# PROJECT 1 MEMORY

# Cross validation

In machine learning, evaluating the model is one of the most important parts of the design process. To ensure a model works as expected, robust testing is crucial. Cross-validation is a powerful technique that helps evaluate the model's performance more reliably.

While a simple 'train-test split' involves setting aside one part of the data for testing, cross-validation takes this further.

Instead of a single split, the data is partitioned into several subsets. The model is then trained and validated multiple times, with a different subsets being held out for testing on each iteration. The results are then averaged.

This method is commonly used to evaluate different models and select the one which outperforms the others on a specific problem, as it provides a more stable and accurate estimate of how the model will perform on unseen data.

There are different types of cross-validation in Machine learning, in this document we will focus on leaving one out cross validation.

## Leave one out cross validation

This validation technique has a unique characteristic: only one single data point from the entire dataset is used for validation, and all the remaining data is used to train the model.

This process is then iterated for every data point in the dataset. In each iteration, a different data point is selected as the single test sample, and a complete model is trained on all other data points.

This means that for a dataset with n data points, you must train n different models. The model's final performance is the average of the results from all n iterations.

# ¿Why can cross-validation be problematic in KNNs?

When developing a machine learning model, it is standard practice to utilize the maximum available dataset. This approach is generally expected to yield superior predictive performance.

However, this reliance on extensive data can lead to significant computational burdens, particularly when employing rigorous validation techniques such as Leave-One-Out Cross-Validation (LOOCV).

## **The Cost of Leave-One-Out Cross-Validation**

LOOCV presents a major computational challenge due to its inherent methodology. Specifically, the technique requires the model to be trained and evaluated N times, where N is the total number of data points. For each iteration, a single data point is reserved for testing, and the remaining N-1 points are used for training.

Consequently, **when dealing with a large volume of data points, the repeated model training and evaluation process translates directly into considerable issues regarding both processing time and computational costs.** This extensive requirement makes LOOCV impractical for models trained on massive datasets.

# Comparing the results of standard and LOOCV and Fast LOOCV

Consistent with the experimentation documented within our Jupyter Notebook environment, a rigorous evaluation was performed contrasting two fundamental strategies for calculating Leave-One-Out Cross-Validation (LOOCV): the conventional computational method versus the optimized approach introduced by Motonobu Kanagawa in his technical publication, *"Fast Computation of Leave-One-Out Cross-Validation for k-NN Regression."*

## Empirical Evidence of Optimization

The resulting data, which is clearly illustrated in the accompanying visualizations, conclusively demonstrates that **Kanagawa's proposed algorithm** offers substantially superior performance.

[Insert Comparative Performance Graphs Here]

## Foundation in Algorithmic Efficiency

The pronounced improvement observed is not merely empirical; it is fundamentally rooted in a **deep restructuring of the computational complexity**.

The standard algorithm for LOOCV in k-NN regression operates with a complexity of order **O()** where n represents the number of instances in the dataset. This quadratic cost rapidly becomes prohibitive as the dataset size scales.

In stark contrast, the core innovation presented by Kanagawa involves reformulating the error metric calculation, enabling his algorithm to execute with **linear complexity, O(n)**. This transition from quadratic to linear dependency means processing time increases only proportionally to the data size, rather than with its square, which fully accounts for the clear advantage demonstrated in the reported execution time metrics.