البرنامج الوطني للذكاء الاصطناعي
NATIONAL PROGRAM FOR ARTIFICIAL INTELLIGENCE

United Arab Emirates

BEST PRACTICES FOR DATA MANAGEMENT IN ARTIFICIAL INTELLIGENCE APPLICATIONS

OCTOBER 2023

BEST PRACTICES FOR DATA MANAGEMENT IN ARTIFICIAL INTELLIGENCE APPLICATIONS

NATIONAL PROGRAM FOR ARTIFICIAL INTELLIGENCE

AUTHORS

Dr. Abdelrahman AlMahmoud
Ahmad AlRubaie
Dr. Ahmed AlDhanhani
Dr. Anis Ouali
Dr. Di Wang
Dr. Dymitr Ruta
Prof. Ernesto Damiani
Dr. Kin Poon
Maryam AlShehhi
Maurizio Colombo
Musab AlHammadi
Nathan Eden
Dr. Rasool Asal
Dr. Siddhartha Shakya

FORWARD

In recent months, we have witnessed a rapid growth in the development of artificial intelligence technologies, which are now deeply ingrained in numerous aspects of our lives. These advancements have the potential to transform industries, streamline processes, and improve the overall quality of life. However, the deployment of AI models is not without its challenges, and effective data management is at the forefront of these challenges.

This guide offers a comprehensive overview of the data managment process for data owners and organizations seeking to develope and deploy AI models. It delves into the intricacies of data assets, their role in the life cycle of AI models, and the best practices for their collection, storage, processing, and usage. Furthermore, it addresses the common pitfalls in using unsuitable data, highlighting the importance of ensuring high data quality.

As we continue to navigate the ever-evolving world of AI, it is essential that we remain mindful of the importance of ethical considerations of using data in AI systems. This guide serves as a critical resource to ensure that AI models are designed, implemented, and operationalized in a responsible and sustainable manner.

It is my hope that this guide will empower organizations and individuals to make informed decisions when implementing AI systems, ultimately contributing to the responsible and effective deployment of this transformative technology.

Omar Sultan AlOlama

Minister of State for Artificial Intelligence, Digital Economy and Remote Work Applications

1 INTRODUCTION

Artificial intelligence (AI) systems have become a reality and affect our lives in many important ways. Data are at the core of artificial intelligence and machine learning (AI-ML) models; they are the main resource which enables AI-ML models to learn and evolve, allowing them to solve classification, prediction and anomaly detection tasks. Collecting, preparing and managing the data assets needed to train and deploy effective AI-ML models is a challenge. It is important that performance is achieved while making sure that AI-ML models abide by data protection regulations. Data usage is a fundamental aspect to consider when AI-ML systems are designed, implemented and operationalised. Being aware of data management problems allows organisations and individuals to understand and follow how their data are collected, transmitted, stored, processed and exploited by AI-ML-powered systems. Anyone who plans to implement such systems should be fully aware of all challenges related to data management in order to ensure consistent and sustainable AI-ML deployment.

This document is for data owners and organisations wishing to adopt and deploy AI-ML models using these data. It discusses how data assets are used in the AI-ML models' life cycle and highlights the best practices for using them.

Throughout the document, the term *data* will be used to define the representation of facts, measurements and various other types of information, while the term *data asset* will designate the incarnation of data in a format that is suitable for management, storage and processing within *information and communication technology* (ICT) systems. The document presents data management practices starting from the *life cycle* of AI-ML models that rely on such data. Indeed, one of the most common pitfalls is to use data that are not suitable for a given stage of AI-ML model development, or data that are not suitable at all, and then expect reasonable performance. It is important to realise that increasing data quantity does not usually mitigate the problem of having low-quality data. As an introduction to the overall data landscape, the next section provides a short introduction to data types. Then the chapter describes a *reference life cycle* for AI-ML models, which will be used throughout the document to map data assets to AI-ML models' development stages, explain the role of the data assets and describe the best practices for their collection and management.

1.1 TYPES OF DATA

Data can be classified into two main categories, based on the way they are organised. The first category is *structured data*, which is data formatted in such a way that each data item has the same standard, predefined structure, usually described via metadata.

Structured data are easy to index, search and manipulate due to their predefined structure. Typical examples of structured data are *relational databases*, where data entities (e.g., the employees of a company) are represented by tables whose rows correspond each to a data item (the individual employee), while columns represent *attributes* (e.g., each employee's name, age, gender). Attributes belong to *elementary data types*[1] (e.g., integer, date, string and timestamps). The structure of each table is described by an essential metadata item, called the database's *schema*.

The second data category is *unstructured data*, where each data item can have a different format and size, without a clear predefined structure. Unstructured data are usually more difficult to deal with, because they require *preprocessing* to extract key information from them. Examples of unstructured data range from text documents and web pages to audio recordings. Advanced techniques such as *natural language processing* (NLP) allow for the extraction of key information from unstructured text.

For both data categories, we can distinguish between *numerical* and *categorical* attributes. This distinction is of capital importance for data representation and for later processing.

1. Besides elementary data types like strings and integers, structured data items can also belong to complex types, like the nested records used for log file entries.

1.2 DATA QUALITY

A well-known saying in the realm of computing is 'garbage in, garbage out', meaning that if you feed any system or algorithm with low-quality data, the output will also be of low quality. AI-ML systems are no exception: they are not able to magically extract valuable insight from low-quality data, nor can they perform well without large quantities of high-quality data. Let us now list and describe some types of low-quality data that, when fed to AI-ML models, usually result in poor-quality output.

- *Redundant, Outdated or Trivial (ROT) data.* It is very easy and common to duplicate data or have the same (or similar) data collected by multiple entities. While the size of data is an important factor, redundant data are not useful for AI applications. Having diverse and comprehensive data is much more effective for AI-ML applications than a large set of redundant data.
- *Dark Data*. This term refers to data collected and stored by an organization but never used. Often, data ends up unused because it is collected without a clear purpose, clobbering up the ICT infrastructure of organizations that do not have the proper skill level to use the data or cannot make the resource and time investment to tap into it. Other times, dark data originate from data collection procedures carried out carelessly, without upholding any collection best practice or standard. Dark data do not just lay dormant: they can become harmful to organizations. Indeed, storing dark data and maintaining the platforms that host them may have a non-negligible cost. While there is no clear answer on how to deal with dark data, simply deleting them can often save organizations time and money.

Generally speaking, it is possible to consider data quality as a series of dimensions describing the quality of the information fed to (and produced by) an AI-ML model; that is, a measure of the success of the system utilizing this information. Therefore, both input and output data can be considered products with a certain quality. We now list a few aspects of data quality that should be pursued when selecting raw data. The most common ones are *completeness, consistency, fitness for use,* relevance and timeliness. For the purposes of this document, two additional aspects of data quality are significant: *uncertainty* and *vagueness*, which can be seen as two different aspects of *indeterminacy*, i.e. how much it is known about the consequences of exchanging a data item. Uncertainty is mainly related to the

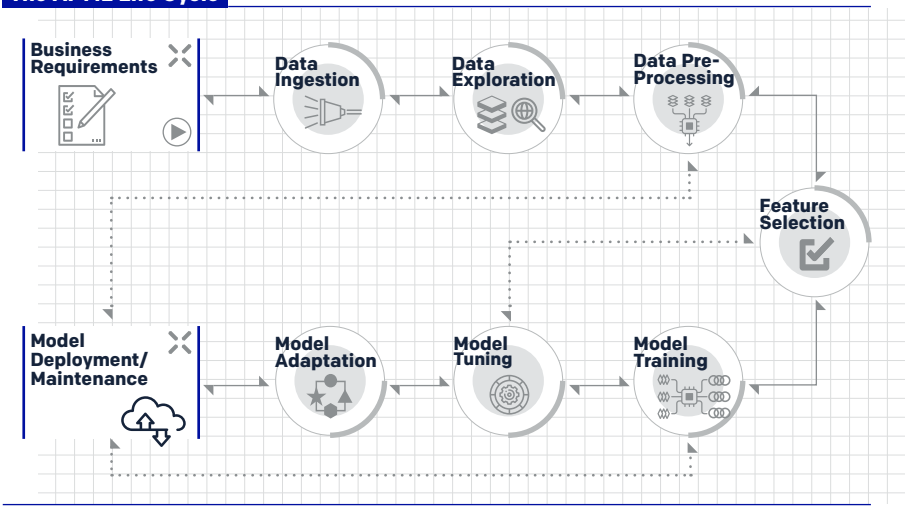
error or imprecision associated with raw data, while vagueness is an inherent issue of categorical values (for example, in the sentence "long text", how many characters does "long" mean?). In the case of information expressed as text, one can distinguish between uncertainty due to the writing style (imprecision, vagueness, polysemy) and uncertainty due to the text content (for example "Mary gave Sally her book").

1.3 THE AI-ML LIFE CYCLE

We are now ready to discuss the different data assets generated and used by AI-ML applications. Our discussion will be driven by a basic notion of systems engineering: the *development life cycle*, which is used to designate the process of planning, developing, testing and deploying an information system. The *AI-ML applications life cycle* (in short, the *AI-ML life cycle*) defines the phases that organisations follow to take advantage of supervised machine learning (*ML*) models to derive practical business value. Most of these stages use and/or generate specific data assets, whose careful management is the goal of this document. The AI-ML life cycle covers only a part of the AI applications landscape; other types of AI models will be discussed in Section 9. Figure1 shows the different stages of the AI-ML life cycle.

In this chapter, we provide a short definition of each stage and outline the individual steps it involves ('Phase in a Nutshell'). For the sake of clarity, we also present an instance of each stage within the framework of a running example concerning a sample AI-ML application. We start by providing a general description of the running example. Then, for each phase of the AI-ML life cycle, we will provide a short description of the phase in the context of our running example (under the title 'Phase in Our Running Example'). This description should help the reader to understand which data assets are concretely needed at each phase and how they are used.

Figure 1:
The AI-ML Life Cycle



1.3.1 A Running Example for the AI-ML Life Cycle

The ACME oil field services company wants to prevent the failure of its mechanical equipment. ACME uses a high-speed rotating machine (internally called a type-A rotatory) to mix components with water to make a frothy mix used to produce shale gas. Rotating machines of type *A* run for weeks without interruptions, leading to frequent breakdowns. The need to find a solution to predict failure of the equipment is dire, since it is a critical component for oil and gas exploration. ACME intends to develop a AI-ML model called a *binary predictor*[2] that will run continuously and assign to each ACME rotating machine a label regarding the next failure (either *IMMINENT* or *NOT-IMMINENT*). Machines labelled IMMINENT are to be immediately stopped for maintenance in the hope that their downtime due to maintenance will be shorter than the downtime that would result from breakdown. The performance of the AI predictor will be validated by comparing the total downtime with the AI predictor in operation to downtime without the predictor, obtained from historical data. Any change (positive or negative) observed when using the AI predictor will indicate the performance gain or loss.

1.3.2 Business Goal Definition

Before carrying out any development or deployment of AI applications, it is important that all stakeholders fully understand the business context of the AI application and the data required to achieve the AI application's business goals, as well as the business metrics to be used to assess the degree to which these goals have been achieved.

Business Goal Definition Phase in a Nutshell: Identify the business purpose of the AI-ML model. Link the purpose with the question to be answered by the AI model. Identify the model type based on the question.

Business Goal Definition in Our Running Example: Using a standard technique for management decisions like the *goal-question-metrics* approach, ACME management can specify the business objectives of the planned AI application as follows. Goal: Decrease the downtime of rotating machines of type A. Question: Is predictive maintenance of type-A equipment before its (estimated) failure time more cost- and downtime-effective than reactive maintenance after breakdown? *Metrics*: The total cost of operation for type-A equipment.

1.3.3 Data Ingestion

Data ingestion is the AI life cycle stage where data are obtained from multiple sources to compose *data records,* for immediate use or for storage in order to be accessed and used later. Data ingestion lies at

2. Binary predictors are techniques that process input data and output one of two possible choices (yes or no, 0 or 1, IMMINENT or NOT IMMINENT).

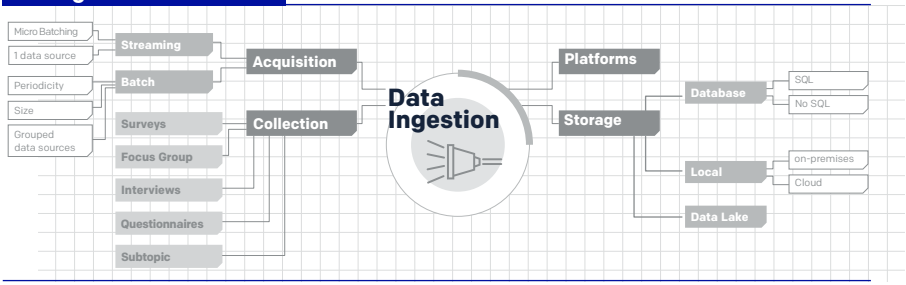
the basis of all AI applications. Data can be ingested directly from their sources as they are generated (*streaming*) or via periodically importing blocks of data called *batches*. Indeed, stream and batch data ingestion can be active in the same AI application simultaneously. For example, the licence plates of cars entering a parking lot can be ingested one by one to check them against a stolen cars database, while batches of the same data are collected periodically for computing the parking lot's average occupancy. An important data management procedure performed at ingestion time is *data filtering* or *access control*. This procedure selects data to be ingested, depending on their *privacy status* (personal/non-personal data, consent given for a given purpose, etc.). We will deal with these issues in detail in Sect. 1.7. For now, we only remark that it is good practice at ingestion time to apply some *anonymity preservation* techniques, taking into account the achievable trade-off between the impact of potential disclosure and the accuracy of the analysis to be computed on the data[3].

Data Collection/Ingestion Phase in a Nutshell: Identify the input data to be collected and the corresponding annotation metadata. Organise ingestion according to the AI application requirements, importing data in a stream, batch or hybrid fashion.

Data Collection/Ingestion Definition in Our Running Example: In the fault prediction application for rotating machines, a stream of sensor data must be ingested about the operation of each rotatory (serial number, working conditions [round/min], input power [kw], input mass [kg], output). Batch ingestion is also needed (usually via a separate database query) for the corresponding context (meta) data: equipment brand, model, serial number, procurement info (supplier, date of construction, date of delivery), installation data (installer, date of installation, installation details). ACME chooses an ingestion mechanism that follows a Π architecture.

3. For multimedia data sources, access control rather than being based on filtering may follow a *digital rights management* approach where some *proof-of-hold* are negotiated with the data owner's license servers before ingesting the data.

Figure 2:
The Ingestion Mechanism



1.4 DATA EXPLORATION

Data exploration is the stage where insights start to be taken from ingested data. While this stage may be skipped in some AI applications where data are well understood, it is often a crucial (and very time-consuming phase) of the AI-ML life cycle. At this stage, it is critical to distinguish between numerical and categorical data. Numerical data lends itself to plotting and allows for computing descriptive statistics and verifying if data fit simple parametric distributions like the Gaussian one. Missing data values can also be detected and handled at the exploration stage.

Data Validation/Exploration in a Nutshell: It is always advisable to plot data after ingestion, to obtain a multidimensional view of all the components of each data vector. Also, it is useful to verify if data fit a known statistics distribution, either by component (*monovariate distribution*) or as vectors (*multivariate distribution*), and estimate the corresponding statistic parameters.

Data Validation/Exploration in Our Running Example: ACME data scientists will periodically plot sensed data about multiple pieces of equipment (e.g., the rounds-per-minute and power consumption variables) and fit the data to a bivariate statistical distribution (e.g., a Gaussian or power-log distribution). If the statistical tests confirm data belong to a distribution, they will display the distribution's parameters, for instance the standard deviation σ, and highlight 'three-sigma' outliers (e.g., the machines whose rotation speed values lie outside an interval of three sigmas around the average).

Figure 3:
The Data Exploration Procedure

1.4.1 Data Pre-Processing

Data preprocessing can be the most critical stage of the life cycle. At this stage, techniques are employed to clean, integrate and transform the data, resulting in an improved data quality that will save time during the analytic models' training phase and promote better quality of results. Data *cleaning* is used to correct inconsistencies, remove noise and anonymise data. Data *integration* puts together data coming from multiple sources, while data *transformation* prepares the data for feeding an AI-ML model, typically by encoding it in a numerical format. A typical encoding is *one-hot encoding* used to represent categorical variables as binary vectors. This encoding first requires categorical values

to be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the position of the integer, which is marked with a 1. Figure4 below shows one-hot encoding of categorical data expressing colours.

Figure 4:
One-Hot Encoding

COLOUR
RED
RED
YELLOW
GREEN
YELLOW

»

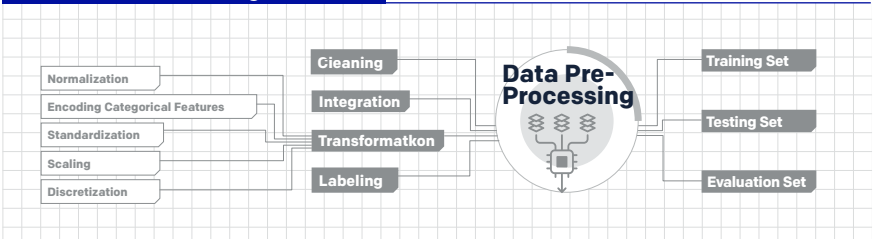
RED	YELLOW	GREEN
1	0	0
1	0	0
0	1	0
0	1	1

Once converted to numbers, data can be subject to further types of transformation: *rescaling, standardisation, normalisation and labelling*. Rescaling expresses numerical data in a suitable representation unit (e.g., from tons to kilograms). *Standardisation* puts data in a standard format, and normalisation maps data to a compact representation interval (e.g., the interval [0, 1], by dividing all values by the maximum). *Labelling* (done by human experts or by another AI application) associates each data item to a category or a prediction. At the end of this process, a numerical data set is obtained, which will be the basis for training, testing and evaluating the AI model.

Data pre-processing in a Nutshell: Convert ingested data to a metric (numerical) format, integrate data from different sources, handle missing/null values by interpolation, increase density to reduce data sparsity, de-noise, filter out outliers, change representation interval. Anonymize the data.

Data Preprocessing in Our Running Example: After having ingested the sensor data about the rotating machines, the ACME AI-ML application interpolates any missing value about equipment rotation speed and power consumption to achieve a uniform samples/time unit rate. The application integrates sensed data about rotation speed and power with data about external temperature and atmospheric pressure at the same time obtained from an open data service; then, it normalizes the data vectors, and adds to each data vector labels *IMMINENT NOTIMMINENT* representing the expected time to next failure. Also, it deletes the human operator code from the data to make sure they do not reference personal information.

Figure 5:
The Data Pre Processing Procedure



1.5 FEATURE SELECTION

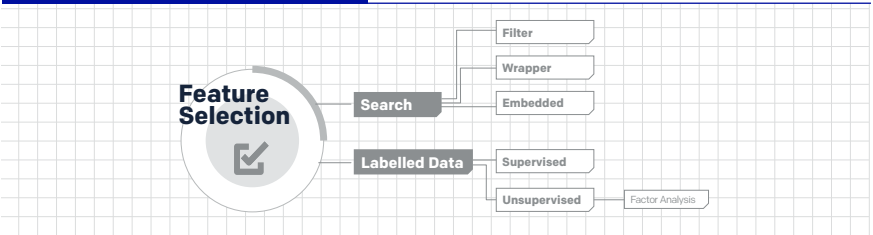
Feature selection is the stage of the life cycle where the number of components of the data vectors (also called *features or dimensions*) is reduced, by identifying the components that are believed to be the most meaningful for the AI model. The result of this phase is a reduced data set, as each data vector has fewer components than before. Besides the computational cost reduction, feature selection can help in obtaining more accurate models. Additionally, models built on top of lower dimensional data are more understandable and explainable. This stage can also be embedded in the model-building phase, to be discussed in the next section.

Feature selection in a Nutshell: Identify the dimensions of the data set that account for a global parameter (e.g., the overall variance of the labels). Project data set along these dimensions, discarding the others.

Feature Selection in Our Running Example: In the predictive maintenance application, the ACME data scientists project the vectors of the data set on the subset of dimensions that maximises input variance[4]. As inputs are mostly numerical data (like the engines' power consumption and rotation speed), ACME data scientists use the principal component analysis (PCA) method. If inputs had been categorical, *multiple correspondence* analysis could have been used to represent categorical data as points in a low-dimensional vector space.

4. In statistics, (sample) variance is the average of the squared differences between sample values and the sample's average. It measures how far a set of numbers is spread out from their average value.

Figure 6:
The Feature Extraction Procedure



1.5.1 ML Model Selection

This stage performs the selection of the best AI-ML model or algorithm for analysing the ingested and preprocessed data. Finding the 'right' AI-ML model to solve a business problem or achieve a business goal is a challenge, often subject to trial and error. Based on the business goal and the type of available data, different types of AI techniques can

be used. It is important to remark that model selection may trigger a transformation of the input data, as different AI models require different numerical encoding of the input data vectors. Two major categories are *supervised learning* and *unsupervised learning* models, the latter including *clustering* and *reinforcement learning*. Supervised techniques deal with labelled data: the AI-ML model is used to learn the mapping between input examples and the target outputs. Supervised models can be designed as *classifiers*, whose aim is to predict a class label, and *regressors*, whose aim is to predict a numerical value function of the inputs (e.g., a counter). Unsupervised techniques extract relations from unlabelled training data, with the aim of organising them into groups (*clusters*, highlighting associations among data, summarising data distribution and reducing data dimensionality [this last already mentioned as a goal of data preparation]).

Reinforcement learning is typically less data dependent: it maps situations with actions, learning behaviours that will maximise a reward.

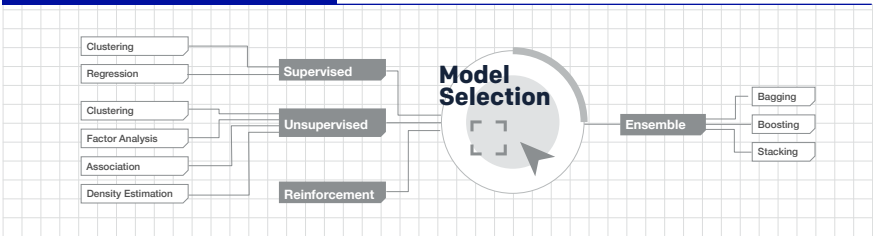
AI-ML models of different types can be composed using composition methods (e.g., by taking the majority of their outputs)[5].

AI Model Selection in a Nutshell: Choose the type of AI model most suitable for the application. Encode the data input vectors to match the model's preferred input format.

AI Model Selection in Our Running Example: For associating an *IMMINENT or NOT-IMMINENT* label to each data vector about the type-A rotating machines, ACME data scientists choose a multidimensional, supervised AI model with memory, as they realise that fault events depend on the history of each piece of equipment and not only on the current values of the input. They choose a two-dimensional long short-term recurrent neural network (2D RNN). They compute one-hot encoding of the categorical inputs and map the input data vectors (dimension n) into 2D tensors (i.e., bi-dimensional matrices with dimensions h, k and $h + k = n$).

5. Model composition techniques, also called *ensemble techniques*, include *bagging, bootstrapping, boosting* and *stacking*. Detailed discussion of ensemble techniques is beyond the scope of this document.

Figure 7:
The Model Selection Procedure



1.5.2 Model Training

When the selected AI analytic is an ML model, the latter must go through a *training phase*, where internal model parameters like *weights* and *bias* are learned from data. The training phase will feed the ML model with batches of input vectors and will use a *learning function* to adapt the model's internal parameters (weights and bias) based on a linear or quadratic measure of the difference between the model's output and the labels. Often, the available data set is partitioned at this stage into a *training set*, used for setting the model's parameters, and a *test set*, where error is only recorded in order to assess the model's performance outside the training set. *Cross-validation* schemes randomly partition the data set multiple times into a training and a test portion of fixed sizes (e.g., 80% and 20% of the available data) and then repeat training and validation phases on each partition.

AI Model Training in a Nutshell: Select and apply a training algorithm to modify the chosen model according to training data. Validate the model training on test set according to a cross-validation strategy.

AI Model Training in Our Running Example: Train the 2D RNN model for type-A equipment failure prediction via a small batch gradient descent algorithm with L2 loss function on the training set. Use the 80-20 cross-validation strategy.

1.5.3 Model Tuning

Certain mathematical parameters define the high-level behaviour of ML models during training, such as the learning function or modality mentioned above. It is important to know that these parameters, often called hyperparameters, cannot be learned from input data. They need to be set up manually, although they can sometimes be tuned automatically by searching the model parameters' space, in practice by repeatedly training the model, each time with a different value of hyperparameters. This procedure is called *hyperparameter optimisation*. It is often performed using classic optimisation techniques like *grid search*, but *random search* and Bayesian *optimisation* can also be used.

For the purposes of this document, it is only important to remark that the model tuning stage uses a special data asset (often called a *validation set*), which is distinct from the training and test sets we described in the previous stages. Also, it is useful to know that a final evaluation phase (after tuning) is sometimes carried out to estimate how the tuned model would behave in extreme conditions, for example, when fed with wrong/unsafe data sets. The extreme data used for the latter procedure is called *held-out data*.

AI Model Tuning in a Nutshell: Apply model adaptation to the hyperparameters of the trained AI model using a validation data set, according to deployment condition.

AI Model Tuning in Our Running Example: ACME data scientists run the 2D RNN model they trained for fault prediction on an additional validation data set and choose the best values h and k for the RNN's tensor dimensions. Then they estimate how the tuned model would behave in extreme conditions by running the model on some held-out data corresponding to extreme rotation speed values.

1.5.4 Transfer Learning

The transfer learning (TL) phase, once relatively rare, has become very frequent as the market of AI services has expanded. It happens when the user organisation, rather than training a model from scratch, sources a pretrained and pretuned AI-ML model, and uses it as starting point for further training to achieve faster and better convergence.

TL in a Nutshell: Source a pretrained model in the same domain and apply additional training to improve in-production accuracy.

TL in Our Running Example: ACME data scientists look for the opportunities of sourcing a predictor for the expected time to next failure from the manufacturer of type-A rotating machines. They get a 2D RNN model, which was trained by the equipment manufacturer on lab data. They know that RNN are not usually transfer learned[6], but they decide to apply an RNN-specific TL technique (e.g., Stephen Merity's TL) to their failure time predictor.

6. Not all ML models are equally transferable.

1.6 MODEL DEPLOYMENT

An ML model will bring knowledge to an organisation only when its predictions become available to the users. Deployment is the process of taking a pretrained ML model and making it available to users.

Model Deployment in a Nutshell: Generate an in-production incarnation of the model as software, firmware or hardware. Deploy the model incarnation to edge or cloud, connecting in-production data flows.

Model Deployment in Our Running Example: ACME management decides to compile the failure time prediction model as firmware on the rotating machines' controller cards. This way they can upgrade the centrifuges by installing an enhanced controller and connecting it to the local data streams.

1.6.1 Model Maintenance

Similar to software systems, ML models also require continuous maintenance. After deployment, AI models need to be continuously monitored and maintained to handle *concept changes* and *concept drifts*. A change of concept happens when the meaning of an input (or of an output label) changes for the model (e.g., due to modified regulations). A concept drift occurs when the change is not drastic but, rather, emerges slowly. Drift is often due to sensor *encrustment*, or the slow evolution over time in sensor resolution (i.e., the smallest detectable difference between two values), or overall representation interval. A popular strategy to handle model maintenance is *window-based relearning*, which relies on recent data points to build an ML model. Another useful technique for AI model maintenance is *backtesting*. In most cases, the user organisation knows what happened in the aftermath of the AI model adoption and can compare model prediction to reality. This highlights concept changes: if an underlying concept switches, organisations see a decrease in performance.

Model Maintenance in a Nutshell: Monitor the ML inference results of the deployed AI model to detect possible concept changes or drifts. Retrain the model when needed.

Model Maintenance in Our Running Example: After ACME has installed its ML-based maintenance models, it revises the label *IMMINENT* to *IN THE NEXT FIVE MINUTES*. Also, the rotation speed sensor on the machine board encrusts every year: sensors older than a year can no longer measure rotation speeds higher than 1000 rpm. ACME data scientists advise immediate retraining to handle concept change and a yearly retraining to handle sensor encrustment.

1.6.2 Business Understanding

Building an AI model is often expensive and always time-consuming. It poses several business risks, including failing to have a meaningful impact on the user organisation as well as missing in-production deadlines after completion. Business understanding is the stage at which companies that deploy AI models gain insight on the impact of AI on their business and try to maximise the probability of success.

Business Understanding in a Nutshell: Assess the value proposition of the deployed AI model. Estimate (before deployment) and verify (after deployment) its business impact.

Business Understanding in Our Running Example: ACME management measures, in a six-month verification procedure, the cost of operation for the rotating machines that include the AI controller as well as the ones that do not. The ACME Board estimates the related business opportunities in terms of service and product innovation and decides to start a product line.

1.7 REGULATORY ISSUES OF DATA MANAGEMENT

Data management practices and the AI-ML life cycle presented above are tightly connected. AI-ML models require large volumes of information to learn from, even potentially including personal data. As a result, national and international regulatory issues need to be considered when planning the deployment of AI-ML models performing automated decision-making on individuals based on their personal data, without any human intervention. By personal data, we mean any data that can be linked directly or indirectly to a user's identity. This is a critical scenario which was intentionally not covered in our running example, where data about the engines' operators were not used to train the AI-ML system. Still, personal data are present in many AI-ML applications: for instance, an AI-ML model could analyse a user's credit card history to compute the user's credit score. Using personal data in the AI-ML model life cycle is a key regulatory issue worldwide. The European General Directive on Privacy's Article 22 specifies that each person has the right not to be subject to automatic decision-making if it might result in legal action concerning them. In this section, we briefly recall four key principles that AI-ML models are expected to support: purpose limitation, data minimisation, fairness, transparency and the right to information. These properties will be connected to AI-ML systems operation in the remainder of this document.

- *Purpose limitation*: The purpose limitation principle states that personal data cannot be used to train AI-ML models other than the ones the data owners have been informed about. This is a critical property, as some AI-ML systems rely on information that is a side product of the original data collection. For instance, an AI-ML Fintech application can use social media data about users (the number of their social network followers) for computing their credit score. The principle of purpose limitation says this secondary use should not be allowed, unless the users had been informed of this side use when they joined the social network. Of course, there are expectations: secondary data processing is admissible for medical or statistical research.
- *Data minimisation*: The data minimization principle ensures that data collected to train an AI-ML model is adequate and relevant to the model's purpose, without unnecessary redundancy. AI experts have to determine what data and what quantity of it is necessary for the project. As we will see, it is not always possible to predict how and what a model will learn from data. Organizations deploying AI-ML models should continuously

verify they are using a minimum quantity of training data needed for their models to operate.

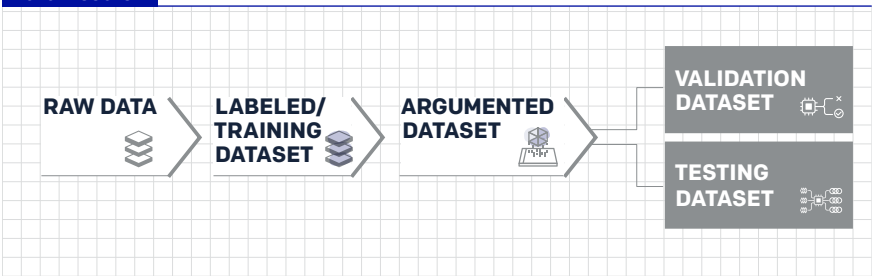
- *Fairness*: The principle of fairness states that the use of the AI-ML system should not result in unfair discrimination against individuals, communities, or groups. The initial data used to train the AI-ML models must be free from bias or characteristics which may cause the models to behave unfairly.
- *Transparency:* This principle states that owners of personal data should know which of their information is used by AI-Ml models. Organizations deploying AI-ML should be prepared to provide a detailed description of what they are doing with personal data to data owners.
- *Right to information*. This principle states that everyone has the right to seek, receive, use, and impart information held by or on behalf of public authorities, or to which public authorities are entitled by law to have access. This applies to AI-ML models that are deployed by authorities to service the community.

Organisations wishing to deploy AI-ML models are expected to find a way to design and use them in a way that is compliant with the above principles, because they will generate value for both service providers and data subjects if done correctly.

1.8 SUMMARY

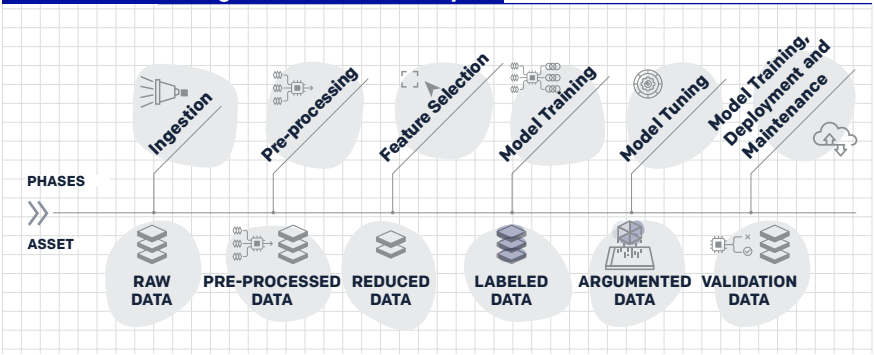
In this chapter, we provided an overview of some AI-ML techniques, focusing on the stages of the AI models' life cycle that require or generate data assets which need attention from the data management point of view. We also reviewed some principles that need to be addressed when managing data assets in the AI-ML life cycle. Figure8 shows these data assets, while Figure9 maps the data assets to the stage of the ML life cycle where they are used or generated.

Figure 8:
Data Assets



In the next chapter, we will examine each stage of the AI-ML life cycle in detail to identify and discuss the data management issues of the corresponding data assets.

Figure 9:
Data Assets and Stages of the AI-ML Life Cycle



2 DATA INGESTION

2.1 SOURCES OF DATA

In recent years, businesses and organisations have been moving more and more processes and services online. These digital processes generate a huge amount of data every second. We can group these data into three distinct categories:

1. **Sensor-generated data** (i.e., *Internet of Things* [IoT] data). These data are generated by equipment without human involvement, such as smart meters, road sensors, street cameras, satellites and many more sources. This type of data source extends our capability of sensing and monitoring the world around us, which, in turn, helps us in automatic decision-making through the analysis of these data. Self-driving cars are a good example of such systems, as they automatically analyse data to navigate their environment and drive autonomously.
2. **User-generated data**. Online user-generated data include informal content posted by individuals on social media platforms such as X, Facebook, Instagram, YouTube, forums, blogs and other mediums of communication, as well as more formal content in newspapers, news agencies, government media outlets, public information websites and so forth. It is worth noting that social media outlets such as X, Instagram and Facebook are also being used by government and news agencies as a formal mode of information sharing. This kind of data provides invaluable insights into people's and communities' views, perception and behaviour, as well as what is happening in the world and the impact on opinions, needs and actions.
3. **Transaction data**. Such data are generated from human actions (e.g., invoices, payment orders, storage records, delivery receipts, ordering a passport online). Analysis of this type of data together with other supporting types provides a more comprehensive situational awareness and highlights the consequences of events that are taking place. For example, the COVID-19 pandemic resulted in lower transactions in terms of physical shopping and increased online shopping. COVID-19 has also resulted in certain goods like masks being more in demand than fuel.

2.2 DATA COLLECTION/INGESTION

Data are key to AI applications and hence data collection is an important stage of the AI application life cycle. There are multiple types of data that can be obtained from multiple sources, as previously described. When data from multiple sources are used, they can be composed as a vector, which is a multidimensional data point.

Data can be received in real time as a data stream or imported in batches at different time points. Receiving data as batches can be in the form of macro-batches (large data chunks) or micro-batches (small data chunks). Data can naturally be imported using one or multiple methods, depending on the data systems used/available, user requirements and the AI application being developed.

In terms of different types of data sources, data ingestion methods vary. For online content and data, harvesters (scripts) are needed to automatically grab the content from websites or social media platforms, either directly from the websites or through application programming interfaces (APIs). Such data-collection methods tend to run periodically to import data in micro- or macro-batches. There are many commercial and open-source products that can help obtain the relevant content from websites. In addition, a number of platforms and websites provide more structured and consistent methods to get the relevant data; examples include RSS feeds for news agency websites and APIs for social media platforms. Thanks to the advances and maturity of sensors and communications technologies, almost all IoT sensors have built-in capabilities to push sensor data in predefined formats to storage systems for easy access and analysis, as presented in Section 2.3. Depending on the application requirements, sensor type and communication capabilities and costs, data can be provided as live stream or in batches. In some rare cases, such as for legacy sensors, or for security reasons, data must be extracted programmatically for the sensors and/or stored locally; hence, specific steps will be needed to extract the data, depending on the use case and type of technology.

For transactional data, the hardware and/or software systems used (e.g., ATMs, point-of-sales terminals, online shopping websites) have systems to automatically log all transactions and related data, such as time, date and type of transaction, into storage databases. Collection/ingestion of this type of data is automated by its devices/systems, and no further effort is needed. There are certain constraints associated with data collection/

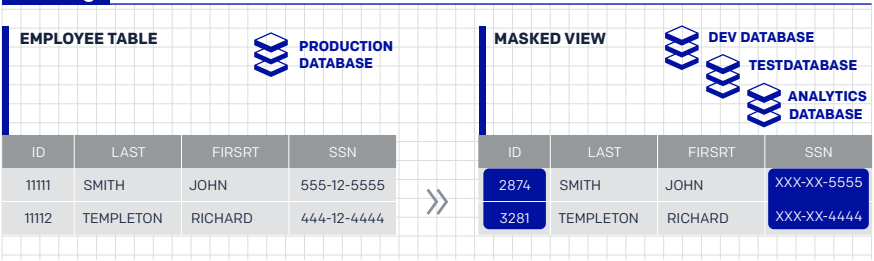
ingestion and usage, drawn from rules, regulations, best practices and potential business and technical matters. Data ownership and its intended use and time of use are often subject to laws and regulations which can also vary by data type. Certain data cannot be used for certain purposes or within or outside a particular time frame. Even if data ownership is clear and the data are not regulated by laws, there are best practices restricting their use (see next section). An entity may have access to health data that can be used for certain applications (e.g., diagnosis) when the data owner (the patient) has agreed or consented. On the other hand, a company might have data regarding the sales of a specific product or their market performance that, though not regulated by law, they choose to use internally but not share outside their organisation. Last but not least, in some cases automatic data collection/ingestion can sometimes be affected by technical issues, such as sensor failure or communications failure, that can impact the quality of data. There are methods and techniques that can help address some or all of these issues (usually done in the data preprocessing stage in the AI-ML life cycle, Section 4).

Additionally, and depending on user requirements, it is possible to collect supplementary data describing the actual data for context. Such data are known as metadata and usually provide additional insight on the data, enabling more effective analysis (Section 2.3).

2.2.1 Data privacy and anonymization

One of the major causes for resistance to data sharing is the risk associated with violating data privacy. Data, in many cases, contains private personal information that should not be made available to the general public, as it might cause harm to the concerned individuals. Simpler approaches such as masking the data might be appealing due to their low complexity. This might include substituting social security numbers with pseudo-random identifiers to hide the private information, as shown in Figure10.

Figure 10:
Masking



However, this approach does not translate well to AI-ML applications, as it destroys many of the mathematical properties that are needed. To facilitate joint privacy-preserving analysis, researchers, academics and private companies have put forth a lot of effort to come up with intelligent and advanced privacy-preserving data-sharing schemes. These data-sharing schemes promise to provide the ability for two or more entities to securely and privately share data to carry out collaborative analytics, without revealing any private information to each other. In recent years, these techniques have matured significantly and now come with strong anonymisation guarantees, while enabling more advanced forms of analytics. Currently, there are three main branches that address data anonymisation which are:

- *Homomorphic encryption*. A class of encryption algorithms that allow for certain mathematical operations to be carried over encrypted data without the need for decryption
- *Secure multi-party computation*. A cryptographic technique that allows two or more entities to jointly perform computations on data without revealing the data to each other
- *Differential privacy*. A model to aggregate data such that no identifying data on any individual is available

To successfully and correctly apply any of the above anonymisation techniques, it is paramount to understand the features of the data set, anonymisation scheme and analytics to be performed – and then make the decisions based on those factors. However, the general principle is that the stronger the anonymisation technique, the less accurate and more time-consuming the analytics become.

2.3 DATA STORAGE

The field of knowledge representation in AI deals with representing the knowledge used and produced by AI models in such a way as to solve complex problems, like communicating with humans using natural spoken or written language. In turn, data representation is the time-honoured computer science field dealing with the different formats for storing and accessing data sets, such as the ones used to feed AI models in the different stages of the AI life cycle. The connection between these two fields is represented by metadata, which provide additional information about the input data to be fed to AI models. The types of data that need to be represented in AI include facts (e.g., trustworthy database records), events (e.g., sensor data) and meta-knowledge (e.g., metadata describing how, when and by whom other data were collected). Managing these data types in AI data storage requires the management of a series of intersecting data representation standards. These include:

1. data structure standards,
2. content value standards,
3. communication standards,
4. syntax standards.

This section describes some of the available standards for data management from an AI data-usage perspective, in an effort to provide a landscape of data representation standards for AI storage.

2.4 AI DATA STORAGE

ML and deep learning rely heavily on the availability of massive data for training purposes. For all AI applications, it is critical to have a standard data infrastructure (AI storage) that is scalable and can apply the FAIR data principles (findable, accessible, interoperable, and reusable) among heterogeneous data sets from various domains. Data storage for AI aims to host or collect quality data of different types and from multiple sources to create an integrated data storage. The purpose of AI data storage is also to set up an environment where AI model designers can easily judge, collect and utilise data. Besides data providers and users, AI storage may also have interfaces for other players, including data distributors that provide mediation between data providers and users.

By making data representations interoperable in the data storage layer, data scientists and other users can focus on the substance of the AI problem they are trying to solve. This allows them to quickly unlock insights and benefits from data analysis. Besides interoperability, a major goal of data representation is protecting data confidentiality and personal privacy. A technology road map for AI data governance and management is therefore critical to any enterprise or organisation wishing to adopt AI. The roadmap needs to express the overall direction of how to manage data generated from the organisation's products and services.

2.4.1 AI Data Formats

The data formats used with AI algorithms are not necessarily unique and can be found in other applications, as well. However, some of the formats are more commonly used than others. The following are the most common:

- The single value representing an integer, a float or a string
- An array of single values, all of them having the same type
- The matrix, a two-dimensional array containing values of the same type (the simplest generalisation of an array)
- A tensor which is obtained by increasing the dimensions (an n-dimensional array)

- An aggregation of several arrays of different types while assigning a name to each one (This produces a data frame, a data structure available in several programming languages and frameworks [e.g., R, Spark, Python, Mathemathica, Matlab]. It is equivalent to a table in a relational database.) Data extracted from databases, or generated by sensors, are examples of these data structures.
    - The simplest data frame, only named columns (A generalisation assigns a name to the rows, as well, and a more generalised version uses a hierarchical index for rows and/or columns.)
- Simple graphs, where between two nodes there is either no arc/edge or a single one, the edge has no direction and there are no loops
- Directed, where the edge has a direction
- Branched (multiple edges between two nodes)

If there are multiple edges, very often each edge has a label that describes the relation type. The analysis of social networks is based on this data structure. In the field of NLP, the simplest data type is the character, represented in one of several formats (ASCII, UNICODE, using a specific encoding, etc). Based on the level of aggregation, we can have any of the following:

- A word which consists of a sequence of characters
- A sentence which consists of a sequence of words, separated by spaces (the words) and punctuation marks (the sentences)
- A paragraph, a sequence of sentences, separated by full stops, exclamation points or question marks
- A section/chapter/document present in a hierarchical organisation (e.g., a book)
- A corpus, a list of documents

The main problem with this format is that it does not have a specific encoding (ASCII, UTF8, ISO8859-1, or Windows-1252) and it is not possible to specify a hierarchical structure. Some alternatives are as follows:

- HTML, which can structure the text in well-defined elements
- XML, a generalisation of the HTML
- Markdown, a lightweight version of the HTML which allows the user to specify a not too complex hierarchical structure

2.4.2 AI File Formats

The data, used by the algorithms, are saved on files with a structure that depends on the data type. The simplest data format is comma-separated values (CSV), a text file in which:

- records are separated by a new line, and
- fields are separated by a comma.

The file can contain, in the first row, the column names. However, this format is not the only standard:

- In certain formats, the fields are separated by a tab rather than a comma.
- The row containing the column names can be removed.
- There is no standard method to represent strings with spaces or commas (or tabs).
- There is no standard method to represent missing values.
- There are multiple methods to represent date/hour/timestamp.
- There is no consensus on whether a CSV file can contain comments or not.

Another problem is that the column's data type is not specified. To obtain this information, it is necessary to analyse the records and to use a heuristic approach to find them, or have them passed from the user. A simple variant format is the attribute-relation file format, or ARFF: it is very similar to a CSV, but it contains a header with the name and the type of each column. Other popular text formats are XML and JSON. The main problem with the text formats is that it is necessary to read the file sequentially to read its context. This is a serious problem if the file is huge and it is used in a big data infrastructure (e.g., Hadoop – refer to Section 2.12).

Some other formats available are as follows:

- Adobe PDF
- Microsoft Word
- OpenXML (proposed by Microsoft) and OpenDocument (proposed by OpenOffice and Sun StarOffice, now IBM): a compressed list of XML files, containing the text, its formatting and the hierarchical organisation

2.5 DATA REPRESENTATION STANDARDS

2.5.1 Basic ISO Working Groups and Standards

To provide a stable base to address the challenges and opportunities of data management in AI and big data scenarios, a comprehensive range of standards and technical reports has been published by the International Organization for Standardization (ISO).

- ISO/IEC JTC 1/SC 32, titled 'Data Management and Interchange' and currently called 'WG2 on Metadata Standards', focuses on three major areas:
  - Specification of generic classes of data, metadata and frameworks for representing the meaning and syntax of data, including metamodels, ontologies, processes, services and behaviour, plus the mappings between them
  - Specification of facilities to manage metadata, including registries and repositories
  - Specification of facilities to enable electronic metadata exchange over the internet, within the cloud, and via other information technology telecommunications avenues
- ISO/IEC JTC 1/SC 42, titled 'Artificial Intelligence', deals with data management in AI pipelines. It has published six relevant standards, among which is the five-part ISO/IEC 20547 series, which provides a big data reference architecture (BDRA) organisations can use to effectively and consistently describe their AI-ML life cycle and its implementation. The BDRA addresses requirements, architecture, security and privacy, use cases and considerations that architects, application providers and decision-makers will want to consider in deploying a big data system. The list of published standards includes the following:
  - ISO/IEC 20546:2019, Information Technology – Big Data – Overview and Vocabulary
  - ISO/IEC TR 20547-1:2020, Information Technology – Big Data Reference Architecture – Part 1: Framework and Application Process
  - ISO/IEC TR 20547-2:2018, Information Technology – Big Data Reference Architecture – Part 2: Use Cases and Derived Requirements
  - ISO/IEC 20547-3:2020, Information technology – Big Data Reference Architecture – Part 3: Reference Architecture

- ISO/IEC TR 20547-5:2018, Information Technology – Big Data Reference Architecture – Part 5: Standards Road Map
- ISO/IEC TR 24028:2020, Information Technology – Artificial Intelligence – Overview of Trustworthiness in Artificial Intelligence

Other ISO standards relevant to AI data representation include the following:

- ISO/IEC 11179:2019, Metadata Registries (MDR) – A framework for registering and managing metadata about data sets
- 11179-2:2019, Part 2: Classifications – Describes the registration of classification schemes and using them to classify registered items in a metadata repository. Any metadata item can be made a classifiable item so it can be classified, including object classes, properties, representations, conceptual domains, value domains, data element concepts and data elements themselves.
- 11179-3:2013, Part 3: Registry Meta Model and Basic Attributes – Specifies the structure of a metadata registry in the form of a conceptual data model, which includes basic attributes that are required to describe metadata items
- 11179-3:2019, Part 3: Registry Meta Model – Core Model – Specifies the structure of a metadata registry in the form of a conceptual data model
- 11179:7:2019, Part 7: Meta Model for Dataset Registration – Provides a specification in which metadata describing data sets, collections of data available for access or download in one or more formats, can be registered
- ISO/IECTR19583, Concepts and Usage of Metadata
- 19583-1, Part 1: Metadata Concepts – Provides the means for understanding the concept of metadata, explains the kind and quality of metadata necessary to describe data and specifies the management of that metadata in an MDR
- 19583-2, Part 2: Metadata Usage – Describes a framework for the provision of guidance on the implementation and use of the registries specified in ISO/IEC 11179, Information Technology – Metadata Registries, and ISO/IEC 19763, Information Technology – Meta Model Framework for Interoperability (MFI)
- ISO/IEC11404:2007, General Purpose Data Types (GPD) – Specifies a collection of data types commonly occurring in programming

languages and software interfaces including both primitive and non-primitive data types, in the sense of being wholly or partly defined in terms of other data types

2.5.2 ISO Work Groups and Activities on Data Governance

ISO/IEC JTC 1/SC 40, titled 'IT Service Management and IT Governance', currently WG1 on Governance Standards, leads the development of standards, tools, frameworks, best practices and related documents on the governance of information technology. Relevant standards potentially beneficial to AI include the following:

- ISO/IEC 38505-1:2017, Part 1: Application of ISO/IEC 38500 to the Governance of Data – Applies to governance of the current and future use of data that is created, collected, stored or controlled by IT systems, affects the management processes and decisions relating to data
- ISO/IEC 38505-2, Part 2: Implications of ISO/IEC 38505-1 for Data Management – Identifies the information that a governing body requires to evaluate and direct the strategies and policies relating to a data-driven business and the capabilities and potential of measurement systems that can be used to monitor data performance and uses.

2.6 REPRESENTATION STANDARDS FOR WEB DATA

Web data are at the core of many AI applications revolving around users' behaviour in cyberspace. Next, we touch upon some of the most well-known representations of the data and metadata designed specifically for web data.

2.6.1 The Dublin Core

This standard emerged to produce a general metadata standard for describing web pages. Originally created in 1995, Dublin Core (DC) included thirteen elements (attributes) that were later extended to fifteen in 1998 and again, as Qualified DC, to eighteen, including audience, provenance, and rights holder. DC was initially based on text and HTML but evolved to include the concept of namespaces for elements (with approved terms for the semantics of element values) coincident with the move to Qualified DC and towards using XML. Later the community realised that relationships among elements were important, and an RDF version was proposed. However, the major volume of DC metadata is still in HTML format and so the benefits of using namespaces – and later relationships – have not been realised. Indeed, this is the major criticism of DC: it lacks referential integrity and functional integrity. The former problem means that it is hard to disambiguate element values in repeating groups.

2.6.2 Data Catalog Vocabulary (DCAT)

The original DCAT was developed at the Digital Enterprise Research Institute, refined by the eGov Interest Group and then finally standardised in 2014 by the Government Linked Data Working Group, leading to a W3C recommendation. It is based on Dublin Core but adopts linked data principles with a schema including links between a data set and a distribution of that data set (i.e., a replicate or version), a data set and a catalogue and also between a data set and an agent (person or organisation).

2.6.3 Common European Research Information Format (CERIF)

CERIF is a European Union Recommendation to Member States. CERIF91 (1987–1990) was quite like the later Dublin Core (late 1990s). CERIF2000 (1997–1999) used full enhanced entity-relationship (EER) modelling with base entities related by linking entities with role and temporal interval (i.e., decorated first-order logic). In this way, it preserves referential and functional integrity. There are commercial CERIF systems, two of which were bought by Elsevier and Thomson-Reuters to include CERIF in their products.

2.7 DATA REPRESENTATION IN KEY VERTICAL DOMAINS

Several vertical domains of interest for AI do not yet have common data representations but have nevertheless started initiatives in data format sharing.

2.7.1 ISO Activities on Space Data

The Consultative Committee for Space Data Systems (CCSDS) was formed in 1982 with the goal of gathering best practices by the major space agencies of the world and developing a common solution to the operation of space data systems. While the CCSDS is concerned primarily with space data, the work of ISO TC20/SC13 is applicable well beyond the space data community. The National Archives and Records Administration and other digital cultural organisations also participate in the group. Much of the work is focused on long-term (long enough to be concerned about obsolescence and usability) preservation and use of information, and interoperability between data repositories, data producers and their users. Relevant standards include the following:

- ISO 16363, Audit and Certification of Trustworthy Digital Repositories (TDR). The OAIS Reference Model is adopted by many 'OAIS-compliant' digital repositories. At the time ISO 14721 was first developed, there was no standard to assess compliance with the reference model. ISO 16363 was developed to fill that gap. In addition to providing for the audit and certification of TDRs, the standard can serve as a road map for developing the policies, procedures, staffing and infrastructure for setting up a TDR that is compliant with the OAIS Reference Model.

2.8 DATA REPRESENTATION FOR BIOINFORMATICS

Applied Proteogenomics Learning and Outcomes (APOLLO) aims to correlate all genomic, proteomic and clinical data with imaging data with a focus on precision medicine or targeted medicine. Three major developments were launched. First, in the Precision Oncology Program (POP, March 2015), the US Department of Veterans Affairs (VA) program focused initially on lung cancer. It was designed to seamlessly merge traditional clinical activities with a systematic approach to exploiting potential breakthroughs in genomic medicine and generating credible evidence in real world settings and in real time. The second program, Apollo (July 2016), was inspired by Moonshot, where a coalition was formed between the US-VA, the US Department of Defense and the US National Cancer Institute to help cancer patients by enabling their oncologist to more rapidly and accurately identify drug treatments based on the patient's unique proteomic profile. The third program was Research POP (RePOP, July 2016), the research arm of POP, consisting of veterans who agreed to share their medical records (clinical, imaging, genomic, etc.) within and outside the VA for the purpose of finding the cure for cancer. The Veterans Health Administration consists of 8,000,000 veterans, 160 VAMC, 800 clinics, 135 nursing homes. It also has the backbone operational infrastructure of the Veterans Information Systems and Technology Architecture (VistA).

2.9 DATA REPRESENTATION FOR SMART CITIES

Smart cities provide a rich environment with heterogeneous data from many diverse IoT sensors. The complexity of such data collection includes different real-time communication protocols, data formats, data stores and data processing methods, either at the edge or at the central office. The combined data enables decision-making from everyone, from the city residents to the city government.

2.10 DATA REPRESENTATION FOR INTELLIGENT MANUFACTURING

Smart manufacturing plays a central role in data integration, from diverse supply chains of raw materials on product specifications to quality monitoring throughout the production life cycle. Additional data and metadata are generated from many different supporting sensors and machinery for real-time analysis and decision-making to provide safe and healthy environments, bring precise and quality processes and deliver reliable and superior products.

2.11 RECOMMENDATIONS

Supporting diversified representations for AI data assets is essential for organisations to reduce the corporate burden of AI. Key recommendations include the following:

- Utilise standard metadata as much as possible to capture precise description, data types, properties, unit of measurement, characteristics, etc., for given data elements.
- Adopt/develop standard metadata registries to support catalogues and types registries.
- Adopt/develop standard interfaces to support online data element definition.
- Adopt/develop standard computable object workflow functionality to trigger non-functional properties, including privacy and ethical issues in AI-ML data assets.

2.12 BIG DATA SYSTEMS

Data storage requirements for AI vary widely according to the application. Medical data, as well as imaging data sets used in military applications, frequently combine petabyte-scale storage size with individual files in the gigabyte range. Numerical data used in industrial areas such as maintenance, like the running example in the previous chapter, are often much smaller.

One of the key requirements of big data storage systems is to handle very large amounts of data and maintain the rates of high input/output operations per second (IOPS) needed to feed some AI-ML models. Indeed, these requirements are incompatible with traditional file system organisation based on files and folders.

When performance is not the top priority and one can accept response times on the order of seconds, scale-out (or clustered) network-attached storage (NAS) can be used. NAS consists of file access shared storage that uses parallel file systems distributed across many storage nodes to handle billions of files without the kind of performance degradation that occurs with ordinary file systems as the folder tree grows. Another storage technology that can handle very large numbers of files is object storage. This tackles the same challenge as NAS – traditional tree-like file systems become unwieldy when they contain too many files. Object-based storage copes with this issue by giving each file a unique identifier and indexing the data and their location. Object storage systems can scale to very high capacity and large numbers of files estimated to be in the billions. Flash storage is commonplace now, while NVMe flash is emerging as the medium of choice for applications that require the fastest access for data stored near the graphics processing unit (GPU). The spinning disk is still there, too, but is increasingly being relegated to bulk storage on lower tiers.

2.12.1 Big Data Platform Structure

Big data platforms are software systems designed to process huge amounts of data in a short time. We can classify these platforms according to their data ingestion modalities (Section 1.3.3):

1 Platforms for stream data
2 Platforms for batch (stored) data

An example of the first category of platforms is the software platform used by X to monitor in real time the messages sent and received; the second category includes the EOSDA cloud-based platform to analyse satellite imagery for business and science purposes. Big data platforms rely on the following principles:

1 *Distribution*: A single high-performance computer is not sufficient to handle the AI-ML workload. Thus, big data platforms use clusters of low-cost machines connected together.
2 *Data chunking*: Data are split into smaller chunks that can be processed independently.
3 *Parallelism*: Each task of the life cycle is subdivided into smaller tasks that can be executed in parallel

However, the existence of a high number of nodes introduces another problem: an increased probability that some nodes can crash or the storage system can fail. To overcome these problems, big data platforms use three main strategies:

1 *Functional paradigm*. The implementation of the tasks follows the functional programming paradigm. The main property of this paradigm is that it has no side effects: different nodes, executing the same task on the same data, in different instants, generate the same result, regardless of which tasks have been previously executed.
2 *Replication*. The data are replicated several times: in this way, if a storage system's component fails, there exists another copy available somewhere else.
3 *Graceful failure*. If a task fails, the same task can be submitted to another node, or, in the real-time systems, the same task can be executed two or more times, and the result can be obtained from the working node.

2.12.2 Hardware Issues

Besides the software architecture, AI-ML models' need for speed has encouraged the use of a high number of GPU-intensive clusters. GPUs were originally used to accelerate memory-intensive geometric calculations such as the rotation and translation of polygons' vertices into different coordinate systems.

Since most of these computations involve matrix and vector operations, GPUs have become increasingly used for non-graphical calculations; they are especially suited to parallel problems. AI-ML models' requirements boosted the interest of GPUs. While training AI-ML models, GPUs can be hundreds of times faster than Central Processing Units (CPU)s. Today, there is some competition between GPU and custom integrated circuits (ASICs), including the tensor processing unit (TPU) designed by Google.

2.12.3 File Formats

Some file formats for big data are more efficient than regular formats for AI-ML applications, as they permit to read only specific parts of the file. The most famous one, now a standard, is Hierarchical Data Format, versions 4 and 5). It is able to save in an efficient way (in binary and compressed format) a data frame with hierarchical indices where the values can be matrices or tensor data used by AI-ML models.

Other specialised data formats include the following:

- *Apache Parquet*. A columnar data structure, defined by Cloudera and X
- *Apache ORC (Optimized Row Columnar)*. Defined by Hortonworks and Facebook

3 DATA EXPLORATION

It is important to rely on a data source only after having carefully examined the data it provides. The huge size of data sets used for AI applications makes it difficult for humans to inspect raw data vectors directly: there are just too many of them. Statistical and visual analysis tools play an important role in the data exploration phase of the AI life cycle.

3.1 STATISTICAL ANALYTICS

In this section, we outline the main statistical procedures carried out in the data exploration stage.

3.1.1 Variables and Covariates

In statistical terminology, the dimensions of each data vector that will be used by prediction or classification are called variables, while the other dimensions are called covariates. When ingesting data about a phenomenon, such as the rotation speed of the rotatory engines mentioned in Section 1, covariates can be constant (that is, fixed at ingestion time, e.g., the engine's model) or change over time (for example, the engine's physical location). The covariate studies carried out in the data exploration stage try to determine which covariates are helpful for the prediction of classification of the variable values. Such studies aim to achieve results analogous to the feature reduction stage of the AI life cycle.

3.1.2 Size Determination

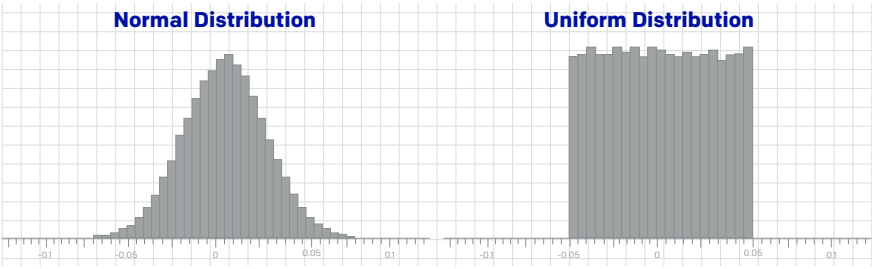
In the data ingestion stage, it is often implicitly assumed that the data points in a data set are samples representative of a wider population. An important requirement to be checked at exploration time is to have data from an adequate number of instances. Whatever the AI technique that will be used, any estimate based on a small number of instances will be less reliable than one based on a larger number, and when statistic models are fitted to small data sets, the estimated impact of the covariates is too imprecise to give reliable answers. A rule of thumb is that even when simple regression models are used, at least ten data points need to be included in the data set for each dimension considered; oth-

erwise, regression coefficients may become biased. Several books and software packages are available to assist the calculation of adequate sample sizes, and many general-purpose statistical packages also perform such calculations.

3.1.3 Distribution Fitting

A major part of exploratory statistical analysis is *probability distribution fitting*, (i.e., checking that a given statistical distribution or model is an appropriate representation of the ingested data). Fitting involves computing the corresponding *distribution parameters* (e.g., the distribution mean and variance based on the data set parameters average and *deviation*). Knowing which probability distribution is a close fit to the ingested data set can be very helpful in the next phases of the AI-ML life cycle, such as model selection. Figure11 depicts two common distribution functions, the Normal (or Gaussian) distribution and the Uniform distribution. Generally speaking, however, distribution fitting is not straightforward, and requires some background in statistics. Here, we only aim to present an overview of some of the major issues involved.

Figure 11:
Distribution Fitting



The first step usually consists of guessing the appropriate statistical distributions to try on the data. Building the *cumulative distribution function* (CDF) of the data values requires listing the frequency of each data value in the data set. From this histogram one can derive the *probability distribution function* (PDF) of the data. Educated guesses about the data distribution made at the data exploration stage usually take into account the presence or absence of symmetry in the data set. For example, data values whose frequencies lie *symmetrically* around a value may fit the different shapes of the symmetrical normal (or Gaussian) distribution, depending on mean μ and variance σ. Other symmetric distributions are the logistic distribution, the Cauchy distribution or the Student's t-distribution. The latter is an example of a symmetric *heavy-*

tailed distribution, meaning that the values farther away from the mean occur relatively more often.

When large data values tend to be farther away from the mean than smaller ones, one has a distribution skewed to the right. Skewed distributions include the log-normal one, where log values of the data are normally distributed, the exponential distribution, the Pareto distribution and many others. When small data values tend to be farther away from the mean than the larger ones, the distribution is *left-skewed*, like the square-normal distribution (i.e., a normal distribution applied to the square of the data values). Of course, the true probability distribution of data may be different from the fitted distribution, as the ingested data may not be totally representative of the underlying phenomenon due to measurement error. Also, there is *non-stationary* behaviour: the occurrence of data in the future may deviate from the fitted distribution. In other words, a change of environmental conditions may cause a change in their probability of occurrence.

To quantify how well a distribution fits the data, it is customary to use *parametric tests*, where the parameters of the distribution are computed from available data. Such tests are available in many statistical packages and libraries. It is also customary to transform right-skewed asymmetric data to fit symmetrical distributions (like the normal and logistic ones) by applying the logarithm or the square root to the data, or to fit a left-skewed distribution by computing the square values. Raising data to a power p, one can try to fit symmetrical distributions to data obeying a distribution of any degree of skewness. This technique enhances the flexibility of probability distributions and increases their applicability in distribution fitting. Another popular technique is *distribution shifting* (i.e., replacing each raw data value V by $V' = V_m$, where V_m is the minimum value of V). This replacement represents a shift of the probability distribution to the right, as V_m is negative. After completing the distribution fitting of V', the corresponding values are computed as $V = V' + V_m$, which represents a back-shift of the distribution to the left. Distribution shifting augments the chances of finding a properly fitting probability distribution.

It is also possible to fit two different probability distributions, one for the lower data range, and one for the higher. The ranges are separated by a break-point. The use of such composite probability distributions may be advisable when the data are collected under different conditions.

3.2 VISUAL ANALYTICS

Visual analytics is an outgrowth of the fields of information visualisation and scientific visualisation that focuses on analytical reasoning facilitated by interactive visual interfaces. It can attack certain problems whose size, complexity and need for closely coupled human and machine analysis may make them otherwise intractable. It integrates machine analysis process, human cognition and perception and information visualisation to lead the researcher in the process of analysis. The main aim of visual analytics is to amplify the analyst perception by providing a visual representation of the data that results from the analysis process. The analyst can interact with both the information visualisation, by zooming and filtering, and the analysis process by choosing the analytics methods or changing attributes. In this context, the cognitive ability of the analyst is the key to building hypotheses and making decisions.

Visual analytics seeks to blend techniques from information visualisation with techniques from computational transformation and analysis of data. Information visualisation forms part of the direct interface between user and machine, amplifying human cognitive capabilities in a few basic ways:

- by increasing cognitive resources, such as by using a visual resource to expand human working memory;
- by reducing the search space, such as by representing a large amount of data in a small space;
- by enhancing the recognition of patterns;
- by supporting the easy perceptual inference of relationships that are otherwise more difficult to infer;
- by perceptual monitoring of a large number of potential events; and
- by providing a manipulable medium that, unlike static diagrams, enables the exploration of a space of parameter values.

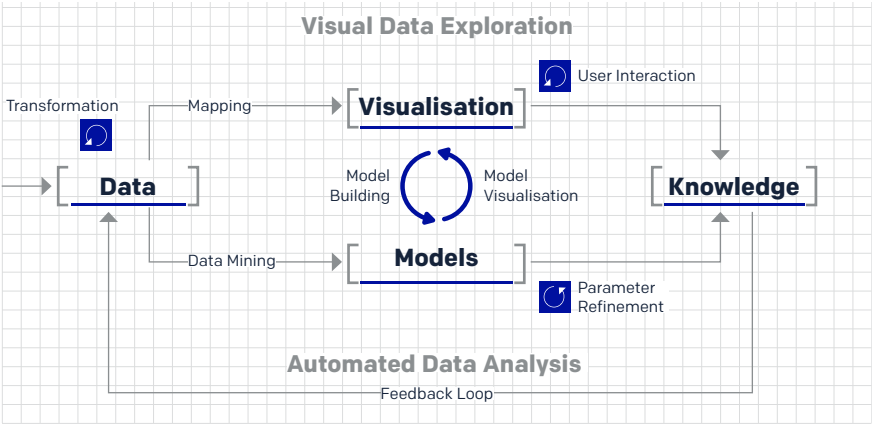
These capabilities of information visualisation, combined with computational data analysis, can be applied to analytic reasoning to support the sense-making process.

3.2.1 Visual Analytics Process

During the visual analytics process, the user alternates data visualisation and analysis of results by trying to gain insight and knowledge that up to that point has been hidden. Figure12 illustrates the visual analytics process: ovals represent stages, and arrows represent transitions. The process is iterated in subsequent steps until the analyst is satisfied with the extracted knowledge:

- Some data sets may require transformations such as integration, cleaning or normalisation before analysis may begin.
- The analyst is typically given two options:
  - First, to visualise the data and come up with a hypothesis or remodel data.
  - Second, to analyse the data and build models using data mining methods, then visualise it.
- The analyst is part of the loop in both cases. For visualisation, they can zoom in/out in the diagram to build hypotheses. Besides, in the analysis processes, they can choose the method of analysis or change parameters to test them.

Figure 12:
Visual Analytics Process



3.2.2 Some Examples

Common features of visual analytics tools include the capability of data visualisation across a number of dimensions, a rich and user-friendly dashboard, the capability of integrating different data sources and sometimes the support for multi-user collaboration in the analysis. An example of a well-known analytics tool with multidimensional visualisation is Gapminder Trendalyzer. It can be described as a bubble chart using animation to illustrate trends over time in three dimensions: one for the X-axis, one for the Y-axis, and one for the bubble size, animated over changes in a fourth dimension (time). Colour and other graphical markings can add extra dimensions. For instance, one can represent the average income of people within a country on the X-axis, life expectancy on the Y-axis and the population as the size of a bubble, and use bubble colours to denote the continent where they are located. Using these conventions, one can observe the time evolution of the first three quantities over time and, for instance, make hypotheses about how the correlation between the first and the second has developed. Selecting one continent rather than another allows us to formulate hypotheses about the different dynamics present in distinct regions of the world. The tool is also effective for storytelling. Rich dashboards are featured by commercial products. They offer a wide selection of gauges, data views, maps, charts, widgets, tables and other data-aware objects for story boarding and data representation.

4 DATA PRE-PROCESSING

The term *data preparation* is used to designate the first step of the pre-processing phase, where raw data are improved to make them fit for AI-ML applications. It typically consists in determining the data quality to execute appropriate data cleansing procedures. Depending on the application, data cleansing may be limited to the elimination of individual or systematic errors or inconsistencies with the data types or proceed further to enable or improve the efficiency of the data application process. The latter may also involve filling in missing data, encoding the data to the less occupant data types or to meet the numerical format requirements or even, in case of supervised ML applications, to label or tag the data samples into application-defined data patterns or classes.

4.1 QUALITY AND CLEANNESS OF DATA

In general terms, data quality or *cleanness* refers to the level of data completeness, consistency, accuracy or precision defined by the data requirements and variable descriptions or implied from the distribution of values. This sort of independent interpretation of the data quality is often extended in a case of known data utility or application to simply inform, to which extent the data in its current state can be readily used to solve the specific problem. In all the cases, the general rule is that the better the quality of the data, the less preprocessing/cleaning effort is required and the sooner they can be utilised to generate better data application outcomes.

Several standard data quality characteristics allow for rapid evaluation of the level of preprocessing effort needed to clean the data and often inform the choice of the appropriate data modelling technique for the data at hand. These data quality characteristics are as follows:

1. **Data completeness** describes the degree to which each field contains a complete or non-missing set of values as defined by the data size. It is usually measured in percentage terms for each data variable/field. It is worth noting that missing data may or may not be specifically marked in the data. Although white space or the markers NULL, NAN, NIL are often used to mark the missing data entry, in general one should receive the

missing data marker from the data provider or otherwise infer it from the data.

2. **Data consistency** refers to the level of agreement in format and type that is observed within the data set or its variables/fields. For example, if in a *date* column full of dates in dd/mm/yyyy format, we suddenly observe the number 45 or the colour red, this is clearly a value that is inconsistent with the date type. In fact, even the value 13-MAR-2020 would be inconsistent because it uses a different date format. For irreconcilable data inconsistency (as in the first case above), the marker must be replaced with the missing value marker; in the case of a minor format inconsistency (as in the second case above), it must be corrected to the correct form.
3. **Data accuracy** refers to the degree of correctness with which data values reflect the measured characteristics. The assumption here is that any inaccurate data are correct in terms of the type and format required by the variable they describe, but their values are simply wrong. Data inaccuracies can be a result of flaws in the measurement theory, mistakes in the data acquisition process or various other possibly compounded errors unaccounted for in data generation, which only surface during a quality check in the form of spikes, outliers, distribution perturbations and other statistical inconsistencies.
4. **Data precision** refers to the required level of detail to sufficiently describe a specific variable with an instance of a measurement. If, for instance, a floating-point temperature measurement with three decimal digits' precision is suddenly met with integer values, or if a millisecond's precision timestamps are mixed with daily dates, we consider such variable instances imprecise or of variable precision. Low-precision data are considered to be a less severe data quality issue, as they do not lead to data removal but simply to the attribution of certain non-zero uncertainty bounds around the precise measurements.

4.2 DATA CLEANSING

Data cleansing covers all the preprocessing activities that are required to be carried out on the data to eliminate or resolve data quality issues. As mentioned before, these activities can be divided into a standard set of activities that are application independent, which improve the state of the data whatever the application, and activities that are dependent on/guided by the data application to efficiently and effectively improve data utility for this application. The boundary between the two cleansing techniques, application dependent and independent, is often blurred. The reason for the reduced distinction is that there are a growing number of possible applications of data; therefore it is increasingly easier to find at least one way to subjectively enhance the improvement by performing the 'standard data cleansing' differently. For the same reason, it is always advisable to keep a copy of uncleansed raw data as a reference backup point that the data can be reverted back to if needed.

The data quality issues that can safely be cleaned whatever application is used consist of removing data inconsistencies, both by marking the inconsistencies as missing values and correcting the inconsistent formats to the correct form. The data imprecisions are often simply left in the same state or are artificially transformed to the dominant precision if possible.

The cleansing of all other data quality issues is mostly dependent on the data application. There are two data cleansing practices that are most often carried out in preparation for AI applications:

1. **Filling in missing data** is typically required by supervised and unsupervised ML but also by many statistical analysis methods that are saturated with mathematical methods and numerical processing techniques. Not filling in the missing values would result in infinities, singularities or other errors. There are a number of standard methods for filling in missing data, and their effectiveness depends on the context and type of data application. They may also be considered in terms of the risk level associated with the introduction of new values of uncertain validity.

   It is important to remark that filling in a missing data point is not a necessity in all circumstances, especially in the first stages of the life cycle. In fact, the most risk-averse or agnostic approach to missing data handling is to simply clean and replace it with the single representation that some methods may be able to use. For example, a number of

mathematical models can handle infinities, as well as so-called *not-a-number* (*NaN*) entries. Moreover, in case of categorical variables, missing values can simply be pulled together and labelled as a new missing value: NULL. In both cases, filling in missing values really becomes an exercise in replacing all their different forms with the single representation accepted by the application and consistent with the specification that can handle them (i.e., NaN or NULL). Following this approach eliminates the risk of introducing new uncertainties along with the missing data.

One of the most conservative methods of filling in missing values is inserting the variable's mean, *mode* or *median* value, whichever is most appropriate. In the case of temporal or sequential data, the least risky method for filling in the missing values is the *copydown* method, which simply populates the missing values with the last known non-missing value in a sequence. In case the sequence starts with some missing values, the next least risky strategy is to fill in this initial missing section with the first following non-missing value found in a sequence.

The riskier methods for filling in missing data in the context of multi-dimensional data are based on the concept of data similarity (i.e., the *nearest neighbour*). Assuming that the data instance is not missing in all dimensions, this method will search for the most similar cases among these other non-missing dimensions. Then, when the target variable value is found, it is simply filled in based on the values from the single most similar neighbour or the average over the k-most similar neighbours. In the sequential data, such a method can be directly replicated with some degree of influence by available temporal constraints such as *data interpolation*, which ensure the continuous flow of a sequential signal. However, interpolation can be very damaging if the data application uses time series forecasting because it would result in illegally passing future information back to the past. There are, in fact, spectacular failures reported in financial algorithmic trading based on interpolated financial time series. Hence, as mentioned above, the most suitable choice here very much depends on the data application.

In addition to the most and least risky strategies for filling in missing data, there are others that try to exploit and preserve other data characteristics, typically higher order statistics such as data distribution and data domain bounds, and apply semi-random data *sampling* from the distributions obtained from the non-missing part of the data examples.

These methods address the missing data problem while preserving marginal data distributions, but they also risk injecting accuracy errors by ignoring conditional dependencies with other variables.

2. **Outlier removal** is considered a removal of the data values that are significantly different from other observations. Even identification of outliers is typically arbitrary, because unless we possess prior knowledge about the data generation and its constraints (which is rarely the case), it is impossible to determine if the anomalous value genuinely reflects a valid measurement or is the result of some kind of error. Statistical *hypothesis tests* may be used here to identify the outliers and generate a likelihood value on how probable it is that the value could have been generated from the measured probability distribution, but a more pragmatic approach relies on the data application to decide where to set the threshold for outlier removal so that the performance of the data application is maximised.

   In practical terms, outlier removal follows a phase of outlier identification, followed by replacing them with missing data markers and optionally further refilling them using the most suitable method for filling in missing data for the data application at hand.

4.3 DATA NORMALIZATION

The term normalisation informally designates all transformations performed on ingested data at the data exploration stage or (more frequently) at the data preprocessing stage of the AI life cycle. It is also possible that the outputs of certain AI-ML models are renormalised before feeding them to other models in scenarios where multiple ML models are connected in a sequence. The overall goal of data normalisation is to bring the values of the components of data vectors to a common scale, without distorting the differences in the value ranges. Normalisation prevents the scales of dimensions in the input data vectors from affecting the dimension's importance for the AI model that will be used. For example, consider a bi-dimensional data set containing only two features, **v** = (V_1, V_2), and assume that V_1 values range 0˜10, while V_2 ranges 0˜100000. In some data analysis techniques, like linear regression, V_2 will end up influencing the regression model's result more than V_1 due to its larger size, although it may be less useful than V_1 for computing an accurate output.

Normalising data also allows designers to neglect the measurement units, enabling AI models to consider all dimensions of a data vector as pure numbers on the same scale.

It is important to remark that not all data sets require normalisation before applying ML models. It is mandatory only when the features have different ranges. Popular ways to normalise data include the following:

- *Normalising data distribution's moments*. Transforming normally distributed data to obtain a distribution whose mean μ = 0 and standard deviation σ = 1
- *Standardising data values*. Transforming data using a z-score. This transformation is usually called standardisation in statistics textbooks. To perform this transformation, one starts from the mean μ and also the standard deviation σ of the data probability distribution. In the AI life cycle, these parameters are estimated in the fitting procedure of the exploration phase. The z-score substitutes each data value with its distance from the mean of the population distribution. This distance is expressed as the number of standard deviations that separate the data point from the population mean. By definition, z-scores are symmetric and follow a normal distribution. They usually span the *six sigma* interval: from **3** standard deviations (at the far left of the normal distribution curve) to

+3 standard deviations (at the far right of the normal distribution curve). Z-scores are also a way to compare normally distributed data values belonging to huge data sets. Estimating how far a given value V lies from the mean may be difficult when V belongs to a data set with millions of entries; the Z-score of V expressed directly where V lies with respect to the population's mean.

- *Rescaling data values*. Transforming data so that they all have values between **0** and **1**. This transformation is also called feature scaling. While feature scaling can in principle be computed by dividing all data values by the largest one ($\frac{V}{Vmax}$), this may bring rounding errors when $Vmax$ is much larger than the other data values. The most popular rescaling formula used by statistics tools and libraries is

$$V' = \frac{V - V_{min}}{V_{max} - V_{min}} \qquad (1)$$

- *Normalizing data vectors:* Divide each data vector **v** by its norm, so all data vectors have a length of one.

4.4 DATA ENCODING

In the context of AI-ML models, the term data encoding refers to reversible transformations upon the data vectors' components or features that map their original values to a new set of discrete values in order to achieve certain benefits for the data application, such as improved model performance. Data encoding offers a way to convert categorical variables to numerical representations such that the AI-ML models exclusively working on numerical data could include additional variables and often deliver improved performance.

The common strategy that underpins all data encoding methods used in the context of AI-ML is mapping inputs into a discrete (typically ordinal) representation. The most common starting point in data encoding discussions is categorical data encoding. We will also cover numerical and text data encoding while mentioning image and sound data encoding techniques.

4.4.1 Categorical (Nominal) Data Encoding

Nominal data encoding is aimed at mapping each unique category of the categorical variable into a discrete numerical format that can be easily absorbed by the numerical algorithms of ML models.

1 *Ordinal*. This encoding simply maps each category of the nominal variable to a discrete number starting from 1. Since the categories are orderless by nature, the order of the encoded categories could be random, but it is a good practise to follow lexicographic order when categories are expressed using alphanumeric symbols. In case of categories typically represented by text, ordinal encoding delivers enormous savings in memory storage requirements because hundreds of bytes required to store one text string entry are replaced by 1 or a handful of bytes to represent the ordinal number.

2 *One-hot*. Even though categorical values do not have any intrinsic order, ordinal encoding introduces it because of ordinal enumeration, which can mislead certain ML methods which would utilise the numerical distance. The technique of one-hot encoding eliminates this problem by marking the occurrence of each unique categorical value in a separate binary indicator variable. Given m categories of the original categorical variable, one-hot encoding converts them to m binary variables, each exclusively marking the occurrence of the corresponding category. One-hot encoding is suitable to numerically represent categorical variables taking a small number of unique

values. Otherwise, the benefits of mapping to orderless numerical representation are outweighed by the growing dimensionality and increased computational cost of processing such data.

4.5 DATA ANONYMISATION

Anonymisation of the data can be done as part of preprocessing the data or as part of data ingestion. The position of this step in the AI life cycle can be related to the data governance policies at the organisation adopting the AI-ML model. The best approach is often to anonymise in the data ingestion phase, to avoid any data leakage or violation of privacy during the AI-ML life cycle. In the case where data preprocessing is the responsibility of the data owner, it is not unusual for anonymisation to be part of preprocessing. Organisations should be careful in setting up their strategy for anonymisation and in positioning it correctly in the life cycle of their AI application, as will be discussed in the remaining sections.

4.6 DATA LABELLING

The classification process associates a label to input data vector. For instance, in health care, an image representing an MRI scan could be associated to a label corresponding to the description of its content: a picture of a skin lesion could be labelled with the diagnosis while labels of chest X-rays could indicate pneumonia. In a sentiment analysis setting, an X post could be associated to a label describing the mood or sentiment of the author with respect to the X post's subject. Similarly, regression requires a numerical value to be associated to each input: the row of values describing details about a real estate property should have a price associated to it.

Labels come in many forms. Prelabelled data sets (open or proprietary) are sometimes available. In other cases, only unlabelled data sets are available, and labelling must be carried out during the AI-ML model development.

When some or all of the labels are missing, one has to devise strategies to fill in the gaps. The main strategies are the following:

- *Manual labelling by one or more experts*
- *Labelling by an 'oracle' algorithm*
- *Use of unsupervised techniques for partitioning followed by labelling block representatives*

Other strategies available, when the data are partially labelled, involve adopting special learning algorithms in the learning phase: semi-supervised algorithms such as self-training and co-training algorithms.

Manual labelling. In manual labelling, human experts take individual examples and attach to them a label taken from a predefined set. This process is typically time consuming, rather expensive and cannot scale to a large number of examples. Manual labelling can either be done in-house, get crowd-sourced or be outsourced to individuals or companies.

Oracle algorithms. The use of an oracle consists of running an available classification algorithm and tagging the data with the labels it issues. Then, the labelled data can be used for training another ML model. The use of oracle algorithms is relatively rare and justified only in special circumstances because if a well-performing classification model is available, then it is hard to justify spending time and resources on preparing the data and training a new one. One such example is if the available classification algorithm

is a black box, while one would like to acquire some understanding of the classification logic; sometimes the available model is known in detail but the architecture is not satisfactory for some reason; for example, the oracle algorithm is available for a limited time or under constraints that cannot be fulfilled in the long run.

Unsupervised partitioning. In certain scenarios, the data can contain regularities that allow for partitioning them into categories based on similarity, yet each partition would be of unknown meaning. The meaning of each partition can be found by human experts at a later stage, which would cut down the labelling effort to simply labelling the partitions rather than the entire data set. This fact can be exploited as follows: the partitioning algorithm is executed on the data (e.g., a *clustering algorithm*); the output is disjoint sets/blocks of data, each of which can be interpreted as an unknown category; a few examples from each set are labelled manually and the label is extended to all the elements of the set.

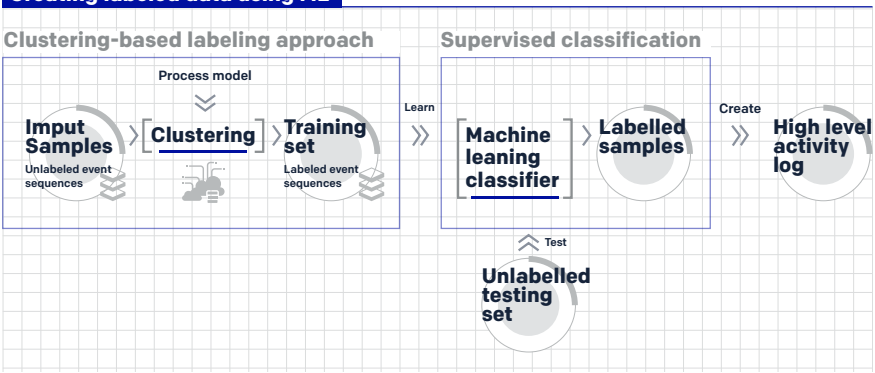
Semi-supervised algorithms. The use of semi-supervised algorithms is possible when only part of the data is labelled. This corresponds to the choice of tagging the unlabelled data as part of the learning phase instead of the preprocessing phase. One of the simplest examples of a semi-supervised learning algorithm is self-training. Let us consider, as usual, classification. In self-training, one has a base classifier model (e.g., a Naive Bayesian classifier) which, once trained, can provide for each example both a label and a measure of confidence in that label. The base classifier is trained on the originally labelled data; then it is used to infer the labels of a part of the originally unlabelled data; those data which are labelled with high confidence are selected and added to the original training set, and then the base classifier is retrained on this extended training set and subsequently used to tag new unlabelled data. The process can be repeated until all the data have been tagged and the base model is trained on the largest possible confidently labelled data set.

4.6.1 Creating labeled data using ML

While semi-supervised algorithms can deal with data that are partially labelled, sometimes no labelled data can be found at all, and manual labelling could be infeasible, time-consuming or too expensive. To cope with this challenge, a clustering-based labelling approach can be used to create samples of labelled examples. For every category, a cluster of data points should be formed. Then, all samples will be automatically labelled with the same labels as their corresponding clusters. Then,

these labelled examples are refined by taking only a subset of examples that contribute to the goodness of the labelling task. In a second phase, the labelled examples are used to train a supervised ML classifier. The classifier will learn the mapping between data points and the categories and then map a new set of unlabelled data points to the corresponding categories.

Figure 13:
Creating labeled data using ML



The goal is to create labelled examples to feed the supervised ML classifier. The input samples are unlabelled data points. To provide labels for these samples, they are clustered into different clusters representing the different categories. The approach is not strictly constrained to a particular clustering algorithm. Different clustering algorithms can be used, depending on the particular application setting considered. For example, *k-prototypes* can be utilised to cluster multivariate time series sequences with numerical and categorical attributes, while *k-medoids* is applied to cluster samples with categorical attributes. The 'orthogonality' of the clustering algorithm should be intended as another amenity of the proposed solution. Each sample within one cluster will be labelled with its cluster label. Specifically, to label the clusters, the domain expert should label only the cluster centres based on the known categories. Then the rest of the samples in a cluster will automatically take the same label as their centre.

4.7 FEATURE SELECTION

Feature selection is the process of reducing the number of dimensions in the data vectors, before feeding them to the selected AI model. The purpose of this process is eliminating non-informative dimensions to decrease the cost of the AI model computation and, in some cases, to increase the execution speed. There are multiple popular feature selection techniques, we focus on the feature selection techniques that are more popular in the AI-ML domain, starting from *filter-based* feature selection methods. Filter-based techniques simply discard dimensions, to discuss more sophisticated techniques that couple selections with transformations similar to the ones discussed in Section 3. From the data management point of view, selecting features always involves projecting the highly multidimensional data space built at ingestion time to a subspace with a smaller number of dimensions – in other words, creating *views* on the data space that correspond to training data sets to be used for specific ML models.

4.7.1 Dimensionality Reduction

Feature selection is the most straightforward way of reducing dimensionality. However, there are more sophisticated techniques for reducing the number of input variables: they operate not just as a selection method for variables but also as a transformation. Among the categories of dimensionality reduction techniques one can distinguish linear methods, such as the matrix factorisation techniques, from non-linear methods, such as manifold learning techniques or more recent techniques based on auto-encoders. Matrix factorisation methods can be used to reduce the data set matrix (whose rows are the individual data vectors) into its constituent parts. Examples include the *eigen-decomposition* and the singular value decomposition. The most important dimensions (e.g., those with largest eigenvalues) are kept, whereas the less important are discarded. The most common matrix factorisation technique for ranking the components is PCA.

4.7.2 Principal Component Analysis

To reduce data sets' dimensionality while preserving the information in the data, PCA computes a data set composed of new, uncorrelated variables that:

1 are linear functions of those in the original data set, and
2 maximise variance

A set of p original variables can be replaced by an optimal set of $q < p$ derived variables which are named principal components. While PCA (and other dimensionality reduction techniques) are usually performed after the preprocessing stage of the AI-ML life cycle, subspaces with $q = 2$ or $q = 3$ are sometimes used at the data exploration stage to obtain a visual representation of a multidimensional data set. Many variants of the PCA technique have been developed to handle different data types. Although a normal distribution of the data set is usually assumed, PCA does not, in principle, need any distribution assumptions and, as such, is very much an adaptive exploratory method that can be used on numerical data of various types.

PCA can be based on either the *covariance* or the correlation matrix of the original data. To understand the covariance-based computation of PCA, let us consider a data set with n data vectors $x_1, \ldots, x_p$, each composed of p numerical variables. These data values define a number p of n-dimensional vectors or, equivalently, an nxp matrix X. Covariance-based PCA computes a linear combination of the columns of X with maximum variance. For the more mathematically aware reader, we recall that such linear combinations can be computed by multiplying X to a vector **a** = $a_1, a_2, \ldots, a_p$. The variance of such a linear combination is given by $var(Xa) = a^T Sa$, where S is the covariance matrix associated with the data set and a^T is the transpose of a. To identify the linear combination Xa with maximum variance, it is sufficient to compute the p-dimensional vector **a** that maximises $a^T Sa$. Algebra tells us that for this linear system to have a well-defined solution, an additional restriction must be imposed. The most common restriction is requiring $a^T a = 1$. This problem is equivalent to maximising $f(a) = a^T Sa\lambda(a^T a_1)$, where λ is a Lagrange multiplier. The maximum corresponds to a point where the derivative of $f(a)$, which is denoted as $f'(a)$ and expresses the slope of $f(a)$, is zero (see Chapter 6). The condition $f'(a) = 0$ tells us that vector **a** that maximises $a^T Sa$ must be a (unit-norm) *eigenvector* of the covariance matrix S, and λ must be the corresponding eigenvalue. Any pxp symmetric matrix such as S has exactly p real eigenvalues, and PCA selects the largest one, λ_1 (and its corresponding eigenvector a_1). The procedure is iterated for all the eigenvalues λ_k, ($k = 1, \ldots, p$), which are the variances of the linear combinations, and the corresponding eigenvectors a_k, which form an orthonormal set of vectors that is the basis for the PCA-selected subspace, whose axes are called the principal components of the data set.

4.7.3 Other Techniques

Manifold learning is used to create a low-dimensional projection of high-dimensional data, a process that – like PCA – can also be exploited for data visualisation. The projection is designed to create a low-dimensional representation of the data set whilst best preserving the salient structure or relationships in the data. Examples of manifold learning techniques include Kohonen self-organising maps (SOM), Sammon mapping, multidimensional scaling (MDS), and t-distributed Stochastic Neighbor Embedding (t-SNE).

5 MODEL TRAINING

AI and ML are iterative methods that uncover valuable information and insights that are not clearly apparent from within data. The process helps unlock the data's full potential to deliver business value decision support and automation. AI-ML methods use models that training algorithms build from relevant data to fulfil the desired tasks and objectives, and, therefore, data are at the core of AI-ML, underpinning its operation and success. Without relevant data of the required quantity, quality and frequency, AI-ML would yield inaccurate and potentially unusable output. It is important to remark that despite the fact that scientists worldwide are developing more dynamic and adaptive methods, the reality remains that different business needs are likely to require the use of different AI and ML methods depending on the user requirements. More importantly, the models built by the algorithms need specific data to address the requirements at hand. Similar to the human brain, if we would like an algorithm to predict traffic jams on a particular road, then a sufficient quantity of relevant historical data about previous traffic jams, dates, times and weather conditions will significantly help in making more accurate predictions. However, the data required to predict a traffic jam are different from the data required to predict a football match outcome, even if we use the same methods.

5.1 TRAINING ALGORITHMS

In this section, we introduce the notion of training algorithms and provide an overview of the *gradient descent* (GD) approach used by algorithms that train supervised ML models. This discussion will require some simple mathematical notation. For simplicity, we will refer to an ML model for classification, defined by a function $F_w : DS \rightarrow C$ from the input data space (DS) to a finite set of classes or categories (C). As an example, the inputs to F_w could be sensor readings about the rotatory engines presented in Section 1, and the outputs could be the (*IMMINENT* and NOT-*IMMINENT*) labels about failure of the running example of Section 1.

Here we use the mathematical notation F_w to designate the ML model mapping inputs to one of the categories. This notation is useful because the subscript in F_w highlights the array **w** of the ML classifier's internal parameters, often called its weights. Of course, the data-flow structure of the computation performed by F_w depends on the specific ML model used, but the result of the computation depends on the values of the weights **w**. For example, using a multi-stage neural network (NN) as our classifier, the output of each stage of the network is computed as a weighted combination of the outputs coming from the previous stage, called activations [7]. Activations to the first stage coincide with the one-hot encoded inputs discussed in the previous sections.

In essence, model training algorithms are the algorithms that adjust the weights **w** of the model so that F_w coincides over S with a target function f: $DS \rightarrow C$, which expresses the correct classification of all points in the input space.

In terms of our notation, the training set mentioned in Chapter 1 is a set S of sensor inputs for which the values of $f: DS \rightarrow C$ (i.e., the right *IMMINENT* and *NOT-IMMINENT* labels), are known. In the remainder of the section, we outline how training algorithms work.

5.1.1 Gradient Descent Training

GD is a popular family of iterative algorithms for training supervised ML models like neural networks. GD is based on a simple mathematical notion, which can be expressed as follows: in any smoothly changing (in mathematical terms, *differentiable*) function f, a maximum or minimum

7. The reader interested in a general introduction to NNs can consult Michael Nielsen's free online book (http://neuralnetworksanddeeplearning.com/).

is always where the function flattens out (i.e., where the function graph's slope is zero). Calculus highlights that a function of a single variable $f(x)$ flattens out where its derivative f', expressing its slope, is zero [8]. For multidimensional functions $f(\mathbf{v})$, where v is an array of variables, we can look for points where the gradient ∇f, the multidimensional analogy of the derivative, becomes zero. The basic version of GD works as follows: at each step, the GD algorithm perturbs the ML model's weights vector **w**, applies the model F_w to one or more elements of the training set *S* and computes the model's current *classification* error E_w (i.e., the difference between the outputs of F_w on those elements and the elements' labels). Then, GD uses the error's variations across these input elements to numerically estimate ∇Ew and updates **w** based on it. The model's classification error E_w can be computed as the linear ($L1$) or quadratic ($L2$) sum of the differences between F_w outputs and the ground truths available as the known labels of the elements of the training S, in our notation $f(S)$.

Informally, it can be said that by this procedure, the GD tries to 'drive' F_w toward smaller values of the error gradient, progressively reducing Ew. The final goal of GD is to find the point where the ∇E_w gradient is zero, corresponding to the vector **w** that minimises E_w on the training set S.

5.1.2 A Closer Look

While the above informal description can be sufficient for a general understanding of GD-based model training, from the more mathematically-aware data manager's point of view, it is also useful to take a closer look to the computation performed by the software implementations of GD to estimate ∇E_w. This requires just a little bit of algebra. At each computation step, given the current weights vector **w**, the GD algorithm generates three nearby vectors w_1, w_2, w_3. This way, computing $E_w(w)E_w(w_i)$ ww_i gives approximately the directional derivative of the error E_w at w in the direction ww_i. The derivative is indeed the projection of the gradient $\nabla E_w(w)$ in the direction of $w - w_i$, or $\frac{\nabla E_w E w w_i}{w w_i}$ Now, let us assume the following approximation holds:

$$E_w(w)E_w(w_i) = \nabla E_w(w)(ww_i). \qquad (2)$$

As the error E_w is itself a scalar, i.e. a single number rather than an array, this is a system of three linear scalar equations in three unknowns (the components of ∇E_w). Basic algebra tells us that, if the three vectors ww_i

8. The derivative can also be zero in points that are neither a maximum nor a minimum, called saddle points. This is, however, outside the scope of our discussion.

are orthogonal, this linear system has a unique solution, so it can be solved numerically by the GD algorithm to obtain the gradient's components.

This computation requires calculating E_w, a computation that can in principle be done using a single element of S. However, the different implementations of the GD algorithm used in ML software libraries differ form one another in terms of the number of elements of the training set S that are used at each step to compute E_w. As intuition suggests, the higher this number, the higher are both the *fidelity* of the GD algorithm in following the error gradient and - unfortunately - its overall computation time.

- *Stochastic Gradient Descent* (SGD), is a variation of the GD approach that computes E_w, estimates ∇E_w and updates F_w using a single random entry e of S. Frequent updates of F_w introduce a noise-like "jerky" effect on E_w, but allow for continuously monitoring the ML model's performance.
- *Batch Gradient Descent* (BGD) computes error E_w (and estimates ∇E_w) for each $e \in S$, but only updates F_w after having scanned all of S (i.e. once for each *epoch*). Our intuition suggests that BGD's lower frequency of updates results in less sign variations in E_w. For our purposes, it is worth remarking that - due to the granularity of ∇E_w estimates - BGD is usually implemented in such a way that all the training set *S* needs to be in memory at the same time.
- *Mini-Batch Gradient Descent* (MBGD) splits randomly f into subsets (the "small batches"), which are used to compute E_w, estimate ∇E_w and update F_w accordingly. In this case what is used to estimate ∇E_w is actually an aggregation $h_{MB}(E_w)$, where MB is the mini-batch. Instead of computing the aggregation *h* as a sum of errors over the mini-batch, it is common practice of implementations to take the average, to keep E_w variance under control.

Today, the MBGD variant of GD has become increasingly popular and widely used for training "deep" ML models. Its update frequency is higher than the one of plain BGD; also, its batch size (one of the algorithm's *hyper-parameters*) acts as a control over the learning process. Small batch size values may give faster convergence at the cost of introducing noise in the training process. Large values give a learning process that converges slowly but provides accurate estimates of E_w gradient.

5.1.3 Federated Learning

The variations of the GD training algorithm described above are all centralized: all the training set *S* is in a single place and all of it is considered for extracting batched for the gradient's computation. In principle, the GD algorithm can be made *parallel* by using multiple batches B at the same time, and training the the ML model on multiple processors. Parallel implementations of GD should not be confused with *federated learning*, which is targeted to addressing data privacy and security as well as data access rights.

Federated learning is based on a different notion: multiple nodes hold each a part of the training data *S*, without sharing it. In terms of our notation, the training set S is partitioned into multiple local training sets $S_1,....S_n$ held by their respective owners, without explicitly exchanging data samples. The general principle of federated learning consists in training local models on local data samples and exchanging the models' internal parameters (e.g. their weights) at some frequency, in order to generate a global ML model shared by all nodes. In federated learning, the local training set's parts S1,....Sn are typically heterogeneous and their sizes may span several orders of magnitude. Moreover, the partners involved in federated learning may be unreliable as they are subject to more failures or drop out. There are two major families of federated learning algorithms.

- *Centralized federated learning* In centralized federated learning algorithms, a central coordinating node orchestrates the different steps of the training algorithm and coordinates the other participating nodes during the algorithm's execution. The coordinator is responsible for the nodes selection at the start of the training process and for aggregating of the received model updates.
- *Decentralized federated learning* In decentralized federated learning algorithms, participating nodes collaborate in a peer-to-peer fashion to obtain a global ML model. This organization aims to prevent single-point failure.

5.2 AUTOMATIC ORGANIZATION OF DATA

AI/ML models help to better understand data and uncover patterns and information hidden within it, to provide additional valuable insight. Hence, it is no surprise that one of the key challenges we first encounter when dealing with data, both structured (numerical or categorical data) and unstructured (text data), is the need to group together similar objects that the data represents. These groups will contain the objects that are more similar to each other than those in other groups based on some attributes of the objects. In many cases, the user does not have a view of the groups themselves or indeed the number of distinct groups. Hence, clustering the objects that the data represents provides an initial understanding of the data, that will help with further analysis. For example, let us consider a call centre for a retail bank, that receives a large volume of calls from customers. If the bank's call centre manager is planning training topics for his employees, then grouping the calls together in groups based on similarity will show the topics that are generating calls, and the volume of calls associated with each topic. This insight will help the bank focus the training on areas of importance to the customers, and help to provide a better service. This kind of grouping or clustering could also uncover topics that the manager may not have previously anticipated. In AI/ML, a clustering algorithm is a technique or method used to automatically group the objects that the data represents into different clusters based on their similarities. This is known as unsupervised learning.

5.3 GENERATING NEW DATA

It is possible to generate brand new data using certain AI techniques. These techniques are able to produce augmented data, i.e. synthetic training data of any size, targeting applications where requirements and results depend on greater quantity. You can use synthetic data when you are required to train an ML model requiring a larger amount of data than what you have, or to cover a different set of data points that have been too difficult to obtain by normal means. Synthetic data can be produced from any type of data including numbers, text, images and sounds. Other than for training purposes, new data requiring creative thinking can be produced in this way. Creating new art pieces, music or writings is possible using ML models that can learn the patterns from data made from the same creator.

6 PARAMETERS FOR MODEL TUNING

There are certain parameters which define high level concepts relating to ML models, such as their learning function or modality, and cannot be learned from input data. These model parameters, often called *hyper-parameters*, need to be set-up manually, although they can be tuned automatically by searching the model parameters' space. This search, called *hyper-parameter* optimization, is often performed using classic optimization techniques like *Grid Search*, but *Random Search* and *Bayesian optimization* can be used. It is important to remark that the Model Tuning stage uses a special data set (often called validation set), distinct from the training and test sets used in the previous stages. An evaluation phase can also be considered to estimate how the model would behave in extreme conditions, for example, by using wrong/unsafe data sets.

AI Model Tuning in a Nutshell: Apply model adaptation to the hyper parameters of the trained AI model using a validation data set, according to deployment condition.

AI Model Tuning in our Running Example: ACME data scientists run the 2D RNN model trained for fault prediction on an additional validation dataset and choose the best values h, k for the RNN's tensor dimensions.

6.1 MODEL HYPER-PARAMETERS

Model hyperparameters represent a higher-level set of variables controlling model design and architecture choices as opposed to the learnt data- or feature- level parameters capturing the content of the data and their characteristics and structure. Hyperparameters typically determine mathematical representation, apparatus and operation logic as well as various algorithm-level choices that decide about the complexity, efficiency, breadth and depth of the optimisation process underpinning the model training and testing.

Hyperparameters always accompany virtually every ML model, although the extent to which the model implementation software exposes their control to the user varies enormously, in part due to the model's intrinsic complexity and in part to different software engineering strategies. The drive towards

AI adoption increasingly sets the tendency to automate setting of the strongly correlated choices to limit their ambiguity and uncertainty for user convenience. From the user's perspective, hyperparameters may not even be visible at first, as the models are always preset with the most common default hyper-parameter choices. This is both an advantage and a risk in practical applications; on the one hand, such a model build can be instantly executed without much knowledge about it, but on the other hand, great models with the wrong hyperparameters can be prematurely discarded based on preliminary, ballpark performance estimates.

ML models vary significantly with respect to the number of hyperparameters and the sensitivity of the model's performance to their choices. Traditionally, ML models evolved from virtually fixed hyperparameterless, stable settings like in linear regression; to a setting with no more than a few hyperparameters like in *k nearest naughbour* (KNN), decision trees, naive Bayes or Gaussian mixture; and finally up to larger and larger numbers of hyperparameters in layered composite network models, where each layer is effectively controlled by its own set of hyperparameters. With this evolution, an individual ML model de facto becomes an ever-growing family of model version choices – hence its growing sensitivity of predictive performance on ever more expanding, subtle sets of model hyperparameters.

In general, models with a small number of hyperparameters are preferred, as long as their predictive power and application flexibility are not compromised. Sometimes mathematical ingenuity can be used to eliminate hyperparameters risk-free, as in the case of Gaussian process generalising multivariate normal distribution-based models. However, this is usually achieved at the expense of significantly grown computational complexity, with only marginal improvements in predictive performance. The practical strategy that emerges from the compromise of these colliding tendencies of exposing the hyperparameters to model fine-tuning is that the ML model release, however complex, has identified several key hyperparameters that account for the vast majority of its modelling and predictive capabilities, and the user is presented with the option to optimise the model along with these hyperparameters if required.

6.2 HYPER-PARAMETERS OPTIMIZATION STRATEGIES

Each configuration of hyper-parameter values set to train and test the ML model represent just a single solution evaluated by the model testing error obtained through the cross-validation or other the generalization error estimate method. Since the whole cycle of model training and testing is typically a costly operation, optimization of the hyper-parameters is rarely a search for an optimal set of hyper-parameters' values that yield the best predictive performance, but for practical reasons often to quickly find a good and stable set of the hyper- parameters' values that consistently return good predictive performance of the model. Depending on the cost and time of an individual model build hyper-parameter optimization can be, and typically is, aided by the parallelized evaluation process but what is the most characteristic about this optimization is a very careful selection of the hyper-parameters' values before passing them for model build and the evaluation. For this reason among the most common optimization of the hyper-parameters are the grid search, probabilistic Bayesian and the greedy-linear iterative methods.

6.2.1 Grid search method

Grid search method's principle is to cover the whole domain of every parameter with a regular grid of a couple of values and then evaluate all the combinations of such grid. Given n hyper-parameters to optimize and a grid of size k, there are however still k^n evaluations required to complete such grid search. The grid size can obviously be reduced if necessary, some conflicting or impossible combinations may be manually excluded from the evaluation to speed up the search. The real problem, though, is that the grid search does not in general exploit the previous iterations' to improve the next iterations' performance. For this reason, given the growing cost of each model evaluation, the iterative probabilistic or greedy search methods are more commonly used in practical applications.

6.2.2 Bayesian hyper-parameters optimization

Bayesian optimisation of the hyperparameters tries to build a simple probabilistic model of the relationship between the hyperparameter values and the predictive model performance and then uses it to improve the selection of the next parameter set based on all the previous set evaluations. Each new evaluation yields more reconciliation evidence

between the probabilistic distribution model and reality and allows experts to rather quickly find the model-inferred near-optimal choices. The Bayesian hyperparameter optimisation method is very quick. However, its performance still depends on the accuracy of the probability distribution assumptions, which must still be made manually for each parameter (although Gaussian is most commonly assumed by default).

6.2.3 Greedy-linear iterative search

Greedy-linear iterative search is a hybrid search that combines the advantages of the grid and Bayesian searches. Starting from the default hyperparameter values, it carries out the search sequentially, optimising single parameters one at a time with other parameters fixed. Once the individual hyperparameter is optimised, its value becomes fixed and the next hyperparameter is optimised the same way. These rounds of single-parameter optimisations continue in a loop until an entire round occurs without a single parameter change. Note that for individual parameter optimisation there is the freedom to use a grid search, Bayesian or any other search. The search may therefore flexibly incorporate various desired elements of other optimisation techniques, and, since it typically completes within just a few rounds, it can be considered near-linearly complex with respect to the number of hyperparameters to optimise.

6.3 TRANSFER LEARNING

ML methods have proven to be useful in analysing a vast amount of data in its various formats to identify patterns, detect trends, gain insight and predict outcomes based on historical data. However, ML models are challenging to reuse from a domain due to the change in the data distribution. Moreover, training ML models with good accuracy requires a massive amount of labelled data, which is expensive and time-consuming. New approaches were developed that can reuse and adapt existing model(s). These techniques fall under the collective name of transfer learning (*TL*). More specifically, TL is a methodology to transfer knowledge learned from one model to another. For example, in the case of a model trained to detect a specific type of object from images, the knowledge it has gained can be transferred, via TL, to another model that detects a different kind of object from images. TL has achieved excellent results in many domains, including image processing and NLP. When reusing ML models, understanding data distribution between the two domains is essential. The difference between the distributions may result in lowering the model accuracy.

7 MODEL ADAPTATION, DEPLOYMENT AND MAINTENANCE

7.1 MODEL DEPLOYMENT

Imagine that you developed an ML model which can forecast the likelihood of a particular outcome with a certain confidence score. It is a good step, but the process is not finished yet. In a perfect world, you want your model to evaluate real-time cases to be able to make decisions accordingly. This is where model deployment comes in. The deployment and maintenance of ML models is one of the most critical challenges that organisations face today. Numerous data science projects fail to get into production because of the many strains in the deployment phase which obstruct the entire process. These challenges originated in the dynamicity and complexity of the environment where heterogeneous and diversified components interact with one another in such contexts.

For applying ML in practice, it is a critical task to evaluate and adapt the existing, established software engineering practices that ML literature has thus far not taken into proper consideration. Actually, ML deployment is a topic that is completely unrelated to data analysis, model selection and model evaluation; therefore, it is not properly received by those without a proper background in software engineering. Recently, the interest in establishing substantial methods and practices in the development of ML systems has started to grow. The deployment of an ML model is referred to as the process of integrating an ML model with an existing production environment in which an input is processed to generate an output. The purpose of deploying a model is to enhance the ability to make predictions through the use of the model which is already trained and made available to different systems. The ML deployment is strongly dependent on the architecture of the overall system, the set of iterations and configurations of its software components. There are a couple of requirements that need to be satisfied before a model may be considered ready for deployment:

- **Portability** deals with the ability of the model, in terms of software and hardware, to be transferred from one system to another. This peculiarity will be important in evaluating where the system will be deployed.
- **Scalability** has to do with the system's ability to grow over time. A model is said to be scalable when it does not require modifications and rede-

sign to maintain performance unaltered.

A similar parallelism exists with cloud workloads, where companies must decide to deploy on premise, in the cloud or adopt a hybrid approach; the same decision must be made for ML algorithms. It is not an easy choice, as the physical location where ML algorithms are trained and deployed can affect their performances as much as the algorithm itself.

7.1.1 Cloud-based Deployment

The use of the cloud for the deployment of ML models is the most common method. Cloud platforms offer a range of services that developers and data scientists can use to design, train and deploy ML models. Furthermore, having this support available allows for the transition to an open and flexible environment where data can be rapidly collected from the cloud storage, prepared and injected directly into the model. The downside of this approach is that it could be extremely demanding to move the data from where they are generated into the cloud storage where they are used for developing and training the model. Sometimes it is difficult if not impossible to move large amounts of data in a centralised repository due to high latency, bandwidth limitations and excessive costs. In modern contexts where an increasing number of devices are connected to each other and hundreds of gigabytes – if not terabytes – are generated per day, those issues represent a real challenge that can force the developers to make unwanted compromises. In fact, the transferring into the cloud of a selected data 'sample' for the model training and maintenance is often the only alternative when the transfer of a large amount to a centralised data centre is not a reasonable choice. As a result, the efficiency in obtaining reliable information from newly generated data in real time is restrained and the ability to analyse a combined data set comprehensive of newly generated and historical data is diminished. Furthermore, the moving of data across jurisdictions can increase privacy and geopolitical concerns.

7.1.2 Edge-based deployment

It is estimated that the daily amount of data generated from personal or enterprise IoT devices is around 250 petabytes, and as mentioned in previous sections, the latency of the network can have a severe impact, especially for IoT devices. As an example, autonomous vehicles need to collect and analyse a massive amount of data from their own sensors and from the nearby devices. If the reaction time of the vehicle depends on the response time from the computing core of the network, every

minimum delay could cost lives. For those specific cases it is mandatory to adopt a different approach. Edge computing lends itself perfectly to managing these data as it provides a sufficient incentive to collect and process data on the devices themselves rather than in the cloud or in a remote data centre. In short, the key benefits for this approach are two: real-time analysis of data and reduced data transmission to the cloud. As a result, IoT devices are less affected by latency and have faster reaction to status changes. In this context, edge computing can provide predictive analytics on the edge devices.

One of the options for businesses is to design and train ML models in the cloud and upload the algorithm onto the device to execute close to where data are generated. This method guarantees the flexibility and simplicity of developing in the cloud and the efficiency of running directly in the proximity of the source data.

A second option is the development and training of the ML method on the device itself, using the collected data passing through. The system relies on intelligent edge devices providing the required functionalities and interfaces for development.

These approaches partially solve the problem of latency, but they also present accuracy problems. Algorithms developed on the cloud and brought to the edge are based solely on data samples, while developing accurate ML models only on the edge could be burdensome because those devices are designed and optimised to work with minimal resources and low power; moreover, the amount of data they can store for analytics can be tight.

A possible solution could be the hybrid approach, allocating computing resources at the edge. Those resources will reside in the near proximity of the devices but not within; this will ease the burden of transferring data from a centralised system. The data analysis could be performed in a distributed way by installing computing resources and storage in the proximity of the devices, such as schools, hospitals, banks and others. This approach offers the most complete solution, allowing the analysis of a nearly unlimited amount of data of any age and without restrictions against crossing different geopolitical areas. It follows that this model can bring the advantages of big data directly to edge computing, and, at the same time, it allows the application of ML algorithms on distributed data on all edge-devices in parallel.

7.2 MODEL MAINTENANCE

Data are the most important part of the ML model. Once the model fits perfectly into the available data set and provides accurate predictions, it is essential to ensure that the system continues to do so over time, with the most up-to-date data. As an example, with an ML model for predicting the real estate market, house prices are read frequently over time, so considering a model that was trained one year earlier could provide very inaccurate predictions when used for current market data. In this case, it is imperative to have up-to-date information for the new training. When designing an ML model, it is important to understand how and how often the data will change over time; a carefully designed system takes this into account before deployment to ensure an easy and smooth model upgrade.

7.2.1 Model Drift

An ML model that is running using static data (i.e., data whose statistical characteristics do not change over time), does not suffer a loss in performance because the data that are used for predictions belong to the same distribution as the data used for training. Unfortunately, in most real-world cases, the model exists in a dynamic environment and so is subject to revision. In this scenario, a concept drift is the performance decay of an ML model; at the origin of this well-known concern is a change in the environment that breaches the initial hypothesis of the model rather than a contraction in the capabilities of the model itself. A model degrades over time due to various factors and variables, depending exclusively on the context in which the algorithm works. The model performances decrease over time, and, with a certain rate, both are difficult to predict in advance. For this reason, it is essential to keep all these factors in mind when diagnosing the problem and determining the most effective method for retraining the model. It is important to understand how to track the drift. There are several approaches, but, depending on the data and the prediction algorithm, not all the solutions may be suitable for a specific case. The most intuitive way to identify drift is to explicitly determine that the model performance is impaired and to try to quantify the deterioration. Measuring model accuracy on live data can be problematic, as it is necessary to access both the predictions generated and the ground truth values (i.e., the information generated by direct observation), but predictions may not be stored or those observations may not be available.

Another way to address the problem is to infer the drifting. Because a decay of the model is expected due to the deviation of the serving data with respect to the training model, a comparison between these two distributions can give an estimate of the drift. This solution is particularly useful when it is not possible to extrapolate the ground truth from direct observation due to the nature of the data generated.

7.2.2 Model Retraining

We have defined the model drift and how to recognise it; now we need to understand what the next step is.

Usually, a model that has been deployed for production should be the result of a rigorous validation process; the resulting algorithm is the best prediction method for a given type of data. Since the performance in predictions decays due to a variation of target data, the model's retraining should not involve any changes in the model generation process. In fact, it is simply a matter of relaunching the process that generated the first instance, but on a new training set. The solution is new training performed on a new data set that reflects the evolution of the environment and the current reality. At this point, it is necessary to understand when to train and which new data set to use. The problem itself could directly provide an answer to the previous questions. Suppose you want to develop a model that generates predictions about the university courses students enrol in. This model can be run on students currently attending the last year of high school in order to prepare the entrance tests for the various universities. This kind of prediction must be done annually; it would not make sense to repeat the process more frequently as the data available for a new training would not exist. Therefore, we can only decide to retrain our model at the beginning of the academic year when we have the new enrolment data. This is an example of periodic retraining. In general, sudden changes in training data require retraining often, even daily or more. Slower variations will require monthly or even annual training. Following are a few approaches regarding methods that can be adopted in different contexts:

- *ONE-OFF*. This method is used when a continuous retraining of the model is not required but can be done periodically. In this case the model is put back into production once an ad-hoc training is carried out and the model stays in place until becomes obsolete again.
- *BATCH*. This method allows you to have a constantly updated version of the model. The model is updated with a subset of the data at

a time without necessarily using the entire set at each update. This method can be used when the model is frequently used but does not necessarily require real-time responses.

- *ONLINE (real-time):* Real-time training is possible with online ML models. The model is trained at every data set submission and is expected to provide a prediction for this data set in (near) real time.

The privileged option is to have an automatic drift management mechanism in case there is availability of technologies and infrastructures for monitoring the parameters discussed in the previous section. This operation requires continuous monitoring and a mechanism for triggering the training process whenever the diagnostics on the active data diverge from those of the training data. Obviously, the challenge is to determine the threshold value for the divergence between the two sets. If this value is too low, there is a risk of having too frequent retraining without benefits and with high costs. On the other hand, if the value is too high, the risk of delaying the retraining, which results in a model that does not perform in production, may increase. A further intrinsic problem is in determining the correct amount of training data to be supplied to faithfully represent the changes in the environment being observed. In fact, even if the world has changed, it could be counterproductive to replace the previous training data set with a considerably smaller one in the absence of additional data.

7.3 DATA DISCLOSURE RISKS AND DIFFERENTIAL PRIVACY IN MODEL DEPLOYMENT

We now informally discuss some data disclosure risks that arise when outsourcing ML models, e.g. by deploying them at the premises of a service provider, who could gain information from the input data or guess the information originally used for training the ML model. To better understand the notion of training set disclosure, we go back to the simple example and mathematical notation we used to describe training: a model F_w trained on a training set S for classifying the items of a data space DS into classes of interest belonging to a set C. This deployment procedure involves a disclosure risk whenever S can be inferred from F_w outputs. Disclosure will happen if by running or observing F in production, an attacker can reconstruct one or more entries of the training set S.

One could be tempted to require that computing F in production (i.e., performing the inference) should reveal absolutely nothing about the training set f. This is unfortunately just a re-phrasing of the classic *Dalenius requirement* for statistical databases, which cannot be fully achieved if enough side information about S is available. However, Cynthia Dwork proposed more than a decade ago the notion of *differential privacy*, which, intuitively, captures the disclosure risk. Dwork's seminal work has turned the "impossible" Dalenius requirement into an achievable goal: observing the execution of F_w, the service provider should be able to infer the same information about an entry $e \in f$ as by observing F'_w, obtained using the training set $S - \{e\} + \{r\}$, where r is a random entry. This will provide the owner of e - assuming she has something to gain by knowing the result of F - with some rational motivation for contributing e to the training set, as she will be able to deny any specific claim on the value of e that anyone could put forward based on F (a notion called *plausible deniability*). The most investigated approach to achieving differential privacy consists in introducing a degree of randomization in the computation of F, making $[F(x)]$ a random variable over DS. Techniques vary on how and where to inject such randomization, depending on the nature of F_w.

Often, random noise is simply added to the training set, with zero average and a standard deviation $\sigma = \frac{1}{\varepsilon n}$,

While the discussion above provides a general idea of data randomization

to prevent disclosure, some additional remarks may be of interest for the mathematically-aware data manager. The probability density often used for such noise is the Laplace distribution:

$$p(z) = e^{\frac{-|z|}{\sigma}} = e^{-|z|\varepsilon} \qquad (3)$$

The distribution of this random variable is "concentrated around the truth": the probability that $[F_w]$ is z units from the true value drops off exponentially with εz. This randomization introduces uncertainty, as the provider no longer computes F_w but the value of a random variable $[F_w]$ with Laplace distribution whose average coincides with F.

8 CASE STUDIES

8.1 CASE STUDY 1 : AUTOMATIC THE DETECTION OF TRAFFIC INCIDENTS

Traffic is known to be one of today's significant issues affecting large cities and one of the main challenges for smart cities. Reducing delayed notification time for traffic incidents has a direct impact on reducing the fatality rate and reducing the cost of roadside assistance. Traffic-related data are being collected en masse from traffic sensors deployed in big cities and, more recently, from social media like X and Weibo. As such, automatic detection of traffic incidents has attracted much interest from traffic control centres over the last decades. The goal is to detect the occurrence of an event causing traffic congestion, including recurrent and non-recurrent congestion. This interest is driven by the need to develop automated incident detectors – an asset in traffic management, as they can make appropriate and timely decisions based on the analysis of collected real-time data.

8.1.1 Data ingestion

The data used in the case study are at the macroscopic level; they are stored in a data hub which receives sensors' data from a road section containing multiple lanes going in the same direction (i.e., a *road link*) and averages the traffic variables for every two minutes. Since the data are aggregated, there are no privacy issues at the individual level. Table 1 shows an example of a reading stored in the data hub.

Table 1:
Traffic Data Structure

Data Structure	101000701,2017-01-01 00:01:05+01,98,120,19.6,0,46
ID of road network link	101000701
Data/time of measurement	2017-01-01 00:01:05+01
Average speed (km/h)	98
Vehicle flow (Vehicles per hour)	120
Headway(average time between vehicles, in seconds)	19.6
Occupancy (percentage of time that road is occupied by vehicles)	0
Travel time for this link, in seconds	46

A road link is positioned between two points in the road and has a single direction, clockwise or anti-clockwise, and the distance each road link covers is different. Junctions may contain one or more links, each one covering a short distance, while road links connecting junctions cover a longer distance.

8.1.2 Data exploration and pre-processing

The data received from the data hub still have certain limitations. There are periods where some road links have missing readings, making the detection of events rather difficult. Also, some readings contain missing data (e.g., occupancy data tend to be missing more often than speed data). Missing values are, however, typical of reproduction (as opposed to synthetic or lab-based) sensor networks.

In preprocessing, infrequent missing values are filled in by averaging between the preceding and the following readings. Detection from locations with an increasing number of missing values is not considered.

We consider automatic event detection a classification problem, which requires labelled data to train the ML model. A list of officially reported traffic events during a single period could be used to provide the ground truth for model training. Therefore, reported events from the period of the traffic readings are collected. Any event that had no effect on road conditions is discarded, as was any event whose readings were irregular or unavailable. For each reported event, the area where it took place (encompassing multiple road links), a single starting time and expected end time are specified.

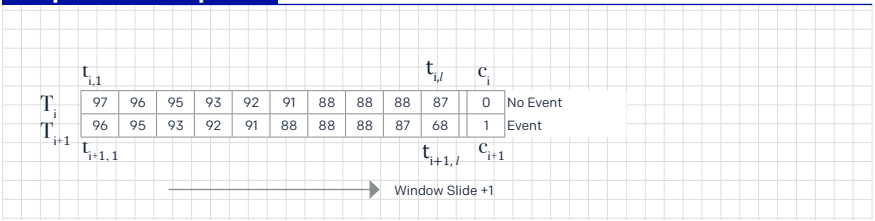
Moreover, it is not feasible to use the full data set for training and testing, as most of the data set will contain no events. A common practice in the field of traffic incidents detection is considering the readings of a certain number of hours (e.g., two hours) before and after an event. This will ensure capturing all the events and the different variations from non-event readings.

We remark that events include – besides accidents – slow-downs due to traffic congestion; both start and end times are approximate.

Regarding the event's location, the road link that is most affected by the event is chosen. We get this information from the speed values along with the duration of the event. A manual check of all events used for

training and testing is also conducted to adjust their timings. We remark again that using the reports without any adjustments could result in inaccurate training and test sets. This problem is commonly encountered in traffic data analysis and has been reported multiple times in the field.

Figure 14:
Temporal Data Sequence



To ease the task of preprocessing the data set, we developed a tool to select the start and end time of an event visually. The tool shows a plot of speed values against time during the targeted period and, if available, a similar plot for road occupancy.

Data Types

The dataset can be created as a temporal time series by taking traffic values (e.g., speed) in sequential time periods. Also, it can be created as a spatial sequence where the traffic values can be taken from neighboring road links at the same time. Another option is to combine both sequences. Part of the project was investigating the best approach.

Time series length

One of the variables that should be set by us is the fixed length of the time series. In the AID domain, we receive readings in fixed periods, which provides us the ability to create a time series of any length. As presented in Figure14, each time series is a continuation of the one preceding it with a sliding window of 1 which mimics a real time AID problem. By assuming a real time problem, we assume no knowledge of an event's duration; hence, we are limited to selecting the most suitable length which provides the best performance. After selecting the most suitable length, there is no added value in using other lengths.

8.1.3 Model Training, Deployment and Maintenance

After obtaining data for training from the data hub, preprocessing them and adding the labels, it is possible to start training the ML model. A variety of algorithms can be used, and at this stage the data scientist can explore and test the various options and compare their results using specific criteria. The criteria included overall accuracy, incidents detection rate and false alarm rate. By comparing multiple algorithms, rotation forest did yield the best results, justifying its use to create the ML model.

A system deployed in the cloud is made ready to accommodate the ML model by pulling the readings from the data hub, preprocessing the data and then feeding them to the model, which outputs the possibility of an event happening. The result of the ML model is exported to the data hub to be matched with the related reading. An external dashboard is also connected to the data hub to enable the traffic control centre to view the readings and any potential events.

There is also a feedback mechanism to tell the system when the predictions were not accurate. The feedback is used for the periodic retraining of the model to increase the performance with time.

8.2 CASE STUDY 2: SOCIAL MEDIA (X) ANALYSIS

The growth in social media applications and the exponential rise in their use have led to the accumulation and availability of a huge number of social media data that cover a variety of topics that are important to the platforms' users and beyond. Discussion topics on social media platforms can provide news, opinions, views, feedback and more on a wide range of subjects from politics and the economy to reviews and feedback on products and services. The opinions expressed and the sentiments associated with them can provide valuable insight into the feelings of communities and individual users. This information can be valuable to businesses, customer service and even government organisations that serve and support their communities.

As a result, the harvesting and analysis of social media content such as X posts have become extremely valuable to many organisations to help them better understand their users and communities and indeed understand their sentiments and views, almost in real time.

One of the first steps towards achieving this objective is to harvest the data; in this case, we will look particularly at X. The platform has a large online community where opinions and sentiments are expressed in as close to real time as possible. Anyone can have an account on X; many news agencies, journalists, politicians, TV broadcasters, businesses, government organisations and more use X as a means of communicating with their constituencies, users, customers and friends. Hence, X posts offer a huge opportunity to analyse and better understand certain communities. X enables us to detect their interest, as well as changes in that interest, allowing us to adjust for user, customer and community needs quickly.

8.2.1 X posts harvesting

X offers APIs at different price points to enable the collection of X posts and additional related data, based on specific terms and conditions about how the data can be used and what controls and restrictions apply. The different types of developer account pricing provide different levels of access in terms of the amount of data that can be collected over a certain time period. There are also restricted but free methods to access a limited number of X posts. It is important to adhere to the X terms and conditions and use policies to avoid discontinuation of service or even additional actions against any misuse.

Developers can write software tools referred to as harvesters, to collect relevant X posts and any additional information, such as author ID, re-posts and location information. The X posts can be harvested by area, keyword or author, among others. The harvester can run periodically and collect data in either micro- or macro-batches depending on user requirements and the number of X posts being collected. Once the data are collected they can be stored in the relevant storage infrastructure. There are also rules and regulations that apply to the archiving and use of X posts.

8.2.2 Classification

Classification of social media messages (such as X posts) is one of the most fundamental and useful data analysis techniques and can be used for many applications. Uses include, but are not limited to the following:

- *Sentiment analysis on customer service*. Can automatically classify short messages as positive, negative or neutral. This helps to automatically monitor sentiment changes about products and customer services. This sentiment analysis can also be more refined, classified to more detailed categories and providing richer information (e.g., social media messages classified as outraged, angry, upset, unhappy, neutral, happy, thankful, satisfied, excited, etc.).
- *Message filter.* Can automatically filter out irrelevant messages and ignore them. For example, if Company A would like all the social media messages relevant to it, a keyword-based search will return messages including those keywords. However, not all messages including these words are relevant to Company A. A filter will classify those messages into one of the two categories: relevant or irrelevant.
- *Topic classification.* Classifies all relevant social media messages we might want to know more about (e.g., whether they are talking about fault for services/connection, complaints about wrong bills, inquiries about new products, or recommendations to friends about the good service). This task can be done by message classification, as well.

To realise classification, three stages are involved: training the model, testing the model and applying the model on the fly.

- **Model training** is the process to build up a classifier model using the training data. The training data must be tagged manually with target

categories (e.g., positive or negative in sentiment analysis; relevant or irrelevant for a filter). During the model training process, the model extracts and learns the patterns from the tagged messages, and the trained model will be tested and used in the later stages.

- **Model testing** is the process that ensures the generated model from stage one satisfies our requirements, mainly in terms of accuracy (e.g., the model can classify 95% of the messages to the target categories correctly). This is done by applying the generated model to the testing data. The tested data need to be manually tagged, as well, to compare the model output (as category) with the target output (tagged manually). This testing process might happen iteratively together with the training process. We obtain a model from the training data and test it on the testing data. If the accuracy is lower than our expectation, we might need more training data to refine the model, or we might consider using another technique of classification to see whether other techniques can complete this classification task better. This iterative process continues until we reach a satisfactory accuracy.
- **Applying the model** means using the satisfied model on real-world data and generating classification results automatically.

As we mentioned in the testing stage, a high accuracy might be our main objective to obtain a good classifier model. If we can not reach our expected accuracy, there might be two reasons. Either the training data is not enough in which case we need to increase the number of training data; or the classification technique we are using is improper to this application which we need to improve the classification technique. The outcome of the classification can be then be stored in the storage infrastructure as attributes of the original social media content (X posts) or any other format depending on the application.

8.2.3 Visualisation

Once the analysis has been completed, the results of the analysis must be displayed in an easy-to-understand format that provides the required insight for the application. The simplest way to show results can be to merely show statistics summarising the analysis outcome, such as the number of users mentioning a company or product or the number of satisfied customers (positive sentiment) and unsatisfied customers (negative sentiment). For further, more advanced visualisation, one can develop their own front-end visualisation to display custom views or use one of the many available visualisation capabilities and tools – open-

source or commercial. Existing visualisation tools enable the plotting of data such as time series or events or, indeed, using MAS and GIS systems. Visualisation tools can also have predefined and easy-to-use templates providing a plethora of options for almost every use. They also provide various customisation capabilities for more advanced requirements. Such tools are usually used for better understanding analysis, insight and decision support.

9 OTHER AI TECHNIQUES

Although very general, the AI-ML life cycle discussed in this document covers only a part (i.e., supervised ML) of the rich landscape of AI applications. Indeed, not all AI applications ultimately require ML. Historically, some AI techniques preceded it. When researchers started to investigate AI, they were focused on the so-called symbolic techniques like automated inference and reasoning; ML had not even occurred to them. An example of the use of AI without ML are rule-based expert systems. Human-defined rules allow expert systems to perform inferences to a limited extent. Another example is syntax-based NLP, where the syntax and semantics of natural language are encoded into algorithms (parsers) used to interpret and generate language. In 1985, this type of AI received the name of Good Old-Fashioned Artificial Intelligence (GOFAI). In many cases, symbolic reasoning is sufficient, especially if a large amount of domain knowledge is made available in the form of rules. Another advantage of using the symbolic form of AI instead of ML or DL is that there is no black box problem. As we discussed in Section 1.7, ML-based decision-making may lack transparency, and bias hidden in the training data may lead to unfair decisions. Besides symbolic techniques, other areas of AI rely on statistical or algorithmic techniques rather than on ML. In this section we quickly review some non-symbolic AI techniques like time series forecasting and optimisation that do not rely on ML but still need sound data management practices. Our introduction will not cover other important meta-heuristic techniques that come under optimisation, such as evolutionary algorithms, tabu search, simulated annealing or swarm intelligence. The interested reader should consult the technical literature.

9.1 PREDICTING TRENDS FROM DATA

Many natural phenomena vary with time in a non-trivial way. Applications may need to find out the behaviour of such phenomena in the short or medium term. This kind of analysis is known as time series forecasting. For instance, one may want to predict, with some approximation, the trend of the air cargo market over the next six months. It is possible to address this

problem by observing the behaviour of the phenomenon in the past: if its

behaviour in a specific time period is correlated to the behaviour of the following time period, then this insight can be used for prediction. For instance, looking at the water level of the River Nile across time has been used for centuries to predict forthcoming floods and droughts. Moreover, different phenomena are often correlated to one another: one can observe, for instance, that the trend of the cargo market follows the trend of the global economy with just a few months of delay. Thus, the global economy can be used to predict the behaviour of the cargo market. This is an example of how knowledge of correlations among different time series can help forecasting. The difficult part in forecasting in the mentioned examples is the discovery of significant enough correlations that are hidden in the data. Noisy data are particularly difficult to deal with. Sifting through large volumes of data in search for regularities thus requires powerful data-mining techniques. There are multiple models and methods used as approaches for time series forecasting. The simplest series forecasting setting is the univariate time series forecasting problem, where data contain only two variables: time and the value to be forecast. In the multivariate time series forecasting method, forecasting problems contain multiple variables, one of which is again time. Modelling techniques include the following:

- *ARIMA model*. ARIMA is a combination of three different models, AR, MA and I, where 'AR' reflects that the evolving variable of interest is regressed on its own prior values, 'MA' states that the regression error is the linear combination of error term values from preceding times, and 'I' expresses the fact that data values are replaced by the difference between themselves and the previous values.
- *Autoregressive conditional heteroscedasticity (ARCH) model:* The ARCH model is the most volatile model for time series forecasting, capable of catching dynamic variations of volatility from time series.
- *Autoregressive model or VAR*: This model computes the independencies between various time series data as a generalisation of the univariate autoregression model.

From the data assets' point of view, time series analysis requires data ingestion at regular intervals, which is best implemented via stream platforms for big data.

9.2 RETRIEVING THE RIGHT INFORMATION FROM LARGE REPOSITORIES

Due to the astronomical growth in data generation and the move towards open data initiatives, there is now an enormous number of data sources available to choose from. However, due to the myriad of choices, it becomes impossible to select what is relevant to the problem at hand. For instance, the internet consists of billions of pages of information, and it would be impossible to find the exact information you are looking for without the help of search engines to rank the pages in order of relevance.

Those engines use the links pointing from one page to another: they are based on the premise that the pages with the most references from highly referenced pages are the most relevant. Ranking algorithms of this kind can be used to retrieve, by relevance, documents from any repository and to extract value from the information, instead of drowning in the deluge of data. Information retrieval techniques can be used with all kinds of data repositories: text documents, images and videos.

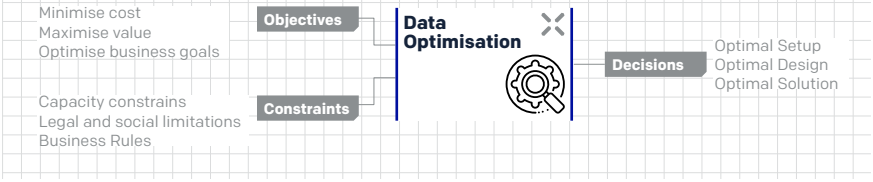
9.3 A.I. OPTIMIZATION AND DATA

Optimisation is an important branch of AI that is often overlooked. It revolves around making optimal decisions, and data play an important role in that. We make decisions every day: for example, choosing the best route to commute, buying better-value products or investing to get maximum returns. This is also the case for businesses and large organisations. A logistics business with a large fleet of vehicles, for instance, needs to make a daily decision about optimal vehicle routing; optimisation in this setting means minimising travel time and maximising the number of deliveries. Sub-optimal routing may result in unnecessary travel and poor customer service. Similarly, in teleco, choosing the best sites to deploy mobile cells is important. Deploying mobile cells in less-ideal locations can have negative cost and revenue implications. This concept generalises to many other sectors and businesses. Logistics, hospitality, airlines, oil and gas, health, education, and government – they all have to make the best possible decisions to run their organisations efficiently, provide the best service to their customers, minimise costs and maximise benefits. In the example of a logistics business, it may be trivial to find the best routes for an organisation with only one or two vehicles. However, with a larger fleet, the possible combination of routes becomes infinitely large, and the routing complexity increases: the decision about where to route one vehicle can depend on other vehicles' potential routes, making it impossible to utilise a manual routing process to find the best options. AI optimisation intelligently searches through the complex set of options and provides optimal (or near-optimal) decisions to implement.

As we have seen in the previous chapters, data availability is an important factor in decision-making and should be given careful consideration when formulating an optimisation model, as well. There are three main components of an optimisation model:

1. Decision variables
2. Objectives

Figure 15:
Othe AI Techniques



3. Constraints

In our example of the logistics business, decision variables define a set of routes that have to be optimised. The two potential objectives are to minimise total travel and maximise delivery volume. And the potential constraints are the capacity of each vehicle and their hours of operation. A clear input data about the domain is crucial to define each of these components. For example, location data about the customers as well as street-level distances are required to specify the routes. Real-time or historical data about traffic and any ongoing construction works could also be useful to calculate the expected travel.

Finally, data about the vehicles' size and speed as well as drivers' roster data are required to calculate the constraints involved in optimising routes. In many real-world scenarios, data are created with the mindset that they will be interpreted by humans, often lacking the required details and rigorousness to be processed automatically. To capitalise on the data for AI optimisation, it should be carefully created to enhance the entire decision-making process.