Estimating Degradation of Machine Learning Data Assets

LARA MAURI, Università degli Studi di Milano, Italy
ERNESTO DAMIANI, Khalifa University, UAE

Large-scale adoption of Artificial Intelligence and Machine Learning (AI-ML) models fed by heterogeneous, possibly untrustworthy data sources has spurred interest in estimating degradation of such models due to spurious, adversarial or low quality data assets. We propose a quantitative estimate of the *severity* of classifiers' training set degradation: an index expressing the deformation of the convex hulls of the classes computed on an *held-out* data set generated via an unsupervised technique. We show that our index is computationally light, can be calculated incrementally and complements well existing ML data assets' quality measures. As an experimentation, we present the computation of our index on a benchmark convolutional image classifier.

CCS Concepts: • **Artificial Intelligence** → **Machine Learning**; • **Data Management** → *Data Assets.*

Additional Key Words and Phrases: data assets, ML models

ACM Reference Format:
Lara Mauri and Ernesto Damiani. 2021. Estimating Degradation of Machine Learning Data Assets. *ACM J. Data Inform. Quality* 1, 1, Article 1 (January 2021), 15 pages. https://doi.org/10.1145/3446331

1 INTRODUCTION AND BACKGROUND

It is well recognized that no important decision should be made based on poor quality data. In the evolving landscape of data-driven Artificial Intelligence technologies and analytics, where data play a crucial role in shaping the learning process and decision-making, it is even more crucial not to take data quality for granted [17]. Classic approaches to data quality often focus on a specific facet of quality, such as the detection of outliers. In this paper, we focus on a distinct though related topic, i.e. estimating the degradation of data assets used by Artificial Intelligence and Machine Learning (AI-ML) classification models, due to the injection of spurious, adversarial or low quality data. This is a frequent scenario in Internet of Things, where faulty or compromised sensors may lower the overall quality of the data on which inference is based [14].

We start from the notion that all data assets can be considered artefacts with a certain quality. In the context of AI-ML, it is possible to consider data quality as a set of properties the information fed to (and produced by) AI-ML models should have [21]:

- *Accuracy*: Reflects the difference between the data and an underlying "true value". This difference includes sensor errors and any error introduced at ingestion (e.g., by quantization).
- *Consistency*: Expresses compliance of data to user-defined rules; for example, the age of a sibling cannot be equal or greater than the one of a parent.
- *Timeliness*: Expresses difference between the time when data is made available and a deadline. It denotes if data is being received in time for performing a task.

Authors' addresses: Lara Mauri, Computer Science Department, Università degli Studi di Milano, Milan, Italy, lara.mauri@unimi.it; Ernesto Damiani, Center for Cyber-Physical Systems (C2PS), Khalifa University, Abu Dhabi, UAE, ernesto.damiani@ku.ac.ae.

1936-1955/2021/1-ART1 $15.00
https://doi.org/10.1145/3446331

- *Completeness*: Expresses whether the number of data points available is enough for the intended purpose (e.g., computing a formula or training an ML model).

For the purposes of this paper, two other aspects of ML data assets' quality are central: data *uncertainty* and *trustworthiness*, which can be seen as two different aspects of *indeterminacy*: how much it is known about the consequences of using a data item. Uncertainty is mainly related to imprecision affecting raw data, while indeterminacy is an inherent issue for pre-processed data assets (for example, training sets) that are targeted by modifications aimed at changing the ML model's behavior in production. In all cases, spurious items (adversarial, poisonous or simply low-quality data points) that become part of data assets can impair ML model accuracy, requiring filtering or other alleviation measures.

1.1 Model Degradation

Even if there is no tampering with data assets, ML models' predictive performance can decrease over time as they are fed with new data. This phenomenon is known as *model degradation*. There are two major heuristics for identifying and tracking model degradation.

- *Explicit quantification*: Conceptually, the simplest way to identify model degradation is to explicitly quantify the amount of the model's performance decrease or its trend. However, measuring the accuracy of a deployed ML model on live input data is a notoriously difficult problem. This difficulty arises because for each input one needs access to both the model's output, i.e. the classification or prediction value proposed by the model, and the ground truth, i.e. the true value. Even when the model's outputs and the ground truth values are both available, it may be difficult to synchronize them to check the performance decrease's trend. Consider a ML model that predicts the total amount of rain that will fall in a month. The actual value will only be observed monthly, so appreciation of the rate at which the model's performance is decreasing may involve considerable delay.
- *Distribution comparison*: These techniques compare the probability distributions of the input features to those of the training data to infer model degradation. This *a priori* technique can be helpful when it is not possible to observe the ground-truth values. This method can be improved by issuing alerts when the divergence between key features distribution is significant. However, in production it is not always possible to continuously monitor all features' distributions.

1.2 The AI-ML Life Cycle

Following the line of the standardization effort within the ISO technical committee IEC JTC 1/SC 42 on Artificial Intelligence, our discussion of ML data assets will be driven by the notion of *AI-ML applications life cycle* (in short, *AI-ML life cycle*), which defines the phases that organizations follow to implement *Machine Learning* models. Generally, the ML life cycle involves a series of independent stages that are performed in an iterative fashion. Most of these stages use and/or generate specific *data assets*, which are the focus of this paper. Figure 1 shows our generic reference AI-ML life cycle model.

The first step of any ML project is to reach a clear understanding of the business context, articulating a definition of the business goals to be achieved along with the data required to achieve them. The *data ingestion* phase of the life cycle collects all data needed for achieving the business goal. Ingested data are usually arranged as *multidimensional data points* (also called *data vectors*). During *data exploration*, data are inspected and displayed by using plots or charts. The *pre-processing* phase creates a consistent data set that is suitable for training, testing and evaluation. Techniques to clean, wrangle and curate the data are employed to convert data to the right format, remove

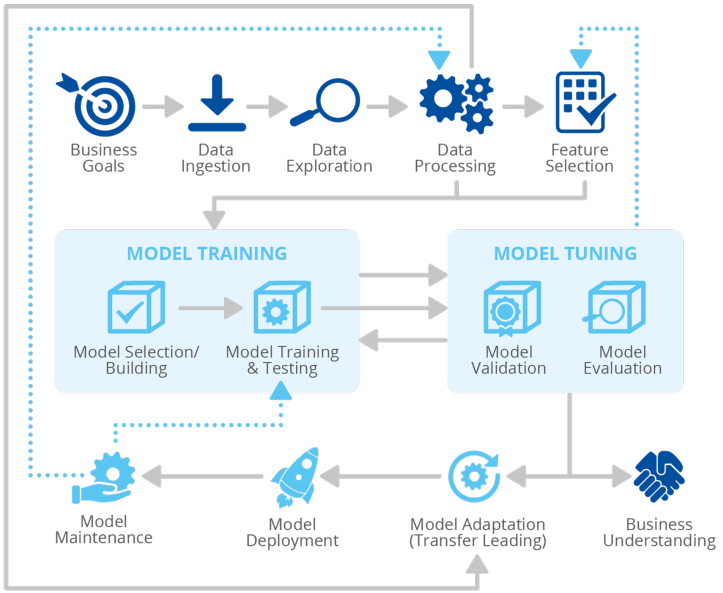

Fig. 1. The AI-ML life cycle. Figure courtesy of ENISA Report *AI Cybersecurity Challenges – Threat Landscape for Artificial Intelligence* [11] and ISO/IEC JTC 1/SC 42/WG 2.

noise and anonymize it as needed. The phase of *feature selection* reduces the number of dimensions composing each data vector to obtain a reduced data set that better represents the underlying problem. The core phase of the AI-ML models' development process is *model training*, which includes selecting the ML model structure and learning the model's internal parameters from data. Depending on the nature of available data and the business goal, different ML techniques can be used. In this paper, we focus on *supervised machine learning*. In the training process of a supervised ML system, the learning algorithm generates by trial-and-error an ML model that works well (i.e. delivers the expected output) on a set of training data and - hopefully - will work on other data as well. Essentially, training is done by computing the error made by the model on the training set data and using it to adjust the model's internal parameters to minimize the error. The achieved error reduction is continuously verified during training by feeding the ML model with some data put aside for testing. While learning sets the values of the ML models' internal parameters, the so-called *hyper-parameters*, which control how the training is done (e.g. how the error is used to modify the ML model's internal parameters), are set separately during the *model tuning* phase. While being tuned, the ML model is also validated, to determine whether it works properly when fed with data collected independently from the original data set. *Transfer learning*, which is sometimes considered part of the model's training phase, sources a pre-trained and pre-tuned ML model and applies it to a different but related problem. The transition from ML model development to production is handled by the *model deployment* phase: the trained/validated ML model is integrated into a production environment, where it can make practical decisions based on the data presented to it. Since the production data landscape may change over time, in-production ML models require continuous monitoring. The life cycle phase of *model maintenance* serves precisely to monitor the ML model and retrain it when needed. Finally, in the *business understanding* phase, ML model users gain insight on the impacts of the model in terms of the value it creates for business.

Of course, the AI-ML life cycle we outlined covers only a part of the AI applications landscape; however, it supports the derivation of a fairly complete *data asset model* for supervised ML. In this paper, we refer to data assets produced and consumed within the ML life cycle shown in Figure 1.

The remainder of the paper is organized as follows: Section 2 introduces the notion of ML data poisoning. Section 3 describes the different types of ML data assets and the role they play in ML models' development. Section 4 describes our technique for estimating the severity of AI-ML models' performance degradation due to spurious additions to training sets, computing an *held-out data set* (Section 4.1) to be used as a "gold standard" for the data assets (training and validation data) used for AI-ML training. Section 5 discusses how *Convex Hulls* (CHs) can be used to approximate class regions of classification models and introduces our CH-based degradation severity index for ML data assets, while Section 6 explains how to compute it. Section 7 reports the experimental evaluation of our approach on the *Belgium Traffic Sign Classification Benchmark* (BTSC). Finally, Section 8 draws our conclusions and outlines our future work in this area.

2 ML DATA POISONING

ML models can be the victim of "data vandalism" as well as of intentional attacks [3, 19]. When training data is exposed to tampering, any ML model built upon it can be fooled and misled in ways that can have profound implications. In the ML security domain, attackers' capability is defined based on the influence the attacker has on the data [2]. Specifically, attacks can be *causative*, if the attacker can inject spurious training data or manipulate existing ones, or *exploratory*, if the attacker cannot influence the training process but can craft malicious samples at test time. These attack scenarios are commonly referred to as *poisoning* and *evasion*, respectively. Our technique aims to estimate the degradation of the data assets used for training an ML model, and hence, it is concerned with the former type of attack. In the following, we describe poisoning in more detail:

- *Append* attacks add random data points to the training set and target the ML model's availability, i.e. prevent it from computing any meaningful inference. Other attacks (also known as *insert* attacks) threaten the ML model's integrity, i.e. the inferences it will compute once deployed. Insert attacks rely on data modification: the attacker changes, removes from or adds data points to the training set.
- *Label Tampering*: The simplest poisoning attack is manipulating the training set's labels. Attackers can randomly draw new labels for a part of the training set, or choose them to cause maximum disruption, manipulating input values to shift classification boundaries or to add invisible watermarks that can later be used to "backdoor" into the model.

However, it is important to underline that poisoning is not necessarily the consequence of a deliberate act. It may occur unintentionally, e.g. when a faulty sensor contributes to the build-up of a training set, or when wrong pre-processing primitives have been used by mistake to augment/complete the training data.

3 DATA ASSETS IN THE ML LIFE CYCLE

Digital assets used throughout the ML life cycle include data, models, actors and processes. Among them, data is one of the most important asset categories to consider. We now briefly describe the different types of data assets and the role they play at each ML development stage.

- *Raw data* refers to data gathered in the context of the problem to be solved. Raw data includes any kind of structured or unstructured information that needs to be transformed before being ready for analysis.

- *Labeled data* denotes data points tagged with one or more informative labels that identify certain properties or characteristics of the data. Usually, labeling makes data suitable for training supervised ML models.
- *Training data* are used in the model training stage and constitute the portion of data that will be used to learn the ML model's internal parameters.
- *Augmented data* are labeled training data synthetically created from existing samples. The underlying assumption is that by extracting information from the original training data, it is possible to increase labeled data's diversity and hence, improve the ability of the ML model to generalize.
- *Testing data* follow the same probability distribution as the data in the training data set, but are independent from them. Testing data is used during the model training phase to assess the performance of the ML model fitted on the training data set.
- *Validation data* differ from ordinary labeled data only in their usage. The validation data set is used to determine the most appropriate time to stop the ML model's training in order to avoid over-fitting.
- *Held-out data* include inputs of special interest for the application. The rationale for held-out data is that if the ML model shows good aggregate accuracy, it can be hard to notice whether its accuracy on specific inputs remains acceptable.

4 ESTIMATING SEVERITY OF AI-ML MODELS' DATA ASSETS DEGRADATION

Model degradation is noticed when the accuracy in production deviates sensibly with respect to the accuracy measured in validation. We propose a novel system that allows a fast estimate of the severity of AI-ML models' performance degradation due to spurious additions to training sets. Our system is designed to be ML model-independent and computationally lightweight. It consists of two major components: an *held-out data set generator* and a *convex-hull engine*. Our notion of held-out data set should not be confused with the *simple hold-out* [25] technique used for ML models' validation, where some manually labeled data kept aside from the training set are used for model validation purposes. Rather, we take a validation-agnostic point of view: ML model training can use either *simple hold-out* or, more probably, *cross-validation* alternating training and test data.

Whatever the validation technique, checking the performance of AI-ML models against an additional *held-out data set* has become customary when models are trained using uncertain data, i.e. data that may or may not be representative of the data space at deployment [22]. Our approach is to start from the training set to generate an held-out data set that can be used as a "gold standard" for quick degradation assessment. Our convex hull engine comes in at this point: we use changes in convex hull of the classes computed by the ML model under assessment on the held-out data set to estimate the severity of the model's degradation. It is important to remark that if the training set is later added to, our technique allows to fuse successive severity estimates, in the line of data uncertainty measures based on Dempster-Shafer theory of evidence [21].

In current practice, held-out data sets are mostly used *during* the training phase for *early stopping* i.e. stopping training when the error on the held-out data set increases, in order to prevent model over-fitting. For the purposes of this paper, we refrain from interactive use of the held-out data set, as the error on the held-out data set may fluctuate during training. We use held-out data only *after* training, in order to provide a quality benchmark for trained models.

4.1 Using Latent Variables to build the Held-out Data Set

In our approach, the held-out data set is a "gold standard" [22] for the data assets used for AI-ML training. In principle, it can be built by manually selecting labeled data points that are considered certain, e.g. because they come from trusted sources or were measured at a time where no spurious

data injection was possible. Also, held-out data may be composed of hand-picked data points whose classification is going to be verified during ML model audit or certification at the request of an auditing authority [8]. However, manual selection does not scale well, and cannot easily generate held-out sets comparable to the training set size.

In this section we show how, for the purpose of quality assessment, the held-out data set can also be built in an unsupervised way, modeling training data as *observables*, or *manifest variables*, and extracting held-out data as *latent variables*, which are not directly observable.
Latent class models were first introduced by Lazarsfeld in the Fifties of the last century [15] and were later made computationally efficient by Goodman [12]. Often, the observed variables are modeled as classifications assigned by different judges. In our own previous work, we used latent class models to correct symmetric disagreements that appear to result from multi-rater bias [23]. The latent class model underlies various unsupervised AI-ML techniques, starting from the seminal Auto Class model [7]. We propose an unsupervised latent class model to build the held-out data set for the degradation assessment of AI-ML data assets.

To fix our mind, let us call D the ML data asset whose degradation we want to assess. We define a *window of observation* W of k-dimensional (complete or partial) data points taken from D. The scope of this window is decided heuristically, according to temporal locality (e.g., the data points may be repeated sensor readings) or to spatial locality (e.g., the data points may be subset of pixels sampled from a set of images showing the same object).

To distillate a single "golden" data point out of W we consider k latent variables, one for each component of the data points in W. We apply the classic naive Bayes approach [12]: we assume the values of data in W (the observed variables) to be all conditionally independent of one another, given the value of each latent variable [1]. Let H, K, J, and L be the observed variables, i.e. the components of the data points in W, and let X_H, X_K,X_J and X_L be the corresponding latent variables, i.e. the components of the "golden" data point we want to compute. For each latent variable X_I ($I \in \{H, K, J, L\}$) our latent model is:

$$p(i_1, i_2, ..., i_k, x_i) = p(i_1|x)p(i_2|x)...p(i_k|x)p(x_i) \tag{1}$$

where the equality holds by our assumption of observable variables' independence. The estimate of the conditional probabilities in Eq. 1 can be carried out by replacing the probabilities on the right-hand side of Eq. 1 by the corresponding estimates.

Several competing techniques (including *max-likelihood*) are available for computing such an estimate. Recent research [5] has proposed the *Laplacian rule of succession* as an alternative method to estimate the conditional probabilities of the candidate x values. The classic Laplacian succession rule [13] provides a setting where, repeating n times an experiment that can result in a success or failure, and getting s successes, and $n - s$ failures, we can estimate the probability of success of the next time. In our setting, "success" corresponds to $x = i_j$. For the sake of computational speed, we use a pre-computed Laplace distribution [24], and estimate each $p(i_j|x)$ as the probability associated to i_j by the Laplace distribution centred in x. Once estimates of $p(i_j|x)$ have been computed for all possible values of x in the representation interval, the most probable latent value is assigned to x. This way, the values chosen to build the held-out data set are the ones for which the agreement among the observable variables within the observation window W is highest.

As a simple example (the complete computation carried out for our experiments is described in Section 7.1), let us consider three integer variables (i.e., $k = 3$) taking values in the interval $[0 - 2]$ and an observation window W containing 4 tri-dimensional points. For each possible value of the

[1]This assumption may of course require filtering out highly correlated features. This is a customary practice in data pre-processing, and lies outside the scope of the paper.

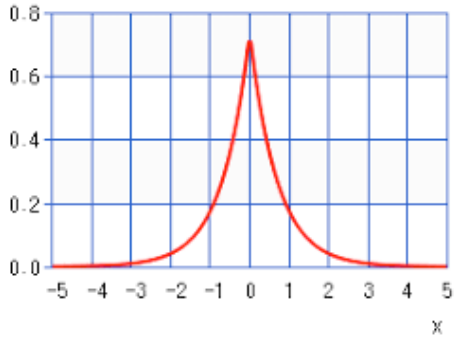

Fig. 2. A differential Laplacian distribution. Values on the x-axis are the difference between x and i_j.

Table 1. Three observable variables.

i_1	i_2	i_3
1	2	0
1	1	2
2	2	1
0	0	0

latent variable x (one of the 3 components of the golden data point), we compute $p(x)$ and estimate $p(i_j|x)$ by querying the Laplacian distribution centered in x (we obtain $L(i_j, x)$).

Let us now estimate x_{i_1} using Eq. 1. Looking at the first column of Table 1, for $x_{i_1} = 0$ we get $p(x_{i_1}) = 0.25$, to be multiplied by $L(1,0)L(1,0)L(2,0)L(0,0)$. For $x_{i_1} = 1$ we get $p(x_{i_1}) = 0.5$, to be multiplied by $L(1,1)L(1,1)L(2,1)L(0,1)$. For $x_{i_1} = 2$ we get $p(x_{i_1}) = 0.25$, to be multiplied by $L(1,2)L(1,2)L(2,2)L(0,2)$. By comparison, the golden value for the first latent variable is $x_{i_1} = 1$. The computation can be repeated for x_{i_2} and x_{i_3}, obtaining the "golden" tri-dimensional vector $GV = (1, 2, 0)$.

5 A SEVERITY INDEX BASED ON CLASSES' CONVEX HULLS

Most classification models make use of a distance metric in some n-dimensional space to compute separation surfaces between classes. Generally speaking, poisoning (cf. Sect. 2) tries to deform such surfaces at training time, in order to "push" future data points across them at run-time. Intuitively, the severity of an adversarial input to an ML model is linked to the attack's effectiveness in achieving such deformation; but if the structure of the ML model being attacked is unknown, it is difficulty to quantify it *a priori*. To achieve model independence for our severity index, we look at the classes computed on the held-out data set by a virtual *Nearest Convex-Hull* (NCH) classifier.

Computational geometry models have been used since long to approximate class regions of data classification models. In geometric terms, we can represent each class by the smallest convex region enclosing all points in the class. Such an envelope corresponds to the so-called *Convex Hull* (CH). Formally, the CH of a set of points S in a n-dimensional space is the smallest convex set that contains S. A point in S is an *extreme* point (with respect to S) if it is a vertex of the convex hull of S. If S is finite then the convex hull of S is a convex polytope with vertices, edges and facets making up its boundary. Figure 3 shows the convex hull of a set of bi-dimensional points.

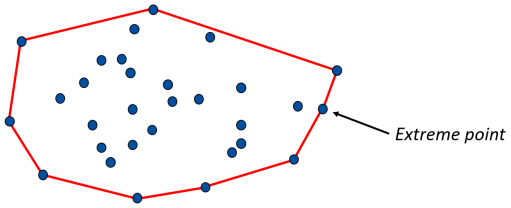

Fig. 3. The convex hull of a set of bi-dimensional points.

The "hard-margin" version of the NCH classifier model assigns a data point x to the group of training points whose CH is closest to x. Performing NCH classification involves solving an optimization problem to find the distance of the input object to each class. Starting from the Eighties, several algorithms for doing so have been proposed under the general heading of finding the minimum distance between convex sets [4]. The NCH model has the property that the extent of proximity of a test point x to a given class is determined without taking into consideration objects from other classes. In classic separable cases, however, a problem arises if the object x to be classified lies inside the *CH*s of two or more classes, since its distance to these CHs is equal to zero, leaving the classification of x undetermined. This problem was solved by introducing the *soft-margin* version of the NCH classifier [16], where overlap between a data point x and a given class C's CH is penalized linearly or quadratically.

5.1 CH Deformation

For the sake of speed, we do not directly apply NCH classification to compute our severity metrics. Rather, we compute the deformation of the CHs of the classes computed by the ML model M under evaluation on an held-out data set computed starting from M's training set. Intuition suggests that for each point p to be classified, the best choice for an attacker would be to cut the CH point that is closest to p. Figure 4 shows the effect of cutting a CH point: the closest CH to the candidate point x, originally the one depicted in red (upper part of Figure 4), becomes the one depicted in green after one point of the red class' CH has been removed (lower part of Figure 4). This cut corresponds to a

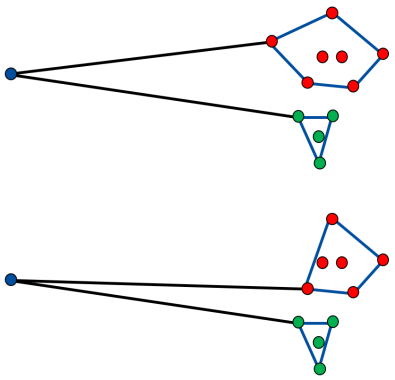

Fig. 4. The link between classes' CH deformation and NCH classification accuracy.

CH deformation, as stated in Theorem 5.1 below.

Theorem 5.1. *For any point p outside the CH of a class C, the point $c \in C$ closest to p lies on the CH of C.*

Proof. Without loss of generality, let us consider p as the origin of a system of coordinates, and let us fix the direction of the x-axis along the line connecting p and c. Then, c is the point of C with the smallest x coordinate. Therefore c lies on the CH of C.

In the following, we will assess the severity of the degradation of a generic ML model M via the deformation of the CHs of the classes computed by M on a held-out data set.

Definition 5.2. Given a model M, let $CH(C)$ be the convex hull of a class C of the VNCH model corresponding to M. If M is modified by a change in a data asset, resulting in a model M', we define the severity S of the degradation as follows:

$$S = \frac{| \, CH(C) \cup CH(C') \, |}{| \, CH(C) \cup CH(C') \, | + | \, CH(C) \cap CH(C') \, |} \tag{2}$$

where $|CH|$ denotes the area within the convex hull CH and C' is computed by the VNCH model corresponding to M' on the same set of data points belonging to model M's class C. Of course $C' \neq C$, as different data assets were used for training M'. Specifically, C' may miss some points that were in C and include some points that were not in C. Note that, when no point of $CH(C)$ is missing in C', and any additional point of C' fall inside $CH(C)$ (for example, when $C = C'$) $S = 1/2$. Our severity index S is modeled after the Dempster-Schafer measures used in data quality [21]. Indeed, such measures are able to represent incomplete knowledge, and update it as new information comes. We looked for the simplest representation that was at the same time easy to compute and provided the following properties:

- *Representation interval*: Our severity index S takes values inside the interval $[0, 1]$, which helps comparing it to individual data assets' quality measures.
- *Weak Monotonicity*: S is weakly monotonic with respect to random points' addition. Given two sets D and D' of random data points with $D \subseteq D'$, $S(D) \leq S(D)$. This property is also held by most data quality indexes [21].

6 COMPUTING THE S INDEX

There exist several methods for computing the CH of a set of finite number of points in a Euclidean space. Discussions on which is the best performing CH algorithm are outside the scope of this paper. We used Quickhull [1], which is able to compute the convex hull in 2D, 3D and higher dimensions with the average time complexity of $O(nlogn)$. The recursive nature of the Quickhull algorithm allows a fast implementation and proves efficient in practice, though there are also variants of the algorithm that can make it run much faster[2].

6.1 Aggregation

Once class-based S indexes have been computed, they need to be aggregated to provide the degradation metrics for the entire model. Several aggregation techniques for uncertainty measures have been proposed [10], which can be briefly summarized as follows:

- *Averaging*. This technique aggregates numerical data points by taking their average or median.
- *Bayesian Systems*. This technique takes data points as samples and computes their *typical value* by means of a conditional Probability Density Function (PDF). The Bayesian approach

[2]When needed, it is possible to approximate the CH of the union with the union of the CHs, and the CH of the intersection with the intersection of the CHs. This yields $O(|CH|)$ complexity, where usually $|CH| << n$

Table 2. Summary information of traffic sign data set.

Class	Type	Shape	Color
C_1	Warning	Triangular (pointing upward)	Red
C_2	Mandatory	Circular	Blue
C_3	Stop and Yield	Octagonal or Triangular (pointing downward)	Red
C_4	Prohibitory	Circular	Red
C_5	Information	Square or Rectangular or Diamond	Blue or Red

is widely used in recommender systems [9]. It also underlies the held-out data set generation proposed in this paper.

- *Belief Models.* In belief-based aggregation, data points are represented as beliefs and some technique (usually, *max-likelihood*) is used to select the most probable one. As the sum of beliefs over all possible data points does not necessarily add up to 1, belief values are easier to generate and process than probabilities.
- *Fuzzy arithmetics.* Fuzzy numbers can be used to represent uncertain values; in this case, aggregation is computed as fuzzy average.
- *Flow Models.* Data points are modeled as paths within probabilistic processes like Markov chains. Aggregation is performed by taking the most probable.

To estimate the overall level of degradation of the classification model M, we need to compute an aggregation putting together the degradation indexes S_{C_i} associated to M's classes $C_1, C_2, ..., C_n$. We follow the literature [9] in using the classic *Ordered Weighted Averaging* operator [26], where the order takes into account the importance of each source. The overall severity index S_M is computed as follows:

$$S_M = \frac{1}{n} \sum_{i=1}^{n} \omega_i S_{C_i} \tag{3}$$

where $\omega_i = \frac{n-i-1}{n+1}$ and n is the number of classes.

7 EXPERIMENTAL EVALUATION

We carried out an experimental evaluation of our approach on the publicly available *Belgium Traffic Sign Classification Benchmark* (BTSC) data set[3], using a Convolutional Neural Network (CNN). BTSC is part of the widely adopted Belgium Traffic Sign Dataset, which features multiple, calibrated images of traffic signs. The subset we used for our experiments originally contains 62 classes of traffic signs with 4591 images in the training set and 2534 images in the test set. On average for each physically distinct traffic sign three images are available with different position/orientation, scale and illumination. The actual data set we used to train the CNN is a gray-scale remapping of the original BTSC data. Based on both shape and color features and types of message they communicate, we split BTSC into five macro classes of traffic signs, namely *warning*, *mandatory*, *stop and yield*, *prohibitory* and *information*. Table 2 summarizes the specification of the data set used for training according to our remapping, whereas our manual prioritization of the five classes is given in Table 3.

[3]https://btsd.ethz.ch/shareddata/

Table 3. Prioritization of the classes.

Prioritization	Type	Class
1	Stop and Yield	C_3
2	Prohibitory	C_4
3	Mandatory	C_2
4	Warning	C_1
5	Information	C_5

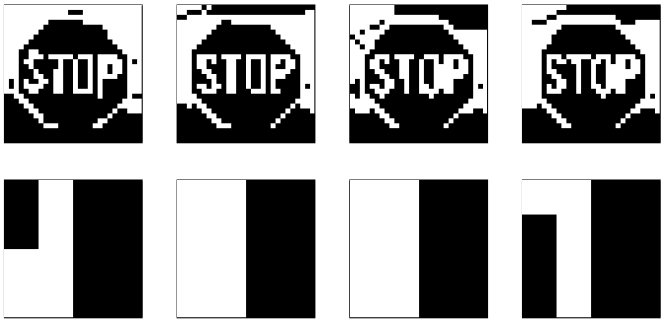

Fig. 5. A binary image from the training set (above) and 16 pixel areas included in the window of observation W (below). We used the classic Otsu's algorithm for thresholding [20].

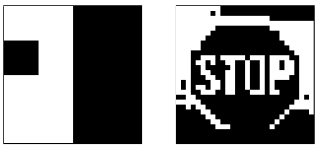

Fig. 6. A "golden" data point (left) and the held-out set image embedding it.

7.1 Building the held-out data set

Before going into the details of our experimentation, we show the computation of an image belonging to the held-out data set for our traffic sign classifier, applying the approach described in Sect. 4.1. For the sake of conciseness, we show the generation process on binary versions of the BTSC images. We define an observation window W composed of four data points. Each data point is a sub-area of a training image, namely a square whose side is four pixels, at the center of the 64-pixel side image. It is important to remark that the definition of an observation window W based on spatial locality may require zooming or other pre-processing activities, in order to normalize the window's size with respect to the entire image. This is however not the case for the BTSC training set, where cropping and zooming were applied, as customary, in the training set construction phase of the ML life cycle, yielding a uniform set of images.

Figure 5 shows four binary images taken from the classifier training set (above), and the 16- pixel ($4x4$) sub-areas included in W (below). Figure 6 shows a "golden point" distilled from the window W (right-hand side) and the image including it[4] (left-hand side). This lightweight technique allows for fast computation of large held-out data sets.

[4]The other pixels of the image were computed by simple majority voting.

Table 4. OWA-based aggregation of S indexes associated to each class for a sample run.

Class	*S Index*	
	10% Spurious Data (Accuracy: 0.972)	20% Spurious Data (Accuracy: 0.972)
C_1	0.501	0.501
C_2	0.500	0.501
C_3	0.500	0.503
C_4	0.501	0.500
C_5	0.500	0.500
S_M	**0.2499**	**0.2504**

7.2 Noise Insertion

In our experiments, we considered degradation due to the insertion of random noise data into the training set of an ML model. This type of *append* attack (see Sect. 2) degrades the performance of the ML classifier by adding new (spurious) data to the training set that may cause some held-out points to be mis-classified.

In order to assess degradation severity, we calculated the *CH* of each class before and after the insertion, computing *CH* deformation. We repeated our experiment considering varying percentages of spurious additions to the training set. Some relevant results are shown in Figure 7, where additions of spurious points amount to about 10 and 20 percent of the original training set. Each row shows the CH of a class *i)* before the insertion, *ii)* after the insertion, and *iii)* their overlap. For cases where the noise insertion has caused changes in the classification, one of the mis-classified images is also shown.

Case *(a)* shows the circumstance where 10% spurious additions resulted in no change of the convex hull of the class under consideration (i.e. C_1). All the points in this class were correctly classified and eventual changes in classification suffered by the other classes did not affect it. Looking at case *(b)*, also concerning class C_1, we notice that by adding 20% spurious data, the CH of the class has slightly changed. In particular, a point that before the noise insertion was on the class' CH has changed class after it, causing a decrease in the area defined by the hull. The image of the traffic sign for which the classification is no longer C_1 is shown. In cases *(c)* and *(d)*, which refer respectively to classes C_2 and C_3, the CH deformation is more pronounced. Of particular interest is the deformation for class C_3. This set of points is relatively small, but it is clearly visible that the classification change of an extreme point with very limited support (i.e., which is far from the other points of the class) has led to a large deformation of the CH. Lastly, like what happened in case *(a)* for class C_1, in case *(e)* the CH of class C_5 has not changed, but the cause is different: the few mis-classified points fell all within the CH and hence, mis-classification did not affect the class perimeter.

Results in Table 4 show that doubling the insertion of spurious data only increases marginally the index. Indeed, this trend is in accordance with the negligible variations in the per-class and overall accuracy in classification of the ML model. The Pearson correlation index between variations of our index and accuracy variations over four randomly chosen runs of the model was $\rho = 0.75$, suggesting that our index's changes are good predictors of accuracy variations.

8 CONCLUSION AND OUTLOOK

In this paper we have proposed a quantitative technique for assessing the degradation of ML models' data assets due to injection of spurious training data. Of course, once degradation has been detected

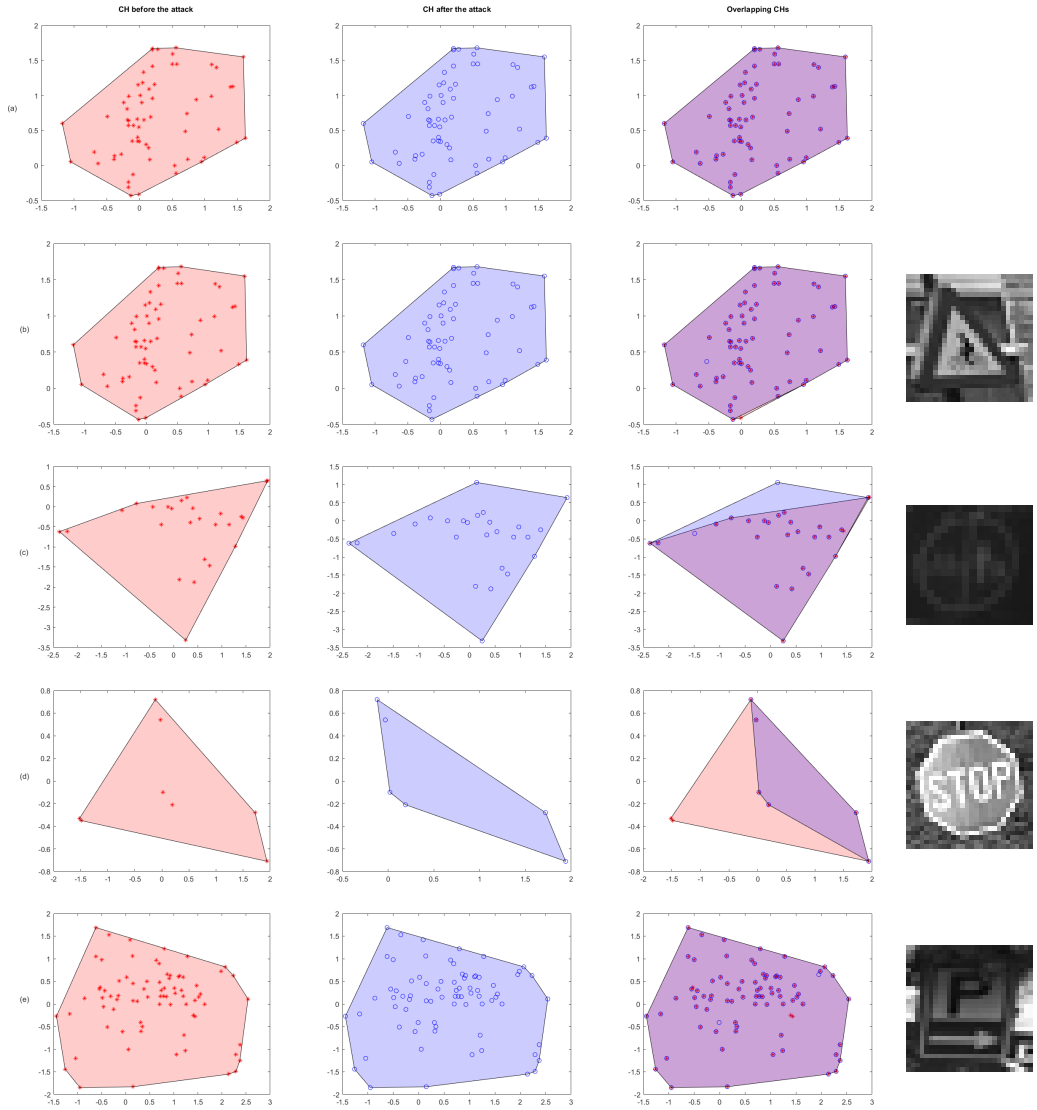

Fig. 7. Example of convex hull (CH) deformations as a result of poisonous additions. Rows (a)–(e) show the CH of the class under evaluation *i)* before the attack, *ii)* after the attack, *iii)* their overlap, and *iv)* greyscale representation of one of the mis-classified images (when applicable).

and measured, some countermeasures should be taken. Filtering would require applying the usual criteria for outlier identification; but, as our experimentation suggests, spurious data values can be "inliers", whose filtering is burdensome and error-prone [27], unless humans are kept in the loop [6]. We argue that a more viable strategy for alleviating ML data assets' degradation is the addition of *compensation data* that counterbalance the damage caused by the insertion of spurious data [18]. The acceptable amount (and collection cost) of compensation data depends on the specific data asset to be restored. In principle, contributing l labeled data that result in an increase ΔS_M of the

severity index for a model M should require the addition of enough compensation data items to offset ΔS_M, bringing S_M back to the original value.

We believe that the notion of compensation can be at the basis of collaborative protocols for ensuring ML data asset quality, as the actor doing the compensation needs not be the same who contributed the initial data. We plan to explore further compensation-based algorithms for preserving ML data assets' quality in our future work; a Blockchain-based approach based on the notion of *reciprocity*, where the computation of compensation is used as *Proof-of-Useful-Work* (PoUW) by community members has been described in our previous work [18].

REFERENCES

[1] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. 1996. The Quickhull Algorithm for Convex Hulls. *ACM Trans. Math. Softw.* 22, 4 (Dec. 1996), 469–483. https://doi.org/10.1145/235815.235821

[2] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. 2010. The security of machine learning. *Mach. Learn.* 81, 2 (2010), 121–148. https://doi.org/10.1007/s10994-010-5188-5

[3] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84 (2018), 317 – 331. https://doi.org/10.1016/j.patcog.2018.07.023

[4] James E. Bobrow. 1989. A Direct Minimization Approach for Obtaining the Distance between Convex Polyhedra. *The International Journal of Robotics Research* 8, 3 (1989), 65–76. https://doi.org/10.1177/027836498900800304 arXiv:https://doi.org/10.1177/027836498900800304

[5] Marco E. G. V. Cattaneo. 2016. Conditional Probability Estimation. In *Probabilistic Graphical Models - Eighth International Conference, PGM 2016, Lugano, Switzerland, September 6-9, 2016. Proceedings (JMLR Workshop and Conference Proceedings, Vol. 52)*. JMLR.org, 86–97. http://proceedings.mlr.press/v52/cattaneo16.html

[6] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-Loop Outlier Detection. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 19–33. https://doi.org/10.1145/3318464.3389772

[7] Peter Cheeseman and John Stutz. 1996. *Bayesian Classification (AutoClass): Theory and Results.* American Association for Artificial Intelligence, USA, 153–180.

[8] Ernesto Damiani and Claudio A. Ardagna. 2020. Certified Machine-Learning Models. In *SOFSEM 2020: Theory and Practice of Computer Science*. Springer International Publishing, Cham, 3–15.

[9] Ernesto Damiani, Paolo Ceravolo, Fulvio Frati, Valerio Bellandi, Ronald Maier, Isabella Seeber, and Gabriela Waldhart. 2015. Applying Recommender Systems in Collaboration Environments. *Comput. Hum. Behav.* 51, PB (Oct. 2015), 1124–1133. https://doi.org/10.1016/j.chb.2015.02.045

[10] Ernesto Damiani, Sabrina De Capitani di Vimercati, Pierangela Samarati, and Marco Viviani. 2006. A WOWA-Based Aggregation Technique on Trust Values Connected to Metadata. *Electron. Notes Theor. Comput. Sci.* 157, 3 (May 2006), 131–142. https://doi.org/10.1016/j.entcs.2005.09.036

[11] ENISA. December 2020. AI Cybersecurity Challenges – Threat Landscape for Artificial Intelligence. https://www.enisa.europa.eu/publications/artificial-intelligence-cybersecurity-challenges

[12] Leo A. Goodman. 1974. Exploratory latent structure analysis using both identifiable and unidentifiable models. *Biometrika* 61 (1974), 215–231. https://doi.org/10.1007/s00138-011-0391-3

[13] E. T. Jaynes. 2003. *Probability Theory: The Logic of Science.* Cambridge University Press. https://doi.org/10.1017/CBO9780511790423

[14] Aimad Karkouch, Hajar Mousannif, Hassan Al Moatassime, and Thomas Noel. 2016. Data Quality in Internet of Things. *J. Netw. Comput. Appl.* 73, C (Sept. 2016), 57–81. https://doi.org/10.1016/j.jnca.2016.08.002

[15] Paul F. Lazarsfeld. 1950. Studies in Social Psychology in World War II Vol. IV: Measurement and Prediction. *Journal of information security and applications* (1950), 362–4121. https://doi.org/10.1007/s00138-011-0391-3

[16] Qiangkui Leng, Zuowei He, Yuqing Liu, Yuping Qin, and Yujian Li. 2020. A soft-margin convex polyhedron classifier for nonlinear task with noise tolerance. *Applied Intelligence* (2020). https://doi.org/10.1007/s10489-020-01854-6

[17] Yang Liu, Lei Ma, and Jianjun Zhao. 2019. Secure Deep Learning Engineering: A Road Towards Quality Assurance of Intelligent Systems. In *Formal Methods and Software Engineering*. Springer International Publishing, Cham, 3–15.

[18] L. Mauri, E. Damiani, and S. Cimato. 2020. Be Your Neighbor's Miner: Building Trust in Ledger Content via Reciprocally Useful Work. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. 53–62. https://doi.org/10.1109/CLOUD49709.2020.00021

[19] Luis Muñoz-González and Emil C. Lupu. 2019. *The Security of Machine Learning Systems.* Springer International Publishing, Cham, 47–79. https://doi.org/10.1007/978-3-319-98842-9_3

[20] Nobuyuki Otsu. 1979. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics* 9, 1 (1979), 62–66. https://doi.org/10.1109/TSMC.1979.4310076

[21] Horacio Paggi, Javier Soriano, Juan A. Lara, and Ernesto Damiani. 2021. Towards the definition of an information quality metric for information fusion models. *Computers Electrical Engineering* 89 (2021), 106907. https://doi.org/10.1016/j.compeleceng.2020.106907

[22] Roland Roller and Mark Stevenson. 2015. Held-out versus Gold Standard: Comparison of Evaluation Strategies for Distantly Supervised Relation Extraction from Medline abstracts. In *Proceedings of the Sixth International Workshop on Health Text Mining and Information Analysis*. Association for Computational Linguistics, 97–102. https://doi.org/10.18653/v1/W15-2612

[23] Abdulhadi Shoufan and Ernesto Damiani. 2017. On Inter-Rater Reliability of Information Security Experts. *J. Inf. Secur. Appl.* 37, C (Dec. 2017), 101–111. https://doi.org/10.1016/j.jisa.2017.10.006

[24] M. C. K. Tweedie. 1947. Functions of a statistical variate with given means, with special reference to Laplacian distributions. *Mathematical Proceedings of the Cambridge Philosophical Society* 43, 1 (1947), 41–49. https://doi.org/10.1017/S0305004100023185

[25] S. Yadav and S. Shukla. 2016. Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification. In *2016 IEEE 6th International Conference on Advanced Computing (IACC)*. 78–83. https://doi.org/10.1109/IACC.2016.25

[26] R. Yager. 1988. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man and Cybernetics* 18, 1 (Dec. 1988), 183–190.

[27] Wei Zhang and Jana Kosecka. 2006. A New Inlier Identification Scheme for Robust Estimation Problems. In *Robotics: Science and Systems II, August 16-19, 2006. University of Pennsylvania, Philadelphia, Pennsylvania, USA*. The MIT Press. https://doi.org/10.15607/RSS.2006.II.018