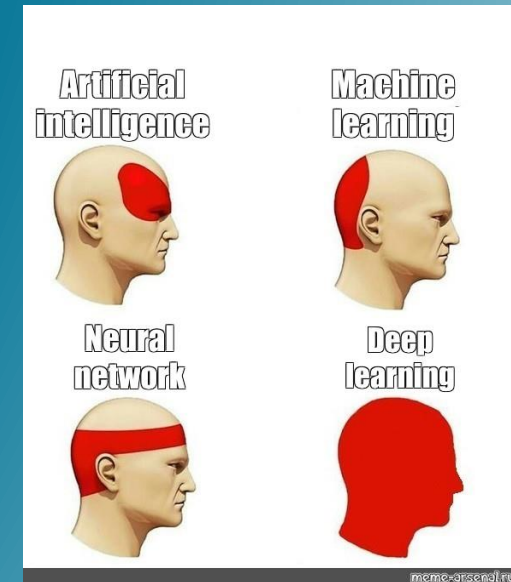


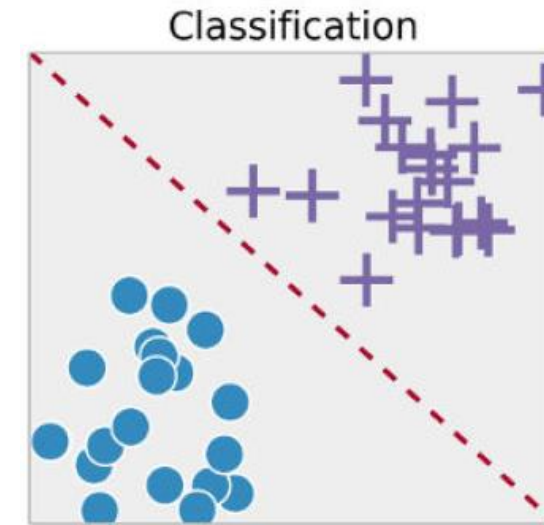
Méthodes Numériques et Modélisation

Fondements de l'intelligence artificielle et applications aux sciences atmosphériques



CLASSIFICATION

- Associer à chaque donnée un label (étiquette) parmi un ensemble de labels possibles
- Classification binaire
 - Deux classes -> on peut faire un "est-ce X" oui ou non
- Classification multiclasse
 - Plusieurs méthodes
 - Binaire pour chaque classe
 - "vrai" multiclasse -> probabilité générale
- Cas particuliers
 - Classification multilabel
 - Une donnée peut appartenir à plusieurs classes
 - Détection d'objets
 - Classer des éléments d'une image (et pas toute l'image)
 - Segmentation
 - Chaque pixel de l'image sera classé (puis associés)



Classification

Prédire une classe (qualitative, discrète)



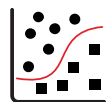
This is a cat



This is a rabbit



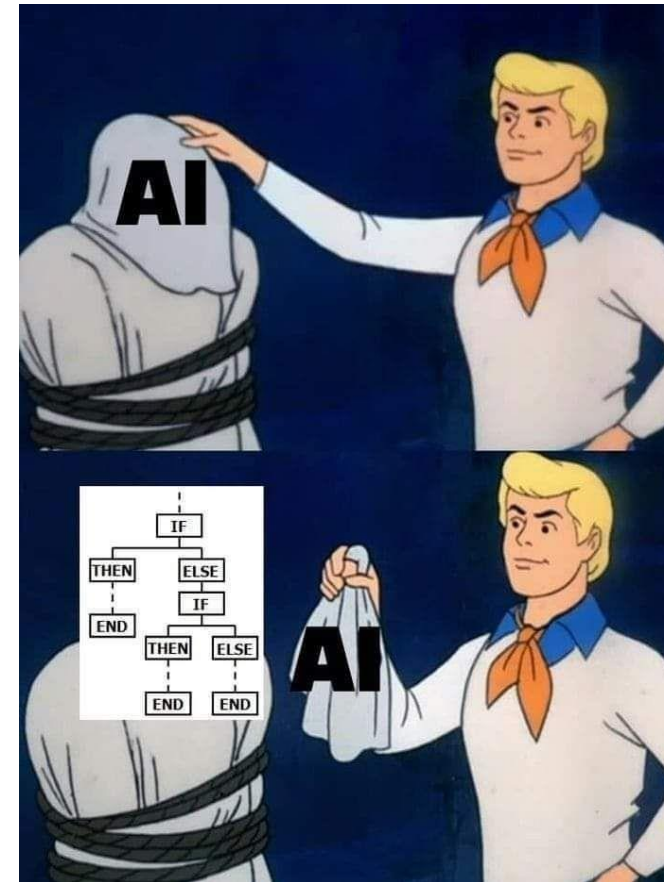
Tell me,
what is it ?



ALGORITHMES POUR LA CLASSIFICATION SUPERVISÉE

Plusieurs algorithmes:

- ☐ Arbre de décision
- ☐ Forêts aléatoires
- ☐ XGBoost
- ☐ KNN (K Nearest Neighbors)
- ☐ Classification naïve bayésienne
- ☐ Régression logistique
- ☐ Machine à vecteurs de support (SVM)
- ☐ Réseau de neurones



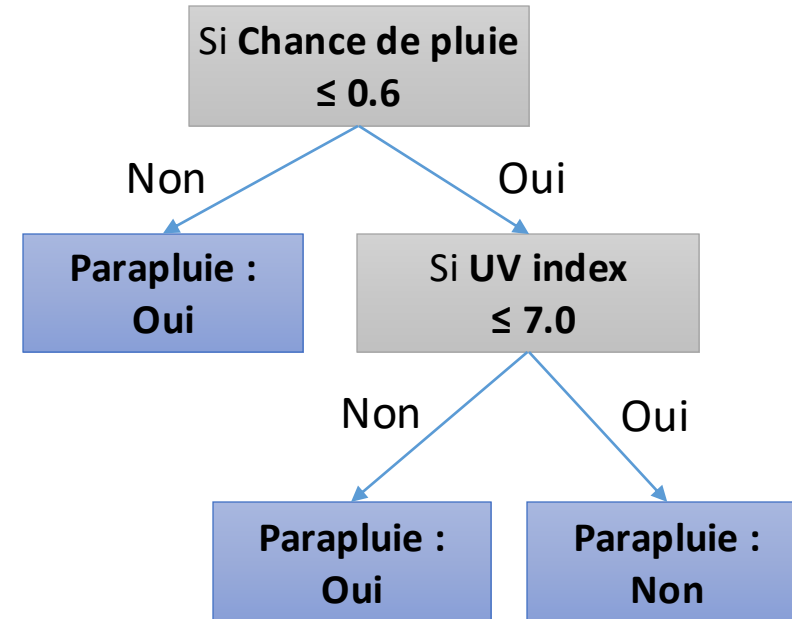
ARBRES DE DÉCISION

■ Bases

- Méthode de classification simple
- Méthode supervisée (données étiquetées)
- Comparable à un enchainement de « questions » (conditions)
if – else

| Features (variables) « X set » | | Target (classes) « Y set » |
|-----------------------------------|----------|-------------------------------|
| Chance de pluie | UV Index | Parapluie |
| 0.1 | 11 | True |
| 0.9 | 1 | True |
| 0.3 | 3 | False |
| 0.1 | 2 | False |

Dois-je prendre un parapluie ?



- La méthode cherche **l'ordre** des « questions » de manière à **minimiser** la taille de **l'arbre**

ARBRES DE DÉCISION EN PYTHON

```
from sklearn.tree import DecisionTreeClassifier
```

Méthode de classification

```
clf = DecisionTreeClassifier()
```

Création de l'**objet** qui contiendra notre **arbre de décision**.
C'est lui qu'on va manipuler pour créer l'arbre et l'utiliser.

L'opération « **fit** » réalise l'**entraînement** du modèle (« **training** » ou « **fitting** »)

```
clf.fit ( x_train[feature_names], y_train[target] )
```

Pour l'entraînement, l'opération « **fit** » a besoin de deux **DataFrames** ...

Données d'entraînement
(**training features**)

Classes (**target**) pour les
données de training

```
y_pred = clf.predict(x_test[feature_names])
```

Données de test
(**testing features**)

Une fois entraîné, le modèle peut être **testé**, puis utilisé pour la **classification**...

■ Exemple

■ Créer le DataFrame de *training*

```
dfUmbrella = pd.DataFrame (
    { 'Chance Rain':[0.1, 0.9, 0.3, 0.1, 0.8],
      'UV Index': [11, 1, 3, 2, 2] ,
      'Umbrella' : [True, True, False, False, True] } )
```

- Identifier dans deux variables les *features* et le *target*

```
feature_names = ['Chance Rain', 'UV Index']
target = 'Umbrella'
```

```
x_test = pd.DataFrame (
    { 'Chance Rain':[0.5, 0.2],
      'UV Index': [5, 8] } )
```

- Créer son objet **DecisionTreeClassifier**

```
clf = DecisionTreeClassifier()
```

- Entraîner son modèle

```
clf.fit ( dfUmbrella[feature_names] , dfUmbrella[target] )
```

- Tester son modèle

```
y_pred = clf.predict(x_test)
```

```
print (y_pred)
```

```
[False  True]
```

ARBRES DE DÉCISION EN PYTHON

- Comment visualiser son modèle ?
 - L'arbre de décision créé peut être visualisé

Mode texte
export_text

```
from sklearn.tree import export_text
```

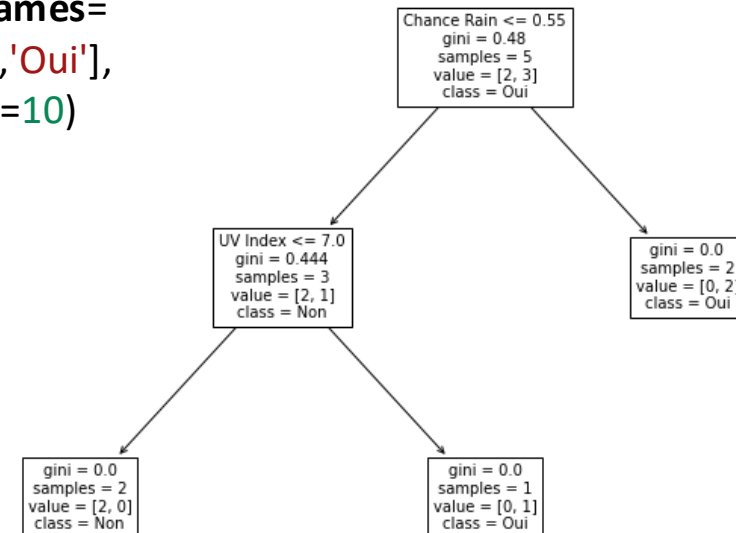
```
texte = export_text(clf,  
                    feature_names=feature_names,  
                    spacing=3, decimals=2)
```

```
print ( texte )  
  
|--- Chance Rain <= 0.55  
|   |--- UV Index <= 7.00  
|   |   |--- class: False  
|   |--- UV Index > 7.00  
|   |   |--- class: True  
|--- Chance Rain > 0.55  
|   |--- class: True
```

Mode graphique (MatPlot)
plot_tree

```
from sklearn.tree import plot_tree
```

```
plot_tree(clf,  
          feature_names=feature_names,  
          class_names=  
              ['Non', 'Oui'],  
          fontsize=10)
```




ARBRES DE DÉCISION EN PYTHON

- Quelle est l'importance de chaque *feature* ?
 - Certains *features* peuvent **contribuer plus** à la décision que d'autres
 - On peut connaître le niveau d'importance des *features* dans un l'arbre entraîné
 - Attribut `feature_importances_`

```
pnd.DataFrame (  
    { 'feature_names' : feature_names ,  
      'importance' : clf.feature_importances_  
    })
```

Création d'un DataFrame avec les
features et leur *importance*
(juste pour **visualiser** plus facilement)

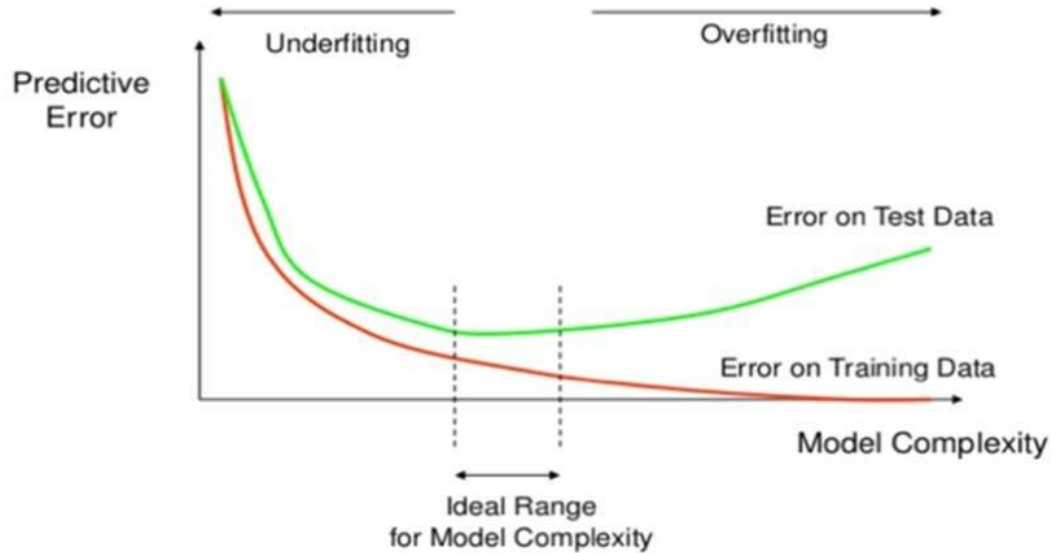
Chaque *feature* va avoir
un niveau d'*importance*
entre 0 et 1.



| | feature_names | importance |
|---|---------------|------------|
| 0 | Chance Rain | 0.444444 |
| 1 | UV Index | 0.555556 |

L'importance indique le niveau d'influence
d'une variable dans la décision

OÙ S'ARRÊTER ?



- Critère d'arrêt

- Par défaut, les algorithmes ne s'arrêtent que lorsque une feuille est pure ou ne contient qu'un seul enregistrement
 - Cela mène souvent au surapprentissage
- Il est possible de changer le critère
 - Profondeur maximale (`max_depth`)
 - Nombre minimal d'enregistrements d'un nœud pour le couper (`min_samples_split`)
 - Nombre minimal d'enregistrements dans une feuille (`min_samples_leaf`)
 - Nombre maximal de feuilles (`max_leaf_nodes`)
 - ...

RÉSUMÉ ARBRES DE DÉCISION

Avantages :

- Facilement interprétables
- Complexité algorithmique faible
- Très facile à transposer sur un programme (suite de if-then-else)

Inconvénients :

- Hypothèse de non-corrélation entre attributs
 - les nœuds ne peuvent tester qu'un seul attribut
- Impossibilité de classer un nouvel exemple dont certains attributs sont inconnus ou imprécis
- Tombe facilement dans le cas du **sur-apprentissage**

MAIS POURQUOI UN SEUL ARBRE ?

- 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'un seul 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



Jordan Goldmeier
@Option_Explicit

Where do data scientists go
camping?

In random forests

- 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

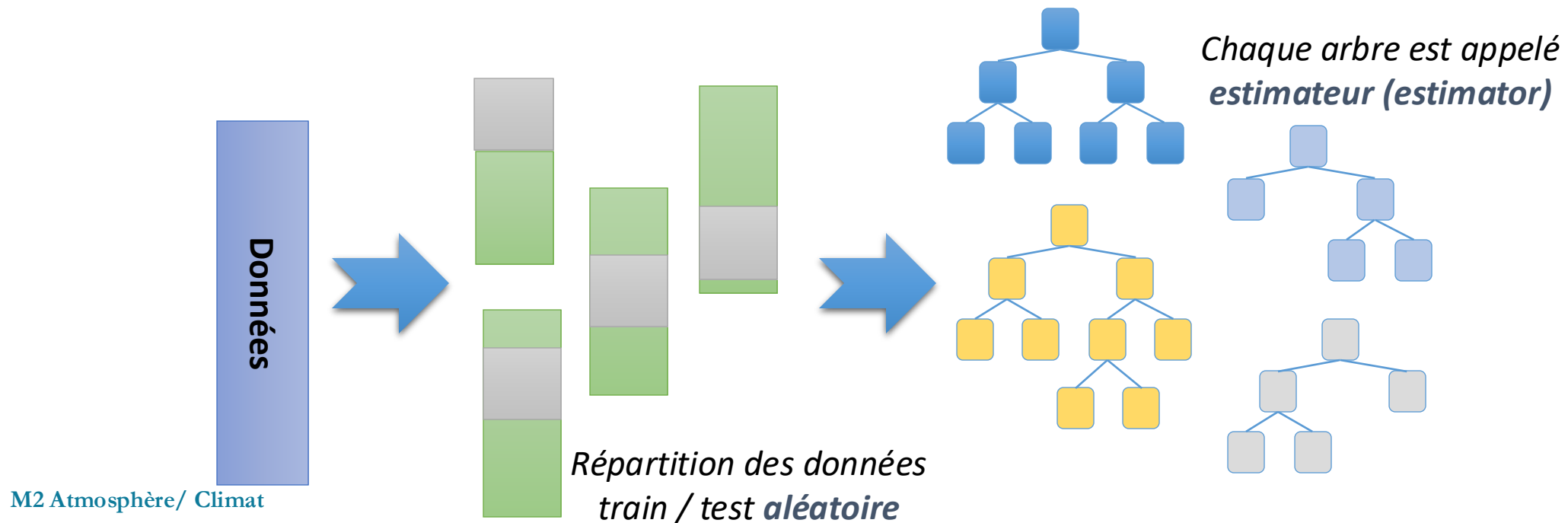
Stacking

- **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
- Il est possible d'accéder à chaque arbre individuellement



Random Forest en Python

*Bibliothèque des
méthodes ensemblistes*

Méthode de classification

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

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

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

***Entraînement** du modèle*

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

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

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

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

*Données de test
(**testing features**)*



On n'oublie pas
les **import**

*Création de l'**objet** qui
contiendra notre **forêt**.
Option (reproductibilité) :
random_state = 42*

XGBOOST (EXTREME GRADIENT BOOST)

- XGBoost

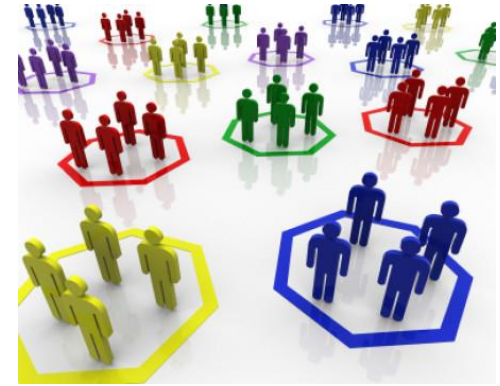
- Méthode **ensembliste** de type « **Boosting** »
- Très populaire, souvent plus efficace que les réseaux de neurones pour les données tabulaires
- Créer **plusieurs arbres**, séquentiellement
 - Chaque fois, les données sont choisies afin de maximiser la classification des résultats erronés
- Prédiction : par vote (majorité) ou moyenne des prédictions des modèles



Autres algorithmes






L'ALGORITHME KNN (K-NEAREST NEIGHBORS)

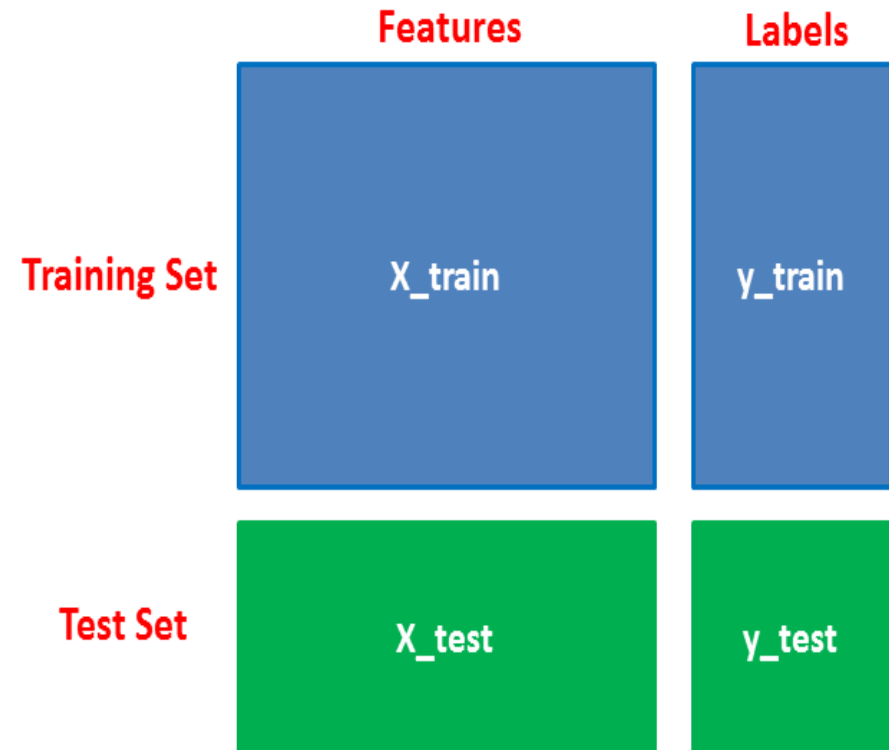
- Regarder la classe des k exemples les plus proches de l'individu à classer, ensuite lui affecter la classe majoritaire
- Proches ???
- On dit que deux individus sont (+|-) proches s'il y a une (petite | grande) distance entre eux
 - À nous de décider comment calculer cette distance









- Se base sur le proverbe:
“Dis moi qui tu fréquentes,
je te dirais qui tu es!”

DÉCOUPAGE EN ENSEMBLE D'ENTRAÎNEMENT / TEST

| X (features) | | | | Y (label) |
|--|-----|--------|------------------|-----------|
| Customer | Age | Income | No. credit cards | Loyal |
| John  | 35 | 35K | 3 | No |
| Rachel  | 22 | 50K | 2 | Yes |
| Hannah  | 63 | 200K | 1 | No |
| Tom  | 59 | 170K | 1 | No |
| Nellie  | 25 | 40K | 4 | Yes |



L'ALGORITHME KNN (K-NEAREST NEIGHBORS)

| Customer | Age | Income | No. credit cards | Loyal |
|---|-----|--------|------------------|-------|
| John  | 35 | 35K | 3 | No |
| Rachel  | 22 | 50K | 2 | Yes |
| Hannah  | 63 | 200K | 1 | No |
| Tom  | 59 | 170K | 1 | No |
| Nellie  | 25 | 40K | 4 | Yes |
| David  | 37 | 50K | 2 | ? |

Distance (David , indiv i)

$$\text{Distance (David, John)} = \sqrt{[(35-37)^2 + (35-50)^2 + (3-2)^2]} = \mathbf{15,16}$$

$$\text{Distance (David, Rachel)} = \sqrt{[(22-37)^2 + (50-50)^2 + (2-2)^2]} = \mathbf{15}$$

$$\text{Distance (David, Hannah)} = \sqrt{[(63-37)^2 + (200-50)^2 + (1-2)^2]} = \mathbf{152,16}$$

$$\text{Distance (David, Tom)} = \sqrt{[(59-37)^2 + (170-50)^2 + (1-2)^2]} = \mathbf{122}$$







$$\text{Distance (David, Nellie)} = \sqrt{[(25-37)^2 + (40-50)^2 + (4-2)^2]} = \mathbf{15,74}$$

L'algorithme KNN (K-Nearest Neighbors)

➤ Les 3 plus proches sont:

- John de la classe **NO**
- Rachel de la classe **Yes**
- Nellie de la classe **Yes**

➤ Affecter David à la classe la plus fréquente « **YES** »

| Customer | Age | Income | No. credit cards | Loyal | Dist |
|---|-----|--------|------------------|-------|-------|
| John  | 35 | 35K | 3 | No | 15,16 |
| Rachel  | 22 | 50K | 2 | Yes | 15 |
| Hannah  | 63 | 200K | 1 | No | 152,1 |
| Tom  | 59 | 170K | 1 | No | 122 |
| Nellie  | 25 | 40K | 4 | Yes | 15,74 |
| David  | 37 | 50K | 2 | Yes | |

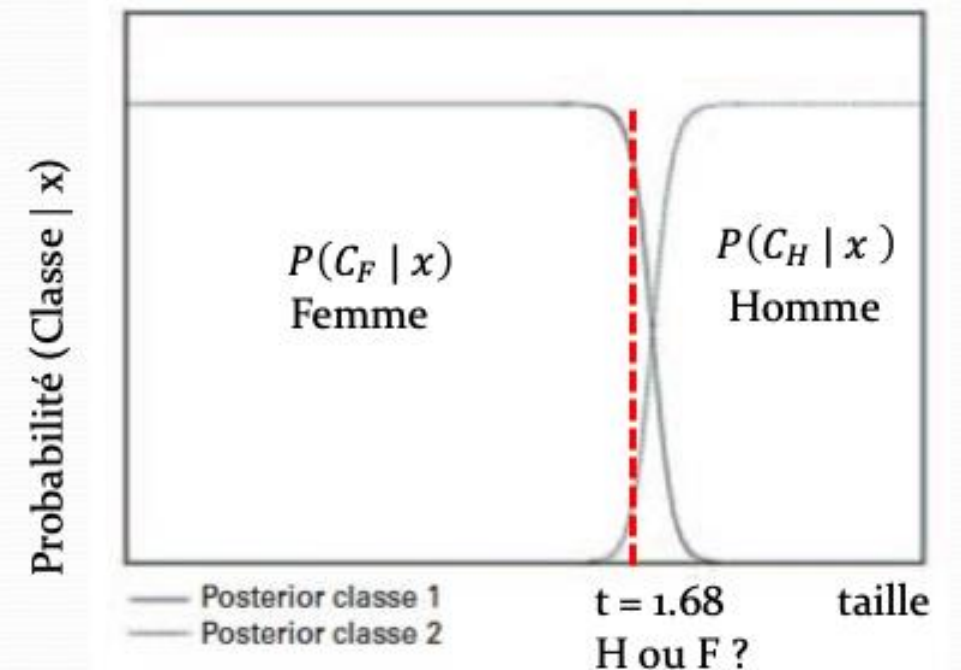
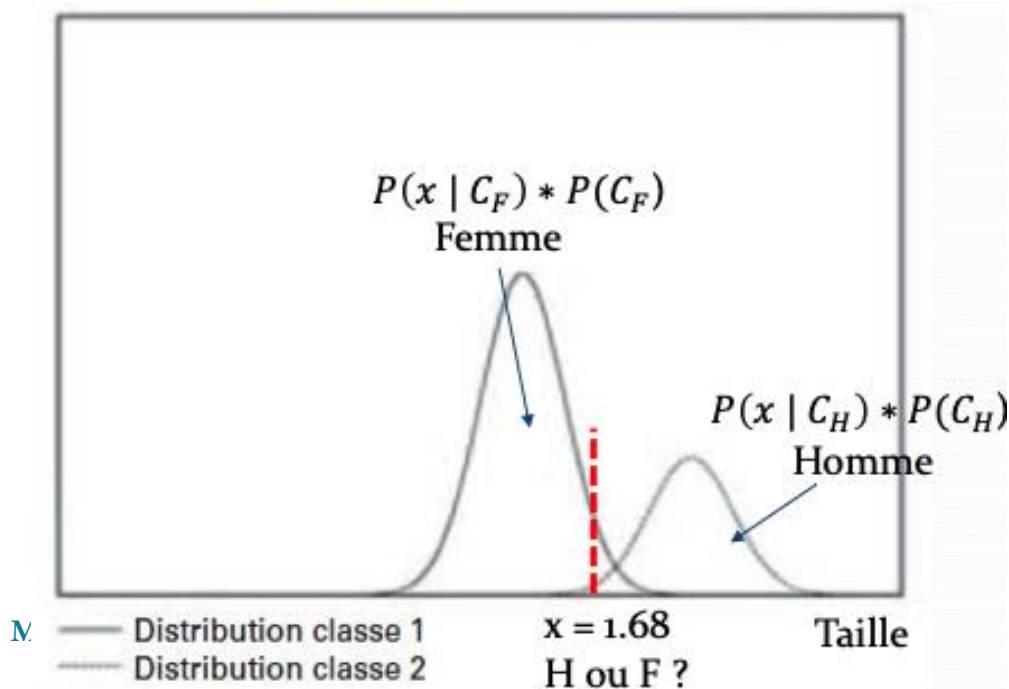
CLASSIFICATION NAÏVE BAYÉSIENNE

- Cet algorithme utilise la probabilité de la répartition des données entre différentes classes
 - La méthode est dite "naïve" car elle suppose que les variables sont indépendantes
 - Ça reste néanmoins une méthode très performante
- Ex : prédire le sexe selon la taille d'un individu

Règle de Bayes :

$$P(C_k | x) = \frac{P(x | C_k) * P(C_k)}{P(x)}$$

$$P(C_H | 1.68) = \frac{P(1.68 | C_H) * P(C_H)}{P(1.68)}$$



Métriques de qualité

COMMENT ÉVALUER UN MODÈLE ?





- Dans une classification, on veut savoir si notre modèle fait bien son boulot
- Dans le cas d'une **classification binaire**, on peut faire usage d'une **matrice de confusion**
- Ex : est-ce que la valeur X appartient à la classe "1" ?

| | | Classe réelle | |
|----------------|---|--|--|
| | | - | + |
| Classe prédite | - | True Negatives <i>(vrais négatifs)</i> | False Negatives <i>(faux négatifs)</i> |
| | + | False Positives <i>(faux positifs)</i> | True Positives <i>(vrais positifs)</i> |

| Nom du cas | Abréviation | Description |
|--------------|-------------|--|
| Vrai positif | VP | La donnée appartient à la classe "1" et a été prédite comme tel |
| Vrai négatif | VN | La donnée appartient à la classe "0" et a été prédite comme tel |
| Faux positif | FP | La donnée appartient à la classe "0" mais a été prédite comme classe "1" -> Erreur type 1 |
| Faux négatif | FN | La donnée appartient à la classe "1" mais a été prédite comme classe "0" -> Erreur type 2 |

IMPACT D'UNE ERREUR

- On voit qu'il y a deux types d'erreur dans une classification binaire
- Erreur type 1 – faux positif
- Erreur type 2 – faux négatif
- En général, une erreur type 1 est moins grave qu'une type 2
- Ex : recherche d'une maladie
 - En cas de faux-positif, les tests cliniques / médecin permettront de vérifier la présence de la maladie
 - En cas de faux-négatif, le patient est renvoyé sans traitement
- Ex : recherche de défauts dans une chaîne de montage
 - FP -> la pièce est écartée puis sera recyclée/jetée
 - FN -> le produit aura une pièce défectueuse

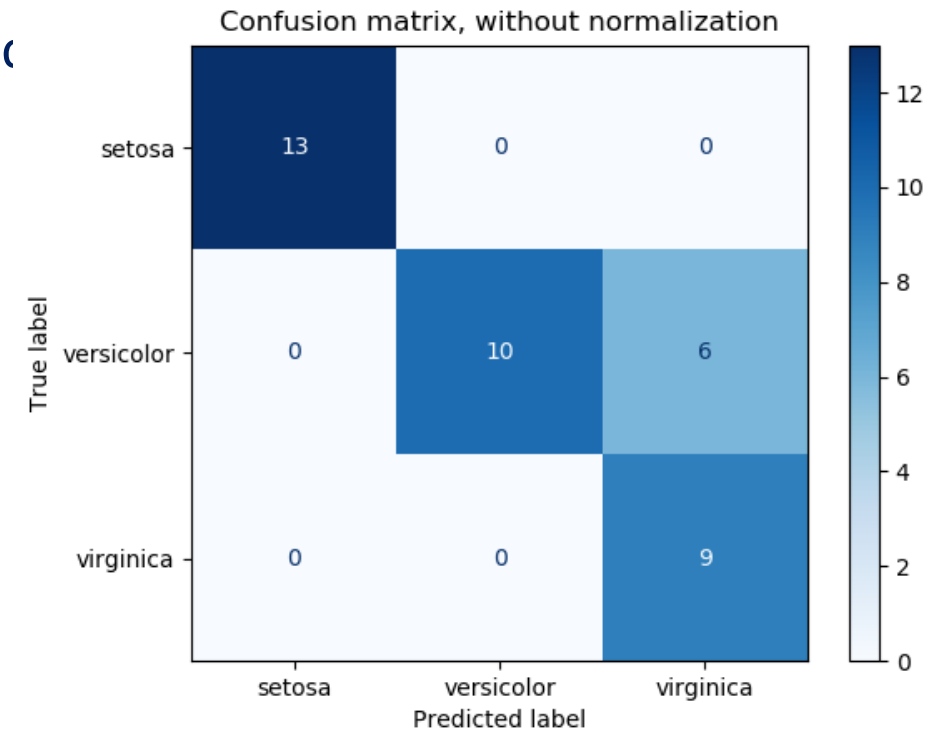
| | | Actual Values | |
|------------------|---|--|---|
| | | 1 | 0 |
| Predicted Values | 1 | TRUE POSITIVE  | FALSE POSITIVE  TYPE 1 ERROR |
| | 0 | FALSE NEGATIVE  TYPE 2 ERROR | TRUE NEGATIVE  |

COMMENT ÉVALUER UN MODÈLE ?

- Dans le cas d'une **classification multiclasse**, on peut également faire usage d'une **matrice de confusion**
- Ceci permet de voir combien d'éléments sont classés (chaque classe)

```
1 print(confusion_matrix(y_test, ypred))  
2
```

```
[[23  0  0]  
 [ 0 19  0]  
 [ 0  1 17]]
```



INDICATEURS DÉRIVÉS DE LA MATRICE DE CONFUSION

- **Accuracy** : un pourcentage à maximiser
 - Désigne la proportion des prédictions correctes effectuées par le modèle

$$\text{Accuracy} = \frac{\text{nombre de prédictions correctes}}{\text{Nombre total de prédictions}} = \frac{VN+VP}{VN+VP+VN+VP}$$

```
1 from sklearn.metrics import accuracy_score # Evaluation
2 print ('KNN accuracy score')
3 print (accuracy_score(y_test, ypred))
```

```
KNN accuracy score
0.9833333333333333
```

- Sur une classification multiclasse, l'Accuracy est calculé en prenant la somme des bonnes prédictions (diagonale) divisé par le nombre de cas.
- Il n'y a pas une valeur Accuracy de référence. Selon le cas, 60% peut être bon (ex : identifier des clients avec potentiel d'investissement), alors que dans d'autres cas il faut chercher le 99,99% (identification de piétons dans une voiture autonome)
 - Note : en français la traduction la plus proche serait "précision", mais il y a une autre métrique *precision*. Certains travaux utilisent "justesse" ou "exactitude", mais c'est **Accuracy** qui reste le plus répandu

INDICATEURS DÉRIVÉS DE LA MATRICE DE CONFUSION

- La métrique Accuracy a trois défauts majeurs:
 1. Ne donne aucune information sur les erreurs commises
 - L'impact n'est pas le même si c'est type 1 ou type 2
 2. Peu informative si les classes sont déséquilibrées
 1. Ex : détection de défauts où seulement 1% des pièces sont défectueuses
 2. Accuracy 99% -> soit il détecte une partie des défauts (type 2), soit il considère que toutes sont bonnes (type 1)
 3. Accuracy 100% -> quasiment impossible, à se méfier !
 1. Surapprentissage ?
 2. Données val/test trop proches des données train ?
 3. Des informations qui "fuitent" la classe ?



RAPPEL ET PRÉCISION

- Le rappel (Recall) est l'un des critères permettant d'évaluer la sensibilité du modèle
 - Proportion de la classe X détectée vraiment

$$\text{Rappel} = \frac{\text{nombre de vrai positifs}}{\text{Nombre de vrai positifs} + \text{faux négatifs}} = \frac{TP}{TP + FN}$$

- Un grand recall indique que presque tous les éléments de la classe cible ont été détectés
 - Plus le recall est grand, moins d'erreurs de type 2 (faux négatifs), qui sont les plus graves
- La précision (Precision) indique la proportion de vrais positifs dans l'ensemble de positifs détectés

$$\text{Precision} = \frac{\text{nombre de vrai positifs}}{\text{Nombre de vrai positifs} + \text{faux positifs}} = \frac{TP}{TP + FP}$$

- Permet d'estimer le pourcentage d'erreurs de type 1

RAPPEL ET PRÉCISION DANS SKLEARN

- ScikitLearn a deux fonctions pour ça : `recall_score()` et `precision_score()`
`sklearn.metrics.recall_score(test_y, pred_y)`
> 0.962929292929
`sklearn.metrics.precision_score(test_y, pred_y)`
> 0.972222222222
- Par défaut, ça ne marche que sur le cas binaire (paramètre "average=binary" par défaut)
- Pour le multiclass, on doit indiquer comment les métriques sont calculées
 - `average=micro` : calcule la métrique au niveau global, avec le nombre total de FP et FN
 - `average=macro` : calcule la métrique pour chaque classe puis fait une moyenne arithmétique
 - `average=weighted` : la moyenne des classes est pondérée par rapport à leur "importance" (# de valeurs)
 - `average=None` : calcule la métrique pour chaque classe puis les affiche individuellement
- Il y a aussi une fonction `classification_report()` qui imprime les différentes variations

LE SCORE F1

- Le F1-score fait la moyenne harmonique entre precision et recall
 - Permet d'éviter les pièges de l'accuracy
 - Favorise les cas où nous avons une haute precision et haute recall

$$F1 = 2 * \frac{precision * rappel}{precision + rappel}$$

```
1 from sklearn.metrics import f1_score, precision_score, recall_score # Evaluation
2 print ('KNN recall score')
3 print (recall_score(y_test, ypred, average=None))
4 print ('KNN precision score')
5 print (precision_score(y_test, ypred, average=None))
6 print ('f1 score')
7 print (f1_score(y_test, ypred, average=None))
```

KNN recall score
[1. 1. 0.94444444]
KNN precision score
[1. 0.95 1.]
f1 score
[1. 0.97435897 0.97142857]

SENSIBILITÉ ET SPÉCIFICITÉ

- Beaucoup moins utilisés, nous avons aussi d'autres métriques pour des cas spéciaux (médecine, par exemple)
- La sensibilité est la capacité à prédire la classe positive quand c'est effectivement le cas
 - Formule similaire au **recall**
- La spécificité est la capacité à donner un résultat négatif pour un cas négatif

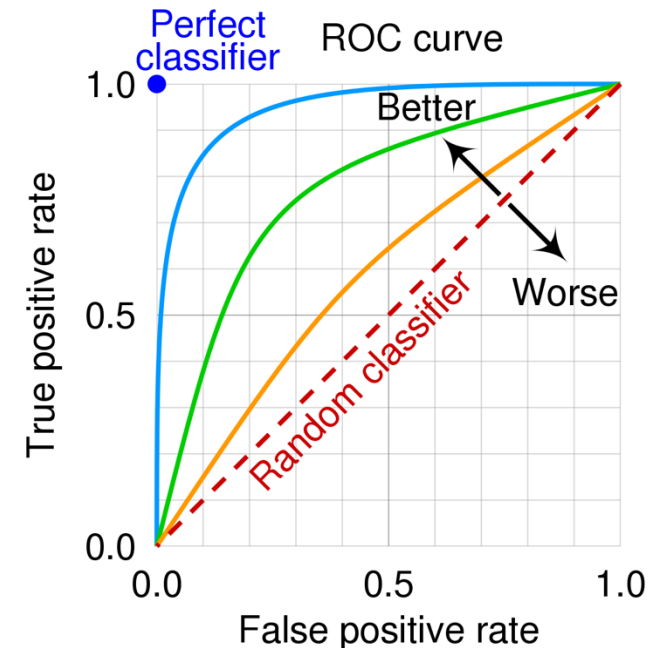
$$\text{Sensibilité} = \frac{TP}{TP + FN}$$

$$\text{Spécificité} = \frac{TN}{TN + FP}$$

- Si l'un des deux indicateurs est faible, alors le modèle n'est pas fiable
 - Une sensibilité faible ne permet pas de décider sur un résultat positif
 - Une spécificité faible ne permet pas de décider sur un résultat négatif
- Ex : un test de grossesse de pharmacie a une forte sensibilité mais une faible spécificité
 - Un résultat positif est presque sûr l'indice d'une grossesse
 - Un résultat négatif n'est pas garanti, des tests complémentaires peuvent être nécessaires

LA COURBE ROC ET L'AUC

- Souvent, les modèles estiment une probabilité d'appartenance à une classe :
 - Ex Titanic : passager 33 -> Survie 76%, Décès 24%
 - En ScikitLearn, on utilise la fonction `predict_proba()` au lieu de `predict()`
- En générale, la matrice de confusion utilise un seuil 0,5
 - Valeurs supérieures à 0,5 sont classés positivement, sinon négativement
- Lors du déploiement du modèle, on peut choisir des seuils différents en fonction des erreurs à minimiser
- **La courbe ROC** permet de visualiser l'impact des différents seuils
- Pour tracer la courbe ROC on a besoin de :
 - Le taux de vrais positifs (TPR) -> le rappel
 - Le taux de faux positifs (FPR) -> 1-specificité
- Idéalement, la courbe devrait monter rapidement pour avoir un fort TPR e un faible FPR
- L'AUC (Area Under Curve) est la surface sous la courbe
 - Idéalement la valeur serait 1, permet de comparer différents curves ROC indépendamment du seuil



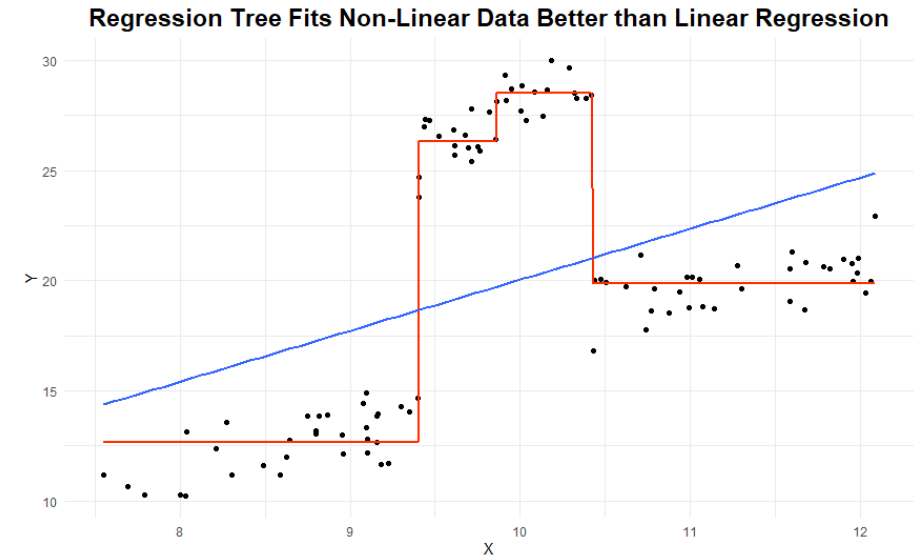
QUELLE MÉTRIQUE FAVORISER ?

- Tout dépend de l'objectif
- Une banque qui veut détecter des fraudes voudra un modèle qui n'en laisse passer rien
 - Quitte à avoir des suspicions, contacter le client et se tromper
- Une messagerie pourra laisser passer quelques spams mais devra éviter au maximum de classer en tant que spam un mail qui n'en n'est pas un
- Un algorithme de reconnaissance de cancer à partir de grains de beauté préférera se tromper et détecter un cancer
 - Le patient aura donc un examen supplémentaire ou un retrait du grain de beauté
 - C'est préférable au fait de laisser passer de véritables cancers

Quels algorithmes de classifications ?

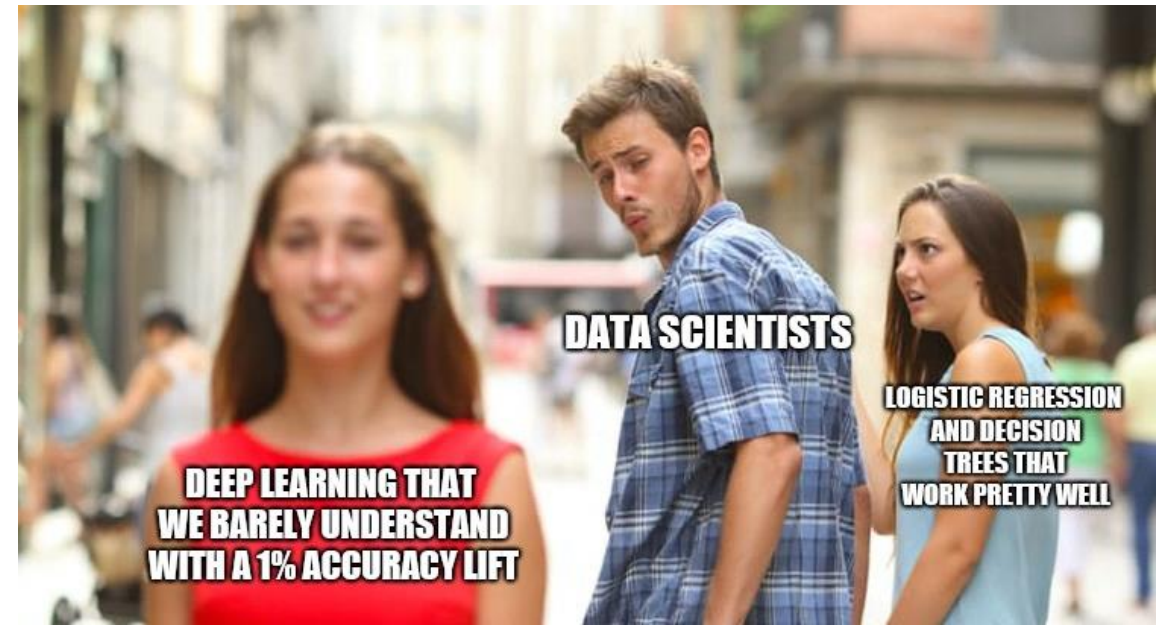
Plusieurs autres modèles de classification se prêtent au jeu

[sklearn.linear_model.HuberRegressor](#)
[sklearn.neighbors.RadiusNeighborsRegressor](#)
[sklearn.neural_network.MLPRegressor](#)
[sklearn.tree.DecisionTreeRegressor](#)
[sklearn.tree.ExtraTreeRegressor](#)
[sklearn.neighbors.KNeighborsRegressor](#)
[sklearn.ensemble.AdaBoostRegressor](#)
[sklearn.ensemble.BaggingRegressor](#)
[sklearn.ensemble.ExtraTreesRegressor](#)
[sklearn.ensemble.GradientBoostingRegressor](#)
[sklearn.ensemble.HistGradientBoostingRegressor](#)
[sklearn.ensemble.StackingRegressor](#)
[sklearn.ensemble.VotingRegressor](#)



EN RÉSUMÉ...

- On a commencé à regarder des algorithmes de classification
- Il n'y a pas un algorithme meilleur que l'autre
 - Ça dépend des jeux de données
 - Ça dépend des métriques qu'on veut favoriser
- Dans le prochain cours nous allons regarder plus en détail d'autres algorithmes, notamment les réseaux de neurones



QUELQUES EXERCICES

- Ouvrir lien suivant et suivre les liens des exercices

- <https://tinyurl.com/yrpunv8>

- ou

- [https://github.com/lsteffene1/M2Atmo et Climat/blob/main/README.md](https://github.com/lsteffene1/M2Atmo_et_Climat/blob/main/README.md)

- Cela vous amènera dans un environnement Google Colab