Chapter 1

The Validation Loop

1.1 Introduction

1.1.1 Case of applicability

For illustration of the proposed validation loop, a case of study has been used alongside the theoretical rationale of the loop to generate explanatory figures and tables and to give proof of the real operation of the loop.

The case of study is an application of NN in aircraft stress engineering. For typical aircraft fuselage panel design, the dominant form of stiffened post buckling failure under shear loading is forced crippling[1]. This occurs when the shear buckles in the panel skin force the attached stiffener flanges to deform out-of-plane. There are other failure modes related to buckling, tension, and compression.

The case of study used for illustration purposes consists of a NN model whose task is to predict Reserve Factors (RF) that quantify failure likelihood for regions of the aircraft subject to different loads happening in flight maneuvers. The possible failure modes are Forced Crippling, Column Buckling, In Plane, Net Tension, Pure Compression, and Shear Panel Failure.

Input data consists of 26 features (loads applied to a specific region of the aircraft and different maneuver specs). Output data consists of 6 reserve factors that quantify the stress failure likelihood (RF_1, RF_2, RF_3, RF_4, RF_5, and RF_6), where $RF_i \in [0,5]$,where 0 means extreme risk and 5 means risk extremely low (in a logarithmic scale).

The NN architecture and further features are irrelevant for this study and therefore the model treatment alongside this work remains the concept of a "black box" whose inputs are the 26 features aforementioned and whose output are the 6 reserve factors corresponding to 6 different failure modes.

For typical aircraft fuselage panel design, the dominant form of stiffened post buckling failure under shear loading is forced crippling. This occurs when the shear buckles in the panel skin force the attached stiffener flanges to deform out-of-plane.

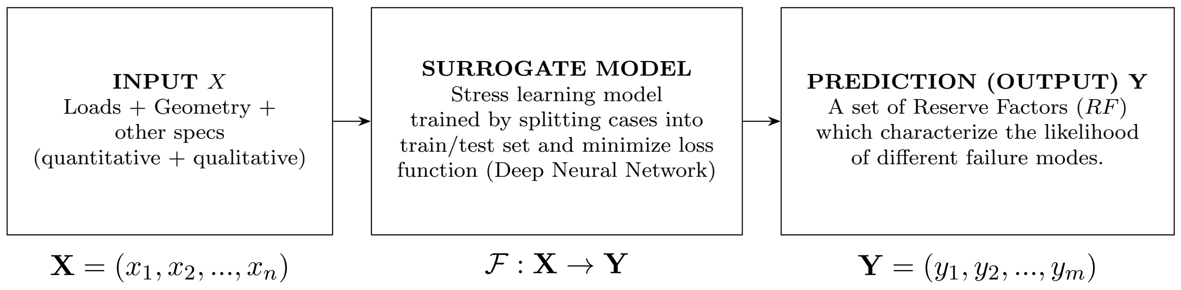
INPUT X
Loads + Geometry + other specs (quantitative + qualitative)
SURROGATE MODEL
Stress learning model trained by splitting cases into train/test set and minimize loss function (Deep Neural Network)
PREDICTION (OUTPUT) Y
A set of Reserve Factors (RF) which characterize the likelihood of different failure modes.
$\mathbf{X} = (x_1, x_2, ..., x_n)$
$\mathcal{F} : \mathbf{X} \to \mathbf{Y}$
$\mathbf{Y} = (y_1, y_2, ..., y_m)$

Figure 1.1: Surrogate model pipeline

1.1.2 Applicability

The applicability region can be intuitively defined as the set of the input and output values for which the model's prediction can be trusted. Applicability region is a refined idea of the *Operational Design Domain* of a model.

For a model F to be applicable to a certain new point x:

- x needs to be inside the *input* applicability region of F.
- The prediction $Y = F(x)$ needs to be inside the *output* applicability region of F.

Knowing whether a new point is either inside or outside applicability is a binary decision problem which can be addressed *a priori* (filtering) or be learned *a posteriori* (model boosting). In other words, the binary classification can be addressed from an unsupervised (geometric) learning approach, or a supervised learning approach. Supervised applicability classifiers are based on exogenous criteria. Examples of them include:

- **Error-filtered convex hull.** Initially the whole set under testing is in applicability range. After filtering those test cases whose prediction error exceeds a tolerance, the applicability region is defined as the convex hull of the remaining subset. The convex hull can be simply defined as the smallest convex set containing the data[2].
- **Error-labelled classifiers.** Initially a whole subset of the dataset (namely, the test set) is in applicability range. After labelling those test cases whose prediction error exceeds a tolerance as outside applicability, different binary classifiers are trained on the full dataset.

On the other hand, geometrical applicability classifiers are based on the premise of NN interpolation capabilities. It has been widely discussed (see *e.g.* [3–5]) that NN's performance relies on their interpolation capabilities, and that no extrapolation capabilities outside their applicability region should be assumed.

Subsequently, from a geometrical approach to the applicability classification problem, we shall define the input applicability region as the geometrical region in the input space where the model is known to work in interpolating regime.

There are plenty of definitions for the interpolating region of a given dataset. Some authors define it as the smallest hypercube enclosing the data[6] although this may seriously challenge interpolation due to isolated points falling into the interpolation region under this definition. Many authors (see v. gr. [7, 8]) define the interpolation region as the *convex hull* of the training data, *i.e.* :

Definition 1. *[9] Standard interpolation occurs for a sample* $\mathbf{x}$ *whenever this sample belongs to the convex hull of a set of samples* $\mathbf{X} \triangleq \{\mathbf{x}_1, ..., \mathbf{x}_N\}$*.*

In the above definition, $\mathbf{X}$ is the training set which has been used for training the model.

Recently, [9] have pointed out that, due to the so-called curse of dimensionality, extrapolation outside the training convex hull always ends up taking place if the input dimension is sufficiently large. The curse of dimensionality[10, pp. 17-18] can be illustrated by the fact that the unitary n-sphere's volume asymptotically diminishes to 0 as n increases. In the end, the effect is that, as the number of dimensions increase, more data is needed in order for the model to work in interpolating regime, as is showed by the following theorem:

Theorem 1.1.1. *[11] Given a d-dimensional dataset* $\mathbf{X} \triangleq \mathbf{x}_1, ..., \mathbf{x}_N$ *with i.i.d. samples uniformly drawn from a hyperball, the probability that a new sample* $\mathbf{x}$ *is in interpolation regime (recall 1) has the following asymptotic behaviour:*

$$\lim_{d\to\infty} p(\mathbf{x} \in Hull(\mathbf{X})) = \begin{cases} 1 \iff N > d^{-1}2^{\frac{d}{2}} \\ 0 \iff N < d^{-1}2^{\frac{d}{2}} \end{cases} \tag{1.1}$$

The main answer to the argument of [9] is that interpolation does not occur in the ambient space, but in a latent, low-dimensional space of the input data[12]. This leads to a new definition of interpolation alternative to 1.

What's more, the authors in [12] not only show that interpolation indeed occurs (at least in a latent representation of the ambient space), but that more important conditions to the model's performance are that testing and training data follow the same cumulative distribution (or the same tail distribution whenever the former cannot be fulfilled) and that testing points be not isolated in the dataset.

Examples of unsupervised applicability classifiers include:

- **Operational design domains.** Based on engineer/scientist expertise, only certain combinations and range of features make physical sense.
- **Input space range classifier.** The applicability region is defined as the boundary of the hypercube whose edges are the ranges $[min, max]$ of each input variable in the training set. In practice, this amounts to checking if each numerical feature X_i of the test point lies inside the range spanned by the training subset.
- **Input space convex hull classifier.** The applicability region is defined as the boundary of the convex hull of the training set.
- **Input space isolated region detector.** Even if a point is inside the convex hull of the training set, it can be very far away from other points, thus interpolation can be challenged. Isolated region detectors address this by checking the statistical vicinity of train points and comparing it to the distance of a given test point to the closest training point.

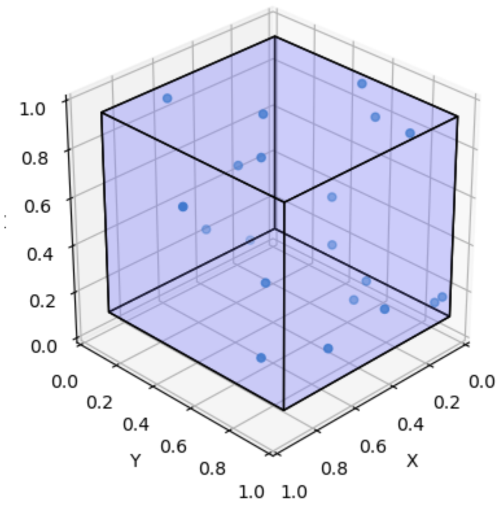

Figure 1.2: Smallest cube enclosing a set of 20 randomly generated points in a 3-dimensional euclidean space.

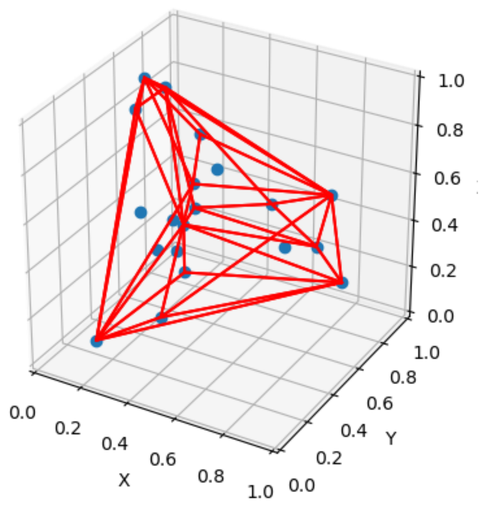

Figure 1.3: Convex hull enclosing the set of Figure 1.2. The convex hull's volume is always smaller than or equal to the volume of the hypercube.

1.1.3 Overview

(Sin acentos) Recorrido sumario por las distintas partes del analisis estadistico (plantillas) centrandose en su funcion en vez de su funcionamiento tecnico. Explicacion de la arquitectura general de la validacion.

1.2 Train-test split

1.2.1 Preliminaries

In supervised learning applications, the dataset is typically divided into the training and the testing sets. Keeping different sets for each task is fundamental in order to prevent model bias. Typical figures for the train-test split ratio are 80%-20%. When the model being trained is very large, a third dataset (the validation set) may be needed for comparing different hyperparameter configurations, in which case the split is typically done at 60%-20%-20% for the train, test, and vaidation sets, resectively[10, pp. 20-21].

Training and evaluating the NN on the same dataset would result in the phenomenon called overfitting[10, pp. 19-20], which basically consists of the NN fitting the noise in the training data and thus losing generalization capabilities.

With the dataset split into the train, evaluation, and test sets, the standard training and validation loop is as follows: the model is trained to "fit" the data in the training set. After a certain amount of training, its performance is measured in the evaluation set. The test set is used to compare the performance of different hyperparameter configurations. Note that the NN is always evaluated with data not previously "seen" during training, and as such cannot develop any bias for the training set (although bias for the validation or test sets may exist).

In the model validation loop, the first question that should be asked, even before training the model, is whether the dataset split is appropriate for training.

For our case of applicability (MSP18, explicar en otra seccion), we suppose we have a dataset which has been split into a training set

$$\mathcal{S}_{\text{train}} = \{(\mathbf{X^{tr}}, \mathbf{Y^{tr}})_i\}_{i=1}^{N_{training}}$$

and a test set

$$\mathcal{S}_{\text{test}} = \{(\mathbf{X^{te}}, \mathbf{Y^{te}})_i\}_{i=1}^{N_{test}},$$

where $\mathbf{X} = (X_1, X_2, \ldots, X_m)$ is the vector of input variables (also called feature vector), m is the dimensionality of the input parameter space, and $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_q)$ is the vector of output variables, with dimensionality q. Individual variables X_k can be numerical or categorical. For instance, for stress problems such as MSP-S18, we have a combination of numerical (*e.g.* external loads) and categorical (eg discrete geometrical variables such as "Frame" or "Stringer", and purely categorical such as "dp"). Individual variables Y_k describe the prediction and these are typically numerical (for stress problems such as MSP-S18, $q = 6$ and these are so-called reserve factors, quantifying the failure likelihood of different failure modes).

The aim of this section is to propose a set of tests to evaluate to which extent splitting fulfils the necessary conditions for the subsequent surrogate model $\mathbf{Y} = F(\mathbf{X})$ to be appropriately trained on $\mathcal{S}_{\text{train}}$ and tested on $\mathcal{S}_{\text{test}}$. This basically means the latter be in the applicability region of a model trained on the former, in order for the training-validation loop of the model to be coherent.

1.2.2 Validation loop for the train-test split

To address the applicability classification problem, points in the test set extremely far away from the training set (effectively outside applicability) need to be detected. If found, that would mean the train-test split is incorrect.

There is a wide range of criteria for classifying a point inside or outside applicability. The proposed validation loop analyses this following a hierarchical flow. For each point in the test set a decision is made based on where that test set point lies with respect to the training set.

Because categorical variables are usually related to different physical conditions, the first step is to condition the analysis of each test point only with respect the training subset of points whose categorical variables have exactly the same values than the test point under analysis. This conditioning on categorical variables defines, for each test point, a *voxel*, *i.e.* a region in input space defined by the precise values of each categorical variable. At this point the first requirement comes into play: the number of training samples inside such voxel needs to be large enough (*e.g.* larger than zero).

Then, inside the voxel the ranges of each feature X_i of the training subset are computed, and then the maximum ranges are checked and defined. Their cartesian product defines the voxel's hypercuboid volume associated to the numerical features. In practice, this amounts to checking if each numerical feature X_i of the test point lies inside the range spanned by the training subset. If all numerical features fall inside these ranges, the test set falls inside the training subset hypercuboid, otherwise, it falls outside. The points lying outside such hypercuboid are outside applicability and are flagged. The "hypercube requirement" is that such percentage of points outside applicability is zero, or very close to it.

If the test point under analysis is found to be outside the hypercuboid, the analysis finishes and the loop moves to the next point. Otherwise, the second step is to check whether this point not only lies inside hypercuboid, but further, whether it also lies inside the convex hull spanned by the training subset in a PCA99[1] projection. This hull has a smaller volume than the hypercuboid. If the test point under analysis is found to be outside the convex hull in the PCA99 projection, the analysis finishes and the loop moves to the next point.

Otherwise, then the third step is to check whether this point (not only lies inside hypercuboid and CH-PCA99, but also) lies inside the convex hull spanned by the training subset in ambient space. The volume of this hull is typically smaller than the one of the PCA99 projection by virtue of Theorem 1.1.1, and if the test point lies inside this hull, interpolating properties of the surrogate model will suggest that the model is not require to generalize outside the region where it cannot generalize. The step of the PCA projection is motivated by the curse of dimensionality, which makes every test point be in extrapolating regime with respect to the training data when the convex hull of the raw, unprocessed training data is used in a high-dimensional input space, as pointed out by [9]. The work of [12] effectively shows how in such cases, a low-dimensional space is more useful for applicability classification.

Each step provides different level of evidence of whether the point under analysis was adequately located, where the best is that it lies inside the CH-ambient, and the worst is that it lies outside the hypercuboid.

An additional check is to analyse whether p-hacking is happening, *i.e.* in this case whether the test set point is "too close" to an actual training point (or indeed equal to a training point), what would falsely induce low test error of the model. For this, we measure the distance of the test point to the closest point of the training subset. This information is latter used to assess different aspects of potential p-hacking and its impact on the decision of train-test split correctness.

[1]A truncated PCA[13] projection in which the selected components explain at least 99% of the whole variance of the data.

Pseudocode:

```
For every point x =  ({x_numerical}, {x_categorical}) in Test_Set:
Compute test_voxel by imposing x_categorical
For every point x =  (x_numerical , x_categorical) in Train_Set:
Compute train_voxel by imposing x_categorical

For every voxel in test_voxel:
Get train_voxel by imposing x_categorical
if size(train_voxel) < voxel_size_req:
annotate and break
else:
Compute main nearest neighbour distance inside train_voxel <d>=d(
    train_voxel , train_voxel) and annotate
for every point x in test_voxel:
Compute minimum nearest neighbour distance d(x, train_voxel) and
    annotate
if x not in train_voxel hypercube:
annotate and break
else:
if x not in CHMP_PCA99:
annotate and break
else:
check if X is in CHMP_ambient and annotate
```

Algorithm 1: TT-split geometrical analysis

```
Data: $\mathcal{S}_{\text{train}}, \mathcal{S}_{\text{test}}$
Result: reqs_results table
1  foreach $\mathbf{X} = (\mathbf{X}_{numerical}, \mathbf{X}_{categorical})$ in $\mathcal{S}_{test}$ do
2      Compute test_voxel by imposing $\mathbf{X}_{\text{categorical}}$;
3  end
4  foreach $\mathbf{X} = (\mathbf{X}_{numerical}, \mathbf{X}_{categorical})$ in $\mathcal{S}_{train}$ do
5      Compute train_voxel by imposing $\mathbf{X}_{\text{categorical}}$;
6  end
7  foreach test_voxel do
8      Get train_voxel by imposing $\mathbf{X}_{\text{categorical}}$;
9      if size(train_voxel) < voxel_size_req then
10         annotate and exit;
11     end
12     else
13         Compute main nearest neighbour distance inside train_voxel $\overline{d}(\text{train\_voxel}, \text{train\_voxel})$
             and annotate;
14     end
15 end
```

Bibliography

[1] P. Bijlaard. “On the Buckling of Stringer Panels Including Forced Crippling”. In: *Journal of the Aeronautical Sciences* 22.7 (1955), pp. 491–501 (cit. on p. 1).

[2] F. P. Preparata and M. I. Shamos. “Convex Hulls: Basic Algorithms”. In: *Computational Geometry: An Introduction*. New York, NY: Springer New York, 1985, pp. 95–149. ISBN: 978-1-4612-1098-6. DOI: 10.1007/978-1-4612-1098-6_3. URL: https://doi.org/10.1007/978-1-4612-1098-6_3 (cit. on p. 2).

[3] D. Barrett et al. “Measuring abstract reasoning in neural networks”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 511–520. URL: https://proceedings.mlr.press/v80/barrett18a.html (cit. on p. 2).

[4] B. M. Lake and M. Baroni. “Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks”. In: *CoRR* abs/1711.00350 (2017). arXiv: 1711.00350. URL: http://arxiv.org/abs/1711.00350 (cit. on p. 2).

[5] D. Saxton et al. “Analysing Mathematical Reasoning Abilities of Neural Models”. In: *CoRR* abs/1904.01557 (2019). arXiv: 1904.01557. URL: http://arxiv.org/abs/1904.01557 (cit. on p. 2).

[6] T. Ebert, J. Belz, and O. Nelles. “Interpolation and extrapolation: Comparison of definitions and survey of algorithms for convex and concave hulls”. In: *2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. IEEE. 2014, pp. 310–314 (cit. on p. 3).

[7] W.-Y. Loh, C.-W. Chen, and W. Zheng. “Extrapolation errors in linear model trees”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.2 (2007), 6–es (cit. on p. 3).

[8] P. Klesk. “Construction of a Neurofuzzy Network Capable of Extrapolating (and Interpolating) With Respect to the Convex Hull of a Set of Input Samples in R”. In: *IEEE Transactions on Fuzzy Systems* 16.5 (2008), pp. 1161–1179. DOI: 10.1109/TFUZZ.2008.924337 (cit. on p. 3).

[9] R. Balestriero, J. Pesenti, and Y. LeCun. “Learning in high dimension always amounts to extrapolation”. In: *arXiv preprint arXiv:2110.09485* (2021) (cit. on pp. 3, 7).

[10] S. Marsland. *Machine Learning: An Algorithmic Perspective*. 2nd ed. Boca Raton, USA: Chapman & Hall/CRC, 2015 (cit. on pp. 3, 5).

[11] I. Bárány and Z. Füredi. “On the shape of the convex hull of random points”. In: *Probability theory and related fields* 77 (1988), pp. 231–240 (cit. on p. 3).

[12] L. Bonnasse-Gahot. “Interpolation, extrapolation, and local generalization in common neural networks”. In: *arXiv preprint arXiv:2207.08648* (2022) (cit. on pp. 3, 7).

[13] H. Hotelling. “Analysis of a complex of statistical variables into principal components.” In: *Journal of educational psychology* 24.6 (1933), p. 417 (cit. on p. 7).

[14] O. Montenbruck and E. Gill. *Satellite Orbits: Models, Methods and Applications.* 1st ed. Wessling, Germany: Springer, 2001 (cit. on p. 10).

[15] K. Yamanaka and F. Ankersen. “New State Transition Matrix for Relative Motion on an Arbitrary Elliptical Orbit”. In: *Journal of Guidance, Control & Dynamics* 25.1 (2002), pp. 60–66. DOI: 10.2514/2.4875 (cit. on p. 11).

[16] D.-W. Gim and K. Alfriend. “State Transition Matrix of Relative Motion for the Perturbed Noncircular Reference Orbit”. In: *Journal of Guidance Control and Dynamics* 26 (Nov. 2003), pp. 956–971. DOI: 10.2514/2.6924 (cit. on p. 11).

[17] S.-S. Exchange. *Practical Uses of an STM.* 2019. URL: https://space.stackexchange.com/questions/32916 (cit. on p. 12).

[18] ESA. *Precise Orbit Determination.* 2011. URL: https://gssc.esa.int/navipedia/index.php/Precise_Orbit_Determination (cit. on p. 12).

[19] C. Chao and F. Hoots. *Applied Orbit Perturbation and Maintenance.* Aerospace Press, 2018. ISBN: 9781523123346 (cit. on p. 17).

[20] W. E. Wiesel. *Modern Astrodynamics.* 2nd ed. Beavercreek, Ohio: Aphelion Press, 2010 (cit. on pp. 20, 21).

[21] M. Eckstein, C. Rajasingh, and P. Blumer. “Colocation Strategy and Collision Avoidance for the Geostationary Satellites at 19 Degrees West”. In: *CNES International Symposium on Space Dynamics.* Vol. 25. Oberpfaffenhofen, Germany: DLR GSOC, Nov. 1989, pp. 60–66 (cit. on pp. 21, 22).

[22] S. D’Amico and O. Montenbruck. “Proximity Operations of Formation-Flying Spacecraft Using an Eccentricity/Inclination Vector Separation”. In: *Journal of Guidance, Control & Dynamics* 29.3 (2006), pp. 554–563. DOI: 10.2514/1.15114 (cit. on pp. 22, 27, 29).