

Best Practices for Machine Learning Experimentation in Scientific Applications

Umberto Michelucci¹ and Francesca Venturini²

¹ Computer Science Department, Lucerne University of Applied Sciences and Arts,
Luzern 6002, Switzerland

`umberto.michelucci@hslu.ch`

² Institute of Applied Mathematics and Physics,
ZHAW - Zurich University of Applied Sciences,
Winterthur 8400, Zurich, Switzerland

`vent@zhaw.ch`

Abstract. Machine learning (ML) is increasingly adopted in scientific research, yet the quality and reliability of results often depend on how experiments are designed and documented. Poor baselines, inconsistent preprocessing, or insufficient validation can lead to misleading conclusions about model performance. This paper presents a practical and structured guide for conducting ML experiments in scientific applications, focussing on reproducibility, fair comparison, and transparent reporting. We outline a step-by-step workflow, from dataset preparation to model selection and evaluation, and propose metrics that account for overfitting and instability across validation folds, including the Logarithmic Overfitting Ratio (LOR) and the Composite Overfitting Score (COS). Through recommended practices and example reporting formats, this work aims to support researchers in establishing robust baselines and drawing valid evidence-based insights from ML models applied to scientific problems.

1 Introduction

When starting a new machine learning (ML) project, one of the most critical steps is to create a solid and well-documented **baseline**. A baseline provides a reference point against which more complex models such as deep neural networks can be objectively compared. Establishing it in the right way ensures that any subsequent improvement truly reflects the model design and not differences in data processing or evaluation.

This document presents a step-by-step procedure for building a sensible baseline, from dataset preparation to model evaluation and reporting. Each step emphasises reproducibility, transparency, and fair comparison between approaches. Table 1 at the end of this section summarises the main recommendations and good practices. In Table 4 you find an overview of the most commonly used non-deep learning algorithms to try. An example of a possible *output* table can be found in Table 3.

The steps defined in the following sections should be followed when designing machine learning experiments.

1. INTRODUCTION

Step	Main Goals and Practical Tips
Data Preparation	Clean, normalize, and save each dataset version. Document all transformations and check for outliers visually.
Classical Baseline	Test simple models (Linear/Logistic Regression, Trees, Random Forest, etc.). Keep hyperparameters simple and record baseline metrics.
Cross-Validation	Use k -Fold or Monte Carlo CV. Record mean and std of metrics, and check for overfitting patterns.
Deep Learning Models	Introduce simple neural nets. Use regularization and early stopping. Ensure same preprocessing as classical models.
Model Selection	Exclude non-learning or overfitting models. Keep reproducible, stable, interpretable ones.
Reporting Results	Summarize in a table (metrics \pm SD, both for training and test). Include preprocessing details and highlight best performing models.

Table 1. Summary of the main steps and recommendations for building a solid ML baseline.

1.1 Design of Experiments

An ML experiment is a controlled study in which you train and evaluate one or more ML modeltypes under specified conditions to answer a concrete question.

Definition: ML Experiment

An ML experiment is a controlled study in which you train and evaluate one or more ML model types under specified conditions to answer a concrete question.

For example, imagine that you are trying to classify some input (for example Raman spectra) to predict an output (for example, a concentration of some chemical components), then an experiment is made of the following main components:

1. dataset with preprocessing steps (more on that later);
2. machine learning model type (for example, logistic regression or linear regression);
3. machine learning model instance (for example, support vector classifier with a regularisation parameter $C = 10$), in other words, the model with a chosen set of parameters (α in ridge regression, regularisation parameter; C in SVC, etc.);
4. metrics on the training and validation datasets;

5. final summary of results (summarised in a table).

Definition: Model Type vs. Model Instance

A **model type** refers to the generic algorithmic family used for learning from data (e.g. Logistic Regression, Support Vector Machine, Random Forest). It defines the overall structure and assumptions of the learning process.

A **model instance** is a concrete realisation of a model type, obtained by choosing a specific set of hyperparameters (e.g. Logistic Regression with ℓ_2 regularisation strength $C = 10$, or SVM with RBF kernel and $\gamma = 0.1$). It is the actual model that is trained and evaluated in an experiment.

IMPORTANT: An experiment is defined by **one single set** of all parameters/preprocessing steps.

To design one or multiple experiments, it is useful to fill out the following matrix (which will be useful later on for documentation). In Table 2 you see some examples.

Exp. ID	Task	Preproc.	Normal.	Instance	Metrics	Dataset	Notes
EX1	Classification	Raw	max = 1	Decision Tree	Accuracy, v1 F1		First quick baseline
EX2	Classification	Baseline removed	None	Random Forest	Accuracy, v2 F1		More complex model

Table 2. Example of the table that you can fill to plan the experiments you want to do.

Once you have performed all your experiments, your results should be put in a similar table. You can see how this table might look like in Table 3 with the MAE as a metric. If you have more metrics you will need more columns.

1.2 Dataset Preparation

While designing experiments you have to decide how to prepare your data. Put the information in the two columns in Table 2 in the columns *Preprocessing* and *Normalisation*.

1. INTRODUCTION

Exp. ID	Model	Preproc.	Normal.	MAE $\pm\sigma$ (train)	MAE $\pm\sigma$ (test)	LOR	COS
EX1	Decision Tree	Raw	max = 1	3.4 \pm 0.2	3.0 \pm 0.3	0.054	0.9
EX2	Random Forest	Baseline Removed	max = 1	3.5 \pm 0.1	2.6 \pm 0.4	0.13	0.80

Table 3. Example of the table to show the different experiments. The metric reported is the Mean Absolute Error (MAE) but could be something else for you, for example the MSE, or the accuracy, or something completely different.

Tips: Dataset preparation

- (i) Always keep the raw data intact; create derived, preprocessed versions.
- (ii) Document every transformation (normalisation, outlier removal, encoding, etc.).
- (iii) Save datasets at each stage as separate pickle files: `data_raw.pkl`, `data_normalized_v1.pkl`, `data_cleaned_v1.pkl`, etc.

1.3 Choosing classical models for the experiments

In this phase you decide which model types and their instances you want to use, train them and record the results. Put this information in Table 2 in the columns *Model Instance* for which model you want to test and in the column *metrics* for which metric you have chosen.

Tips: Choosing the right model type

- (i) Start simple: linear or logistic regression often give surprising insight into data (see Table 4 for a list of the most commonly used algorithms).
- (ii) Use simple models to check data leakage and confirm label consistency.
- (iii) Keep hyperparameters minimal at this stage; focus on understanding baseline performance.
- (iv) Record training time and model simplicity (helps later justify complexity of advanced models).
- (v) **Always record training and test metrics** to check for overfitting.

1.4 Cross-Validation (CV) and Evaluation

Always use a proper validation model. You should use: Leave One Out if you have very small datasets (say 20-30 elements). k -Fold or Monte-Carlo CV if you have slightly larger datasets. The standard deviation in the metric columns in Table 3 is the standard deviation over the different folds (in k -fold CV) or over the multiple splits (in Monte Carlo CV).

Algorithm Name	Short Description	Python (Scikit-learn)
Regression Algorithms		
Linear Regression	Models the relationship between a scalar dependent variable and explanatory variables using a linear equation.	<code>from sklearn.linear_model import LinearRegression</code>
Random Forest Regressor	An ensemble method that constructs multiple decision trees and outputs the average prediction of the individual trees.	<code>from sklearn.ensemble import RandomForestRegressor</code>
AdaBoost Regressor	An ensemble boosting method that combines multiple weak regressors (e.g., decision trees) to create a strong predictor.	<code>from sklearn.ensemble import AdaBoostRegressor</code>
Support Vector Machine (SVR)	Uses kernel functions to find a hyperplane that fits the data while keeping errors within a maximum margin (ϵ).	<code>from sklearn.svm import SVR</code>
Classification Algorithms		
Logistic Regression	Uses the logistic function to model a binary dependent variable; estimates the probability of an event occurring.	<code>from sklearn.linear_model import LogisticRegression</code>
Support Vector Machine (SVC)	Finds an optimal hyperplane in an N -dimensional space that distinctly classifies data points and maximizes the margin.	<code>from sklearn.svm import SVC</code>
K-Nearest Neighbors (KNN)	A non-parametric algorithm that classifies a new data point based on the majority class of its k nearest neighbors.	<code>from sklearn.neighbors import KNeighborsClassifier</code>
Random Forest	An ensemble method that constructs multiple decision trees and outputs the average prediction of the individual trees.	<code>from sklearn.ensemble import RandomForestClassifier</code>

Table 4. Often Used Machine Learning Algorithms for Regression and Classification**Tips: Cross validation**

- (i) Prefer Monte Carlo CV; k -Fold CV is good for larger, stable datasets.
- (ii) Compute and log both mean and standard deviation of your metrics.

1. INTRODUCTION

- (iii) Track signs of overfitting: large gaps between train/test performance indicate data leakage or model variance. Always calculate and report LOR (see below) and COS (see below).
- (iv) Store evaluation results in structured form (CSV, DataFrame, etc.) for reproducibility.
- (v) **Always** plot loss function vs. epochs to check learning (for deep learning).
- (vi) **For Classification:** always use confusion matrix and not only accuracy.
- (vii) **For Regression:** always plot prediction vs. true values and calculate R^2 values not only MAE.

CV is done to check generalisation properties of models, meaning how they behave on new data. In more simple terms, it is done to check if models overfit (for a more nuanced discussion about overfitting you can check Chapter 7 in [1]). To do that, two metrics (or scores) can and should be used.

Logarithmic Overfitting Ratio A note on notation is in order. When we write MAE, we intend the average of the single MAEs on each fold (if you are doing k -fold CV) or the single MAEs from each split in Monte-Carlo CV³.

The Logarithmic Overfitting Ratio (LOR) is defined by (we will use the MAE as metric as an example, but it can also be defined with the MSE for example) the following.

$$\text{LOR} = \log \frac{\text{MAE}_{\text{train}}}{\text{MAE}_{\text{test}}} \quad (1)$$

Its value can be interpreted as follows.

- **LOR = 0:** Ideal model. No overfitting, as the MAE on the train and on the test is the same.
- **LOR < 0:** Overfitting is present. The higher the value of LOR (in absolute value), the larger the overfitting.
- **LOR > 0:** There is underfitting. The higher the value of LOR (in absolute value), the larger the underfitting.

Definition: Logarithmic Overfitting Ratio (LOR)

The Logarithmic Overfitting Ratio (LOR) is defined by (we will use the MAE as metric as an example, but it can be also defined with the MSE for example) the following.

$$\text{LOR} = \log \frac{\text{MAE}_{\text{train}}}{\text{MAE}_{\text{test}}} \quad (2)$$

You should choose the model with the LOR value closest to zero. The LOR is a nice metric, but it does not take into account the spread of the values over

³ Note that since each fold or split has the same number of elements the average of the fold averages is equal to the total average of the absolute errors.

folds. Overfitting rears its ugly head in subtle ways. One is that although the metric average could be very similar on the training and test datasets, their standard deviation (σ_{test} and σ_{train}) around the averages may be quite different. Typically, in overfitting regimes, we observe $\sigma_{\text{test}} \gg \sigma_{\text{train}}$. It is important to choose a model instance that not only shows similar metric averages but also a similar standard deviation. To facilitate the choice, you should use, in addition to the LOR, the **Composite Overfitting Ratio** (COS) defined by

$$\text{COS} = \alpha \frac{\text{MAE}_{\text{train}}}{\text{MAE}_{\text{test}}} + \beta \frac{\sigma_{\text{train}}}{\sigma_{\text{test}}} \quad (3)$$

The value of α and β can, in principle, be freely chosen, but I suggest you use $\alpha = \beta = 1/2$. The COS values can be interpreted this way.

- **COS = 1**: Optimal model. Train and test errors match, and the variance across folds (or splits in MC CV) is stable. Thus, the model shows good generalisation properties.
- **COS > 1**: Overfitting is present and there is some instability across folds (or splits in MC CV). The higher the value, the worse the situation.
- **COS < 1**: There is underfitting. The higher the value, the worse the situation.

Definition: Composite Overfitting Ratio (COS)

the Composite Overfitting Ratio (COS) is defined by

$$\text{COS} = \alpha \frac{\text{MAE}_{\text{train}}}{\text{MAE}_{\text{test}}} + \beta \frac{\sigma_{\text{train}}}{\sigma_{\text{test}}} \quad (4)$$

with $\alpha, \beta > 0$.

In Figure 1 you can see how such a table might look like for a real project. Note that this table reports only metrics and no other information, since in the project we generated one table for each experiment. But it will give you an idea on how to report results systematically. The red lines are those for which $R^2 < 0$ (thus unusable), yellow those for which $R^2 < 0.85$ (therefore bad) and green ones for which $R^2 > 0.85$ (possibly useful model instances). The model instances that have $R^2 > 0.85$ are the ones that should be considered. In simple bold face, we have highlighted the model instance with the LOR closest to zero, while in bold face and italics the one with the COS closer to 1.

1.5 Extending to Deep Learning

If you are testing deep learning, you should optimise the network’s architecture and then only report the one that gives you the best results. Unless your focus is on discussing hyperparameters.

1. INTRODUCTION

	MAE_test_mean	MAE_train_mean	MAE_test_std	MAE_train_std	R2_mean_train	R2_mean_test	Overfitting Log Ratio	Composite Overfitting Score
LinearRegression	0.25	0.00	0.1	0.00	1	0.96	-inf	0.00
Ridge($\alpha=0.18$)	0.25	0.2	0.11	0.02	0.98	0.97	-0.223144	0.490909
Ridge($\alpha=0.5$)	0.3	0.25	0.13	0.01	0.97	0.96	-0.182322	0.455128
Ridge($\alpha=0.75$)	0.35	0.29	0.14	0.01	0.97	0.95	-0.188052	0.45
Ridge($\alpha=1.0$)	0.39	0.33	0.15	0.01	0.96	0.93	-0.167054	0.45641
PLSRegression(n_component=5)	0.31	0.09	0.14	0.02	1	0.94	-1.236763	0.21659
PLSRegression(n_component=10)	0.28	0.00	0.13	0.00	1	0.95	-inf	0.00
PLSRegression(n_component=20)	0.28	0.00	0.13	0.00	1	0.95	-inf	0.00
Lasso	1.87	1.77	0.32	0.16	0.00	-0.45	-0.054959	0.723262
ElasticNet	1.87	1.77	0.32	0.16	0.00	-0.45	-0.054959	0.723262
SVR(C=1.0)	1.11	0.97	0.47	0.05	0.57	0.38	-0.134819	0.490128
SVR(C=10.0)	0.26	0.17	0.1	0.02	0.98	0.9	-0.424883	0.426923
SVR(C=50.0)	0.21	0.12	0.1	0.02	0.99	0.91	-0.559616	0.385714
SVR(C=100.0)	0.2	0.11	0.1	0.02	0.99	0.91	-0.597837	0.375
RandomForestRegressor	0.42	0.17	0.1	0.01	0.98	0.82	-0.904456	0.252381
GradientBoostingRegressor	0.47	0.00	0.23	0.00	1	0.88	-inf	0.00
ICA (8 comp.)	0.22	0.15	0.12	0.03	0.99	0.97	-0.382992	0.465909
ICA (5 comp.)	0.26	0.2	0.1	0.04	0.98	0.96	-0.262364	0.584615

Fig. 1. An example on how a table summarising results of experiments might look like. Note that this table reports only metrics and no other information, since in the project we generated one table for each experiment. But it should give you an idea about how it could look like. The red lines are those for which $R^2 < 0$, yellow those for which $R^2 < 0.85$ and green ones for which $R^2 > 0.85$. Model instances that have $R^2 > 0.85$ are the one that should be considered. In simple bold face we have highlighted the model instance for the LOR closest to zero, while in bold face and italics the one with the COS closest to 1.

Tips: Deep Learning

- Start with minimal architectures (1–2 hidden layers for FFNN, shallow CNN for images).
- Use early stopping, dropout, and batch normalization to control overfitting.
- Maintain consistent data splits and preprocessing between ML and DL models.
- Compare neural network results fairly against classical baselines.

1.6 Selecting Meaningful Results

You should always report all your findings, but in deep learning sometime you find yourself in a situation where a model does not learn anything (something you can easily check by looking at the plot of the loss function vs. epochs). In this case the model instance result should not be reported. This manifest itself often in a model instance predicting blindly always one of the classes (for example in a classification task). You can also see that by checking the confusion matrices in case you are working on a classification problem. In a regression problem, this manifest in a model instance that predict always almost the same value.

Tips: Selecting meaningful results

- (i) Exclude models that clearly overfit or fail to learn (e.g., constant predictions).
- (ii) Include training/test metrics and standard deviations in all reports.

2 Result Table and Reporting

Tips: Results

- (i) Use a uniform metric across models (e.g., MAE or accuracy) for fair comparison.
- (ii) Highlight the best model(s) in bold or with visual cues.
- (iii) Include preprocessing details explicitly: they often explain differences in performance.
- (iv) Keep the table concise, but add references to detailed results in an appendix if needed.

3 Acknowledgments

We would like to thank Michael Wuss for the helpful review and feedback.

References

1. Umberto Michelucci. *Fundamental Mathematical Concepts for Machine Learning in Science*. Springer.