EPSI l'École d'ingénierie informatique

Machine Learning

Cours 4

Jean-Claude Houbart

houbart.jc@gmail.com

EPSI l'École d'ingénierie informatique

Modèles d'ensemble

Jean-Claude Houbart

houbart.jc@gmail.com

Vocabulaire

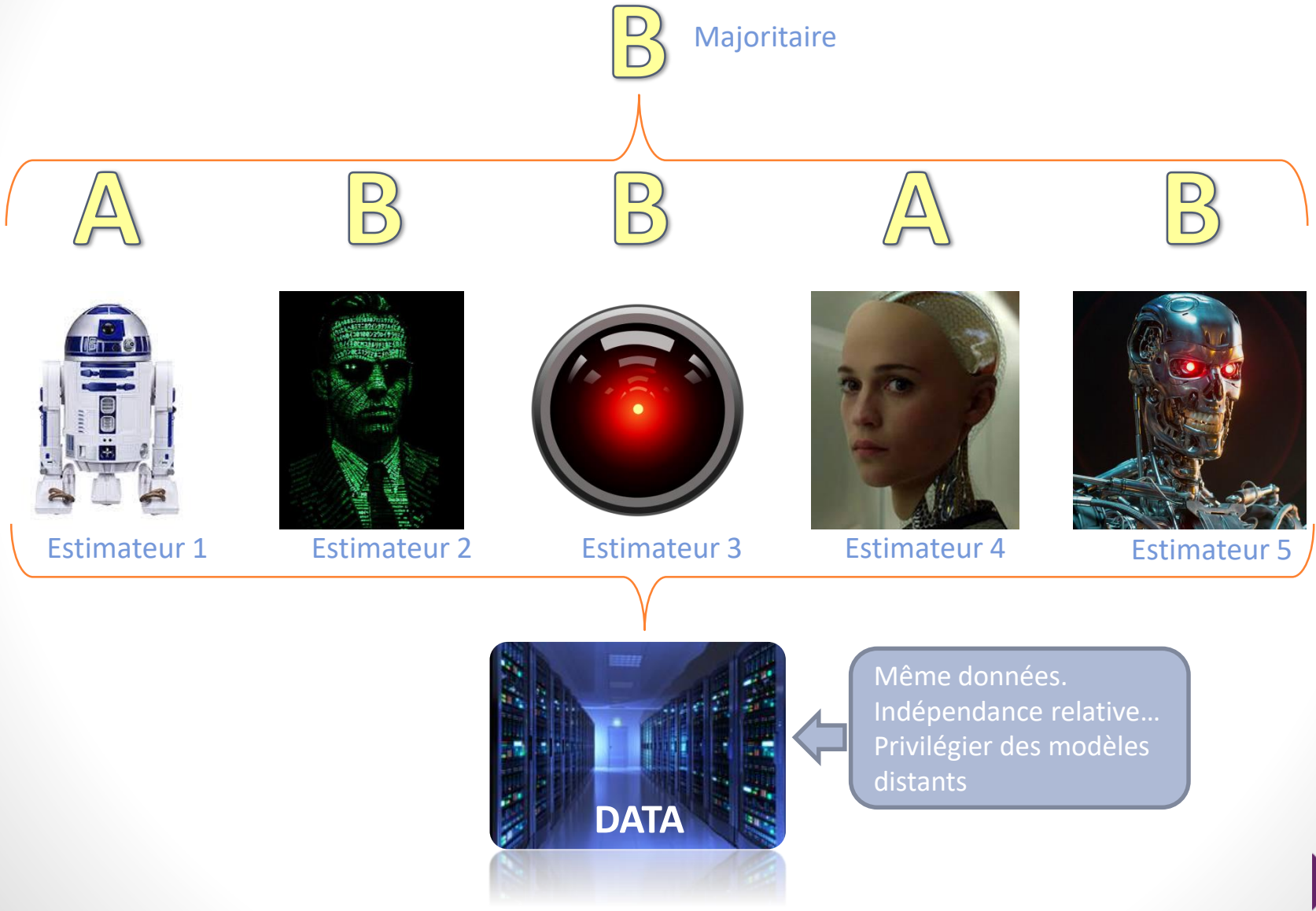
- Pour parler d'un apprentissage particulier, on peut parler d'un **modèle** ou d'un **estimateur** (ou **estimator**).
- Un **estimateur** pour la classification peut être appelé **Classifier**.
- Un **estimateur** pour la Régression peut être appelé **Regressor**.
- Dans SkLearn, les modèles ont souvent une version « Classifier » et une version « Regressor ».

Principe des méthodes d'ensembles

- Les meilleurs résultats sont obtenus en combinant plusieurs estimateurs indépendants. On parle de « Sagesse des foules ».
- Le résultat surpasse généralement le meilleur modèle.
- Ainsi une combinaison de nombreux estimateurs faibles (c'est-à-dire à peine meilleurs que le hasard) permet d'obtenir un estimateur fort.
- Par exemple, imaginons des estimateurs qui donnent 51 fois sur 100 la bonne réponse :
  - Si on combine 1000 estimateurs **indépendants** de ce type, nous avons 75% de probabilité d'avoir la bonne réponse.
  - Si on combine 10000 estimateurs **indépendants** de ce type, nous avons 97% de probabilité d'avoir la bonne réponse.
- C'est une conséquence de la loi des grands nombres :
  - Pour une approche intuitive des mathématiques impliquées : https://www.youtube.com/watch?v=2Wq6H8GMVm0

Systèmes de vote « hard »

- Le diagnostic majoritaire parmi les différents modèles est retenu.

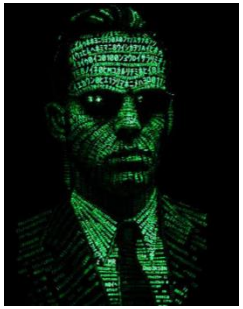
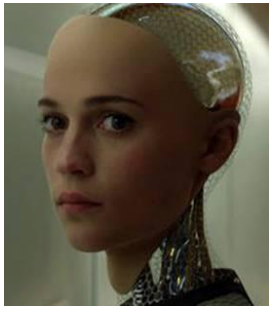


Systèmes de vote « soft »

- Le diagnostic dont la probabilité moyenne est maximale est retenu.

A

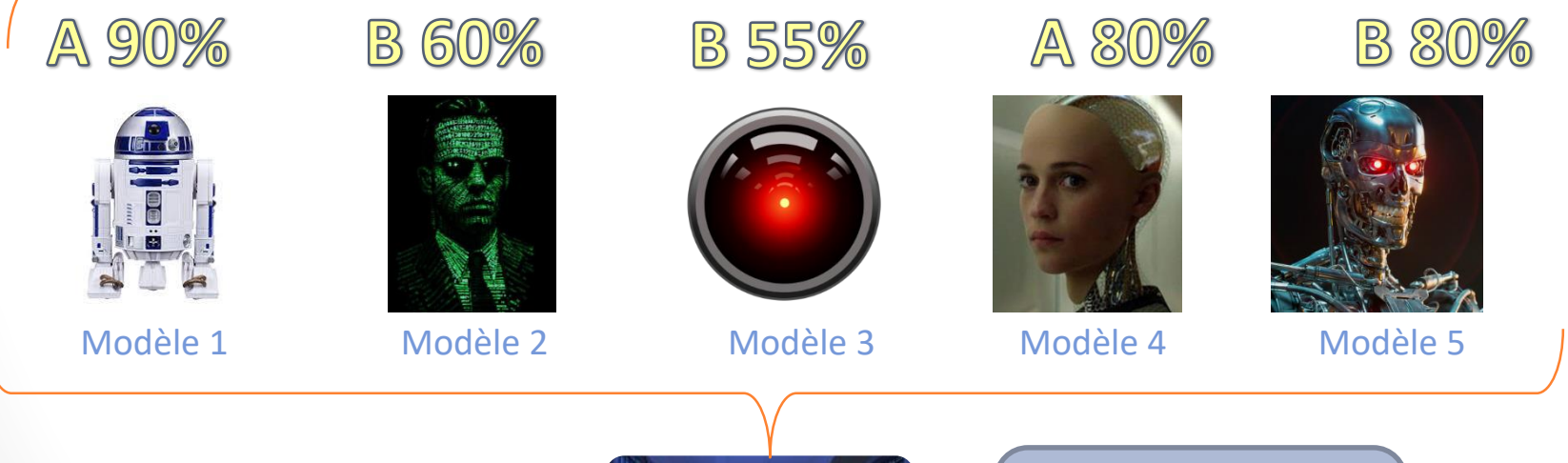
$\frac{0,9+0,4+0,45+0,8+0,2}{5}$ = A 55% B 45% = $\frac{0,1+0,6+0,55+0,2+0,8}{5}$

A 90%	B 60%	B 55%	A 80%	B 80%
				
Modèle 1	Modèle 2	Modèle 3	Modèle 4	Modèle 5

Même données.
Indépendance relative…
Privilégier des modèles distants

Stacking

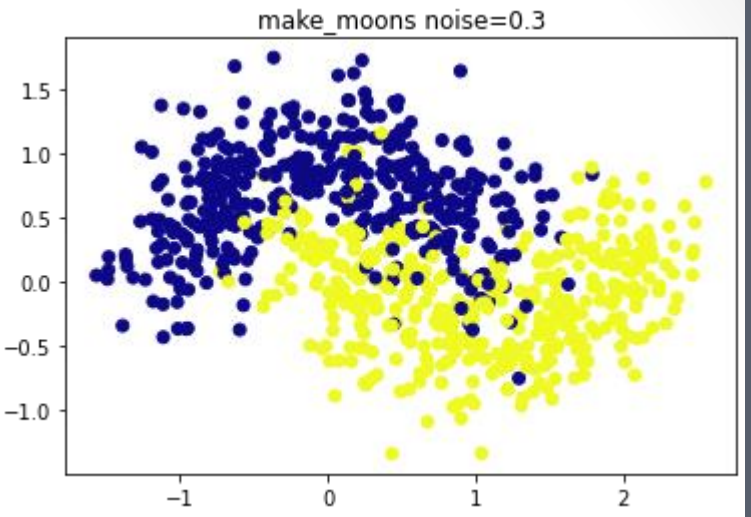
- Au lieu d'utiliser un système de vote, empiler un autre estimateur. Pas de méthode prédéfinie dans sklearn.
- La formule magique pour gagner sur Kaggle.



Même données.
Indépendance relative…
Privilégier des modèles distants

Moon DataSet

- scikit-learn comprend divers générateurs d'échantillons aléatoires.
- Ils peuvent être utilisés pour construire des ensembles de données artificielles
- La taille et la complexité des données est contrôlée.



- make_moons génère un Dataset aléatoire à 2 dimension avec un label de classification :

```
sklearn.datasets.make_moons(
        n_samples=100,  # Nombre d'exemples
        shuffle=True,       # mélange ou pas des exemples
        noise=None,         # Bruit ajouté aux données
        random_state=None
                # Graine, permettant de reproduire les mêmes donnée
)
```

Exemple de « Voting Classifier »

```
# Librairies
from sklearn.datasets import make_moons
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB

#Génération et Organisation Dataset
X, y = make_moons(n_samples=500, noise=0.3, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

#Classifiers
log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42)
bayes_clf = GaussianNB()

voting_clf = VotingClassifier(estimators=[('lr', log_clf),
                                          ('rf', rnd_clf),
                                          ('by', bayes_clf)],
                              voting='hard')

#Training et Prédiction
for clf in (log_clf, rnd_clf, bayes_clf, voting_clf):
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
```

Voir cours4_ex4_VotingClassifiers.ipynb

Bagging

- **Un seul modèle** (ou **estimateur**) est utilisé. Par contre, il est entraîné de multiples fois avec des **sous-ensembles de données** différents.
- Ensuite on va prendre le **résultat majoritaire** des résultats en classification ou la **moyenne** en régression.
- Dans Sklearn, le paramètre **max_features** permet limiter le nombre de features pour chaque modèle.
- Le paramètre **bootstrap_features** perme d'utiliser tirer aléatoirement plusieurs fois la même features pour le même modèle.

Bagging et Pasting

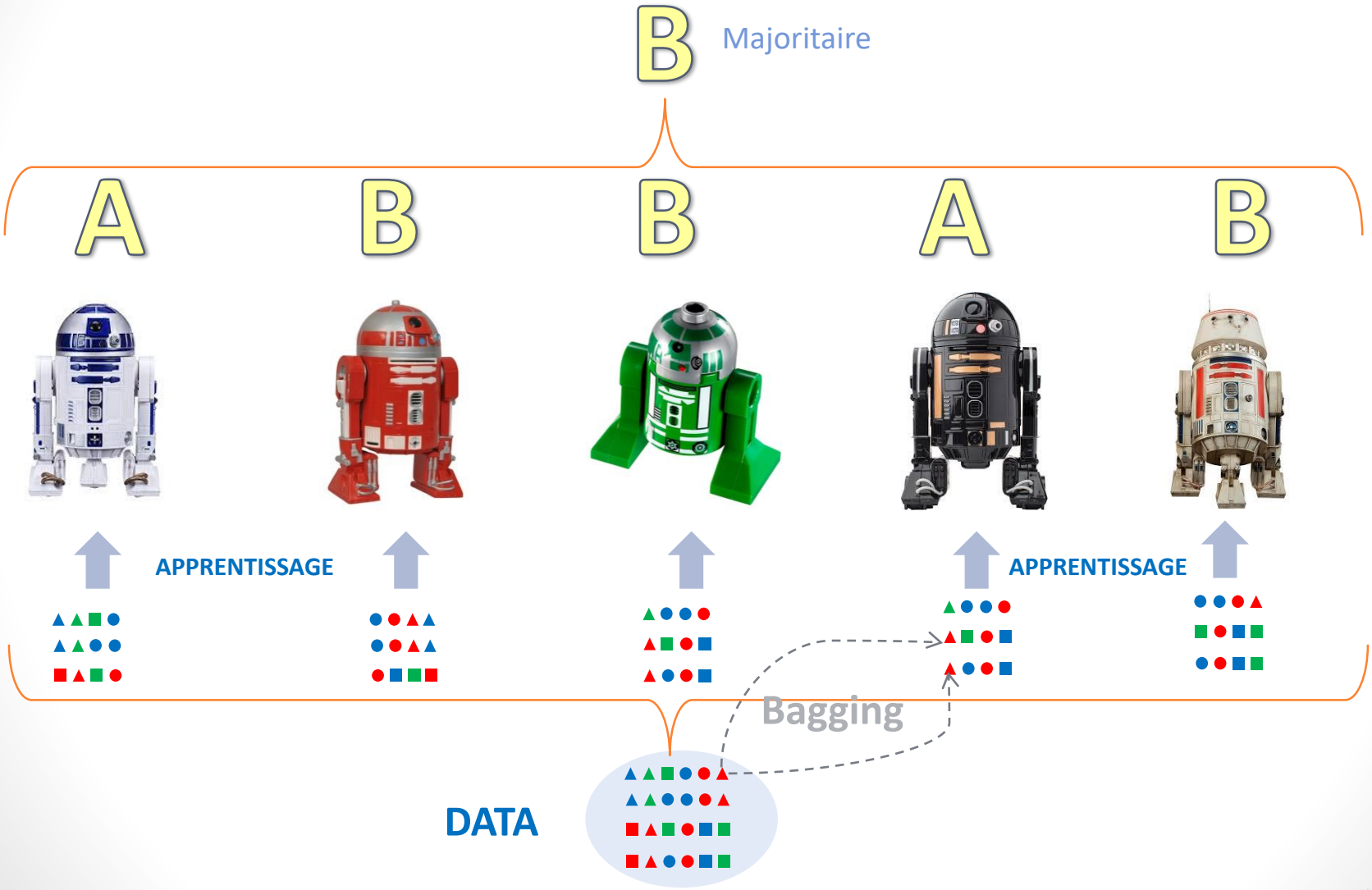
- Un même exemple peut être utilisé pour plusieurs apprentissages.

- Seul le **bagging** permet qu'un exemple soit utilisé plusieurs fois dans un même apprentissage (**bootstrap ou « Tirage avec remise »**).

- un modèle **Random Forest** est un Bagging d'arbres de décision, dont chaque échantillon a la taille du jeu de données initial (mais n'est pas identique !).

Bagging et Pasting

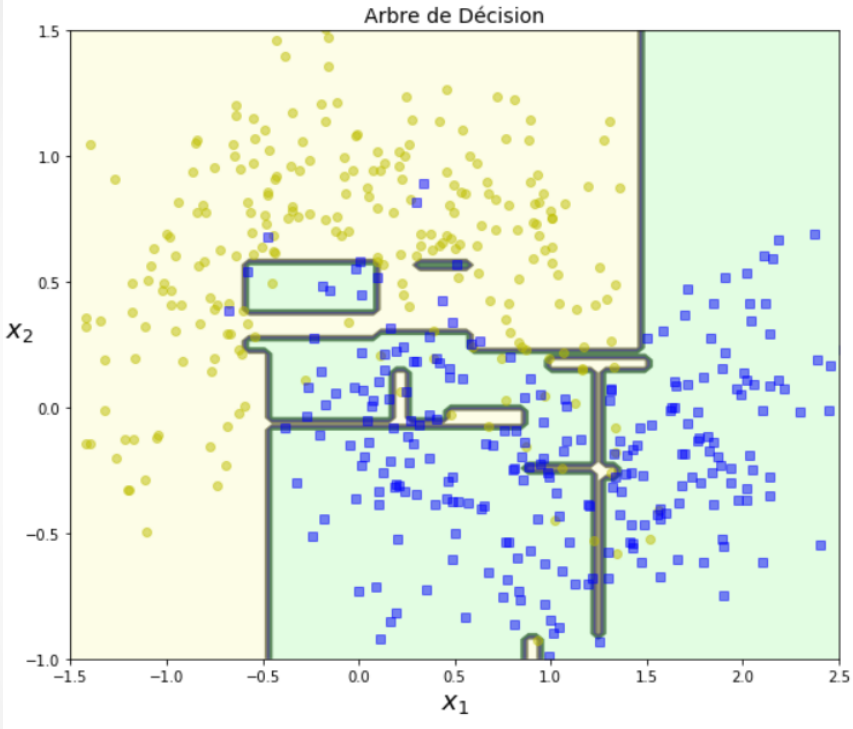
- Le diagnostic majoritaire parmi les différents modèle est retenu.

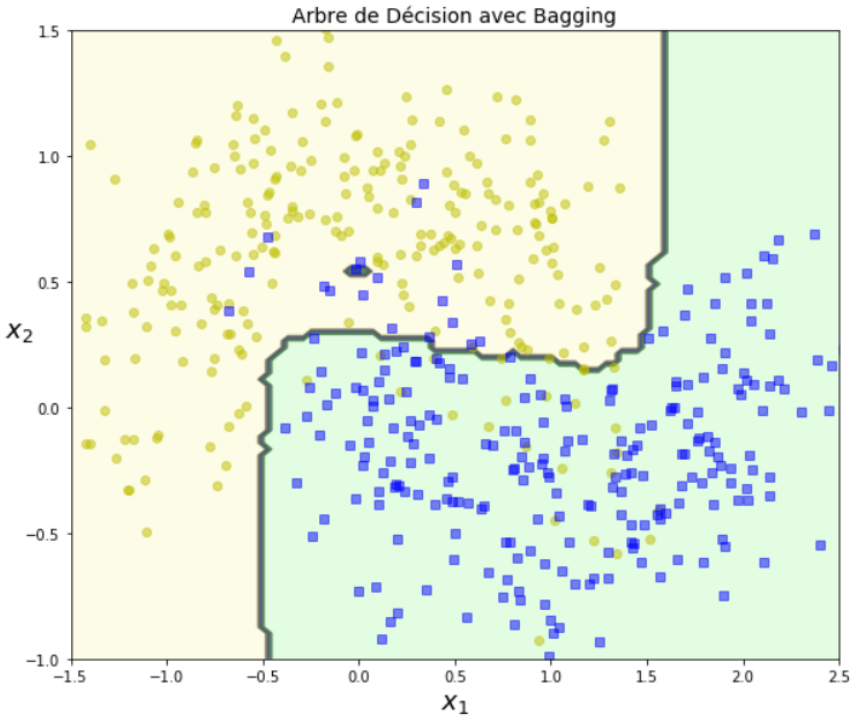


Exemple de Bagging

```
#Génération et Organisation Dataset
X, y = make_moons(n_samples=500, noise=0.3, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=42)

#Classifiers
bag_clf = BaggingClassifier(
   DecisionTreeClassifier(), # Classifier de base (ici un arbre)
   n_estimators=500, # Nombre d'estimateurs utilisés
   max_samples=100, # Taille échantillon pour chaque apprentissage
   bootstrap=True, # Tirage avec remise (Bagging)
   n_jobs=-1, # Parralélisation (-1 pour utilisé tout les coeurs dispos)
   oob_score=True, # Calcul Score sur chaque estimateur à partir des données non
   sélectionnées
   random_state=7 # Pour obtenir toujours le même tirage aléatoire
)
bag_clf.fit(X_train, y_train)
print("out of the bag score :", bag_clf.oob_score_)
y_pred = bag_clf.predict(X_test)
print("score :", accuracy_score(y_test, y_pred))
```

Effet du Bagging





- Biais équivalent (même nombre d'erreurs) .
- Variance diminuée (moins de sur-apprentissage).
- Dans le cas du bagging, il reste des données out of the bag => Ont peut faire une évaluation de chaque apprentissage avec la propriété oob_scores_

Voir cours4_ex5_Bagging.ipynb

EPSI l'École d'ingénierie informatique

Boosting

Jean-Claude Houbart

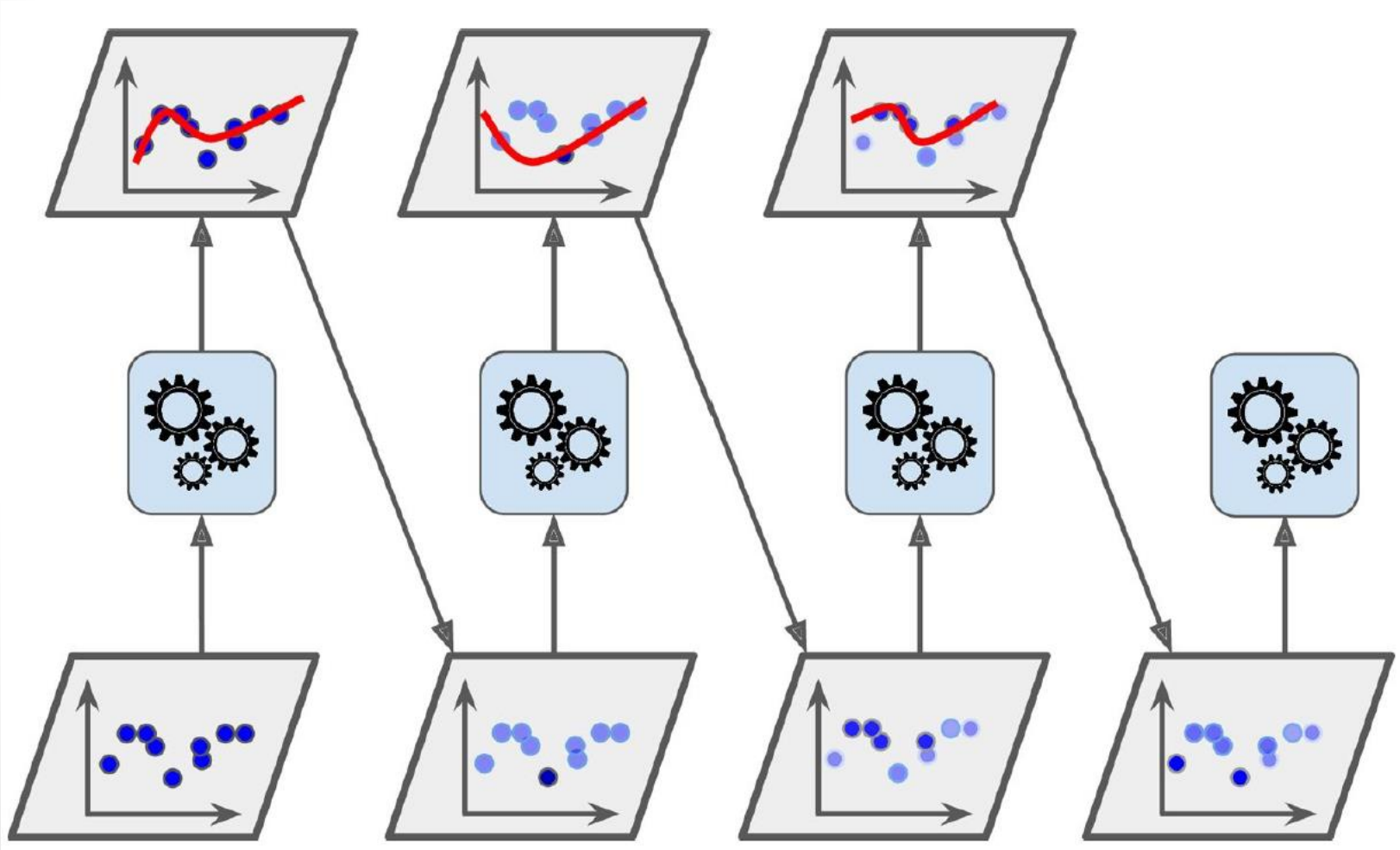
houbart.jc@gmail.com

Types de Boosting

- Le Boosting est une méthode d'ensemble d'estimateurs faibles traités **séquentiellement**.
- Chaque estimateur essaie de corriger son prédécesseur.
- L'**Adaptative Boosting** (**Adaboost**) va insister sur les exemples les plus erronés.
- Le **Gradient Boosting** va chercher à prévoir les erreurs de son prédécesseur.
- Le gradient boosting (librairie XGBoost) est en tête des compétitions Kaggle pour les données tabulaires.
- On peut utiliser un paramètre de **Learning rate** qui réduit l'importance de chaque itération.
- Il existe d'autres méthodes, moins répandues.
- Le problème de ces méthodes est qu'elles ne sont pas parallélisables.

Adaptative Boosting

- A chaque itération, l'algorithme surpondère les points les plus erronés.
- Ensuite, on applique une moyenne pondérée par la précision aux modèles de chaque itération.



Exemple Adaboost

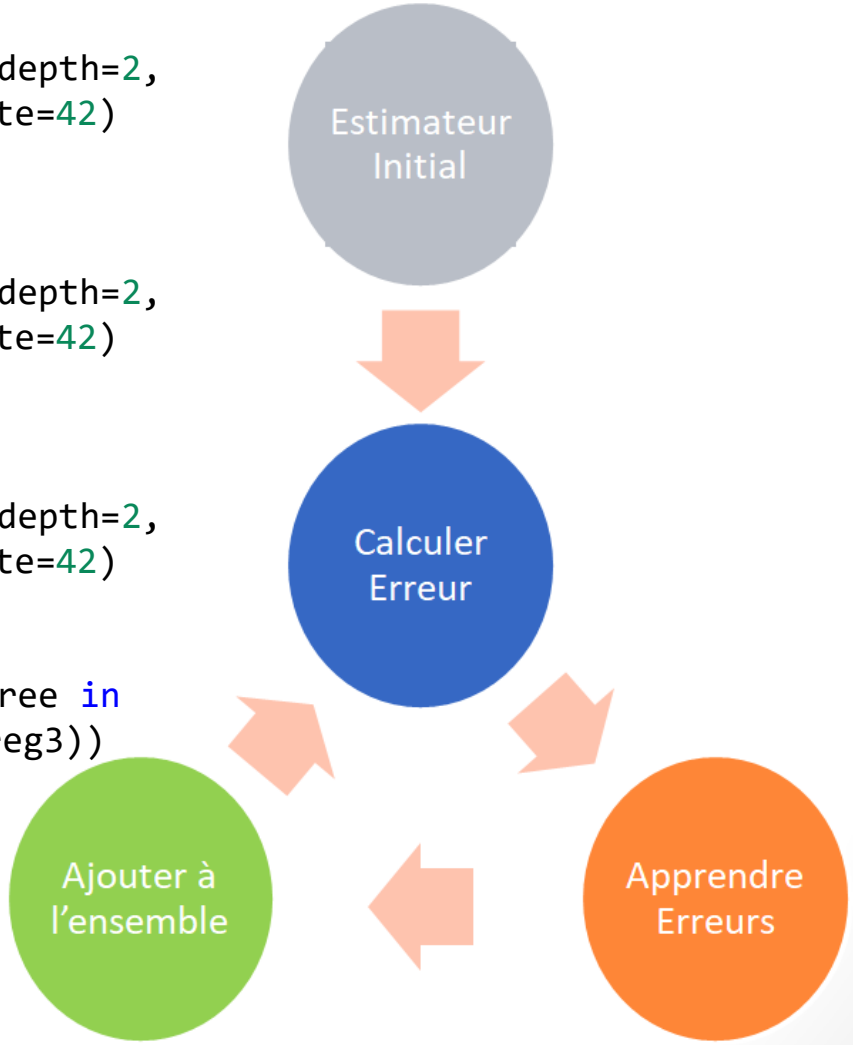
```
# Importation des librairies
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Création des données
X, y = make_moons(n_samples=500, noise=0.3, random_state=7)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=43)

#Création du modèle et apprentissage
ada_clf = AdaBoostClassifier(
        DecisionTreeClassifier(max_depth=1), n_estimators=100,
        algorithm="SAMME.R", learning_rate=0.3,  random_state=42)
ada_clf.fit(X_train, y_train)
y_pred_ada = ada_clf.predict(X_test)
print('score Adaboost :' , accuracy_score(y_test,y_pred_ada))
```

Voir cours3_ex6_Boosting.ipynb

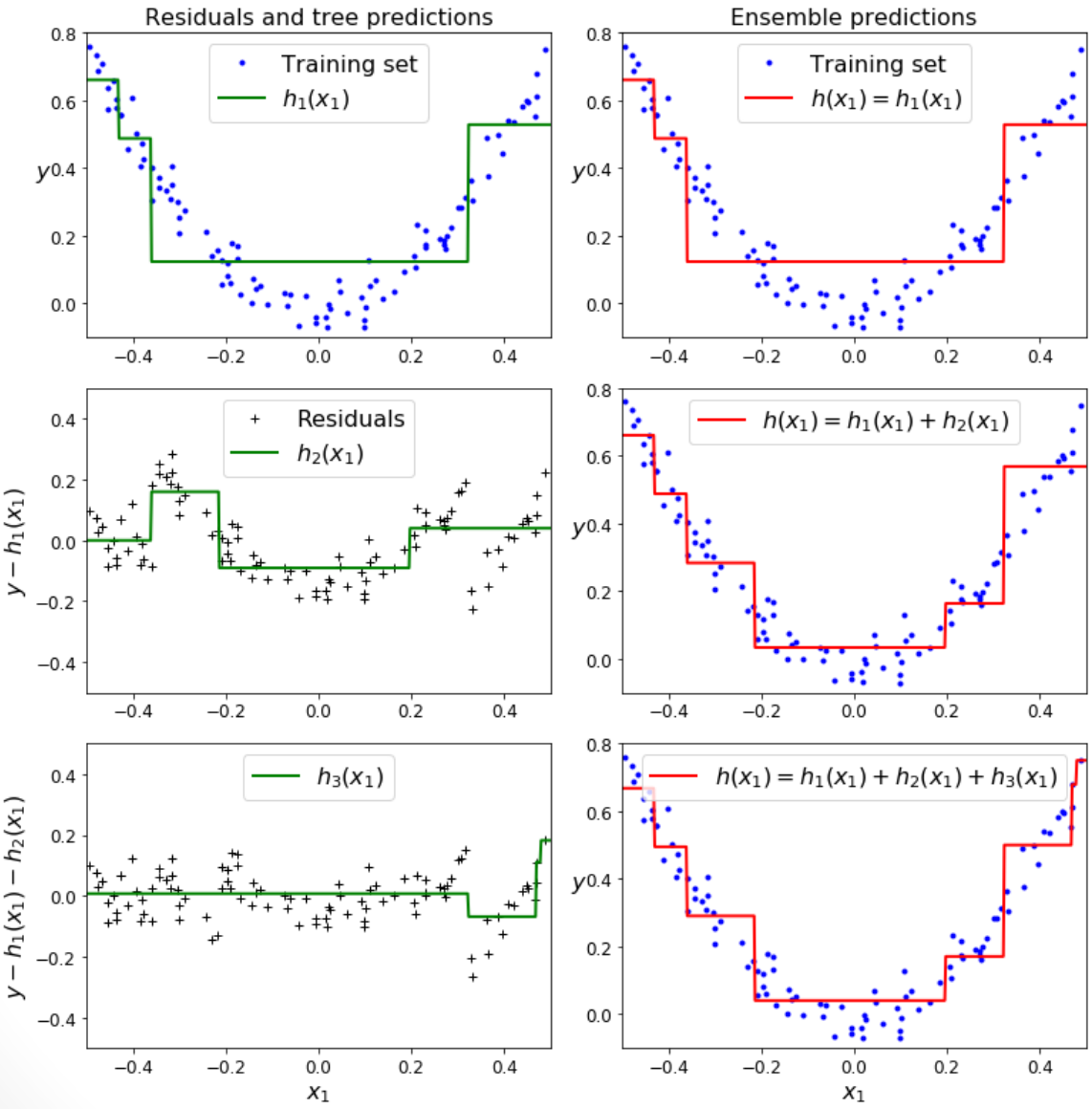
Gradient Boosting

- A chaque itération, l'algorithme cherche à modéliser le **résidu** (l'erreur) du modèle précédent.
- A la fin, on somme les modèles.

```
tree_reg1 = DecisionTreeRegressor(max_depth=2,
                                  random_state=42)
tree_reg1.fit(X, y)

y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2,
                                  random_state=42)
tree_reg2.fit(X, y2)

y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2,
                                  random_state=42)
tree_reg3.fit(X, y3)

y_pred = sum(tree.predict(X_new) for tree in
         (tree_reg1, tree_reg2, tree_reg3))
```



Processus de boosting



Exemple Gradient Boosting

```
#On passe à 80000 pour mieux voir, sinon on aura trop peu d'itérations

X, y = make_moons(n_samples=80000, noise=0.3, random_state=42)

X_train, X_dev, y_train, y_dev = train_test_split(X, y, random_state=43)

n_estimators=500

# Si pas d'amélioration > 0.0001 sur les 5 derniers estimateurs => Arrêt
gbes = GradientBoostingClassifier(
          n_estimators=n_estimators,
          n_iter_no_change=5,
          tol=0.0001,
          random_state=0)


gbes.fit(X_train, y_train)
score_gbes= gbes.score(X_dev, y_dev)
```

Voir cours4_ex6_Boosting.ipynb