Chapter 1

The Validation Loop

1.1 Introduction

1.1.1 Case of study

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, demonstrating the loop's effectiveness in real operations.

The case of study is an application of neural networks (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 aim of the NN model is to predict Reserve Factors (RF) that quantify failure likelihood for regions of the aircraft subject to different loads happening in flight maneuvers (vid. Figure 1.1). 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).

This study prioritizes assessing the efficacy of the developed validation tool, not optimizing the neural network model itself. Therefore, the specific architecture and additional features of the NN are intentionally treated as a black box. Our focus remains solely on its inputs –the 26 aforementioned features– and its outputs –the 6 reserve factors representing 6 distinct failure modes–. This simplification allows us to isolate and evaluate the performance of the validation tool without introducing confounding variables related to the specific neural network design.

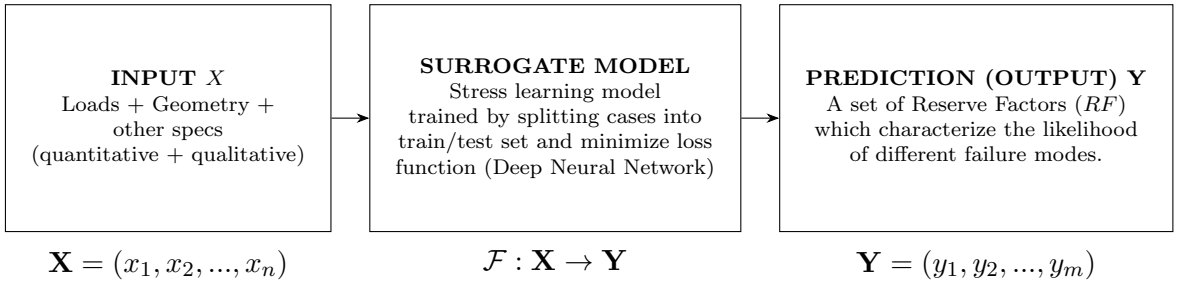

Figure 1.1: Surrogate model pipeline

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:

A. Supervised applicability classifyers

Supervised applicability classifiers are based on exogenous criteria. Examples of them include:

- **Operational design domains.** Based on engineer/scientist expertise, only certain combinations and range of features make physical sense. These combinations are used to define a (geometrical) applicability region.
- **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] (vid. Figure 1.3).

- **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.

B. Unsupervised applicability classifyers

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] (vid. Figure 1.2) 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.

Interestingly, the authors in [12] go beyond demonstrating that interpolation occurs (at least in a latent representation of the ambient space). They unveil two more crucial conditions for optimal model performance:

- The training and testing data should share the same cumulative distribution. In situations where this is impossible, matching the tail distribution becomes essential.
- The testing points should not be isolated within the dataset. This means they should have similar characteristics to other data points and not represent extreme outliers 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:

- **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.

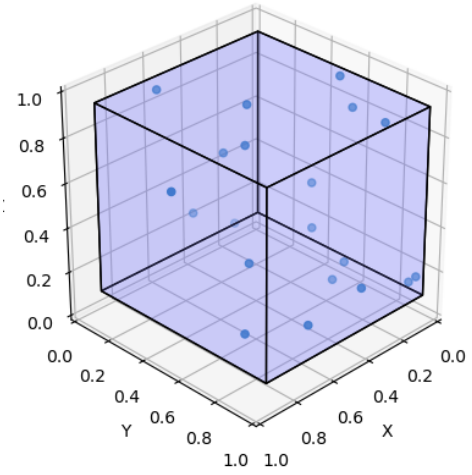

Figure 1.2: Smallest cube enclosing a set of 20 randomly generated points in a 3-dimensional euclidean space.

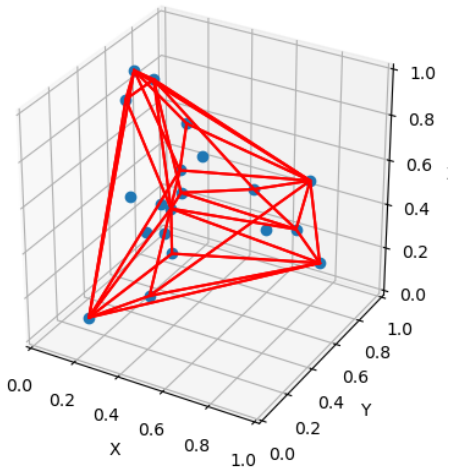

Figure 1.3: Convex hull enclosing the set of Figure 1.2. The convex hull's volume is always smaller than or equal to the volume of the hypercube.

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 (vid Figure 1.4).

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

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

and a test set

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

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

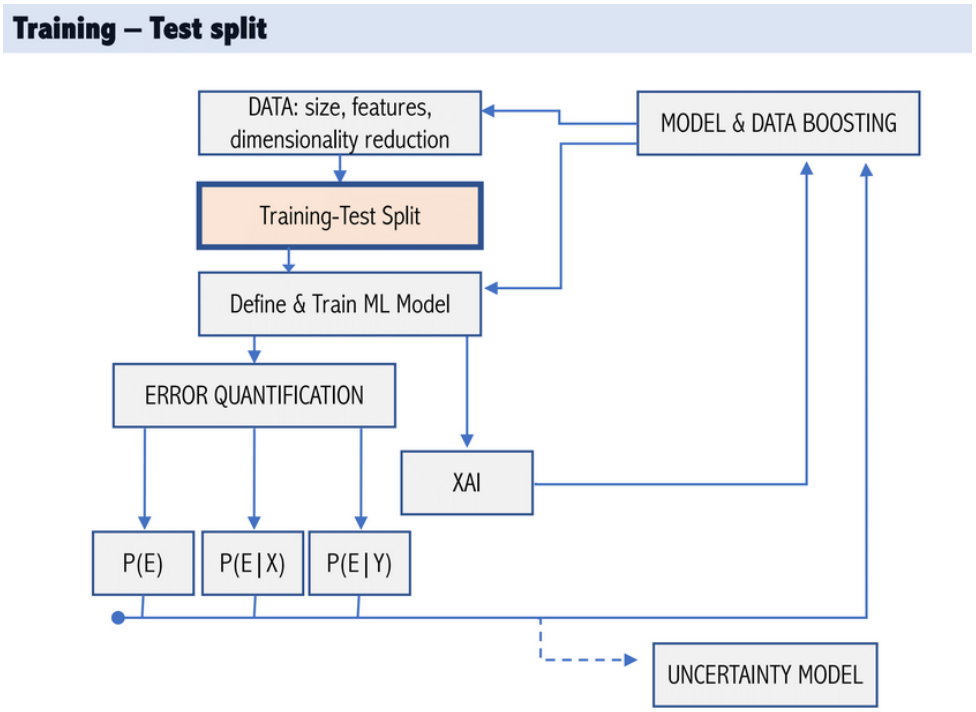

Figure 1.4: Location of the train-test split assessment in the overall validation pipeline

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

1.2.2.1 Geometrical analysis

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 makes such analysis 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 to 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 the 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. It is checked whether the test point is inside or outside the hypercube. 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 the curse of dimensionality, 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 convex hull of the corresponding voxel in ambient space, and the worst is that it lies outside the hypercuboid.

An additional check is to analyse whether p-hacking is happening, *i.e.* 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 later used to assess different aspects of potential p-hacking and its impact on the decision of train-test split correctness.

The former procedure for classifying the applicability of test points is summarised in algorithm 1.

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

Output of algorithm 1 is depicted in Table 1.1. This information has to be processed to complete Table 1.4. The first column identifies the test point inside the test set. The next three columns show information about the corresponding voxel: existance, identification and size. Columns 5 to 7 show information related to pointwise distances inside the voxel. The last three columns show the relative position of the test point inside the voxel (inside the hypercube/convex hull in PCA99/convex hull in ambient space). In this algorithm, if the test point is found to be outside the hypercube (CH_{PCA99}), then its position with respect to CH_{PCA99} and $CH_{ambient}$ ($CH_{ambient}$) is not computed in the sake of computational efficiency.

To better understand how the voxels are computed, a reference of their categorical variables is given in Table 1.2. Points inside each voxel all have the same categorical values, which effectively act as a unique identifyer for the voxel. We see each voxel represents a different stringer section between two concrete frames, plus the categorical variable "np" which can take the values either 0 or 0.9.

Data in the column "inside vox. hypercube" of Table 1.1 has been rearranged for the sake of interpretability in Table 1.3. This table shows the points which lie outside the training voxel hypercube for every voxel where such points exist. Similar tables could be arranged with the number of points belonging to the hypercube but not to the convex hull in a PCA99 projection, or with those points belonging to the convex hull in a PCA99 projection but not in ambient space.

This information is used to check the following parameters and requirements, which are stored in Table 1.4:

- Whether there are enough training samples inside each training subset is compared with `voxel_size_req`.
- The percentage of test points inside the ranges of the training samples is compared with `hypercube_req`.
- The number of points inside the hypercube but outside `CH_PCA99` is compared with `CHMP_PCA99_negative_`
- The number of points inside the hypercube and `CH_PCA99` but outside `CH_ambient` is compared to the requirement `CHMP_ambient_negative_req`.
- The total number of points inside `CH_ambient` is compared to `CHMP_ambient_abs_req`.
- All these requirements are stored in Table 1.4.

Algorithm 1: Train-test split geometrical analysis

Data: $\mathcal{S}_{\text{train}}, \mathcal{S}_{\text{test}}$, voxel_size_req
Result: inside_hypercube, inside_PCA99, inside_ambient, avg_train_dist, min_test_dist

Initialize empty lists: test_voxels, train_voxels $\leftarrow []$;
Initialize boolean arrays:
inside_hypercube, inside_PCA99, inside_ambient $\leftarrow [\text{False}, \ldots, \text{False}]_{1\times \text{len}(\mathcal{S}_{\text{test}})}$;
Initialize arrays for distances: avg_train_dist, min_test_dist $\leftarrow [\text{NaN}, \ldots, \text{NaN}]_{1\times \text{len}(\mathcal{S}_{\text{test}})}$;
foreach $\mathbf{X} = (\mathbf{X}_{numerical}, \mathbf{X}_{categorical})$ *in* $\mathcal{S}_{test}$ **do**
    Compute test_voxel by imposing $\mathbf{X}_{\text{categorical}}$;
    Append test_voxel to test_voxels;
end
foreach $\mathbf{X} = (\mathbf{X}_{numerical}, \mathbf{X}_{categorical})$ *in* $\mathcal{S}_{train}$ **do**
    Compute train_voxel by imposing $\mathbf{X}_{\text{categorical}}$;
    Append train_voxel to train_voxels;
end
$i \leftarrow 0$;
foreach *test_voxel in test_voxels* **do**
    Get train_voxel by imposing $\mathbf{X}_{\text{categorical}}$;
    Compute main nearest neighbour distance inside train_voxel:
    $\overline{d}_{\text{train}}(\text{train\_voxel}, \text{train\_voxel})$;
    avg_train_dist$[i] \leftarrow \overline{d}_{\text{train}}$;
    if *size(train_voxel)* $\geq$ *voxel_size_req* **then**
        foreach $\mathbf{X}$ *in test_voxel* **do**
            Compute minimum nearest neighbour distance: $d_{\text{test}}(\mathbf{X}, \text{train\_voxel})$;
            min_test_dist$[i] \leftarrow d_{\text{test}}$;
            if $\mathbf{X}$ *not in hypercube* **then**
                break;
            else
                inside_hypercube$[i] \leftarrow$ True;
                if $\mathbf{X}$ *not in CH_PCA99* **then**
                    break;
                else
                    inside_PCA99$[i] \leftarrow$ True;
                    if $\mathbf{X}$ *in ambient_hull* **then**
                        inside_ambient$[i] \leftarrow$ True;
                end
            end
            $i \leftarrow i + 1$;
        end
    end
end

Table 1.1: Output of algorithm 1.

	Vox. exists	Vox. ID	Vox. size	Min. test to train vox. dist.	Min. train to train vox. dist.	Avg. train to train vox. dist.	Inside vox. hypercube	Inside vox. CH (PCA99)	Inside vox. CH (ambient)
0	True	729	10	4.2222	2.3103	4.9535	False	NaN	NaN
1	True	703	10	6.8161	3.5068	6.2151	True	False	NaN
2	True	211	10	3.2955	1.6819	4.4763	False	NaN	NaN
3	True	799	8	NaN	NaN	NaN	NaN	NaN	NaN
4	True	531	6	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
1995	True	102	8	NaN	NaN	NaN	NaN	NaN	NaN
1996	True	536	11	3.3157	2.8551	4.4765	True	False	NaN
1997	True	770	6	NaN	NaN	NaN	NaN	NaN	NaN
1998	True	634	10	1.1911	1.8775	5.0745	True	False	NaN
1999	True	277	10	4.8021	3.1150	5.6874	True	False	NaN

2000 rows × 9 columns

Table 1.2: Voxel reference table

	Voxel
Voxel ID	
0	(0.0, Fr68-Fr69, Str40)
1	(0.9, Fr66-Fr67, Str36p)
2	(0.0, Fr69-Fr70, Str29)
3	(0.9, Fr69-Fr70, Str35)
4	(0.0, Fr65-Fr66, Str01)
...	...
827	(0.9, Fr64-Fr65, Str21)
828	(0.9, Fr66-Fr67, Str32)
829	(0.0, Fr64-Fr65, Str26p)
830	(0.0, Fr64-Fr65, Str29p)
831	(0.0, Fr67-Fr68, Str07)

751 rows × 1 columns

Table 1.3: Points outside voxel hypercube

Voxel ID	Voxel	Num. of test points belonging to voxel	Num. of test points outside voxel's hypercube
729	(0.0, 'Fr66-Fr67', 'Str28')	2	2
703	(0.0, 'Fr65-Fr66', 'Str29')	3	2
211	(0.0, 'Fr66-Fr67', 'Str40')	4	3
219	(0.0, 'Fr64-Fr65', 'Str35')	4	3
413	(0.9, 'Fr68-Fr69', 'Str05p')	5	4
...	...	...	...
561	(0.9, 'Fr69-Fr70', 'Str43')	2	1
436	(0.0, 'Fr64-Fr65', 'Str40p')	1	1
276	(0.0, 'Fr64-Fr65', 'Str31')	1	1
109	(0.0, 'Fr69-Fr70', 'Str04')	2	1
446	(0.9, 'Fr69-Fr70', 'Str21')	1	1

104 rows × 3 columns

Table 1.4: `reqs_results_table`. The first and the last requirements compare absolute figures of points at some isolation level with the overall number of points. Whereas the second and third requirements compare the difference in number of points between two consecutive levels with the total number of points.

	Desired rate (equal or less)	Obtained rate
Hypercube rate	0.1000	0.5780
CHMP PCA99 negative rate	0.1000	0.3880
CHMP ambient negative rate	0.1000	0.0340
CHMP ambient abs rate	0.1000	0.3880

Having analised the isolation level of test points, it is necessary to check wheter p-hacking is taking place. That is, whether test points are unrealistically close to train points, thus making the model evaluation flawed. This is measured using the Mann-Whitney U test[14]. In this test, the initial hypothesis H0 is that both distributions are the same. H1 is that the distribution underlying test-training distances is stochastically less than distribution underlying training-training distances, which would involve risk of p-hacking. H0 is rejected only with a 95% confidence. For instance, for the dataset of MSP-18, the resulting p-value is $p = 0.9793$, thus the null hypothesis is not rejected and the dataset can be assumed to be free of p-hacking.

1.2.2.2 Non-geometrical analysis

To help (the engineer in charge) make a decission about the dataset split, another approach is developed, complementary to the geometrical analysis carried out so far. This approach consists of focusing on the statistical distributions of the train and the test datasets. A proper train-test split is characterised by similar statistical distributions of both sets (test and training)[12]. When this cannot be achieved, a softer requirement is that each of the feature variables X_i need to have reasonably similar marginal distributions in the training and the test set.

The following analysis checks whether, for each individual input feature, the distribution of the training and test set is reasonably similar. This is also checked for the output variables (the ground true ones, thus this is independent from the surrogate model). Finally, the splitting can be done locally, *i.e.* region by region, by binning each numerical variable. The implementation of this analysis tackles each input variable X_i independently and for each input variable, it compares the training and the test set such that:

- If the variable X_i is 'categorical': both (train and test) categorical frequency distributions are plotted (Figure 1.6), a 2-sample χ^2 test[30, p. 431] is performed, and the p-value of such test is introduced in Table 1.5.
- If the variable X_i is numerical: both (train and test) numerical frequency distributions are plotted (Figure 1.5), a 2-sample Kolmogorov-Smirnov test[30, p. 454] is carried out, and the p-value of such test is introduced in the same table.

The null hypothesis for both tests is that both train and test distributions are the same. H0 is only rejected with a 95% confidence.

As stated earlier, applicability is not only a matter of the *input* space, but also of the

Table 1.5: P-values of the input variables. The test performed is a 2-sample Kolmogorov-Smirnov or a 2-sample χ^2 test, depending on the data being numerical (first 19 rows) or categorical ("dp", "Frame" and "Stringer").

p-values	
	p-value
factors	0.97409
FU.0410.14	0.08896
FU.0410.15	0.77249
FU.0410.16	0.53206
FU.0410.24	0.31840
FU.0410.25	0.54828
FU.0410.26	0.58953
FU.0420.14	0.19180
FU.0420.15	0.50018
FU.0420.16	0.58953
FU.0420.24	0.54015
FU.0420.25	0.34948
FU.0420.26	0.74838
FU.0430.14	0.16084
FU.0430.15	0.17195
FU.0430.16	0.51601
FU.0430.24	0.55646
FU.0430.25	0.42458
FU.0430.26	0.66518
dp	0.99969
Frame	1.00000
Stringer	1.00000

output space. The 2-sample Kolmogorov-Smirnov test has been performed again on each of the six output variables. The p-values and their statistical distributions are depicted in Table 1.6 and Figure 1.7, respectively.

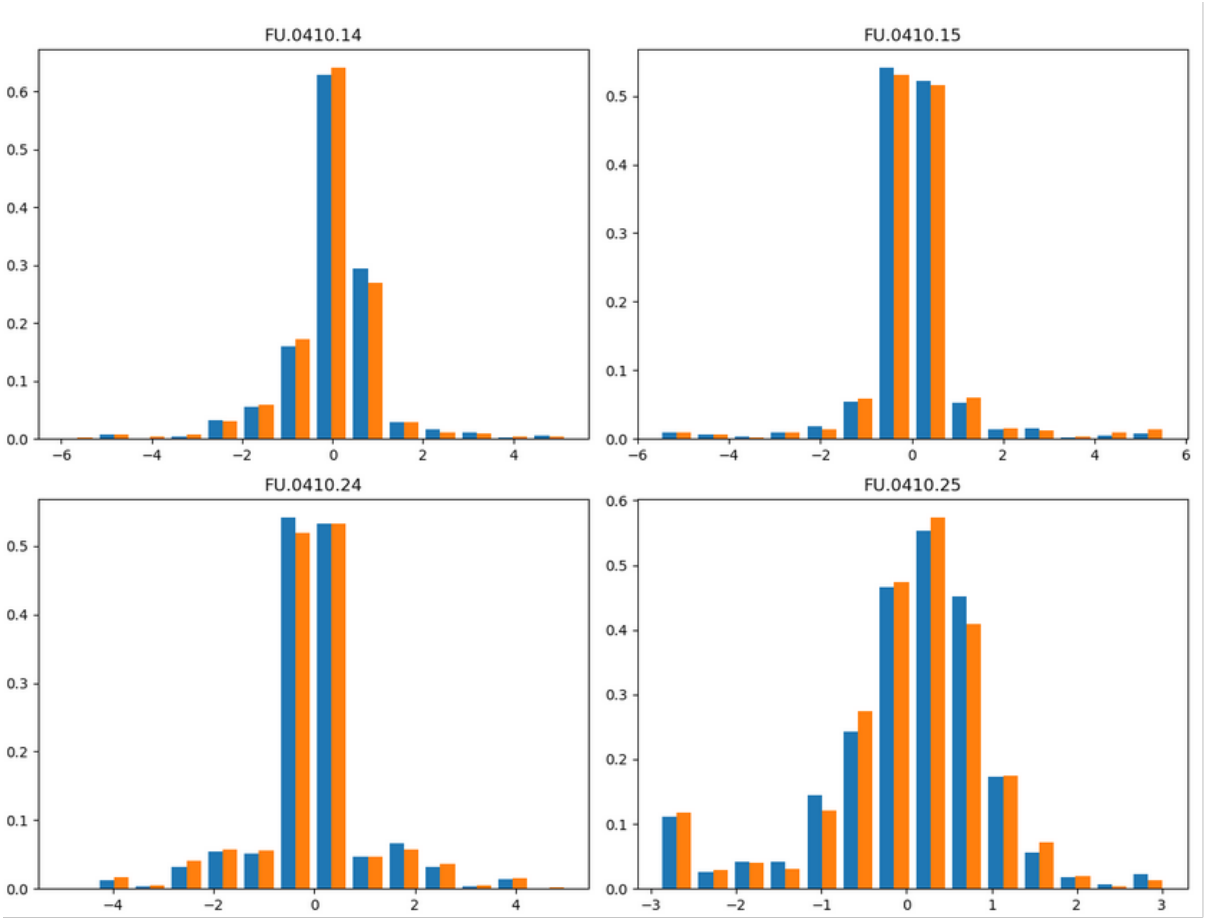

Figure 1.5: Input [numeric] variables distributions double histogram. Only the first four input variables have been plotted.

Table 1.6: P-values of the output variables distributions

p-values	KS
RF Forced Crippling	1.00000
RF Column Buckling	1.00000
RF In Plane	1.00000
RF Net Tension	0.24668
RF Pure Compression	1.00000
RF Shear Panel Failure	0.99995

1.2.2.3 Train-test split. Conclusions

The proposed validation loop for the train-test split focuses, on the one hand, on the applicability classification problem, which makes use of a geometrical analysis which classifies

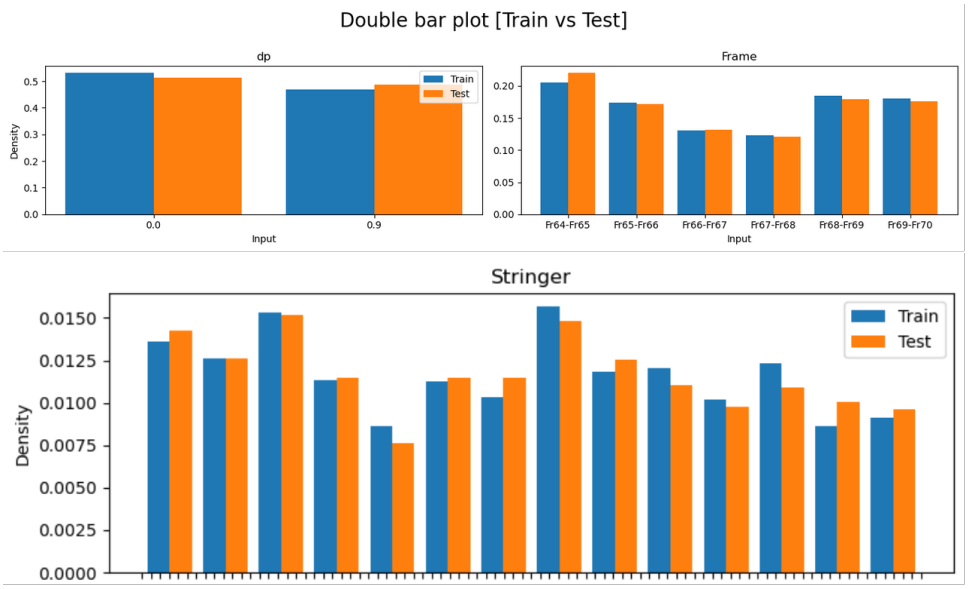

Figure 1.6: Input categorical variables distributions double histograms

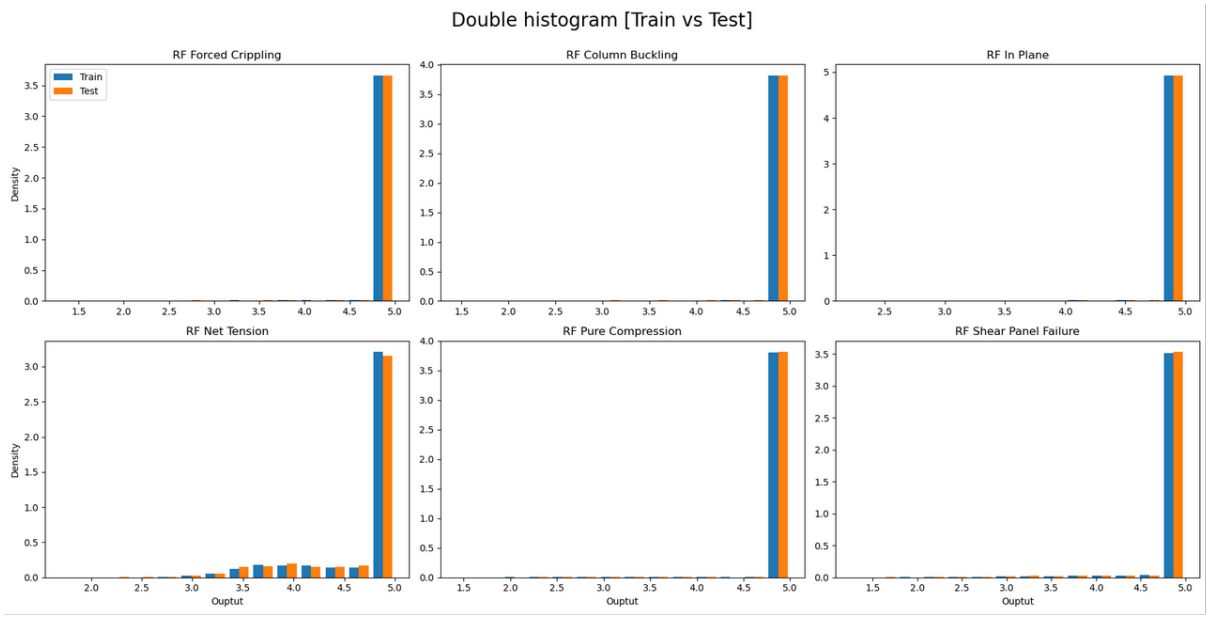

Figure 1.7: Output variables distributions double histograms

test points at different levels of isolation. A series of requirements are defined based on the proportions of such isolated points. Plus, p-hacking is checked. The output of this analysis is showed in Figure 1.8.

On the other hand, the distributions of both input and output variables in the training set are checked to be reasonably similar to that of the test set. The main outputs are the p-values shown at Table 1.5 and Table 1.6.

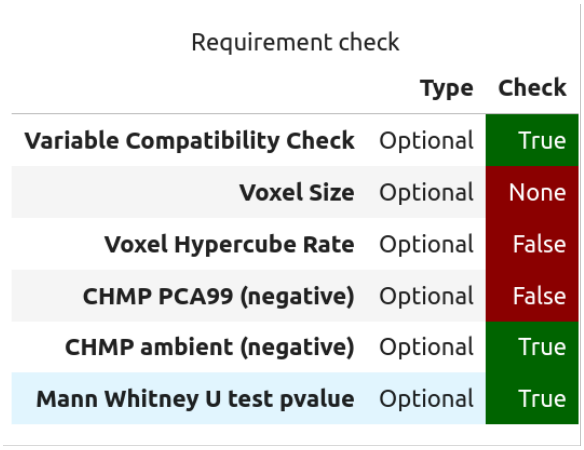

Figure 1.8: Requirements of the geometrical analysis

Of course, any given dataset may meet some of the requirements, but not all (as is the case with the MSP-18 dataset). The engineer in charge should evaluate the ensemble output and decide whether it is necessary to redo the split completely or partially, depending on data availability.