

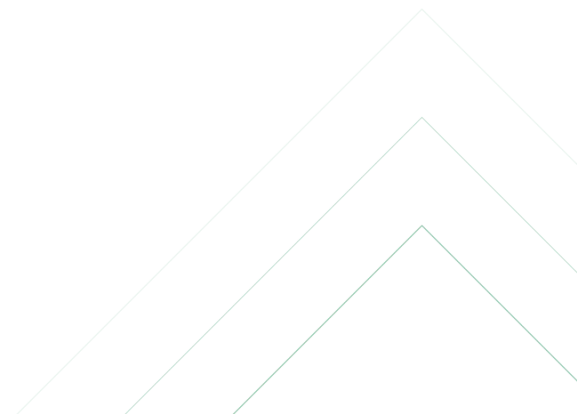
# Cours 5 : Sélection de modèles

Prof: Gauthier Gidel  
16 Septembre 2024

# Annonces

- Heure de bureau (cours sous-gradué) en Salle

- 



# Retour sur la matrice de confusion

- Si le **taux de positif est connu** ( $P/(P+N)$ ), alors **deux métriques différentes** suffisent pour remplir toute la matrice.  
Exemple (exemple du cancer de la poitrine du cours précédent)
- Sinon, on manque d'information.



|              |                     | Classe prédite            |                           |
|--------------|---------------------|---------------------------|---------------------------|
|              |                     | Positif ( $\hat{Y} = 1$ ) | Négatif ( $\hat{Y} = 0$ ) |
| Vraie classe | Positif ( $Y = 1$ ) | Vrais positifs (VP)       | Faux négatifs (FN)        |
|              | Négatif ( $Y = 0$ ) | Faux positifs (FP)        | Vrais négatifs (VN)       |

Nombre de bonnes classifications: VP+VN

Nombre d'erreurs: FN+FP

- Note: connaître la taille de l'échantillon n'est pas important (on ne s'intéresse qu'aux **taux**.)



# Introduction

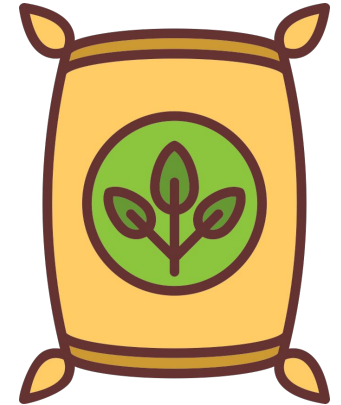




# Plan d'expérience

Facteurs:

$d = 4$



(quantité)

Niveaux:





Facteurs:

$$d = 4$$

(quantité)

Niveaux:



# Plan d'expérience



































# Plan d'expérience

s niveaux  
 $\{0, \dots, s-1\}$

 **simple**

 **moyen**





 **élevé**

| essais:  |   | 0   | 1   | 2   | 3   | 4  | 5   | 6   | ... |
|----------|---|---|---|---|---|--|---|---|-----|
| facteurs |  |  |  |  |  |  |  |  | ... |
|          |  |  |  |  |  |  |  |  | ... |
|          |  |  |  |  |  |  |  |  | ... |
|          |  |  |  |  |  |  |  |  | ... |

# Plan d'expérience

$s$  niveaux  
 $\{0, \dots, s-1\}$



| runs:    |   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... |
|----------|---|---|---|---|---|---|---|---|-----|
| facteurs |  | 0 | 0 | 0 | 1 | 1 | 1 | 2 | ... |
|          |  | 0 | 1 | 2 | 0 | 1 | 2 | 0 | ... |
|          |  | 0 | 1 | 2 | 1 | 2 | 0 | 2 | ... |
|          |  | 0 | 1 | 2 | 2 | 0 | 1 | 1 | ... |



# Situation courante au début d'un projet en apprentissage automatique (AA)

- On a un jeu de données que l'on veut utiliser.

**métrique de perf. connue**

- La tâche est connue à l'avance, car elle dépend de ce que l'on veut faire avec nos données:
  - Régression
  - Classification

- Quel modèle utiliser?
- Quels hyperparamètres utiliser?



# Rappel: paramètres *versus* hyperparamètres (HP)

Paramètres: quantités apprises avec les données.

Hyper paramètres: détermine comment l'apprentissage se fait.

Les performances d'un modèle sont contrôlées par plusieurs types d'**HP**.  
complexité du modèle (nombre de couches, taille des filtres, etc.),

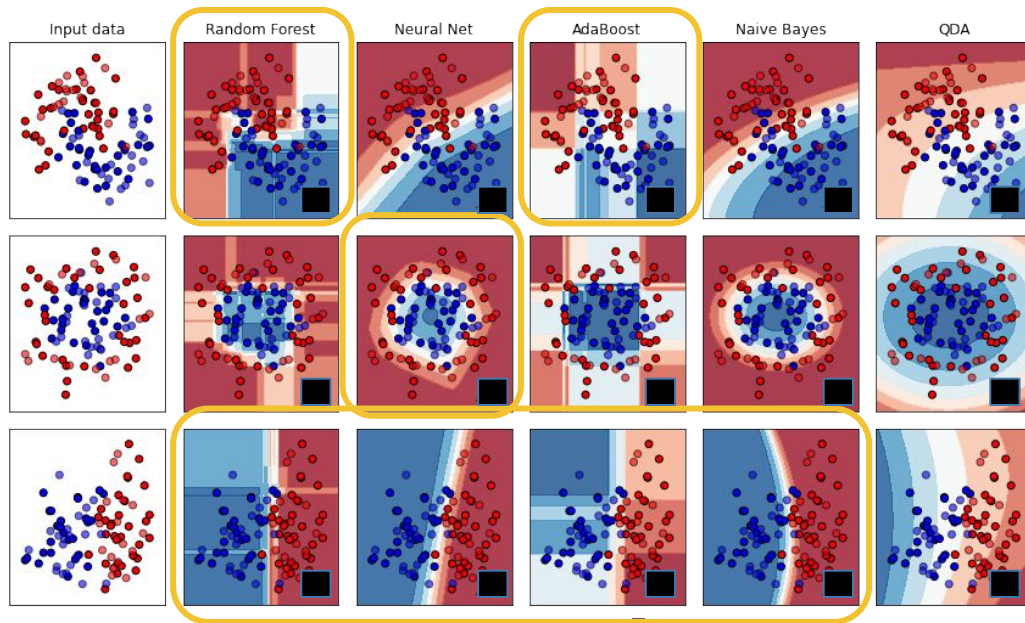
- fonction de perte, taux d'apprentissage, etc.,
- optimiseur,
- méthode de régularisation

Chaque combinaison d'**HP** détermine et influence:

- le nombre de **paramètres** du modèle,
- leurs valeurs finales des **paramètres** après entraînement,
- les performances du modèle.



# Exemples de classifications avec des modèles différents (2)

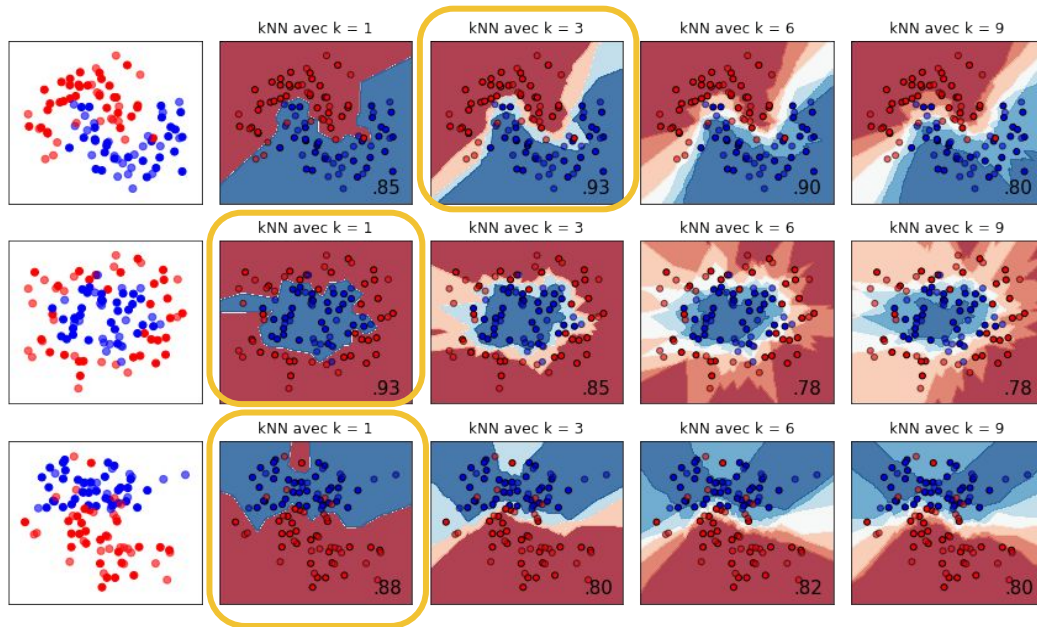


Exactitude  
(accuracy)

- Chaque modèle utilise les valeurs par défaut des HP de la librairie Scikit-learn.
- L'exactitude en validation est affichée pour chacun.
- Pour chaque nuage de points, le meilleur modèle change.
- **Aucun modèle n'est le meilleur dans tous les cas!**



# Classifications avec un même modèle, mais en variant un seul HP



- On utilise le même modèle en variant le nombre  $k$  de plus proches voisins.
- Tous les autres HP du modèle sont constants.
- Pour chaque nuage de points, la valeur optimale de  $k$  change.
- **Aucun HP n'est optimal dans tous les cas!**



- Peu probable que notre modèle préféré soit le meilleur.
- Tester plusieurs modèles sur un même jeu de données.
  - Différents jeux de données affectent les performances des modèles.
  - Il faut séparer la variance des modèles de la variance des données.
- Beaucoup de combinaisons (*modèle, HP*) à tester
  - Chacune correspond à un entraînement possiblement long à effectuer.
  - Les interactions entre les modèles et les HP sont complexes.

- Il faut procéder **systematiquement** et **methodiquement**.



- Fonctions spécialisées simplifiant la recherche de la meilleure méthode. (intégrée à Python, Keras, TensorFlow, etc.)
- Plans d'expériences (discutés plus loin): permettent d'optimiser la recherche d'HP en minimisant le nombre d'entraînements.



# Étapes préliminaires

...



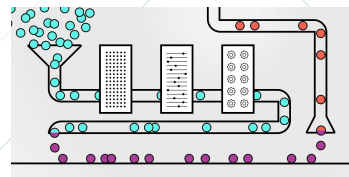
# Le prétraitement et la séparation des données

Pour optimiser un modèle ou de sélectionner un algorithme il faut:

- Construire un pipeline de **prétraitement des données**.
- Prétraiter les données **une seule fois!** (en particulier le test!!!!)

- ★ Séparer les données en **ensembles d'entraînement**, de **validation** et de **test**.
- Utiliser ceux-ci dans **toutes** les optimisations qui suivent.

N.B.: Les détails sur le prétraitement des données et les pipelines sont discutés dans un autre cours.



# La sélection des métriques de performance

- Important d'identifier les métriques que l'on veut utiliser

- Métriques classiques/académiques
  - **Classification**: exactitude, précision, rappel, etc.
  - **Régression**: moindres carrées, etc...
- Métriques définies en entreprise
  - Dépendent du problème étudié.

- Plusieurs métriques sont souvent mesurées en même temps lors d'entraînements.



# Procédure à effectuer pour optimiser un modèle

...

# Exemple typique pour un modèle de forêt aléatoire

Nombre d'hyperparamètres de la fonction *Random Forest* de la librairie Scikit-learn 1.1.1:

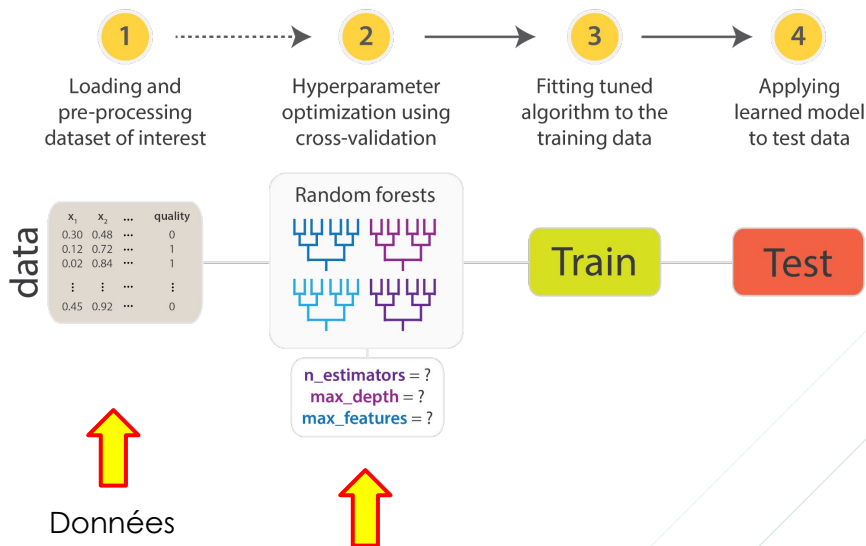
- RandomForest**Classifier**: 18
- RandomForest**Regressor**: 17

En pratique: On ajuste **les plus importants** d'entre eux, souvent 3 ou 4 seulement.

On entraîne et on teste le modèle avec la validation croisée.

## Exemple typique d'HP

- LR : taux apprentissage (réel > 0)
  - Batch size : taille mini-lot (entier > 0)
  - Beta 1 : momentum (réel entre 0 & 1)
- } cnt choisir intervalle de recherche?



# Exemple typique d'hyperparamètre

- LR; taux d'apprentissage (réel  $>0$ ) (Mon conseil: grille logarithmique)
- Batch-size: taille du mini-lot (entier  $>0$ ) (Mon conseil: aussi grand que possible en puissance de 2)
- beta\_1: momentum (réel entre 0 et 1) (Mon conseil: grille logarithmique proche de 1)

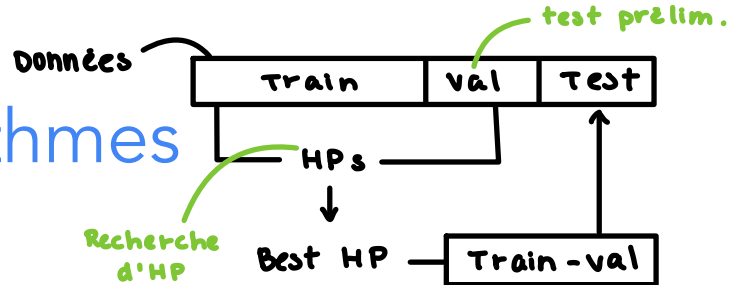
Comment choisir l'intervalle de recherche?



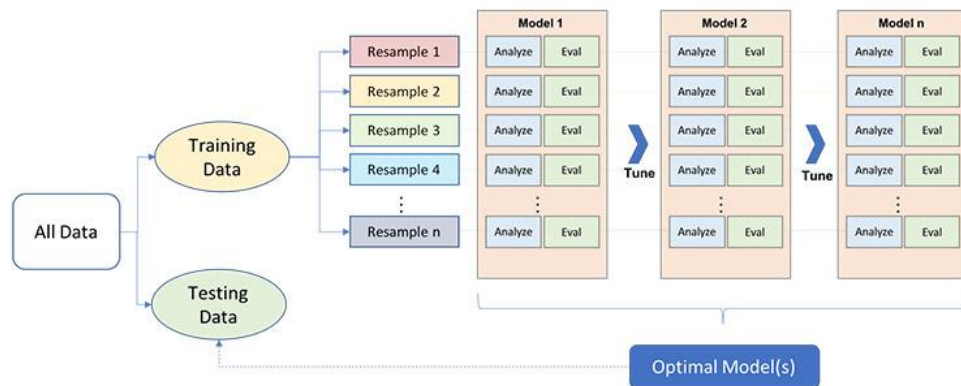
# Procédure à effectuer pour sélectionner l'algorithme optimal parmi plusieurs

...

# Test de plusieurs algorithmes



- Déterminer le meilleur algorithme parmi plusieurs (forêts aléatoires, XGBoost, etc.)
- Chacun pourrait être optimisé avant d'être comparé aux autres.
- Plutôt: tester toutes les combinaisons d'algorithmes et d'HP en même temps.
- Permet de comparer la robustesse des algorithmes.
- Les librairies (Python, Keras, TensorFlow, etc.) ont des fonctions spécialisées pour cela.



- Chaque 'modèle' correspond à une combinaison (algorithme, HP)
- Chaque modèle est entraîné et évalué
- Le meilleur des  $n$  modèles est sélectionné

# Optimisation d'un modèle avec plan d'expériences









# Plans d'expériences

$s$  niveaux discrets  
 $\{0, \dots, s-1\}$



| runs:    |   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... |
|----------|---|---|---|---|---|---|---|---|-----|
| facteurs |  | 0 | 0 | 0 | 1 | 1 | 1 | 2 | ... |
|          |  | 0 | 1 | 2 | 0 | 1 | 2 | 0 | ... |
|          |  | 0 | 1 | 2 | 1 | 2 | 0 | 2 | ... |
|          |  | 0 | 1 | 2 | 2 | 0 | 1 | 1 | ... |





# Plans d'expériences

✗ Tester toutes les combinaisons est très coûteux: ???  $s=3, d=4$

$s$  niveaux discrets  
 $\{0, \dots, s-1\}$

$$N = s^d = 3^4 = 81$$



| runs:   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... | 80 |
|---|---|---|---|---|---|---|---|-----|----|
|  0 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | ... | 2  |
|  1 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | ... | 2  |
|  2 | 0 | 1 | 2 | 1 | 2 | 0 | 2 | ... | 1  |
|  3 | 0 | 1 | 2 | 2 | 0 | 1 | 1 | ... | 0  |

# Plans d'expériences en grille et aléatoires

## Plan d'expériences en grille

- Plan par défaut dans Scikit-learn
- Le plus intuitif
- Le plus utilisé
- Le moins efficace de tous!

Exemple:

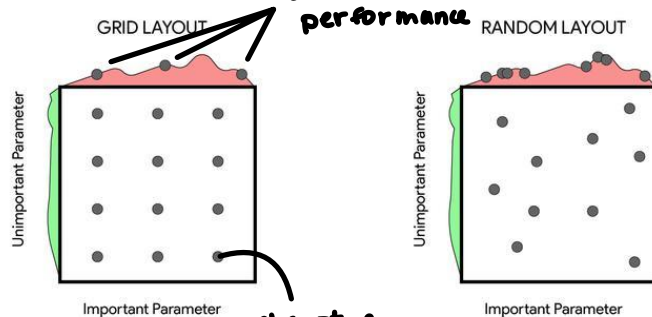
- On échantillonne une fonction en 12 points.
- La fonction dépend **seulement** de la variable **rouge**.

val. performance

= hauteur courbe verte

+ hauteur courbe rouge

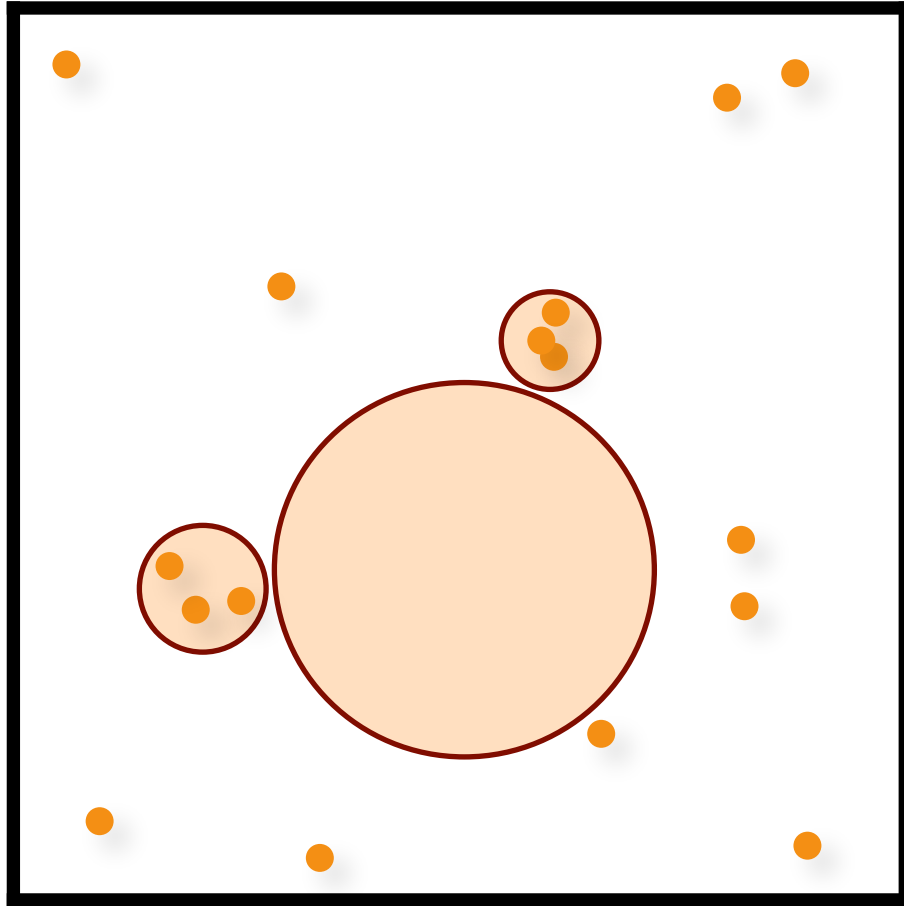
3 mesures de performance



Seulement trois mesures distinctes de la fonction...

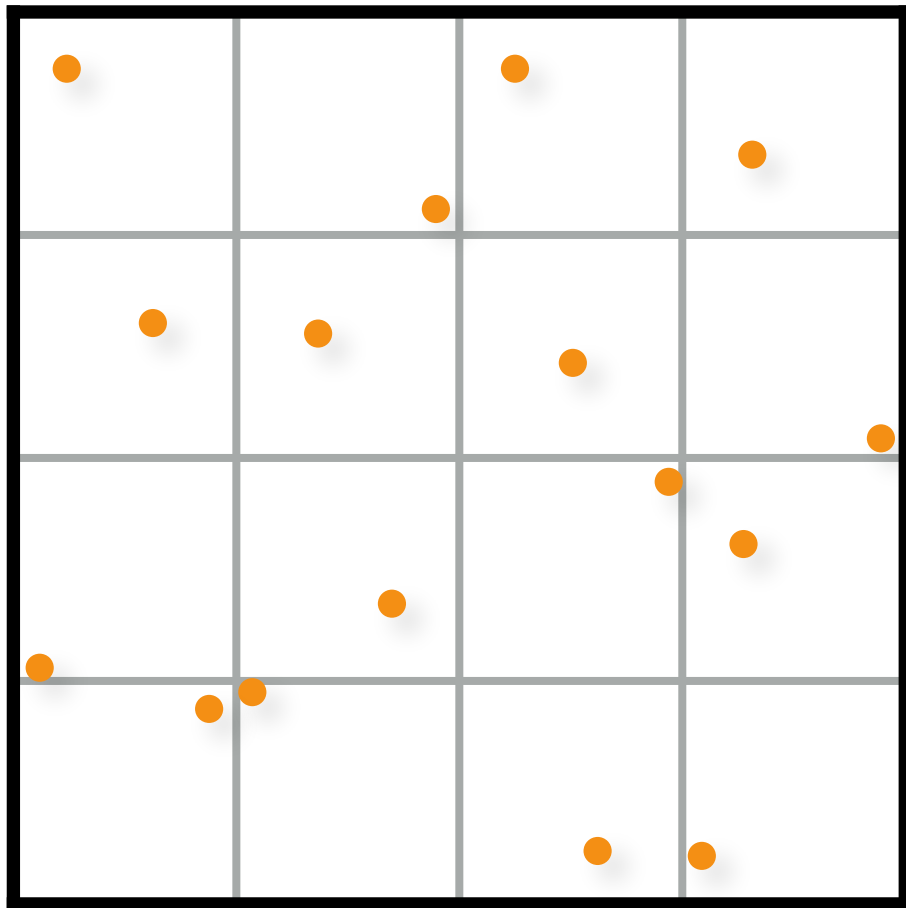
Les 12 mesures sont toutes distinctes!

# Échantillonnage aléatoire indépendant



Des trous et des points  
d'accumulation

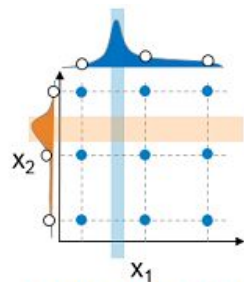
# échantillonnage saccadé (Jittered sampling)



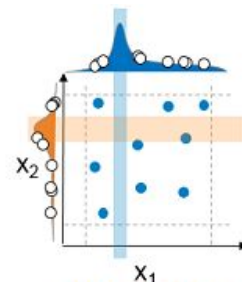
(un point par boîte:  
meilleure couverture de  
l'espace)

# Autres avantages des plans d'expériences aléatoires (1)

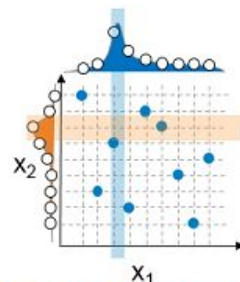
- Il est possible de mal échantillonner l'espace si la grille n'est pas optimale
- L'échantillonnage aléatoire évite ce problème.
- L'hypercube latin aléatoire répartit les points aléatoires plus uniformément.
- Intuition: positionner les tours qui ne se menacent pas sur un échiquier.



Standard Grid Search



Random Search



Random Latin Hypercube

Les mesures  
laissent supposer  
que la fonction ne  
dépend que de la  
variable bleue.

# Tableaux orthogonaux

---

Un tableau orthogonal de force  $t=2$  contient toutes les interactions par paires.

Chaque combinaison est testée. 4 variables avec 3 valeurs.

Question: Combien  
d'interactions par paires  
possibles ?

$$2^3 = 9$$

# Tableaux orthogonaux

Un tableau orthogonal de force  $t = 2$  contient toutes les interactions par paires.

Chaque combinaison est testée. 4 variables avec 3 valeurs.  $s^2 = 9$   
combinations possible

9

Question 2:  $\emptyset \heartsuit$   
Taille minimale et maximale d'un  
tableau orthogonal?

9 x 6



$\{0,0\},$   
 $\{0,1\},$   
 $\{0,2\},$   
 $\{1,0\},$   
 $\{1,1\},$   
 $\{1,2\},$   
 $\{2,0\},$   
 $\{2,1\},$   
 $\{2,2\}$



# Tableaux orthogonaux

Un tableau orthogonal de force  $t = 2$  contient toutes les interactions par paires.

Chaque combinaison est testée.

| runs:    |   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----------|---|---|---|---|---|---|---|---|---|---|
| facteurs |  | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
|          |  | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
|          |   |   |   |   |   |   |   |   |   |   |




$s^2 = 9$   
combinations  
possible

{0,0},  
{0,1},  
{0,2},  
{1,0},  
{1,1},  
{1,2},  
{2,0},  
{2,1},  
{2,2}

# Tableaux orthogonaux

Un tableau orthogonal de force  $t = 2$  contient toutes les interactions par paires.

Chaque combinaison est testée.

| runs:    |   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----------|---|---|---|---|---|---|---|---|---|---|
| facteurs |  | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
|          |  | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
|          |  | 0 | 1 | 2 | 1 | 2 | 0 | 2 | 0 | 1 |
|          |   |   |   |   |   |   |   |   |   |   |





$s^2 = 9$   
combinations  
possible

{0,0},  
{0,1},  
{0,2},  
{1,0},  
{1,1},  
{1,2},  
{2,0},  
{2,1},  
{2,2}

# Tableaux orthogonaux

Un tableau orthogonal de force  $t = 2$  contient toutes les interactions par paires.

Chaque combinaison est testée.

| runs:    |   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----------|---|---|---|---|---|---|---|---|---|---|
| facteurs |  | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
|          |  | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
|          |  | 0 | 1 | 2 | 1 | 2 | 0 | 2 | 0 | 1 |
|          |  | 0 | 2 | 1 | 1 | 0 | 2 | 2 | 1 | 0 |





$s^2 = 9$   
combinations  
possible

$\{0,0\},$   
 $\{0,1\},$   
 $\{0,2\},$   
 $\{1,0\},$   
 $\{1,1\},$   
 $\{1,2\},$   
 $\{2,0\},$   
 $\{2,1\},$   
 $\{2,2\}$

# Tableaux orthogonaux

Un tableau orthogonal de force  $t = 2$  contient toutes les interactions par paires.

Chaque combinaison est testée.

| runs:  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|--|---|---|---|---|---|---|---|---|---|
| facteurs  | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
|           | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
|           | 0 | 1 | 2 | 1 | 2 | 0 | 2 | 0 | 1 |
|           | 0 | 1 | 2 | 2 | 0 | 1 | 1 | 2 | 0 |

On a seulement besoin de  $s^t = 3^2 = 9$  expériences (avec  $s = 3$  niveaux pour la force  $t = 2$ )!

# Tableaux orthogonaux

Un tableau orthogonal de force  $t = 2$  contient toutes les interactions par paires.

Chaque combinaison est testée. 4 variables avec 3 valeurs.  $s^2 = 9$   
combinations possible

Question 3: (*pas nécessaire pour l'examen*)

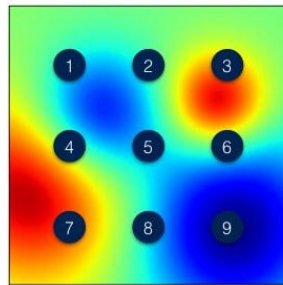
- Coder un algo pour vérifier si un tableau est orthogonal. (facile)
- Coder un algo qui crée un tableau orthogonal de force 2 avec  $d = s+1$ , et  $s$  un nombre premier (dur)

$\{0,0\},$   
 $\{0,1\},$   
 $\{0,2\},$   
 $\{1,0\},$   
 $\{1,1\},$   
 $\{1,2\},$   
 $\{2,0\},$   
 $\{2,1\},$   
 $\{2,2\}$

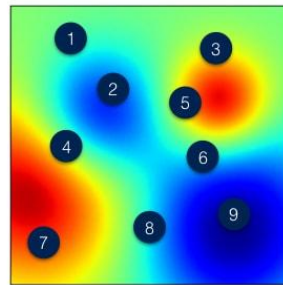
# Optimisation: sélection séquentielle en fonction des résultats précédents

+ on voit → +  
on est certain  
du maximum

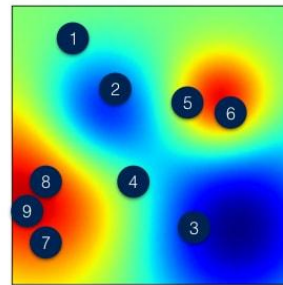
- L'échantillonnage bayésien apprend par essais et erreurs!
- Il modélise la fonction à partir des **échantillons précédents**.
- Il utilise ensuite la **position estimée du minimum** de la fonction comme **nouvel échantillon**.
- Problème: plus difficile à paralléliser.



Grid Search



Random Search



Adaptive Selection

Les points 7, 8 et 9  
convergent vers le  
minimum global.

# En pratique

## Utiliser Scipy:

### scipy.stats.qmc.LatinHypercube

```
class scipy.stats.qmc.LatinHypercube(d, *, centered=False, strength=1,
optimization=None, seed=None)
```

[\[source\]](#)

Latin hypercube sampling (LHS).

A Latin hypercube sample [1] generates  $n$  points in  $[0, 1]^d$ . Each univariate marginal distribution is stratified, placing exactly one point in  $[j/n, (j+1)/n)$  for  $j = 0, 1, \dots, n-1$ . They are still applicable when  $n \ll d$ .

Parameters:  $d$  : *int*

Dimension of the parameter space.

*centered* : *bool, optional*

Center the point within the multi-dimensional grid. Default is False.

*optimization* : *{None, "random-cd"}, optional*

Whether to use an optimization scheme to construct a LHS. Default is None.

- *random-cd*: random permutations of coordinates to lower the centered discrepancy [5]. The best design based on the centered discrepancy is constantly updated. Centered discrepancy-based design shows better space filling robustness toward 2D and 3D subprojections compared to using other discrepancy measures [6].

**New in version 1.8.0.**

*strength* : *{1, 2}, optional*

Strength of the LHS. *strength=1* produces a plain LHS while *strength=2* produces an orthogonal array based LHS of strength 2 [7], [8]. In that case, only  $n=p**2$  points can be sampled, with  $p$  a prime number. It also constrains  $d \leq p + 1$ . Default is 1.

**New in version 1.8.0.**

1.12.1+cu102



Search Tutorials

PyTorch Recipes [ + ]

Introduction to PyTorch [ - ]

Learn the Basics

Quickstart

Tensors

Datasets &amp; DataLoaders

Transforms

Build the Neural Network

Automatic Differentiation with `torch.autograd`

Optimizing Model Parameters

Save and Load the Model

Introduction to PyTorch on YouTube [ - ]

Introduction to PyTorch - YouTube Series

Introduction to PyTorch

Introduction to PyTorch Tensors

The Fundamentals of Autograd

Building Models with PyTorch

PyTorch TensorBoard Support

Training with PyTorch

Model Understanding with Captum

Tutorials &gt; Hyperparameter tuning with Ray Tune



Shortcuts



Run in Google Colab



Download Notebook



View on GitHub

# HYPERPARAMETER TUNING WITH RAY TUNE

Hyperparameter tuning can make the difference between an average model and a highly accurate one. Often simple things like choosing a different learning rate or changing a network layer size can have a dramatic impact on your model performance.

Fortunately, there are tools that help with finding the best combination of parameters. **Ray Tune** is an industry standard tool for distributed hyperparameter tuning. Ray Tune includes the latest hyperparameter search algorithms, integrates with TensorBoard and other analysis libraries, and natively supports distributed training through **Ray's distributed machine learning engine**.

In this tutorial, we will show you how to integrate Ray Tune into your PyTorch training workflow. We will extend **this tutorial from the PyTorch documentation** for training a CIFAR10 image classifier.

As you will see, we only need to add some slight modifications. In particular, we need to

1. wrap data loading and training in functions,
2. make some network parameters configurable,
3. add checkpointing (optional),
4. and define the search space for the model tuning

To run this tutorial, please make sure the following packages are installed:

- `ray[tune]`: Distributed hyperparameter tuning library
- `torchvision`: For the data transformers

Hyperparameter tuning with Ray Tune

[Setup / Imports](#)

Data loaders

Configurable neural network

+ The train function

Test set accuracy

Configuring the search space



# Conclusion

Avoir un plan d'expérience peut économiser beaucoup de **temps** et **d'énergie**.

Plusieurs approches:

- Grilles aléatoires qui couvrent bien l'espace (plan latin)
- Optimization (échantillonnage Bayésien, ou autre...)

Utiliser les librairies disponibles pour générer ces plans d'expérience pour vous!

# Sources

- IVADO MOOC
- <https://cs.dartmouth.edu/wjarosz/publications/jarosz19orthogonal.html>