School of Electronic Engineering and Computer Science
Queen Mary University of London

CBU5201 Principles of Machine Learning Methodology I

Dr Chao Liu

Sep 2023

Credit to Dr Jesús Requena Carrión

Ventris' decisive check

"... and a decisive check, preferably with the aid of **virgin material**, to ensure that the apparent results are not due to **fantasy**, **coincidence** or **circular reasoning**"

Don't fool yourself!

What's different about machine learning?

Machine learning aims at building solutions that work well on **samples** coming from a **target population**.

Many other areas have similar goals, so what's different?

- In machine learning, we **lack a description** of the target population.
- All we can do is extract samples from the population (known as **sampling the population**).

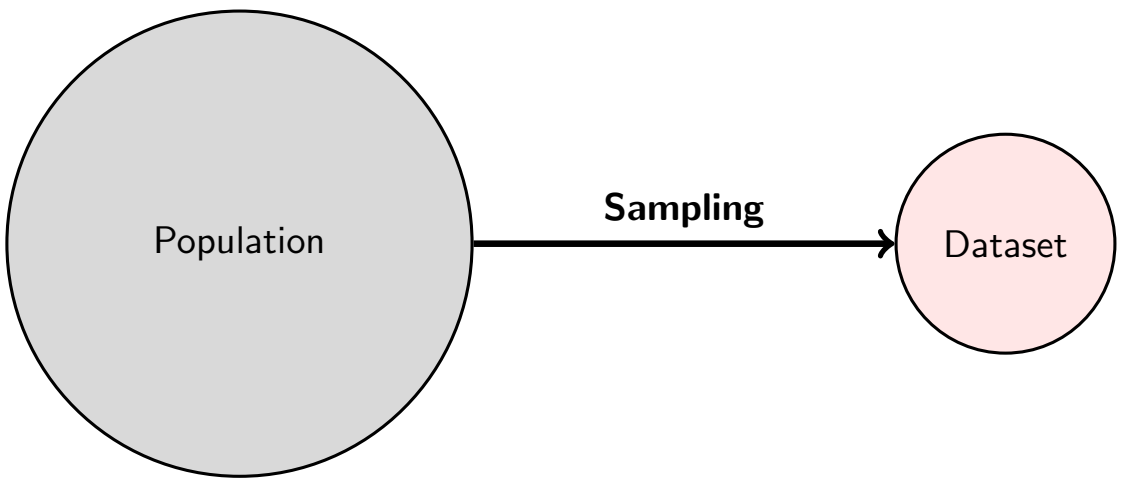
A collection of samples extracted from a population forms a dataset. A dataset provides an **empirical** description of our target population.

Sampling a population

If we use datasets as population surrogates, we need to ensure that:

- Datasets are **representative**, i.e. provide a complete picture of the target population.
- Sampling mimics the mechanism that generates samples during deployment: Samples need to be extracted **independently**.

Samples need to be **independent and identically distributed** (iid).



What can machine learning do with data?

Agenda

Testing a model to evaluate its deployment performance

Optimisation and the error surface

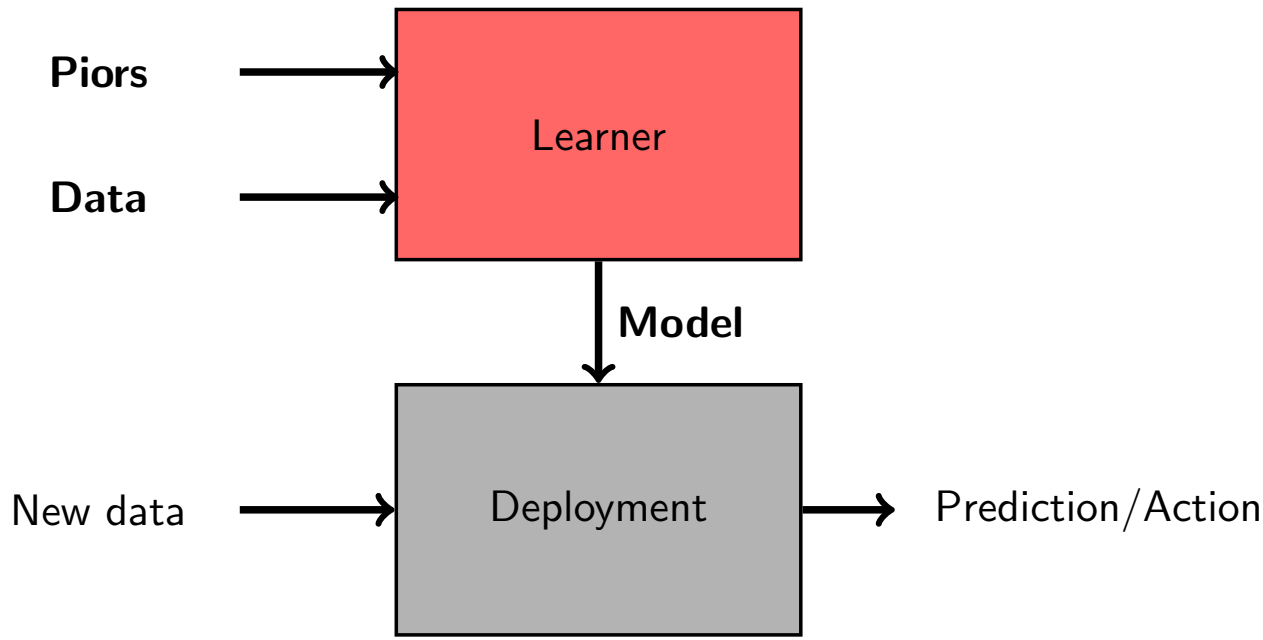
Training a model

Validating our models

Summary

Deployment performance

Machine learning models can be built, sold and deployed.



The best model is the one with the highest **deployment performance**, i.e. the one that works best on the **target population**.

Deployment performance

Every machine learning project needs to include a strategy to **evaluate the performance** of a model during deployment.

In machine learning, a performance evaluation strategy includes:

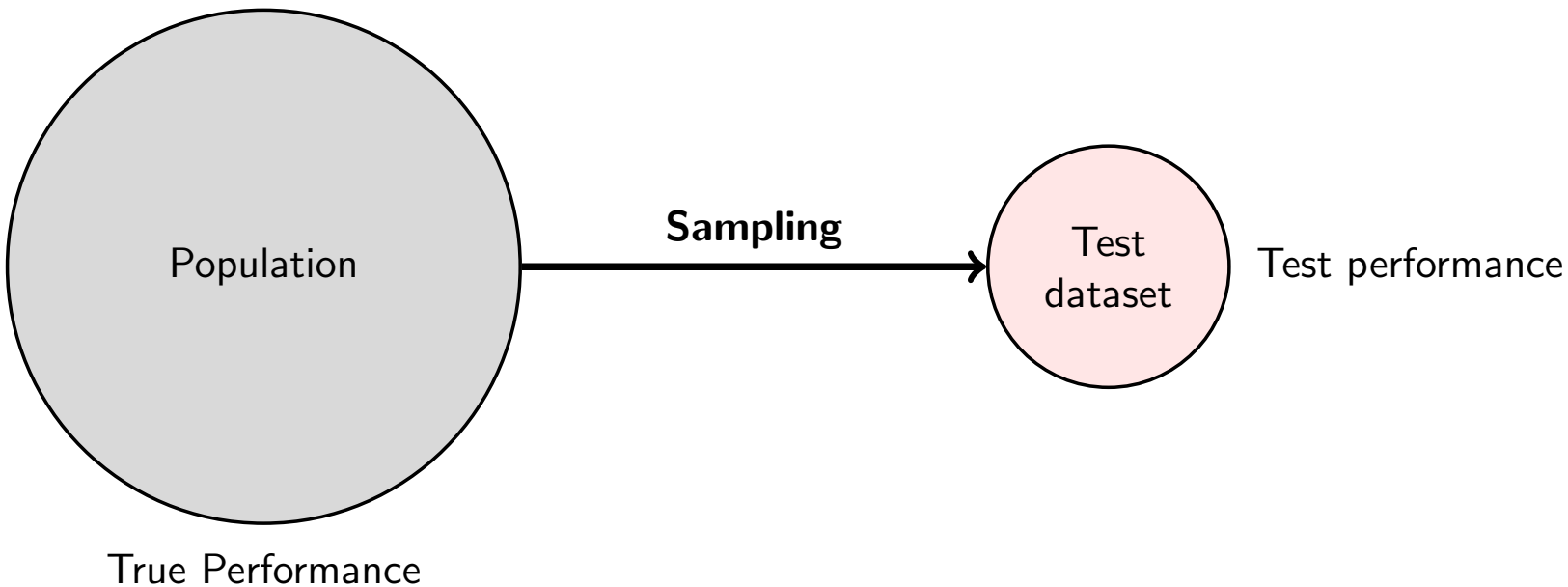
1. A **quality metric** used to quantify the performance.
2. How **data** will be used to assess the performance of a model.

The performance evaluation strategy has to be designed **before** creating a model to avoid falling into our own data-traps, such as confirmation bias.

Assessing the deployment performance

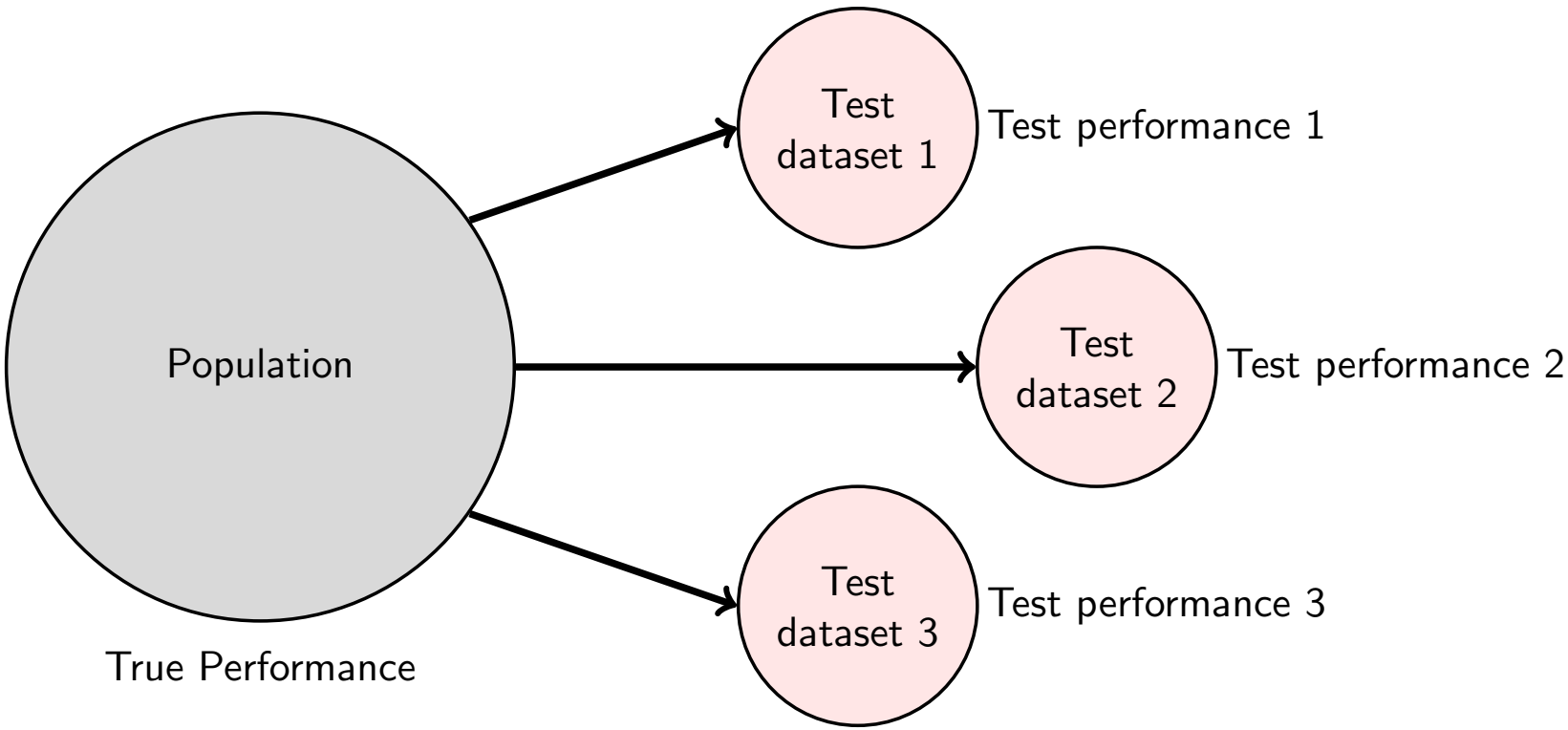
If we could use **all the data** that models can be shown (population), we would be able to quantify their **true** deployment performance.

Instead we use a subset of data, the **test dataset**, to compute the **test deployment performance** as an **estimation** of the true performance.



Random nature of the test performance

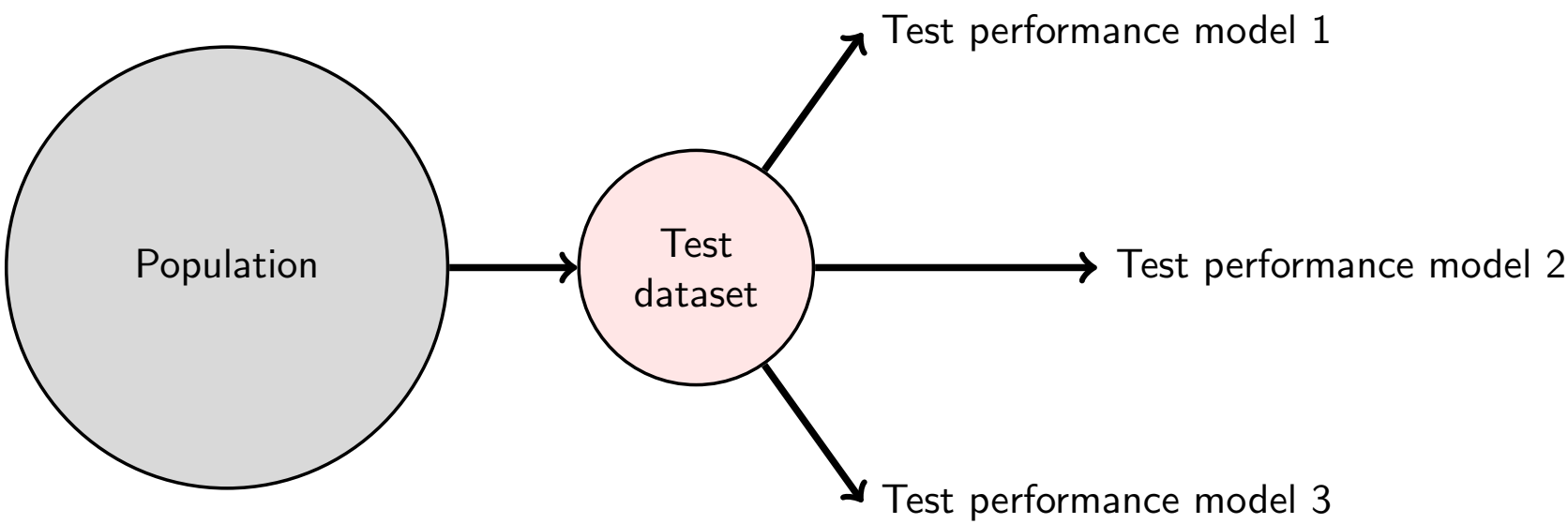
Test datasets are extracted randomly. Hence, the **test performance** is itself **random**, as different datasets generally produce different values.



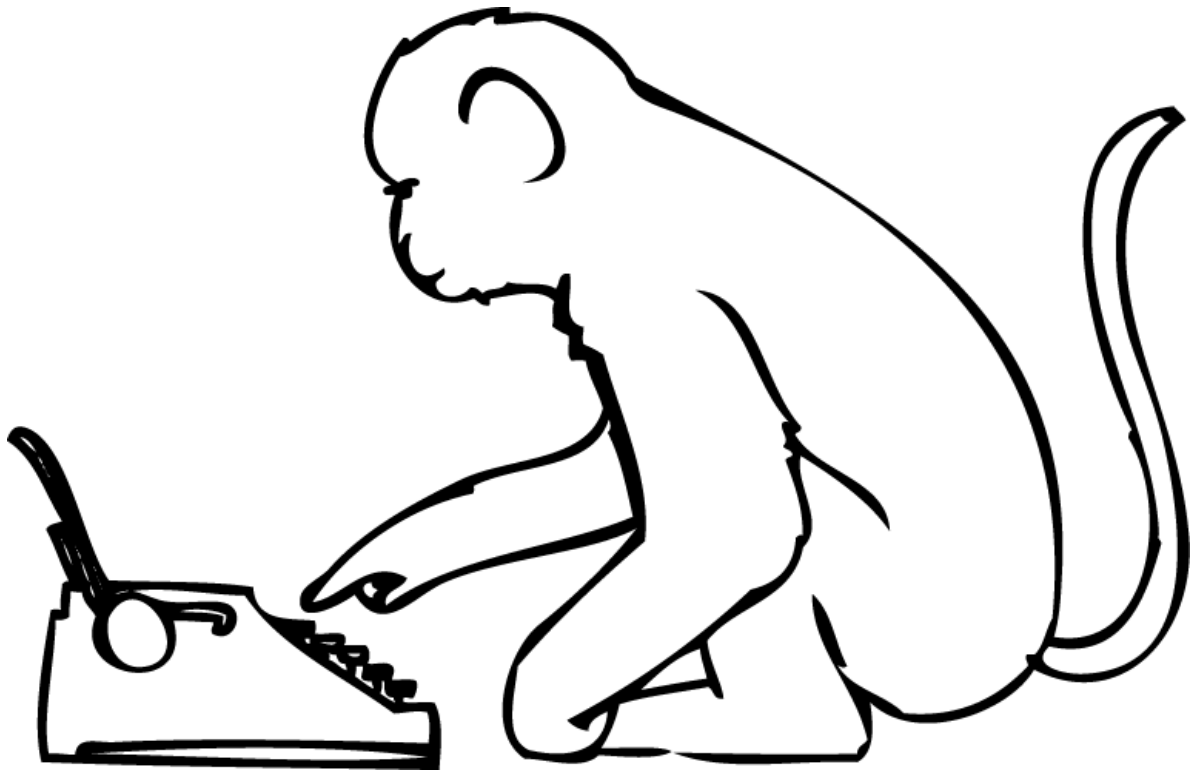
Comparing models

Models built by different teams can be compared based on their test performances.

Caution should be used, as the test performance is a random quantity, hence some models might appear to be **superior by chance**!



The Infinite Monkey Theorem



Agenda

Testing a model to evaluate its deployment performance

Optimisation and the error surface

Training a model

Validating our models

Summary

Optimisation theory

Assume that we have:

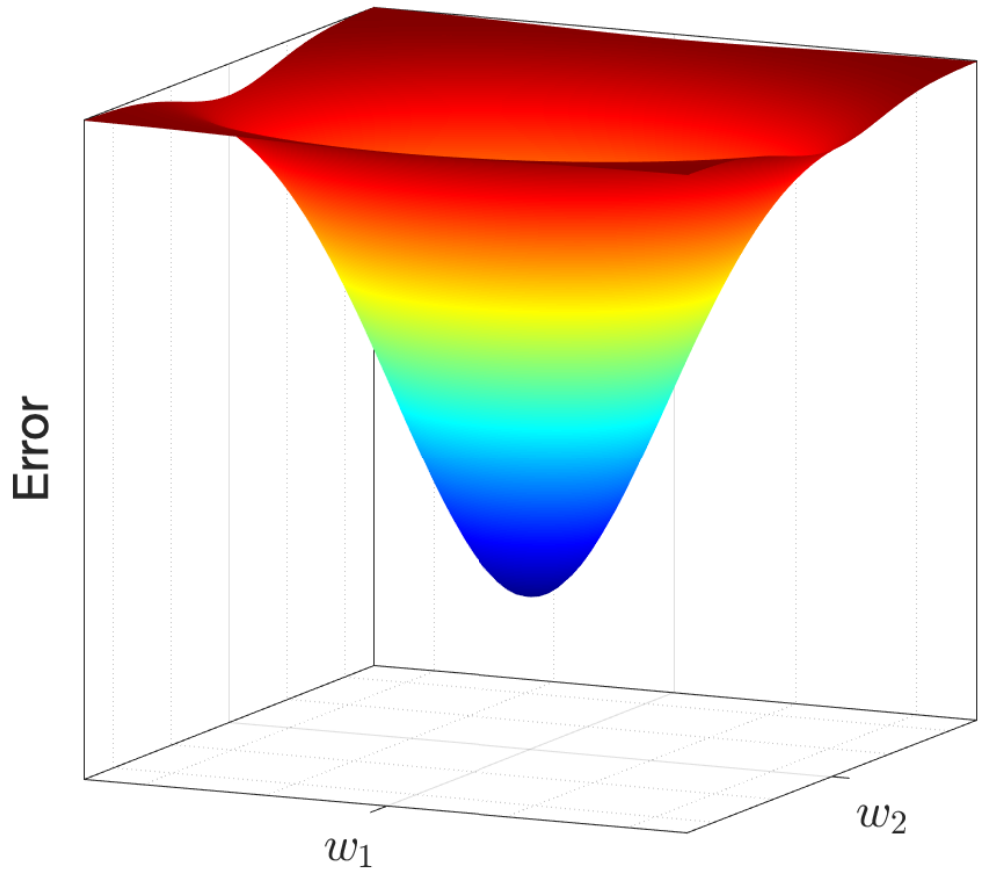
- A collection of **candidate models**, e.g. all the models resulting from tuning a linear model.
- A **quality metric**, e.g. a notion of error.
- An **ideal description** of the target population.

Optimisation allows us to identify among all the candidate models the one that achieves the highest quality on the target population, i.e. the **optimal** model.

(*Note: Even though we are looking for the best model, this is not machine learning. We don't need data, as we know the target population.*)

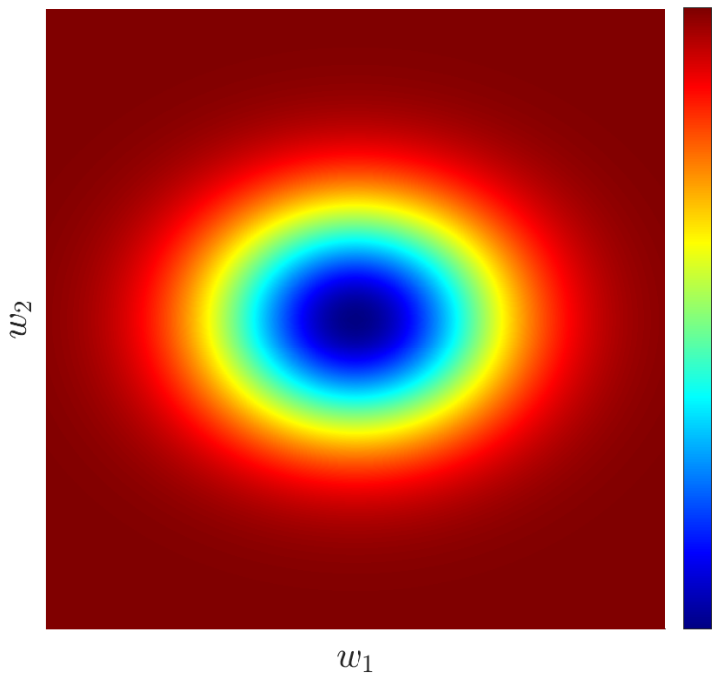
The error surface

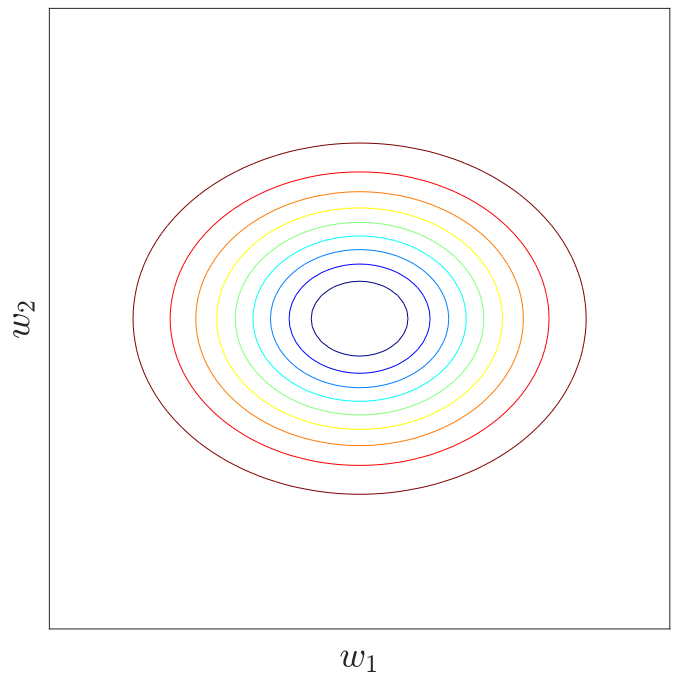
The error surface (a.k.a. error, objective, loss or cost function) denoted by $E(\boldsymbol{w})$ maps each **candidate model** $\boldsymbol{w}$ to its **error**. We will assume that we can obtain it using the ideal description of our target population.



The error surface

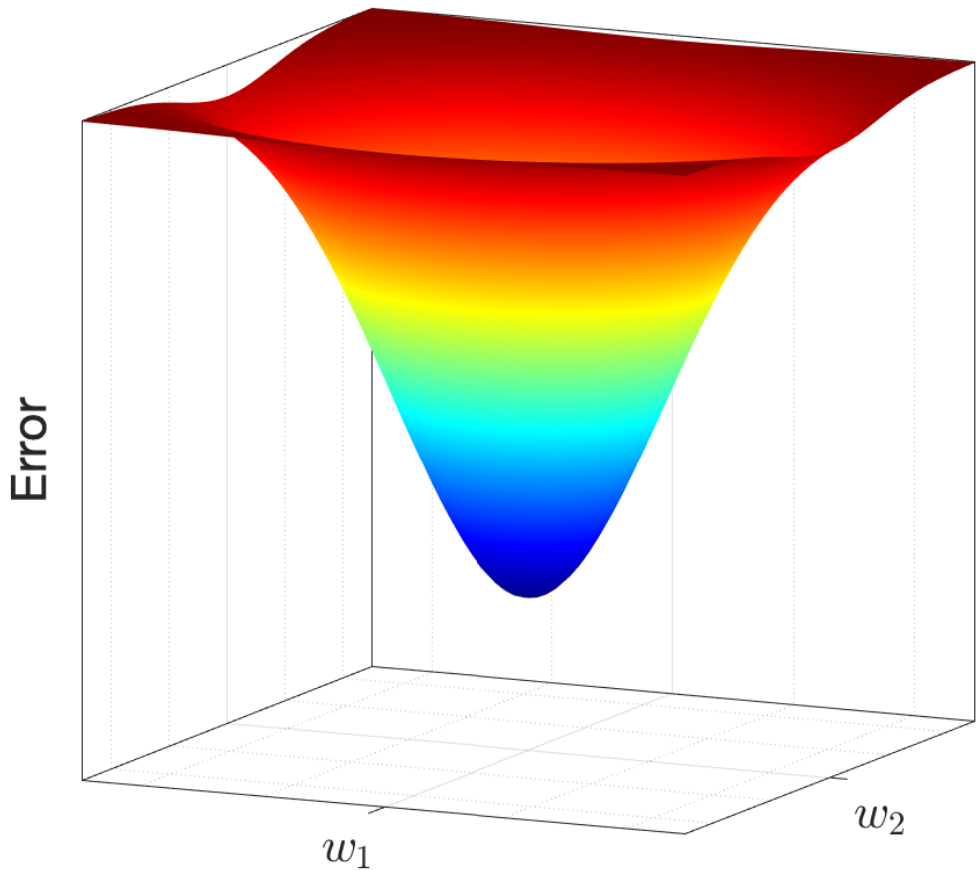
Error surfaces can also be represented as colour-coded and contour maps, where a colour scheme encodes the error values.





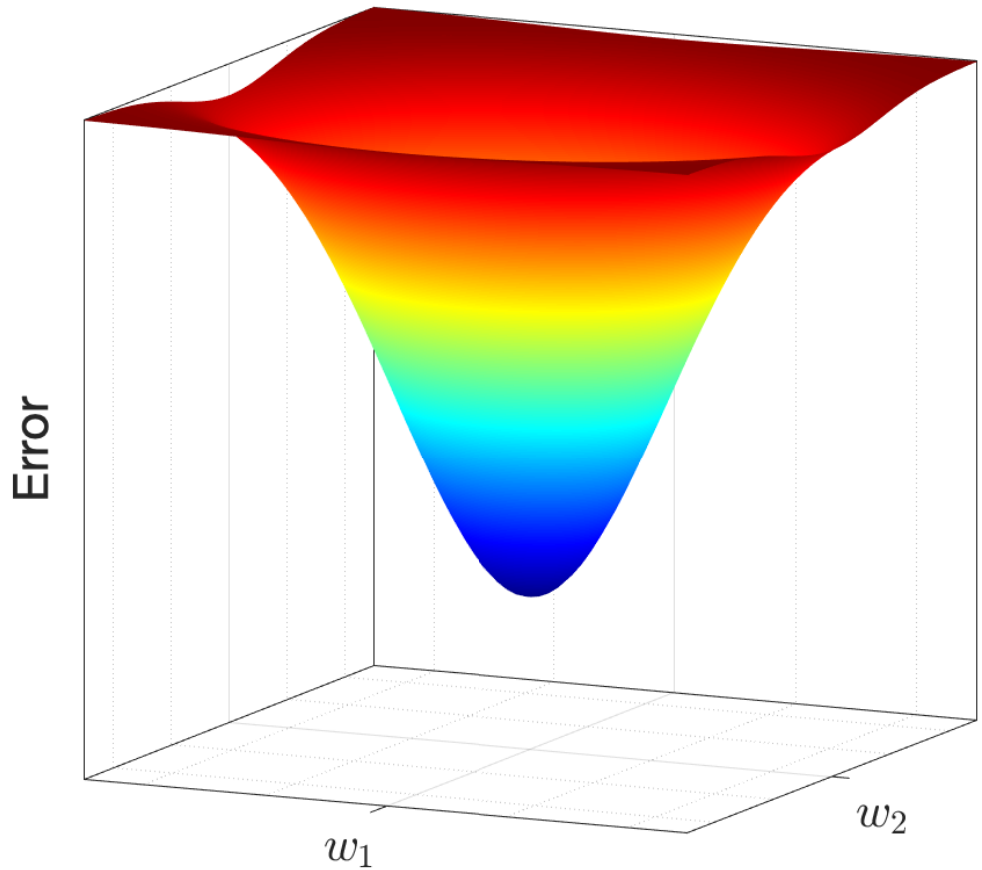
The error surface and the optimal model

The **optimal** model can be identified as the one with the lowest error.



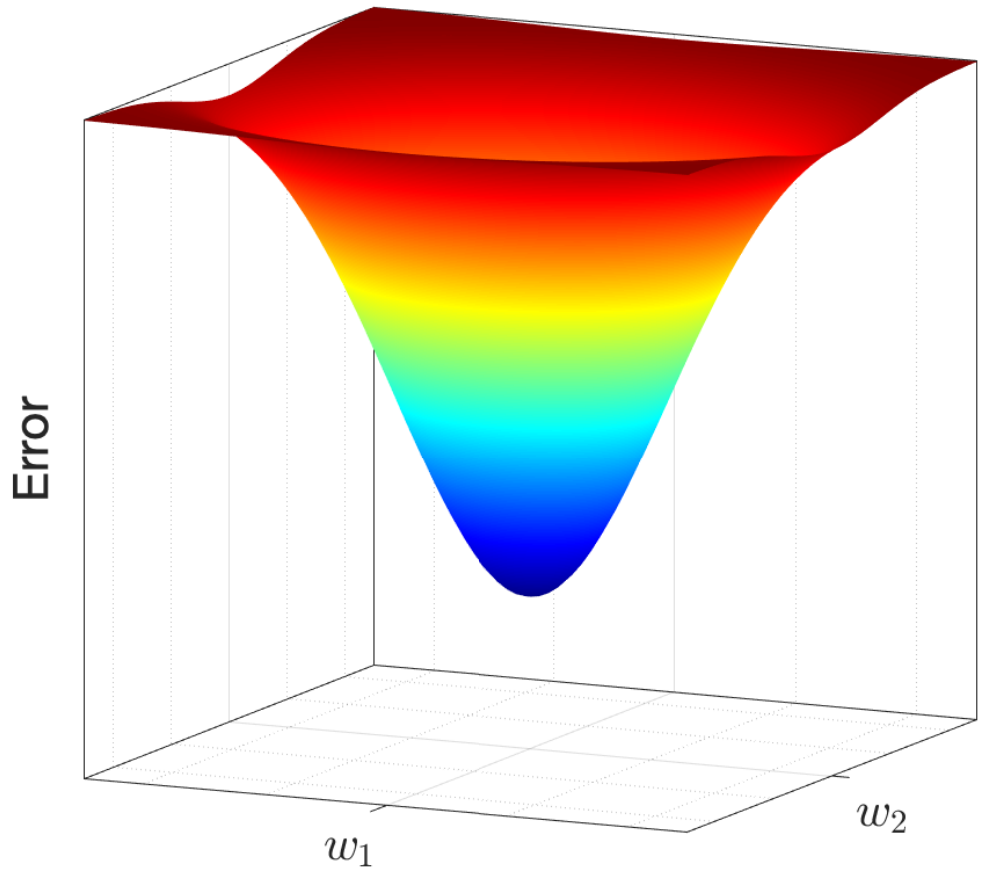
The error surface and the optimal model

The **gradient** (slope) of the error surface, $\nabla E(\boldsymbol{w})$, is zero at the optimal model. Hence we can look for it by identifying where $\nabla E(\boldsymbol{w}) = 0$.



The error surface and the optimal model

What if do not have enough computation power to obtain the error or the gradient of **every candidate model**, how can we find the optimal model?



Ask yourself: can you climb up/down a mountain in total darkness?

Gradient descent

Gradient descent is a numerical optimisation method where we **update iteratively** our model using the gradient of the error surface.

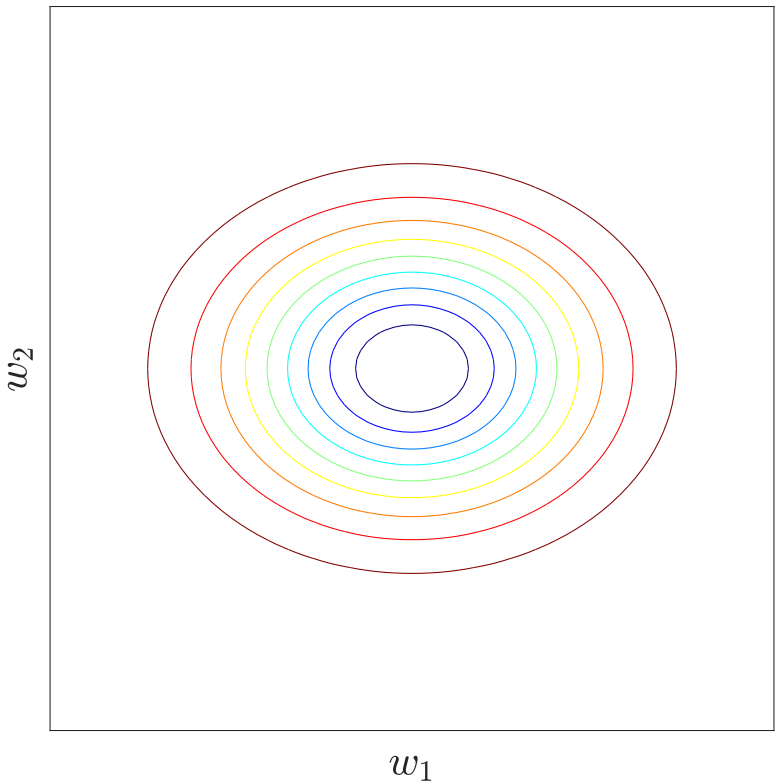
The gradient provides the direction along which the error increases the most. Using the gradient, we can create the following update rule:

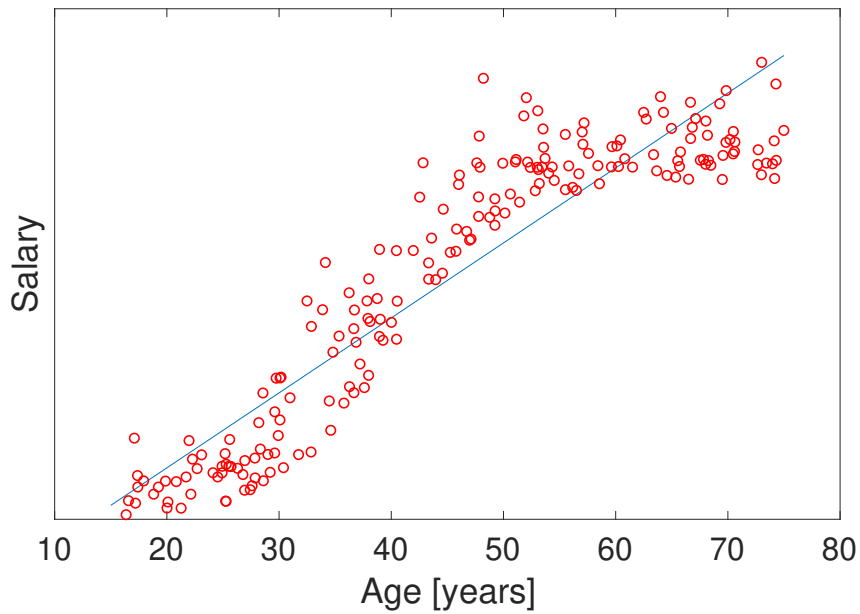
$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w}_{\text{old}} - \epsilon \nabla E(\boldsymbol{w}_{old})$$

where ϵ is known as the **learning rate** or **step size**.

With every iteration we adjust the parameters $\boldsymbol{w}$ of our model. This is why this process is also known as **parameter tuning**.

Gradient descent





Note that we are plotting a dataset for illustration purposes: the error surface is supposed to have been derived from the ideal description of the population.

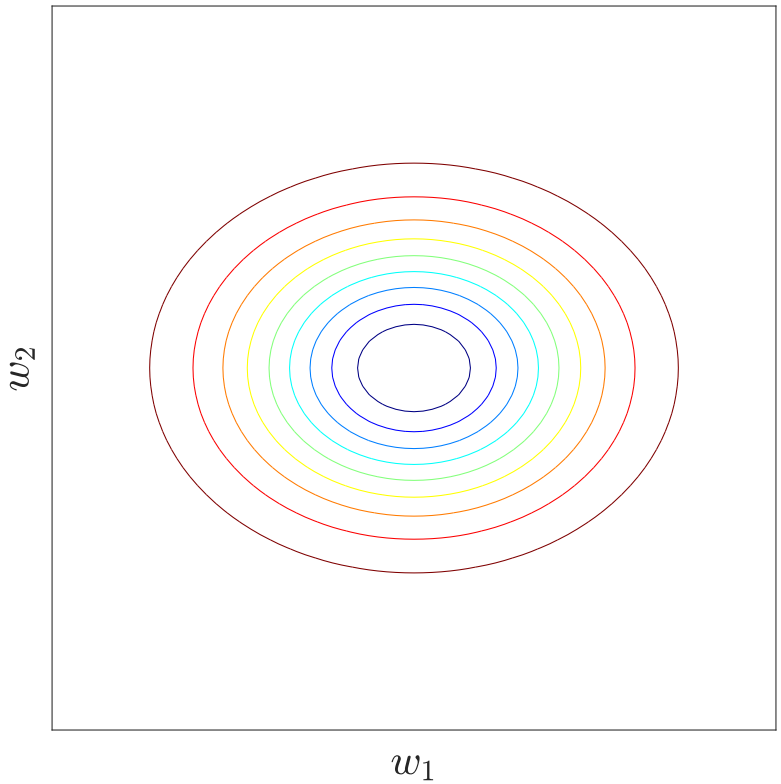
The learning rate

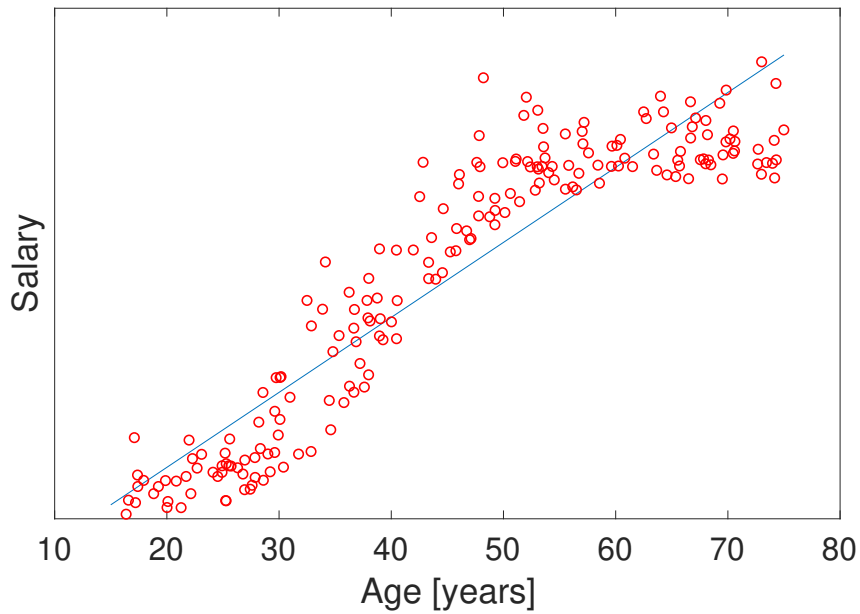
The learning rate ϵ controls how much we change the parameters $\boldsymbol{w}$ of our model in each iteration of gradient descent:

- Small values of ϵ result in slow convergence to the optimal model.
- Large values of ϵ risk overshooting the optimal model.

Adaptive approaches can be implemented, where the value of the learning rate decreases progressively.

The learning rate





Starting and stopping

For gradient descent to start, we need an initial model. The choice of the initial model can be crucial. The initial parameters $\boldsymbol{w}$ are usually chosen **randomly** (but within a sensible range of values).

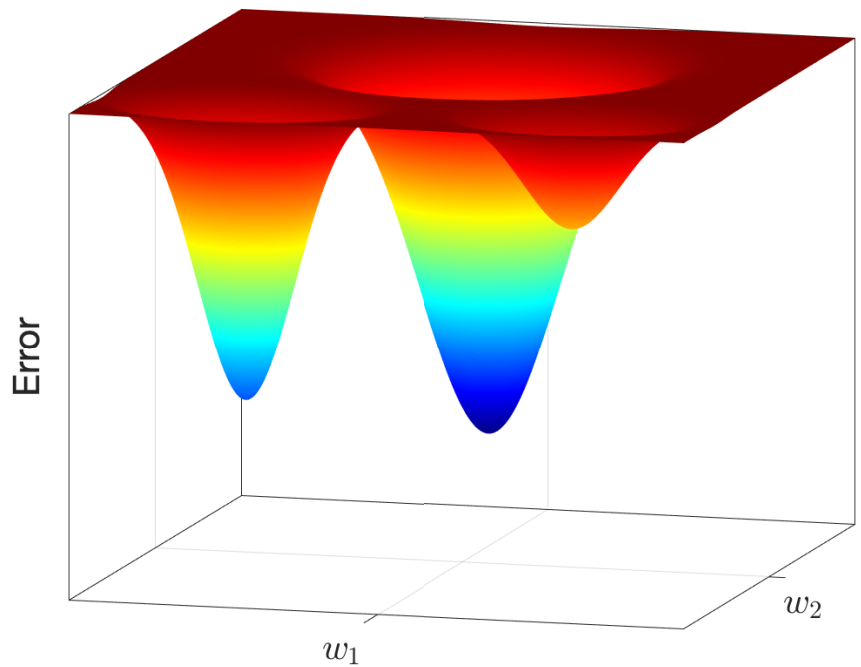
In general, gradient descent will not reach the optimal model, hence it is necessary to design a stopping strategy. Common choices include:

- Number of iterations.
- Processing time.
- Error value.
- Relative change of the error value.

Local and global solutions

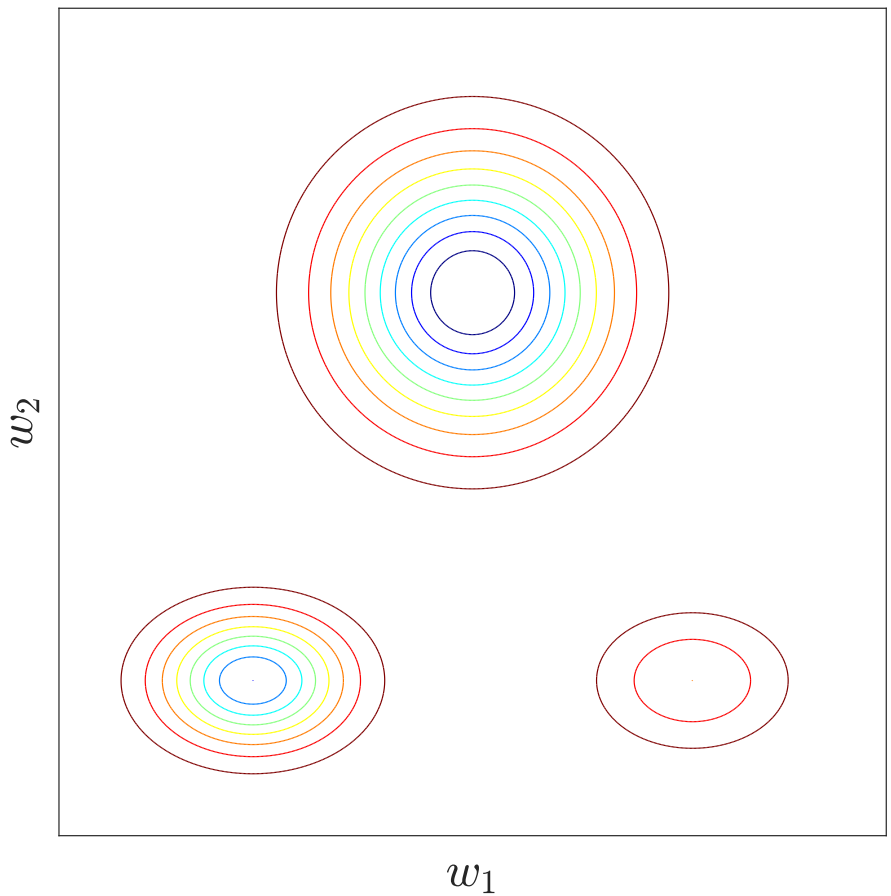
So far we have considered **convex** error surfaces. Error surfaces can however be complex and have

- **Local** optima (model with the lowest error within a region).
- **Global** optima (model with the lowest error among all the models).



Local and global optimal solutions

Gradient descent can get **stuck** in local optima. To avoid them, we can repeat the procedure from several **initial models** and select the best.



Agenda

Testing a model to evaluate its deployment performance

Optimisation and the error surface

Training a model

Validating our models

Summary

Where is my error surface?

In machine learning we have:

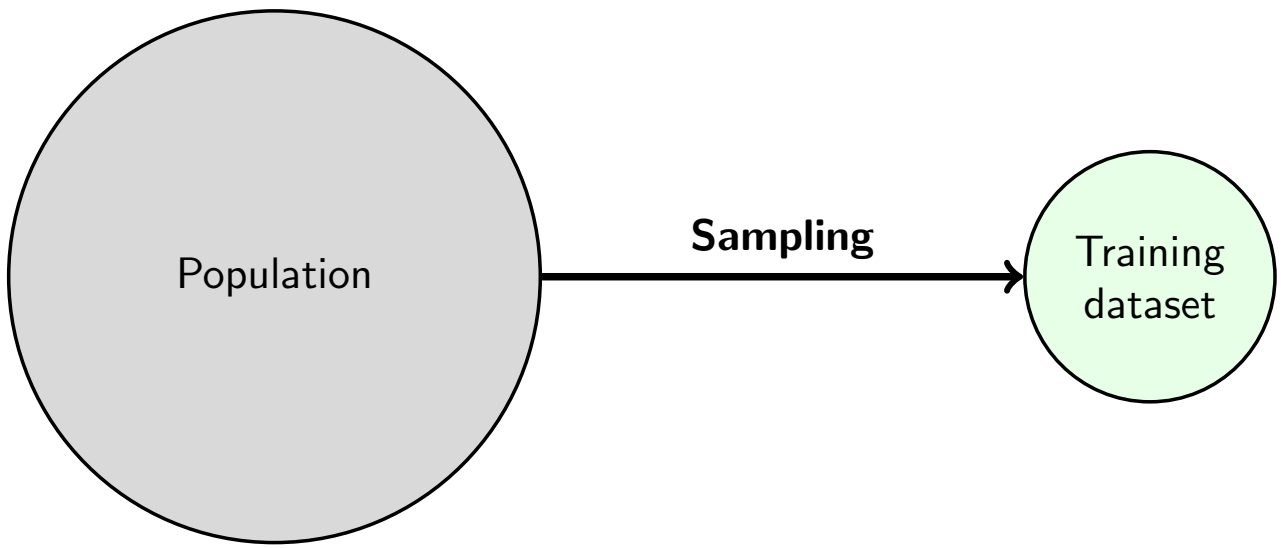
- A family of **candidate models** (e.g. linear models).
- A **quality metric** (e.g. the error).
- **Data** extracted from the population (i.e. **not** an ideal description).

The notion of error surface allows us to identify the model with the lowest error among the candidate models. However, where is my error surface?

If we had an ideal description of the target population we could calculate the error surface and its gradient. In machine learning, our starting point is that we do not have it and we only have data.

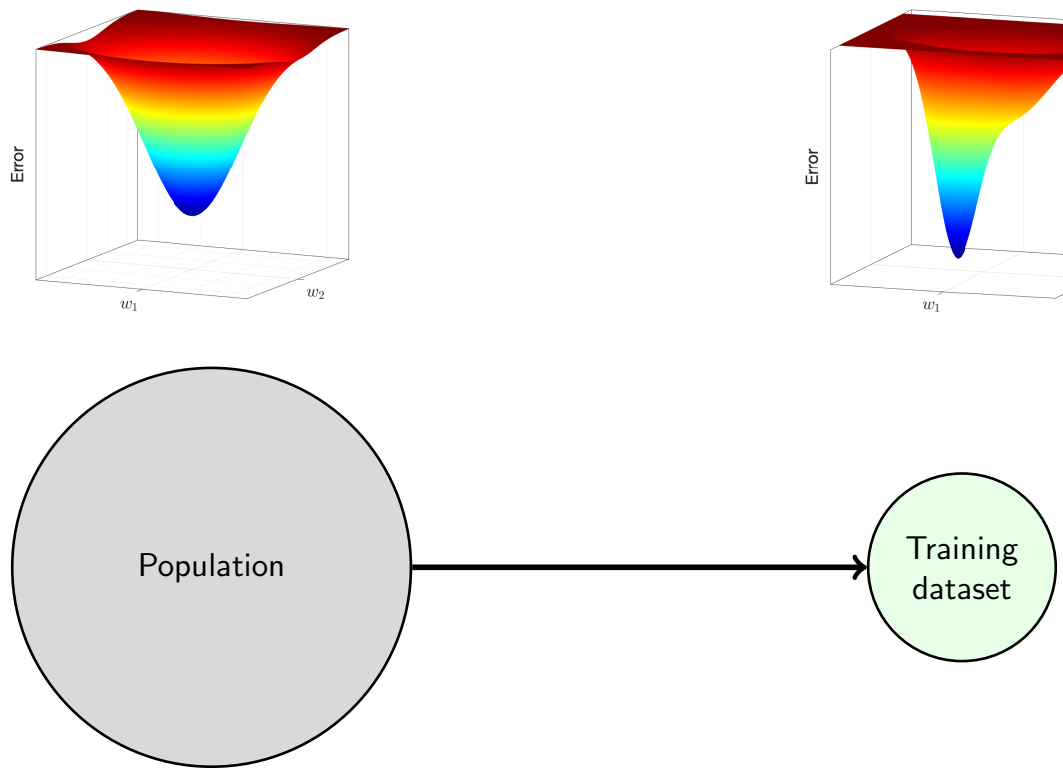
The training dataset

We use a subset of data, known as the **training dataset**, to (implicitly or explicitly) **reconstruct** the error surface needed during optimisation. We will call this the **empirical error surface**.



The training dataset

The empirical and true error surfaces are in general different. Hence, their optimal models might differ, i.e. the best model for the training dataset might not be the best for the population.



Least squares: minimising the error on a training dataset

Least squares provides one of the few analytical solutions defining an optimal model from data. A linear model applied to a training dataset can be expressed as

$$\hat{\boldsymbol{y}} = \mathbf{X}\boldsymbol{w}$$

where $\mathbf{X}$ is the design matrix, $\hat{\boldsymbol{y}}$ is the predicted label vector and $\boldsymbol{w}$ is the coefficients vector. The MSE on the training dataset can be written as

$$\begin{aligned} E_{MSE}(\boldsymbol{w}) &= \frac{1}{N}\left(\boldsymbol{y} - \hat{\boldsymbol{y}}\right)^T \left(\boldsymbol{y} - \hat{\boldsymbol{y}}\right) \\ &= \frac{1}{N}\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)^T \left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right) \end{aligned}$$

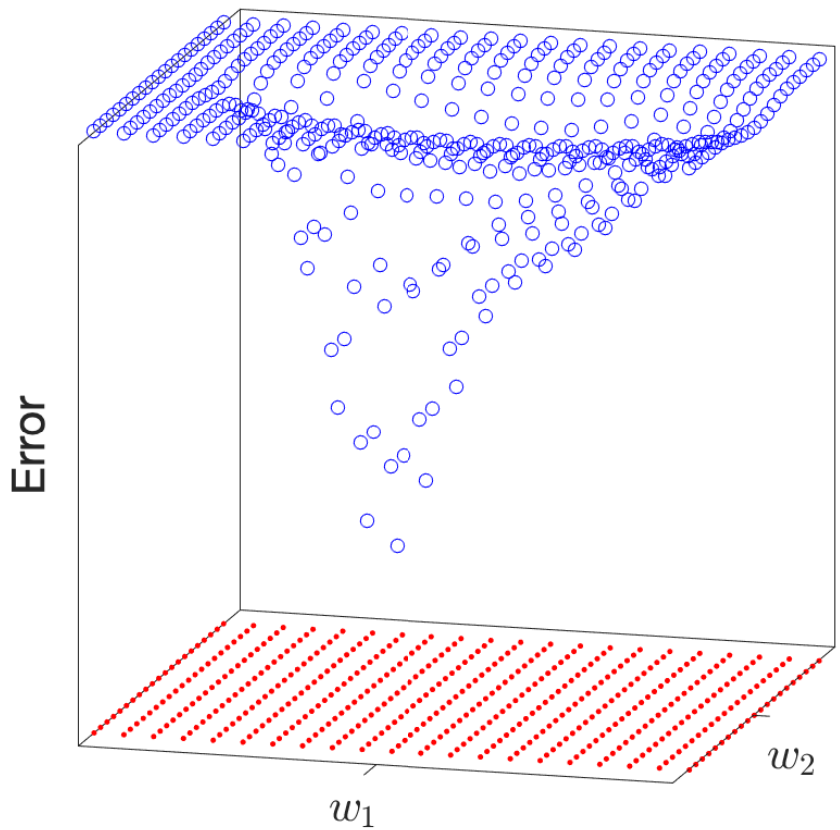
where $\boldsymbol{y}$ is the true label. The resulting gradient of the MSE is

$$\nabla E_{MSE}(\boldsymbol{w}) = \frac{-2}{N}\mathbf{X}^T\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{w}\right)$$

and it is zero for $\boldsymbol{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\boldsymbol{y}$.

Exhaustive approaches

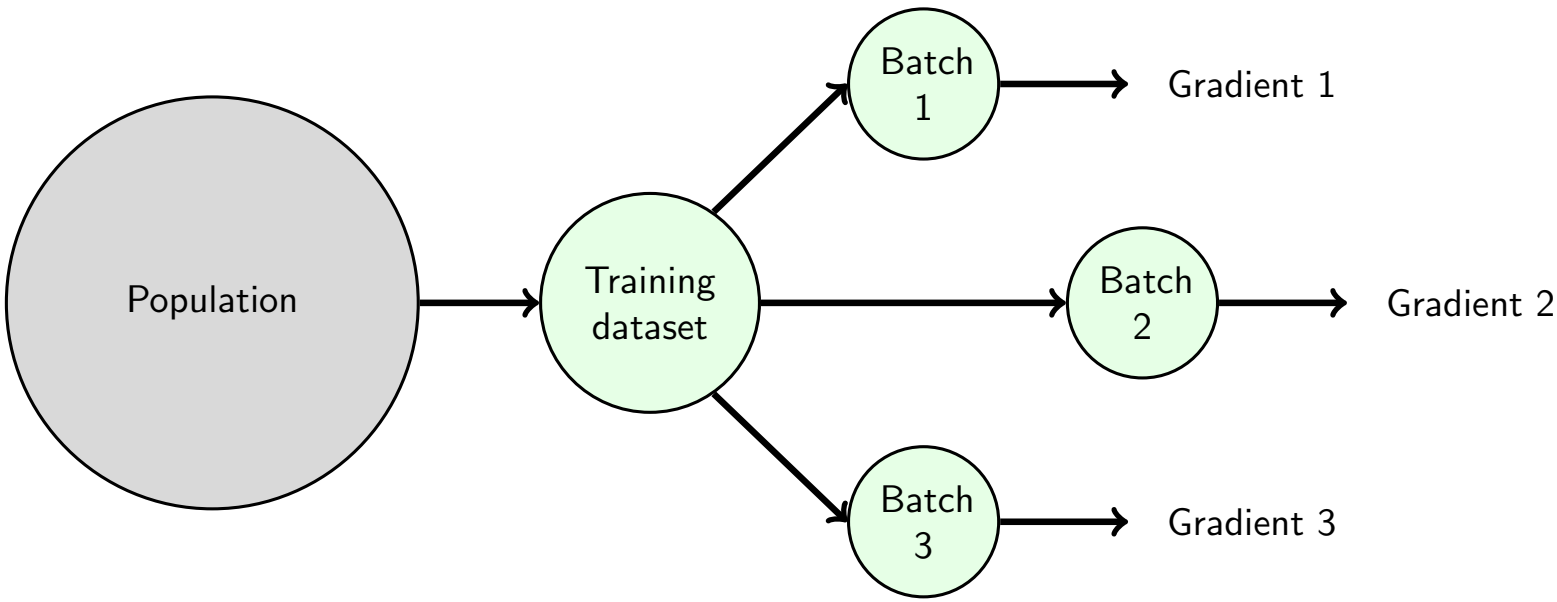
In general, we will not have analytical solutions. We can reconstruct the empirical error surface by evaluating each model on training data. This is called **brute-force** or **exhaustive search**. Simple, but often **impractical**.



Data-driven gradient descent

Gradient descent can be implemented by **estimating the gradient** using our training dataset.

During each iteration, a subset (**batch**) of the training dataset is used to compute the gradient of the error surface.



Data-driven gradient descent

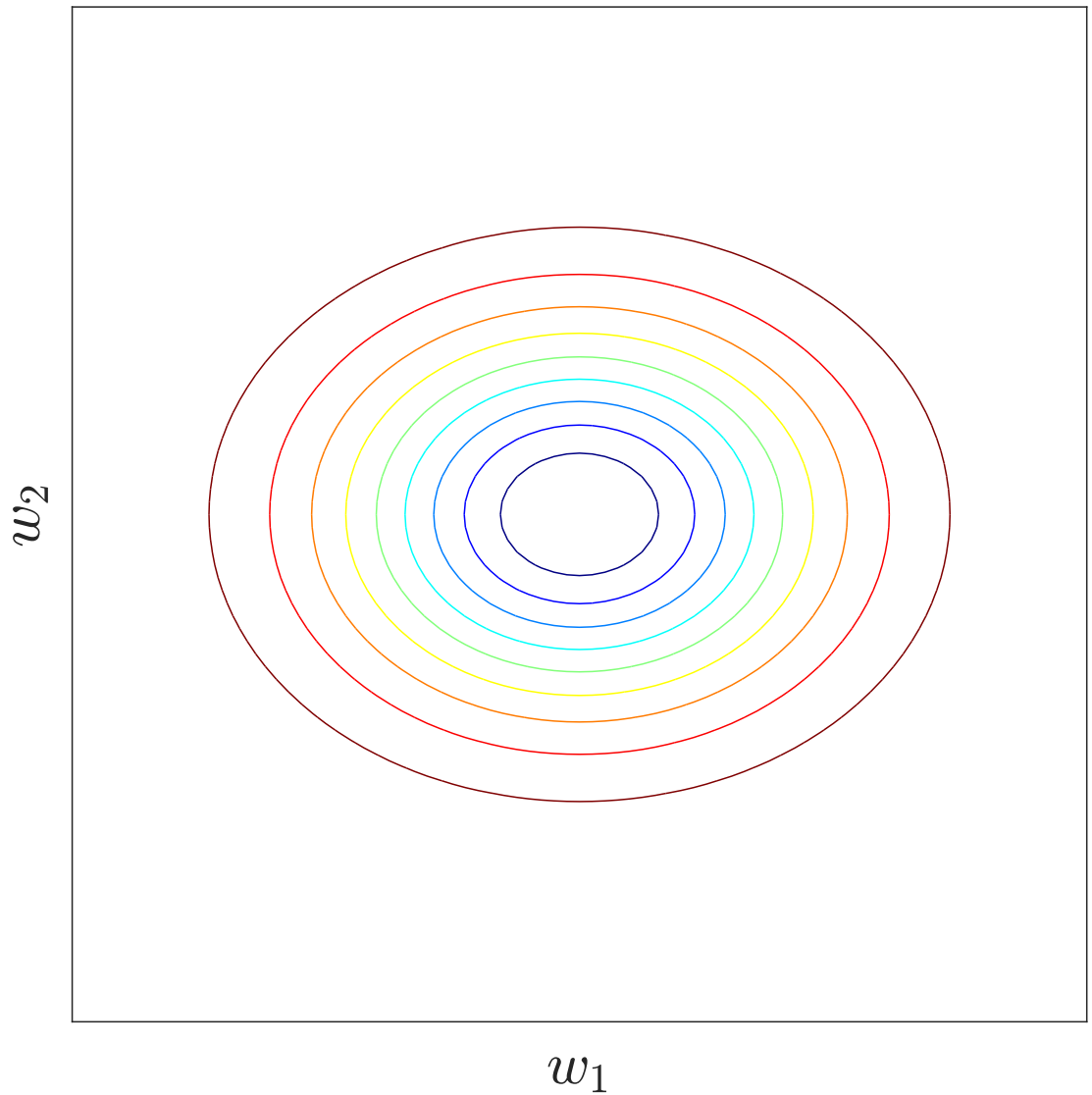
Depending on the amount of data used in each iteration, it is common (although not really useful) to distinguish between:

- **Batch** gradient descent (the whole training dataset is used).
- **Stochastic** (or online) gradient descent (one sample is used).
- **Mini-batch** gradient descent (a small subset from the training dataset is used).

It is more useful to talk about the **batch size** (a number between 1 and the size of the training dataset). Irrespective of the value of the batch size, we will use the term **stochastic gradient descent** for this approach.

Small batches produce noisy versions of the gradient of the empirical error surface, which can help to escape local minima.

Batch size in data-driven gradient descent



Other gradient-based optimisation approaches

Stochastic gradient descent is the most used optimisation algorithm, although it can sometimes be slow. Other popular gradient-based optimisation algorithms include:

- **Momentum** defines a velocity (direction and speed) for the update step, which depends on past gradients.
- **RMSProp** adapts the learning rate by scaling them using the past gradients.
- **Adam** combines some features from the Momentum and RMSProp approaches.

Overfitting and fooling ourselves

The empirical and true error surfaces are in general different. When **small datasets** and **complex models** are used, the differences between the two can be very large, resulting in trained models that work very well for the empirical error surface but very poorly for the true error surface.

This is, of course, another way of looking at **overfitting**. By increasing the size of the training dataset, empirical error surfaces becomes closer to the true error surface and the risk of overfitting decreases.

Never use the same data for testing and training a model. The test dataset needs to remain **inaccessible** to avoid using it (inadvertently or not) during training.

Regularisation

Regularisations modifies the empirical error surface by adding a term that constrains the values that the model parameters can take on. A common option is the regularised error surface $E_R(\boldsymbol{w})$ defined as:

$$E_R(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

For instance, the MSE in regression can be regularised as follows:

$$E_{MSE+R} = \frac{1}{N} \sum_{i=1}^{N} e_i^2 + \lambda \sum_{i=1}^{K} w_k^2$$

and its MMSE solution is

$$\boldsymbol{w} = \left(\mathbf{X}^T \mathbf{X} + N\lambda \mathbf{I} \right)^{-1} \mathbf{X}^T \boldsymbol{y}$$

As λ increases, the complexity of the resulting solution decreases and so does the risk of overfitting.

Cost vs quality

Regularisation provides an example where we use a notion of quality during training (E_{MSE+R}) that is different from the notion of quality during deployment (E_{MSE}). Doesn't it sound strange?

Our goal is always to produce a model with that achieves the highest **quality** during deployment. How we achieve it, is a different question.

Factors such as overfitting might result in models that are optimal during training, but not deployment. A well-designed notion of quality during training can produce models that perform better during deployment.

We usually call our notion of quality during training **cost** or **objective function**, to distinguish it from the **target quality metric**.

Agenda

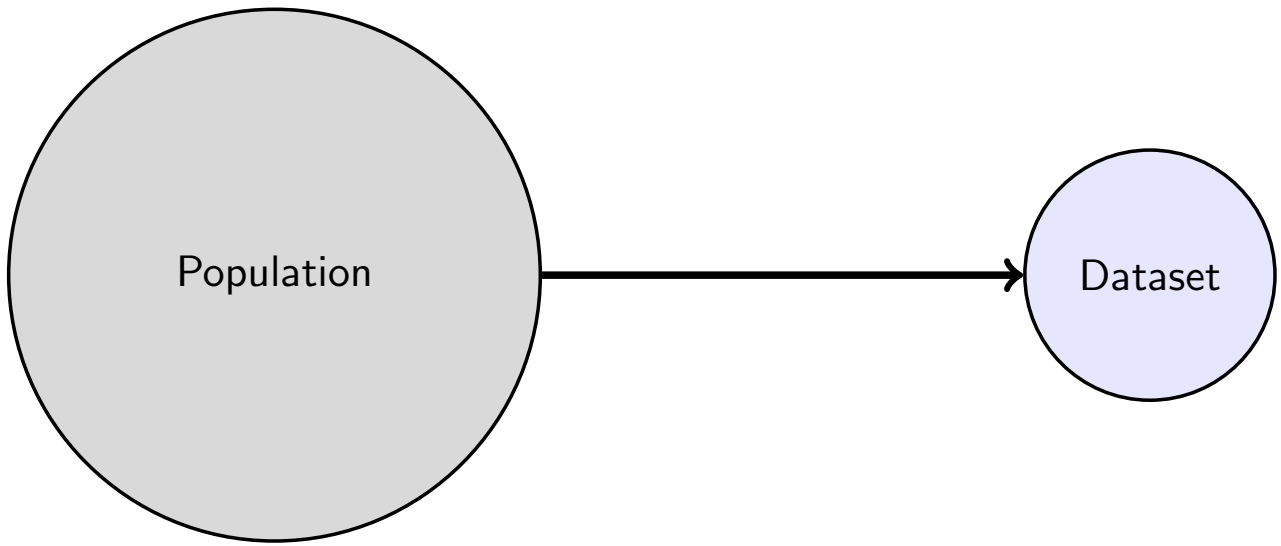
Why do we need validation?

Machine learning uses datasets for different purposes, for instance to assess the deployment performance of a final model (**test dataset**) or to tune a model (**training dataset**).

Often, we need to explore different options before training a final model. For example, consider polynomial regression. The polynomial degree D is a **hyperparameter**, as for each value of D a different family of models is obtained. How can we select the right value of a D?

Why do we need validation?

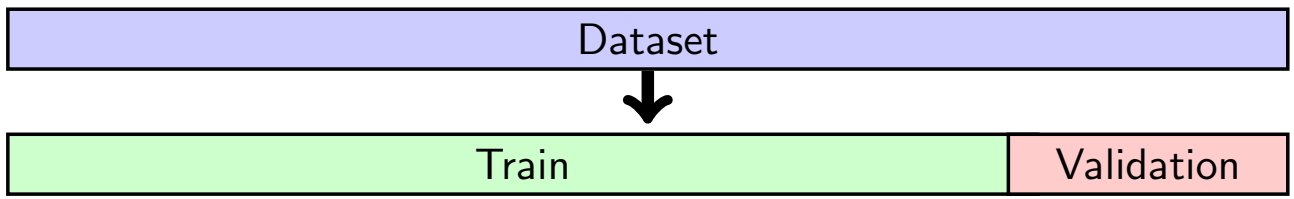
Validation methods allow us to use data for **assessing** and **selecting** different families of models. The same data used for validation can then be used to train a final model.



Validation involves one or more **training** and **performance estimation** rounds per model family followed by **performance averaging**.

Validation set approach

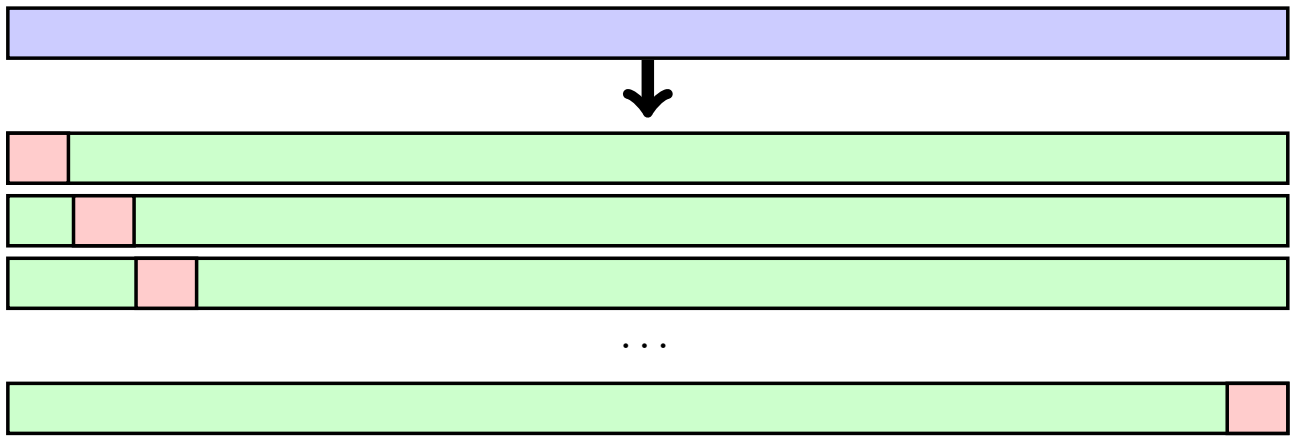
The validation set approach is the simplest method. It randomly splits the available dataset into a **training** and a **validation** (or hold-out) dataset.



Models are then fitted with the training part and the validation part is used to estimate its performance.

Leave-one-out cross-validation (LOOCV)

This method also splits the available dataset into training and validation sets. However, the **validation set contains only one sample**.

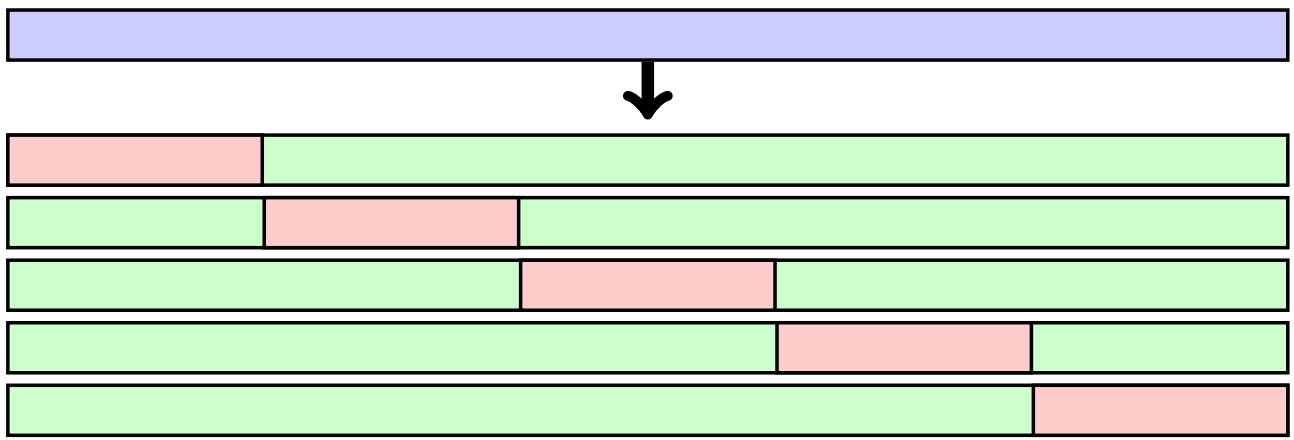


Multiple splits are considered and the final performance is calculated as the average of the individual performances (for N samples, we produce N splits and obtain N different performances).

k-fold cross-validation

In this approach the **available dataset is divided into k groups** (also known as folds) of approximately equal size:

- We carry k **rounds of training followed by validation**, each one using a different fold for validation and the remaining for training.
- The final estimation of the performance is the average of the performances from each round.



LOOCV is a special case of k-fold cross-validation, where $k = N$.

Validation approaches: Comparison

- The **validation set** approach involves one training round. Models are however trained with fewer samples and the final performance is highly variable due to random splitting.
- **LOOCV** requires as many training rounds as samples there are in the dataset, however in every round almost all the samples are used for training. It always provides the same performance estimation.
- k-**fold** is the most popular approach. It involves fewer training rounds than LOOVC. Compared to the validation set approach, the performance estimation is less variable and more samples are used for training.

Agenda

Testing a model to evaluate its deployment performance

Optimisation and the error surface

Training a model

Validating our models

Summary

Machine learning methodology: Basic tasks

In machine learning we can identify the following tasks:

- **Test**: This is the **most** important task. It allows us to estimate the deployment performance of a model.
- **Training**: Used to find the best values for the parameters of a model, i.e. to tune a model.
- **Validation**: Necessary to compare different modelling options and select the best one, the one that will be trained.

The role of data

In machine learning we do not have an ideal description of the target population, all we can do is **extract data**.

- Test, training and validation tasks all involve data.
- Hence we talk about test dataset, as the data used for testing a model, training dataset, as the data used for training a model, etc.
- Think about the tasks first, then create the datasets that you need.

So you've read about **splitting a dataset**? Splitting datasets is **not** an ML task. What we need is to **create the right dataset** for each task. This might involve splitting an existing dataset, but not necessarily.

Using data correctly

- Datasets need to be **representative** of the target population and its samples need to have been extracted **independently**.
- Any test strategy has to be **designed before training**. Avoid looking at the test dataset during training and test a final model only once (Monkey Theorem!).
- Any performance estimation that is obtained from a dataset is a **random quantity**: use with caution.
- The quality of a final model depends on the **type** of model, the **optimisation** strategy and the representativity of the training **data**.

Disappointed you didn't decipher Linear B?

Don't worry, we still have a few undeciphered scripts:

- Proto-Elamite
- Indus
- Meroitic
- Linear A
- Rongorongo
- Zapotec
- Voiynichese

The Voynich Manuscript, 15th century