



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Forêts Aléatoires (Random Forest) avec SciKit Learn

Fichiers sur

<https://github.com/mkirschpin/CoursPython>

<http://kirschpm.fr/cours/PythonDataScience/>

Random Forest

- La répartition des données **train/test** a une grande influence sur les **arbres de décision**
 - Répartition \neq \rightarrow Arbre \neq \rightarrow Qualité \neq
- Pourquoi ne générer qu'une seule arbre ?

```
1 |--- sex_male <= 0.50
|   |--- passengerClass_1st <= 0.50
|   |   |--- passengerClass_3rd <= 0.50
|   |   |   |--- age <= 56.00
|   |   |   |   |--- class: 1.0
|   |   |   |   |--- age > 56.00
|   |   |   |   |   |--- class: 0.0
|   |   |--- passengerClass_3rd > 0.50
|   |   |   |--- age <= 1.50
|   |   |   |   |--- class: 1.0
|   |   |   |   |--- age > 1.50
|   |   |   |   |   |--- class: 1.0
```

```
6 |--- age <= 8.50
|   |--- passengerClass_2nd <= 0.50
|   |   |--- passengerClass_1st <= 0.50
|   |   |   |--- age <= 0.38
|   |   |   |   |--- class: 0.0
|   |   |   |   |--- age > 0.38
|   |   |   |   |   |--- class: 1.0
|   |   |--- passengerClass_1st > 0.50
|   |   |   |--- class: 1.0
|   |--- passengerClass_2nd > 0.50
|   |   |--- class: 1.0
|--- age > 8.50
```

Méthodes ensemblistes

- Méthodes proposant de combiner **plusieurs modèles**
 - **Objectif** : compenser les erreurs et réduire le sur-apprentissage

Bagging

- Agréger **plusieurs modèles** d'un **même algorithme**
- Modèles créés par \neq **sous-ensembles** des données
- Sous-ensembles choisis **aléatoirement**
- **Prédiction** par **vote** ou **moyenne** des prédictions

Boosting

- Agréger **plusieurs modèles** d'un **même algorithme**
- Modèles créés en **ordre**
- **Sous-ensembles** \neq des données
- Données choisies en **fonction des prédictions** précédentes : \uparrow **erreur**
 \uparrow **probabilité** d'être choisi
- **Prédiction** par **vote** ou **moyenne** des prédictions

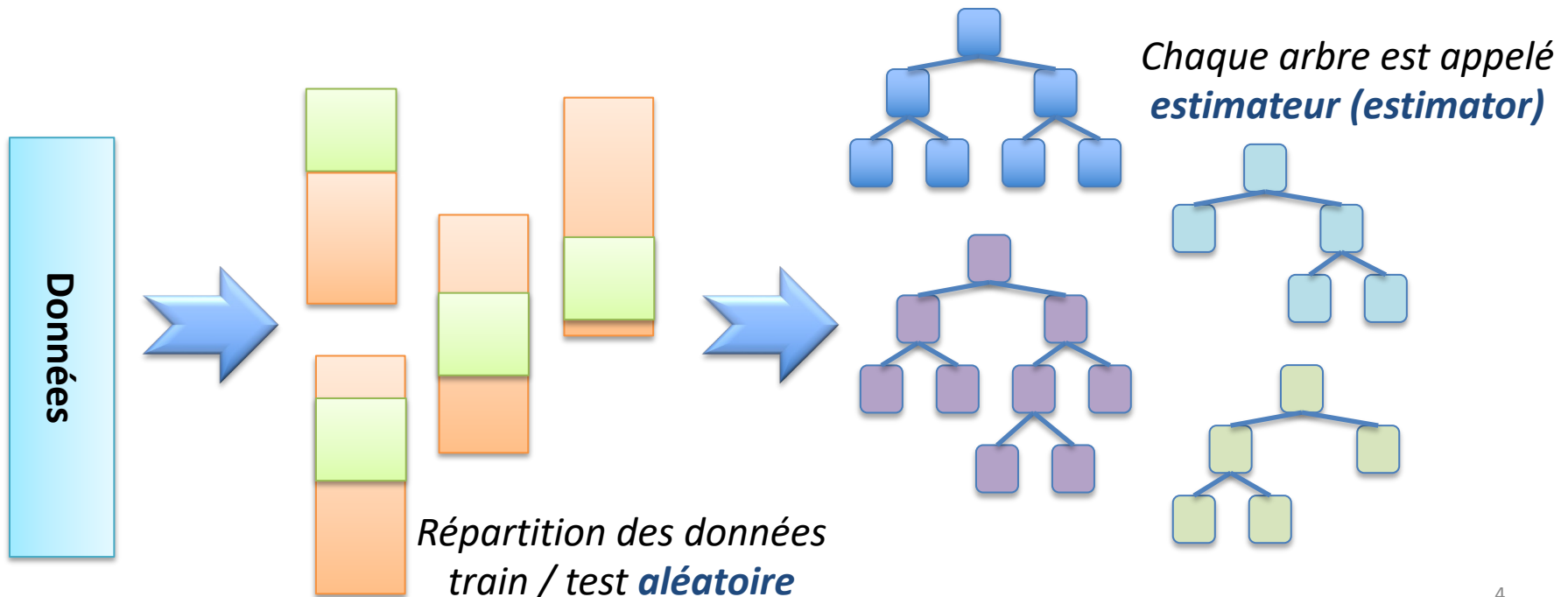
Stacking

- **Modèles** \neq provenant d'**algorithmes** \neq
- **Même** ensemble des **données**, mais **algo** \neq
- **Prédiction hiérarchique** : les prédictions des modèles alimentent un **métamodèle**, qui les agrège

Random Forest

- **Random Forest**

- Méthode **ensembliste** de type « **Bagging** »
- Créer **plusieurs arbres**, chacun à partir d'un **sous-ensemble des données** différents (choix **aléatoire**)
- **Prédiction** : **moyenne** des **prédictions** des modèles



Random Forest en Python



On n'oublie pas
les **import**

Méthode de classification

Bibliothèque des
méthodes ensemblistes

```
from sklearn.ensemble import RandomForestClassifier
```

Nombre d'**arbres**
(**estimateurs**) à créer

```
foret = RandomForestClassifier ( n_estimators=50 )
```

Entraînement du modèle

Création de l'**objet** qui
contiendra notre **forêt**.

Option (reproductibilité) :
random_state = 42

```
foret.fit ( X_train, Y_train )
```


Données d'**entraînement**
(**features** X et **target** Y)

```
pred_foret = foret.predict ( X_test )
```

Données de test
(**testing features**)

Usage de la forêt pour la
prédiction (**classification**)

Random Forest en Python



On peut connaître
l'**importance globale**
de chaque **feature**


foret.**feature_importances_**

*Liste avec les noms des
features*

*Importance de ces
features*

```
for f,i in zip(features_names, foret.feature_importances_ ) :  
    print ( " {} : {:.4f} ".format(f,i))
```

```
age : 0.3826  
sex_male : 0.4693  
passengerClass_1st : 0.0574  
passengerClass_2nd : 0.0156  
passengerClass_3rd : 0.0751
```

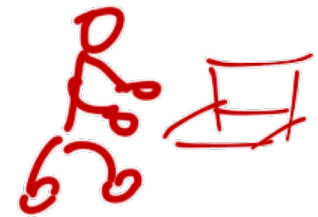


On peut accéder à
chaque **arbre**
(**estimator**)

foret.**estimators_**

*Liste contenant
toutes les arbres*

```
for tree in foret.estimators_ :  
    print ( export_text(tree, feature_names=features_names,  
                        spacing=3, decimals=2))
```



• Dataset Titanic

- Lire le fichier « **TitanicSurvival.csv** » et observer le DataFrame avec **head** et **info**

On n'oublie pas la bibliothèque **Pandas**

```
import pandas as pnd
```

```
titanic = pnd.read_csv(
    'http://www.kirschpm.fr/cours/PythonDataScience/files/TitanicSurvival.csv')
titanic.info()
titanic.head(15)
```

	Unnamed: 0	survived	sex	age	passengerClass
0	Allen, Miss. Elisabeth Walton	yes	female	29.00	1st
1	Allison, Master. Hudson Trevor	yes	male		
2	Allison, Miss. Helen Loraine	no	female		
3	Allison, Mr. Hudson Joshua Crei	no	male		
4	Allison, Mrs. Hudson J C (Bessi	no	female		
5	Anderson, Mr. Harry	yes	male		

*Plusieurs soucis
observés sur les données*

RangeIndex: 1309 entries, 0 to 1308

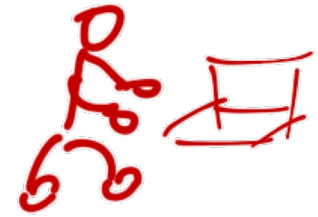
Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	Unnamed: 0	1309 non-null	object
1	survived	1309 non-null	object
2	sex	1309 non-null	object
3	age	1046 non-null	float64
4	passengerClass	1309 non-null	object

dtypes: float64(1), object(4)



Hands On !



- **Dataset Titanic**

- Corriger les défauts

- Nom 1ère colonne avec **rename**

Dictionnaire
{ ancien nom : nouveau nom }

*inplace = True pour
modifier le DataFrame*

```
titanic.rename(columns={'Unnamed: 0': 'passenger'}, inplace=True)
```

- Compléter les valeurs de l'âge avec **fillna**

```
titanic['age'].fillna(titanic['age'].mean(), inplace=True)
```

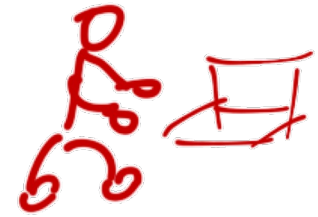
On remplit avec la moyenne

```
titanic.head(10)
```

	passenger	survived
0	Allen, Miss. Elisabeth Walton	yes
1	Allison, Master. Hudson Trevor	yes
2	Allison, Miss. Helen Loraine	no
3	Allison, Mr. Hudson Joshua Crei	no
4	Allison, Mrs. Hudson J C (Bessi	no

```
titanic['age'].describe()
```

count	1309.00
mean	29.88
std	12.88
min	0.17
25%	22.00
50%	29.88
75%	35.00
max	80.00
Name: age, dtype: float64	



• Dataset Titanic

- Corriger les défauts : **encoding**
convertir valeurs symboliques en numérique

getdummies
équivalent Pandas du
OneHotEncoding

```
titanic = pnd.get_dummies (titanic, columns=['survived', 'sex'],  
                           drop_first=True)
```

titanic.head(15)

	passenger	age	passengerClass	survived_yes	sex_male
0	Allen, Miss. Elisabeth Walton	29.00	1st	1	0
1	Allison, Master. Hudson Trevor	0.92	1st	1	1
2	Allison, Miss. Helen Loraine	2.00	1st	0	0
3	Allison, Mr. Hudson Joshua Crei	30.00	1st	0	1

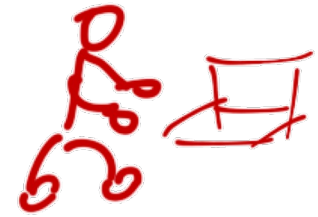
drop_first = True

0 équivaut à une classe

survived = 'no' → survived_yes = 0



Hands On !



- **Dataset Titanic**

- Corriger les défauts : **encoding**
convertir valeurs symboliques en numérique

```
titanic = pnd.get_dummies (titanic, columns=['passengerClass'])
```

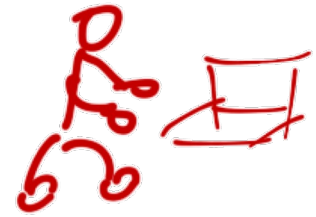
```
titanic.sample(15)
```



getdummies

*Chaque valeur de la colonne
passengerClass devient une colonne*

	passenger	age	survived_yes	sex_male	passengerClass_1st	passengerClass_2nd	passengerClass_3rd
224	Partner, Mr. Austen	45.50	0	1	1	0	0
1090	Oreskovic, Miss. Jelka	23.00	0	0	0	0	1
1144	Rice, Master. Eugene	2.00	0	1	0	0	1
1097	Palsson, Master. Paul Folke	6.00	0	1	0	0	1
861	Heininen, Miss. Wendla Maria	23.00	0	0	0	0	1



• Dataset Titanic

– On peut aussi vérifier l'équilibre du *dataset*

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1309 entries, 0 to 1308
```

```
Data columns (total 7 columns):
```

#	Column	Non-Null Count	Dtype
0	passenger	1309 non-null	object
1	age	1309 non-null	float64
2	survived_yes	1309 non-null	uint8
3	sex_male	1309 non-null	uint8
4	passengerClass_1st	1309 non-null	uint8
5	passengerClass_2nd	1309 non-null	uint8
6	passengerClass_3rd	1309 non-null	uint8

```
dtypes: float64(1), object(1), uint8(5)
```

```
memory usage: 27.0+ KB
```

```
titanic.groupby('survived_yes').size()
```

$\pm 62\%$ survived 'no'

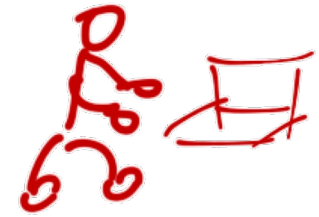
$\pm 38\%$ survived 'yes'

```
survived_yes
```

```
0      809
```

```
1      500
```

```
dtype: int64
```



• Dataset Titanic

– Séparer le *dataset* en **test** et **train**

L'option « **stratify** » permet de garder les mêmes proportions entre les classes que le **Y set** indiqué

```
train_titanic, test_titanic = train_test_split(titanic, test_size=0.3, stratify=titanic['survived_yes'] )
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 916 entries, 966 to 773
```

```
Data columns (total 7 columns):
```

#	Column	Non-Null Count	Dtype
0	passenger	916 non-null	object
1	age	916 non-null	float64
2	survived_yes	916 non-null	uint8
3	sex_male	916 non-null	uint8
4	passengerClass_1st	916 non-null	uint8
5	passengerClass_2nd	916 non-null	uint8
6	passengerClass_3rd	916 non-null	uint8

```
dtypes: float64(1), object(1), uint8(5)
```

```
memory usage: 25.9+ KB
```

```
train_titanic.info()
```

Répartition des données :

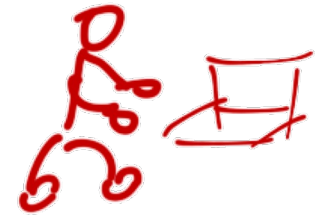
2/3 pour le **training**

1/3 pour le **test**

```
train_titanic.groupby('survived_yes').size()
```

	survived_yes
± 62 % survived 'no'	0 566
± 38% survived 'yes'	1 350

dtype: int64



• Dataset Titanic

– Séparer le dataset en **test** et **train**

```
train_titanic, test_titanic = train_test_split(titanic, test_size=0.3,  
                                              stratify=titanic['survived_yes'] )
```

– Séparer les **features (X)** et le **target (Y)**

```
features_names = ['age', 'sex_male', 'passengerClass_1st',  
                  'passengerClass_2nd', 'passengerClass_3rd']
```

```
X_train = train_titanic[features_names]  
X_test = test_titanic[features_names]
```

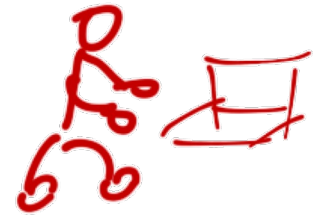
```
Y_train = train_titanic['survived_yes']  
Y_test = test_titanic['survived_yes']
```

```
920    0  
4      0  
452    1  
304    1
```

Y_train[16:30]

```
Int64Index: 393 entries, 1273 to 629  
Data columns (total 5 columns):  
#   Column                Non-Null Count  Dtype  
---  ---  
0   age                   393 non-null   float64  
1   sex_male              393 non-null   uint8  
2   passengerClass_1st    393 non-null   uint8  
3   passengerClass_2nd    393 non-null   uint8  
4   passengerClass_3rd    393 non-null   uint8
```

X_test.info()



- **Dataset Titanic**

- Création du modèle et entraînement

```
from sklearn.ensemble import RandomForestClassifier
```

```
foret = RandomForestClassifier (n_estimators=15, max_depth=8)
```

```
foret.fit(X_train, Y_train)
```

*Nombre d'estimateurs
(arbres) dans la forêt*

*Si on veut limiter la
complexité des arbres
générées*

age : 0.3719

sex_male : 0.4617

passengerClass_1st : 0.0477

passengerClass_2nd : 0.0079

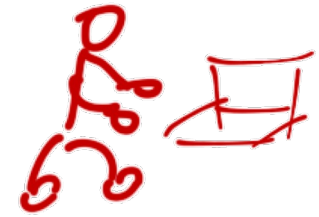
passengerClass_3rd : 0.1107

Suggestion :

Si on veut afficher les **features names**

```
for f,i in zip(features_names,  
               foret.feature_importances_) :  
  
    print (" {} : {:.4f} ".format(f,i))
```

Hands On !



- **Dataset Titanic**

- Test et évaluation du modèle

```
from sklearn.metrics import confusion_matrix
```

```
pred_foret = foret.predict(X_test)
```

Prédiction par l'ensemble de la forêt

```
confusion_matrix (Y_test, pred_foret)
```

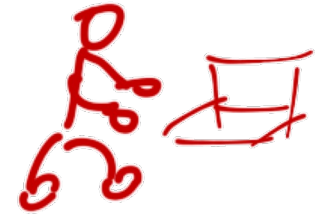
*On compare les données de test
(Y_test) avec les prédictions*

```
array([[210, 33],  
       [ 52, 98]])
```

Classe prédite par le modèle

Classe réelle	Classe prédite par le modèle	
	0 survived = No	1 survived = Yes
0	210	33
1	52	98

Hands On !



- **Dataset Titanic**

- Test et évaluation du modèle

```
from sklearn.metrics import classification_report

print (classification_report(Y_test, pred_foret))
```

On compare à l'aide de plusieurs indicateurs

	precision	recall	f1-score	support
0	0.80	0.86	0.83	243
1	0.75	0.65	0.70	150
accuracy			0.78	393
macro avg	0.77	0.76	0.76	393
weighted avg	0.78	0.78	0.78	393



F1 Score :

Score combinant la **précision**
et le **rappel**

$$F1 = \frac{2 * \text{précision} * \text{rappel}}{\text{précision} + \text{rappel}}$$