INTERNATIONAL STANDARD

ISO/IEC 23053

First edition
2022-06

Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML)

Cadre méthodologique pour les systèmes d'intelligence artificielle (IA) utilisant l'apprentissage machine



Reference number
ISO/IEC 23053:2022(E)

© ISO/IEC 2022



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2022
All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO/IEC 2022 – All rights reserved

Contents

Page

Foreword ... **iv**

Introduction ... **v**

1 Scope ... **1**

2 Normative references ... **1**

3 Terms and definitions ... **1**
3.1 Model development and use ... 1
3.2 Tools ... 2
3.3 Data ... 2

4 Abbreviated terms ... **3**

5 Overview ... **4**

6 Machine learning system ... **4**
6.1 Overview ... 4
6.2 Task ... 5
6.2.1 General ... 5
6.2.2 Regression ... 6
6.2.3 Classification ... 6
6.2.4 Clustering ... 6
6.2.5 Anomaly detection ... 6
6.2.6 Dimensionality reduction ... 7
6.2.7 Other tasks ... 7
6.3 Model ... 7
6.4 Data ... 8
6.5 Tools ... 9
6.5.1 General ... 9
6.5.2 Data preparation ... 9
6.5.3 Categories of ML algorithms ... 10
6.5.4 ML optimisation methods ... 14
6.5.5 ML evaluation metrics ... 16

7 Machine learning approaches ... **19**
7.1 General ... 19
7.2 Supervised machine learning ... 20
7.3 Unsupervised machine learning ... 22
7.4 Semi-supervised machine learning ... 23
7.5 Self-supervised machine learning ... 23
7.6 Reinforcement machine learning ... 23
7.7 Transfer learning ... 24

8 Machine learning pipeline ... **25**
8.1 General ... 25
8.2 Data acquisition ... 26
8.3 Data preparation ... 27
8.4 Modelling ... 28
8.5 Verification and validation ... 30
8.6 Model deployment ... 30
8.7 Operation ... 30
8.8 Example machine learning process based on ML pipeline ... 31

Annex A (informative) **Example data flow and data use statements for supervised learning process** ... **34**

Bibliography ... **36**

© ISO/IEC 2022 – All rights reserved

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 42, *Artificial Intelligence*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

© ISO/IEC 2022 – All rights reserved

Introduction

Artificial intelligence (AI) systems, in general, are engineered systems that generate outputs such as content, forecasts, recommendations or decisions for a given set of human-defined objectives. AI covers a wide range of technologies that reflect different approaches to dealing with these complex problems.

ML is a branch of AI that employs computational techniques to enable systems to learn from data or experiences. In other words, ML systems are developed through the optimisation of algorithms to fit to training data, or improve their performance based through maximizing a reward. ML methods include deep learning, which is also addressed in this document.

Terms such as knowledge, learning and decisions are used throughout the document. However, it is not the intent to anthropomorphize machine learning (ML).

This document aims to provide a framework for the description of AI systems that use ML. By establishing a common terminology and a common set of concepts for such systems, this document provides a basis for the clear explanation of the systems and various considerations that apply to their engineering and to their use. This document is intended for a wide audience including experts and non-practitioners. However, some of the clauses (identified in the overview in Clause 5), include more in-depth technical descriptions.

This document also provides the basis for other standards directed at specific aspects of ML systems and their components.

© ISO/IEC 2022 – All rights reserved

Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML)

1 Scope

This document establishes an Artificial Intelligence (AI) and Machine Learning (ML) framework for describing a generic AI system using ML technology. The framework describes the system components and their functions in the AI ecosystem. This document is applicable to all types and sizes of organizations, including public and private companies, government entities, and not-for-profit organizations, that are implementing or using AI systems.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 22989, *Information technology—Artificial intelligence — Artificial intelligence concepts and terminology*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 22989 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1 Model development and use

3.1.1
classification model
<machine learning> machine learning model whose expected output for a given input is one or more classes

3.1.2
regression model
<machine learning> machine learning model whose expected output for a given input is a continuous variable

3.1.3
generalization
<machine learning> ability of a trained model to make correct predictions on previously unseen input data

Note 1 to entry: A machine learning model that generalizes well is one that has acceptable prediction accuracies using previously unseen input data.

Note 2 to entry: Generalization is closely related to overfitting. An overfit machine learning model will not generalize well as the model fits the training data too precisely.

© ISO/IEC 2022 – All rights reserved

3.1.4
overfitting
<machine learning> creating a model which fits the training data too precisely and fails to generalize on new data

Note 1 to entry: Overfitting can occur because the trained model has learned from non-essential features in the training data (i.e. features that do not generalize to useful outputs), excessive noise in the training data (e.g. excessive number of outliers) or because the model is too complex for the training data.

Note 2 to entry: Overfitting can be identified when there is a significant difference between errors measured on training data and on separate test and validation data. The performance of overfitted models is especially impacted when there is a significant mismatch between training data and production data.

3.1.5
underfitting
<machine learning> creating a model that does not fit the training data closely enough and produces incorrect predictions on new data

Note 1 to entry: Underfitting can occur when features are poorly selected, insufficient training time or when the model is too simple to learn from large training data due to limited model capacity (i.e. expressive power).

3.2 Tools

3.2.1
backpropagation
neural network training method that uses the error at the output layer to adjust and optimise the weights for the connections from the successive previous layers

3.2.2
learning rate
step size for a gradient method

Note 1 to entry: Learning rate determines whether and how fast a model converges to an optimal solution, making it an important hyperparameter to set for neural networks.

3.3 Data

3.3.1
class
human-defined category of elements that are part of the dataset and that share common attributes

EXAMPLE "telephone", "table", "chair", "ball bearing" and "tennis ball" are classes. The "table" class includes: a work table, a dining table, a study desk, a coffee table, a workbench.

Note 1 to entry: Classes are typically target variables and designated by a name.

3.3.2
cluster
automatically induced category of elements that are part of the dataset and that share common attributes

Note 1 to entry: Clusters do not necessarily have a name.

3.3.3
feature
<machine learning> measurable property of an object or event with respect to a set of characteristics

Note 1 to entry: Features play a role in training and prediction.

Note 2 to entry: Features provide a machine-readable way to describe the relevant objects. As the algorithm will not go back to the objects or events themselves, feature representations are designed to contain all useful information.

© ISO/IEC 2022 – All rights reserved

3.3.4
distance
<machine learning> measured proximity of two points in space

Note 1 to entry: Euclidean, or straight-line, distance is ordinarily used in machine learning.

3.3.5
unlabelled
property of a sample that does not include a target variable

4 Abbreviated terms

AI	artificial intelligence
API	application programming interface
AUC	area under the curve
BM	Boltzmann machines
CapsNet	capsule neural network
CG	conjugate gradient
CNN	convolutional neural network
DBN	deep belief networks
DCNN	deep convolutional neural network
FFNN	feed forward neural network
FNR	false negative rate
FPR	false positive rate
GRU	gated recurrent unit
LSTM	long short-term memory
MAE	mean absolute error
MDP	Markov decision process
ML	machine learning
NN	neural network
NNEF	neural network exchange format
NPV	negative predictive value
ONNX	open neural network exchange
PCA	principal component analysis
PHI	personal or protected health information
PII	personally identifiable information
PPV	positive predictive value

© ISO/IEC 2022 – All rights reserved

REST	representational state transfer
RNN	recurrent neural network
ROC	receiver operating characteristics
SGD	stochastic gradient descent
SVM	support vector machine
TNR	true negative rate
TPR	true positive rate

5 Overview

ISO/IEC 22989 defines ML as the process of optimising model parameters through computational techniques, such that the model's behaviour reflects the data or experience. Since the early 1940s, modelling of neurons (i.e. neural networks) and the development of computer programs that can learn from data have been explored. ML is an expanding field with the emergence of new applications in a wide array of industry sectors. This progression is enabled by the availability of large amounts of data and computation resources. ML methods include neural networks and deep learning.

In ISO/IEC 22989, an AI ecosystem is presented in terms of its functional layers and ML is a significant component of this AI ecosystem. Figure 1 illustrates the ML system which breaks down into the components of model, software tools and techniques and data.

Clause 6 in this document describes in further detail the different components of the ML system.

Clause 7 in this document describes different ML approaches and describes their dependency on training data.

Clause 8 in this document describes an ML pipeline: the processes involved in developing, deploying and operating an ML model.

Clauses 6.5 and 7 are more technical than the rest of the document. A stronger technical background can help the reader to better understand this content.

6 Machine learning system

6.1 Overview

Figure 1 depicts the elements of an ML system. They delineate the roles and their ML-specific functions that can be implemented by different entities (e.g. different vendors). The examples provided in Figure 1 are not meant to be an exhaustive list. Further explanation on each section of Figure 1 continues through Clause 6.

© ISO/IEC 2022 – All rights reserved

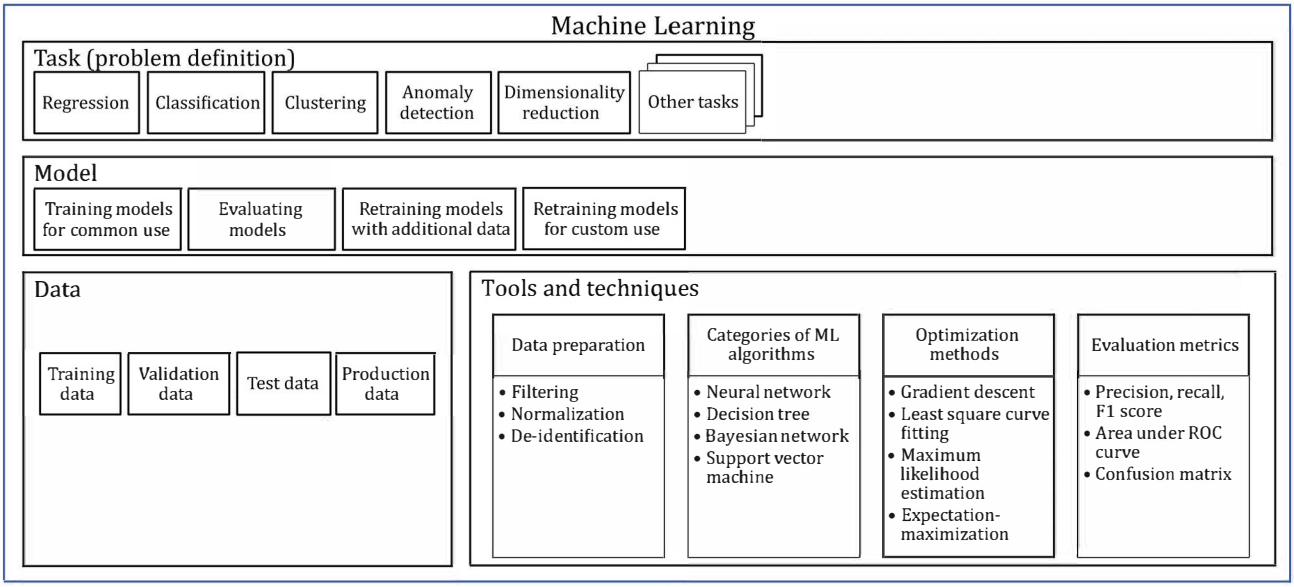


Figure 1 — Elements of an ML system

In Figure 1, the sub elements of model development and use can be considered as a layered approach, i.e. applications are built from models which are used to solve tasks. Model development and use in turn have a dependency on software tools and techniques and data.

A single ML system can be composed of several ML models used in combination. The system components can be described in terms of their input, output and their intent or function. The components can be tested independently.

ML models, when deployed, produce outputs such as predictions or decisions. A pre-trained model is an ML model already trained when it was obtained. In some cases, the developed model can be applied to a similar task, in a different domain. Transfer learning is a technique for modifying a pre-trained ML model to perform a different related task.

In this document, application refers both to the intended use of one or more ML models and to the concrete piece of software that implements that use. ML models are usually integrated with other software components to create applications. Applications using ML differ in the types of the input data they process and in the types of tasks they perform. In some applications, ML makes high-level predictions or decisions, while in other applications, ML provides answers to narrowly defined problems.

Differences in input data and tasks, as well as factors such as deployment options, accuracy and reliability, result in different application designs. AI applications can use proprietary custom designs or follow domain-specific design patterns.

Application logic is informed by the format of the input data, the output data, and potentially the transformation and the flow of data between the ML models in use. In all cases, the choice of ML algorithms and data preparation techniques is tailored to the application's tasks.

6.2 Task

6.2.1 General

The term "task" refers to actions required to achieve a specific goal. In ML, this implies identifying a problem to be solved using the ML model. One or more ML tasks can be defined for an ML application. Instead of solving a problem using a specific function represented as a set of steps and implemented in a software code, the defined problem is solved by applying a trained ML model to production data.

© ISO/IEC 2022 – All rights reserved

Effectively, the trained ML model implements a target function, which is an approximation of the hypothetical function that would have been written by a programmer to solve the problem.

An ML task setup involves defining the problem, the data format and the features.

The tasks described in the following subclauses are examples and are not exhaustive.

6.2.2 Regression

Regression tasks comprise predicting a continuous variable by learning a function that best fits a set of training data. In a regression task, the trained regression model represents a custom space. When the trained model is applied to a new production data instance, the instance is projected into the custom space defined by the trained regression model.

Regression is mainly used to predict numerical values of a real-world process based on previous measurements or observations from the same process. Use cases for regression include:

— predicting stock market price;

— predicting the age of a viewer of streaming videos;

— predicting the amount of prostate-specific antigen in the body based on different clinical measurements.

6.2.3 Classification

Classification tasks comprise predicting the assignment of an instance of input data to a defined category or class. Classification can be binary (i.e. true or false), multi-class (i.e. one of several possibilities) or multilabel (i.e. any number out of several possibilities). For example, classification can be used to predict whether an object in an image is a cat or a dog, or even from a completely different species. The classes are typically from a discrete and unordered set, such that the problem cannot be formalised as a regression task. For example, a medical diagnosis of a set of symptoms can be {stroke, drug overdose, seizure}, there is no order to the class values, and there is no continuous change from one class to another.

Use cases for classification include:

— Document classification and email spam filtering, where documents are grouped into several classes. A spam filter for instance uses two classes, namely “spam” and “not spam”;

— Classifying the species of a specimen. For example, an ML classification model can predict the species of a flower when provided with data that specifies the sepal length and width, and the petal length and width;

— Image classification. Given a set of images (e.g. of furniture), an ML system can be used to recognize and name the objects shown in those images.

6.2.4 Clustering

Clustering tasks comprise grouping input data instances. Unlike classification tasks, the classes are not predefined in clustering tasks but are determined as part of the clustering process. Clustering can be used as a data preparation step to identify homogenous data which can then be used as training data for supervised machine learning. Clustering can also be used to detect outliers or anomalies by identifying input data instances that are not like other samples. Example applications of clustering tasks include the sorting or organizing of files.

6.2.5 Anomaly detection

Anomaly detection comprise identifying input data instances that do not conform to an expected pattern. Anomaly detection can be useful for applications such as detecting fraud or unusual activities.

© ISO/IEC 2022 – All rights reserved

For anomaly detection, the ML model predicts whether an input data instance is typical for a given distribution.

6.2.6 Dimensionality reduction

Dimensionality reduction consists of reducing the number of attributes or dimensions per sample while retaining most of the useful information.

Dimensionality reduction can promote a dataset's most useful features and thereby mitigate computation costs.

Dimensionality reduction alleviates the various less-than-ideal effects of keeping too many features, collectively known as "the curse of dimensionality". Dimensionality reduction is also useful for data exploration and model analysis.

Methods for dimensionality reduction are unsupervised, supervised or semi-supervised[1].

6.2.7 Other tasks

There exist many other tasks which have different purposes and expected outputs. These tasks can be specific to a given application. Examples of other tasks include semantic segmentation of text or images, machine translation, speech recognition or synthesis, object localisation and image generation.

In planning, the task is to optimise a sequence of actions from an agent or agents through observing the environmental state.

Despite their diversity, a number of concepts have been formulated to draw connections between some of these other tasks. Structured prediction, corresponding to tasks in which the expected output of the model is a structured object as opposed to a single value, is one such concept.

Structured prediction requires computational methods that can account for regularities in the output, either by explicitly modelling them or by jointly predicting the whole structure with a model that internally models the regularities.

Use cases for structured prediction include:

— constructing a parse tree for a natural language sentence;

— translating a sentence in one language into a sentence in another language;

— predicting protein structure;

— semantic segmentation of an image.

6.3 Model

ISO/IEC 22989:2022, 3.2.11, defines an ML model as a mathematical construct that generates an inference or prediction, based on input data or information. The ML model comprises a data structure and software to process the structure, both determined by a chosen ML algorithm. The model is configured with inputs and outputs essential to solving the given problem.

The model is populated (also known as "trained") to represent the relevant statistical properties of the training data. Effectively, through the training process the model "learns" how to solve the problem for the training data with the goal to apply this acquired knowledge to a real-world application.

ML models produce results that are approximations of optimal solutions. ML algorithms utilise statistical optimisation methods to perform this approximation. The resultant mapping from the inputs to the outputs of the model reflects the patterns learned from the training data. Patterns can relate to correlations, causal relationships or categories of data objects. ML models are the result of the training data used. Thus, if the data used is incomplete, or reflects inherent societal bias, then the model performance will reflect this as well. Therefore, care should be taken with the datasets used for

© ISO/IEC 2022 – All rights reserved

training models. The logic created in the process of machine learning and represented by the trained model is not specified by a programmer but evolves during the training activities.

To see how well the model performs, it is evaluated using evaluation metrics.

Retraining consists of updating a trained model by training with different training data. It can be necessary due to many factors, including the lack of large training data, data drift and concept drift.

In data drift, the accuracy of the model's predictions decays over time due to changes in the statistical characteristics of the production data. In this case, the model needs to be retrained with new training data that better represents the production data.

In concept drift, the decision boundary appears to move, which also degrades the accuracy of predictions, even though the data has not changed. In the case of concept drift, the target variables in the training data need to be relabelled and the model retrained.

Retraining can also occur for purposes such as transfer learning and optimisation or modification of the ML model.

Some models can be readily available which can then be retrained and optimised for a specific use case or used as is. An example can be a commercially available machine translation model that can be retrained to be used for translating legal documents.

Continuous learning is a special case of retraining in that the model's performance is continuously evolving, resulting from on-going training of the model with production data. In such cases, it can be necessary to continuously monitor the model's performance or to implement "guard-rails" on the acceptable behaviour for the model.

6.4 Data

Figure 2 is a rake diagram showing that the data concept is partitioned into four mutually exclusive categories:

a) Training dataset, used to estimate the parameters of candidate models;

b) Validation dataset, also known as development dataset depending on the AI field (e.g. in natural language processing), used to select the best model according to a performance criterium;

c) Test dataset, used to check the generalisation capacity of a model and determines its performance on future data;

d) Production data, comprised of operational data to be used by the model for prediction. The distribution of production data can differ from that of training, validation and test dataset.

Training, validation and test datasets can be supplemented with simulated and perturbed data.

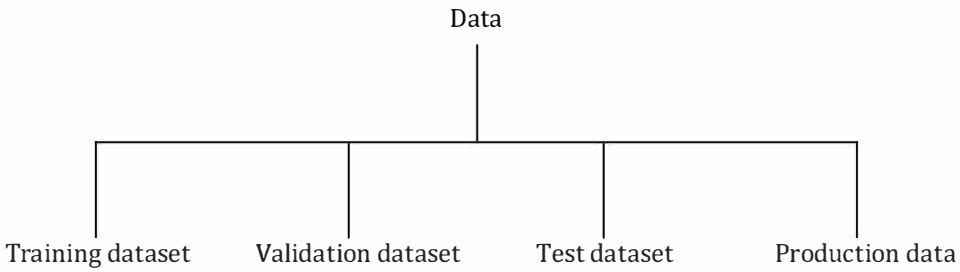


Figure 2 — Concept diagram: data and datasets

These various types of data can be comprised of either input data alone, or input data associated with labels (the expected output data). Validation and test data are often labelled, and production data is typically unlabelled. For training data, it depends on the ML approach: it can be unlabelled, partially

© ISO/IEC 2022 – All rights reserved

labelled, or fully labelled. Labelled training data allows the ML algorithm to identify statistical relationships between input variables and the target variable. Unlabelled training data allows the ML algorithm to identify statistical correlations and structure within the input data.

Validation and test data are both used with statistical performance measures (which are discussed in 6.5.5), but their uses differ: validation data is used to tune the hyperparameters, whereas test data is about evaluating the model. The aim of test data is to verify that the trained model will perform, or generalize, well on production data. Trained models that do not generalize well are called "overfitted" (to the training data).

Note that this use of the term test data is limited to ML-specific processes, and distinct from usage of the term in the context of verifying and validating an integrated system that uses ML components, in which case it would refer to any data used for verification and validation purposes, without a particular relation to ML. Note also that the ML-specific use of the term validation data is unrelated to verification and validation processes, as it pertains to model development.

For reliable application of the ML processes, training, validation and test data need to be disjoint. Training, validation and test sets can be obtained from the same dataset, using data splits, or they can be acquired separately. In an ideal configuration, they all have the same statistical distribution, but depending on the use case and ML approach it can be necessary to proceed differently.

For faithful evaluation of the trained model, the test data needs to have a distribution as similar as possible to that of production data.

Production data is seen by the model only after its deployment. For the model to make accurate predictions, in general the production data needs to have a distribution similar to that of the training and validation data, although specific techniques exist that can alleviate the degradation in case of discrepancy.

Over time, the production data distribution can drift which can require the model to be retrained on new data. In cases where the model needs to adapt dynamically to new patterns in the production data, the model can be continuously retrained by leveraging information gained from production data. Such cases are discussed in 6.3.

6.5 Tools

6.5.1 General

ML model creation uses tools categorized as data preparation, ML algorithms, optimisation methods and evaluation metrics. ML model performance is assessed through tools that generate evaluation metrics.

ML model creation often requires high-performance compute workloads due to computational demands and the use of large training datasets. Compute and storage performance can also affect how quickly ML models can be developed and trained.

Fundamental challenges with ML include statistical analysis, algorithm design and optimisation. Statistical analysis involves the principles of mathematical models derived from training data. Algorithm design is the manner of implementation of algorithmic techniques used to build the ML model. Optimisation of a given ML model is also an important issue in implementing ML. A further challenge is in understanding the potential and possibilities of ML. It relies on data and so will replicate, amplify and expedite existing faults and inequities in many cases.

6.5.2 Data preparation

Data preparation is discussed in 8.3.

© ISO/IEC 2022 – All rights reserved

6.5.3 Categories of ML algorithms

6.5.3.1 General

The choice of the ML algorithm defines the computational structure of the ML model and its training approach.

Algorithms can be used for different ML purposes, including:

— an information representation algorithm that can take part of the data preparation stage. This is related to feature engineering;

— an algorithm used in the creation of an ML model.

The relationship between ML algorithms and ML models can be illustrated by considering the solving of a univariate linear function $y = \theta_0 + \theta_1 x$ where y is an output, or result, x is an input, θ_0 is an intercept (the value of y where x=0) and θ_1 is a weight. In ML, the process of determining the intercept and weight for a linear function is known as linear regression. If a univariate linear function ($y = \theta_0 + \theta_1 x$) has been trained using linear regression, the resulting model can be $y = 3 + 7x$.

Figure 3 shows examples of various categories of ML algorithms.

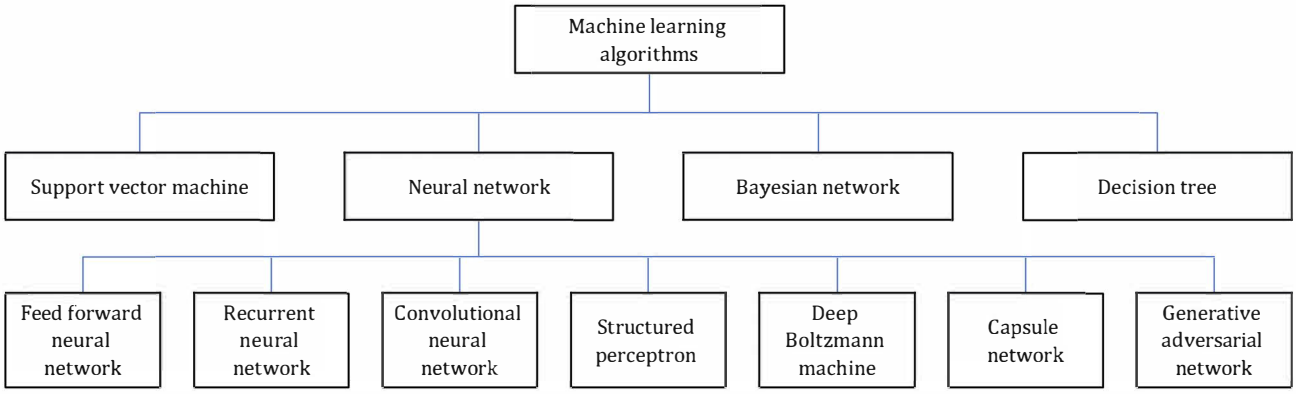


Figure 3 — Examples of various categories of ML algorithms

The choice of an ML algorithm is often not sufficient to define the structure of the ML model and its training approach. For many algorithms, hyperparameters shall also be chosen.

Hyperparameters are characteristics of an ML algorithm that affect its learning process. They can be used in processes to help estimate model parameters. Examples of hyperparameters for neural networks include the number of network layers, the width of each layer and the type of activation function. One practical approach to determine the optimal hyperparameter set among all possible combinations is to conduct random searches guided by a constraint function and measure the performance against a validation dataset. This step is called model selection, or hyperparameter tuning.

6.5.3.2 Neural network

6.5.3.2.1 General

Neural networks (NN) attempt to simulate intelligent capability in observing, learning, analysing and decision making for complex problems. Hence, the design of NNs draws inspiration from the way neurons are connected in the brains of humans and animals. They use multilayered networks of simple computing units (or "neurons"). In these NNs each unit combines a set of input values to produce an output value, which in turn is passed on to other neurons downstream. The structure of a NN is composed of interconnected processing elements. A network of nodes (or neurons) is connected by weighted edges. The simplest form of network consists of one or more input neurons that accept one or more features from the input data, as well as an output layer which includes one or more neurons.

© ISO/IEC 2022 – All rights reserved

Each neuron can accept one or more inputs from the previous layer and its output is given by an activation function that typically nonlinearly depends on the weighted combination of the inputs from the previous layer. The NN "learns" by training with known inputs, comparing actual output with the expected one and using the error to adjust weights.

Hidden layers can exist between the output and input layers of a NN, in which case it is called a multilayer perceptron or multilayer NN. Deep neural network or deep learning specifically refers to neural networks with many hidden layers. Deep learning is an approach to creating rich hierarchical representations through the training of NNs with many hidden layers. This process allows the neural network to progressively refine the final output. Deep learning can reduce or eliminate the need for feature engineering as the most relevant features are identified automatically.

NNs can be categorized into three general types: discriminative, generative and hybrid (combination of the two former types).

NNs can be considered as a connectionist approach. Connectionism uses a network of interconnected units which generally are simple computational units[2]. The behaviour of the network can be refined by modification of the weights associated with each connection which as described above is achieved through training. The nodes of the network process information in parallel.

Deep learning can require significant time and computing resources. ML NNs often require a considerable amount of computing power and memory to run. Reducing the model complexity can be beneficial for using these networks on mobile and embedded devices. The runtime impact and energy consumption can be minimised, and in some cases the reduction even enables the NN to be run in real-time on mobile devices without relying on cloud services. Compressing weights or squeezing the architecture can reduce the model complexity by a significant factor while maintaining almost identical performance.

6.5.3.2.2 Feed forward neural networks

Feed forward neural networks (FFNN) are the most straightforward NN architectures. They feed information from the input layer to the output layer in one direction only. There are no connections between the neurons within a particular layer. Two adjacent layers are typically "fully connected" in that each neuron in one layer has a connection to each neuron in the subsequent layer. Each connection has a weight associated with it. FFNNs are typically trained through the use of back-propagation with labelled training data, where each sample is labelled with the ground truth for the expected value of the output data. The difference between the actual output of the FFNN and the ground truth is termed the error[3]. Backpropagation involves using the error at the output layer to adjust the weights for the connections from the successive previous layers. It is often used together with gradient descent[4].

6.5.3.2.3 Recurrent neural network

6.5.3.2.3.1 General

Recurrent neural networks (RNN)[5] are NNs that aim to deal with sequential inputs that appear in an ordered sequence and where the ordering of the inputs in the sequence matters. Examples of such inputs include dynamic sequences like sound and video streams, but also static sequences like text or even single images. RNNs can in principle be used in many fields as most forms of data that do not have a timeline (i.e. unlike sound or video) can be represented as a sequence. A picture or a string of text can be fed one pixel or character at a time, so the time dependent weights are used for what came before in the sequence, not actually from what happened earlier in time.

RNNs have a stateful property influenced by past learning. Input is expected to take place in a sequence of passes over time and the RNN holds information from one pass to be used in a later pass, i.e. the RNN has a memory. Thus, in an RNN, neurons in a layer not only have weighted connections from neurons in a previous layer for the current pass, they also have inputs from neurons from previous passes. RNNs are widely used in speech recognition, machine translation, time series forecasting and image recognition. In general, RNNs are a good choice for advancing or completing information, such as autocompletion.

© ISO/IEC 2022 – All rights reserved

RNNs are well suited to process sequential input data of variable length or to output sequential data of variable length. Common types of RNNs include long short-term memory (LSTM) and gated recurrent unit (GRU) networks, a simpler variant of LSTM.

6.5.3.2.3.2 Long short-term memory networks

Long short-term memory (LSTM) networks are a form of RNN designed for problems that require remembering information with both longer and shorter time differences, making them suitable to learn long-term connections. They have been introduced to solve the vanishing gradient problem in RNNs associated with backpropagation. Each LSTM cell has an internal memory and a hidden state. RNNs depend on the use of backpropagation during training but this approach can suffer from either the vanishing gradient problem or the exploding gradient problem[6]. During training, the RNN weights are updated based on the partial derivative of the error function in each iteration of training. The derivative can either be very small, resulting in a RNN that is effectively untrainable, or it can be very large, resulting in a RNN that is highly unstable.

An LSTM is designed for learning long-term dependencies and has an architecture based on a combination of neurons, with a cell (which has the memory capability), an input gate, an output gate and a forget gate. The memory cell is associated with each neuron containing information about its previous state. The function of these gates is to safeguard the information by stopping or allowing it to flow. The input gate determines how much of the information from the previous layer gets stored in the cell. The output gate on the other end determines how much of the state about this cell is passed to the next layer. The forget gate allows the network to clear its internal state. For example, if the input is a paragraph and a new sentence begins, it can be necessary for the network to forget some characters from the previous sentence. Since each of the LSTM neurons has a separate weight for each of the gates, this is more complex than other kinds of NN and as a result LSTMs typically require more resources to train and to operate. LSTMs have been shown to be able to learn complex sequences, such as writing like Shakespeare or composing music.

6.5.3.2.4 Convolutional neural network

Convolutional neural networks (CNNs) are a type of NN that includes at least one layer of convolution to filter useful information from inputs. CNNs are primarily used for image processing[7] and video labelling but are also applicable to other types of input such as audio or text (sometimes using recurrent versions, or recurrent CNNs). The connectivity patterns of CNNs for image processing resemble the organization of the animal visual cortex. As opposed to RNNs, each neuron placed on a given layer of the CNN is connected to neurons from a previous layer only and is not fed with information on its previous state. The set of input neurons of a given neuron is called its receptive field. In a CNN, it is typical for the NN to contain different kinds of layers, including convolutional layers and pooling layers. Convolutional neural networks assume that their input have a grid-like topology, like images (2-D grid) or time series (1-D grid). The convolutional layers treat their input as a vector or a matrix (e.g. a 2-dimensional image) and apply a convolution to the input, which is a sliding dot product or cross-correlation, where a vector or matrix called a kernel is applied to the input data. A particular convolution is designed to extract particular features from the input and produces a feature map for the next layer in the CNN. This approach can make the CNN location-invariant with respect to the features in the input data, something highly desirable when dealing with real-world images.

Pooling layers reduce the dimensions of their input data either locally (over a small number of inputs) or globally over all inputs from the prior layer. Pooling effectively filters out inconsequential details. Multiple convolutional and pooling layers can be used in sequence to produce better results.

Fully connected layers are typically used near the output from the CNN. Real world implementations of CNNs often glue an FFNN to the end to further process the data, which allows for highly non-linear abstractions.

6.5.3.2.5 Structured perceptron

The structured perceptron is an extension of the perceptron algorithm used by NNs. Structured perceptrons use one neuron layer multiple times in succession to predict different parts of the output.

© ISO/IEC 2022 – All rights reserved

The structured perceptron training procedure reflects this design: a single target variable is used to assess multiple predictions at the same time.

6.5.3.2.6 Deep Boltzmann machine

Bolzmann machines (BM) are networks of binary units {0,1} comprised of a set of visible units and a set of hidden units. Connections are only between neighbouring units. They are bidirectional and are capable of learning unknown probabilistic distributions. They are useful in object or speech recognition. BMs are a generative algorithm.

The simplest form of BMs is called restricted Boltzmann machines (RBMs). When multiple layers of RBMs are stacked, the network is named deep Boltzmann machines (DBMs). Deep belief networks (DBNs) are similar to DBMs but also include neuron layers (i.e. unidirectional connections).

6.5.3.2.7 Capsule network

Capsule networks (CapsNet) are NNs that implement dynamic routing that can train on sparse data. They improve some limitations of CNNs by replacing neurons with structures called capsules, which consist of sets of tightly interleaved neurons that are updated all at the same time. Connections between capsules are also improved, so that CapsNets can better capture hierarchical relationships and create higher order representations. CapsNets are a discriminative type.

6.5.3.2.8 Generative adversarial network

Generative adversarial networks (GANs) are NNs containing one or more generators, which attempt to create samples that are the most representative of the dataset, and one or more discriminators, which try to distinguish generated samples from real ones. Generator and discriminator components are trained together to improve the inner representations of the network.

GANs can be used for various tasks (e.g. classification), but also for other purposes such as generating artificial data or adapting to a new domain.

6.5.3.3 Bayesian network

Bayesian networks are graphical models used for generating predictions on the dependencies between variables. They can be used to derive probabilities for the causes or variables that can contribute to the outcome. This causality is very useful in applications such as medical diagnosis. Bayesian networks are also useful for data analysis, addressing incomplete data, and mitigating overfitting of models to data. Bayesian networks rely on Bayesian probability: the probability of an event is considered to be the degree of belief in that event. Various Bayesian statistical methods exist which can be used in conjunction with Bayesian networks to determine causality or perform data analysis. Bayesian networks often utilise a graphical representation of variables referred to as directed acyclic graphs. These graphs have the property that by following the links between a variable *x* and other variables, the graph will never direct back to x. Further information on Bayesian networks can be found in Reference [8].

6.5.3.4 Naïve Bayesian algorithm

Naïve Bayesian algorithm is a classification technique based on Bayes' theorem. The theorem determines the probability of an event occurring, based on knowledge of prior related events. This helps to improve the accuracy of determining whether the event will occur. For example, a diagnosis of a medical condition can be made more accurate by considering the patient's lifestyle, e.g. sedentary versus active. With naïve Bayesian classification, it is assumed that features of data are statistically independent of each other. This classification method has the advantages that it is relatively simple and does not rely on a very large dataset for training.

© ISO/IEC 2022 – All rights reserved

6.5.3.5 Support vector machine

A support vector machine (SVM) is an ML method widely used for classification and regression. An SVM classifier marks samples into two different categories and then assigns new input data instances to one category or the other. SVM are maximum-margin classification algorithms. They define a hyperplane to separate the data into two classes, providing the maximal distance between the classifying plane and the closest data points. The points that are closest to the boundary are called "support vectors." The orthogonal distance between support vectors and the hyperplane is half of the "margin" of SVM. SVMs also use kernel functions to map data from the input space into a higher-dimensional (sometimes infinite) space, in which the classifying hyperplane will be chosen.

The training of an SVM involves maximizing the margin subject to the data from the different categories that are on the opposite side of the hyperplane. These "hard-margin" SVM are rarely used in practice. A hard-margin classifier only works if the data is linearly separable in the embedded space. Even a single sample can make it impossible to find a separating hyperplane. In contrast, soft-margin classifiers allow data samples to violate the margin (i.e. to be situated on the wrong side of the hyperplane). Soft-margin classifiers attempt to achieve maximal margin while limiting margin violations. Categorization of unlabelled data and use in prediction and pattern recognition are examples of the application of SVM.

When using an SVM for regression, the objective is the reverse of an SVM classifier. In SVM regression, the objective is to fit as many data instances as possible inside the margin, while limiting margin violations (i.e. those samples outside the margin).

6.5.3.6 Decision trees

Decision trees use a tree structure of decisions to encode possible outcomes. Decision tree algorithms are widely used for classification and regression. The tree is formed of decision nodes and leaf nodes. Each decision node has at least two branches, where leaf nodes represent the final decision or classification. Generally, the nodes are ordered in terms of the decision that gives the strongest predictor. Input data needs to be split into various factors in order to determine the best outcome. Decision trees are analogous to flow charts where at each decision node a question can be asked to determine which branch to proceed to.

6.5.4 ML optimisation methods

6.5.4.1 General

ML optimisation methods are used to fit an ML model to ML data. The challenge in ML optimisation is to find the optimal parameters of an ML model to minimise a given loss function for the given ML data. Most of the following methods rely on a loss (sometimes called "cost") function to determine if the model is converging to an acceptable solution or diverging. Prevalent ML optimisation methods are described in the following subclauses.

Beyond optimisation methods, other techniques can improve how the ML model fits the data. One such aspect is the choice of a better loss function. For example, regularization is a technique used to reduce overfitting and high prediction variance, by introducing new terms in the loss. It works by penalising a complex model and favouring a more general, less precise model.

6.5.4.2 Gradient descent methods

Gradient descent is an iterative technique to find the minimum of a function: while feeding the whole dataset as a single batch the parameters are updated step by step, in the same direction as the first-order derivative (gradient) of the function. The solution is a global optimum when the objective function is convex. When updated parameters are calculated via randomly selected smaller batches instead of on the whole dataset as a single batch, this is called stochastic gradient descent (SGD). SGD reduces the calculation costs while risking a solution that is trapped at a local minimum. The iterative step size of the gradient is referred to as the learning rate.

© ISO/IEC 2022 – All rights reserved

Momentum methods introduce speed as a variable to represent the direction and rate of a parameter's change in its space; this operation preserves the influence of the previous update direction for the next iteration. Nesterov accelerated gradient descent applies an adaptive momentum term resulting in faster convergence.

Adjusting the learning rate by varying the step size of each parameter update improves gradient descent methods. This is called an adaptive learning rate. An example is AdaGrad (adaptive gradient algorithm) which automatically tunes the learning rate.

Redundant information in training data can lead to slow convergence with SGD methods. Stochastic average gradient maintains a parameter to record the sum of the latest gradients. This sum is updated randomly per iteration and replaces the old gradient with a new gradient each cycle. This technique achieves variance reduction.

6.5.4.3 Newton's method

Newton's method uses the first order derivative (gradient) and second order derivative matrix, otherwise known as the Hessian matrix, to approximate the objective function with a quadratic function. Newton's method fits the local surface of the current position with a quadratic surface; this contrasts with gradient descent method which fits the current local surface with a plane.

6.5.4.4 Conjugate gradient

The conjugate gradient (CG) method calculates the Hessian vector product without directly calculating the Hessian matrix as in Newton's method. CG generates a new search direction only using the previous vector. It avoids the compute penalty of calculating the inverse of the Hessian matrix.

6.5.4.5 Gaussian processes

A Gaussian process is a collection of random variables with consistent, joint Gaussian distributions. Gaussian process is based on Bayesian theory and statistical learning. Its advantages are in flexible nonparametric inference and strong interoperability at the expense of complexity and large storage requirements.

6.5.4.6 Least square curve fitting

Least square fitting is a technique to fit a polynomial function to given data by minimising the squared sum of differences between the outputs of the polynomial function and the given data. The order of the polynomial needs to fit the measurement data to achieve good results. Least square curve fitting can operate in a batch manner on a complete dataset or in a recursive manner with growing datasets.

6.5.4.7 Maximum likelihood estimation

Maximum likelihood estimation is a method for estimating the parameters of a probability distribution by maximising the likelihood function. To build a probabilistic model from observed data, maximum likelihood estimation chooses the hypothesis that maximises the likelihood of the observed data given the hypothesis. In its general form it does not use any a priori knowledge to favour a particular hypothesis. Maximum likelihood estimation methods often use the logarithm of the likelihood for calculation reasons.

6.5.4.8 Expectation-maximisation

Expectation-maximisation is an iterative method to learn model parameters with hidden variables not observable in the data. It consists in an alternation between expectation steps (estimate the hidden variables based on current parameters) and maximisation steps (re-estimate the parameters to optimise the likelihood of the data, given the current value of the hidden variables), up to convergence.

© ISO/IEC 2022 – All rights reserved

6.5.5 ML evaluation metrics

6.5.5.1 General

The ML tasks (discussed in 6.2) typically determine suitable evaluation metrics. Understanding a model's use case is vital to selecting appropriate evaluation metrics. Multiple metrics can be required to adequately express and examine model performance. Using a single metric, even an aggregate one such as the F1 score, is risky without thorough inspection of the underlying results. Further, class imbalance in the training data is a confounding factor that can distort various metrics.

Many of those metrics, in particular in the case of classification, are based on the concepts of true positives (T_P, corresponding to samples which have been correctly detected), false positives (F_P, corresponding to samples which have been detected when they should not have), true negatives (T_N, corresponding to samples which have correctly remained undetected) and false negatives (F_N, corresponding to samples which have been undetected when they should have been detected).

There are many metrics available to represent the performance of trained models, such as:

— Accuracy, receiver operating characteristic (ROC), confusion matrix, precision, recall and F1 score can be used to evaluate classification algorithms;

— Mean absolute error (MAE), root mean squared error, relative absolute error, relative squared error, mean zero one error and coefficient of determination are common metrics for regression models;

— Evaluation metrics for clustering models include average distance to cluster centre, average distance to other centre, number of points and maximal distance to cluster centre.

The following subclauses describe examples of such metrics and are not meant to be exhaustive.

6.5.5.2 Precision, recall, sensitivity and specificity

A series of metrics can be computed directly as proportions between T_P, F_P, T_N and F_N:

— True positive rate (TPR) is the proportion of true positives (T_P) among all actual positives ($T_P + F_N$). Its complement is called false negative rate (FNR);

— True negative rate (TNR) is the proportion of true negatives (T_N) among all actual negatives ($T_N + F_P$). Its complement is called false positive rate (FPR);

— Positive predictive value (PPV) is the proportion of true positives (T_P) among all predicted positives ($T_P + F_P$);

— Negative predictive value (NPV) is the proportion of true negatives (T_N) among all predicted negatives ($T_N + F_N$).

As they offer complementary points of view, for evaluation these metrics are typically used by pair: either PPV and TPR, in which case they are called precision and recall; or TPR and TNR, in which case they are called sensitivity and specificity.

There are multiple ways to apply those definitions, and the meaning of the terms differs depending on whether the classification is binary or multi-class. In binary classification, a single class is chosen as positive, and the definitions are applied to that class only. Thus, for instance the term precision refers to the precision of the positive class, and precision of the other class is not computed. In multi-class classification, the definitions are applied in turn to all classes, and precision refers to some combination of precision for each class.

6.5.5.3 F1 score

F1 score expresses model performance through a combination of recall and precision. The F1 score is determined by the harmonic mean of precision and recall. F1 score reaches its best value at 1; this would represent perfect precision and recall.

© ISO/IEC 2022 – All rights reserved

6.5.5.4 Accuracy

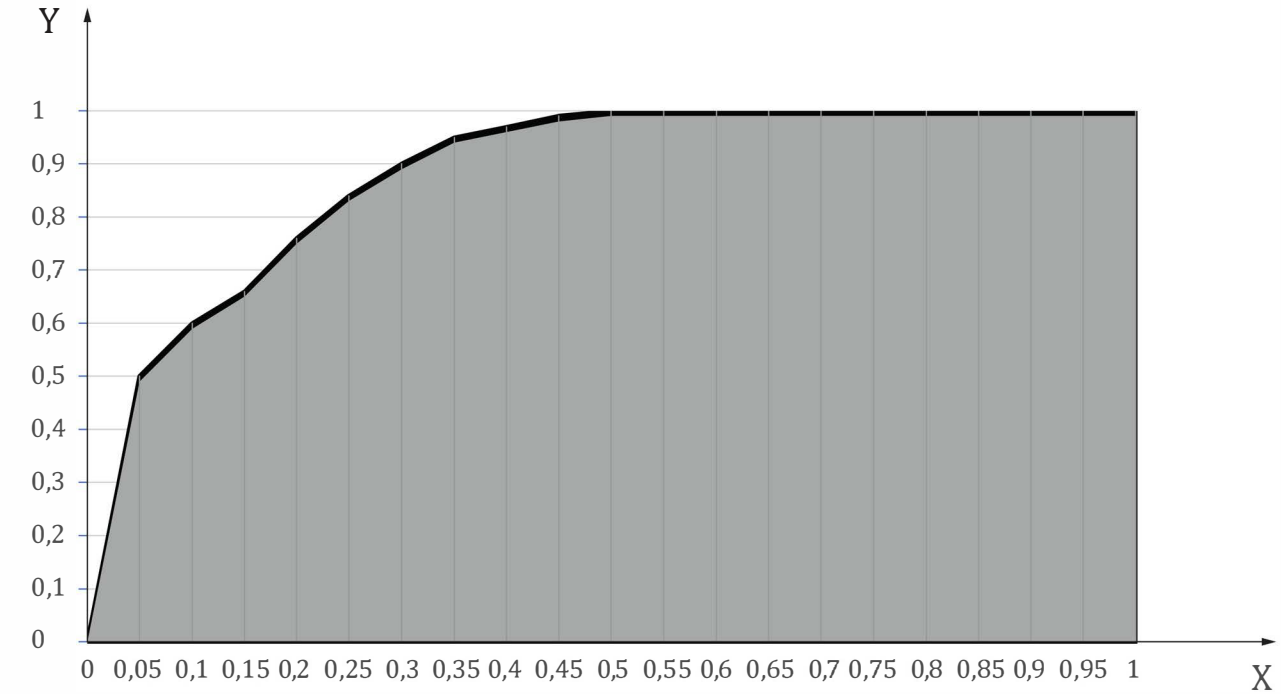
Accuracy expresses a model's correct classifications out of the total incorrect and correct classifications.

It is mostly relevant in binary cases. In multi-class classification, it can be understood either as per-class accuracy or as model accuracy. Per-class accuracy equates to the recall of that class and the term recall is preferred. Model accuracy ignores the particular class to which the sample belongs and only considers whether its classification was correct or not, by cumulating results from all samples indiscriminately. However, this metric is typically less informative than the precision and recall pair or the F1 score.

In multi-label classification, the term accuracy often refers to subset accuracy, i.e. the accuracy of predicting the correct label set (as a whole).

6.5.5.5 Receiver operating characteristics and area under the curve

The receiver operating characteristics (ROCs) are modelled as a curve representing the ability of a model to distinguish between classes. It is plotted with the TPR against the FPR. Any point on the curve can be selected as the operating point for a model thus providing information on TPR and FPR. As such, depending on the application and importance afforded to TPR and FPR, the operating point can be selected to maximise the desired precision and recall goals. Figure 4 shows an example of ROC curve.



Key

X	FPR
Y	TPR
	AUC
	ROC

Figure 4 — Receiver operating characteristics curve

The area under the ROC curve (AUC) is a measure of performance across all classification thresholds. Its value is always in the [0, 1] range and can be used to rank models. A higher AUC value translates to a better prediction value than a lower one. The best possible AUC is 1,0 while the worst expected AUC is 0,5 (because the diagonal line corresponds to random predictions). Any value less than 0,5 means

© ISO/IEC 2022 – All rights reserved

a user can simply do the exact opposite of what the model recommends to get the value back above 0,5[9]. It is scale and classification-threshold invariant which can be good or bad depending on the applications. The AUC generally is used when there is class imbalance in the training data. Figure 4 shows an example of AUC as the grey area under the ROC curve.

6.5.5.6 Confusion matrix

The confusion matrix is a table enabling visualisation of the performance of a classification model. Usually, the rows represent counts of elements predicted to be in a particular class while the columns represent counts of elements in their true class. The more the counts are concentrated on the diagonal, the less confusion there exists between classes and the better the performance of the model. Figure 5 shows a two-class case. In this particular binary setup, the top left cell is the number of true positives (T_P), the top right cell is the number of false positives (F_P), the bottom left cell is the number of false negatives (F_N) and the bottom right cell is the number of true negatives (T_N). The confusion matrix enables computation of TPR, FPR, Accuracy, F1 score and other metrics.

		Class truth	
		Class 0	Class 1
Class prediction	Class 0	T_P	F_P
	Class 1	F_N	T_N

Figure 5 — Confusion matrix

Apart from classification, a confusion matrix can similarly be drawn for a number of other tasks. But depending on the task, the table can remain partly filled: for instance, in object detection the T_N value is ill-defined.

6.5.5.7 Kappa coefficient

The kappa coefficient (also known as Cohen's kappa) measures inter-rater reliability for categorical elements. In ML, the kappa coefficient measures the agreement between the model classification and the ground truth data labels. It can also be used to test the agreement of multiple models and to discard models with low agreement. The kappa coefficient is useful in the case of imbalanced datasets because it expresses model performance relative to random guessing using the target distribution of the application.

This coefficient (κ) {0,1} measures the degree of interobserver agreement. Measurement of the extent to which data collectors (raters) assign the same value or score to the same variable or observable is termed inter-rater reliability. This measure captures the chance occurrence that subjective raters guess on the value of variables due to uncertainty.

While sometimes used as a metric for model performance, it is not designed for such purposes and this use is not recommended in the scientific literature[10][11].

© ISO/IEC 2022 – All rights reserved

6.5.5.8 Matthew's correlation coefficient

Matthew's correlation coefficient (MCC) expresses binary classification performance quality. It is statistically known as the phi (φ) coefficient and is related to the chi-square (χ^2) statistic. It is calculated directly from the confusion matrix (see 6.5.5.6):

$$\varphi = \frac{T_P \times T_N - F_P \times F_N}{\sqrt{(T_P + F_P)(T_P + F_N)(T_N + F_P)(T_N + F_N)}}$$

7 Machine learning approaches

7.1 General

ML methods can be classified into three approaches: supervised machine learning, unsupervised machine learning and reinforcement machine learning. Semi-supervised machine learning, self-supervised machine learning, transfer learning and ensemble learning are inspired by multiple ML approaches at the same time and warrant separate discussion.

The ML approaches described in the following clauses can utilise numerous ML methods as shown in Figure 6. Some of the methods used are described in Clause 6. The lists illustrated in Figure 6 are exemplary and not all methods are described in this document. Figure 6 shows that some methods are utilised in more than one ML approach. Neural networks for example are used in all types of ML approaches.

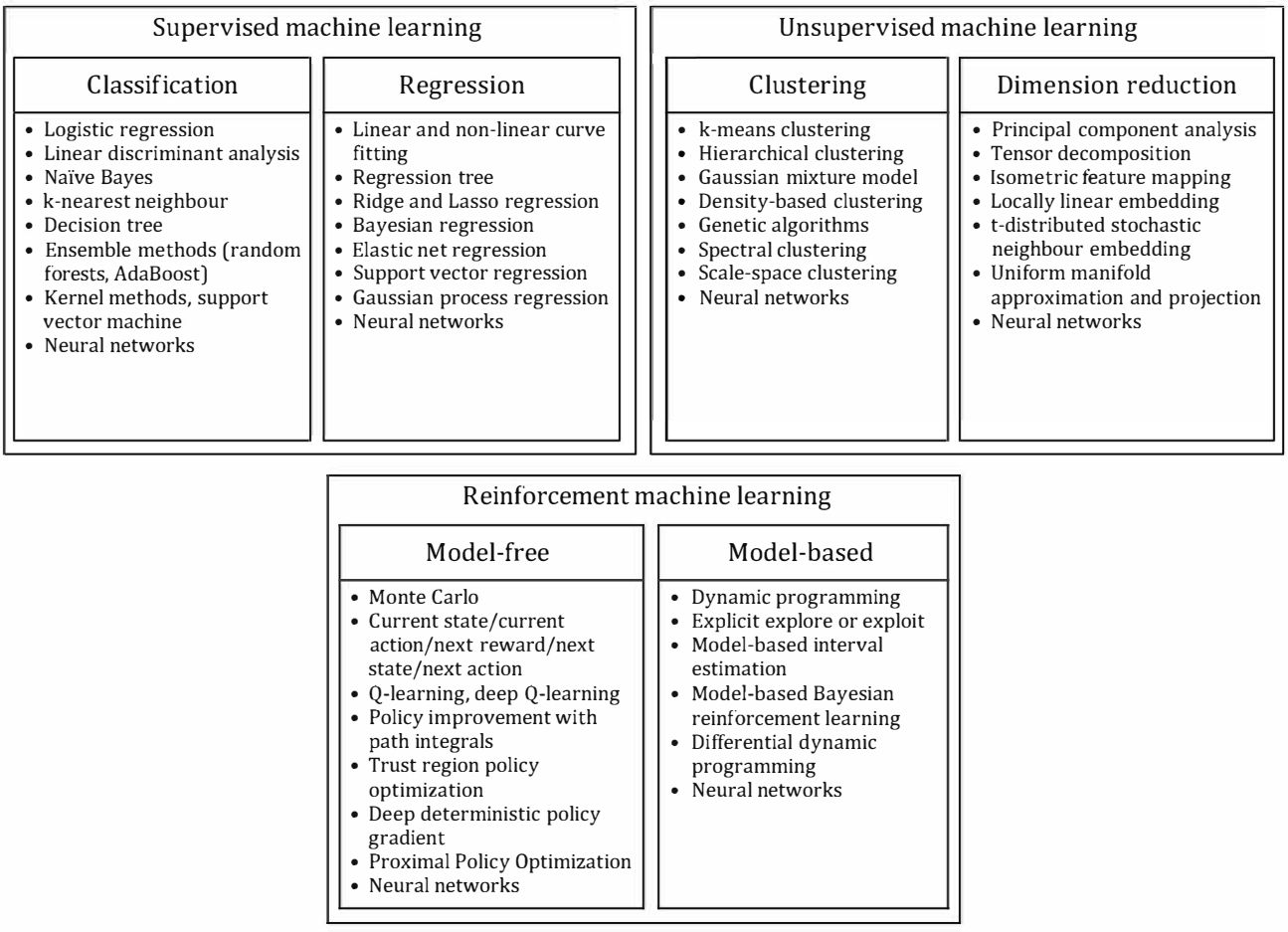


Figure 6 — ML methods categorized as supervised machine learning, unsupervised machine learning and reinforcement machine learning

© ISO/IEC 2022 – All rights reserved

7.2 Supervised machine learning

In supervised machine learning, ML models are trained using labelled data. Labelled data consists of samples with inputs mapped to correct, or true, outputs. Thus, the training data are organized as pairs of input variables and "true" outputs. In different contexts, true outputs are also referred to as labels, target variables and ground truth. During the supervised learning process as illustrated in Figure 7, the algorithm is fitted to the inputs and outputs to produce a model.

© ISO/IEC 2022 – All rights reserved

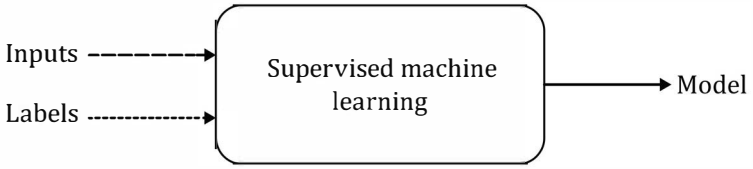


Figure 7 — ML model creation using supervised machine learning

Labels can be part of the original data, but it is often necessary to generate labels manually or through other AI processes. Depending on the targeted ML task, labels can take various forms:

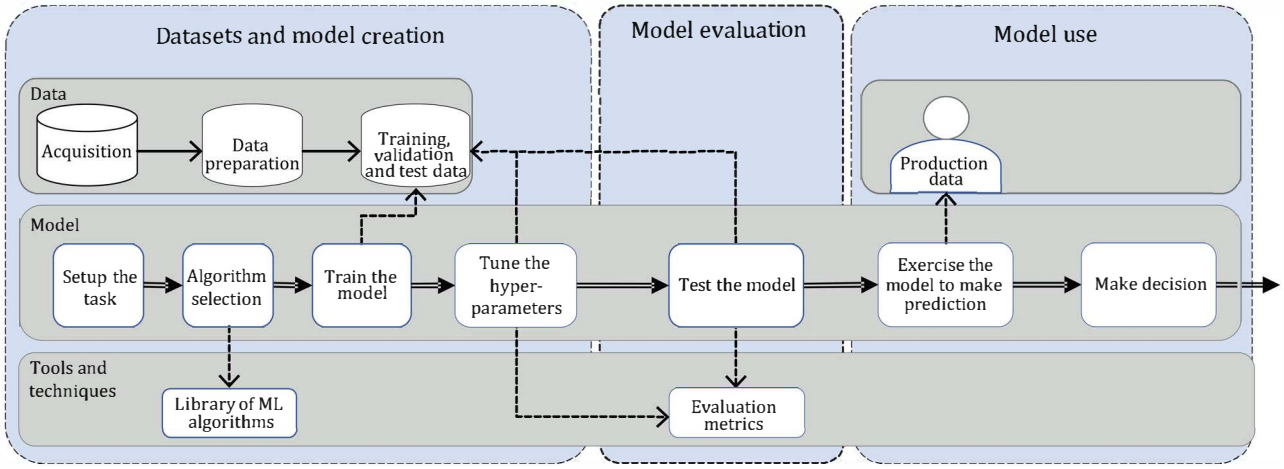
— classification requires categorical labels (the category to which the data instance belongs, such as a dog or a building);

— for regression they are numerical (continuous values such as degrees, likelihoods or probabilities);

— in case of structured prediction, they can also take the form of a structured object (such as a sequence, an image, a tree or a graph).

A typical supervised machine learning process is exemplified in Figure 8, showing the various processes in the creation, evaluation and use of an ML model. The "datasets and model creation" stage corresponds to the preparation, training and selection of the model, as well as any data needed for model creation or evaluation. "Model evaluation" is when the model is tested using evaluation metrics, in order to assess its performance and conformity. In the "model use" stage, the model is applied on production data to make predictions. While the horizontal dimension corresponds to the three stages, the vertical dimension indicates whether the depicted components and processes are associated to data, model or tools.

© ISO/IEC 2022 – All rights reserved



Key

data flow

use of resources

ML process flow

Figure 8 — Typical supervised machine learning process

The performance and robustness of a trained model relies heavily on training data diversity (e.g. various kinds of pedestrians), training data quality (e.g. lighting effects or resolution in photos) and label accuracy (e.g. correctly labelling a pedestrian within a crosswalk). All aspects of supervised machine learning data are prone to error, and special care should be taken across the full cycle from dataset creation to model testing.

7.3 Unsupervised machine learning

In contrast to supervised machine learning, unsupervised machine learning maps inputs to outputs directly without being trained on labelled data. The process of training is, however, similar to the supervised machine learning process shown in Figure 8. During the unsupervised learning process as illustrated in Figure 9, the algorithm is fitted on the inputs only to produce a model, without prior access to labels. Labels are usually produced as a by-product of model training.

In clustering tasks using, for example, the K-means algorithm, samples are continuously iterated over through a clustering algorithm until the minimum distance from each sample to a centroid is achieved. In dimensionality reduction tasks using, for example, the principal component analysis (PCA) algorithm, the variance of each feature in the input data is calculated and a predefined number of features with the highest variances are returned. Models developed in this fashion can be used to identify similarities, patterns or anomalies, and can be used to reduce dimensionality (where the most statistically relevant features are determined irrespective of any label). Unsupervised ML often results in knowledge discovery.

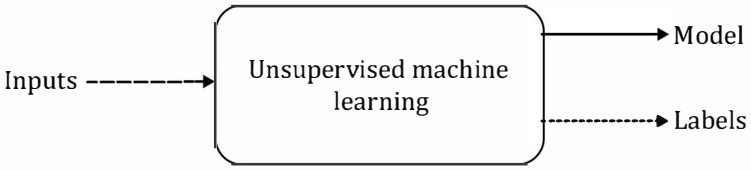


Figure 9 — ML model creation using unsupervised machine learning

© ISO/IEC 2022 – All rights reserved

Use cases for unsupervised machine learning include:

— discovery of clusters in the input dataset;

— discovery of latent factors. For high dimensional data, it is often desirable to reduce the dimensionality by projecting the data to a lower dimensional subspace which captures the "essence" of the data;

— identification of correlations within a set of measurements of several variables;

— image inpainting of damaged or otherwise impaired pictures;

— market basket analysis, where groups of goods which are usually purchased or sold together are identified.

7.4 Semi-supervised machine learning

Semi-supervised machine learning is defined as "machine learning that makes use of both labelled and unlabelled data during training". Semi-supervised machine learning is a hybrid of supervised and unsupervised machine learning.

One approach to semi-supervised learning is the creation and use of pseudo-labels to improve the overall performance of a model. In this case, the model is first trained using labelled data. Then the trained model is used to predict pseudo-labels for the unlabelled data samples. Finally, a training dataset is composed using both labelled and pseudo-labelled samples and used to retrain the model. This approach is called self-training.

Semi-supervised learning is useful in cases where labelling all the samples in a large training dataset would be prohibitive from a time or cost perspective.

7.5 Self-supervised machine learning

Self-supervised machine learning is an approach for training on unlabelled data using algorithms that normally belong to supervised machine learning. This is achieved by using implicit labels, such as the input itself (e.g. an image to reconstruct alike), part of the input (e.g. one word in an input sentence), or any other label that can be easily generated from the raw data (e.g. the unshuffled version of a sequence that has been artificially unshuffled). It is typically applied on large amounts of data.

Self-supervised machine learning is most often used for learning representations: the final outputs of the model are discarded, and intermediate outputs can be reused as inputs to another ML model. This is particularly useful for processing unstructured data while reducing feature engineering efforts.

Note that self-supervised machine learning is unrelated to self-training, which is a specific method for semi-supervised machine learning.

7.6 Reinforcement machine learning

Reinforcement learning differs from the other approaches, as its principle is that the model is initialised at a state, an action is taken, a reward for the action is determined and the model is advanced to a new state that attempts to maximise the reward. Training can be used to initialise the model or to determine policies the model uses to take actions.

Reinforcement machine learning is the process of training one or more agents to interact with their environment to achieve a predefined goal. In reinforcement machine learning, ML agents learn through an iterative process of trial and error. The agent's goal is to find the strategy (i.e. build a model) for obtaining the best rewards from the environment. For each trial (successful or unsuccessful), indirect feedback is provided by the environment. The agent then adjusts its behaviour (i.e. its model) based on this feedback. This process is illustrated in Figure 10. The agent determines which interactions consistently provide the maximum reward for its actions in an attempt to meet the objective.

© ISO/IEC 2022 – All rights reserved

The agent's action and interaction with the environment are usually modelled as a Markov decision process (MDP). Figure 10 is a typical representation of an MDP. Reinforcement machine learning algorithms do not assume knowledge of an exact mathematical model of the MDP (but some techniques try to approximate it). Reinforcement machine learning algorithms typically target large MDPs where exact methods become infeasible. Unlike supervised machine learning, pairs of inputs mapped to labelled, true outputs are not required. Instead, the objective is to use trial and error and converge outcomes on a specified goal through a delayed reward. Each time the model makes a prediction, a reward is calculated, and further trials are run to optimise the reward. The reward is generally a calculated number that represents how close the system is to achieving the objective for a given trial. In some cases, additional information about the environment is modelled or additional information is provided to the agent to improve training performance. The objective, or definition of success, is typically defined by the system designer.

Reinforcement machine learning can also be combined with supervised machine learning: training on labelled data is used to initialise the model and reinforcement is used to determine subsequent policies the agent uses to take actions.

In some cases (e.g. structured prediction), the application of reinforcement machine learning itself can depend on the pre-existence of a (labelled) dataset, because it is needed to generate the environment: in such cases, the environment acts as a proxy to the information included in the dataset.

Reinforcement machine learning facilitates the application of continuous learning, as the training environment can be either a simulated one, or the actual environment encountered during the operation phase, as long as there is enough information to compute the reward in that phase.

Reinforcement machine learning is often used for control purposes. Control is the application of reinforcement machine learning to interact with an environment. Finding a policy which maximises reward when actions following this policy are performed allows for action planning and optimal control.

Use cases of reinforcement machine learning include playing video and board games, where the goal is to maximise the utility of game moves and thus gain control over the outcome of the game.

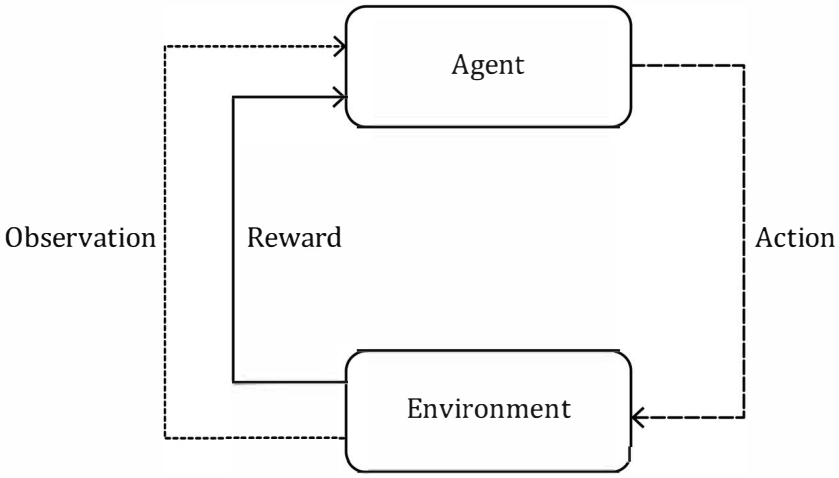


Figure 10 — Typical reinforcement machine learning process

7.7 Transfer learning

Transfer learning refers to a series of methods focused on storing and abstracting knowledge gained from data intended for solving one problem to apply it to a different problem, which can be loosely related (e.g. solving the same task in a substantially different domain). For example, knowledge gained from recognizing house numbers in a street view can be used to recognize handwritten numbers.

© ISO/IEC 2022 – All rights reserved

A well-known example of this learning paradigm is the fine-tuning technique, where a model already trained to solve the first problem is repurposed and further trained to solve the new problem. For example, a model that has learned to recognize furniture and objects can be fine-tuned to identify scenery (e.g. living room, beach, kitchen and so forth) by training it on pictures of the new domain.

Various other methods have been designed to achieve transfer, including instance synthesis, feature-based algorithms or regularisation techniques[13].

In practice, transfer learning relies on other learning approaches and complements them with additional steps for storing and abstracting knowledge for further reuse. As illustrated in Figure 11, transfer methods can be applied before, during or after learning occurs.

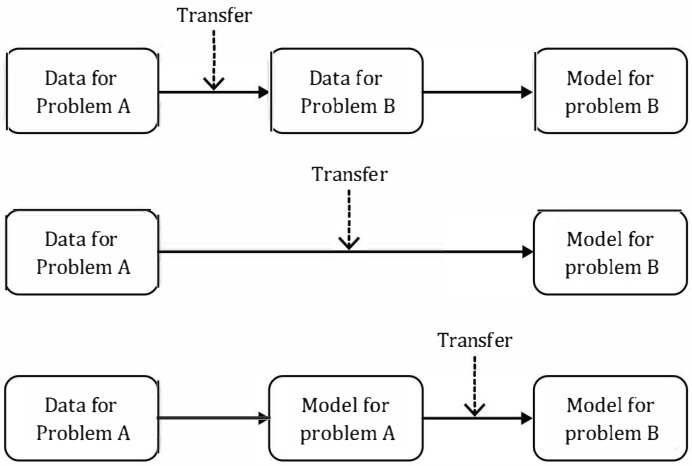


Figure 11 — Typical transfer learning processes

8 Machine learning pipeline

8.1 General

To reach a particular application goal using ML, an ML model is created, evaluated and put into use. This process typically involves data, algorithms and computational resources. This clause describes a representative ML pipeline, including the processes that apply at each step. Before entering that pipeline, it is necessary to define the task, or problem to be addressed. This establishes the goals and requirements of the solution. A thorough definition of the problem (including e.g. a precise definition of the input and output formats) aids in selecting the appropriate ML algorithms and for acquiring the relevant datasets needed to train the ML model.

Figure 12 illustrates specific ML processes involved in developing, verifying, deploying and operating an ML model, and how they relate to the AI system life cycle stages. The following aspects apply across the entire ML pipeline:

— risk management and governance;

— security and privacy;

— accountability, transparency and explainability;

— safety, resilience, robustness and fairness.

© ISO/IEC 2022 – All rights reserved

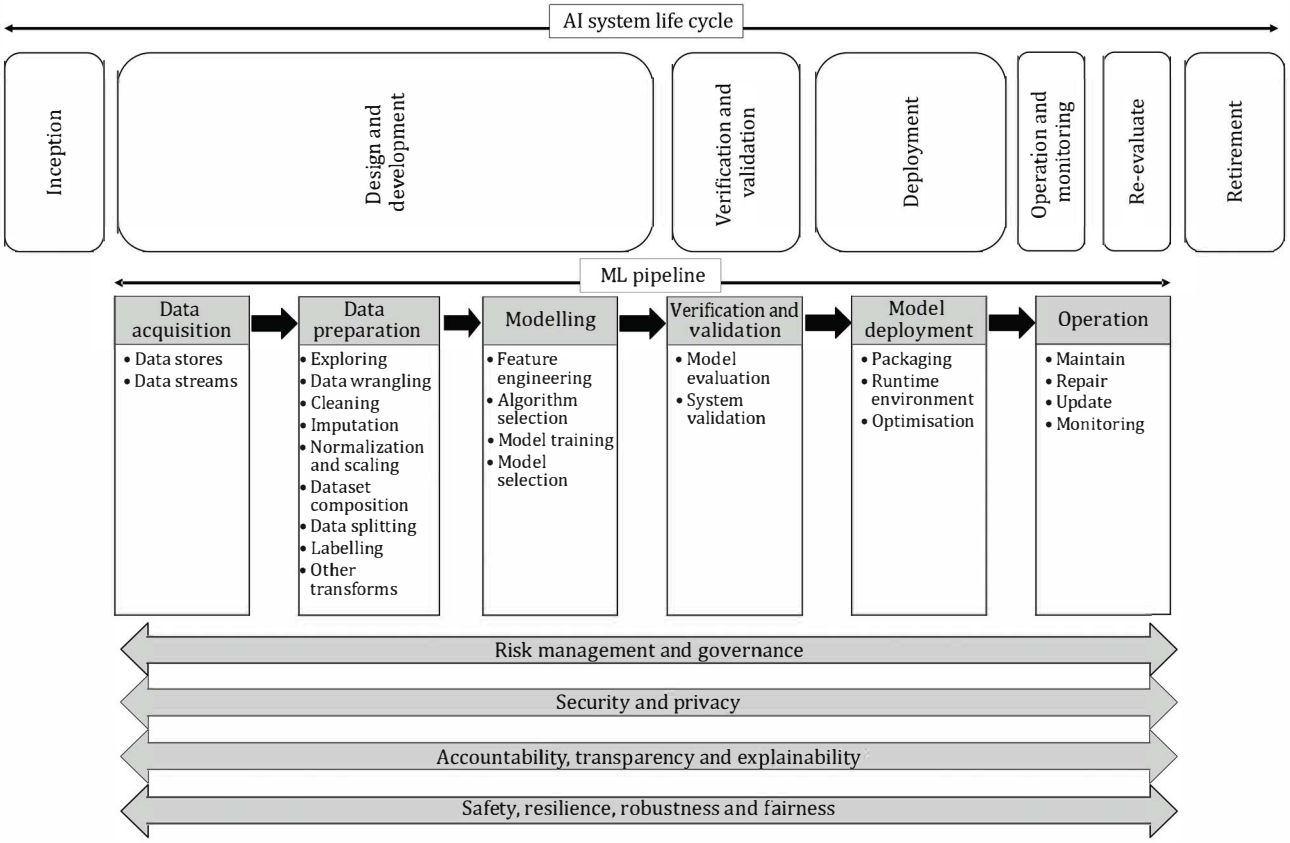


Figure 12 — Machine learning pipeline and mapping to the AI system life cycle

The following subclauses describe the stages of the pipeline shown in Figure 12 and the processes that are likely to occur as part of each stage. The design and development of an ML model consists of several sequential and non-sequential steps. It starts with some preparatory work encompassing the three steps of acquiring and preparing a dataset, of performing feature engineering, and of selecting a given algorithm. There is no general dependency between those processes and they can in principle be conducted in any order, or simultaneously. But depending on the exact use case there can be additional technical constraints that enforce a particular order, or a need to iterate between those processes. For instance, analysing the dataset can help to design better features; and the granularity of the designed features can drive the choice of an algorithm that relies more or less on those features; but the choice of algorithm can also constrain the type or granularity of the labels to annotate on the collected data, in which case it needs to be chosen first.

Actual model creation occurs after all preparatory steps have been performed. This encompasses training one or more candidate models, and then selecting the best candidate among those.

8.2 Data acquisition

The initial stage for development, deployment and operation of an ML model is the acquisition of data. Data is a key component in the ML model life cycle, as both training and evaluation rely on it. Its acquisition and preparation need special care such that the acquired data is appropriate to the business purpose of the model. Training data is often a subset of a larger population that is ideally representative of that population. Data used to train supervised machine learning models will include one or more labels or target variables, that represent a truth about the data record or sample. For example, each email in a set of training emails can be labelled as “spam” or “not spam”.

ML models will need validation and test data to determine their accuracy and other performance metrics. Validation and test data can be acquired together with the training data or they can be similar datasets acquired by different means.

© ISO/IEC 2022 – All rights reserved

The type of data needed depends on the problem being solved. Forecasting sales, for example, can require access to transactional data. Classification of pictures can require access to image files. Likewise, the problem to be addressed states what information (i.e. which features) are needed in the dataset. The data sources are identified, and the required data is acquired (e.g. purchased or collected). Data can come from many sources and can originate from a fixed data store or from a stream. Common sources of training data include transactions, sensors, queries, images, documents, sounds and social media.

8.3 Data preparation

Data is rarely ready for immediate use in training ML models. ML models often have requirements on the data used to train it. Data preparation usually consumes a significant amount of the time and effort to develop an ML model. Data preparation can involve the use of a number of automated tools for formatting and cleansing the various datasets. Data preparation includes formatting data into a form usable by the specific system and data cleansing. Data cleansing can include removal of duplicate data and spurious data (data not useful for solving the problem) and fixing missing data. Noisy data which can contain erroneous records or outliers can be dealt with at this stage. This also includes de-identification: although the acquired data can have been de-identified earlier, additional de-identification can be required because of data processing (e.g. joining datasets) in this stage. Examples of these processes are discussed in the following paragraphs for exploring, wrangling, cleaning, imputation, normalization and scaling, dataset composition, data splitting and labelling. They can be applied in another order.

Exploring: Exploring involves the examination of the raw data. The aim is to gain insights that help in determining which processes are needed to prepare the data for model training. Insights, for example, include the shape of the dataset, data types, mean, variance and range of numerical data, existence of an index, features and labels.

Explorations on data distribution over features also benefits the selection of data preparation actions. For instance, distribution analysis can help identify outliers, relations and patterns of data. These characteristics can be used as evidence to choose proper actions for improving data preparation, while a comparison between the distributions of training data and production or input data can help AI system stakeholders foresee the effectiveness of a planned solution.

Data wrangling: Data wrangling is carried out to create a well-structured, orderly dataset. Data wrangling processes for tabular data include combining columns, splitting columns, creating columns, changing data types and formats, removing, replacing or adding characters in strings and creating or renaming column headers.

Cleaning: Data cleaning is akin to data wrangling processes but at a more granular level. Common data cleaning processes include removing duplicate entries, filling in missing data and dealing with incorrect data and outliers. In some cases, it can be possible to interpolate to fill missing or invalid data. Examples of invalid data include a zip code in an attribute that requires a latitude and longitude pair or a number in a name attribute. When it is not possible to fill in missing or invalid data without corrupting the meaning of the data, it can be necessary to delete entire entries or attributes from the dataset. Training of some ML models will fail on missing or invalid data. Outliers can create noise in the data that causes the trained model to fail to generalise well on production data when it is deployed. It can be necessary to remove entries from the dataset that include outliers.

Imputation: Data imputation refers to a cleaning process using substituted values to replace missing data. Imputation methods include single imputation and multiple imputation. Single imputation (such as hot-deck, cold-deck, regression and mean value) directly generates the targeted value for use, under the assumption that there is no substantial uncertainty about missing values. Multiple imputation reflects uncertainty about missing values from an imputation model, generating the targeted value by considering several imputed values randomly selected from slightly different models.

Normalisation and scaling: Significant differences in the range of numerical features in a dataset can cause the features to be not comparable when training an ML model. Normalisation can be used to scale individual data samples to a unit norm (between 0 and 1). Scaling on the other hand can be used to

© ISO/IEC 2022 – All rights reserved

create a normal distribution from the individual samples or to compress the samples into a specified range.

Scaling or normalisation can be applied to the data to ensure the results of the model are appropriate. Examples for data normalisation techniques are:

— Min-max normalisation: linear transformation of the data to fit within minimum and maximum values (often zero and one);

— Z-score normalisation: data is scaled based on the mean and standard deviation;

— Decimal scaling: moving the decimal point of the attribute values.

Dataset composition: Dataset composition refers to the selection or compilation of acquired data from various data sources into a single dataset. This compilation is usually done taking into account the features to be trained and a sufficient distribution and representation of the objects for training the intended features.

Dataset splitting: ML models need training data, but also validation data for model selection, and test data for model evaluation. Training, validation and testing data need to be disjoint. In some cases, they are all acquired by separate means. But it also happens that they are acquired together from the same source(s), in which case it is necessary to split the available data into separate sets. The validation and test parts are typically chosen to be much smaller than the training one.

Labelling: Data labelling is the process of assigning a target variable to a sample. Linking data to its corresponding labels establishes ground truth. Data labelling can be based on extraction of meaningful information from a sample, such as the number of people in an image. A sample can also be augmented with information about its content to create a label, such as whether two people pictured in an image are arguing. Data can be labelled manually by humans or in some use cases it can be labelled automatically by computer.

Data labelling of samples can be required for training, as in the case of supervised learning. It can also be required for evaluation, even without supervised learning. In such cases, the corresponding data will include one or more labels or target variables, that represent a ground truth about the data record or sample.

De-identification is the process of removing the association between a set of identifying attributes and the data principal as described in ISO/IEC 20889. It can be needed to remove personally identifiable information (PII) if PII is included in the dataset. In particular, to use healthcare data to train an ML model, it can be necessary to de-identify personal or protected health information (PHI). Furthermore, checking for data poisoning is crucial to ensure training data has not been contaminated with data that can cause harmful or undesirable performance. If training datasets are incomplete, insufficient or misrepresent the distribution needed for training, the resulting predictions can be biased or discriminatory.

8.4 Modelling

Modelling is the process of developing a trained and tested ML model that is ready for deployment.

Feature Engineering: Features are the quantitative descriptors of the elements in a dataset, often seen as column headings in tabular data. Features can be numeric, textual, continuous or categorical. Categorical features can be nominal or ordinal. Date, time, name, address, age and sales amount are examples of features. Feature engineering is the process of selecting, characterizing and optimising features for use in an ML model. Examples of feature engineering include:

— **Encoding:** For efficiency and ease-of-use, textual and categorical features are often converted to numeric identifiers;

— **Data type conversion:** It can be necessary to convert the feature's data type to meet the requirements of a given model;

© ISO/IEC 2022 – All rights reserved

— **Dimensionality reduction:** High dimensionality in the input data can be troublesome for some ML algorithms. Therefore, it can be necessary to reduce the number of attributes per sample (often by using another ML model in turn). See also 6.2.6;

— **Feature selection:** The foundation of feature selection is that data can contain features that are either redundant or irrelevant and therefore, can be removed. Certain features introduce noise into the model or skew the model's predictions inappropriately, hence the need to remove them. Selecting the right features can be very important to the performance of a given ML model. This involves selecting attributes of variables from the available datasets for reasons including, but not limited to:

  — simplification of models to make them easier to interpret;

  — shorter training times;

  — ensure suitable dimensionality. Datasets that have a large number of features can encounter "the curse of dimensionality" where certain features introduce noise into the model or skew the model's predictions inappropriately;

  — reduce overfitting of the model.

  There are several feature selection algorithms that can be utilised to optimally select useful features efficiently. A simple approach is to eliminate the features that are intuitively irrelevant to the problem being solved. A second method is to visualise the data to see which features cause little effect on the performance of the model. In some cases, it can be necessary to train and test the model with and without particular features to see the difference in performance. Some ML algorithms such as decision trees incorporate feature selection as part of the learning process.

Algorithm selection: At this stage in the pipeline, it is necessary to select an ML algorithm appropriate to the task (e.g. classification, regression, clustering, recommendation). ML algorithms are the basis for creating an ML model. There are several algorithms available for each type of ML task with new techniques being introduced. Libraries or collections of ML algorithms are readily available. In some cases, it can be necessary, or desirable, to create a new algorithm or modify an existing algorithm. One or more algorithms can be selected from a library, or created, to address the defined problem or task. When packaged within a library, the appropriate application of the algorithm is sometimes described in the library documentation. In the general case, scientific and comparative literature is often widely available.

It is also possible to use an ensemble of models where the final prediction is based on a function applied to the predictions output from each model.

Model training: An ML model is trained by iterating over the training data to establish constants or weights, for each parameter in the model. Training is also referred to as "fitting" the model. The objective of training is to produce a model that generalises well on production data. Overfitting occurs when the model memorises the training data but doesn't generalise well; it can happen when there are insufficient training samples. Underfitting can occur when features are poorly selected or there are insufficient model parameters to properly learn on a large number of training samples. The training process is primarily applied during the design and development stage, but it can also occur again later, typically for retraining or for continuous training during model use. In practice, the model can be continuously trained during its use; however, this process causes significant challenges in ensuring consistently good performance on both old and new data.

Model selection: Also known as hyperparameter tuning, model selection is the process of using the validation dataset to assess and optimise hyperparameters to determine which values provide the best results in order to select the best model. Techniques such as exhaustive grid search and randomised parameter optimisation can be used to automate hyperparameter tuning. Additionally, some methods have the built-in capability to train and score the model using a grid of values for specified parameters.

Cross-validation is used if data is scarce and separate datasets for training and validation would be too small. k-fold cross-validation is often used for hyperparameter tuning. In k-fold cross-validation the original training set is partitioned into k subsets (folds). For each fold, the model is trained on the union

© ISO/IEC 2022 – All rights reserved

of the other folds and then the error of its output is estimated using the fold. The special case when each fold consists of one sample only is called "leave-one-out". Cross-validation can help prevent overfitting from data leakage.

Techniques such as Mallow's Cp, Akaike information criterion and Bayesian information criterion can also be used to compare the performance of different models without explicitly computing that performance.

8.5 Verification and validation

Model evaluation: Model evaluation is the process of comparing the predictions made by the model on test data to the actual labels in the data. There are many metrics available to represent the performance of trained models. These are described in 6.5.5.

It is common to test multiple models to see which ones best meet the initial design objectives coupled with mitigating the risks identified throughout development and use. However, such practice is to apply with special caution, so that it remains distinct from model selection, which should not be performed on test data. Comparative evaluation should only be conducted for a few models and which are very different (e.g. based on different training data, different ML algorithms or different ML approaches, not just different hyperparameters). Otherwise, there is an important risk of overfitting the hyperparameters to test data, which means effectively over-estimating model performance.

System validation: System validation is the check, if the trained and deployed model meets all the needs of the stakeholders and whether the initial assumptions about the environment on which the features were derived were correct. It is unrelated to validation data, which is used for tuning the hyperparameters during the design and development phase.

8.6 Model deployment

Once an ML model is trained and evaluated, it needs to be deployed into an operating environment where it can be used to make predictions on production data. ML models can be deployed as Web applications (where they are accessed through a REST API), or as native applications.

Packaging: Trained ML models have traditionally been deployed to servers, clusters or virtual machines. Several open formats are available to represent or exchange ML models such as open neural network exchange (ONNX)[15] and neural network exchange format (NNEF)[16]. More recently, models are being deployed in ready-to-run containers packaged with all dependencies, substantially simplifying deployment. With the rapid increase in processing power and storage capacity of devices, trained ML models are being deployed on devices such as unmanned aerial vehicles and drones.

Run-time environment: High-level languages such as R or Python are often used in the development of ML models. The trained model can then be deployed on any run-time environment that has a corresponding abstraction layer, such as an interpreter or just-in-time compiler. The run-time environment of the model can be of a lower-level than its development environment. Special care is needed for the run-time environment and how it can differ from the development environment.

Optimisation: The model can be optimised for a target platform, such as by adjusting data types from exact to approximate or from machine independent to those natively supported on the target platform (e.g. quantising weights of a neural network to integer values). Optimisation can also take place through rearranging or aggregating operations for which efficient hardware implementations are available. Depending on the type and extent of the optimisation, a new evaluation of the optimized model can be necessary, hence going back to the verification and validation stage.

8.7 Operation

After the trained model is deployed, it is available for use as a stand-alone application or as a function for integration into other applications. ML models are used in batch mode with fixed sets of production data or in continuous mode on real-time streams of production data. The output of a model can be sent back to an application or in some cases it can be interpreted by a human.

© ISO/IEC 2022 – All rights reserved

Maintenance and repair: During the operation phase, the ML model and the corresponding application are subject to maintenance and repair needs similar to other AI systems or conventional software systems. Specificities of ML in that phase pertain to the update and monitoring processes.

Update: The retraining process is carried out using training data that reflects the characteristics of current production data. Updates can also implement algorithmic changes which, depending on the change, can require only some retraining or training a new model from scratch.

Monitoring: The execution of the ML model is typically monitored during the operation phase, since factors such as data drift, concept drift or poor generalization can affect performance. In the case of continuous learning, further care can be necessary, and monitoring can include additional evaluations of the ML model.

8.8 Example machine learning process based on ML pipeline

The ML pipeline processes can be applied to different learning techniques and ML model development. In this subclause, the processes are applied to supervised machine learning.

Figure 13 illustrates the different subcomponents of the ML system and illustrates how these processes can fit together. Note that not all of the processes described in the ML pipeline are depicted in Figure 13 as it is meant to be a simplified example. Some processes can be performed in a different order or performed in parallel. Furthermore, the invocation of tools during the various phases of model development as well as their dependency on data is depicted. The distinction is made between invoking tools for model development and the operation of processes on data.

© ISO/IEC 2022 – All rights reserved

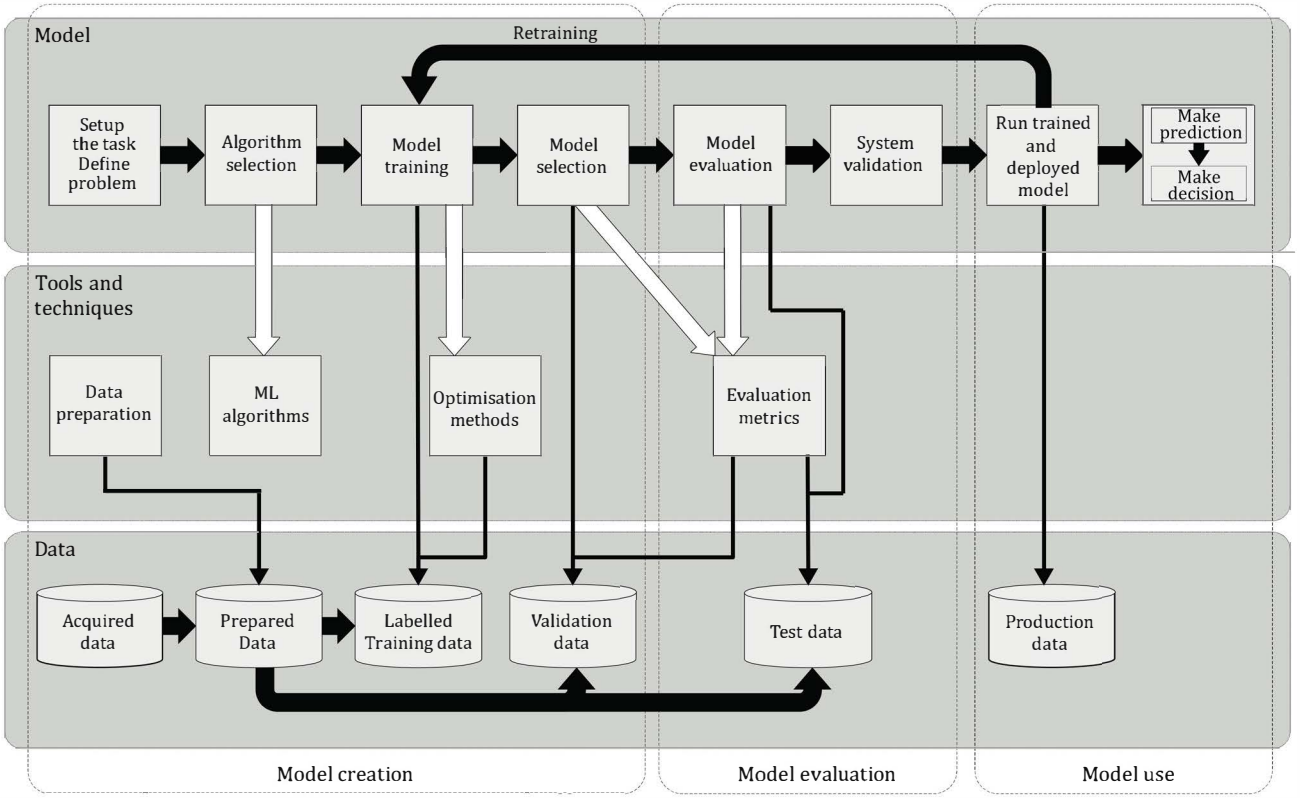


Figure 13 — Example supervised machine learning process based on the ML pipeline

In Figure 13, in the top model layer are the process steps for a system using supervised learning. In the middle software tools and techniques layer are the tools and data processing elements required to conduct the process steps. In the bottom data layer are the various data elements required throughout the pipeline.

Model creation initially involves the steps of "define problem" and "setup the task", which establish the goals and requirements of the solution. Then follows the "algorithm selection" step, which aims to choose an ML technique most suited to providing the required solution. The "model training" step involves the creation of the trained candidate model through the use of the labelled training data. Model selection computes evaluation metrics on validation data to select a final ML model among the candidates.

Model evaluation applies evaluation metrics on test data to assess the performance of the trained model. If sufficient, it is followed by system validation.

Model use involves the "run trained and deployed model" step which deploys the trained model produced during model creation, with the model operating on the production data for the ML application. Running the model on production data results in the system making predictions and making decisions based on the output of the model.

Retraining of the model can take place, either through the use of continuous learning, or based on an evaluation of the performance of the trained model in use. The performance evaluation can indicate a

© ISO/IEC 2022 – All rights reserved

need for additional training, possibly for some kinds of production data where the performance does not meet the requirements established for the system.

Figure 13 can be further utilised to describe data flows and their associated data use statements as presented in Annex A. Such statements utilise the taxonomy and data use statement format that is described in ISO/IEC 19944-1.

--`,,```,,,

Annex A
(informative)

Example data flow and data use statements for supervised learning process

A.1 General

This annex demonstrates the use of ISO/IEC 19944-1 to describe data flows and develop data use statements for ML. The content of this annex is exemplary and can be built upon further if necessary. By applying ISO/IEC 19944-1 to ML processes, the building blocks and fundamental elements needed for AI accountability and transparency can be developed.

A.2 Data flows in the supervised machine learning process

A.2.1 General

This clause provides example data flow descriptions based on the ML process shown in Figure 13.

It should be noted that the data flows in A.2.2 are not described as flows between functional components within the cloud computing reference architecture (ISO/IEC 17789), but instead provide a view of data flows through the various phases of the ML pipeline and ML model development processes.

A.2.2 Data flow descriptions

A.2.2.1 Data flow 1

The following data flow can be identified in Figure 13 and described as:

Acquired data→ prepared data

Data is acquired from one or more sources with the intent to build a dataset for ML. Numerous processes are ordinarily applied to a dataset to make it ready for the model training phase of the ML pipeline.

Such processes are described in 8.3.

A.2.2.2 Data flow 2

The following data flow can be identified in Figure 13 and described as:

Prepared data→ labelled training data

For supervised machine learning, the ground truth of the target variable must be specified. The value of the target variable is known as its label. In practice, a proxy can be used as the ground truth and proxies can introduce errors and bias.

A.2.2.3 Data flow 3

The following data flow can be identified in Figure 13 and described as:

Run trained and deployed model→ make prediction

When the model is deployed and run, production data is applied to the model. The model generates output data, usually with probabilities assigned to the outputs of the model. These outputs can in

© ISO/IEC 2022 – All rights reserved

turn be used to make a decision, either automated or taken as recommendations by humans to make decisions.

A.3 Data use in machine learning

A.3.1 General

This clause builds example data use statements based on the data use statement structure described in ISO/IEC 19944-1:2020, 10.2. The following examples of data use within supervised machine learning are based on the data flows described in A.2.2, based on Figure 13.

A.3.2 Example data use statement A

This example considers a use case where ML is used by an insurance provider. Within the data flow described in A.2.2.1 (data flow 1), the following example data use statement can be formulated:

EXAMPLE Data preparation tools use anonymized financial data from home insurance claims to improve and prepare the acquired data for ML model development.

A.3.3 Example data use statement B

This example considers a use case where ML is used in an automated road vehicle's object recognition systems. Within the data flow described in A.2.2.2 (data flow 2), the following example data use statement can be formulated:

EXAMPLE "Model training" process uses aggregated and labelled sensor data from onboard vehicle cameras to train the object recognition ML model.

A.3.4 Example data use statement C

This example considers a use case where ML is used in the predictive maintenance systems of an aircraft engine. Such systems include temperature sensors to help monitor engine performance. Within the data flow described in A.2.2.3 (data flow 3), the following example data use statement can be formulated:

EXAMPLE Input device sensor data from engine mounted temperature sensors is used by the ML model to predict failure of components and improve engine performance and maintenance.

© ISO/IEC 2022 – All rights reserved

Bibliography

[1] STUHLSATZ A., LIPPEL J., ZIELKE T., Feature Extraction With Deep Neural Networks by a Generalized Discriminant Analysis. IEEE Transactions on Neural Networks and Learning Systems. 2012, **23** (4), 596—608. doi: 10.1109/tnnls.2012.2183645

[2] MACLENNAN B.J., Connectionist Approaches. International Encyclopaedia of the Social & Behavorial Sciences. 2001, 2568-2573. doi: 10.1016/b0-08-043076-7/00537-4

[3] ROSENBLATT F., The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review. 1958, Nov, **65**(6):386-408. doi: 10.1037/h0042519

[4] RUMELHART D., HINTON G., WILLIAMS R., Learning representations by back-propagating errors. Nature 1986, **323**, 533–536. doi: 10.1038/323533a0

[5] ELMAN Jeffrey L., Finding structure in time. Cognitive science. 1990, **14**.2, 179-211. doi: 0.1016/0364-0213(90)90002-E

[6] HOCHREITER Sepp, SCHMIDHUBER Jürgen, Long short-term memory. Neural computation.1997, **9**.8, 1735-1780. doi: 10.1162/neco.1997.9.8.1735

[7] LECUN Y, BOTTOU L, BENGIO Y, HAFFNER P, Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998, **86**(11), 2278-2324. doi: 10.1109/5.726791

[8] HECKERMAN D., A Tutorial on Learning with Bayesian Networks. *Innovations in Bayesian Networks. Studies in Computational Intelligence*, **156**. Berlin, Heidelberg: Springer. 2008. doi: 10.1007/978-3-540-85066-3_3

[9] FAWCETT T., ROC graphs: Notes and practical considerations for researchers. Machine learning. 2004, **31**, 1—38.

[10] DELGADO R, TIBAU X. A., Why Cohen's Kappa should be avoided as performance measure in classification. PloS one. 2019, **14**(9). doi: 10.1371/journal.pone.0222916.

[11] GILES M., Foody. Explaining the unsuitability of the kappa coefficient in the assessment and comparison of the accuracy of thematic maps obtained by image classification. Remote Sensing of Environment. 2020, **239**. doi: 10.1016/j.rse.2019.111630

[12] ISO/IEC 19944-1:2020, *Cloud computing and distributed platforms — Data flow, data categories and data use — Part 1: Fundamentals*

[13] YANG Q., ZHANG Y., DAI W., PAN S., Transfer Learning. Cambridge: Cambridge University Press. 2020. doi:10.1017/9781139061773

[14] ISO/IEC 20889, *Privacy enhancing data de-identification terminology and classification of techniques*

[15] *Open neural network exchange.* https://onnx.ai/

[16] *Neural network exchange format.* https://www.khronos.org/nnef

[17] ISO/IEC 17789, *Information technology — Cloud computing — Reference architecture*

© ISO/IEC 2022 – All rights reserved

ICS 35.020

Price based on 36 pages

© ISO/IEC 2022 – All rights reserved