable a finer granularity of changes storage, thus reducing the storage overhead of spatiotemporal graphs in the context of constraint environments.



In addition to the storage of discrete information as state chunks, GreyCat can also segment continuous data (or *signals*) by time range and capture them as polynomial regressions (cf. figure below). This method allows GreyCat to further increase the storage of data streams, avoiding to store many similar recordings and saving disk space, while minimizing accuracy losses.



This method significantly reduces the read and write access time of the data samples acquired by the sensors (*e.g.*, altitude, longitude, latitude) that will be continuously collected by drones while in operation. The following figures highlight the performances of GreyCat for writing and reading data streams, no matter the underlying physical laws captured by the streams.

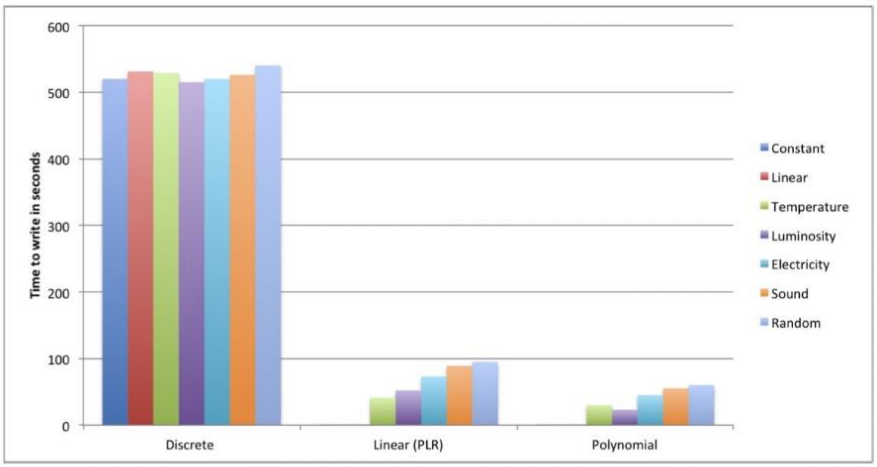


Figure 6.5: Time to write on Leveldb for the 3 methods

Comparison of discrete vs. regression-based write performance in GreyCat.

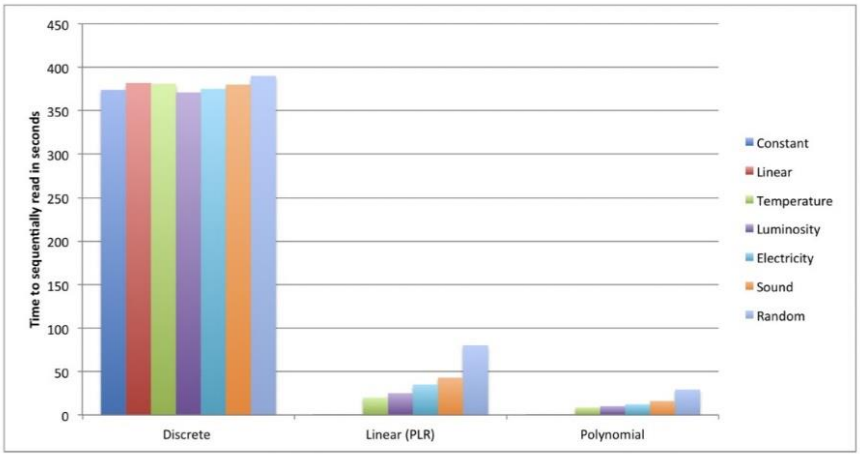


Figure 6.6: Time to read sequentially from Leveldb for the 3 methods

Comparison of discrete vs. regression-based sequential read performance in GreyCat.

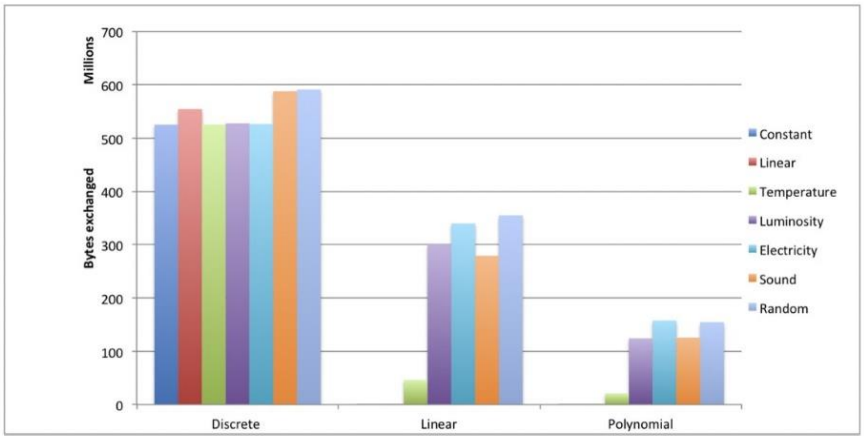
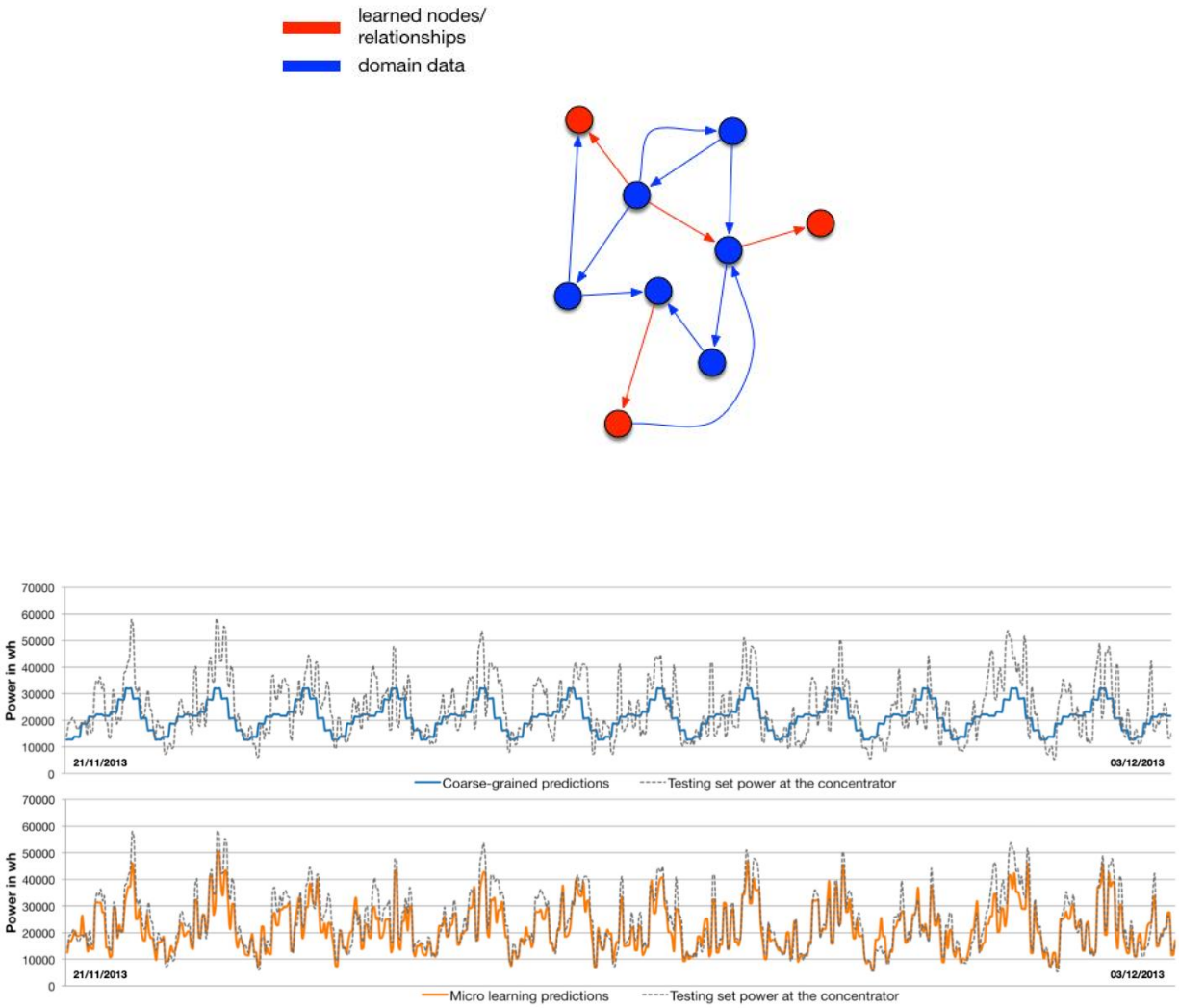


Figure 6.8: Bytes exchanged between NoSQL storage and models during 5 000 000 write operations

Comparison of storage capacity savings achieved by regression-based data representation.

Based on this capacity of adjusting the node-scale data encoding strategy, GreyCat can also embed specific ML algorithms (*e.g.*, Gaussian Mixture Models), known as *micro learning*, to continuously compute advanced AI models atop of raw measurement streams reported by sensors with a finer granularity than state-of-the-art algorithms that either work at a coarse-grained level (cf. figures below). ML algorithms are connected to the input data streams and can add/remote or update nodes, edges and attributes to reflect the outcome of integrated algorithms as nodes that can be queried by the firs-responders' applications. Thus, hiding the complexity of triggering complex ML tasks. Furthermore, by distributing the learning at the scale of node, the accuracy of predictions can be greatly improved compared to state-of-the-art approaches.



Comparison of micro vs. Macro learning prediction strategies.

Finally, while GreyCat supports the storage of BLOB (like streams), it is recommended to store and index the associated metadata as part of the graph and store the multimedia payload by using a third-party solution, like MinIO (https://min.io). Therefore, as part of the AIDERS AI toolkit, we intend to combine both GreyCat with MinIO to ensure the storage of data streams and implement ML algorithms for first-responders.

4. Conclusion

This deliverable provided fundamental insights in collecting and utilizing multi-sensor data for the purposes of developing AI algorithms for the AIDERS project. An initial mapping of user requirements to algorithms enabled the identification of data management solutions that can be utilized to support real-time machine learning algorithms. We provided an overview of existing solutions that can be adapted or expanded to cover the needs of the AIDERS project.

In addition, we identified the data required for training machine learning algorithms, we have presented publicly available datasets that can be utilized for the purposes of the project and a methodology for developing a dataset. Furthermore, we provided the state-of-the art approaches in data pre-processing and popular approaches and tools for data cleansing, data reduction and data wrangling have been presented. Finally, we have provided an overview of other projects and systems developed towards the handling multi-sensor data in emergency response, and in fires, earthquakes and search and rescue in particular.

Using the insights presented in this deliverable, the DG ECHO AIDERS consortium will proceed to the design and implementation of machine learning algorithms to solve the problems faced by first responders, assist them in capacity building and improve emergency response.