# [Evaluez les performances d'un modele de machine learning](https://openclassrooms.com/fr/courses/4297211-evaluez-les-performances-dun-modele-de-machine-learning)

Table des matières

[Evaluez les performances d'un modele de machine learning 1](#_Toc57708799)

[**Comprenez ce qui fait un bon modèle d’apprentissage** 5](#_Toc57708800)

[**Généralisation** 5](#_Toc57708801)

[**Sur-apprentissage et compromis biais-variance** 5](#_Toc57708802)

[**Les problèmes mal posés** 9](#_Toc57708803)

[**Ressources informatiques** 9](#_Toc57708804)

[**En résumé** 10](#_Toc57708805)

[**Mettez en place un cadre de validation croisée** 11](#_Toc57708806)

[**Jeux d’entraînement et de test** 11](#_Toc57708807)

[**Validation croisée** 12](#_Toc57708808)

[**Stratification** 13](#_Toc57708809)

[**Leave-one-out** 14](#_Toc57708810)

[**En résumé** 15](#_Toc57708811)

[**TP – Sélectionnez le nombre de voisins dans un kNN** 16](#_Toc57708812)

[**Sélection de modèle** 16](#_Toc57708813)

[**Recherche sur grille (*grid search*)** 17](#_Toc57708814)

[Sélectionnez le nombre de voisins dans un kNN sur un jeu de données 18](#_Toc57708815)

[**Les données** 18](#_Toc57708816)

[**Sélection de modèle** 21](#_Toc57708817)

[**Alternatives à la recherche exhaustive** 24](#_Toc57708818)

[**Résumé** 25](#_Toc57708819)

[Entraînez-vous : implémentez une validation croisée 26](#_Toc57708820)

[À vous de jouer ! 26](#_Toc57708821)

[Vérifiez votre travail 26](#_Toc57708822)

[Évaluez un algorithme de classification qui retourne des valeurs binaires 27](#_Toc57708823)

[Matrice de confusion 27](#_Toc57708824)

[Critères à optimiser et valeurs dérivées 28](#_Toc57708825)

[Exemple : test clinique 29](#_Toc57708826)

[Résumé 30](#_Toc57708827)

[Évaluez un algorithme de classification qui retourne des scores 31](#_Toc57708828)

[Certains modèles de classification retournent des valeurs réelles 31](#_Toc57708829)

[La courbe ROC 31](#_Toc57708830)

[Utiliser une courbe ROC 36](#_Toc57708831)

[Résumé 37](#_Toc57708832)

[Comparez votre algorithme à des approches de classification naïves 38](#_Toc57708833)

[La performance d'un modèle dépend du jeu de données 38](#_Toc57708834)

[Approches naïves pour des problèmes de classification (binaire) 40](#_Toc57708835)

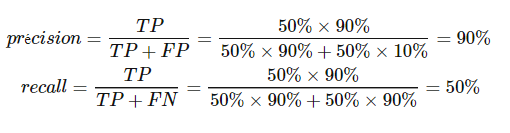
[Résumé 40](#_Toc57708836)

[Partie 2 41](#_Toc57708837)

[Compétences évaluées 41](#_Toc57708838)

[ Question 1 41](#_Toc57708839)

[ Question 2 41](#_Toc57708840)

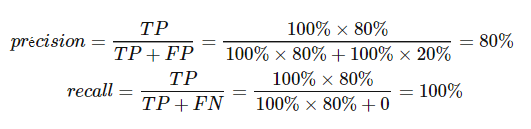
[**** 42](#_Toc57708841)

[ Question 3 42](#_Toc57708842)

[ Question 4 43](#_Toc57708843)

[ Question 5 44](#_Toc57708844)

[ Question 6 44](#_Toc57708845)

[**** 45](#_Toc57708846)

[ Question 7 45](#_Toc57708847)

[ Question 8 46](#_Toc57708848)

[Évaluez un algorithme de régression 47](#_Toc57708849)

[Mesures d'erreur 47](#_Toc57708850)

[Coefficient de détermination 48](#_Toc57708851)

[Résumé 50](#_Toc57708852)

[Comparez votre algorithme à des approches de régression naïves 51](#_Toc57708853)

[Approches naïves 51](#_Toc57708854)

[TP : Comparez un modèle kNN à des approches naïves 51](#_Toc57708855)

[Entraînez-vous : sélectionnez le nombre de voisins dans un kNN pour une régression 56](#_Toc57708856)

[À vous de jouer ! 56](#_Toc57708857)

[Vérifiez votre travail 56](#_Toc57708858)

‌Dans le cours [Initiez-vous au machine learning](https://openclassrooms.com/courses/initiez-vous-au-machine-learning), vous avez découvert les fondements de l'analyse de donnée automatisée. Dans ce deuxième cours, vous apprendrez à évaluer vos algorithmes pour les rendre plus performants.

**De nombreux choix d'algorithmes d'apprentissage** et de leurs hyperparamètres s'offrent aux Data Scientists. La nature du problème à résoudre permet en partie de guider ce choix. Par exemple, on n'appliquera pas un algorithme de classification à un problème de régression.

Néanmoins, il est nécessaire de **savoir évaluer** n'importe quel algorithme d'apprentissage sur son jeu de données, en évitant au mieux l**e biais de sur-apprentissage**. Une évaluation rigoureuse des performances d'un algorithme est une étape indispensable à son déploiement.

Suivez ce cours pour apprendre à **évaluer un modèle d'apprentissage supervisé** afin de choisir le **bon modèle**pour votre problème, en évitant de tomber dans un des **principaux pièges** qui guettent les Data Scientists.



Ce cours a été créé en partenariat avec CentraleSupélec

**Objectifs pédagogiques :**

À la fin de ce cours, vous saurez...

* Choisir une ou plusieurs **mesures de la performance d'un algorithme** d'apprentissage supervisé, adaptées à la question posée
* Mettre en place **une procédure de validation** qui réduise le risque de sur-apprentissage (séparation du jeu d'apprentissage et de validation, validation croisée)
* Mettre en place une procédure de **grille de recherche** (*grid search*) pour choisir les hyperparamètres d'un algorithme

**Prérequis :**

 Ce cours de Data Science se situe au croisement des mathématiques et de l'informatique. Pour en profiter pleinement, n'hésitez pas à vous rafraîchir la mémoire, avant ou pendant le cours, sur :

* **Python** pour le calcul numérique que nous utiliserons dans la partie TP du cours (librairie numpy et création de graphes avec pyplot)
* Quelques notions d'**algèbre linéaire**, telles que manipulation de vecteurs, multiplications de matrices, normes
* Quelques notions de **probabilités** et **statistiques**, telles que distribution de loi de probabilité et variance
* [**Le cours d'initiation au machine learning**](https://openclassrooms.com/courses/initiez-vous-au-machine-learning), afin d'avoir une meilleure idée du cycle global de travail d'un data scientist et comprendre où ce situe cette phase d'amélioration & mesure de performances

**Comprenez ce qui fait un bon modèle d’apprentissage**

Les algorithmes de machine learning sont nombreux. Vous avez peut-être déjà entendu parler de régression linéaire, de kNN (algorithme des k plus proches voisins), de réseaux de neurones ou de random forests (forêts aléatoires). La plupart de ces algorithmes ont des **hyperparamètres**, comme le nombre k de voisins dans le kNN, qu’il faut se fixer.

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

Naturellement, vous n’allez pas appliquer un algorithme de classification à un problème de régression. Vous n’allez pas non plus utiliser un algorithme qui s’applique à des données décrites par des nombres réels quand les vôtres sont décrites par des catégories (par exemple, une des variables que vous utilisez pour prédire quel film un client aimera est son genre, de fantastique à art et essai en passant par action). Mais cela ne va pas vous suffire pour choisir.

Ce que vous voulez vraiment, c’est construire un modèle qui vous donne **de bons résultats**. Mais qu’est-ce que ça veut dire, exactement ? C’est ce que nous allons voir.

**Généralisation**

Un **bon modèle de machine learning**, c’est un modèle qui généralise.

Qu’est-ce que c’est, déjà, la généralisation ?

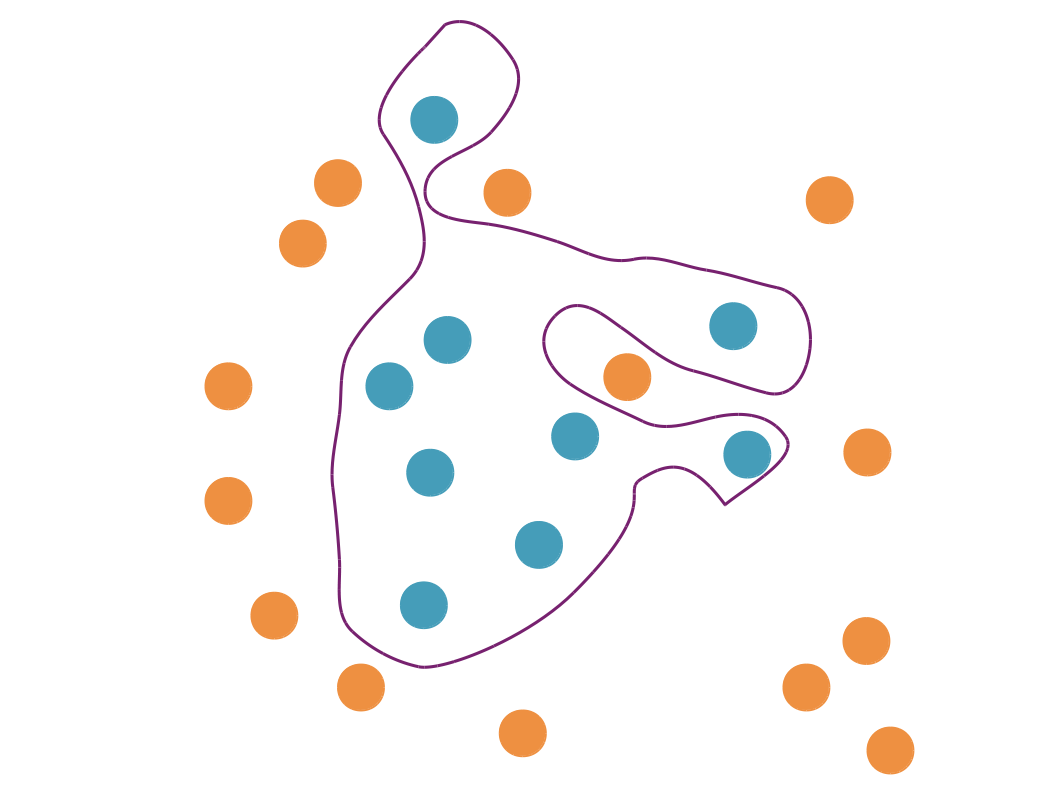
La **généralisation**, c’est la capacité d’un modèle à faire des prédictions non seulement sur les données que vous avez utilisées pour le construire, mais surtout sur **de nouvelles données** : c’est bien pour ça que l’on parle **d’apprentissage** !

Pour vous donner un exemple un peu extrême, imaginez que vous vouliez catégoriser des images : certaines représentent des chats, d’autres non. Pour construire votre modèle, vous disposez d’un jeu de données de 500 images de chats, et 500 images qui ne sont pas des chats. Vous pouvez construire un algorithme très simple : pour classer une image, regarder pixel par pixel si elle est exactement identique à une de celles dans vos données. Si c’est le cas, retournez l’étiquette (chat ou pas-chat) associée. Dans le cas contraire, répondez au hasard. Cet algorithme marche très bien sur le jeu de données initial, mais n’a rien appris du tout : il ne sait pas faire de prédictions !

Évaluer un modèle sur le jeu de données sur lequel on l’a construit ne nous permet donc pas du tout de savoir comment il se comportera sur de nouvelles données, celles sur lesquelles il est vraiment intéressant de faire de la prédiction.

**Sur-apprentissage et compromis biais-variance**

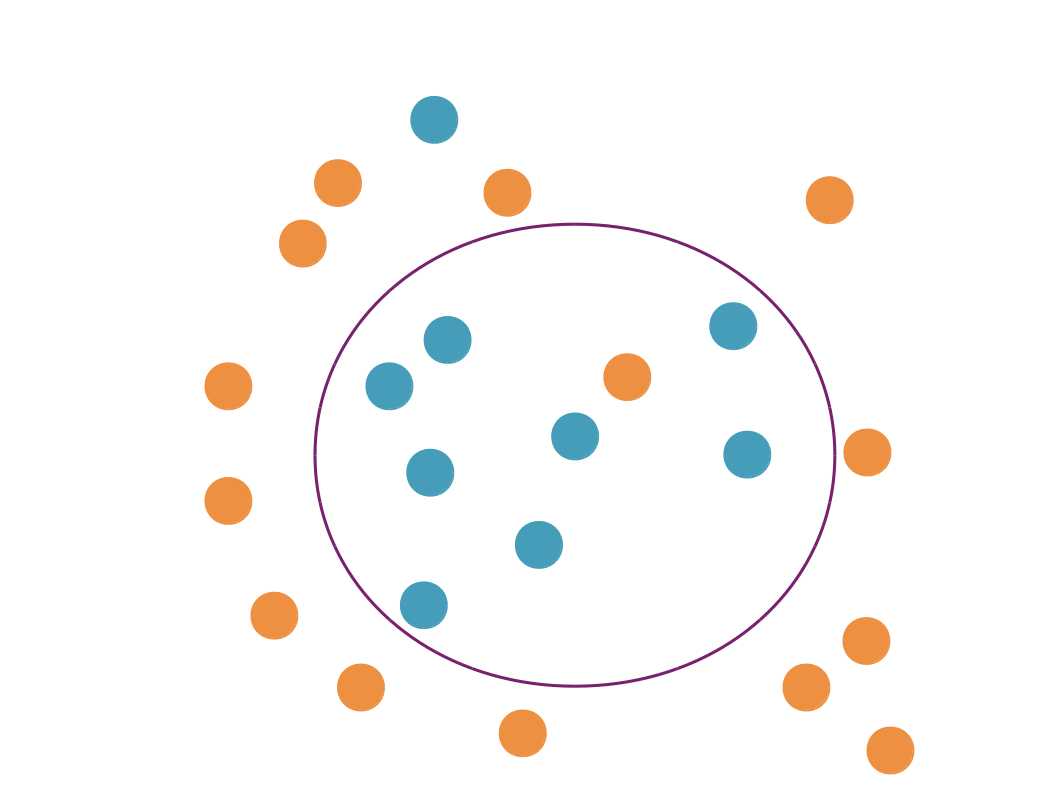
Sans tomber dans le cas extrême que j’ai décrit plus haut, on peut facilement se retrouver sans le vouloir avec un modèle qui colle trop aux données sur lesquelles on apprend (aussi appelées "jeu d’entraînement"), et qui soit trop sensible à leurs moindres variations pour bien les représenter. Un tel modèle aura de très bonnes performances sur le jeu d'entraînement mais sera mauvais sur de nouvelles données. On parle alors de **sur-apprentissage** (*overfitting* en anglais).



Sur cet exemple, le modèle (la ligne violette) qui sépare les points bleus des points oranges colle bien aux données, il ne fait aucune erreur. Mais est-ce qu’il modélise bien la réalité ?

Un modèle qui sur-apprend est un modèle qui est trop complexe par rapport à la réalité qu’il essaie de représenter. Nous avons tendance à préférer des modèles simples ; c’est le principe du **rasoir d’Ockham** (ou Occam), selon lequel les hypothèses suffisantes les plus simples sont les plus vraisemblables. Par ailleurs, coller de trop près aux données est une mauvaise idée car elles sont inévitablement bruitées :

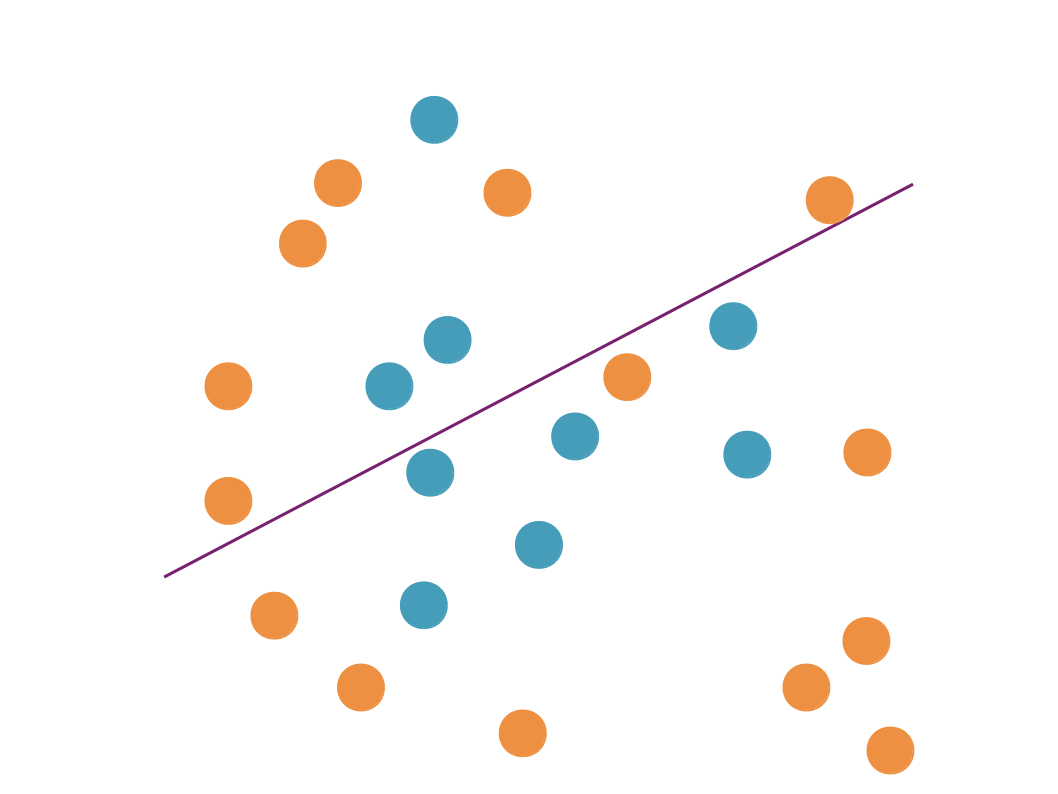
* Par des erreurs de mesure (les appareils que nous utilisons pour mesurer les variables qui représentent nos données peuvent faire des erreurs techniques) ;
* Par des erreurs d’étiquetage (l’erreur est humaine, et il se peut que certaines des étiquettes ne soient pas les bonnes) ;
* Parce que nous n’avons pas mesuré les variables les plus pertinentes, soit parce qu'on ne les connaît pas, soit parce qu'elles sont très compliquées à mesurer.



Ce modèle fait des erreurs sur le jeu d’apprentissage, mais il va probablement mieux généraliser

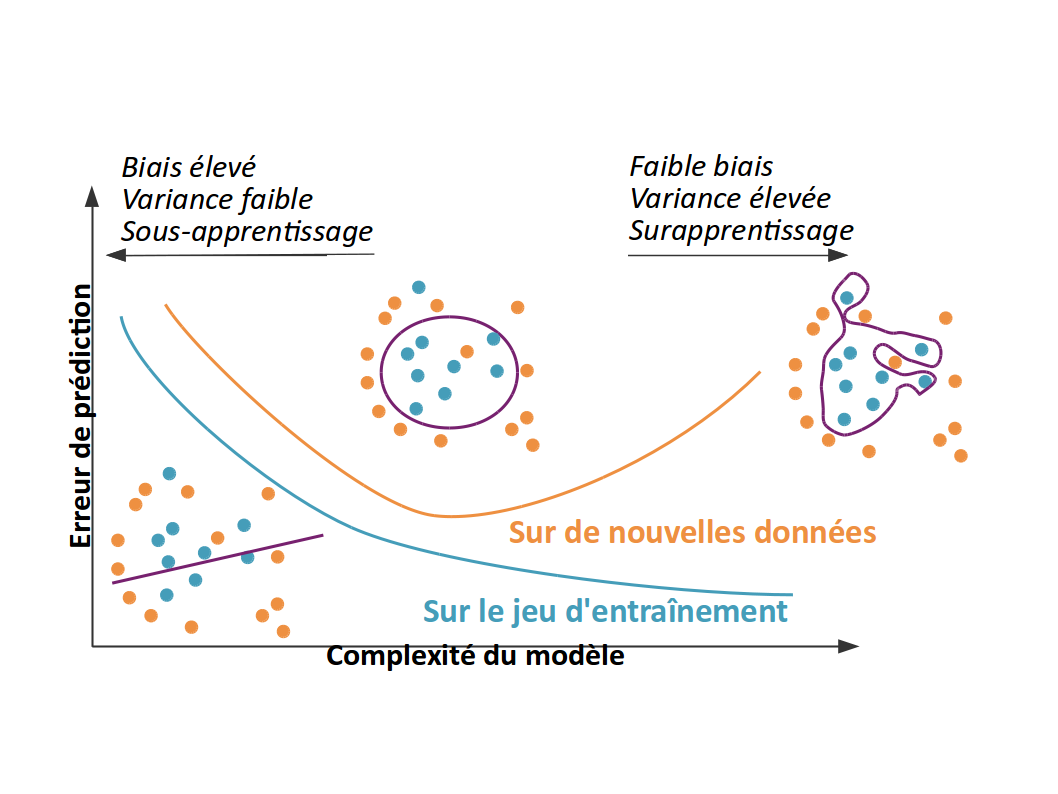
Retrouvez plus d’informations sur le rasoir d’Ockham sur [**Wikipedia**](https://fr.wikipedia.org/wiki/Rasoir_d%27Ockham?oldformat=true).

Il faut néanmoins aussi éviter les modèles trop simples, qui ne parviendront pas à bien représenter le phénomène qui nous intéresse, et qui ne feront pas de bonnes prédictions. On parle dans ce cas de "**sous-apprentissage"**.



Ce modèle, trop simple, représente trop mal les données pour prédire

Le concept de **compromis biais-variance** nous permet de bien résumer la situation :



**Compromis biais-variance :** Un modèle simple (variance faible) risque le sous-apprentissage (biais élevé y compris sur les données d’entraînement). Un modèle complexe (variance élevée) risque le sur-apprentissage (biais faible sur les données d’entraînement mais élevé sur de nouvelles données). On souhaite trouver un modèle intermédiaire, vers le creux de la courbe orange, là où le biais de prédiction est le plus faible et la généralisation la meilleure.

La complexité d’un modèle peut être mesurée de différentes façons. L’une d’entre elles est le nombre de paramètres utilisés par ce modèle : plus il y a de paramètres, plus le modèle est complexe.

**Les problèmes mal posés**

Pourquoi ne peut-on pas choisir a priori la complexité de notre modèle ?

En fait, toutes nos difficultés viennent du fait que les problèmes d'apprentissage sont des problèmes **mal posés** (*ill-posed* en anglais). Nous n'arrivons pas à les formuler de sorte à avoir une solution unique. Les données que nous observons ne sont pas suffisantes pour modéliser correctement le phénomène que nous cherchons à comprendre pour répondre à notre problématique, et nous devons rajouter des hypothèses (par exemple, que la frontière séparant les points bleus des points orange est un cercle) pour finalement arriver à un bon modèle. Ces hypothèses forment ce que l’on appelle le **biais d’induction** (i*nductive bias* en anglais).

Allez plus loin sur la question des problèmes mal posés avec cet article [**Wikipedia**](https://www.wikiwand.com/en/Well-posed_problem).

**Ressources informatiques**

Attention ! Vous pouvez être amenés à faire des compromis sur la complexité d’un modèle pour respecter des contraintes sur le temps de calcul (que ce soit pour l’entraînement ou pour la prédiction) ou les ressources en mémoire. Votre application nécessite-t-elle d’utiliser un modèle que vous devrez faire tourner pendant une heure pour produire une réponse, si un autre modèle peut vous donner une réponse de qualité légèrement inférieure en quelques secondes ?

**En résumé**

* En apprentissage supervisé, le but est de produire des modèles qui **généralisent**, c’est-à-dire qui sont capables de faire de bonnes prédictions sur de nouvelles données
* De bonnes performances sur le jeu d’entraînement **ne garantissent pas** que le modèle sera capable de généraliser !
* On cherche à développer un modèle qui soit suffisamment complexe pour bien capturer la nature des données (et éviter ainsi le sous-apprentissage), mais suffisamment simple pour éviter le **sur-apprentissage**.
* Attention aux **contraintes de temps de calcul** et aux **ressources en mémoire** !

**Mettez en place un cadre de validation croisée**

Comme nous l’avons vu dans le chapitre précédent, nous voulons nous assurer que notre modèle ne souffre pas de sur-apprentissage, et qu’il saura faire des prédictions sur de nouvelles données

Comment peut-on mesurer la performance d’un modèle sur des données inconnues ?

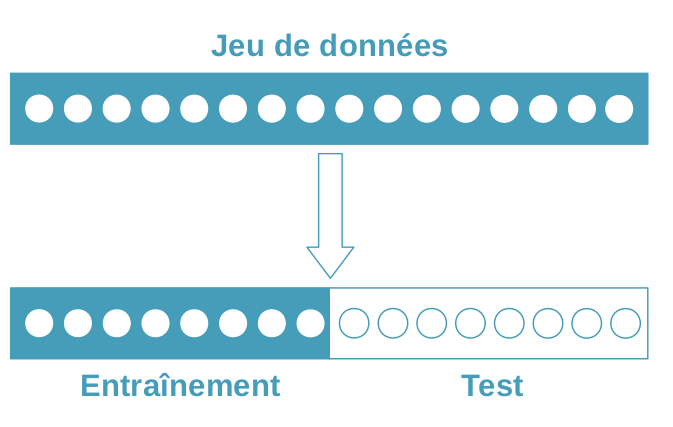
Attends, comment fait-on déjà pour mesurer la performance d’un modèle sur des données connues ? Il s’agirait déjà de savoir mesurer la performance d’un modèle sur des données dont nous connaissons les étiquettes !

C’est quelque chose que nous verrons en détail dans les deux prochaines parties de ce cours, mais pour l’instant, disons que l’on mesure [**le risque empirique du modèle**](https://openclassrooms.com/courses/initiez-vous-au-machine-learning/comment-se-passe-l-apprentissage-d-un-modele#/id/r-4120988) :

* Pour un problème de classification, la proportion de points que le modèle étiquette mal ;
* Pour un problème de régression, la moyenne des erreurs quadratiques.

**Jeux d’entraînement et de test**

La première idée, c’est de **couper notre jeu de données en deux parties** : un jeu d'entraînement, et un jeu de test. On n’utilise ensuite pas du tout le jeu de test quand on choisit et qu’on entraîne notre modèle. Comme ça, on peut calculer la performance sur le jeu de test, et le résultat est une bonne approximation de la performance sur des données inconnues.



On sépare le jeu de données en un jeu d’entraînement et un jeu de test. Le jeu de test n’est pas utilisé pour entraîner le modèle, mais uniquement pour l’évaluer. Ici les deux jeux font la même taille mais ce n’est pas obligatoire !

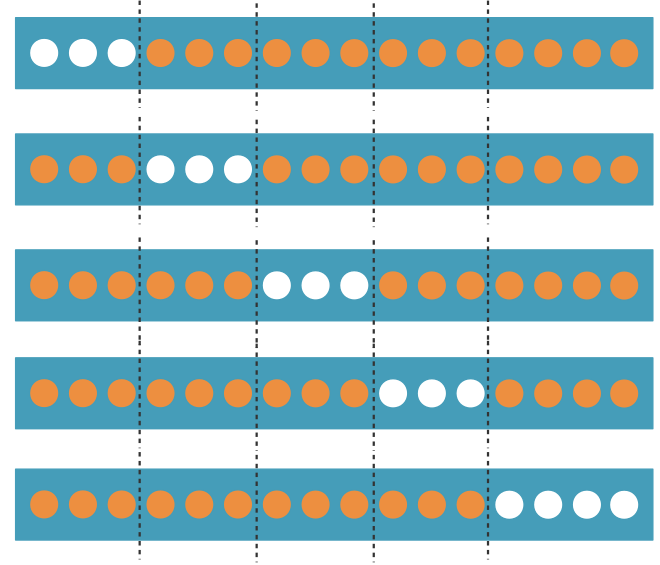
**Validation croisée**

C’est dommage, nous n’avons utilisé qu'une partie du jeu de données pour entraîner, et qu'une partie pour tester ! Et si nous avions par hasard créé un jeu de test vraiment difficile — ou vraiment facile — à prédire ? L’estimation de la performance serait biaisée !

Par ailleurs, moins on a de données, moins bien on apprend. Ne sommes-nous donc pas en train de créer des modèles moins bons, juste pour pouvoir les valider ?

**La validation croisée** va nous permettre d’utiliser l'intégralité de notre jeu de données pour l’entraînement et pour la validation ! Voilà comment ça marche :

On découpe le jeu de données en k parties (***folds*** en anglais) à peu près égales. Tour à tour, chacune des k parties est utilisée comme jeu de test. Le reste (autrement dit, l’union des k-1 autres parties) est utilisé pour l'entraînement.



Une cross-validation à 5 folds : Chaque point appartient à 1 des 5 jeux de test (en blanc) et aux 4 autres jeux d’entraînements (en orange)

À la fin, chaque point (ou observation) a servi 1 fois dans un jeu de test, (k-1) fois dans un jeu d'entraînement. J'ai donc 1 prédiction par point de mon jeu initial, et aucune de ces prédictions n'a été faite avec un jeu d'entraînement qui contienne ce point. Je n'ai pas violé le principe de ne pas valider sur le jeu d'entraînement !

Je peux finalement rapporter la performance de mon modèle :

* soit en évaluant les prédictions faites sur l’ensemble des données (puisque j’ai fait une prédiction par point du jeu de données complet) ;
* soit en moyennant les performances obtenues sur les k folds, auquel cas je peux aussi rapporter l’erreur type, pour quantifier la variation de ces performances sur les k folds.

Entrée : données X (dimension nxp), étiquettes y (dimension n), nombre de folds k

Couper [0, 1, ..., n-1] en k parties de taille (n/k). (La dernière partie sera un peu plus petite si n n'est pas un multiple de k)

for i=0 to (k-1):

Former le jeu de test (X\_test, y\_test) en restreignant X et y aux indices contenus dans la i-ième partie.

Former le jeu d'entraînement (X\_train, y\_train) en restreignant X et y aux autres indices.

Entraîner l'algorithme sur le jeu d'entraînement

Utiliser le modèle ainsi obtenu pour prédire sur le jeu de test

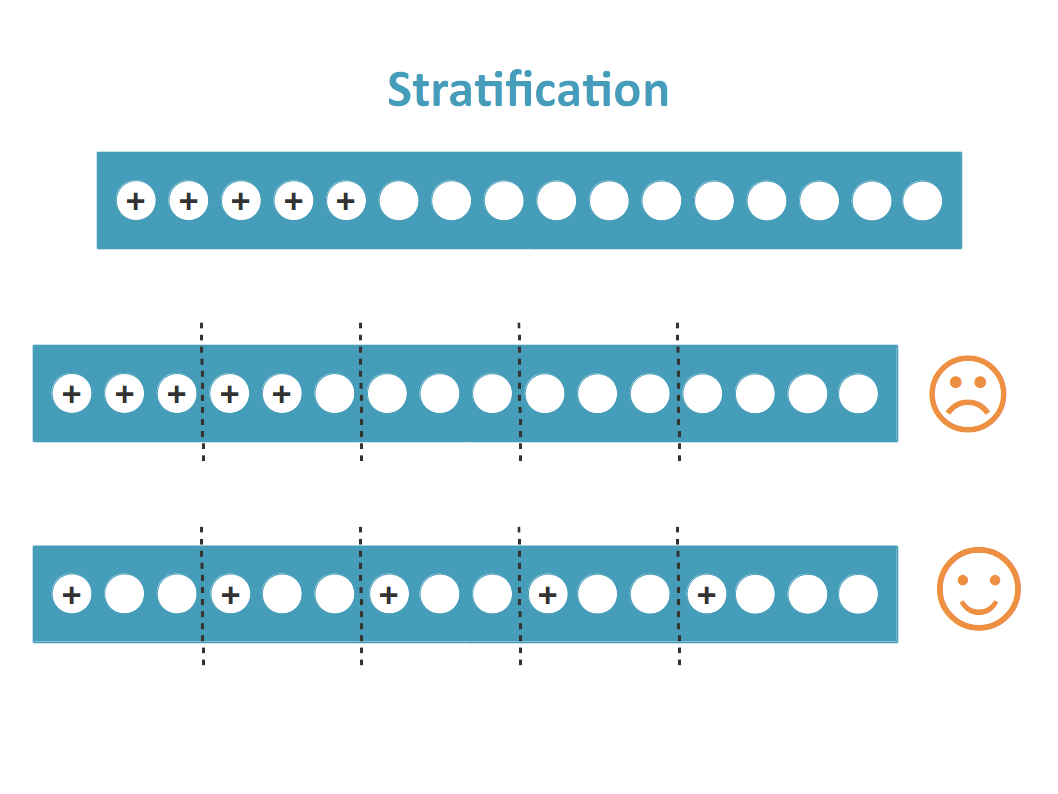
Calculer l'erreur du modèle en comparant les étiquettes prédites aux vraies étiquettes contenues dans y\_test

Sortie : la valeur moyenne des erreurs calculées sur les k folds.

Dans scikit-learn, la méthode [**model\_selection.KFold**](http://scikit-learn.org/stable/modules/cross_validation.html#k-fold) permet de créer les folds d’une validation croisée.

**Stratification**

Dans le cas d’un problème de classification, on s’efforce généralement de créer les k folds de sorte à ce qu’elles contiennent à peu près les mêmes proportions d’exemples de chaque classe que le jeu de données complet. On cherche à éviter qu’un jeu d’entraînement ne contienne que des exemples positifs et que le jeu de test correspondant ne contienne que des exemples négatifs, ce qui va affecter négativement la performance du modèle !



Si l’on ne répartit pas les points positifs de manière équilibrée entre les différents folds, les jeux d’entraînement et de test auront des proportions différentes de positifs et négatifs, ce qui peut biaiser les résultats

Dans scikit-learn, la méthode [**model\_selection.StratifiedKFold**](http://scikit-learn.org/stable/modules/cross_validation.html#stratified-k-fold) permet de créer les folds d’une validation croisée stratifiée.

Au moment de l'*apprentissage* (et non pas de l'évaluation), on peut compenser le déséquilibre entre les classes dans le jeu d'entraînement en utilisant une méthode de ré-échantillonnage : on tire aléatoirement parmi la classe majoritaire autant d'observations que dans la classe minoritaire, ce qui crée un jeu équilibré, opération que l'on répète de nombreuses fois. On crée ainsi plusieurs modèles, que l'on peut ensuite combiner en moyennant leurs scores ou en choisissant l'étiquette la plus fréquemment prédite.

**Leave-one-out**

Comment choisir le nombre de folds, k ?

Rappelez-vous, moins il y a de données disponibles pour l’entraînement, moins on est capable de bien apprendre. Or chaque fold contient points (si **n** est la taille du jeu complet). Si on pousse ce raisonnement, on fait autant de folds que de points dans le jeu complet (c’est à dire n) , et les jeux d’entraînement font quasiment la même taille que le jeu complet ! C’est ce qu’on appelle le ***leave-one-out*** (on ne laisse de côté qu’un seul exemple pour chaque jeu d’entraînement).

Ça a l’air bien ! Sauf que…

* on augmente fortement le temps de calcul. Imaginez ça sur une base de données de 100 000 images ! Il faudrait entraîner 100 000 modèles sur 99 999 images chacun.
* on forme ainsi des jeux d’entraînements très similaires entre eux, et des jeux de test très différents les uns des autres. On va avoir quasiment le même modèle sur chaque fold, et la qualité de ses prédictions risque de beaucoup varier. Il sera difficile de tirer des conclusions.

En pratique, on choisit le plus souvent k=5 ou k=10.

**En résumé**

* Il ne faut **jamais** évaluer un modèle sur des points qui ont été utilisés pour l’entraîner.
* On sépare donc les données entre **un jeu d’entraînement**, sur lequel on apprend le modèle, et **un jeu de test**, sur lequel on l’évalue.
* Pour utiliser l’intégralité de nos données pour entraîner et pour tester, et pour éviter un biais potentiel lié au fait de faire une évaluation unique, **on préfère faire une validation croisée**.
* Dans le cas d’un problème de classification, on fait attention à **stratifier la validation croisée** pour éviter d’introduire des biais.

**TP – Sélectionnez le nombre de voisins dans un kNN**

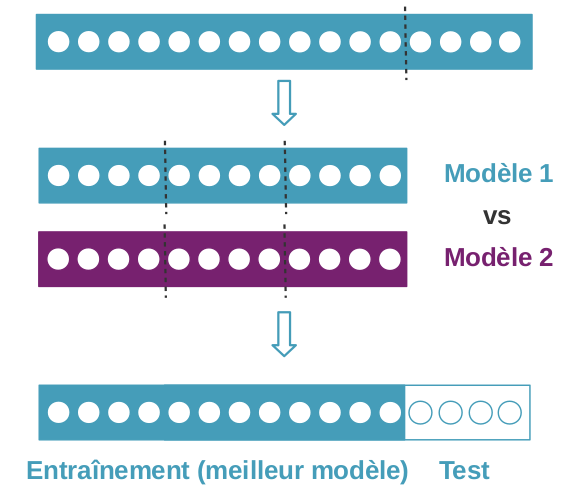
Jusqu’à présent, nous avons parlé de l’évaluation de la performance d’un seul modèle. Mais, généralement, nous voulons essayer plusieurs modèles pour choisir le plus performant, et ensuite donner sa performance.

**Sélection de modèle**

Attention ! Il ne suffit pas de faire une validation croisée sur l’ensemble des données, pour chaque modèle, puis de donner la meilleure performance obtenue : ce n’est pas une bonne estimation de l’erreur en généralisation. En effet, en faisant ça, nous utilisons les données de test pour choisir le modèle… Il y a un risque de sur-apprentissage.

Pour faire ça correctement, il faut séparer les données en trois parties : un **jeu d’entraînement**, un **jeu de validation** et un **jeu de test**. Le jeu d’entraînement sert à entraîner divers modèles. Le jeu de validation sert à *sélectionner* un modèle : on choisit celui qui a la meilleure performance sur ce jeu. Enfin, le jeu de test sert à *estimer la performance* en généralisation du modèle.

Alternativement, au lieu de créer un jeu d’entraînement et un jeu de validation, on peut séparer les données uniquement en deux parties : un jeu d’entraînement et un jeu de test. On fera ensuite une *validation croisée* sur le jeu d’entraînement. Cela nous permet de choisir un modèle (celui qui a la meilleure performance), que l’on va ensuite entraîner sur la totalité du jeu d’entraînement, puis tester sur le jeu de test. C’est cette performance finale qui est la meilleure approximation de la performance que le modèle pourra atteindre sur de nouvelles données.



On sépare le jeu de données en un jeu d’entraînement et un jeu de test. On évalue chaque modèle en validation croisée sur le jeu d’entraînement pour choisir le meilleur, que l’on applique ensuite au jeu de test.

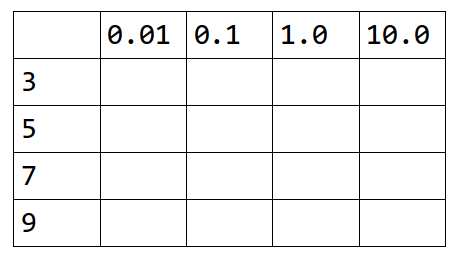
On peut appliquer le même raisonnement qui nous a conduit à faire une validation croisée plutôt qu’une simple séparation entraînement / test et répéter cette procédure au sein d’une autre validation croisée. On parle alors de **validations croisées imbriquées** (« *nested cross-validation* » en anglais). Le défaut de ce type d’approche, en pratique, et qu’elle peut sélectionner des modèles différents sur chaque fold…

**Recherche sur grille (*grid search*)**

Un cas particulier de la sélection de modèle est celui de la sélection du ou des hyperparamètres d’un même algorithme : par exemple, le nombre k de voisins dans un kNN. Dans ce cas, on crée **une grille d’hyperparamètres**, contenant plusieurs valeurs possibles pour chacun d’entre eux, que l’on explore pour tester toutes les combinaisons possibles. C’est ce que l’on appelle une « ***grid search*** » en anglais.

Attention à ne pas confondre paramètres et hyperparamètres.

Les **paramètres** interviennent dans les modèles dits paramétriques. Ils sont généralement notés **θ** et déterminent le modèle ; par exemple une droite peut être représentée par l’équation . Ce sont eux que l’on cherche à apprendre. Les **hyperparamètres** sont des paramètres de l’algorithme d’apprentissage qui va nous permettre de déterminer le modèle (autrement dit, dans le cas d’un modèle paramétrique, les paramètres du modèles). Par exemple, le nombre k de voisin dans un kNN.



Une grille de recherche d’hyperparamètres pour deux paramètres, chacun prenant 4 valeurs

 Dans le cas où on évalue un seul hyperparamètre, on parle parfois de ligne de recherche (« *line search* » en anglais).

### Sélectionnez le nombre de voisins dans un kNN sur un jeu de données

Mettons donc tout cela en pratique !

Dans ce TP j’utilise la version 3.7 de Python, la version 0.21 de [**scikit-learn**](http://scikit-learn.org/), la version 0.25 de [**pandas**](https://pandas.pydata.org/), la version 3.1 de [**matplotlib**](https://matplotlib.org/) et la version 1.17 de [**numpy**](https://numpy.org/).

**Les données**

Nous allons travailler avec un jeu de données qui contient des informations physico-chimiques de vins portugais (*vinho verde*), ainsi que leur qualité telle que notée par des humains.

Le problème, sur ces données, est de prédire automatiquement la qualité sur la base de ces informations, afin d’assister le travail d’évaluation des œnologues, d’améliorer la production de vin, et de cibler le goût des consommateurs sur des marchés de niche.

Ce jeu de données est disponible dans les archives UCI (un des répertoires les plus connus de problèmes de machine learning).

Vous pouvez le télécharger [ici](https://s3-eu-west-1.amazonaws.com/course.oc-static.com/courses/4297211/winequality-red+(7).csv).

Commençons par regarder les données :

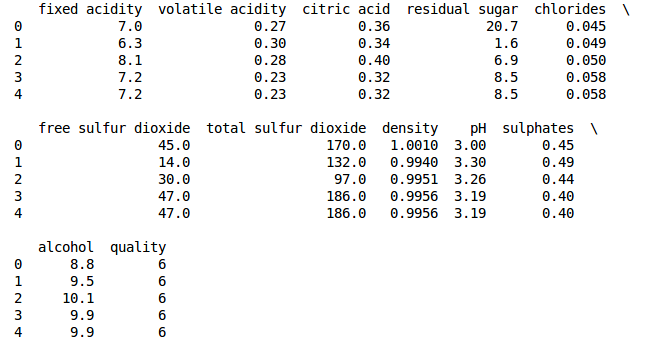
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data = pd.read\_csv('winequality-white.csv', sep=";")

print(data.head())



Nos données contiennent 11 colonnes, 10 qui correspondent à divers indicateurs physico-chimiques et 1 qui est la qualité du vin.

Nous allons extraire deux *arrays* numpy de ces données, un qui contient les points et l’autre qui contient les étiquettes

X = data[data.columns[:-1]].values

y = data['quality'].values

On peut maintenant afficher un histogramme pour chacune de nos variables :

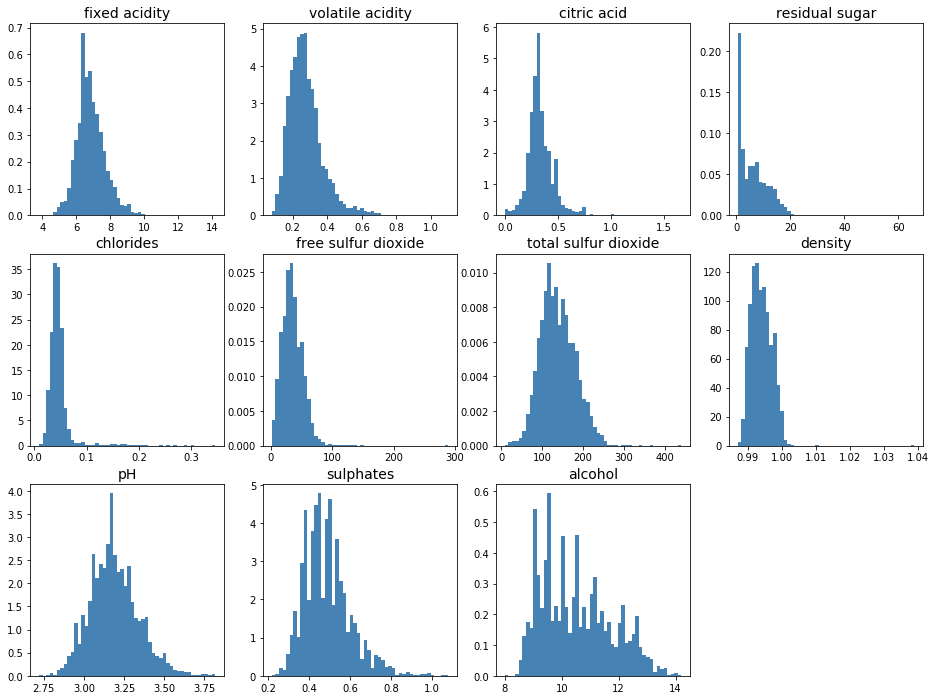
fig = plt.figure(figsize=(16, 12))

for feat\_idx in range(X.shape[1]):

ax = fig.add\_subplot(3,4, (feat\_idx+1))

h = ax.hist(X[:, feat\_idx], bins=50, color='steelblue', density=True, edgecolor='none')

ax.set\_title(data.columns[feat\_idx], fontsize=14)



On remarque en particulier que ces variables prennent des valeurs dans des ensembles différents. Par exemple, “sulphates” varie de 0 à 1 tandis que “total sulfur dioxide” varie de 0 à 440. Il va donc nous falloir standardiser les données pour que la deuxième ne domine pas complètement la première.

**Sélection de modèle**

Nous allons commencer par transformer ce problème en un problème de classification : il s’agira de séparer les bons vins des vins médiocres :

y\_class = np.where(y<6, 0, 1)

Séparons nos données en un jeu d’entraînement et un jeu de test. Le jeu de test contiendra 30% des données.

from sklearn import model\_selection

X\_train, X\_test, y\_train, y\_test = \

model\_selection.train\_test\_split(X, y\_class,

test\_size=0.3 # 30% des données dans le jeu de test

)

Nous pouvons maintenant standardiser les données d’entraînement et appliquer la même transformation aux données de test :

from sklearn import preprocessing

std\_scale = preprocessing.StandardScaler().fit(X\_train)

X\_train\_std = std\_scale.transform(X\_train)

X\_test\_std = std\_scale.transform(X\_test)

On peut visualiser de nouveau les données pour vérifier que les différentes variables prennent des valeurs qui ont maintenant des ordres de grandewur similaires.

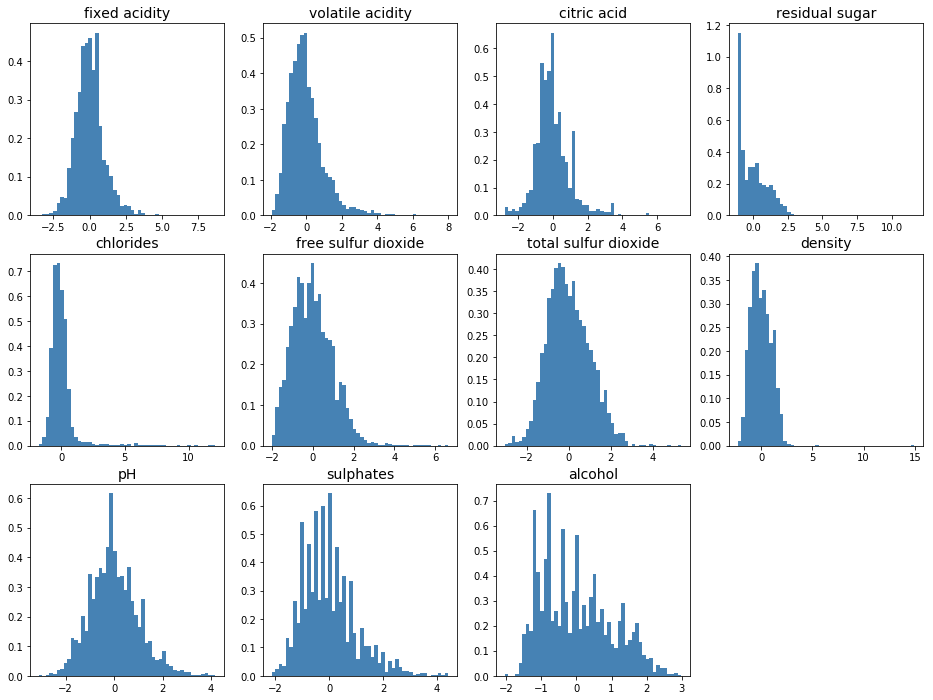
fig = plt.figure(figsize=(16, 12))

for feat\_idx in range(X\_train\_std.shape[1]):

ax = fig.add\_subplot(3,4, (feat\_idx+1))

h = ax.hist(X\_train\_std[:, feat\_idx], bins=50, color = 'steelblue', density=True, edgecolor='none')

ax.set\_title(data.columns[feat\_idx], fontsize=14)



Nous allons maintenant utiliser la méthode "GridSearchCV" pour faire une validation croisée du paramètre k d’un kNN (le nombre de plus proches voisins) sur le jeu d’entraînement :

from sklearn import neighbors, metrics

# Fixer les valeurs des hyperparamètres à tester

param\_grid = {'n\_neighbors':[3, 5, 7, 9, 11, 13, 15]}

# Choisir un score à optimiser, ici l'accuracy (proportion de prédictions correctes)

score = 'accuracy'

# Créer un classifieur kNN avec recherche d'hyperparamètre par validation croisée

clf = model\_selection.GridSearchCV(

neighbors.KNeighborsClassifier(), # un classifieur kNN

param\_grid, # hyperparamètres à tester

cv=5, # nombre de folds de validation croisée

scoring=score # score à optimiser

)

# Optimiser ce classifieur sur le jeu d'entraînement

clf.fit(X\_train\_std, y\_train)

# Afficher le(s) hyperparamètre(s) optimaux

print("Meilleur(s) hyperparamètre(s) sur le jeu d'entraînement:")

print(clf.best\_params\_)

# Afficher les performances correspondantes

print("Résultats de la validation croisée :")

for mean, std, params in zip(

clf.cv\_results\_['mean\_test\_score'], # score moyen

clf.cv\_results\_['std\_test\_score'], # écart-type du score

clf.cv\_results\_['params'] # valeur de l'hyperparamètre

):

print("{} = {:.3f} (+/-{:.03f}) for {}".format(

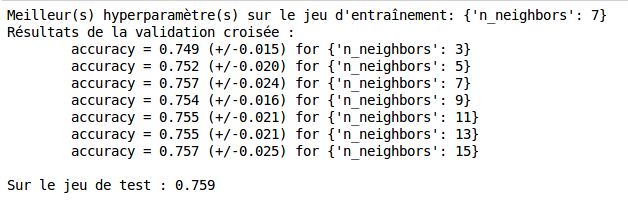
score,

mean,

std\*2,

params

) )



Les performances obtenues dépendent des jeux d’entraînement et de validation, vous pouvez donc obtenir d’autres résultats que moi.

La meilleure performance (~0.757) est ici atteinte avec 7 voisins.

Nous pouvons maintenant regarder la performance sur le jeu de test. GridSearchCV a automatiquement ré-entraîné le meilleur modèle sur l’intégralité du jeu d’entraînement,

y\_pred = clf.predict(X\_test\_std)

print("\nSur le jeu de test : {:.3f}".format(metrics.accuracy\_score(y\_test, y\_pred)))

J’obtiens ici une performance de 0.759.

**Alternatives à la recherche exhaustive**

La méthode de validation croisée que nous venons d’étudier est une méthode de recherche exhaustive (ou encore de recherche par force brute, ou “*brute force*” en anglais) : on définit des valeurs que peuvent prendre les hyperparamètres, puis on teste toutes ces valeurs pour choisir celles qui nous conviennent le mieux.

Il existe pour certains algorithmes d’apprentissage statistique des méthodes alternatives.

Pour certains algorithmes d’apprentissage, il est possible de déterminer efficacement les modèles obtenus pour *toutes* les valeurs d’un certain hyperparamètre comprises dans une fourchette donnée, sans recalculer le modèle à chaque fois ; on pourra donc faire une recherche certes toujours exhaustive, mais plus efficace. C’est le cas des approches de régression linéaire régularisée.

Dans certains cas, la théorie de l’information nous permet de **calculer la valeur optimale d’un hyperparamètre** grâce à une formule fermée, c’est-à-dire explicitement. L’idée générale de ces approches est d’estimer la perte d'information subie lorsque l’on utilise un certain modèle pour représenter le processus qui a généré les données ; leur but est de choisir le modèle qui donnera lieu à la plus petite perte d’information possible.

Les deux approches les plus fréquemment mentionnées dans ce contexte sont le [**critère d’information de Akaike (ou *Akaike Information Criterion*, AIC)**](https://www.wikiwand.com/fr/Crit%C3%A8re_d'information_d'Akaike) et le [**critère d’information bayésien (ou *Bayesian Information Criterion*, BIC)**](https://www.wikiwand.com/fr/Crit%C3%A8re_d%27information_bay%C3%A9sien). En pratique, on s’en sert principalement pour certaines approches de régression linéaire régularisée.

**Résumé**

* Pour sélectionner un modèle, on **compare les performances en validation croisée***sur un jeu d’entraînement*.
* Pour sélectionner les valeurs des hyperparamètres d’un algorithme donné, on fait une ***grid search***, dans laquelle on essaie de couvrir l’espace des valeurs pertinentes de ces hyperparamètres.
* On peut implémenter cela très simplement en Python avec **sklearn.model\_selection.GridSearchCV**.

## Entraînez-vous : implémentez une validation croisée

### 

### À vous de jouer !

Pour vous entraîner, réalisez cet exercice étape par étape. Une fois terminé, vous pouvez comparer votre travail avec les pistes que je vous propose.

Dans cette activité, vous devez **ré-implémenter la fonction de validation croisée** de la libraire scikit-learn (la fonction GridSearchCV), dans l’objectif d’effectuer la classification du dataset sur la qualité du vin. Pour cela, vous pourrez réutiliser le code du TP du chapitre précédent comme base de travail.

L’algorithme devra permettre d’optimiser l’accuracy du modèle. La fonction prendra en entrée le tableau des hyperparamètres à tester ainsi que le nombre de folds. On utilisera des folds exacts (non randomisé) afin de pouvoir comparer les résultats.

Vous pourrez ensuite comparer les performances de votre implémentation par rapport à l’implémentation scikit-learn effectuée lors du TP. Pour cela, je vous conseille dans un premier temps de ne pas [randomiser la sélection des sets](http://scikit-learn.org/stable/modules/cross_validation.html), mais de faire une sélection exacte afin de pouvoir comparer des résultats qui doivent être identiques entre votre implémentation et celle de scikit.

À l'issue de votre mission, vous devez fournir :

* le classeur iPython avec l’implémentation de la fonction.
* le code modifié du TP permettant l’application et l’évaluation du K-nn avec votre fonction de validation croisée.

À vous de jouer !

### Vérifiez votre travail

Vous avez rempli votre mission si :

* Votre fonction de validation croisée fournit le même résultat que la fonction scikit originale.
* L'algorithme fonctionne.
* Vous avez correctement effectué l'évaluation.

## 

## Évaluez un algorithme de classification qui retourne des valeurs binaires

Nous allons maintenant nous concentrer sur les modèles de **classification** : on utilise des données étiquetées pour prédire à quelle classe un objet appartient. Nous allons surtout parler de **classification binaire**, où il s'agit de distinguer si un objet appartient ou non à une classe. Par exemple, dire si une image représente une girafe ou non. Si oui, on dit que cette image est positive ; sinon, qu'elle est négative.

Jusqu'à présent, en évaluant des modèles de classification, nous avons utilisé **le nombre d’erreurs** comme mesure de performance. Mais ce n'est pas le seul critère ! En effet, **toutes les erreurs ne se valent pas**.

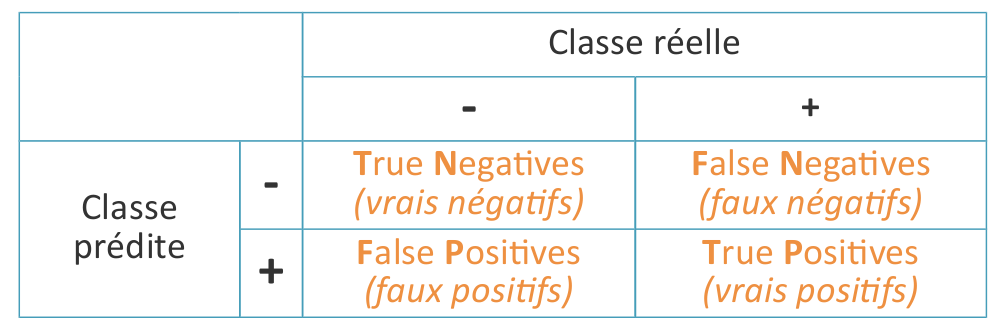
Prenons un algorithme qui prédit s’il y a un incendie à un endroit donné. Déclencher une alerte incendie quand il n'y a pas le feu est moins grave que de ne pas déclencher d'alerte quand l’appartement est en flamme.

Dans ce chapitre, je vais vous montrer différentes manières d'évaluer un modèle de classification.

### 

### Matrice de confusion

Prédire un incendie quand il n’y en a pas, ne pas prédire un incendie quand il y en a un, ne pas en prédire quand il n’y en a pas… On s’y perd vite ! Pour y voir plus clair, on utilise une **matrice de confusion** (normalement le terme fait allusion à la confusion du modèle, pas à celle du Data Scientist ;) ).



Matrice de confusion

Appelons "**positive"** la classe correspondant à un incendie et "**négative**" l’autre. Si on prédit un incendie quand il y en a bien un, on fait une prédiction "positif" qui est correcte, c’est un **vrai positif.** Si par contre cette prédiction est incorrecte, il s’agit d’un **faux positif.** Et ainsi de suite. On appelle aussi parfois "**erreur de type I"** les faux positifs, et "**erreur de type II"** les faux négatifs.

Pour obtenir une matrice de confusion avec scikit-learn, il suffit d’utiliser la fonction [confusion\_matrix](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix)

metrics.confusion\_matrix(y\_true, y\_pred)

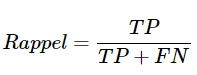
### 

### Critères à optimiser et valeurs dérivées

À partir de la matrice de confusion on peut dériver tout un tas de critères de performance. De manière générale, on préfère donner une fraction d'erreurs à un nombre total d'erreurs (5% d'erreurs, ça parle plus que 10 erreurs, quand on ne connaît pas le nombre de points dans le jeu de test).

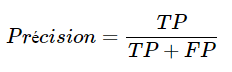
Voici quelques exemples de mesures de performance souvent utilisées :

Le **rappel** ("recall"  en anglais), ou **sensibilité** ("sensitivity" en anglais), est le **taux de vrais positifs**, c’est à dire la proportion de positifs que l’on a correctement identifiés. C’est la capacité de notre modèle à détecter tous les incendies.



On peut facilement avoir un très bon rappel… En prédisant systématiquement "positif". On ne ratera aucun incendie, mais notre modèle ne sert pas à grand-chose.

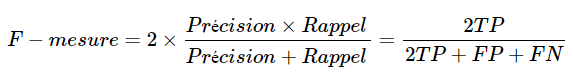
On s’intéressera donc aussi à la **précision**, c’est-à-dire la proportion de prédictions correctes parmi les points que l’on a prédits positifs. C’est la capacité de notre modèle à ne déclencher d’alarme que pour un vrai incendie.



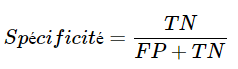
En anglais on distingue "precision" (la précision dont je viens de parler) et "accuracy" (la proportion de points correctement prédits) ; en français il n’y a pas de bonne traduction qui différencie les deux…

Mais on peut relativement facilement avoir une très bonne précision… en prédisant très peu de positifs (on risque moins de se tromper).

Pour évaluer un compromis entre rappel et précision, on peut calculer la "F-mesure", qui est leur moyenne harmonique.



Pour finir cette longue liste, on s’intéresse aussi souvent à la **spécificité** ("specificity" en anglais), qui est le **taux de vrais négatifs**, autrement dit la capacité à détecter toutes les  situations où il n’y a pas d’incendie. C’est une mesure complémentaire de la sensibilité.



Toutes ces mesures de performance sont disponibles dans le module metrics de scikit-learn.

### 

### Exemple : test clinique

Pour comprendre un peu mieux à quoi toutes ces mesures correspondent, prenons l’exemple d’un test clinique. Il s’agit ici d’une étude conduite sur 4000 femmes, âgées de 40 ans et plus, apparemment en bonne santé. On leur a fait passer deux tests de dépistage du col de l'utérus :

* Un examen histologique, plutôt lourd, qui requiert d’être interprété par un expert, et qui servira de vérité terrain.
* Un frottis de dépistage, qui est un examen beaucoup plus simple et moins invasif, qui sera ici l’analyse de notre algorithme d’apprentissage.

Voici les résultats de l'expérience :

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Cancer** | **Pas de cancer** | **TOTAL** |
| **Frottis +** | 190 | 210 | 400 |
| **Frottis -** | 10 | 3590 | 3600 |
| **TOTAL** | 200 | 3800 | 4000 |

La probabilité de ne pas avoir de cancer quand le frottis est négatif est de 3590/3600 soit 99.7%, ce qui fait de ce test un bon outil de **dépistage**. À l’inverse, la probabilité d'avoir un cancer quand le frottis est positif est de 190/400 = 47.5%, ce qui en fait un très mauvais outil **diagnostique**.

Il ne tient ainsi qu'à vous de sélectionner la mesure la plus pertinente au vu de la problématique à traiter.

Cette situation est reflétée par les valeurs suivantes :

* rappel = 95%
* spécificité = 94.5 %
* précision = 47.5%

### 

### Résumé

* **Toutes les erreurs ne se valent pas**.
* On peut représenter les performances d’un modèle de classification dans une**matrice de confusion**.
* On peut extraire de cette matrice différentes mesures de performance, comme le **rappel**, la **précision**, la **spécificité** et le **F-score**, qui reflètent différents aspects du modèle.

## Évaluez un algorithme de classification qui retourne des scores

### 

### Certains modèles de classification retournent des valeurs réelles

Bien que le but d'un algorithme de classification soit de faire des prédictions binaires, un grand nombre d'algorithmes retournent **un nombre réel**. Plus cette valeur est élevée, plus le point est susceptible d'être positif. Dans de nombreux cas (régression logistique, méthodes bayésiennes, réseaux de neurones…), il s’agit d’ailleurs d’une estimation de la probabilité que ce point soit positif.

Par exemple, l’algorithme des k plus proches voisins (kNN) pourrait retourner non pas la classe majoritaire parmi celles de k plus proches voisins du point, mais la proportion d’exemples positifs parmi ces voisins.

Dans ce cas, pour retourner une prédiction binaire, il faut seuiller : si le score retourné est supérieur au seuil, alors on prédit positif ; s’il est inférieur, on prédit négatif.

Mais quel seuil choisir ?

### 

### La courbe ROC

Dans le chapitre précédent, nous avons évoqué l’évaluation du rappel (ou sensibilité) et la spécificité d'un modèle de prédiction binaire. C’est ce que nous allons faire ici, pour chaque seuil de décision possible. Nous allons ensuite construire une courbe pour montrer comment la sensibilité évolue en fonction de la spécificité (en fait, de 1 moins la spécificité, que l'on appelle parfois antispécificité).

Cette courbe s’appelle la courbe ROC, pour « Receiver-Operator Characteristic ». Ce terme vient des télécommunications, où ces courbes servent à étudier si un système arrive à séparer le signal du bruit de fond.

En bas à gauche de la courbe ROC, on prend la plus grande valeur prédite pour seuil, et on prédit donc que tous les points sont négatifs. En haut à droite, on prend la plus petite valeur prédite comme seuil, tous les points sont au-dessus et donc prédits positifs. Le reste de la courbe décrit toutes les situations intermédiaires.

Prenons un exemple pour mieux comprendre comment construire cette courbe. Nous avons 6 observations, pour lesquelles notre classifieur a retourné les scores suivants (j'ordonne ici les observations par leur score) :

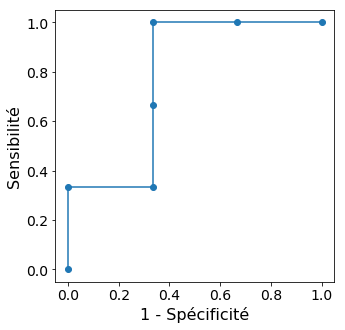
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Étiquette | + | - | + | + | - | - |
| Score | 0.99 | 0.95 | 0.51 | 0.45 | 0.10 | 0.01 |

Quels seuils pouvons-nous considérer ? Nous pouvons commencer par une valeur supérieure à 0.99 : que l'on choisisse 1.0, 10.0 ou 42.7, toutes nos observations seront prédites négatives. Comme aucune observation n 'est prédite positive, notre sensibilité (qui est le taux de vrais positifs TP/P) est égale à 0, de même que l'antispécificité (qui est le taux de faux positifs FP/P).

Considérons ensuite un seuil entre 0.95 et 0.99. Seule la première observation est prédite positive. La sensibilité est donc de 1/3, et l'antispécificité reste nulle. Et ainsi de suite, comme résumé ici :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Seuil | > 0.99 | 0.95 - 0.99 | 0.51 - 0.95 | 0.49 - 0.51 | 0.10 - 0.45 | 0.01 - 0.10 | < 0.01 |
| TP/P | 0 | 1/3 | 1/3 | 2/3 | 1 | 1 | 1 |
| FP/P | 0 | 0 | 1/3 | 1/3 | 1/3 | 2/3 | 1 |

Et voilà la courbe ROC correspondante !

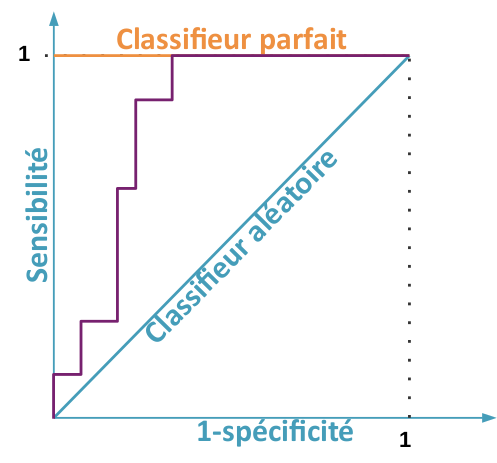


Courbe ROC pour l'exemple ci-dessus

Chaque fois que l'on considère une nouvelle fourchette de valeur de seuil (ou, de manière équivalente, que l'on prédit positive une observation de plus), seulement l'une des deux valeurs (sensibilité ou antispécificité) change (la première si l'observation est effectivement positive, la seconde sinon). La courbe ROC est donc une courbe en escaliers.

Un modèle parfait va systématiquement associer des valeurs plus faibles aux exemples négatifs qu’aux exemples positifs. La courbe ROC correspondante dessine donc le coin supérieur gauche du carré.

Un modèle aléatoire, par contraste, va dessiner la diagonale du carré : quelle que soit le seuil utilisé, comme le modèle est aléatoire, on aura la même proportion de prédictions positives correctes que de prédictions positives incorrectes. Le taux de faux positifs et le taux de vrais positifs seront donc égaux, et ainsi la sensibilité sera égale au complément à 1 de la sensitivité.



Courbe ROC

Traçons la courbe ROC du kNN utilisé à la fin du chapitre 1

y\_pred\_proba = clf.predict\_proba(X\_test\_std)[:, 1]

[fpr, tpr, thr] = metrics.roc\_curve(y\_test, y\_pred\_proba)

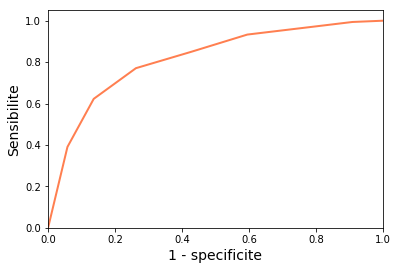
plt.plot(fpr, tpr, color='coral', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('1 - specificite', fontsize=14)

plt.ylabel('Sensibilite', fontsize=14)



Courbe ROC

#### Autres courbes

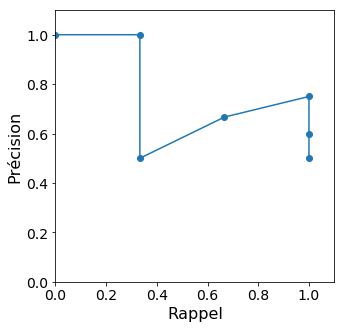
Une autre courbe fréquemment utilisée est la **courbe précision-rappel** (précision en ordonnée et rappel en abscisse), souvent appelée « ***PR curve*** ».

Reprenons notre exemple. Les valeurs de la précision (rapport du nombre de vrais positifs et du nombre de prédits positifs) et du rappel (égal à la sensibilité) sont les suivantes :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Seuil | > 0.99 | 0.95 - 0.99 | 0.51 - 0.95 | 0.49 - 0.51 | 0.10 - 0.45 | 0.01 - 0.10 | < 0.01 |
| TP/P | 0 | 1/3 | 1/3 | 2/3 | 1 | 1 | 1 |
| Précision | - | 1 | 1/2 | 2/3 | 3/4 | 3/5 | 3/6 |

Petit problème : pour le seuil le plus élevé, la précision n'est pas définie car aucune observation n'est prédite positive... Par convention, on choisira souvent une précision de 1 si la première observation à considérer est positive, et une précision de 0 sinon.

Voilà donc notre courbe précision-rappel :

C

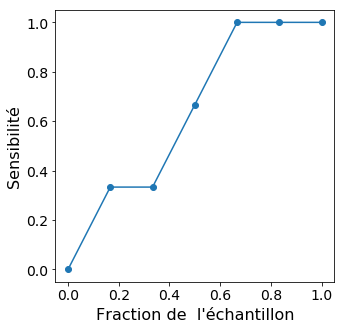
Courbe précision-rappel pour notre exemple.

L'interpolation linéaire que l'on a faite ici (c'est-à-dire de relier les points par des segments) n'est en fait pas une interpolation appropriée, même si c'est souvent ce qui est fait en pratique. Pour en savoir plus, vous pouvez lire l'article suivant : <https://classeval.wordpress.com/introduction/introduction-to-the-precision-recall-plot/>

Enfin, la courbe **lift**, surtout utilisée dans le ciblage marketing, se construit aussi en parcourant le jeu de données ordonné par score. On représente en abscisse la fraction du jeu de données parcourue, et en ordonnée le taux de vrais positifs. Sur notre petit exemple, les valeurs à considérer sont donc :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Étiquette | + | - | + | + | - | - |
| Fraction du jeu de données | 1/6 | 2/6 | 3/6 | 4/6 | 5/6 | 1 |
| TP/P | 1/3 | 1/3 | 2/3 | 1 | 1 | 1 |

On obtient alors la courbe ci-dessous



Courbe lift pour l'exemple ci-dessus

### 

### Utiliser une courbe ROC

On peut résumer la courbe ROC par un nombre : "**l'aire sous la courbe"**, aussi dénotée AUROC pour « Area Under the ROC », qui permet plus aisément de comparer plusieurs modèles.

Un classifieur parfait a une AUROC de 1 ; un classifieur aléatoire, une AUROC de 0.5

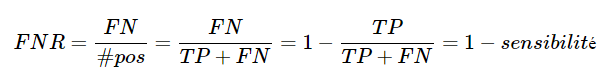
Revenons à l’exemple du vin portugais : l’aire sous la courbe ROC peut être calculée par :

print(metrics.auc(fpr, tpr))

J’obtiens une AUROC de 0.815.

Comment choisir un seuil de décision à partir de cette courbe ? On se fixe soit la spécificité, soit la sensibilité que l'on désire, et on cherche le seuil correspondant.

Prenons l’exemple du vinho verde. Imaginons que l’algorithme doit être capable de détecter efficacement les vins de mauvaise qualité, qui ne seront pas ensuite examinés par un expert humain. On veut alors limiter le nombre de faux négatifs, pour limiter le nombre de rejets infondés. Fixons-nous un taux de faux négatifs tolérable (la proportion de positifs incorrectement prédits négatifs) de 5%. Cela équivaut à une sensibilité de 0.95 :



# indice du premier seuil pour lequel

# la sensibilité est supérieure à 0.95

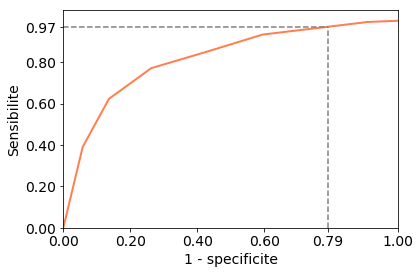
idx = np.min(np.where(tpr > 0.95))

print("Sensibilité : {:.2f}".format(tpr[idx]))

print("Spécificité : {:.2f}".format(1-fpr[idx]))

print("Seuil : {:.2f}".format(thr[idx]))

Utiliser un seuil de 0.29 nous garantit une sensibilité de 0.97 et une spécificité de 0.21, soit un taux de faux positifs de… 79%.



Une sensiblité de 0.97 correspond à une spécificité de (1-0.79)=0.21

### 

### Résumé

* De nombreux modèles de classification retournent des **valeurs réelles**, qui peuvent souvent être interprétées comme la probabilité que le point appartient à la classe positive.
* Dans ce cas, il faut se fixer un **seuil** sur cette valeur réelle pour séparer les négatifs des positifs.
* La **courbe ROC** permet de visualiser comment la spécificité et la sensibilité d’un modèle évolue en fonction de ce seuil.
* L’**AUROC** permet de résumer la courbe ROC en un seul nombre : l’aire sous cette courbe.

## Comparez votre algorithme à des approches de classification naïves

Comment interpréter une mesure de performance ? Si mon modèle atteint une AUC de 0.85 sur mon jeu de test, comment l’interpréter ? Est-ce que ça veut dire que le modèle a bien appris, ou pas ?

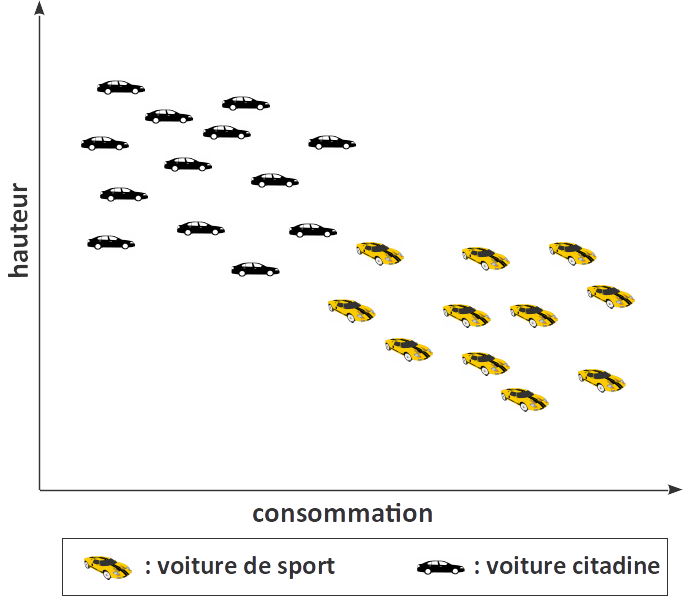
### 

### La performance d'un modèle dépend du jeu de données

Le théorème du "no free lunch" ("il n'y a pas de déjeuner gratuit", c'est-à-dire qu'on ne peut pas avoir le beurre et l'argent du beurre), nous dit qu'aucun algorithme de machine learning ne fonctionne bien sur tous les jeux de données.

Comme nous l'avons vu [précédemment](https://openclassrooms.com/courses/evaluez-et-ameliorez-les-performances-d-un-modele-de-machine-learning/evaluez-un-algorithme-de-classification-qui-retourne-des-valeurs-binaires), les données que nous observons ne suffisent pas à déterminer le modèle. Nous devons rajouter des hypothèses, par exemple le fait que les classes peuvent être séparées par des hyperplans, pour pouvoir construire un modèle. Ces hypothèses et leur validité dépendent du problème étudié.

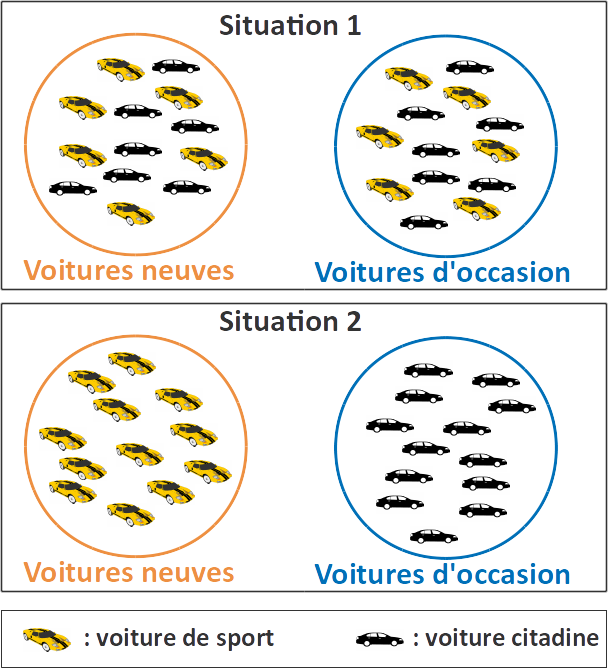
Prenons l'exemple suivant : imaginons que nous cherchions à classifier des voitures en deux classes : citadines ou sportives. Nos voitures sont représentées par leurs dimensions et leur consommation. A priori, deux voitures de la même classe vont avoir des tailles et consommations similaires, et nous allons être capables de construire un bon modèle.



Les citadines sont faciles à distinguer des sportives en utilisant "consommation" et "hauteur"

Imaginons maintenant que nous cherchions à utiliser les mêmes descripteurs et le même algorithme pour un autre problème de classification binaire : séparer les voitures d'occasion des voitures neuves.

C'est a priori beaucoup plus difficile. Nous voulons alors beaucoup d'efforts à développer un modèle compliqué pour notre problème, et mesurons un taux d'erreur très faible sur notre jeu de test. Mais il se pourrait que dans nos données, toutes les voitures d'occasion soient des citadines et toutes les voitures neuves des voitures de sport ! Dans ce cas, nous pourrions obtenir le même taux d'erreur avec un algorithme beaucoup plus simple…



Dans la situation 1, les voitures neuves sont difficiles à distinguer des voitures d’occasion. Mais dans la situation 2, les voitures neuves sont toutes des voitures de sport et les voitures d’occasion sont toutes des citadines

C'est ce qu'on appelle parfois un "**jeu de données faciles"**: tous les algorithmes classiques donnent de bonnes performances.

Pour mieux comprendre la difficulté de notre problème, et évaluer les performances que l'on peut facilement atteindre, on peut utiliser des **approches naïves** : des approches très simples mais qui s'appuient néanmoins sur le jeu de données pour construire un modèle. Ces méthodes ne permettent pas réellement de faire de l'apprentissage, mais servent de point de comparaison pour évaluer nos modèles.

### Approches naïves pour des problèmes de classification (binaire)

Voici quelques approches naïves pour des problèmes de classification. Elles sont implémentées dans la classe  DummyClassifier  du module  dummy  de scikit-learn.

* Prédire la même classe pour tous les échantillons : la classe la plus fréquente dans le jeu d'entraînement. Cette approche naïve nous permet d’évaluer si le modèle que nous proposons a appris « plus » que simplement quelle est la classe la plus fréquente. C’est particulièrement intéressant si une des classes est beaucoup plus fréquente que les autres. Pensez que pour un problème de classification binaire sur des données contenant 90% d’échantillons positifs, un classifieur qui retourne systématiquement « positif » aura une accuracy de 90%.
* Prédire une classe aléatoirement, dans les mêmes proportions que dans le jeu d'entraînement. Cette approche naïve nous permet d’évaluer si les performances que nous observons ne seraient pas simplement dûes aux proportions relatives des classes.
* Retourner aléatoirement des scores selon une distribution uniforme, puis leur appliquer un seuil pour obtenir une prédiction binaire. Cette méthode est recommandée quand on cherche à interpréter une courbe ROC ou une AUROC, elles-mêmes construites à partir de classifieurs qui retournent des scores.

### 

### Résumé

* Les performances d'un modèle **dépendent du jeu de données**.
* Les**approches naïves** sont des approches simples qui n'apprennent pas vraiment mais servent de point de comparaison pour évaluer nos modèles.
* Pour un problème de classification, on peut utiliser **une des approches naïves suivantes** :
  + Retourner toujours la même classe ;
  + Retourner une classe aléatoire ;
  + Retourner un score aléatoire, puis utiliser un seuil.

# Partie 2

Bravo ! Vous avez réussi cet exercice !

### Compétences évaluées

* Mettre en place une procédure de validation qui réduise le risque d'apprentissage d'une classification
* Choisir une ou plusieurs mesures de la performance d'un algorithme de classification

### Question 1

**Vous devez implémenter un algorithme de détection de transactions frauduleuses.**

**Les transactions frauduleuses que votre algorithme ne détectera pas coûteront plus cher à la banque que le coût de traitement d'une transaction non frauduleuse prédite comme fraude.**

**Dans la mesure du raisonnable, vous devez donc minimiser le taux de :**

* + 

Vrai négatif

* + 

Faux négatif

* + 

Faux positif

* + 

Vrai positif

*Le taux de faux négatif représente le taux de fraudes non détectées. C'est bien ce qui coûte le plus cher à la banque.*

### Question 2

**Un algorithme de classification binaire retourne "négatif" ou "positif" de manière aléatoire (avec une probabilité de 0.5 sur chaque classe). On l'évalue sur un jeu de données contenant 90% de "positif" et 10% de "négatif". Quelles seront sa précision et son recall ?**

* + 

précision = 90%

recall = 10%

* + 

précision = 100%

recall = 10%

* + 

précision = 90%

recall = 50%

* + 

précision = 100%

recall = 100%

### 

### Question 3

**On évalue deux modèles dont le but est de classer des baies entre celles qui sont comestibles et celles qui ne le sont pas. On obtient les deux matrices de confusion suivantes :**

**Algorithme 1 :**

|  |  |  |
| --- | --- | --- |
|  | comestible | non comestible |
| prédit comestible | 100 | 3 |
| prédit non comestible | 0 | 97 |

**Algorithme 2 :**

|  |  |  |
| --- | --- | --- |
|  | comestible | non comestible |
| prédit comestible | 96 | 0 |
| prédit non comestible | 4 | 100 |

**Quel est selon-vous l'algorithme le plus intéressant ?**

* + 

Algo 1

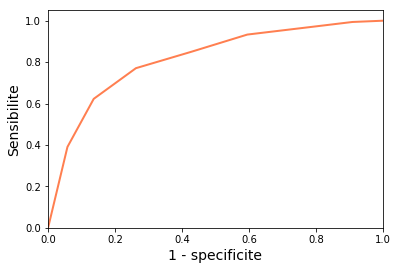
* + 

Algo 2

*Même si l'algorithme 1 possède un taux d'erreur plus faible (3/200 = 1.5% vs l'algorithme 2 4/200 = 2%), on va choisir l'algorithme 2 puisque le taux de faux positif est très important dans le problème étudié, puisque l'on ne veut surtout pas consommer de baies non comestibles.*

### Question 4

**D'après la courbe ROC ci-dessous, quel est le taux de faux négatif le plus bas que l'on puisse avoir si  
on veut une spécificité au moins égale à 0.8 ?**

courbe ROC

* + 

0.7

* + 

0.3

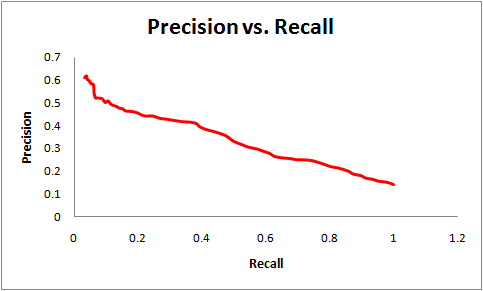
* + 

0

*Le taux de faux négatif est de un moins la sensibilité. Pour une spécificité supérieure à 0.8, la sensibilité est supérieure à 0.95 et donc le taux de faux négatif doit être inférieure à 1 - 0.95 = 0.0.5.*

### Question 5

**D'après la courbe Précision-Rappel ci-dessous, quel est le taux de faux négatif le plus bas que l'on puisse avoir si  
on veut une précision au moins égale à 0.4 ?**

Courbe precision-rappel

* + 

0.4

* + 

1

* + 

0.6

*Le taux de faux négatif est de un moins le recall. Donc pour une précision de 0.4, on a 0.4 de recall soit 0.6 de taux de faux négatif.*

### Question 6

**Un algorithme de classification binaire naïf retourne toujours la classe "positif". On l'évalue sur un jeu de données contenant 80% de "positif" et 20% de "négatif". Quelles seront sa précision et son recall ?**

* + 

précision = 100%,

recall = 100%

* + 

précision = 80%,

recall = 20%

* + 

précision = 100%,

recall = 20%

* + 

précision = 80%,

recall = 100%

### 

### Question 7

**Vous avez un problème de classification binaire dans lequel la classe présente majoritairement dans le dataset est négative. La F-mesure du classifieur naïf  most\_frequent est alors de :**

* + 

1

* + 

0

* + 

0.5

*La précision d'un tel classifieur est de zéro puisqu'il a un taux de TP = 0. Il s'en suit d'après la formule de la F-mesure qu'elle est de même nulle.*

### Question 8

**Dans le cas ou la classe majoritaire présente dans le dataset est négatif, comment, selon vous, avoir une baseline qui donne un F-score utile ?**

* + 

Utiliser un classifieur naïf  most\_frequent

* + 

Utiliser un classifieur naïf  constant de classe positive

* + 

Utiliser un classifieur naïf  constant de classe négative

*Comme on l'a vu dans la question précédente, dans le cas où on utilise  most\_frequent la F-mesure sera nulle donc peu utile. En revanche, si on utilise  constant avec la classe positive on aura bien une précision non nulle (de 1) et par extension la F-mesure sera aussi non nulle (0 < F < 0.67)*

## Évaluez un algorithme de régression

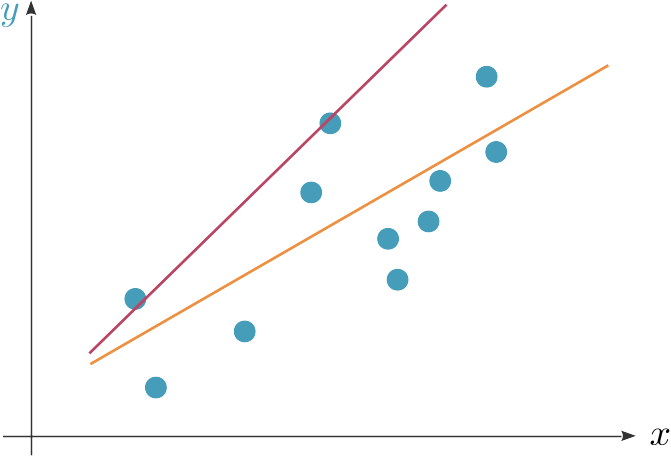
Dans cette troisième partie, nous allons nous concentrer sur les modèles de régression. Il s'agit ici d'utiliser des données étiquetées pour associer une valeur numérique à un objet, comme par exemple de prédire le nombre d'utilisateurs d'un service à un moment donné.

Comme pour les problèmes de classification, il existe plusieurs critères d'évaluation d'un modèle de régression.

### 

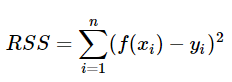
### Mesures d'erreur

Quand nous avons parlé d'évaluer des modèles de classification, nous avons commencé par compter le nombre d'erreurs de prédiction que fait le modèle. **Ce n'est pas approprié pour un problème de régression**. En effet, à quelle précision numérique peut-on dire qu'une prédiction est correcte ou non ? Par ailleurs, on préfère généralement un modèle qui est globalement « plus près des vraies valeurs » à un modèle presque exact pour quelques points.

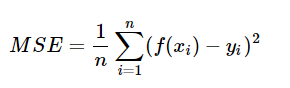


La ligne rose fait des prédictions presque exactes pour deux des points, mais la ligne orange est globalement plus près des vraies valeurs à prédire et représente mieux les données

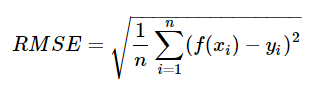
Quantifions cette notion de « plus près des vraies valeurs » : nous allons calculer pour chaque point **xi** du jeu de test la distance entre son étiquette et la valeur prédite et en faire la somme. Le résultat s'appelle la **somme des carrés des résidus**, ou **RSS**, pour Residual Sum of Squares.



Le problème de la RSS, c'est qu'elle est d'autant plus grande qu'on a de données. Pour cette raison, nous allons la normaliser par le nombre **n** de points dans le jeu de test. On obtient ainsi l'**erreur quadratique moyenne**, ou **MSE**, pour Mean Squared Error.



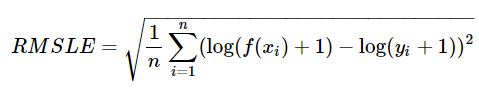
Pour se ramener à l'unité de **y**, on peut prendre la racine de la MSE. On obtient ainsi la **RMSE**, ou Root Mean Squared Error.



Mais la RMSE ne se comporte pas très bien quand les étiquettes peuvent prendre des valeurs qui s'étalent sur plusieurs ordres de grandeur. Imaginons faire une erreur de 100 unités sur une étiquette qui vaut 4 ; le terme correspondant dans la RMSE vaut 100 au carré=10000. C'est exactement la même chose que si on fait une erreur de 100 unités sur une étiquette qui vaut 8000.

Cependant, une prédiction de 104 au lieu de 4, soit une erreur de 2 ordres de grandeur, me parait être une erreur bien plus grande qu'une prédiction de 8100 au lieu de 8000 !

Pour prendre cela en compte, on peut passer les valeurs prédites et les vraies valeurs au log avant de calculer la RMSE. On obtient ainsi la **RMSLE** (Root Mean Squared Log Error) :



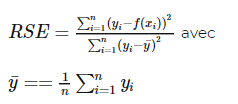
Revenons à notre exemple. Pour la prédiction de 104 au lieu de 4, le terme correspondant dans la RMSLE vaut maintenant 1.75. Celui correspondant à la prédiction de 8100 au lieu de 8000 vaut maintenant 3.10 puissance −5 .

L'astuce du passage au log a fonctionné !

### Coefficient de détermination

Même si les valeurs à prédire ont toutes le même ordre de grandeur, la RMSE peut être difficile à interpréter. Imaginons que notre modèle serve à prédire des prix de vente. Si nous travaillons avec des DVDs, une erreur de 10€ sera importante. Si nous travaillons avec des voitures, une erreur de 10€ sera très faible. Dans le cas d'un DVD ou d'une voiture, vous n'avez pas besoin de voir le jeu de données pour connaître les ordres de grandeur en jeu. Mais ce n'est pas nécessairement le cas !

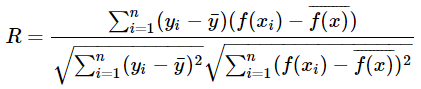
C'est pour cette raison que l'on peut choisir de normaliser la somme des carrés des résidus non pas par le nombre de points **n** dans le jeu de données, mais par une mesure de ce qu'il serait raisonnable de faire comme erreur : la somme des distances entre chacune des valeurs à prédire et leur moyenne. Le résultat s'appelle l'**erreur carrée relative**, ou **RSE** pour Relative Squared Error.



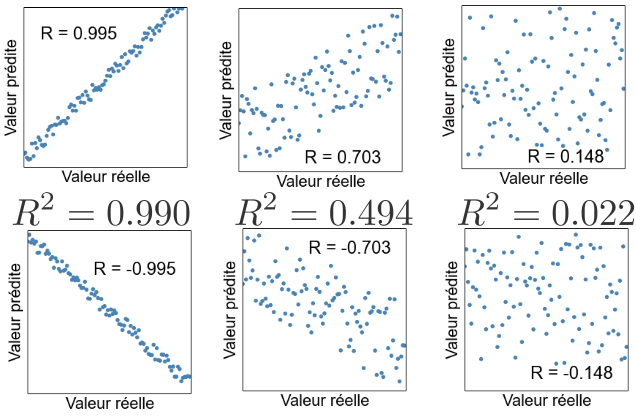
Vous rencontrerez souvent, au lieu de la RSE, son complémentaire à 1, noté .

C'est le **coefficient de détermination**.

On note ce coefficient de détermination  car il s'agit du **carré de la corrélation de Pearson** entre les vraies valeurs et les valeurs prédites.



Le coefficient de détermination nous indique donc à quel point les valeurs prédites sont corrélées aux vraies valeurs. Attention, si les prédictions sont fortement anti-corrélées aux vraies valeurs, le coefficient de détermination sera élevé aussi.



Coefficient de détermination pour quelques exemples de valeurs prédites vs. valeurs réelles

### 

### Résumé

Pour évaluer un modèle de régression, on peut calculer la distance entre valeurs prédites et vraies valeurs. Cela nous donne :

* la somme des carrés des résidus (**RSS**) ;
* la moyenne de cette somme (**MSE**) ;
* la racine carrée de cette moyenne (**RMSE**).

On peut préférer calculer la **corrélation** entre valeurs prédites et vraies valeurs :

* l'**erreur carrée relative** (**RSE**) est la RSS normalisée par la somme des carrés des distances entre les étiquettes et leur moyenne ;
* elle est le complément à 1 du **coefficient de détermination** (R2), qui est le carré de la corrélation de Pearson entre valeurs prédites et vraies valeurs.

## Comparez votre algorithme à des approches de régression naïves

Notre jeu de données est-il « facile » ou « difficile » ? Notre modèle a-t-il vraiment appris quelque chose ?

Comme dans le cas des algorithmes de classification, des approches naïves peuvent nous aider à mieux comprendre nos données et les performances de nos modèles.

### 

### Approches naïves

Dans le cas d'algorithmes de régression, on pourra envisager les deux approches naïves suivantes :

* Un modèle qui « prédit » une **valeur aléatoire**, uniformément entre la plus petite et la plus grande des étiquettes du jeu d'entraînement.
* Un modèle qui retourne **toujours la même valeur**, par exemple la moyenne ou la médiane des étiquettes du jeu d'entraînement.

Ces modèles utilisent le jeu d’entraînement, mais en tirent très peu d’informations. Ils nous donnent une idée du « strict minimum » que l’on peut attendre d’un modèle d’apprentissage. Le modèle que nous avons entraîné est-il capable de faire mieux qu’une de ces approches naïves ?

### TP : Comparez un modèle kNN à des approches naïves

Reprenons notre jeu de données sur le vinho verde blanc.  
Comme les étiquettes sont des nombres entiers, nous pouvons traiter la prédiction de la note de chaque vin à partir de ses caractéristiques physico-chimiques comme un problème de régression.

Commençons par charger les données :

import pandas as pd

import numpy as np

data = pd.read\_csv('winequality-white.csv', sep=";")

X = data[data.columns[:-1]].values

y = data['quality'].values

Séparons nos données en un jeu de test et un jeu d'entraînement :

from sklearn import model\_selection

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.3 ) # 30% des données dans le jeu de test

Standardisons les données :

std\_scale = preprocessing.StandardScaler().fit(X\_train)

X\_train\_std = std\_scale.transform(X\_train)

X\_test\_std = std\_scale.transform(X\_test)

Entraînons un kNN avec k=11 sur ces données :

from sklearn import neighbors

knn = neighbors.KNeighborsRegressor(n\_neighbors=11)

knn.fit(X\_train\_std, y\_train)

Et appliquons le pour prédire les étiquettes de notre jeu de test :

y\_pred = knn.predict(X\_test\_std)

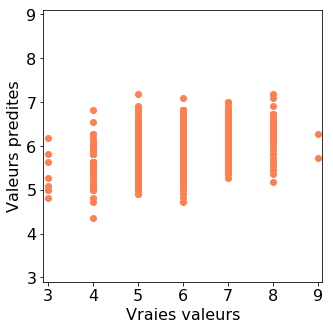
Calculons la RMSE correspondante :

print("RMSE : {:.2f}".format(np.sqrt( metrics.mean\_squared\_error(y\_test, y\_pred) )))

J'obtiens une RMSE de 0.83. Nos étiquettes étant des nombres entiers, nous faisons en moyenne une erreur inférieure à la plus petite différence possible entre deux notes.

Nous pouvons visualiser les résultats sur un graphique, en représentant en abscisse les vraies valeurs des étiquettes, et en ordonnée les valeurs prédites.

plt.scatter(y\_test, y\_pred, color='coral')



Valeurs prédites vs valeurs réelles

Comme nos étiquettes prennent des valeurs entières entre 3 et 8, nous avons beaucoup de points superposés aux mêmes coordonnées. Pour mieux visualiser les données, nous pouvons utiliser comme marqueurs des cercles dont la taille est proportionnelle au nombre de points qui sont présents à ces coordonnées.

sizes = {} # clé : coordonnées ; valeur : nombre de points à ces coordonnées

for (yt, yp) in zip(list(y\_test), list(y\_pred)):

if (yt, yp) in sizes:

sizes[(yt, yp)] += 1

else:

sizes[(yt, yp)] = 1

keys = sizes.keys()

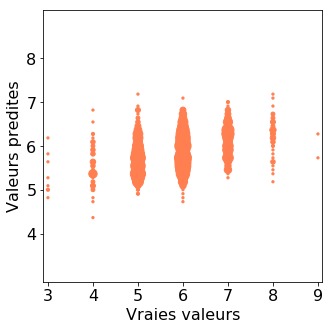
plt.scatter(

[k[0] for k in keys], # vraie valeur (abscisse)

[k[1] for k in keys], # valeur predite (ordonnee)

s=[sizes[k] for k in keys], # taille du marqueur

color='coral', alpha =0.8)



Valeurs prédites vs. Valeurs réelles. La taille des marqueurs est proportionnelle à celle du nombre de points à ces coordonnées

On note ainsi une accumulation de prédictions correctes sur la diagonale. Néanmoins le modèle n'est pas très précis dans ses prédictions.

Pour mieux comprendre notre modèle, comparons-le à une première approche naïve, qui consiste à prédire des valeurs aléatoires, distribuées uniformément entre les valeurs basse et haute des étiquettes du jeu de données d'entraînement.

y\_pred\_random = np.random.randint(np.min(y), np.max(y), y\_test.shape)

Calculons la RMSE correspondante :

print("RMSE : {:.2f}".format(np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred\_random))))

J'obtiens une RMSE de 1.93, ce qui est bien supérieur à la RMSE obtenue par notre modèle kNN. Notre modèle a ainsi réussi à bien mieux apprendre qu'un modèle aléatoire.

Cependant, beaucoup de nos vins ont une note de 6, et beaucoup de nos prédictions sont autour de cette valeur. Comparons maintenant notre modèle à un modèle aléatoire qui retourne systématiquement la valeur moyenne des étiquettes du jeu de données d'entraînement.

Nous pouvons utiliser pour cela la fonction correspondante du module "dummy" de scikit-learn.

from sklearn import dummy

dum = dummy.DummyRegressor(strategy='mean')

# Entraînement

dum.fit(X\_train\_std, y\_train)

# Prédiction sur le jeu de test

y\_pred\_dum = dum.predict(X\_test\_std)

# Evaluate

print("RMSE : {:.2f}".format(np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred\_dum)) ))

J'obtiens une RMSE de 0.91, qui est supérieure à celle de 0.83 obtenue par le kNN. Le kNN a donc appris plus que la moyenne des étiquettes…

En fait, l'algorithme des k plus proches voisins ne donne pas de très bons modèles sur ce problème, mais a l'avantage d'être assez simple à comprendre.

## Entraînez-vous : sélectionnez le nombre de voisins dans un kNN pour une régression

### 

### À vous de jouer !

Pour vous entraîner, réalisez cet exercice étape par étape. Une fois terminé, vous pouvez comparer votre travail avec les pistes que je vous propose.

#### Votre mission

Dans cette activité, vous utiliserez la version régression du K-nn afin de prédire la qualité du vin. Vous évaluerez votre modèle à l’aide des méthodes étudiées dans cette partie pour **optimiser votre algorithme** et **choisir les meilleurs hyper-paramètres** (le nombre de voisins), à nouveau à l’aide d’une grid search, à implémenter vous même.

#### Objectif

Dans cette activité, vous devez **optimiser l’erreur quadratique moyenne** (Mean Squared Error, MSE). Vous pourrez observer le comportement de la MSE et la comparer à celui de R^2.

Vous comparerez les performances à l’aide d’une baseline naïve ainsi que les différentes heuristiques, comme effectué dans le TP précédent.  Pour cela, vous utiliserez cette fois le second dataset (<https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv>)

À l'issue de votre mission, vous aurez créé un classeur iPython où vous interpréterez les valeurs des différentes heuristiques utilisées ainsi que le choix final.

À vous de jouer !

### Vérifiez votre travail

Alors, vous êtes allé au bout ? Suivez le guide pour vérifier votre travail !

Vérifiez que les interprétations sont correctes :

* La meilleure performance en validation croisée est bien obtenue pour la valeur de k choisie.
* La performance est meilleure pour le modèle choisi que pour la baseline naïve.
* Remarque : on s’attend à obtenir des performances proches pour différentes valeurs de k, et donc à ce que la valeur optimale de k soit différente pour différents étudiants (qui auront utilisé des folds différents).