

# ZOIDBERG 2.0

## BOOTSTRAP

1. Introduction au Machine Learning
2. Réaliser un projet de machine learning
3. Modélisation
4. Réduction de dimension
5. Les algorithmes de classification
6. Évaluation du modèle
7. Optimisation des hyperparamètres

# LES MÉTRIQUES D'ÉVALUATION

## 1. L'ACCURACY

$$\text{Accuracy} = \frac{\text{Toutes les prédictions correctes}}{\text{ensemble des données}} = \frac{TP+TN}{TOT}$$



Le taux de bonnes réponse ou accuracy est une mauvaise métrique lorsqu'il y a un grand déséquilibre de classes !

On pourrait croire que notre modèle est bon même lorsqu'il n'est bon que pour la classe majoritaire

# LES MÉTRIQUES D'ÉVALUATION

## 2. LA MATRICE DE CONFUSION

La matrice de confusion est une matrice qui permet de mesurer la qualité d'une prédiction en comparant les données réelles et les prédictions.





Prenons l'exemple d'un modèle qui prédit si des individus sont malades ou sains

Les prédictions justes :

- **Vrais positifs TP** : les personnes malades sont bien détectées comme malades
- **Vrais négatifs TN** : les personnes en bonne santé sont bien détectées comme étant en bonne santé

Les prédictions fausses :

- **Faux négatifs FN** : les personnes malades sont détectées comme étant en bonne santé
- **Faux positifs FP** : les personnes en bonne santé sont détectées comme étant malades

		 	
		Valeur prédite	
 	Valeur réelle	TN	FP
		FN	TP

# LES MÉTRIQUES D'ÉVALUATION

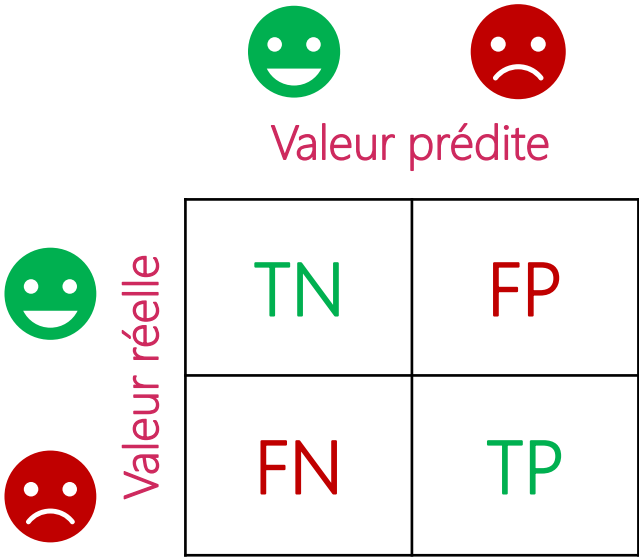
## 3. PRÉCISION & RAPPEL

**Recall (sensibilité)** =  $\frac{\text{vrai positif}}{\text{vrai positif} + \text{faux négatif}}$   $\frac{TP}{TP + FN}$

**Precision** =  $\frac{\text{vrai positif}}{\text{vrai positif} + \text{faux positif}}$   $\frac{TP}{TP + FP}$

**F1\_score** =  $\frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{rappel}}}$   $\frac{TP}{TP + \frac{1}{2}(FP + FN)}$   
moyenne harmonique de la précision et du rappel

**Spécificité** =  $\frac{\text{vrai négatif}}{\text{vrai négatif} + \text{faux positif}}$   $\frac{TN}{TN + FP}$

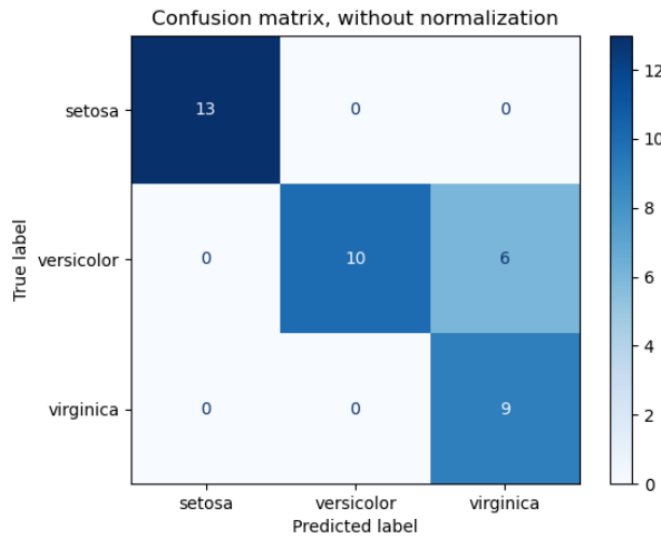


# LES MÉTRIQUES D'ÉVALUATION

## 4. IMPLÉMENTATION EN PYTHON

```
#import du module
```

```
from sklearn.metrics import confusion_matrix , accuracy_score, f1_score ....
```



Voir la doc : [https://scikit-learn.org/stable/modules/model\\_evaluation.html#classification-metrics](https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics)

# RAPPORT DE CLASSIFICATION

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
```

	precision	recall	f1-score	support	
class 0	0.50	1.00	0.67	1	Effectifs de chaque classe
class 1	0.00	0.00	0.00	1	
class 2	1.00	0.67	0.80	3	
accuracy			0.60	5	
macro avg	0.50	0.56	0.49	5	
weighted avg	0.70	0.60	0.61	5	



# COURBE ROC

## 1. DÉFINITION

Lorsque les classes sont très déséquilibrées, la matrice de confusion et surtout le taux d'erreur donnent une fausse idée de la qualité de l'apprentissage.

		Valeur prédite		Taux d'erreur?
		0	1	
Valeur réelle	0	95	0	
	1	5	0	

D'où l'utilisation d'une nouvelle métrique appelée la **courbe ROC** ou **courbe sensibilité/spécificité**, qui va pouvoir mesurer la performance d'un classifieur binaire **même si les classes sont disproportionnées** ou que l'on n'a pas d'échantillon **représentatif**. Elle donne le taux de vrais positifs en fonction du taux de faux positifs pour différents seuils de classification (ou seuil de coupure)

# COURBE ROC

## 1. DÉFINITION

Lorsque les classes sont très déséquilibrées, la matrice de confusion et surtout le taux d'erreur donnent une fausse idée de la qualité de l'apprentissage.

		Valeur prédite	
		0	1
Valeur réelle	0	95	0
	1	5	0

Taux d'erreur : 5% → très faible !

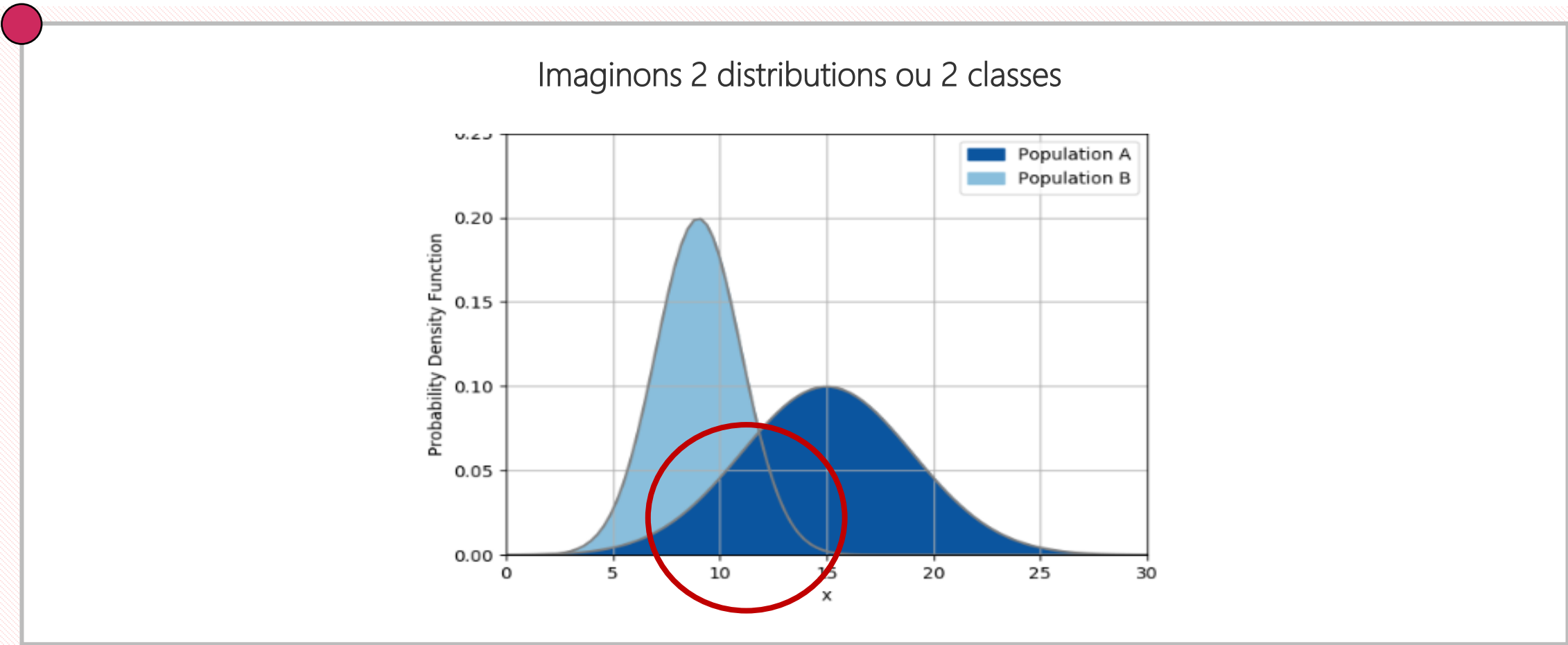
Pourtant aucun élément de la classe 1 n'a été détecté !

D'où l'utilisation d'une nouvelle métrique appelée la **courbe ROC** ou **courbe sensibilité/spécificité**, qui va pouvoir mesurer la performance d'un classifieur binaire **même si les classes sont disproportionnées** ou que l'on n'a pas d'échantillon **représentatif**. Elle donne le taux de vrais positifs en fonction du taux de faux positifs pour différents seuils de classification (ou seuil de coupure)



# COURBE ROC

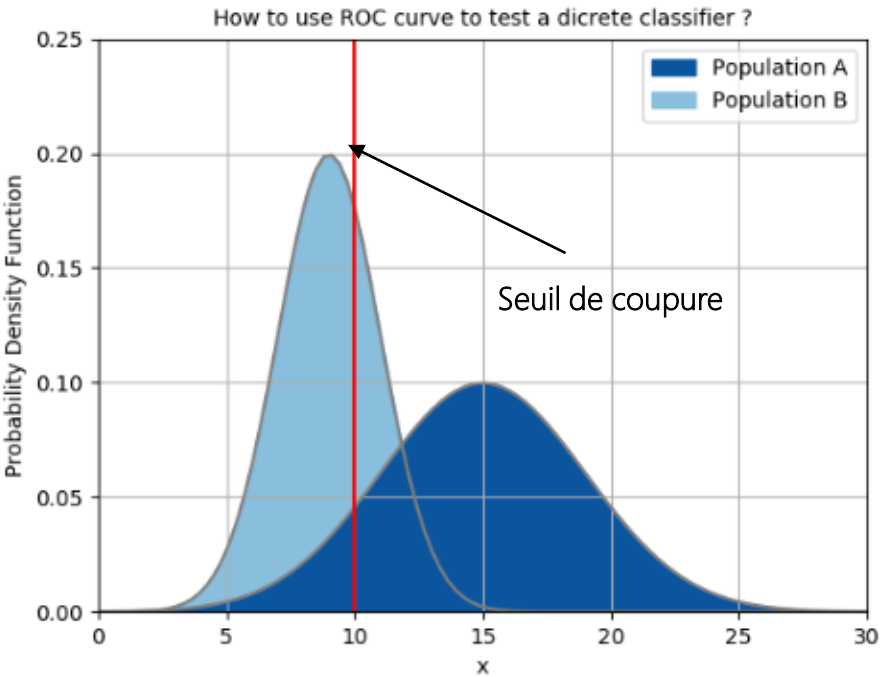
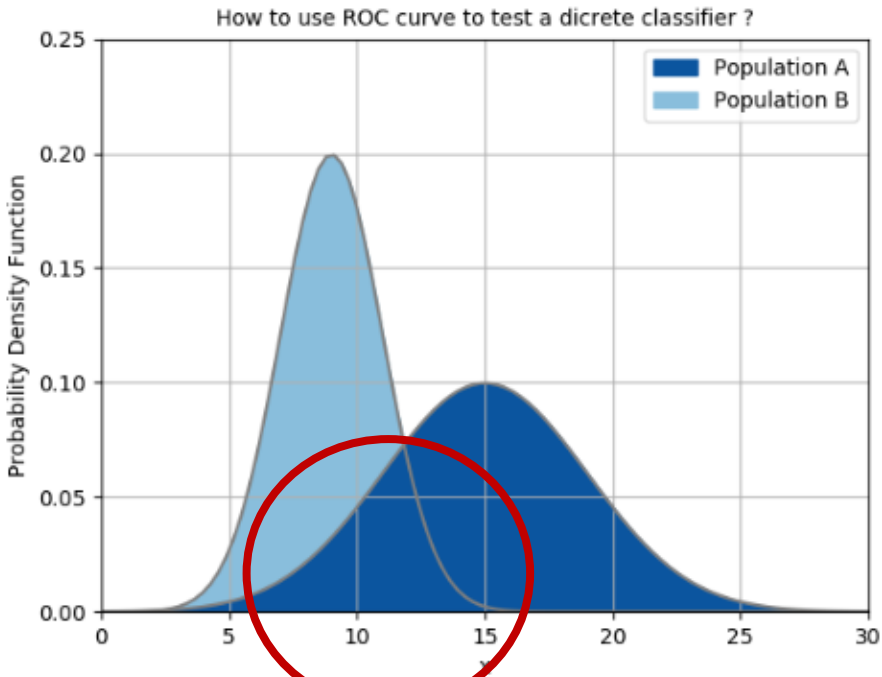
## 1. DÉFINITION



# COURBE ROC

## 1. DÉFINITION

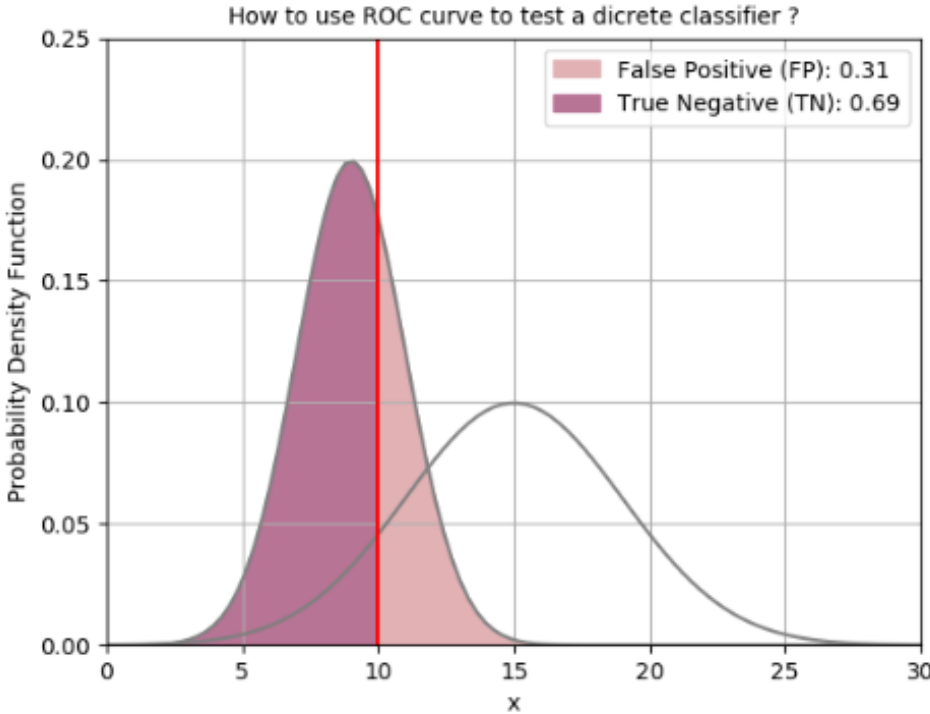
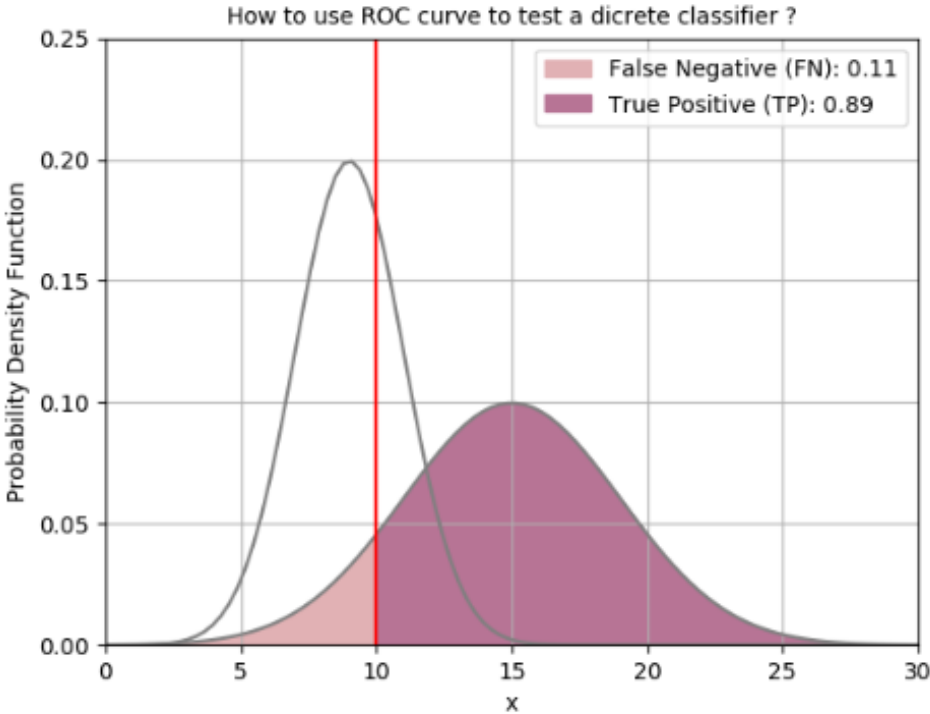
Plus l'aire commune aux deux distributions est petite, plus le modèle est confiant.



Courbe R.O.C pour tester la performance d'une classification discrète avec python

# COURBE ROC

## 1. DÉFINITION

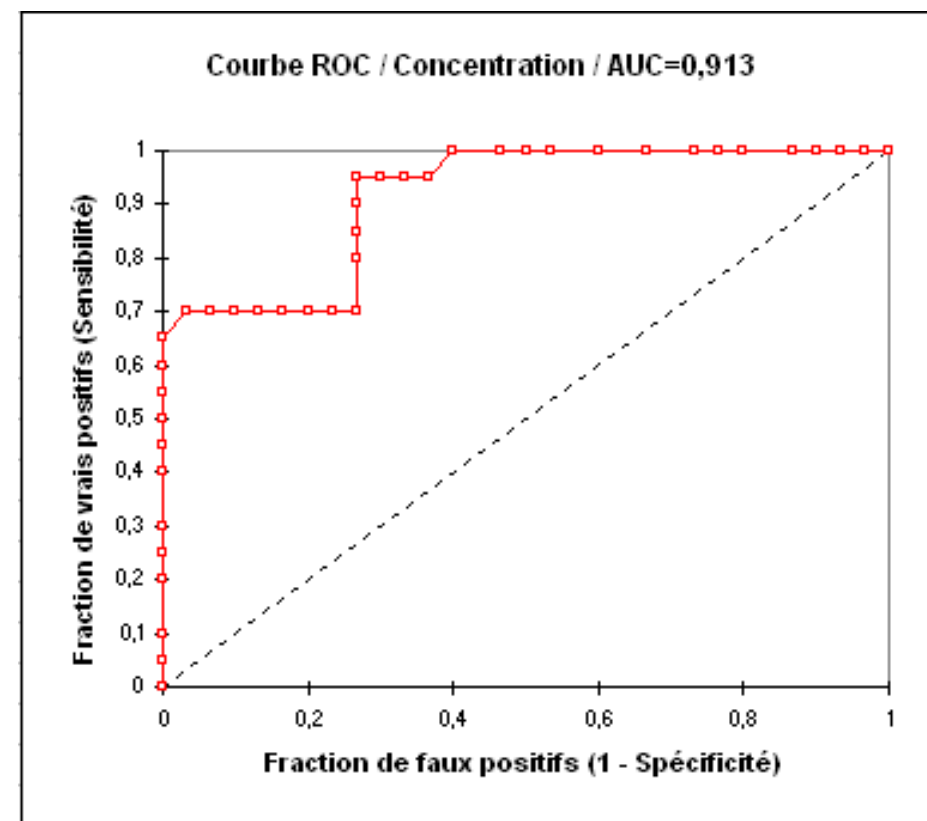


Courbe R.O.C pour tester la performance d'une classification discrète avec python

# COURBE ROC

## 2. CONSTRUCTION

1. On place le taux de vrais positifs / sensibilité / TPR en ordonnée
2. On place le taux de faux positifs / 1-spécificité / FPR en abscisse
3. On fait varier le seuil de coupure et on note le rapport de TPR en fonction de FPR

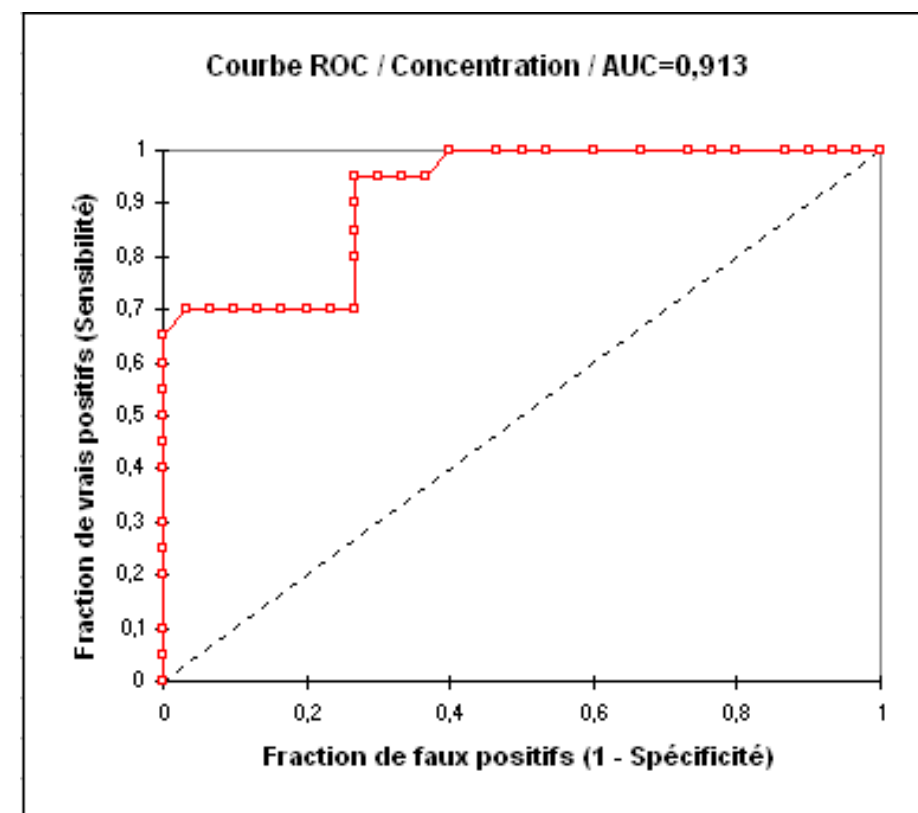


# COURBE ROC

## 3. INTERPRÉTATION : L'AIRE SOUS LA COURBE

La mesure **AUC** ou **Area Under the Curve** est l'aire sous la courbe. Elle est égale à la probabilité que le score d'un exemple bien classé soit inférieur à un exemple mal classé.

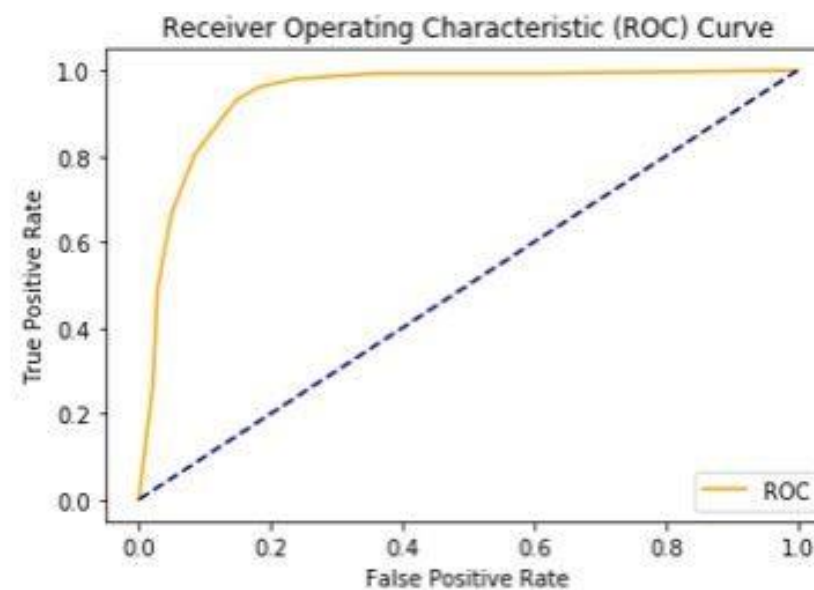
- En (0, 0) le classifieur classe tout en négatif
- En (1, 1) le classifieur classe tout positif
- Pour un classifieur aléatoire : il y a une chance sur 2 qu'il se trompe d'où la droite passant par (0, 0) à (1, 1)
- En (0, 1) le classificateur n'a aucun faux positif ni aucun faux négatif : c'est le modèle parfait



# COURBE ROC

## 4. IMPLÉMENTATION EN PYTHON

```
#import du module  
from sklearn.metrics import roc_curve  
  
# on calcule la courbe ROC  
...  
  
# on définit une fonction qui trace la courbe :  
  
...
```



Voir la doc : [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc\\_curve.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html)

# ZOIDBERG 2.0

## BOOTSTRAP

1. Introduction au Machine Learning
2. Réaliser un projet de machine learning
3. Modélisation
4. Réduction de dimension
5. Les algorithmes de classification
6. Évaluation du modèle
7. Optimisation des hyperparamètres

# PRINCIPE DE L'OPTIMISATION

## 1. CHOIX DU MODÈLE

Comment faire pour choisir l'algorithme et les hyperparamètres qui permettent de construire le modèle le plus adapté à mon problème ?

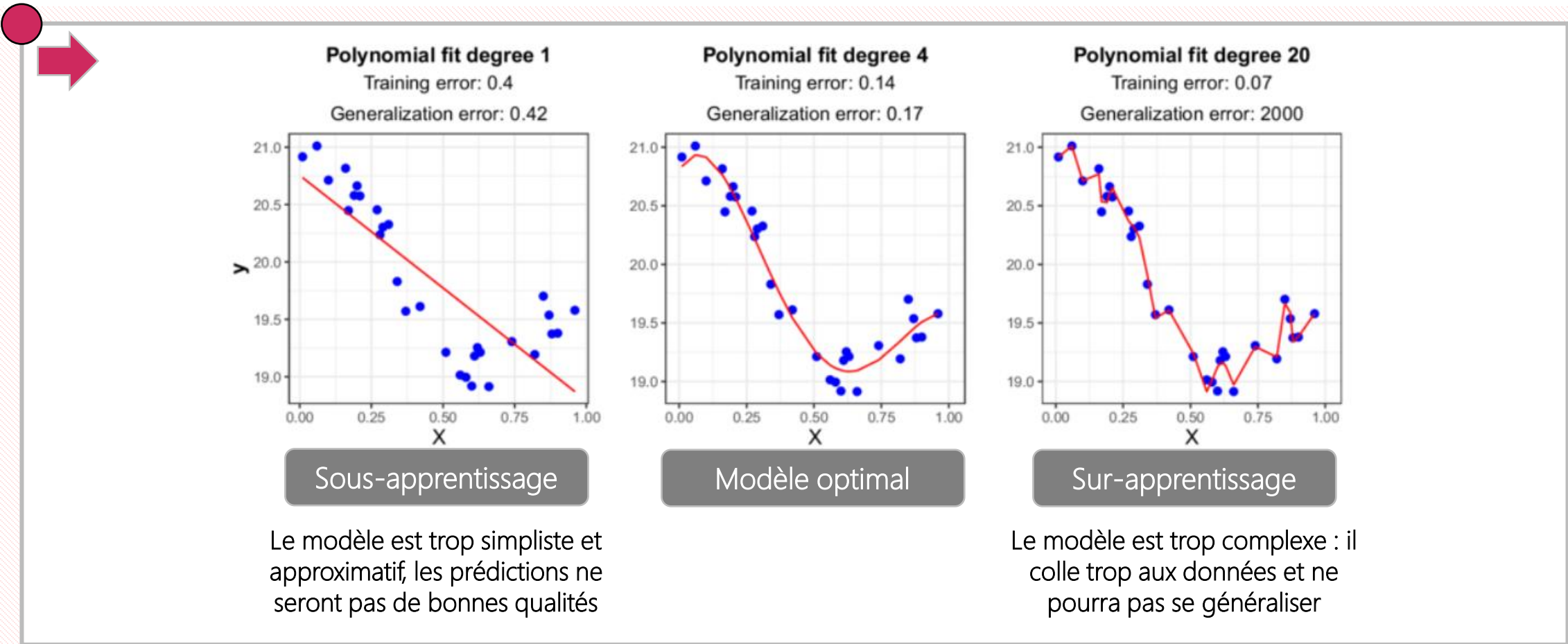
Ce que l'on veut :

- ➡ un modèle **qui modélise de façon précise** les phénomènes que l'on observe sur les données d'apprentissage
- ➡ mais qui puisse aussi **se généraliser aux données tests** et modéliser correctement les phénomènes que l'on souhaite observer !



# PRINCIPE DE L'OPTIMISATION

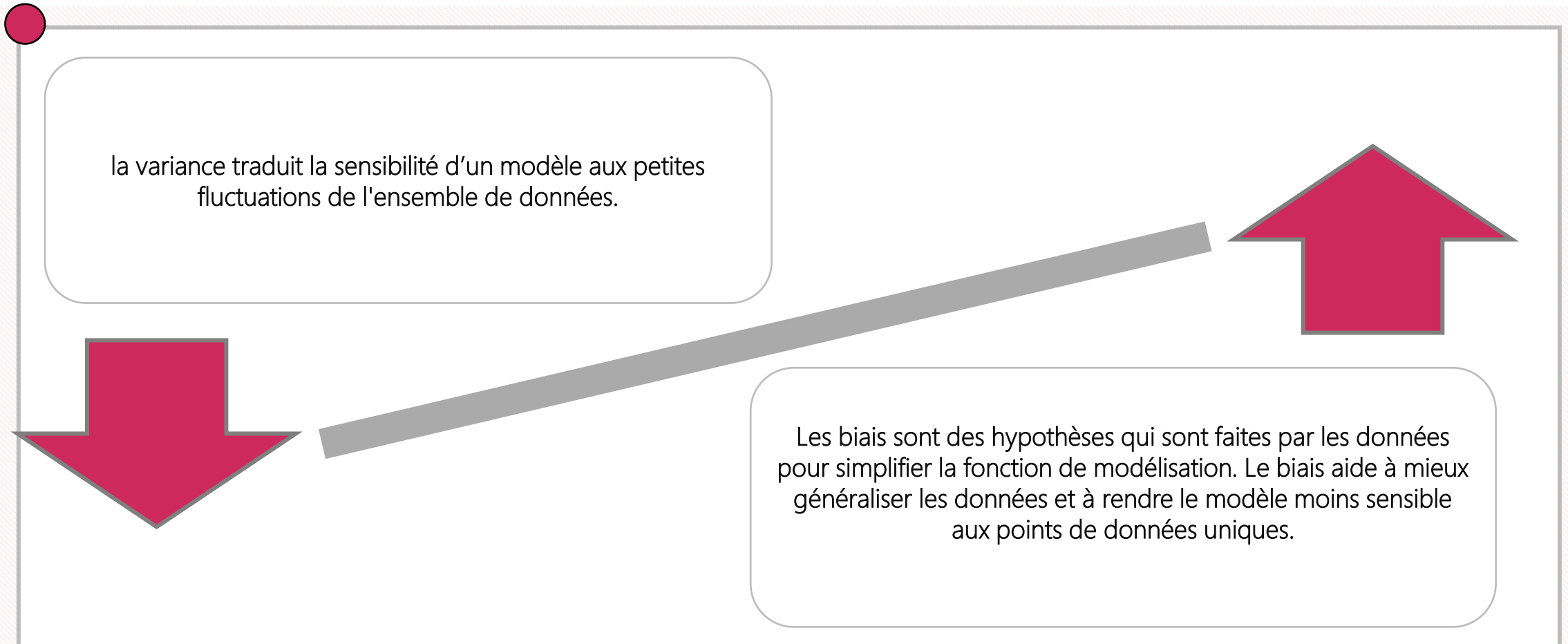
## 2. CAS DE FIGURE



[https://www.researchgate.net/figure/Illustration-of-the-underfitting-overfitting-issue-on-a-simple-regression-case-Data\\_fig2\\_339680577](https://www.researchgate.net/figure/Illustration-of-the-underfitting-overfitting-issue-on-a-simple-regression-case-Data_fig2_339680577)

# PRINCIPE DE L'OPTIMISATION

## 3. COMPROMIS BIAIS VARIANCE



# PRINCIPE DE L'OPTIMISATION

## 3. COMPROMIS BIAIS VARIANCE

la variance traduit la sensibilité d'un modèle aux petites fluctuations de l'ensemble de données.

Si la variance est élevée → l'algorithme prend en compte dans son modèle toutes les valeurs, même celles très éloignées et/ou aberrantes présentes dans le jeu de données ce qui entraîne de l'overfitting, le modèle ne pourra pas se généraliser.

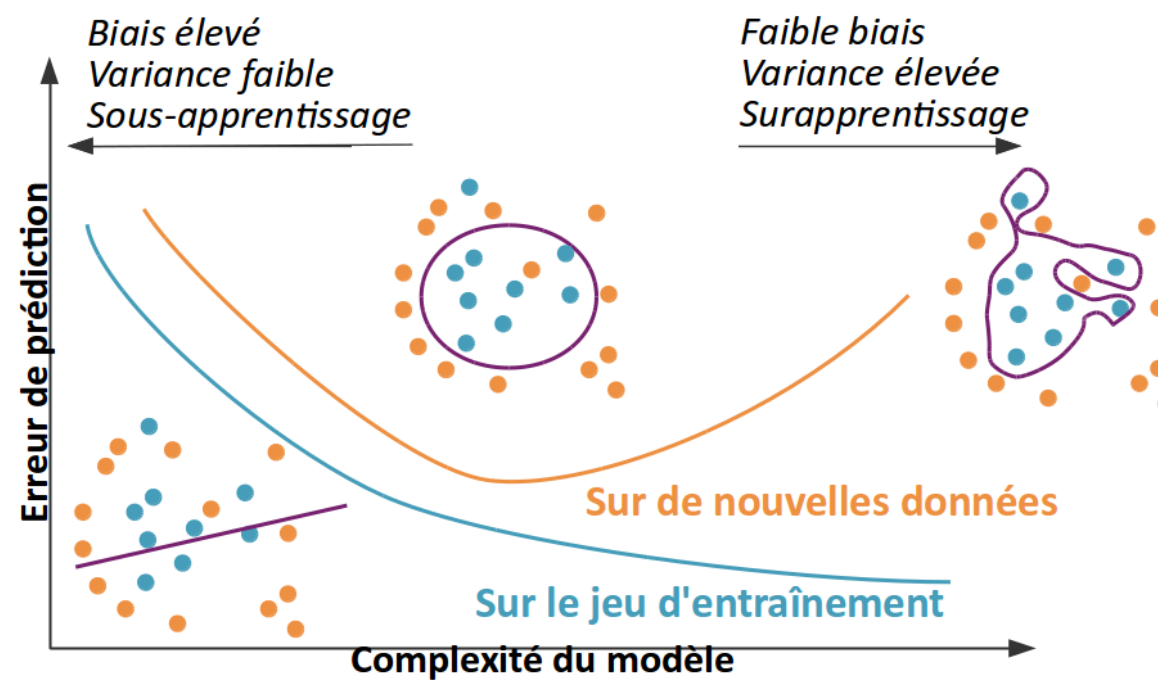
Les biais sont des hypothèses qui sont faites par les données pour simplifier la fonction de modélisation. Le biais aide à mieux généraliser les données et à rendre le modèle moins sensible aux points de données uniques.

Si le biais est élevé → l'algorithme fait trop d'hypothèses pour décomplexifier le modèle ce qui entraîne de l'underfitting, le modèle devient trop simpliste.

# PRINCIPE DE L'OPTIMISATION

## 3. COMPROMIS BIAIS VARIANCE

Il faut donc trouver un compromis entre le biais et la variance pour avoir un modèle ajusté



# PRINCIPE DE L'OPTIMISATION

## 4. LES SOLUTIONS

Il existe plusieurs solutions envisageables :

- On peut sélectionner les variables les plus pertinentes

# PRINCIPE DE L'OPTIMISATION

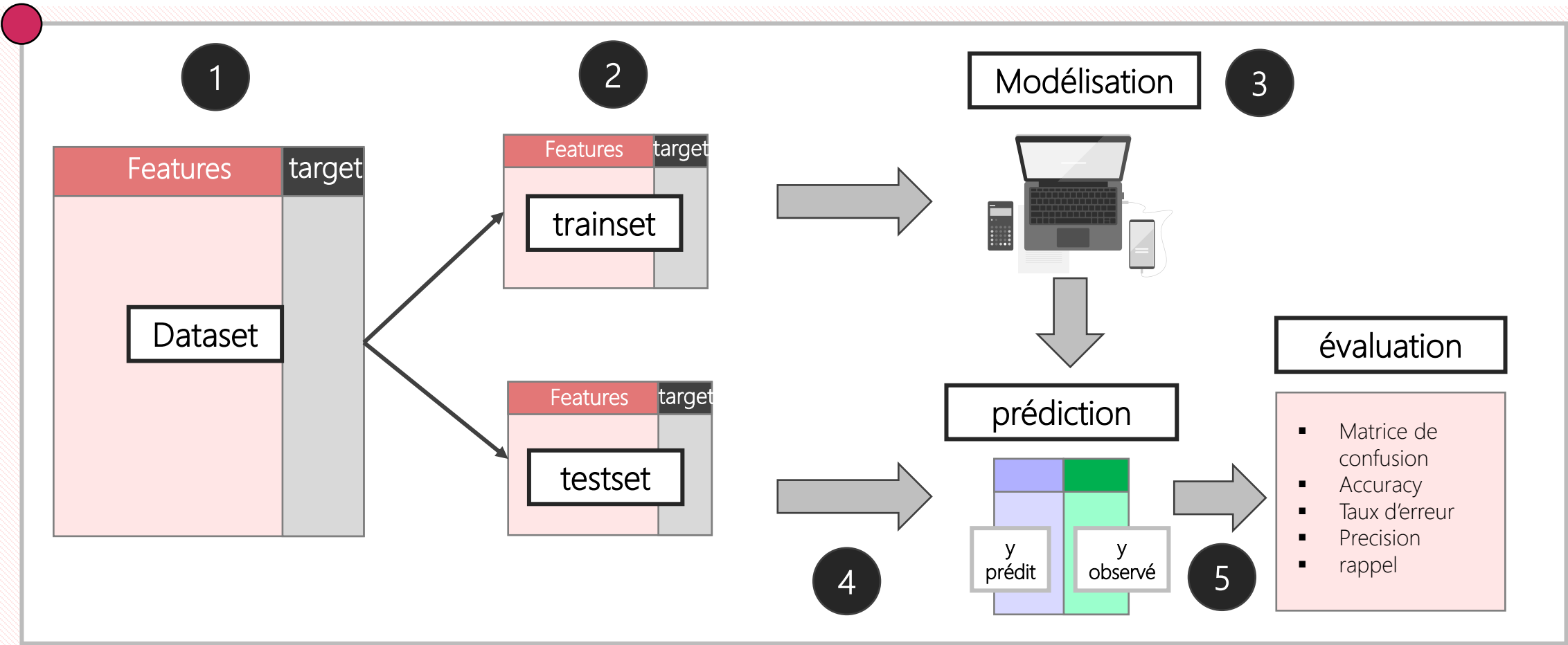
## 4. LES SOLUTIONS

Il existe plusieurs solutions envisageables :

- On peut sélectionner les variables les plus pertinentes
- On réalise une validation croisée
- 
-

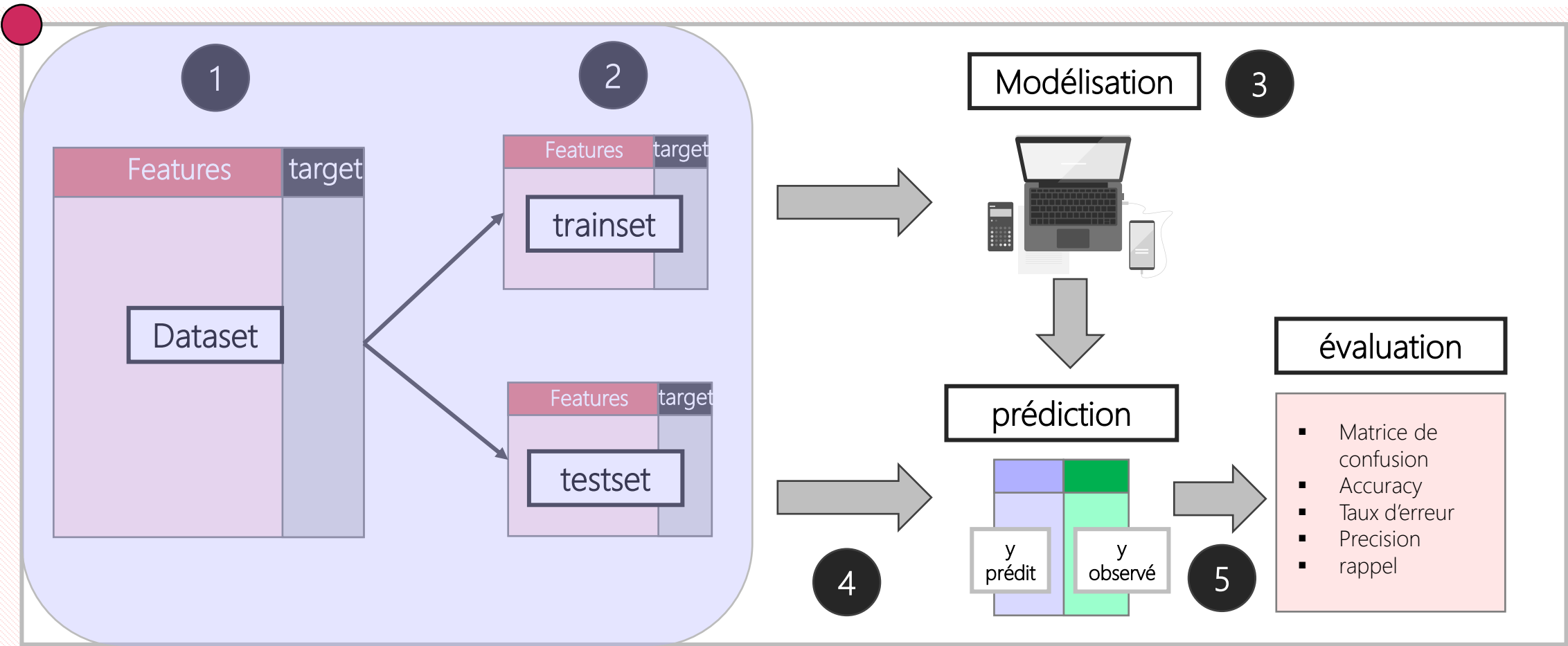
# CROSS VALIDATION

## 5. RAPPEL DES DIFFÉRENTES ÉTAPES D'UN PROCESSUS DE ML



# CROSS VALIDATION

## 5. RAPPEL DES DIFFÉRENTES ÉTAPES D'UN PROCESSUS DE ML





# CROSS VALIDATION

## 1. LE PRINCIPE

On sait qu'un modèle doit être évalué sur une base de test différente de celle utilisée pour l'apprentissage. Mais la performance est peut-être juste l'effet d'un hasard ou d'un découpage particulièrement avantageux.

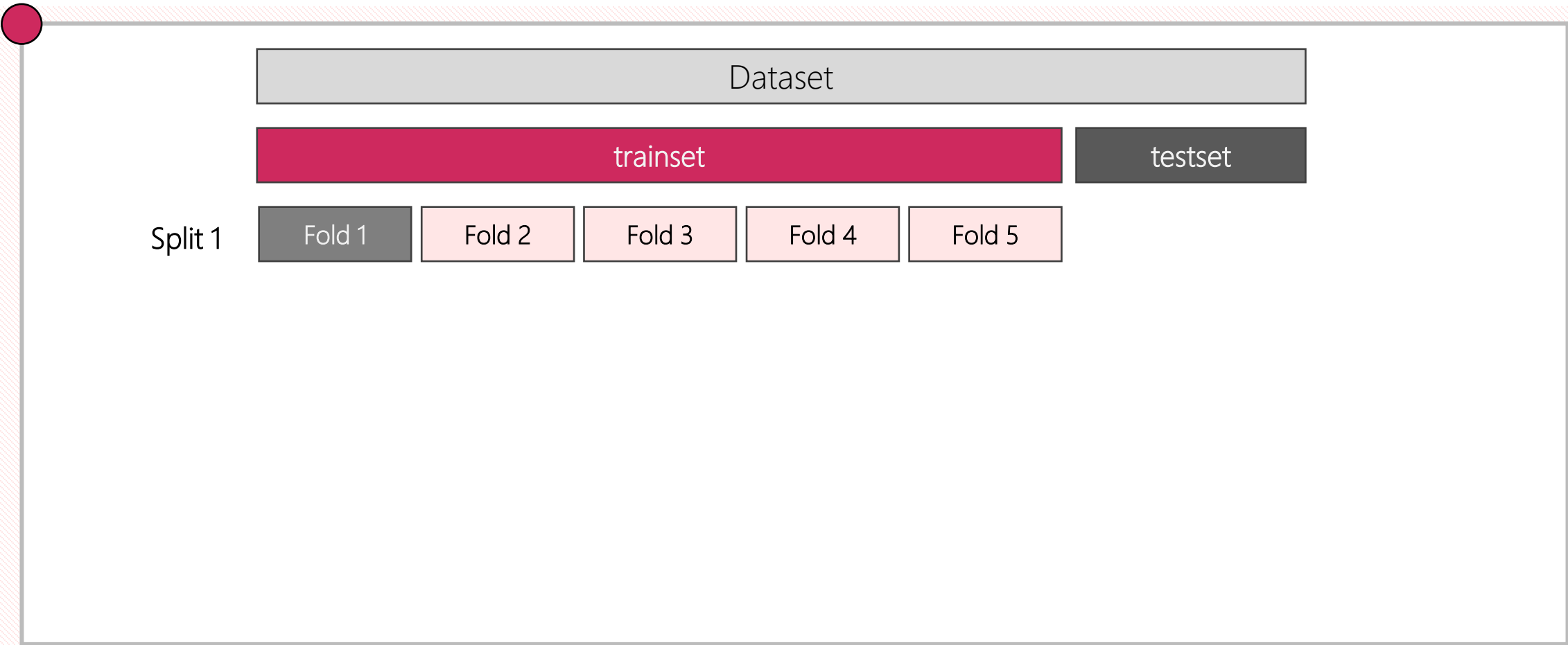
Pour être sûr que le modèle est robuste, on recommence l'opération plusieurs fois. C'est ce qu'on appelle cela la **validation croisée** ou **cross validation**

- Le **training set** est découpé en K fold ou K parties égales .
- K-1 fold servent à apprendre. Ils constitueront la base d'entraînement
- Le fold restant servira de test.
- On réitère l'opération K fois pour obtenir K scores.

Si le modèle est robuste nous devrions avoir sensiblement les même résultat de scores.

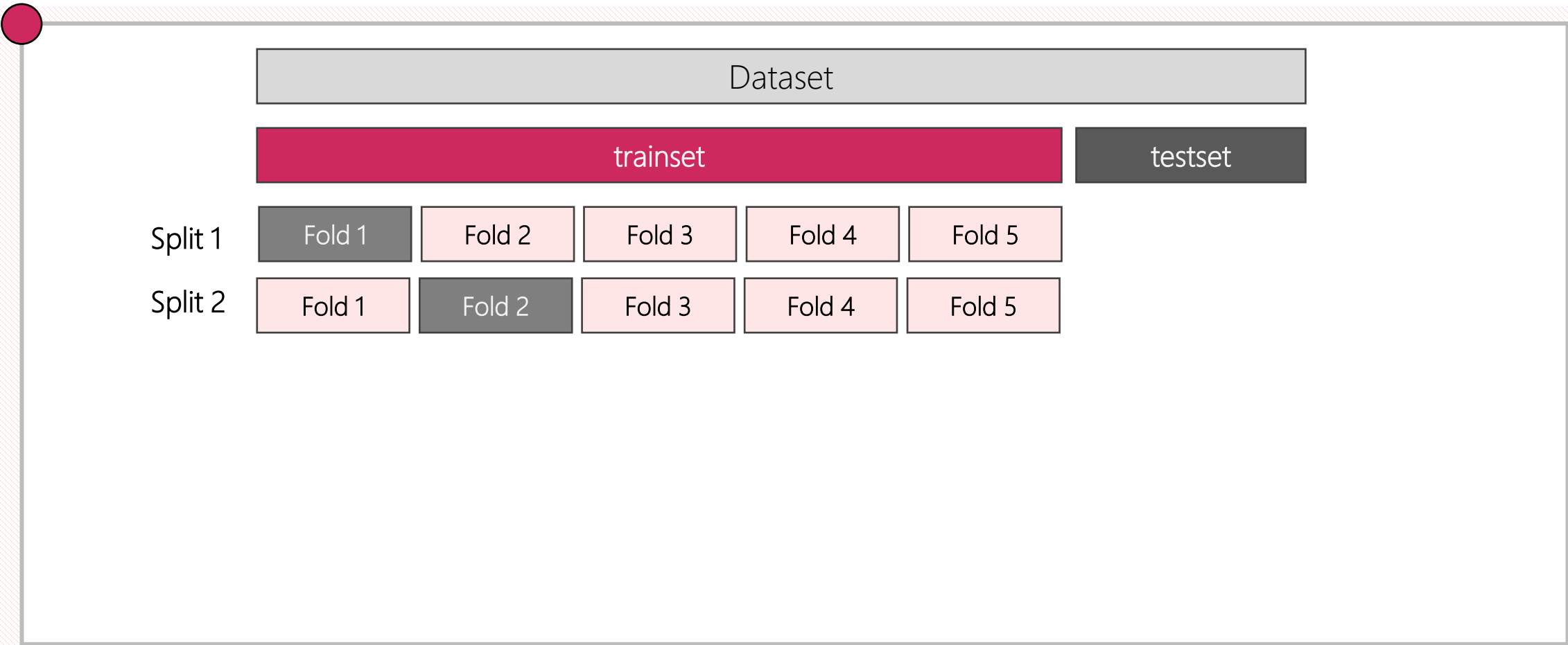
# CROSS VALIDATION

## 1. LE PRINCIPE



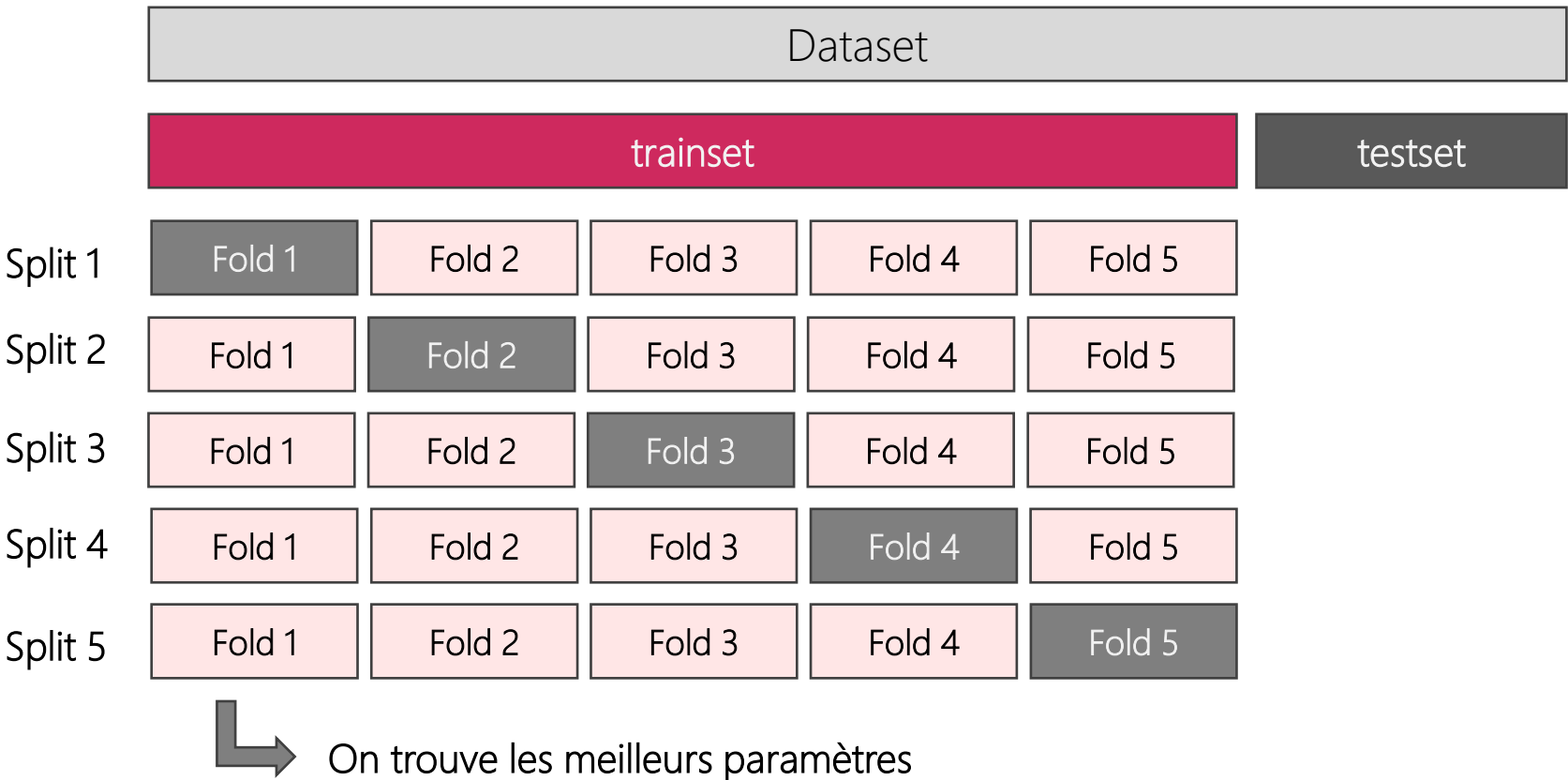
# CROSS VALIDATION

## 1. LE PRINCIPE



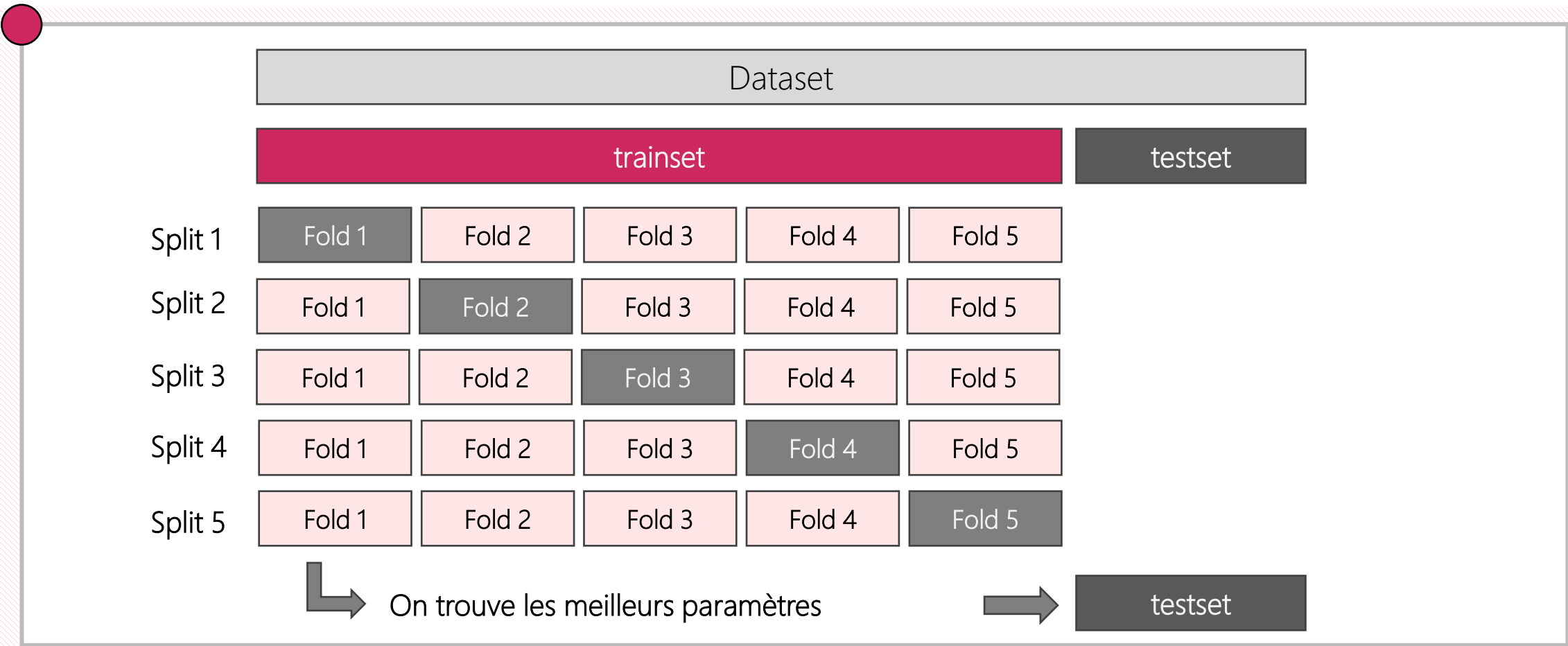
# CROSS VALIDATION

## 1. LE PRINCIPE



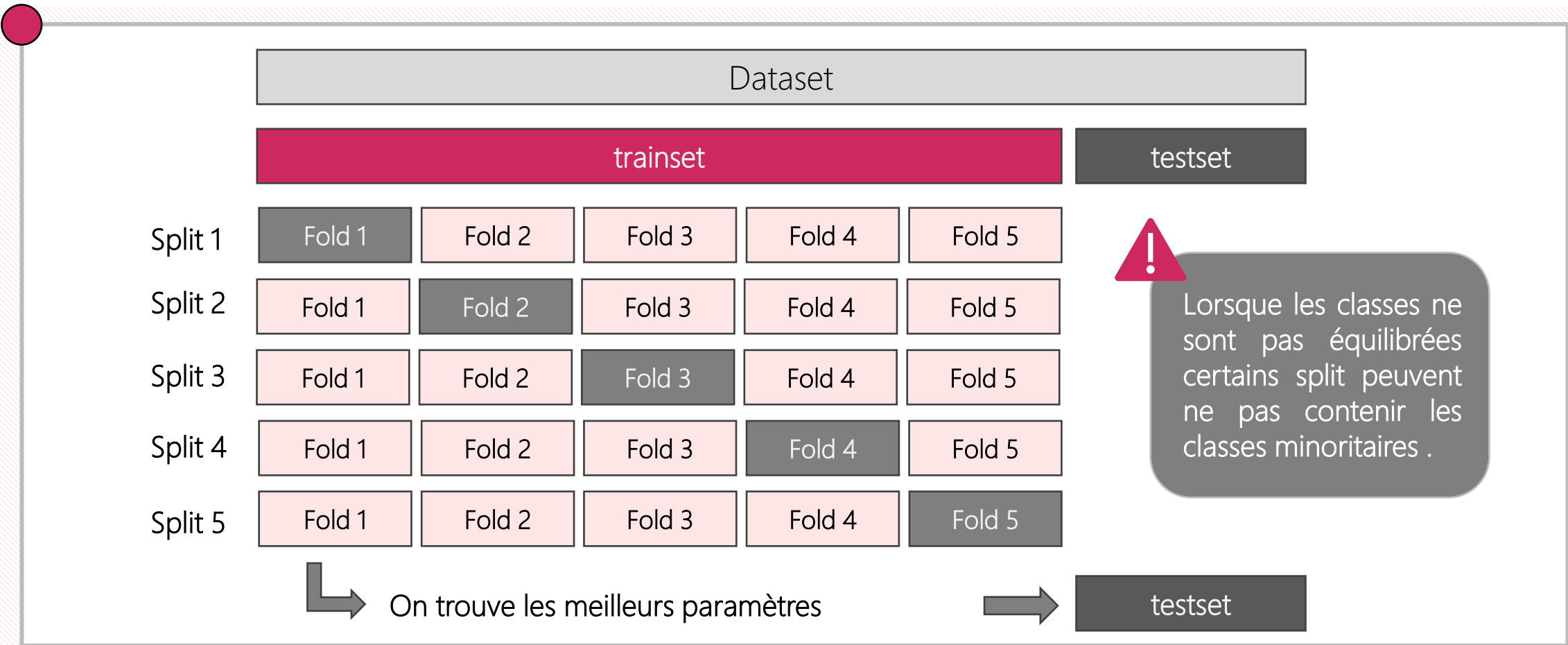
# CROSS VALIDATION

## 1. LE PRINCIPE



# CROSS VALIDATION

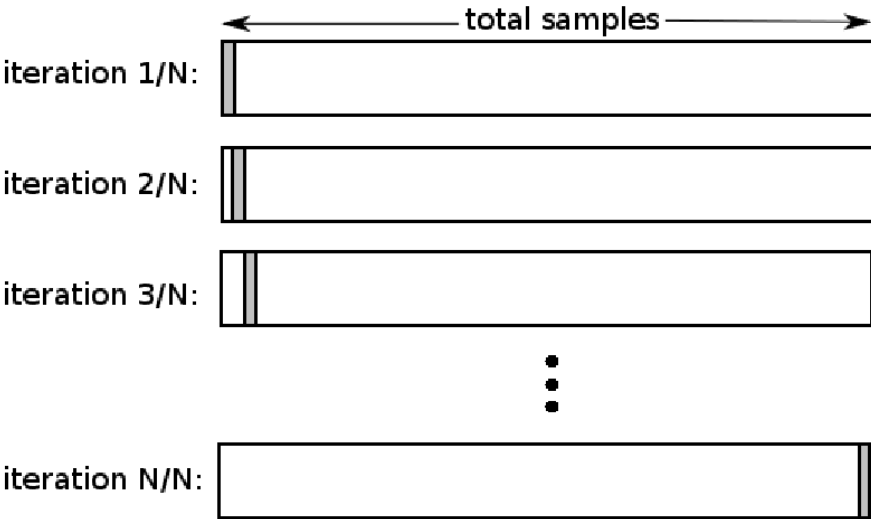
## 1. LE PRINCIPE



# CROSS VALIDATION

## 2. AUTRES TECHNIQUES

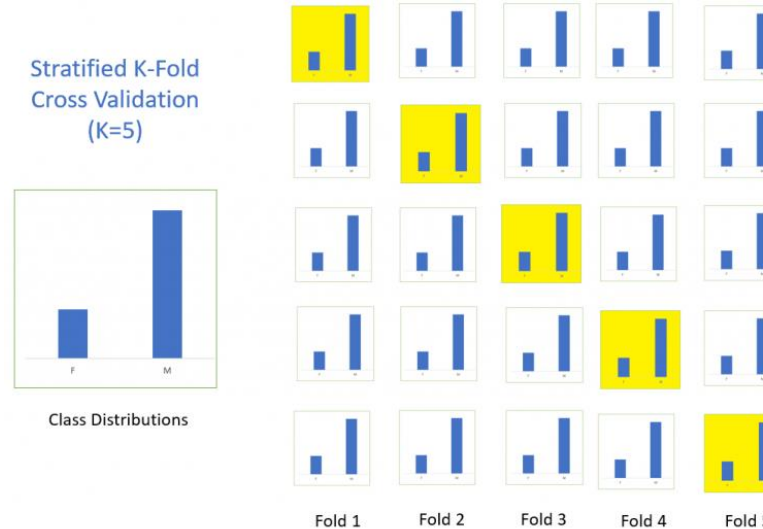
➤ **LEAVE ONE OUT CROSS-VALIDATION** : nous avons autant de fold que de données. Ex : Si notre dataset contient 100 échantillons dans le train, on aura  $K = 100$ , 99 serviront de train et 1 servira à évaluer le modèle. Puis on recommence l'opération 99 fois.



# CROSS VALIDATION

## 2. AUTRES TECHNIQUES

- **LEAVE ONE OUT CROSS-VALIDATION** : nous avons autant de fold que de données. Ex : Si notre dataset contient 100 échantillons dans le train, on aura  $K = 100$ , 99 serviront de train et 1 servira à évaluer le modèle. Puis on recommence l'opération 99 fois.
- **STRATIFIED K-FOLD** : même approche que le K-fold en gardant la proportion de chacune des classes dans chaque fold.





# CROSS VALIDATION

## 2. AUTRES TECHNIQUES

- **LEAVE ONE OUT CROSS-VALIDATION** : nous avons autant de fold que de données. Ex : Si notre dataset contient 100 échantillons dans le train, on aura  $K = 100$ , 99 serviront de train et 1 servira à évaluer le modèle. Puis on recommence l'opération 99 fois.
- **STRATIFIED K-FOLD** : même approche que le K-fold en gardant la proportion de chacune des classes dans chaque fold.
- **SHUFFLESPLIT CROSS-VALIDATION** : le dataset est mélangé et découpé en 2 : une partie de Train, et une partie de Test. C'est à l'utilisateur de décider la proportion de chaque échantillon. On réitère l'opération autant de fois que l'on souhaite . (même problématique que la méthode du K-fold pour des classes déséquilibrées.)

# CROSS VALIDATION

## 2. AUTRES TECHNIQUES

- **LEAVE ONE OUT CROSS-VALIDATION** : nous avons autant de fold que de données. Ex : Si notre dataset contient 100 échantillons dans le train, on aura  $K = 100$ , 99 serviront de train et 1 servira à évaluer le modèle. Puis on recommence l'opération 99 fois.
- **STRATIFIED K-FOLD** : même approche que le K-fold en gardant la proportion de chacune des classes dans chaque fold.
- **SHUFFLESPIT CROSS-VALIDATION** : le dataset est mélangé et découpé en 2 : une partie de Train, et une partie de Test. C'est à l'utilisateur de décider la proportion de chaque échantillon. On réitère l'opération autant de fois que l'on souhaite . (même problématique que la méthode du K-fold pour des classes déséquilibrées.)
- **GROUP K-FOLD** : On fait généralement l'hypothèse que les données sont indépendantes et tirées de la même distribution. Mais ce n'est pas toujours le cas. Par exemple, les données d'un Dataset médical **peuvent dépendre les unes des autres** : si des gens d'une même famille sont diagnostiqués d'un cancer, alors le facteur génétique crée une dépendance entre les différentes données. Il faut donc Découper le Dataset en **Groupe d'influence** : c'est ce que l'on fait avec GROUP K-FOLD.

Pour plus d'explications : <https://www.youtube.com/watch?v=VoyMOVfCSfc>

# CROSS VALIDATION

## 3. IMPLÉMENTATION AVEC PYTHON

### # import des modules

```
from sklearn.model_selection import cross_validate      # module pour appliquer une validation croisée
from sklearn.model_selection import Kfold, StratifiedKFold, ShuffleSplit, .... # choix du split
from sklearn import svm , RandomForest ....            # l'estimateur
```

### # implementation de la cross validation

```
cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0) # choix du split
cross_val_score (estimateur , X, y, cv=cv)                 # on applique la cross validation sur nos
données en ayant choisi notre estimateur
```

Pour en savoir plus : [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)

# PRINCIPE DE L'OPTIMISATION

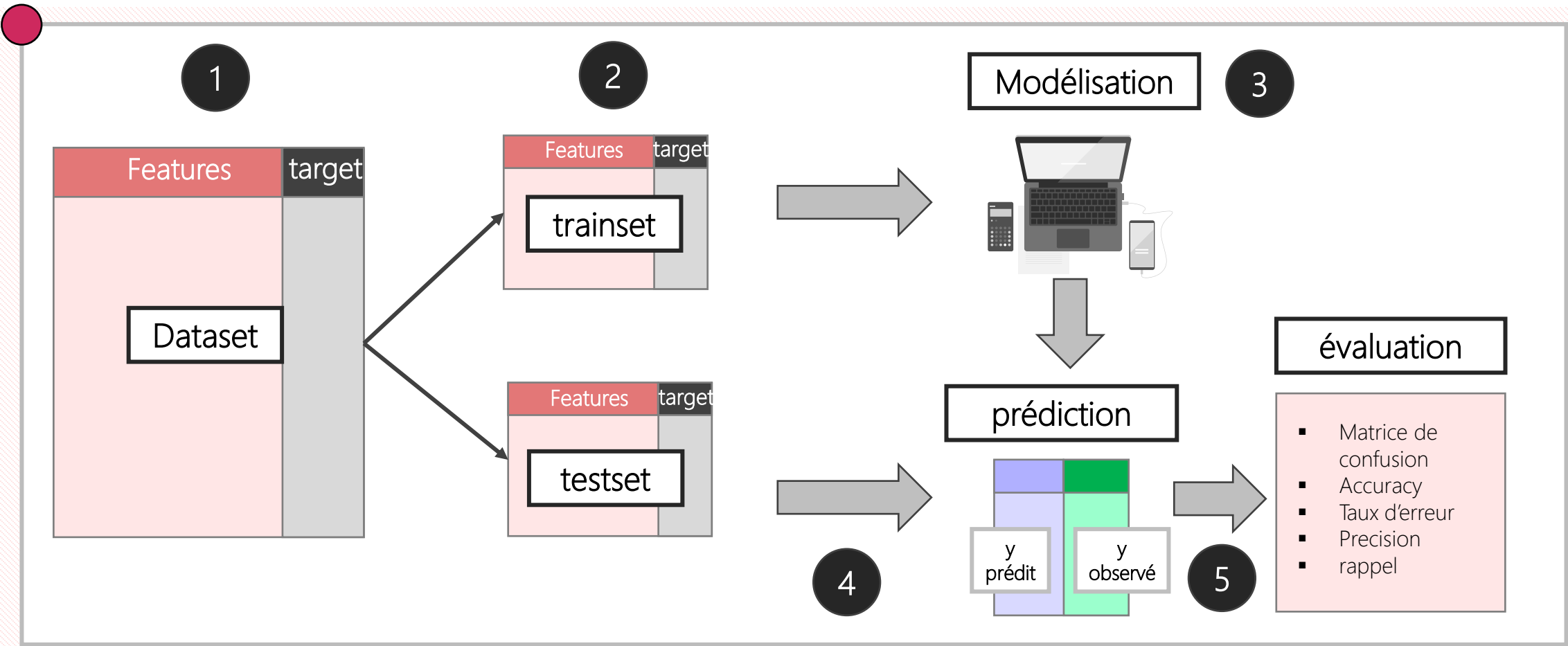
## RAPPEL OPTIMISATION DU MODÈLE : LES SOLUTIONS

Il existe plusieurs solutions envisageables :

- On peut sélectionner les variables les plus pertinentes
- On réalise une validation croisée
- On peut optimiser les hyperparamètres de notre modèle
-

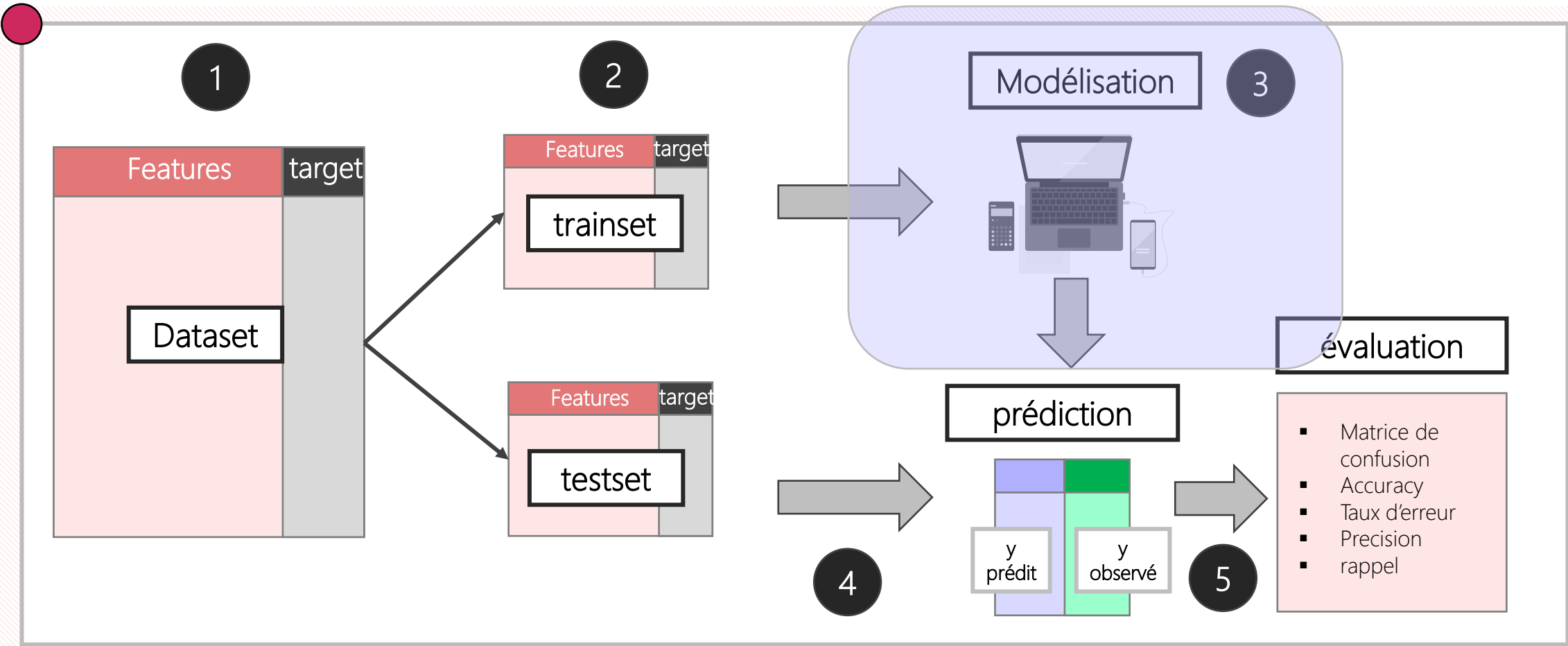
# PRINCIPE DE L'OPTIMISATION

RAPPEL DES DIFFÉRENTES ÉTAPES D'UN PROCESSUS DE ML



# OPTIMISATION DES HYPERPARAMÈTRES

RAPPEL DES DIFFÉRENTES ÉTAPES D'UN PROCESSUS DE ML



# OPTIMISATION DES HYPERPARAMÈTRES

GRIDSEARCHCV

Comment régler les hyperparamètres de mon algorithme ?

**Solution 1 :** On utilise les paramètres par défaut → non adapté à notre jeu de données

Nécessité de mettre en place une procédure la moins biaisée possible

On ne peut pas régler les hyperparamètres avec l'échantillon test, sinon il rentre dans le processus d'apprentissage !

# OPTIMISATION DES HYPERPARAMÈTRES

## 2. GRIDSEARCHCV

Comment régler les hyperparamètres de mon algorithme ?

**Solution 1 :** On utilise les paramètres par défaut → non adapté à notre jeu de données

Nécessité de mettre en place une procédure la moins biaisée possible

**On ne peut pas régler les hyperparamètres avec l'échantillon test, sinon il rentre dans le processus d'apprentissage !**

**Solution 2 :** On utilise un validation set qui servira de tuning set → non valable pour de petits dataset



# OPTIMISATION DES HYPERPARAMÈTRES

## 2. GRIDSEARCHCV

Comment régler les hyperparamètres de mon algorithme ?

**Solution 1 :** On utilise les paramètres par défaut → non adapté à notre jeu de données

Nécessité de mettre en place une procédure la moins biaisée possible

**On ne peut pas régler les hyperparamètres avec l'échantillon test, sinon il rentre dans le processus d'apprentissage !**

**Solution 2 :** On utilise un validation set qui servira de tuning set → non valable pour de petits dataset

**Solution 3 :** La solution on utilise une **GridsearchCV**

# OPTIMISATION DES HYPERPARAMÈTRES

## 2. GRIDSEARCHCV

Dans le processus de validation croisée on teste différentes combinaisons d'hyperparamètres .  
Tout paramètre fourni lors de la construction d'un estimateur peut être optimisé de cette manière.

# import du module

```
from sklearn.model_selection import GridSearchCV
```

#on définit une liste d'hyperparamètres sur lesquels on veut jouer

```
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
```

# on applique la méthode

```
GridSearchCV (estimateur, param_grid = parameters, scoring=métrieque_choisie, cv=nbr_de_fold)
```

# quelques méthodes utiles

best\_params\_ = renvoie la meilleure combinaison de paramètres trouvés

best\_score\_ = renvoie le meilleur score obtenu

best\_estimator\_ = renvoie le meilleur estimateur trouvé

cv\_results\_ = affiche les résultats pour toutes les combinaisons réalisées

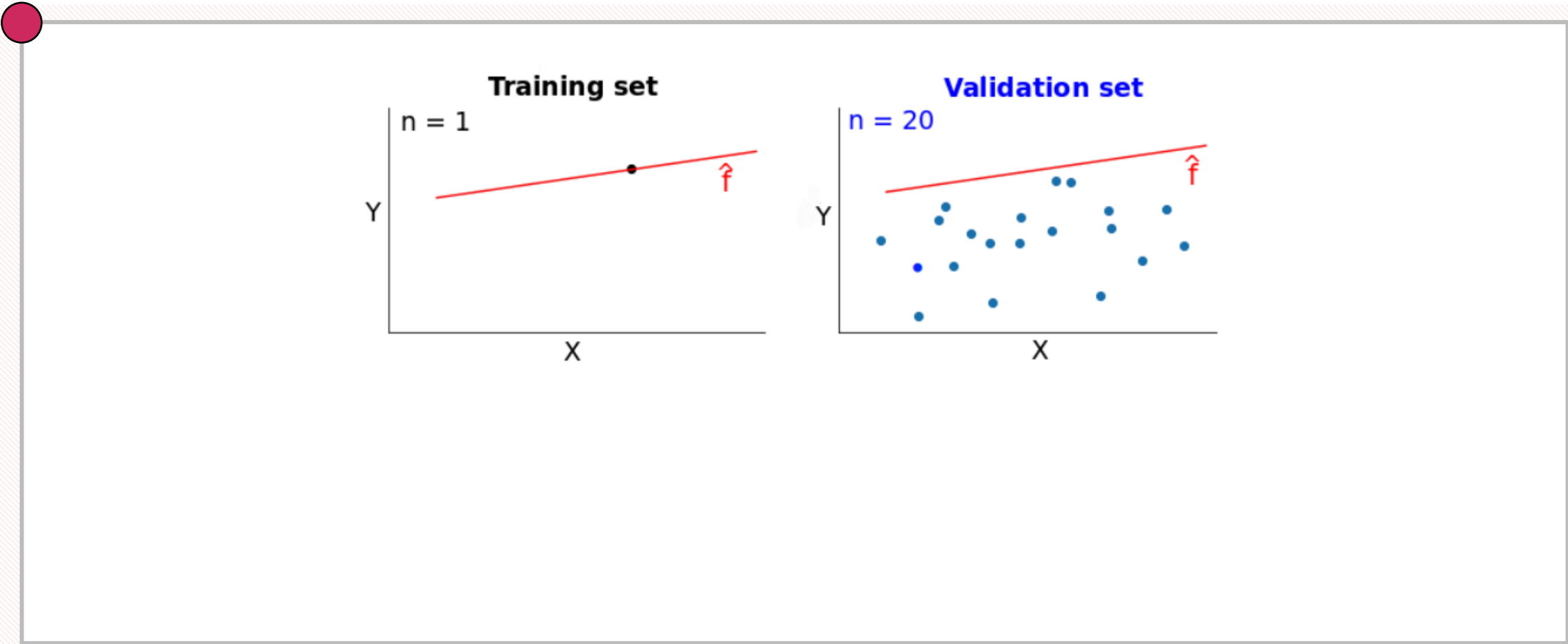
# LEARNING CURVE

## 1. DÉFINITION

Une **courbe d'apprentissage** ou **learning curve** est une courbe qui montre à la fois la performance du **train set** et celle du **validation set** d'un estimateur pour un nombre variable de données dans le jeu d'entraînement . Elle permet de savoir **dans quelle mesure l'ajout de plus de données d'entraînement est bénéfique pour le modèle** et si l'estimateur souffre davantage d'une erreur de variance ou d'une erreur de biais.

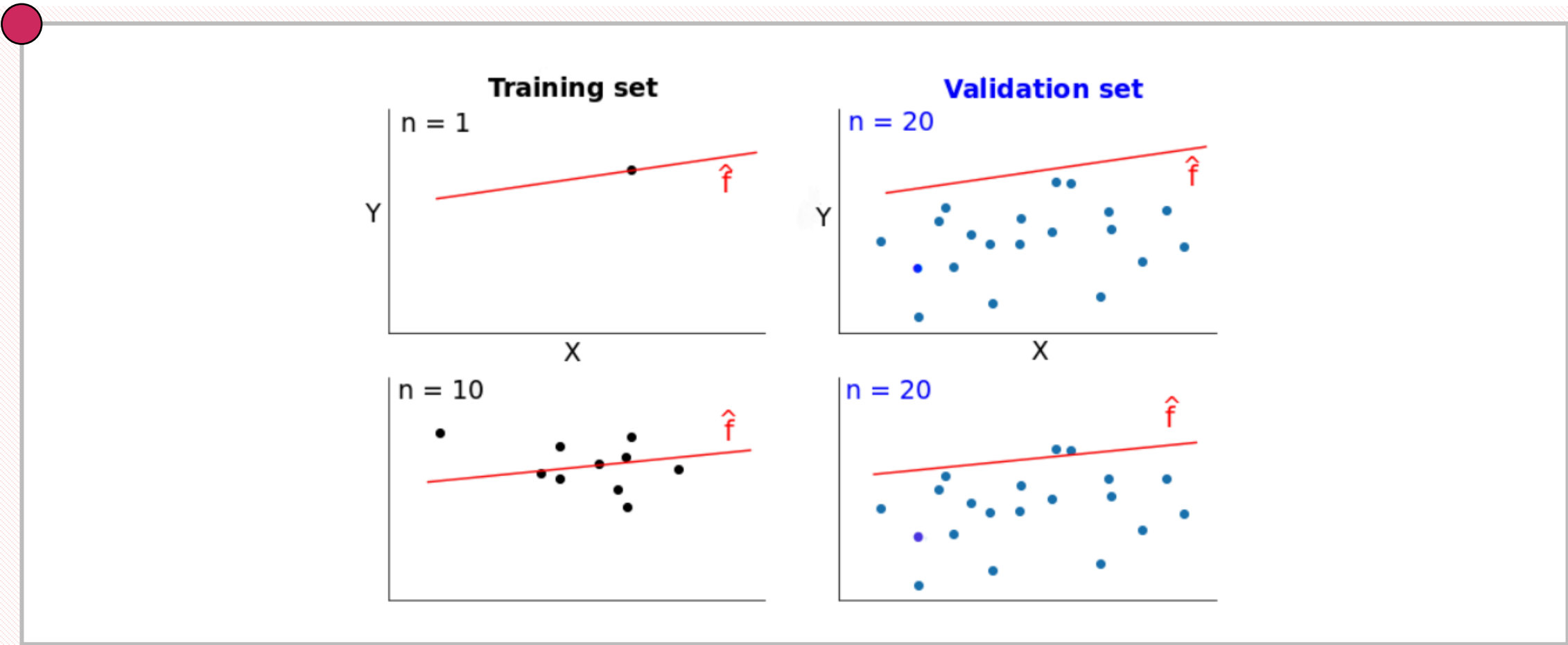
# LEARNING CURVE

## 2. LE PRINCIPE



# LEARNING CURVE

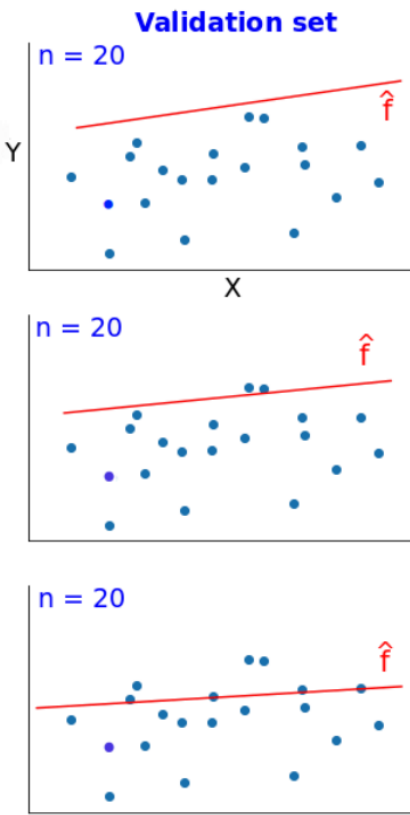
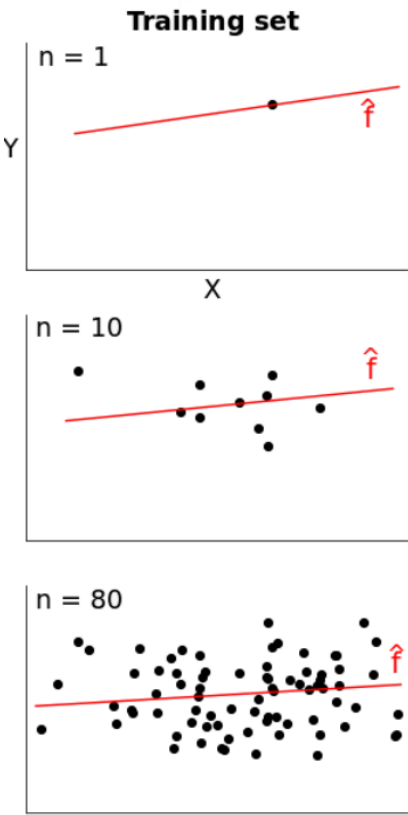
## 2. LE PRINCIPE



# LEARNING CURVE

## 2. LE PRINCIPE

Plus on ajoute des données à l'entraînement plus il y a d'erreur en terme de performance du training set

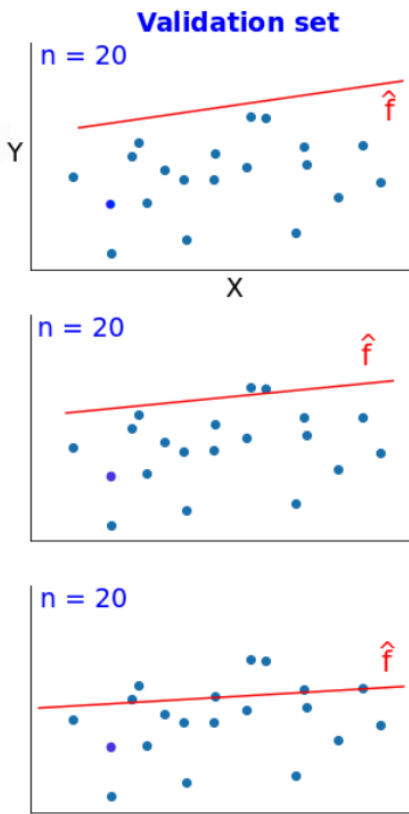
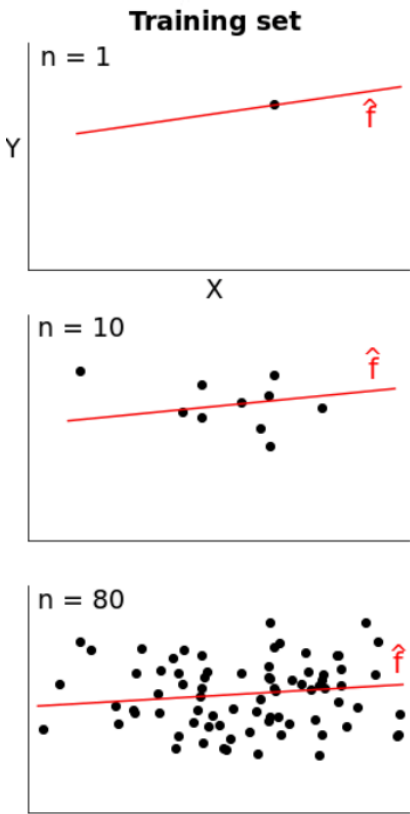
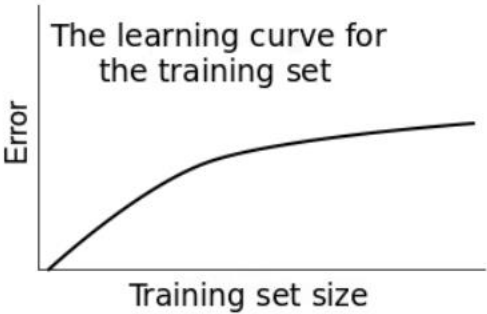


Plus on ajoute des données à l'entraînement moins il y a d'erreur au niveau du validation set !

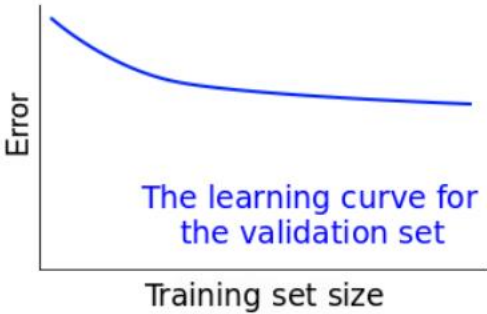
# LEARNING CURVE

## 2. LE PRINCIPE

Plus on ajoute des données à l'entraînement plus il y a d'erreur en terme de performance du training set

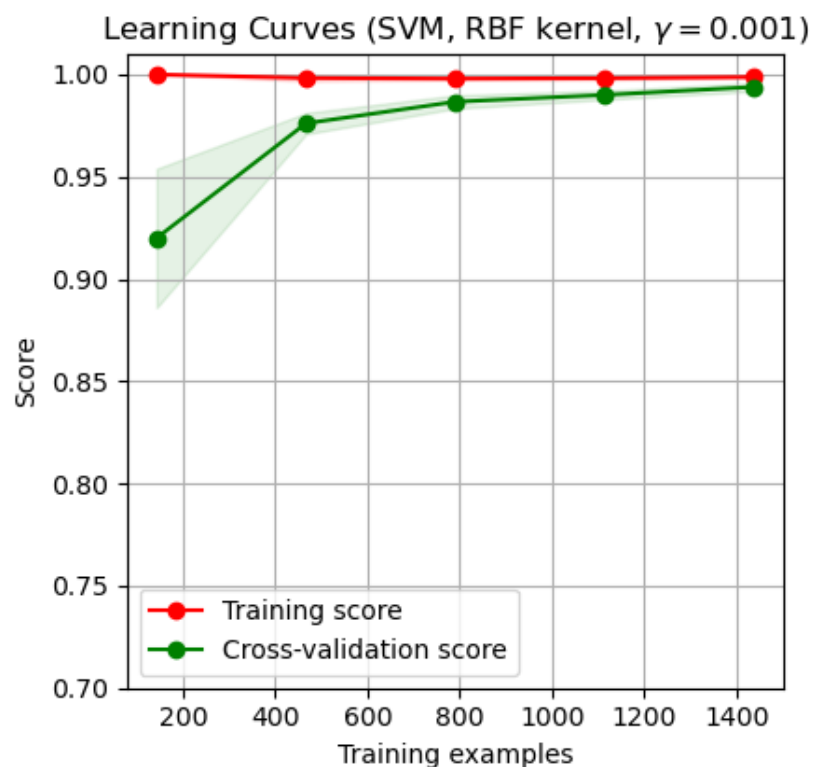


Plus on ajoute des données à l'entraînement moins il y a d'erreur au niveau du validation set !



# LEARNING CURVE

## 3. UN EXEMPLE



Dans cet exemple , l'estimateur est l'algorithme SVM.  
pour de petites quantités de données, le score d'entraînement de la SVM est beaucoup plus élevé que le score de validation. Et on remarque que plus on ajoute des données , plus le score de la cross validation augmente et converge vers celui du trainset . l'ajout de données d'entraînement bénéficie à la généralisation du modèle.



# LEARNING CURVE

## 4. IMPLÉMENTATION EN PYTHON

#import du module :

```
from sklearn.model_selection import learning_curve
```

# la méthode :

```
train_sizes, train_scores, test_scores = learning_curve(estimateur , X, y, cv = 10, scoring='métrique',  
train_sizes=np.linspace(intervalles choisis ))
```

Un exemple d'implémentation :

[Validation Curves Explained - Python Sklearn Example - Data Analytics \(vitalflux.com\)](https://vitalflux.com/validation-curves-explained-python-sklearn-example-data-analytics/)

voir la doc associée :

[Plotting Learning Curves — scikit-learn 1.1.1 documentation](https://scikit-learn.org/stable/tutorial/model_selection/plotting_learning_curves.html)

# ENSEMBLE LEARNING

## RAPPEL OPTIMISATION DU MODÈLE : LES SOLUTIONS

Il existe plusieurs solutions envisageables :

- On peut sélectionner les variables les plus pertinentes
- On peut réaliser une validation croisée
- On peut optimiser les hyperparamètres de notre modèle
- On peut utiliser des méthodes ensemblistes

# ENSEMBLE LEARNING

## 1. LE PRINCIPE

Il s'agit d'entraîner plusieurs modèles et de considérer l'ensemble de leurs prédictions.

**L'objectif** : atteindre de meilleures performances grâce à la combinaison de chacun des modèles.

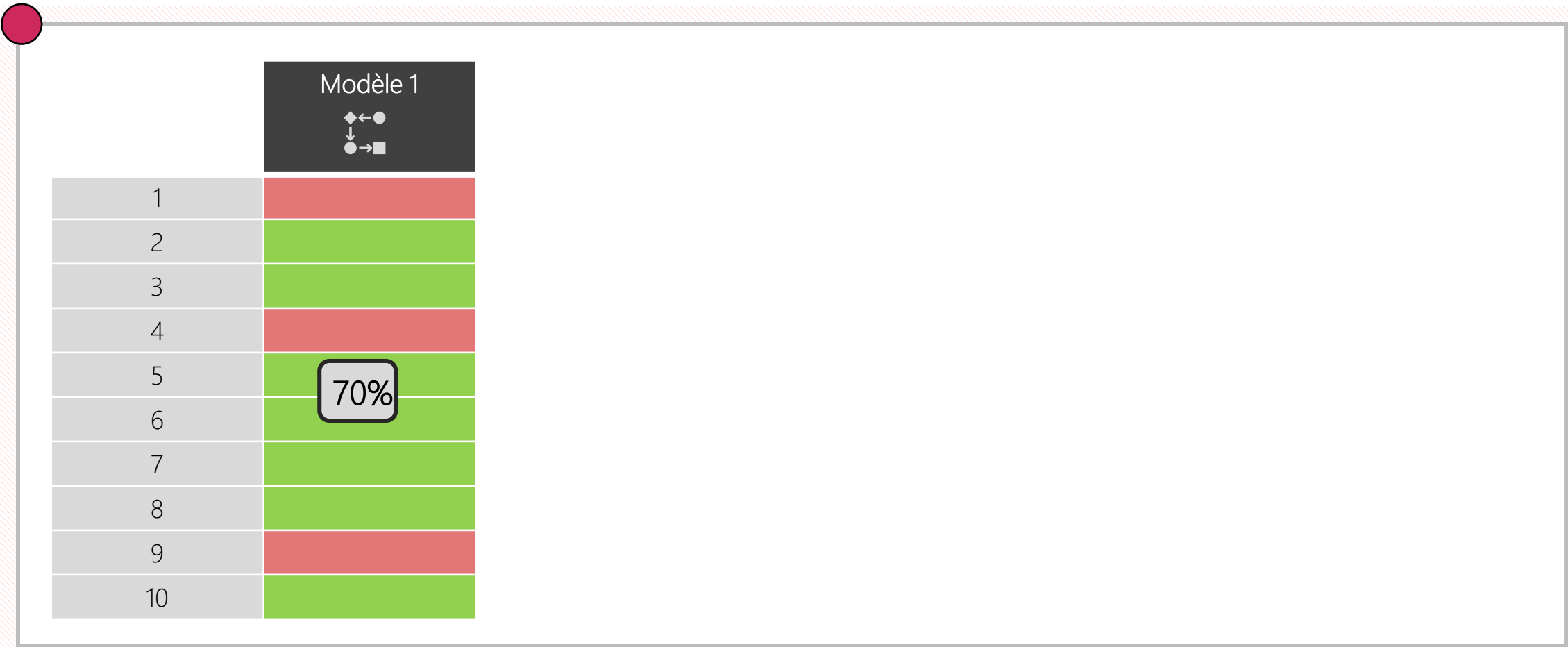
Bien qu'il y ait un nombre illimité de combinaisons possibles , il existe 3 approches différentes :

**BOOSTING BAGGING STACKING**

Ce sont les algorithmes actuels les plus performants

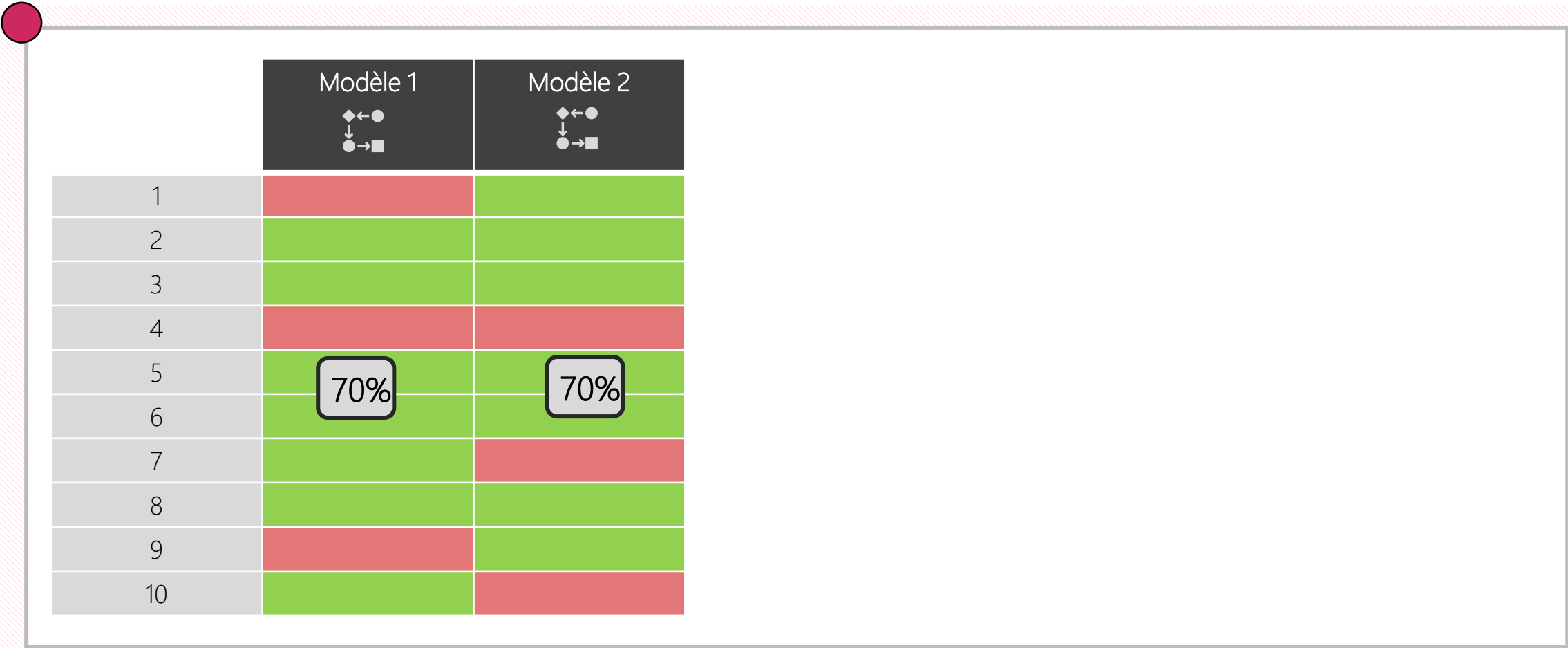
# ENSEMBLE LEARNING

## 1. LE PRINCIPE



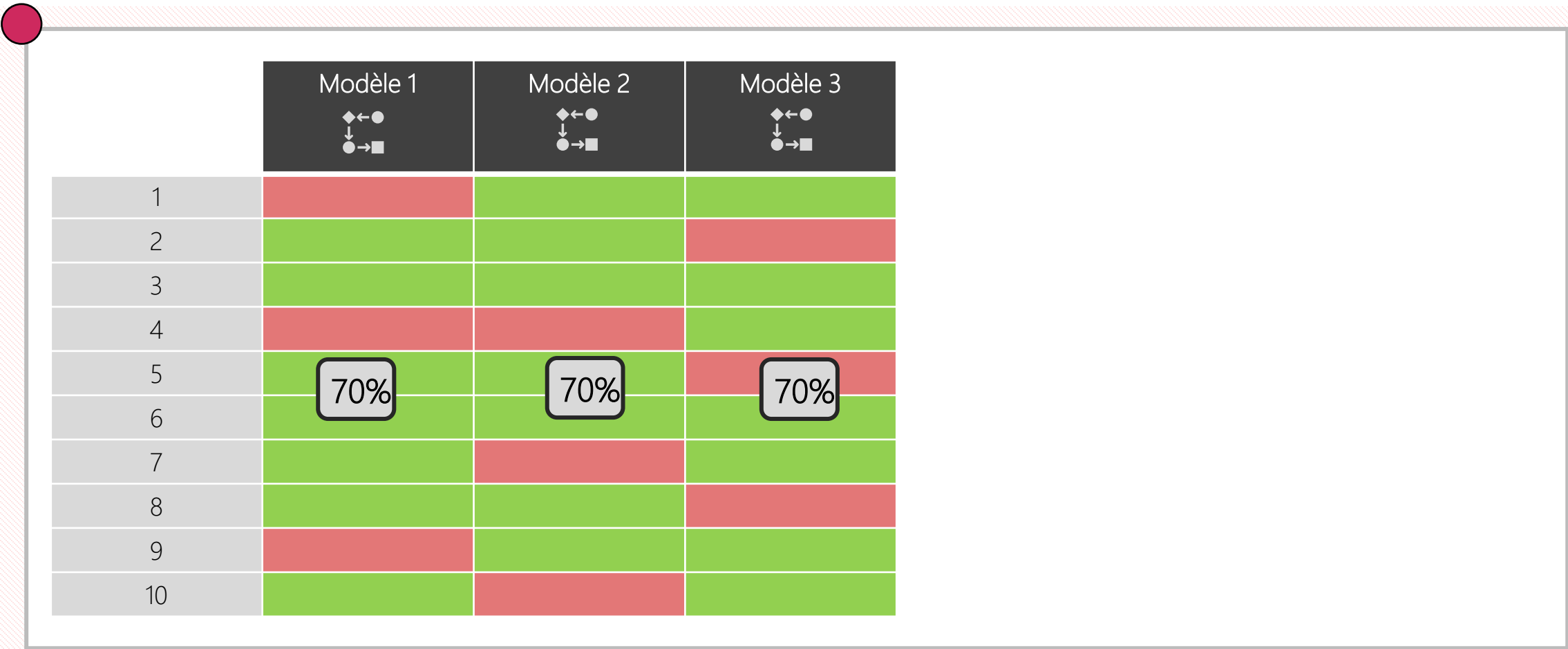
# ENSEMBLE LEARNING

## 1. LE PRINCIPE






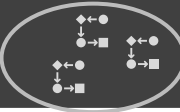
# ENSEMBLE LEARNING

## 1. LE PRINCIPE



# ENSEMBLE LEARNING

## 1. LE PRINCIPE

	Modèle 1 	Modèle 2 	Modèle 3 	Ensemble 
1	Red	Green	Green	Green
2	Green	Green	Red	Green
3	Green	Green	Green	Green
4	Red	Red	Green	Red
5	Green	Green	Red	Green
6	Green	Green	Green	Green
7	Green	Red	Green	Green
8	Green	Green	Red	Green
9	Red	Green	Green	Green
10	Green	Red	Green	Green

En regroupant plusieurs modèles on obtient une meilleure performance que si l'on prenait chaque modèle à part

# ENSEMBLE LEARNING

## 2. CONDITIONS

➡ Plus le nombre de modèles augmentent plus la performance de la majorité s'approchent de 100% de bonnes prédictions .

⚠ A condition:

- que les modèles aient au moins 50% de bonnes réponses. Sinon la performance collective convergera vers 0% ➔ **critère de compétence minimum**
- Les algorithmes doivent présenter un minimum de différences sinon les mauvaises prédictions des uns ne seront pas corrigées les bonnes prédictions des autres ➔ **critère de diversification minimum**

l'idée est que les faiblesses des uns soient compensées par les forces des autres !

➡ **Suit la loi des grands nombres**



# ENSEMBLE LEARNING

## 3. LES DIFFÉRENTES APPROCHES

### Bagging

on teste un modèle sur plusieurs échantillons aléatoires d'un même dataset on fait la moyenne des prédictions.

On utilise une technique d'échantillonnage appelée Bootstrapping: après chaque tirage les données sont remplacées dans le dataset de sorte à ce chaque test soit suffisamment diversifié mais partage des connaissances en commun avec les autres tests

Ex : Random forest

### Boosting

on entraîne l'un après l'autre des modèles ayant de faibles performances.

l'ajout séquentiel de modèles corrigent les prédictions faites par les modèles précédents. Les modèles deviennent alors complémentaires

Ex : Adaboost, XGBoost  
Gradientboosting

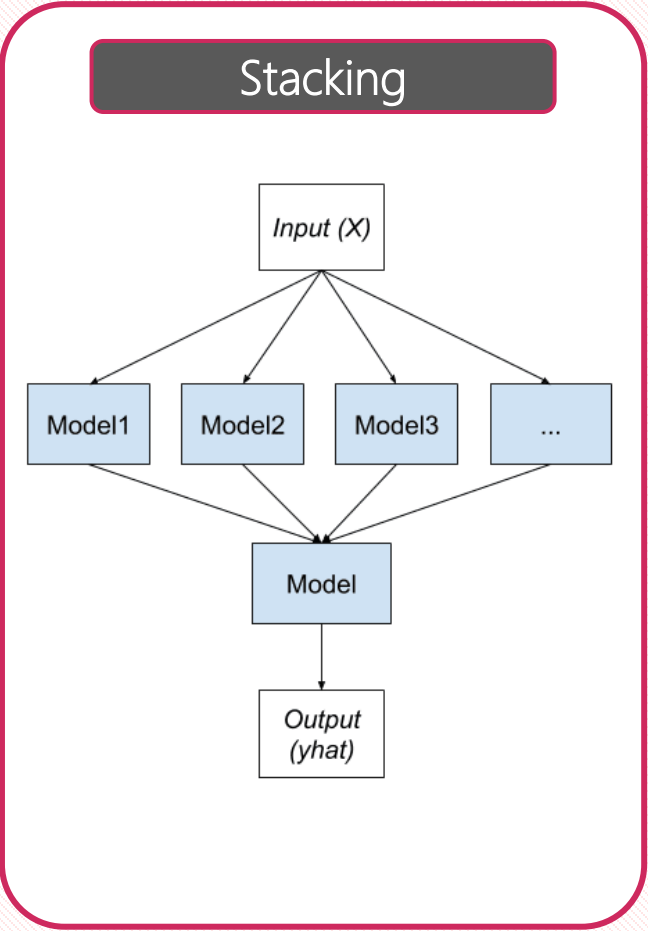
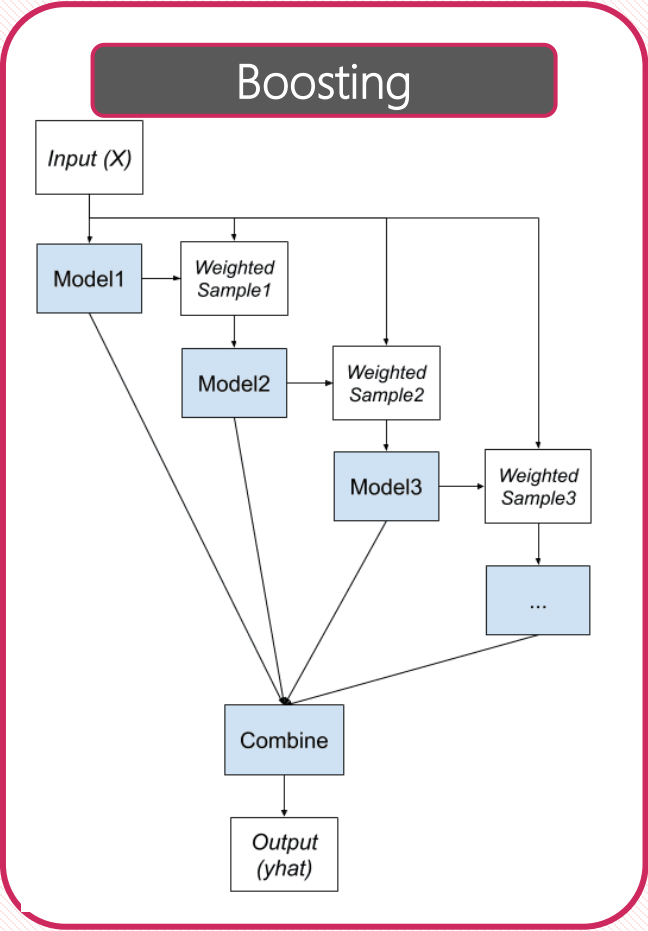
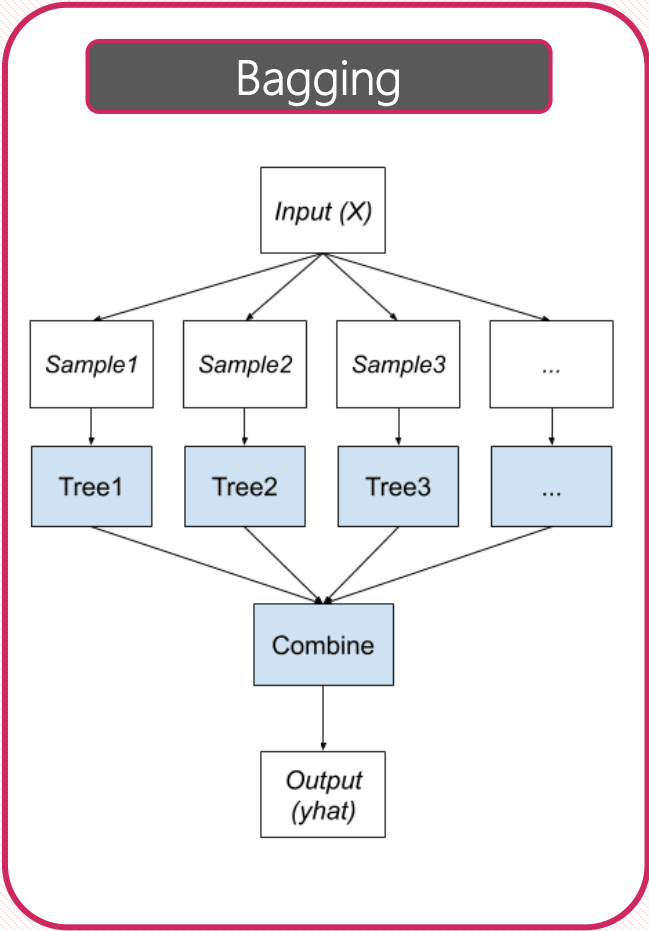
### Stacking

on empile les modèles sur les mêmes données et on utilise un autre pour apprendre à combiner au mieux les prédictions.

Autrement dit, Au lieu de rassembler les prédictions des différents modèles comme en bagging, on détermine un nouvel estimateur qui apprend à prédire en fonction des prédictions fournies par les autres modèles.

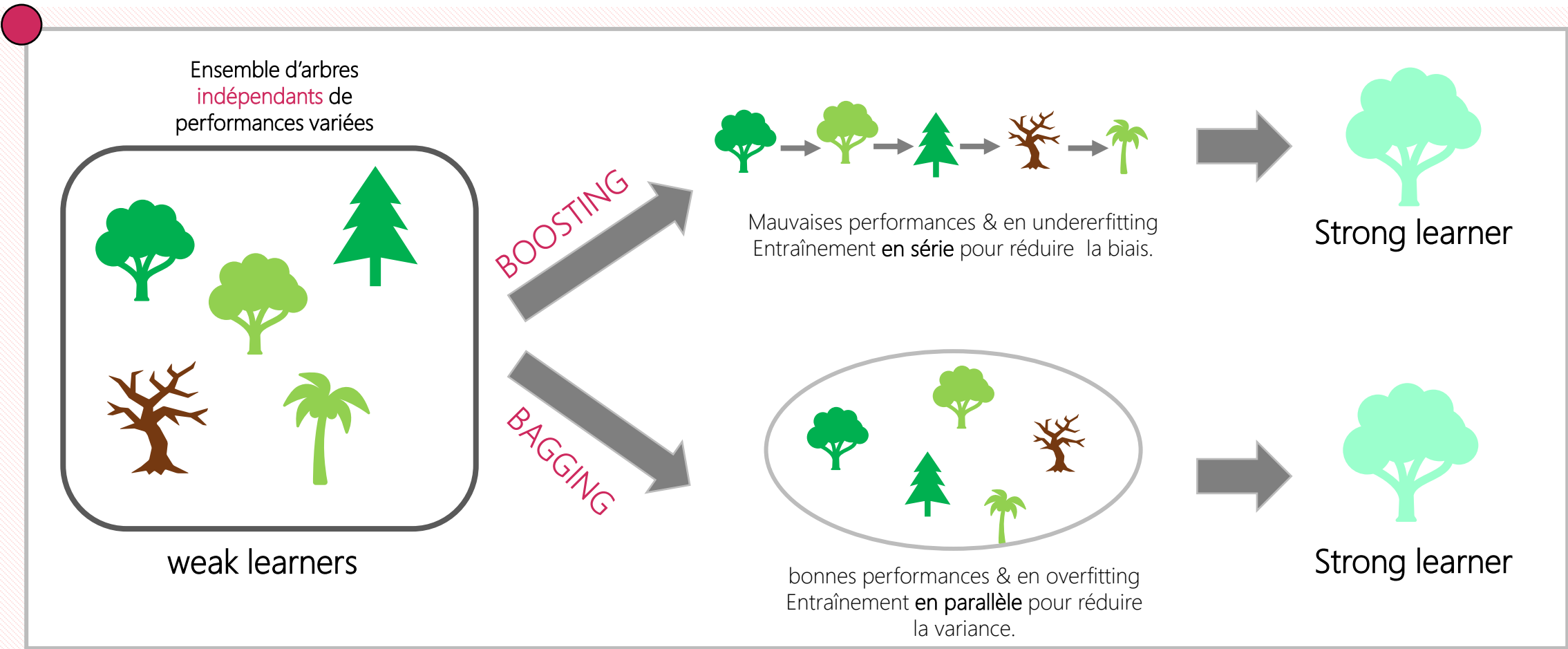
# ENSEMBLE LEARNING

## 3. LES DIFFÉRENTES APPROCHES



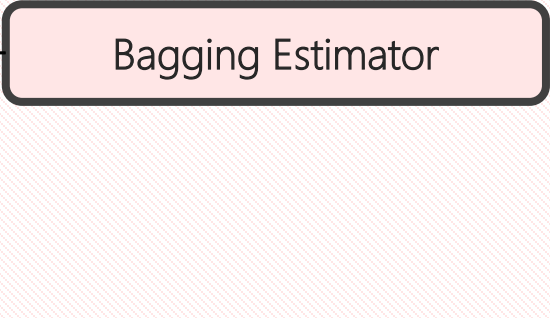
# ENSEMBLE LEARNING

## 4. BOOSTING VS BAGGING



# ENSEMBLE LEARNING

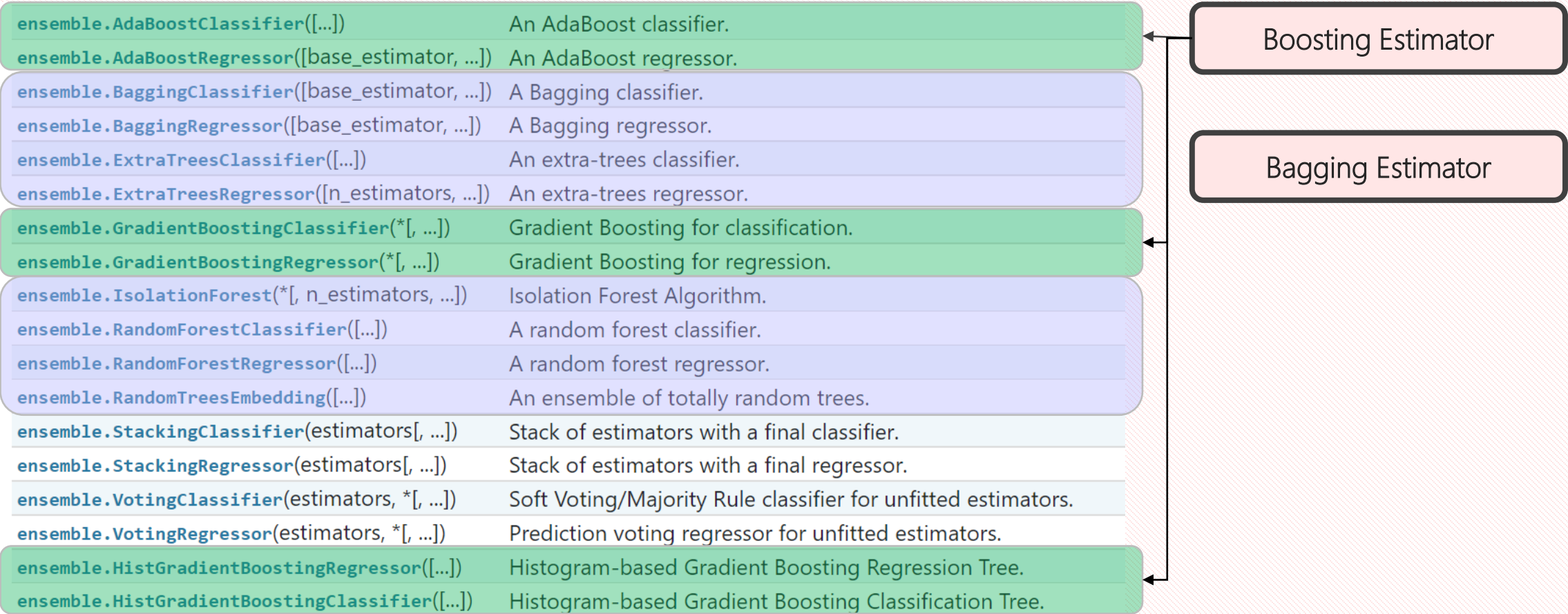
## 3. LES ESTIMATEURS

<code>ensemble.AdaBoostClassifier([...])</code>	An AdaBoost classifier.	
<code>ensemble.AdaBoostRegressor([base_estimator, ...])</code>	An AdaBoost regressor.	
<code>ensemble.BaggingClassifier([base_estimator, ...])</code>	A Bagging classifier.	
<code>ensemble.BaggingRegressor([base_estimator, ...])</code>	A Bagging regressor.	
<code>ensemble.ExtraTreesClassifier([...])</code>	An extra-trees classifier.	
<code>ensemble.ExtraTreesRegressor([n_estimators, ...])</code>	An extra-trees regressor.	
<code>ensemble.GradientBoostingClassifier(*[, ...])</code>	Gradient Boosting for classification.	
<code>ensemble.GradientBoostingRegressor(*[, ...])</code>	Gradient Boosting for regression.	
<code>ensemble.IsolationForest(*[, n_estimators, ...])</code>	Isolation Forest Algorithm.	
<code>ensemble.RandomForestClassifier([...])</code>	A random forest classifier.	
<code>ensemble.RandomForestRegressor([...])</code>	A random forest regressor.	
<code>ensemble.RandomTreesEmbedding([...])</code>	An ensemble of totally random trees.	
<code>ensemble.StackingClassifier(estimators[, ...])</code>	Stack of estimators with a final classifier.	
<code>ensemble.StackingRegressor(estimators[, ...])</code>	Stack of estimators with a final regressor.	
<code>ensemble.VotingClassifier(estimators, *[, ...])</code>	Soft Voting/Majority Rule classifier for unfitted estimators.	
<code>ensemble.VotingRegressor(estimators, *[, ...])</code>	Prediction voting regressor for unfitted estimators.	
<code>ensemble.HistGradientBoostingRegressor([...])</code>	Histogram-based Gradient Boosting Regression Tree.	
<code>ensemble.HistGradientBoostingClassifier([...])</code>	Histogram-based Gradient Boosting Classification Tree.	



# ENSEMBLE LEARNING

## 3. LES ESTIMATEURS



# ENSEMBLE LEARNING

## 3. LES ESTIMATEURS

<code>ensemble.AdaBoostClassifier([...])</code>	An AdaBoost classifier.
<code>ensemble.AdaBoostRegressor([base_estimator, ...])</code>	An AdaBoost regressor.
<code>ensemble.BaggingClassifier([base_estimator, ...])</code>	A Bagging classifier.
<code>ensemble.BaggingRegressor([base_estimator, ...])</code>	A Bagging regressor.
<code>ensemble.ExtraTreesClassifier([...])</code>	An extra-trees classifier.
<code>ensemble.ExtraTreesRegressor([n_estimators, ...])</code>	An extra-trees regressor.
<code>ensemble.GradientBoostingClassifier(*[, ...])</code>	Gradient Boosting for classification.
<code>ensemble.GradientBoostingRegressor(*[, ...])</code>	Gradient Boosting for regression.
<code>ensemble.IsolationForest(*[, n_estimators, ...])</code>	Isolation Forest Algorithm.
<code>ensemble.RandomForestClassifier([...])</code>	A random forest classifier.
<code>ensemble.RandomForestRegressor([...])</code>	A random forest regressor.
<code>ensemble.RandomTreesEmbedding([...])</code>	An ensemble of totally random trees.
<code>ensemble.StackingClassifier(estimators[, ...])</code>	Stack of estimators with a final classifier.
<code>ensemble.StackingRegressor(estimators[, ...])</code>	Stack of estimators with a final regressor.
<code>ensemble.VotingClassifier(estimators, *[, ...])</code>	Soft Voting/Majority Rule classifier for unfitted estimators.
<code>ensemble.VotingRegressor(estimators, *[, ...])</code>	Prediction voting regressor for unfitted estimators.
<code>ensemble.HistGradientBoostingRegressor([...])</code>	Histogram-based Gradient Boosting Regression Tree.
<code>ensemble.HistGradientBoostingClassifier([...])</code>	Histogram-based Gradient Boosting Classification Tree.

Boosting Estimator

Bagging Estimator

Stacking Estimator



# ENSEMBLE LEARNING

## 3. LES ESTIMATEURS

<code>ensemble.AdaBoostClassifier([...])</code>	An AdaBoost classifier.
<code>ensemble.AdaBoostRegressor([base_estimator, ...])</code>	An AdaBoost regressor.
<code>ensemble.BaggingClassifier([base_estimator, ...])</code>	A Bagging classifier.
<code>ensemble.BaggingRegressor([base_estimator, ...])</code>	A Bagging regressor.
<code>ensemble.ExtraTreesClassifier([...])</code>	An extra-trees classifier.
<code>ensemble.ExtraTreesRegressor([n_estimators, ...])</code>	An extra-trees regressor.
<code>ensemble.GradientBoostingClassifier(*[, ...])</code>	Gradient Boosting for classification.
<code>ensemble.GradientBoostingRegressor(*[, ...])</code>	Gradient Boosting for regression.
<code>ensemble.IsolationForest(*[, n_estimators, ...])</code>	Isolation Forest Algorithm.
<code>ensemble.RandomForestClassifier([...])</code>	A random forest classifier.
<code>ensemble.RandomForestRegressor([...])</code>	A random forest regressor.
<code>ensemble.RandomTreesEmbedding([...])</code>	An ensemble of totally random trees.
<code>ensemble.StackingClassifier(estimators[, ...])</code>	Stack of estimators with a final classifier.
<code>ensemble.StackingRegressor(estimators[, ...])</code>	Stack of estimators with a final regressor.
<code>ensemble.VotingClassifier(estimators, *[, ...])</code>	Soft Voting/Majority Rule classifier for unfitted estimators.
<code>ensemble.VotingRegressor(estimators, *[, ...])</code>	Prediction voting regressor for unfitted estimators.
<code>ensemble.HistGradientBoostingRegressor([...])</code>	Histogram-based Gradient Boosting Regression Tree.
<code>ensemble.HistGradientBoostingClassifier([...])</code>	Histogram-based Gradient Boosting Classification Tree.

Boosting Estimator

Bagging Estimator

Stacking Estimator

Voting Estimator

# ENSEMBLE LEARNING

## 4. FOCUS SUR VOTING ESTIMATOR

```
class sklearn.ensemble.VotingClassifier(estimators, *, voting='hard', weights=None,  
                                       n_jobs=None, flatten_transform=True, verbose=False)
```

Les hyperparamètres importants :

**Estimator** : les modèles choisis

**Voting** : la façon de voter

*Hard* → vote sur les prédictions (on additionne la prédiction de chaque donnée, et la classe qui a le plus de prédiction est choisit)

*Soft* → vote sur les probabilités de chaque classe (on additionne les proba de chaque classe fournit par les modèles et on choisit la classe qui a la plus grande)



# ENSEMBLE LEARNING

## 5. FOCUS SUR LES BAGGING ESTIMATOR

- `class sklearn.ensemble.BaggingClassifier(base_estimator, n_estimators, max_samples, max_features, bootstrap, bootstrap_features, oob_score, warm_start, n_jobs, random_state, verbose)`
- `class sklearn.ensemble.RandomForestClassifier(n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, min_impurity_decrease, bootstrap, oob_score, n_jobs, random_state, verbose, warm_start, class_weight, ccp_alpha, max_samples)`

Les hyperparamètres importants :

Base\_estimator : l'estimateur sur lequel on travaille

N\_estimator : combien de fois on veut réitérer le processus

# ENSEMBLE LEARNING

## 6. LA RANDOM FOREST

La randomForest est un **ensemble d'arbres de décision**, autrement dit une **forêt**

Chaque arbre est entraîné sur un sous-ensemble du dataset.

Ces **sous ensemble** sont constitués de façon **aléatoire**, d'où le terme **random**

Les résultats de tous les arbres de décision sont alors combinés pour donner une réponse finale. Chaque arbre "vote" et la réponse finale est celle qui a eu la majorité de vote.

# ENSEMBLE LEARNING

## 6. LA RANDOM FOREST

### Quelques remarques

- On cherche à obtenir des arbres les plus **décorrélés possibles** : Plus les arbres seront différents les uns des autres plus ils pourront se compléter pour former un modèle robuste . → On introduit une perturbation « aléatoire » dans la construction des arbres, en jouant sur le mécanisme de sélection de variables de segmentation sur les nœuds.
- Dans les forêts aléatoires, il n'est pas nécessaire de réaliser une validation croisée pour obtenir une estimation de l'erreur de l'ensemble de test. Il est estimé en interne c'est ce qu'on appelle l'**oob\_error** . C'est une méthode de détermination de l'erreur de prédiction qui permet à la forêt aléatoire d'être adaptée et validée tout en étant entraînée.
- Il est impossible d'analyser l'ensemble des arbres pour évaluer l'influence de chaque variable prédictive dans la modélisation, c'est pourquoi on utilise '**feature importance**'

Pour aller plus loin :

- <https://www.datacamp.com/community/tutorials/random-forests-classifier-python>
- <https://www.quora.com/What-is-the-out-of-bag-error-in-Random-Forest-algorithm>

# ENSEMBLE LEARNING

## 6. LA RANDOM FOREST



### Avantages

- Pas de sur-apprentissage
- meilleure performance que les arbres de décision, calcul de l'erreur "Out-of-Bag" direct
- paramètres faciles à calibrer
- souvent utilisées comme benchmark dans les compétition de machine learning



### Inconvénients

- boîte noire : difficilement interprétable, difficilement améliorable
- entraînement plus lent
- Problème si nous avons un nombre de variables pertinentes très faibles car les arbres individuels risquent de ne pas être performants

# ENSEMBLE LEARNING

## 7. FOCUS SUR LES BOOSTING ESTIMATOR

- `class sklearn.ensemble.AdaBoostRegressor(base_estimator=None, *, n_estimators=50, learning_rate=1.0, loss='linear', random_state=None)`
- `class sklearn.ensemble.AdaBoostClassifier(base_estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)`

Les hyperparamètres importants :

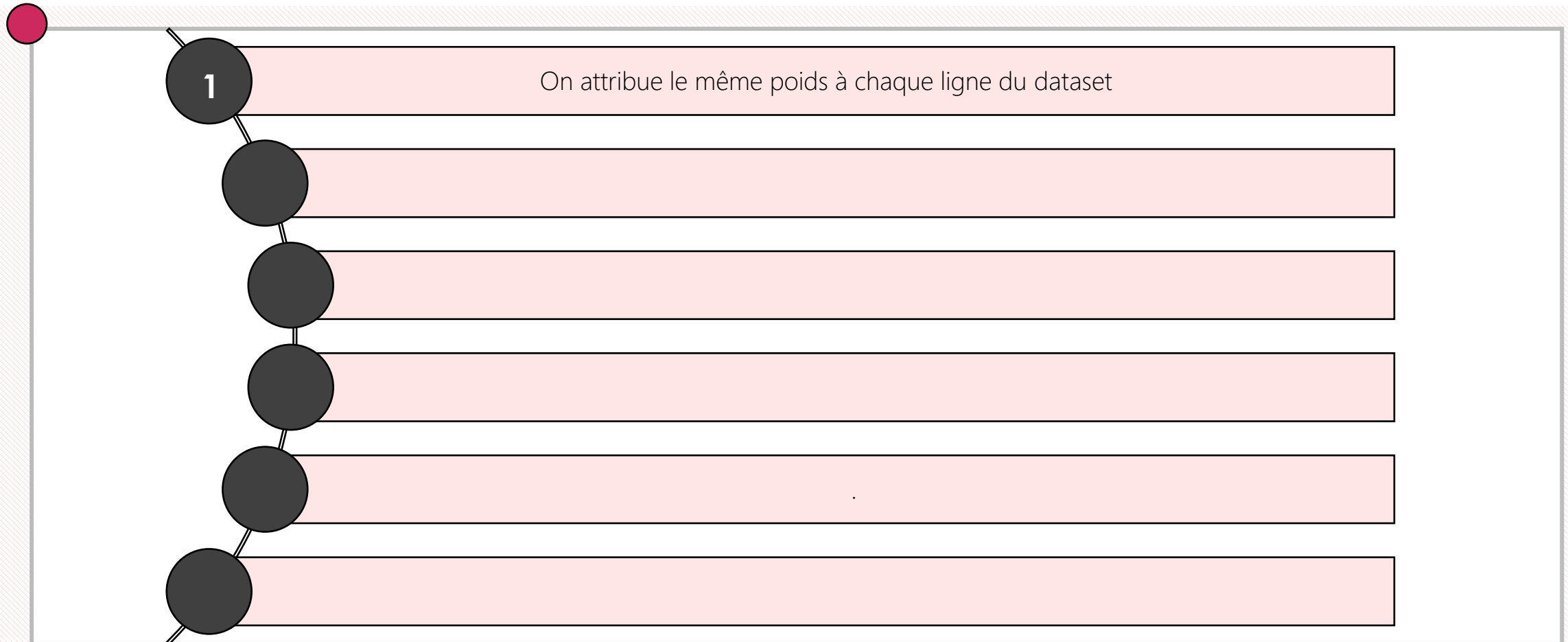
**Base\_estimator** : l'estimateur sur lequel on travaille

**N\_estimator** : combien de fois on veut réitérer le processus

**Learning\_rate** : Pondération appliquée à chaque classifieur pour chaque itération de boosting.

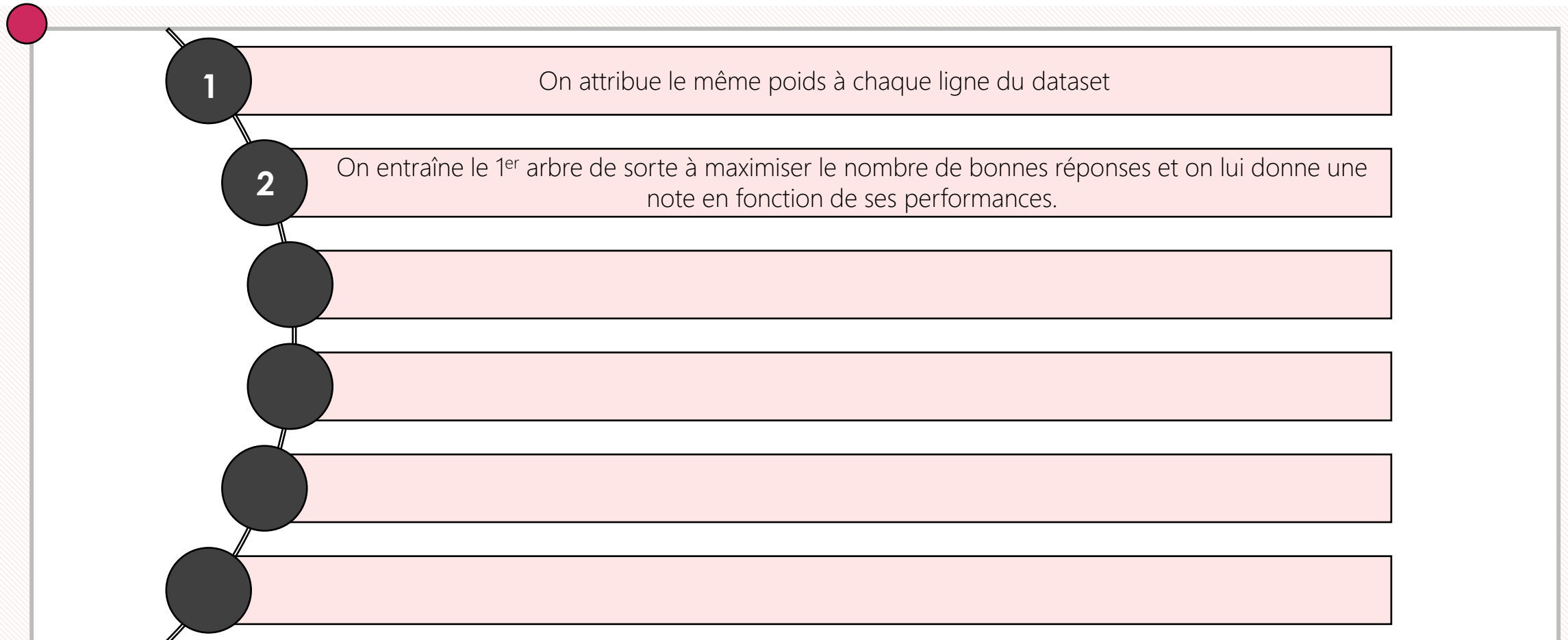
# ENSEMBLE LEARNING

## 4. L'ALGORITHME ADABOOST



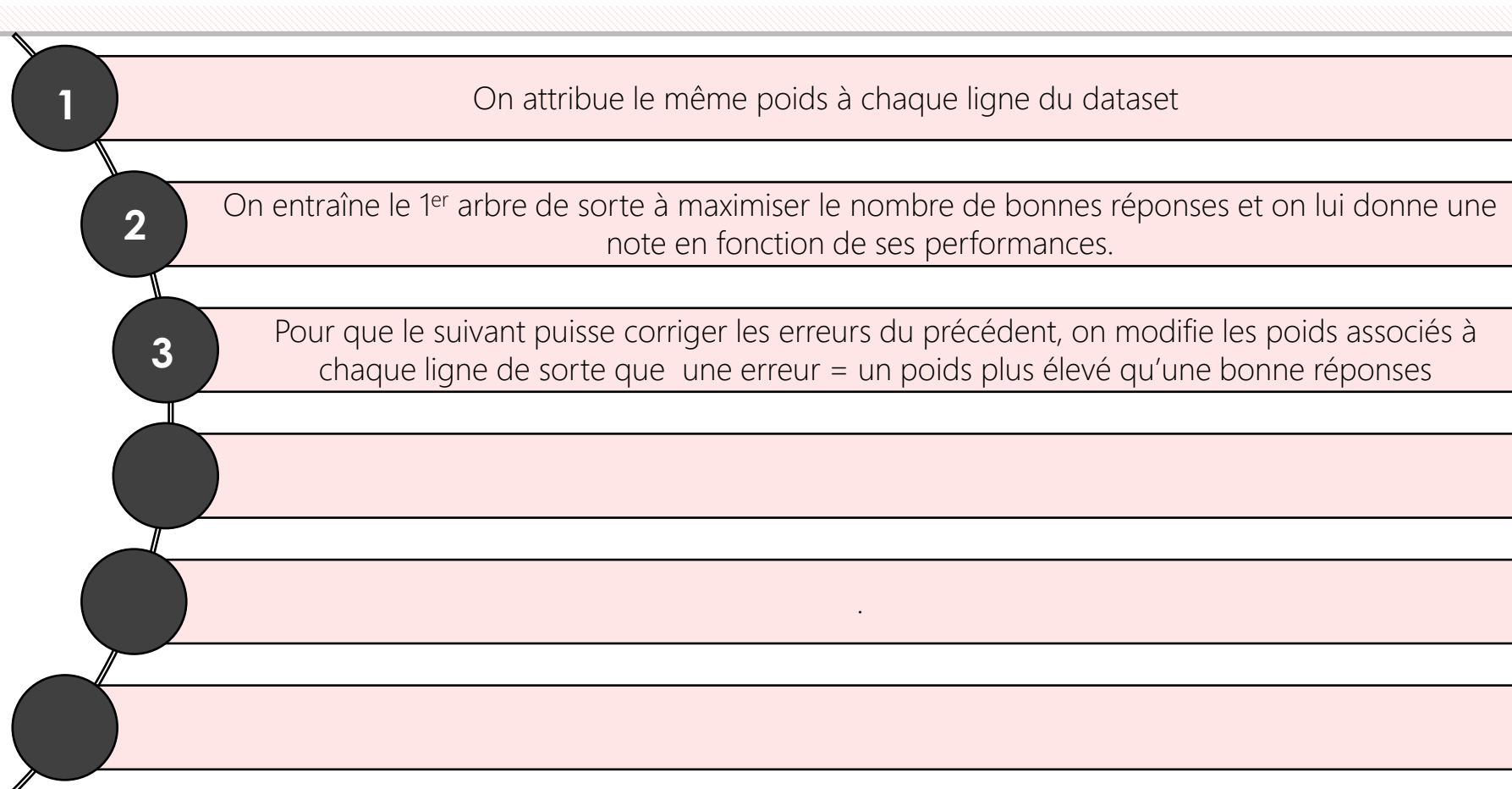
# ENSEMBLE LEARNING

## 4. L'ALGORITHME ADABOOST



# ENSEMBLE LEARNING

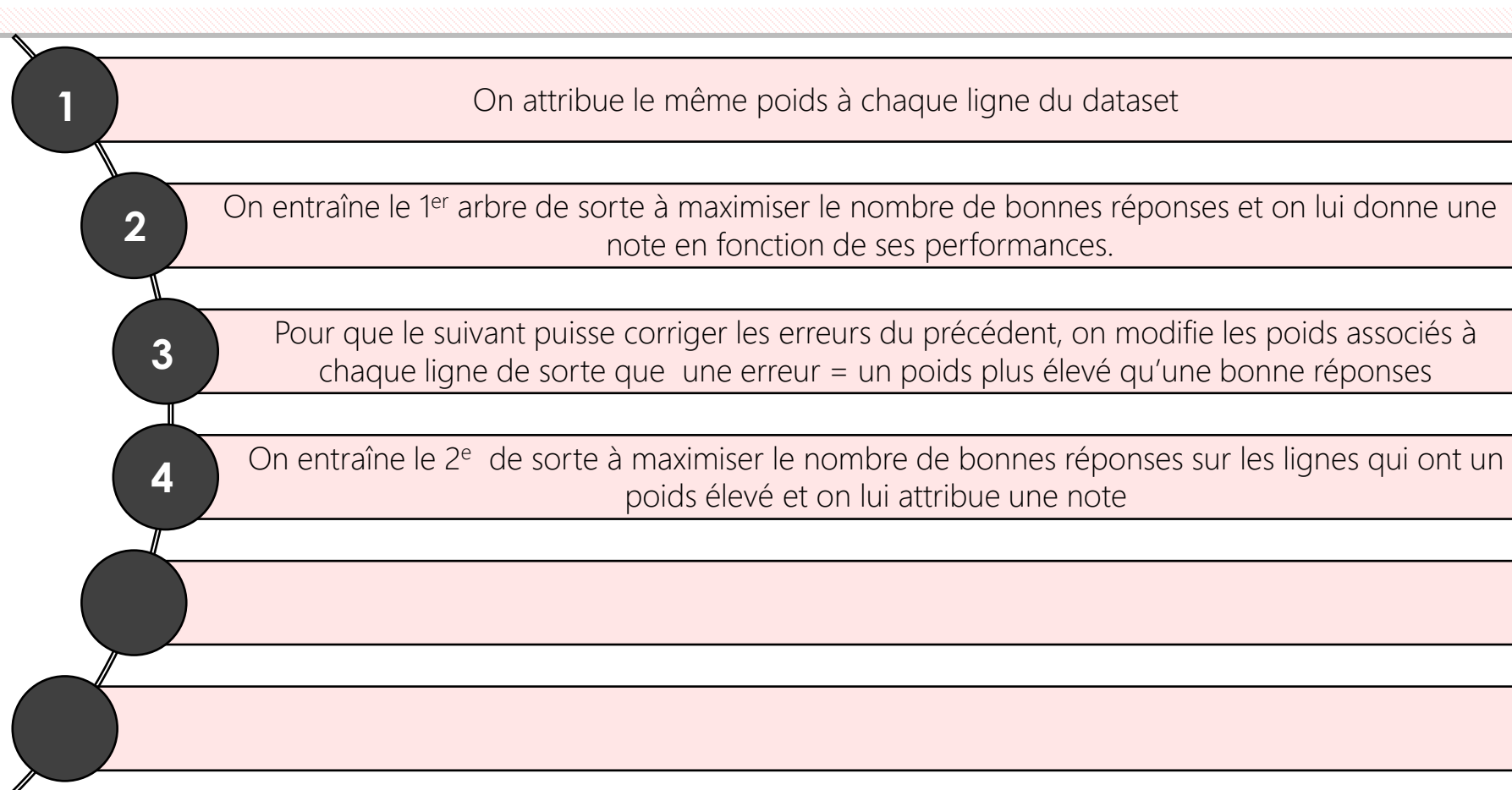
## 4. L'ALGORITHME ADABOOST





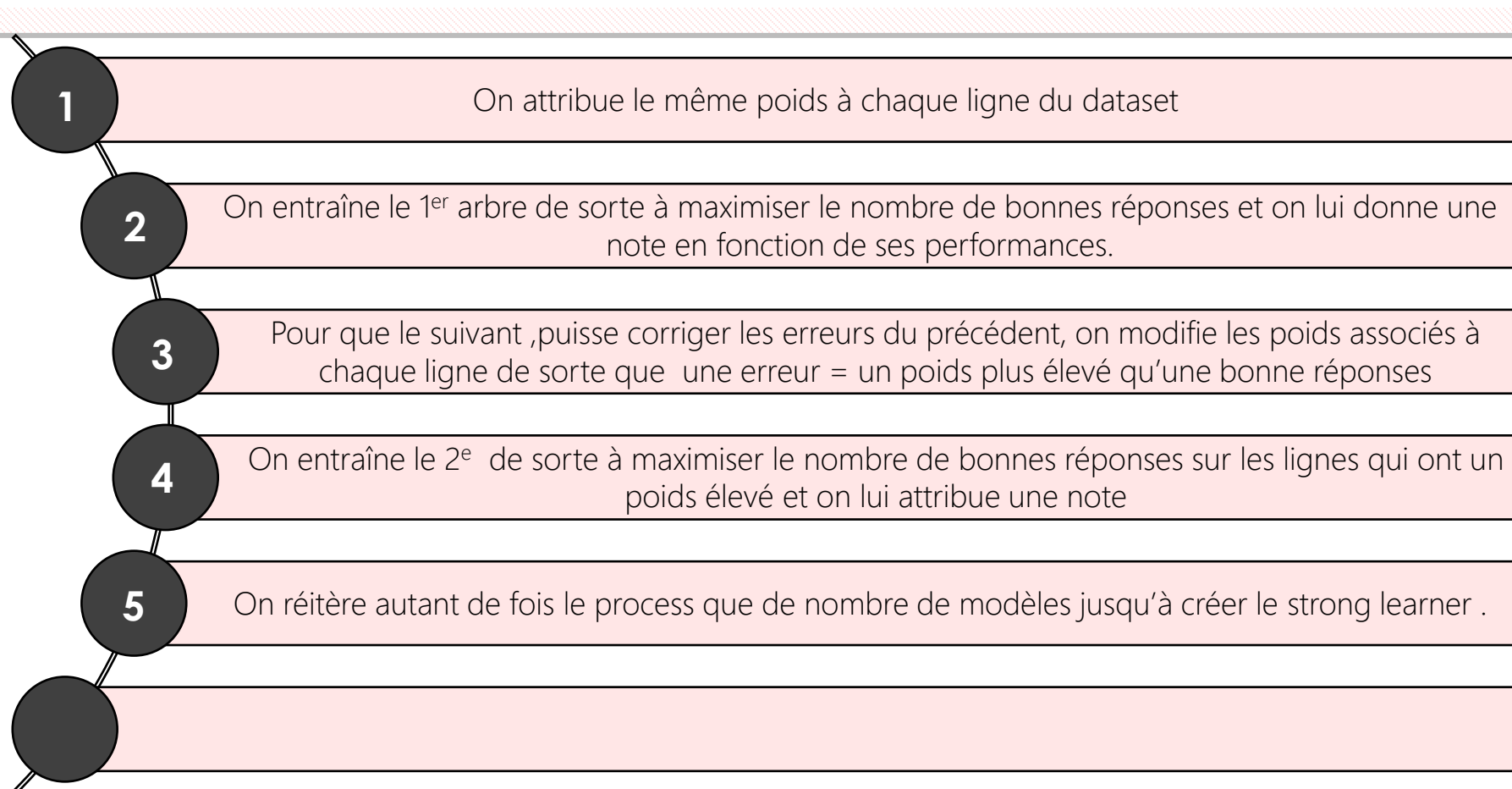
# ENSEMBLE LEARNING

## 4. L'ALGORITHME ADABOOST



# ENSEMBLE LEARNING

## 4. L'ALGORITHME ADABOOST



# ENSEMBLE LEARNING

## 4. L'ALGORITHME ADABOOST

