Intelligence Artificielle

Pr. Hiba Chougrad
Année-universitaire: 2023-2024

Plan

1. Introduction générale et Agents Intelligents
2. Logique du premier ordre
3. Machine Learning : Pré-traitement des données
4. Machine Learning : Supervised vs Unsupervised
5. Machine Learning : Construire un bon modèle
6. Machine Learning : Raisonnement probabiliste et réseaux bayésiens
7. Machine Learning: Algorithmes d’apprentissage automatique
8. Machine Learning: Apprentissage par renforcement, vision par ordinateur, NLP, Deep Learning

Plan

1. Introduction générale et Agents Intelligents
2. Logique du premier ordre
3. Machine Learning : Pré-traitement des données
4. Machine Learning : Supervised vs Unsupervised
5. **Machine Learning : Construire un bon modèle**
6. Machine Learning : Raisonnement probabiliste et réseaux bayésiens
7. Machine Learning: Algorithmes d'apprentissage automatique
8. Machine Learning: Apprentissage par renforcement, vision par ordinateur, NLP, Deep Learning

Machine Learning : Construire un bon modèle

Objectifs

- Dans ce chapitre on va essayer de développer une intuition de la notion d'**apprentissage.**

- Ce que **l'apprentissage** représente lorsqu'on parle de Machine Learning.

- C'est à dire **d'algorithmes** qui peuvent **améliorer leur performance à partir d'expériences (d'exemples)**.

Construire un modèle

- L’objectif du Machine Learning est de **trouver un modèle** qui effectue une **approximation** de la réalité (le phénomène à l’origine des données), à l’aide de laquelle on va pouvoir effectuer des prédictions.

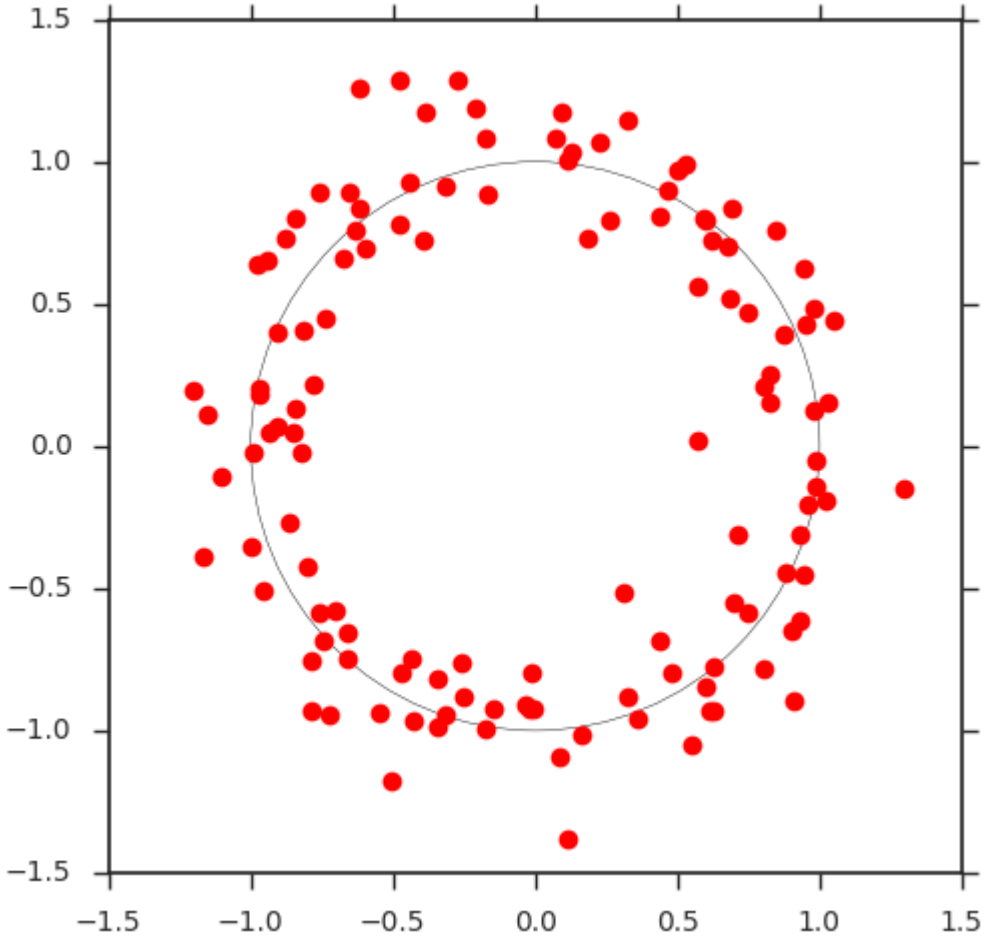
- Et forcément, parce qu’on fait une approximation, on a une **perte d’information** qui est un bruit non modélisé, qu’on estime *indépendant* (c’est à dire non représentatif du phénomène).

- La formule se résume ainsi à l'équation suivante :

Données = modèle + bruit indépendant

Construire un modèle

Données = modèle + bruit indépendant

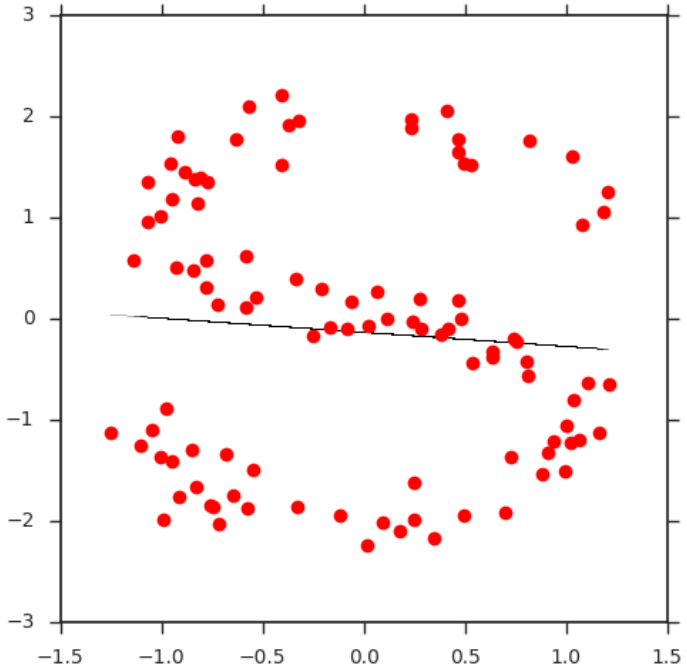


Ici on voit facilement qu'on peut approximer le phénomène à l'origine des données par un cercle

Construire un modèle

- Le **modèle** représente une **contrainte** de “forme” non dépendante des données.
- Par exemple, si je décide d’effectuer une *régression linéaire*, je contrains mon modèle à avoir la forme d’une droite, pas d’un cercle. Malgré tous mes efforts, je ne pourrais alors pas obtenir d’autres formes qu'une droite pour modéliser mes données, c’est donc un choix très important !

- Dans ce cas, par exemple, le modèle ne convient pas aux données
- Le type de modèle est mal choisi.



Construire un modèle

- Une fois ce choix de **contrainte** est effectué la question d'**apprentissage** est alors équivalente à construire le **modèle qui se rapprochera *le plus* des données d'exemples**.

- Les modèles sont le plus souvent représentés par un ensemble de **paramètres** qu'on mettra dans un vecteur θ. Par exemple une droite peut être représentée par l'équation $y = \theta^T x$

- On parle alors de ***modèle paramétriques***, et l'apprentissage du modèle revient dans ce cas à trouver la valeur optimale de θ.

Utiliser la fonction *loss*, ou la perte d'information

- En apprentissage supervisé, la notion principale est celle de **perte d'information** (***loss*** en anglais): Elle détermine à quel point notre modélisation du phénomène, qui est une approximation de la réalité, perd de l'information par rapport à la réalité observée à travers les données d'exemple.

Plus la perte d'information diminue, plus on se rapproche de la réalité, et meilleur est notre modèle.

- On peut retrouver plusieurs types de perte.

Minimiser le Risque Empirique

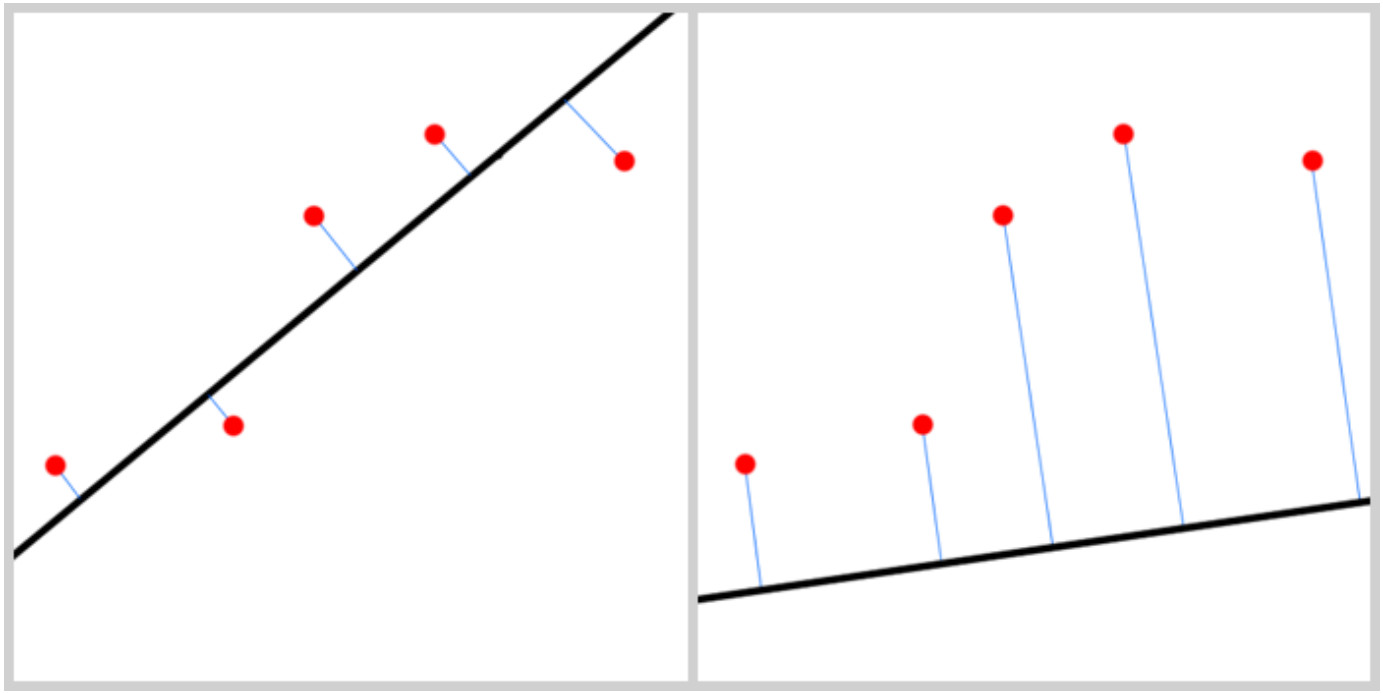
- Une première manière de représenter cette perte se fait par ce qu'on appelle **l'erreur** ou le **risque**, qui traduit l'éloignement des données par rapport à la prédiction effectuée par le modèle considéré.

- La distance la plus utilisée pour mesurer cet éloignement est l'**erreur quadratique** (la distance euclidienne entre un point et le modèle).

- Souvent, on ne peut pas calculer directement l'erreur mais on va utiliser une approximation à partir des données qui sont notre seule ressource. On va ainsi sommer sur toutes nos données d'exemples l'erreur effectuée du modèle. On appelle cette erreur le **risque empirique**.

- Lors de l'apprentissage de notre modèle, on cherche alors à **converger vers le minimum** de risque empirique.

Minimiser l’erreur du modèle



À gauche, on ne perd pas trop d'information. À droite par contre, on est trop éloigné de la réalité représentée par les points.

- **Depuis la figure, donnez un exemple d’erreur quadratique?**
- **Depuis la figure, quel est le risque empirique de la figure à gauche?**
- **En comparant le risque empirique des deux cas devant, lequel des deux va être le plus grand?**

Une histoire d’optimisation numérique

- Au final, **l'apprentissage c'est souvent une histoire d'optimisation numérique** ! Les fonctions de perte sont des exemples illustratifs de l'approche qu'on développe pour construire un algorithme de machine learning.

- Une grande partie des algorithmes s’appuient donc sur des **méthodes d’optimisation numérique**, c’est à dire des méthodes qui vont rechercher un maximum ou un minimum d’une fonction déterminée (vous entendrez parler de *loss*, *entropy*, *energy*, *likelihood*, etc.) de manière **exacte** ou **approximée**.

Une histoire d'optimisation numérique

- En pratique, **les algorithmes d'optimisation sont déjà implémentés dans les librairies** que vous utiliserez (par exemple dans ***scikit-learn***), et sont intégrés au modèle que vous voudrez créer. Toute les notions abordées dans ce cours sont donc encapsulées dans des fonctions, que vous aurez juste à appeler en leur passant vos données d'entraînement.

- Cependant, vous aurez souvent **besoin de paramétrer ces algorithmes d'optimisation**, ou même de trouver les hyperparamètres du modèle.

Exemple

Exemple (Loyer à payer)

- La **régression linéaire** est un premier exemple simple de la manière dont un algorithme peut *apprendre* un modèle.

- Reprenons notre problématique des loyers abordée dans la partie précédente du cours. La question qu'on essaie de résoudre est : *Étant donné les caractéristique de mon appartement,* ***combien*** *devrais-je normalement payer mon loyer?*

- Imaginons pour l'instant que la seule caractéristique dont nous disposons est la surface de l'appartement. Notre *training set* est un ensemble de N = 545 observations de surface et leur loyer associé : (x,y) = (surface, loyer)

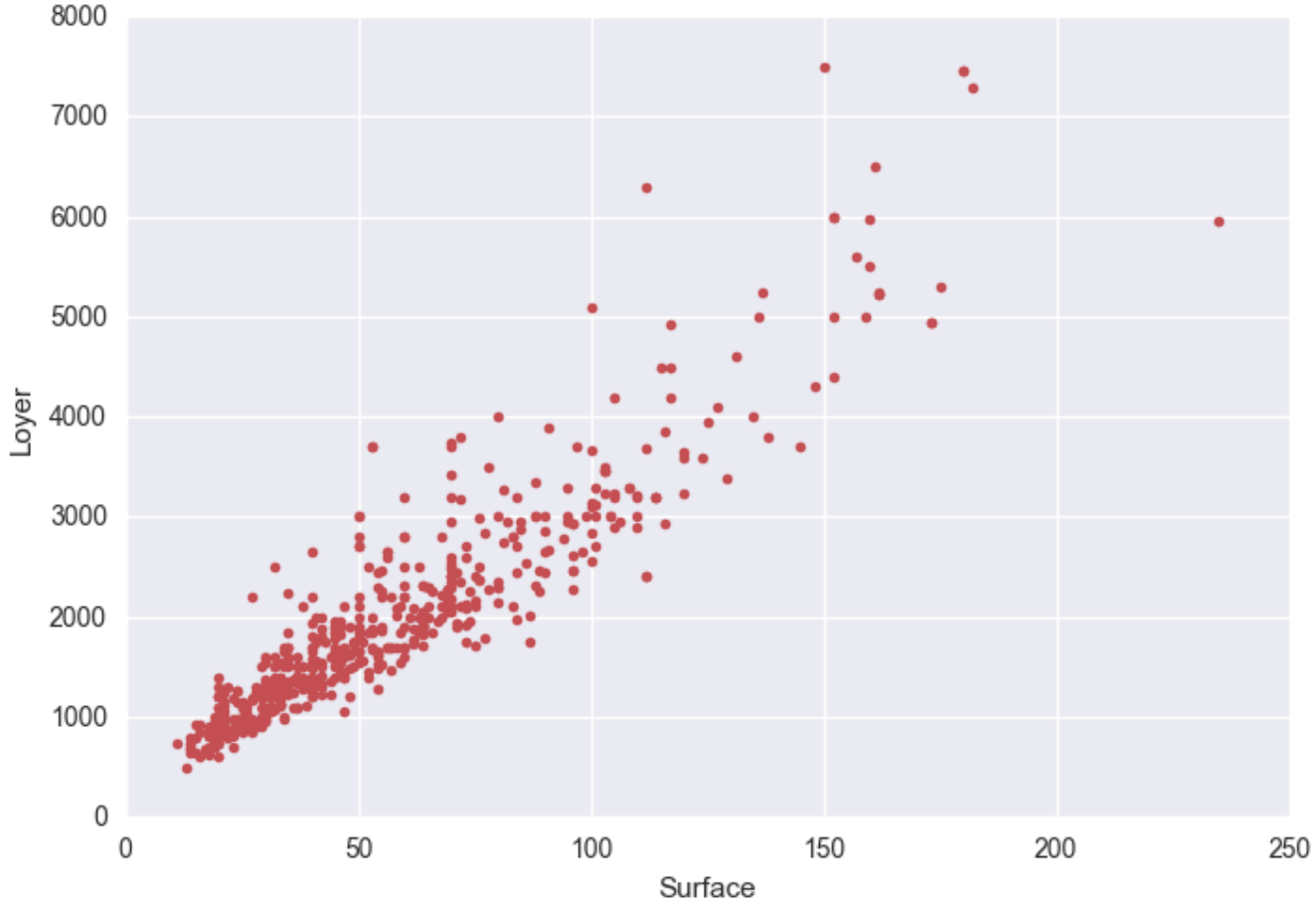
Exemple (Loyer à payer)

- On va charger et visualiser ces données là, juste pour avoir une meilleur idée. On va afficher le loyer en fonction de la surface

loyer,surface
1330,37
1400,32
904,26
955,3
2545,7
970,24
1560,41
1960,67
2000,63
2600,7
3280,81
16000,347
980,26
1250,36
752,19
815,2
1147,25
1500,35
3587,12
1355,4
1245,3
1100,25
1120,21
1225,44

1612,48
1650,5
1533,54
1663,54
1584,45
1680,53
1700,4
1700,35
1700,4
1700,35
1700,6
1600,3
1700,34
1710,64
1600,47
1750,43
1765,62
1790,5
1689,53
1795,5
1800,5
1700,6
1800,42
1850,52
1850,45
...

Exemple (Loyer à payer)



Clairement d'après la visualisation, on peut se dire **que le montant du loyer dépend de manière linéaire de la surface du logement**. On peut donc émettre une hypothèse de modélisation qui est que le phénomène possède la forme d'une droite.

Reformuler le problème dans l'espace d'hypothèse : une droite

- La régression linéaire s'appuie sur l'hypothèse que les données proviennent d'un phénomène qui à la forme d'une droite, c'est à dire qu'il existe une relation linéaire entre l'entrée (les observations) et la sortie (les prédictions).

- Nous avons donc notre contrainte de modèle qui doit être sous la forme $\hat{y} = \theta^T x$ avec $x = (x_1, x_2, \dots, x_N)$ et $\hat{y} = (\hat{y}_1, \hat{y}_2, \hat{y}_3, \dots, \hat{y}_N)$
- Dans notre cas, puisqu'on est en une dimension, on peut écrire pour un point x_i, $\hat{y}_i = \theta_0 + \theta_1 x_i$
- Notre objectif est donc de trouver la droite paramétrée par $\theta = (\theta_0, \theta_1)$ qui convient (**fit**) le mieux aux données d'entraînement.

Définir la fonction *loss*

- On rappelle que les données sont ainsi générées :

  Données = modèle + bruit indépendant

C'est à dire ici :

  Données = $\boldsymbol{\theta^T x}$ + bruit indépendant

- Pour effectuer une régression linéaire, on doit ensuite choisir une fonction de perte, on choisit par exemple la distance euclidienne.
- La distance euclidienne d'une observation y par rapport au modèle $\hat{y}$ est: $\|\hat{y} - y\|_2 = \|\theta^T x - y\|_2$
- Le risque empirique est: $E = \sum_{i=1}^{N} \|\theta^T x_i - y_i\|_2$
- On va donc chercher à trouver le θ qui minimise cette fonction de perte E tel que: $\hat{\theta} = argmin_\theta E$

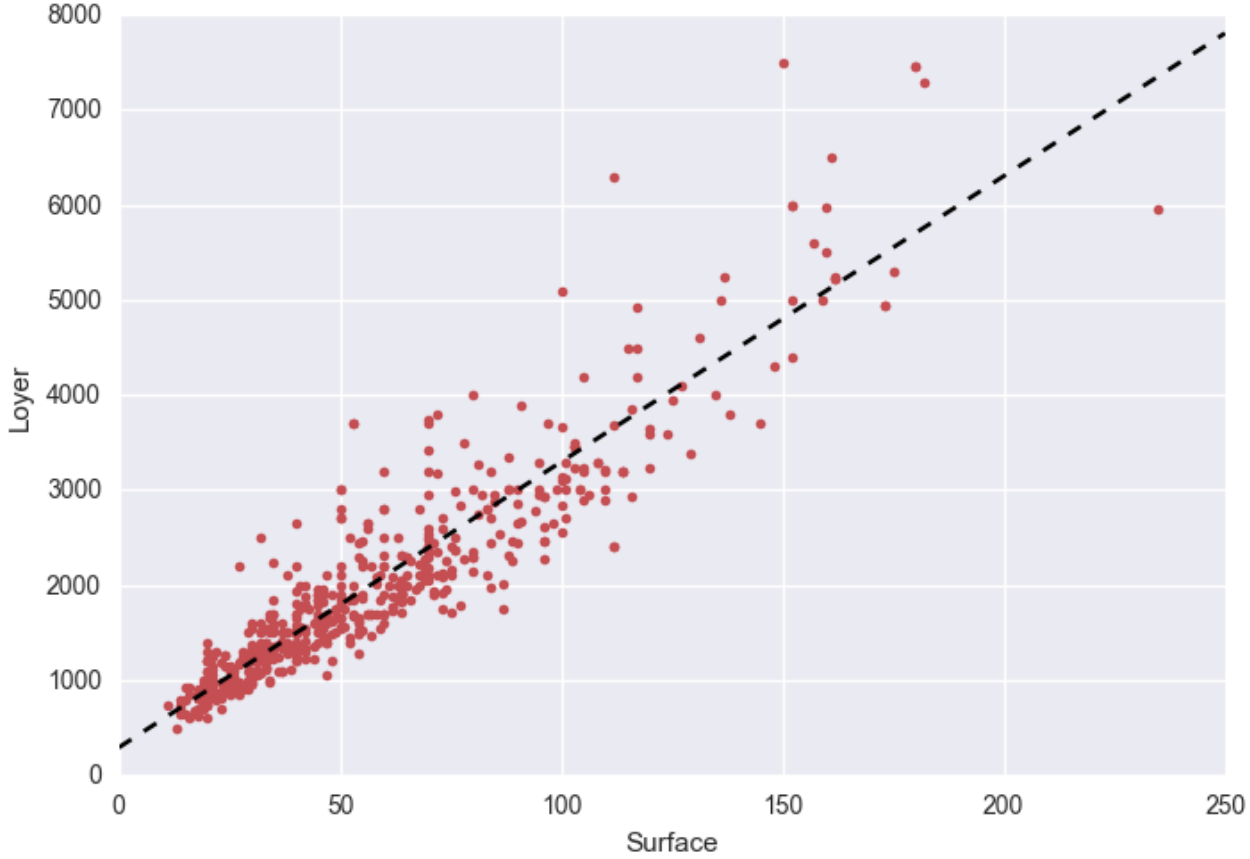
Apprentissage : trouver le θ optimal

- Pour la régression linéaire, la solution de l'équation de minimisation est exacte: $\widehat{\boldsymbol{\theta}} = (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T y}$
- On cherche à minimiser la fonction E en fonction de $\boldsymbol{\theta}$. La fonction est convexe (c'est une somme de carrés) donc elle possède un optimum global (en l'occurrence un minimum), qui se trouve à l'endroit où la dérivée première en $\boldsymbol{\theta}$ est nulle.
- On trouve donc $\boldsymbol{\theta_0} = \boldsymbol{294.30011913}$ et $\boldsymbol{\theta_1} = \boldsymbol{30.04180999}$ ***et***

N.B: Dans ce cas précis, il existe une solution exacte. D'autres fois on se contentera seulement d'une approximation en utilisant un algorithme appelé *descente du gradient.*

Apprentissage : trouver le θ optimal

- Notre modèle final qui convient (**fit**) les données sera donc dans notre cas (approximativement) : **Loyer= 30×Surface + 294.3**
- On représente alors cette droite pour voir si elle colle bien aux données:



Utiliser le modèle pour effectuer des prédictions

- Maintenant qu'on a notre paramètre $\boldsymbol{\theta}$, c'est à dire qu'on a trouvé la droite qui **fit** le mieux nos données d'entraînement, on peut effectuer des prédictions sur de nouvelles données, c'est à dire prédire le loyer en fonction de la surface qu'on nous donne en entrée, en appliquant directement la formule du modèle.

- Par exemple, si on l'applique pour une surface de 35m carré : ***Loyer (surface de 35m²) ?***

Utiliser le modèle pour effectuer des prédictions

- Maintenant qu'on a notre paramètre $\boldsymbol{\theta}$, c'est à dire qu'on a trouvé la droite qui **fit** le mieux nos données d'entraînement, on peut effectuer des prédictions sur de nouvelles données, c'est à dire prédire le loyer en fonction de la surface qu'on nous donne en entrée, en appliquant directement la formule du modèle.

- Par exemple, si on l'applique pour une surface de 35m carré : ***Loyer (surface de 35m²) ?***

Loyer = 30× 35 + 294.3 = 1299,3 €

Résumé

- À partir d'une problématique et d'une dataset, nous avons considéré une **hypothèse** de travail pour contraindre le modèle : ici nous nous sommes placés dans le cas d'une régression linéaire, qui correspond à contraindre la forme du modèle à une droite.

- Nous avons décomposé l'entraînement de ce modèle sur les observations, afin de déterminer le paramètre (Slope et Intercept) de la droite optimale pour ces données. C'est cette partie que l'on appelle **apprentissage du modèle**.

- À l'aide du modèle ainsi trouvé, nous pouvons maintenant effectuer des **prédictions** du montant de loyer à partir de n'importe quelle surface donnée.

- On peut toujours **améliorer ce modèle** (une fois qu'on saura évaluer ses performances) en testant par exemple d'autres hypothèses, en ajoutant de nouvelles caractéristiques sur les observations ou en testant d'autres types de *loss* qui seront peut-être plus appropriés pour ce cas...

Qu’est ce qui fait un bon Modèle?

La Généralisation

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

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 (i.e. qu'il n'a jamais vu) : c'est bien pour cela que l'on parle d'apprentissage.

Le problème?!

- Prenons un exemple:
  - *Imaginez que vous voulez catégoriser des images : certaines représentent des chats, d'autres non.*
  - *Pour construire votre modèle, vous disposez d'une dataset 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). Dans le cas contraire, répondez (pas-chat) .*
- Quel est le problème?

Le problème?!

- Les images de chats sont **très différents**, s'il apprend à reconnaitre les pixels de la dataset, ceci ne l'aiderait pas à catégoriser de nouvelles images de chats **jamais-vues** (qu'on ne trouve pas dans le dataset initial)

- Cet algorithme marche très bien sur le dataset initial, mais **il n'a rien appris (***il ne comprend pas ce que c'est « un chat »***)** : et donc, il ne sait pas faire de **prédictions** !

Évaluer un modèle sur le dataset 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.

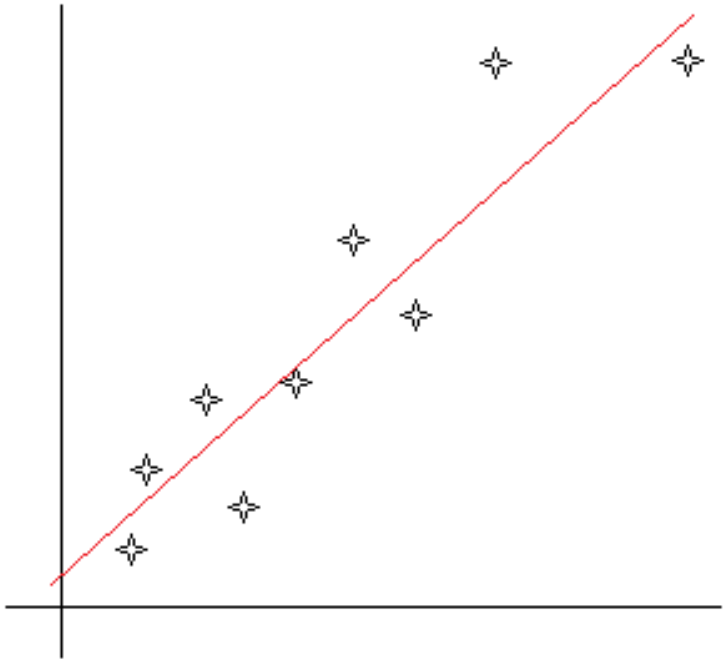
L’Overfitting

- L’ **Overfitting** (sur-apprentissage):

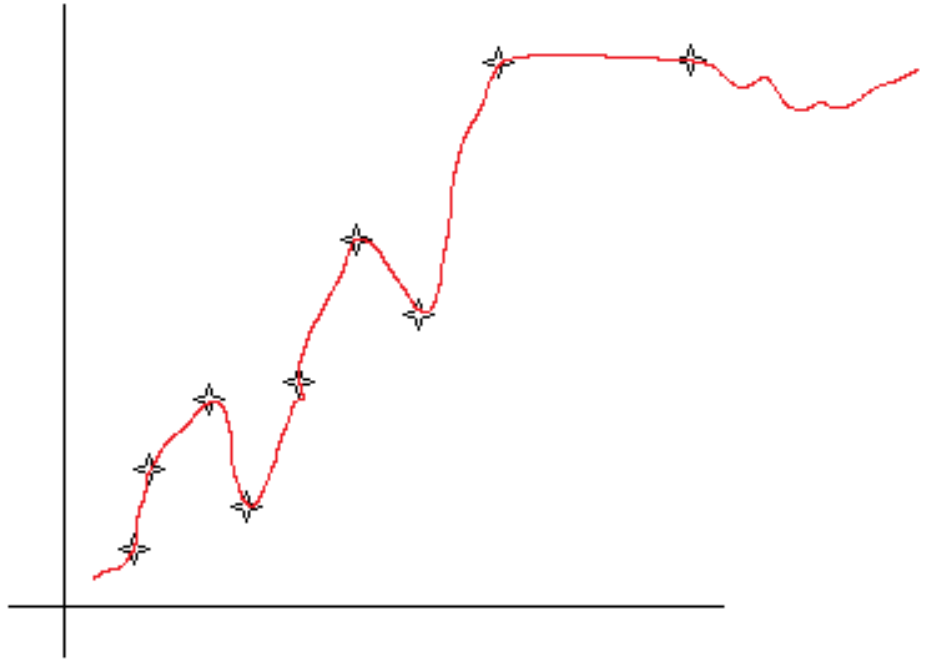
On peut facilement se retrouver sans le vouloir avec un modèle qui **colle trop aux données** sur lesquelles on apprend (Les données d’entraînement), et qui soit trop sensible à leurs moindres variations pour bien les représenter.

Un tel modèle aura de très bonnes performances sur le training set mais sera mauvais sur de nouvelles données.

L'Overfitting

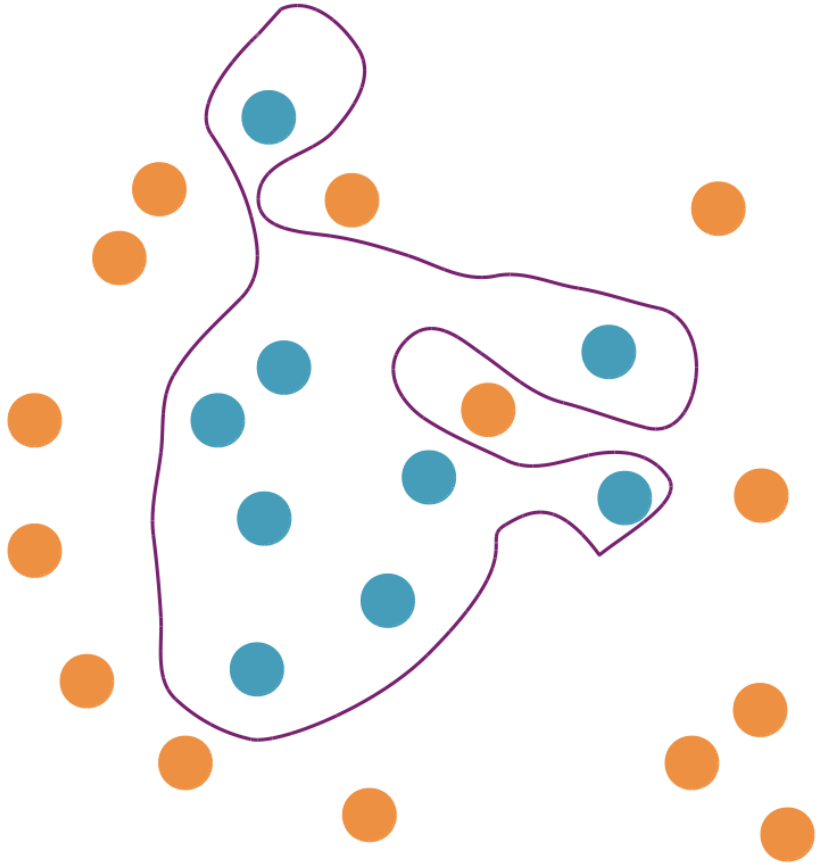


Apprentissage correct



Sur-apprentissage

L’Overfitting



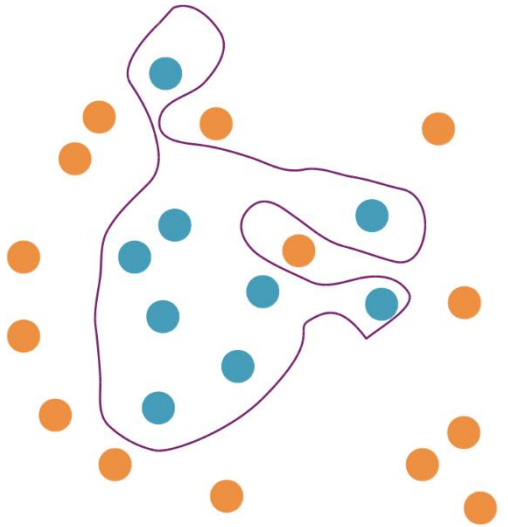
Sur cet exemple, le modèle (la ligne violette) qui sépare les points bleus des points oranges colle bien aux données, il ne fait aucune erreur.

Mais est-ce qu’il modélise bien la réalité ?

L'Overfitting

- Un modèle qui **sur-apprend** est un modèle qui est **trop complexe** par rapport à la réalité qu'il essaie de représenter.

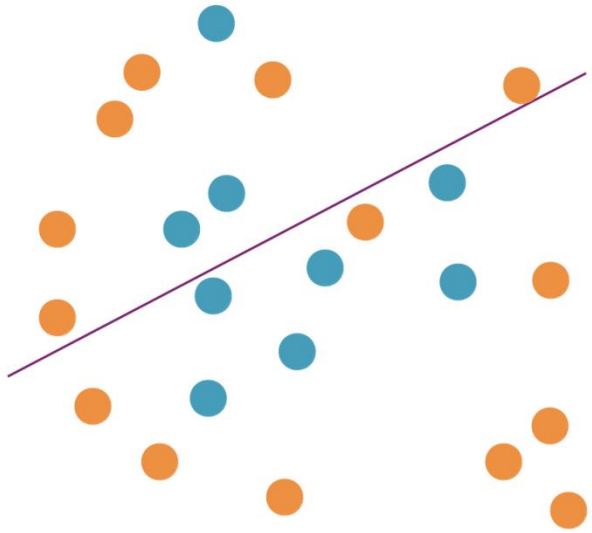
- Nous avons tendance à préférer des modèles simple. Ainsi, **coller de trop près aux données est une mauvaise idée**.



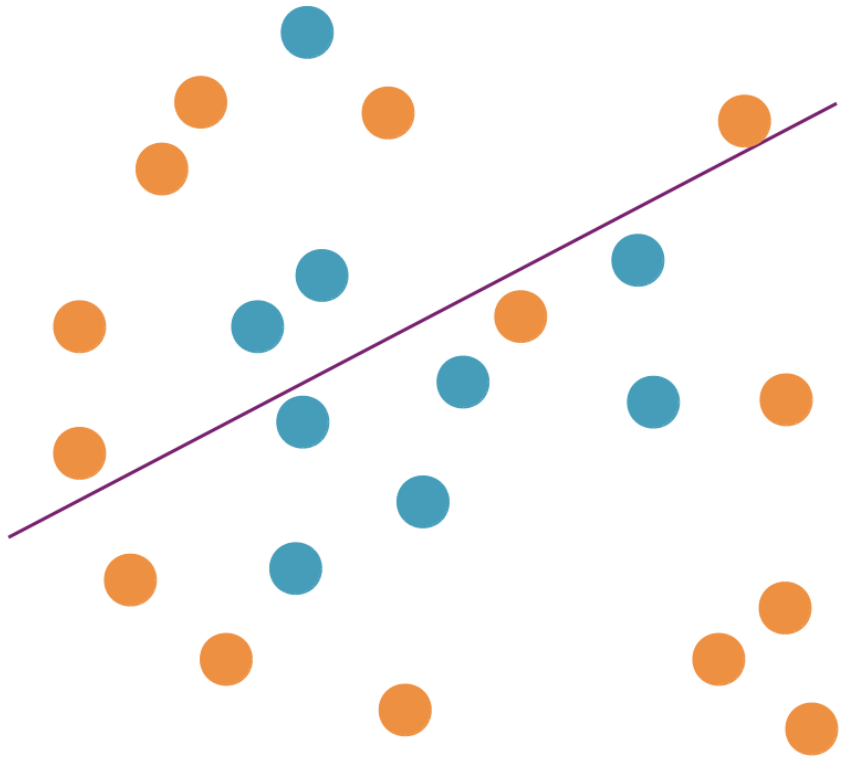
Underfitting

- Le **Underfitting** (sous-apprentissage):

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.

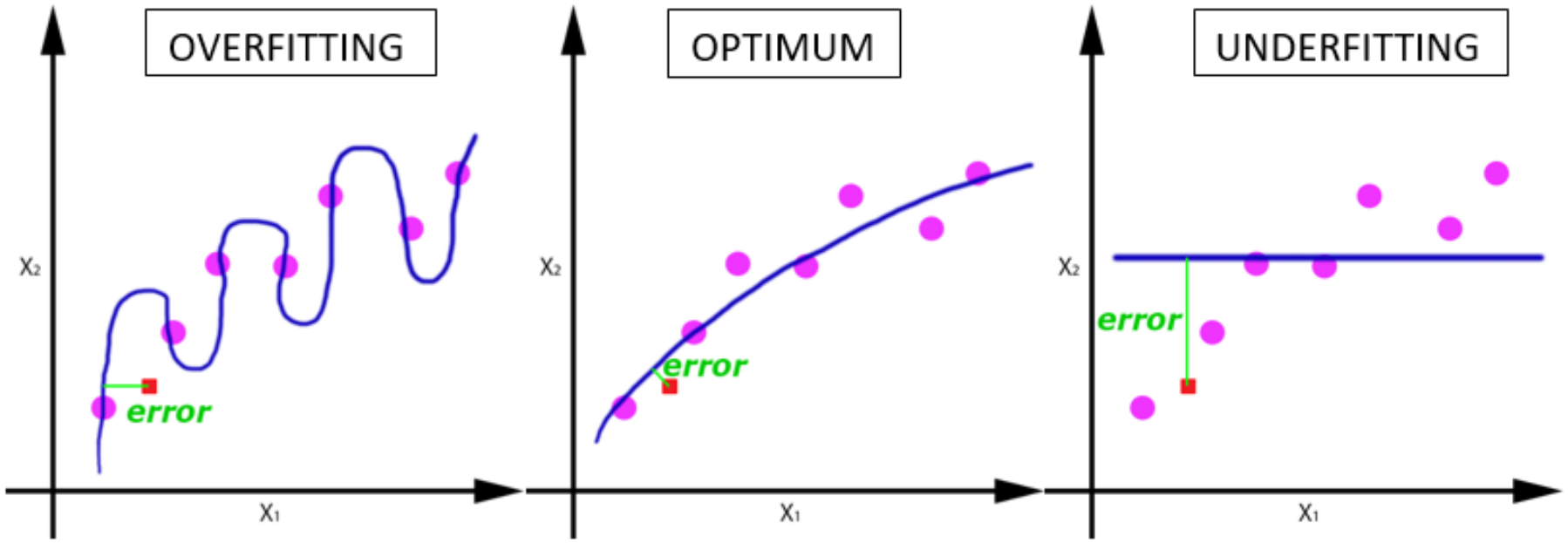


Underfitting

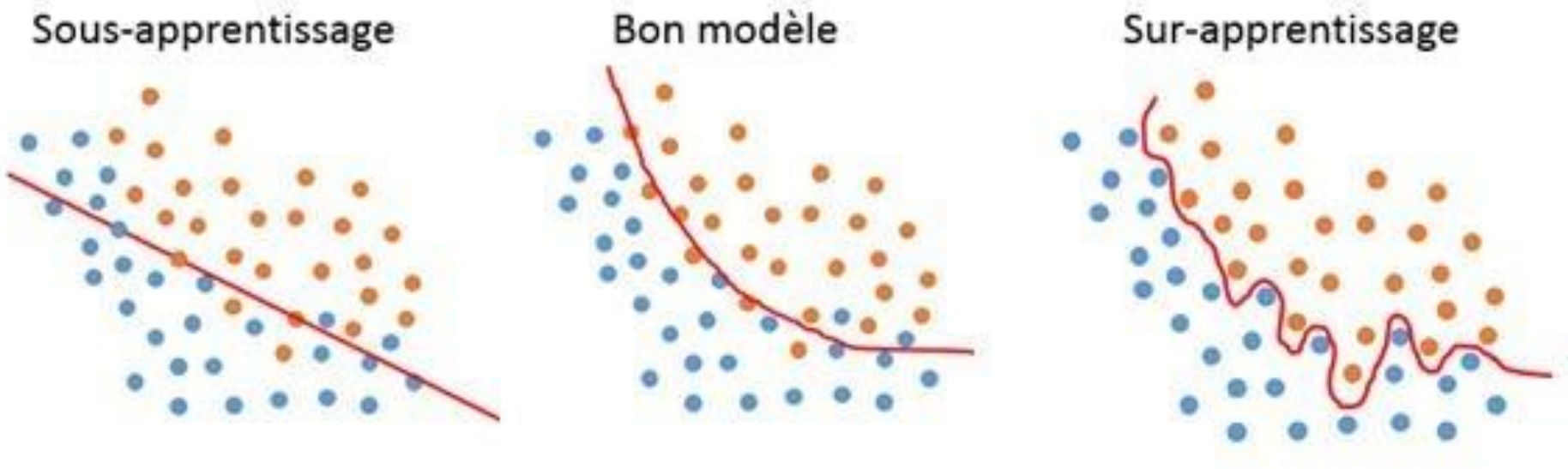


Ce modèle, trop simple, représente trop mal les données pour prédire.

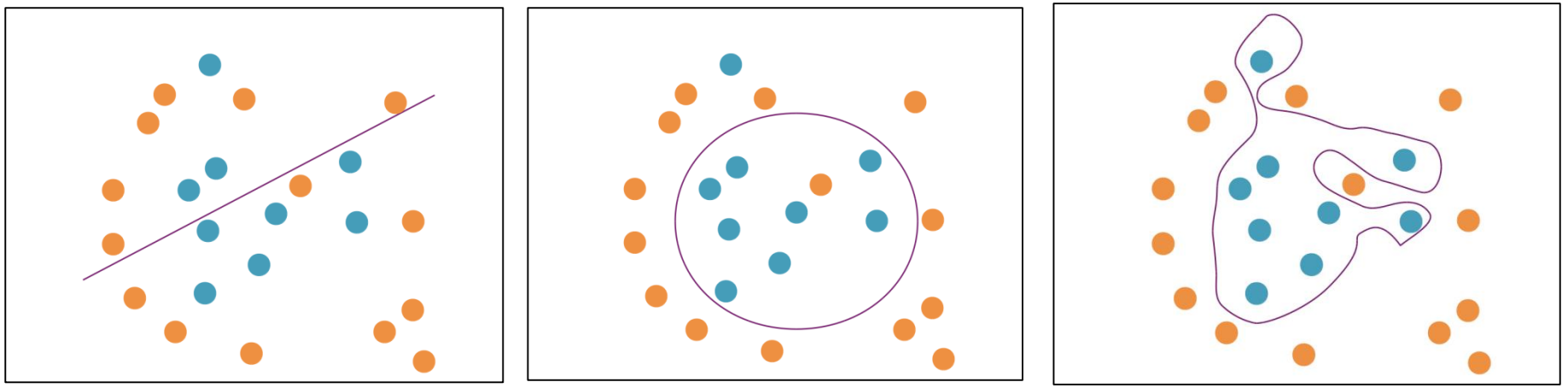
Overfitting vs Underfitting vs Bon modèle



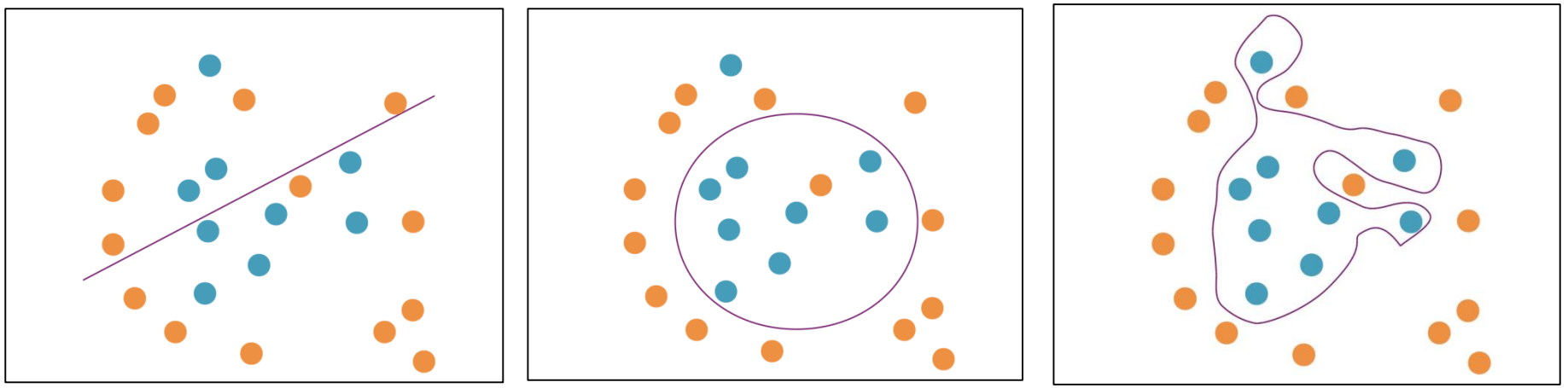
Overfitting vs Underfitting vs Bon modèle



Overfitting vs Underfitting vs Bon modèle



Overfitting vs Underfitting vs Bon modèle

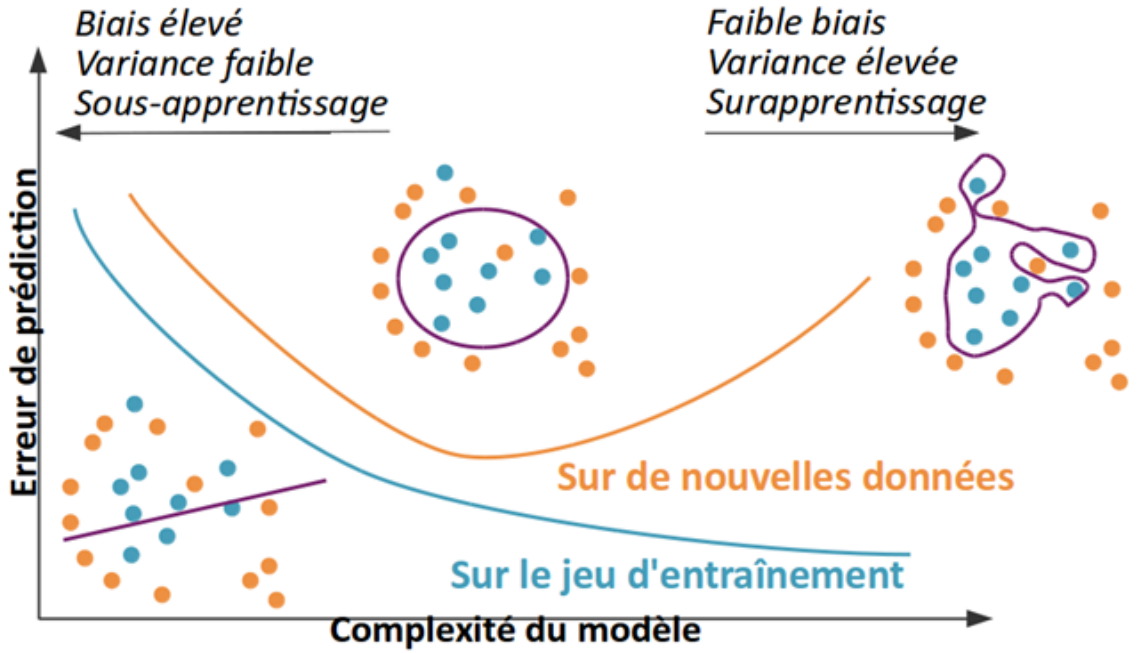


Le modèle du milieux fait des erreurs sur le jeu d'apprentissage, mais il va probablement **mieux généraliser**

Compromis biais-variance

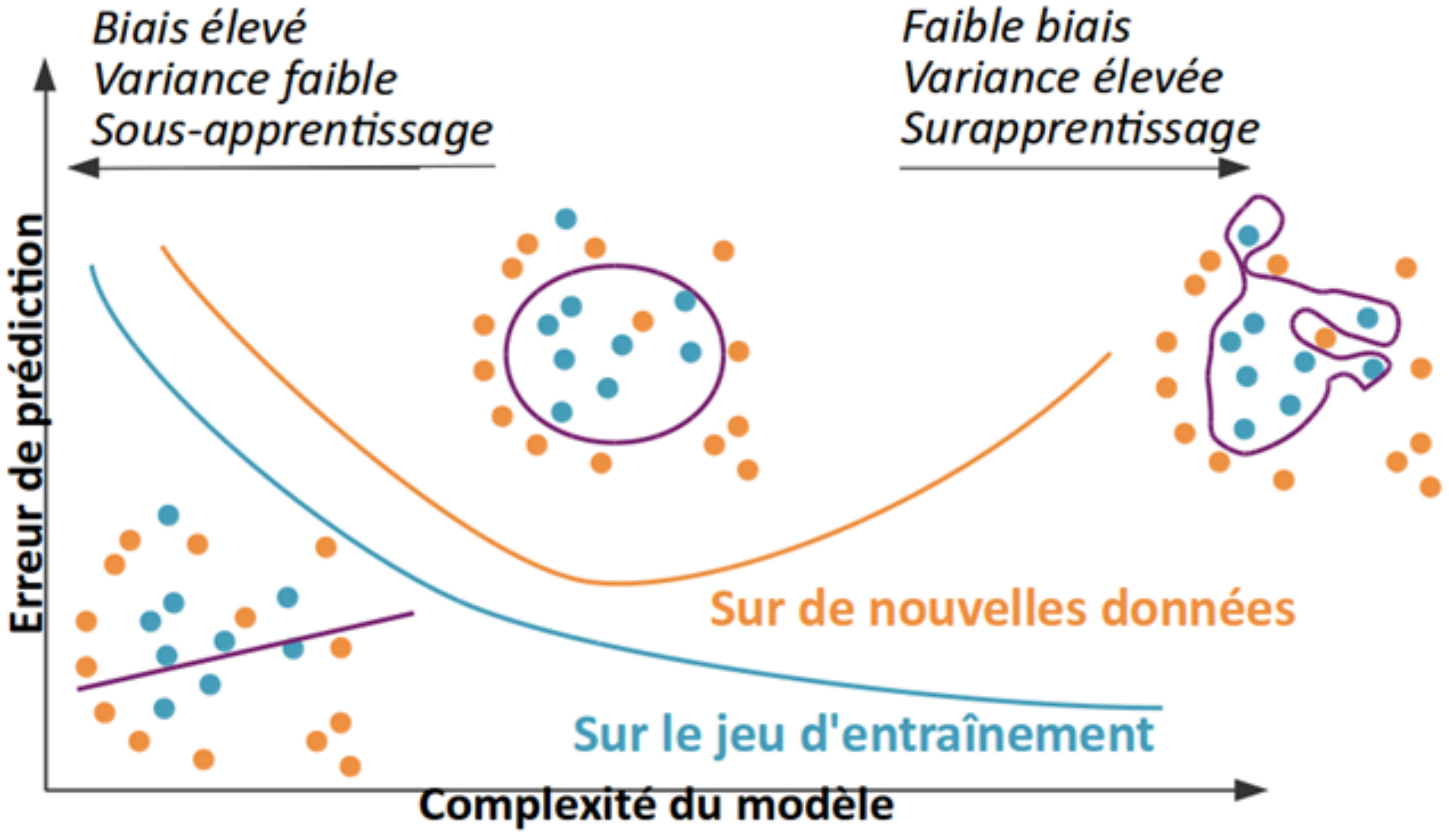
- **Un modèle simple** (variance faible) risque le ***sous-apprentissage*** (biais élevé y compris sur les données d'entraînement).

- **Un modèle complexe** (variance élevée) risque le ***sur-apprentissage*** (biais faible sur les données d'entraînement mais élevé sur de nouvelles données).



Compromis biais-variance

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***.



Exploiter bien ses données

- Nous voulons nous assurer que **notre modèle ne souffre pas d'Overfitting** ou **d'Underfitting** , et qu'il saura faire des prédictions sur de nouvelles données.

- Pour cela on devra savoir **très bien exploiter** les données du **dataset**.

Exploitation des données

Exploitation des données

- La ***dataset*** dont vous disposez constitue une ressource précieuse, il faut bien l'utiliser afin de pouvoir à la fois **choisir un modèle** et l'**entraîner,** mais aussi de pouvoir **tester** la qualité de ce modèle.

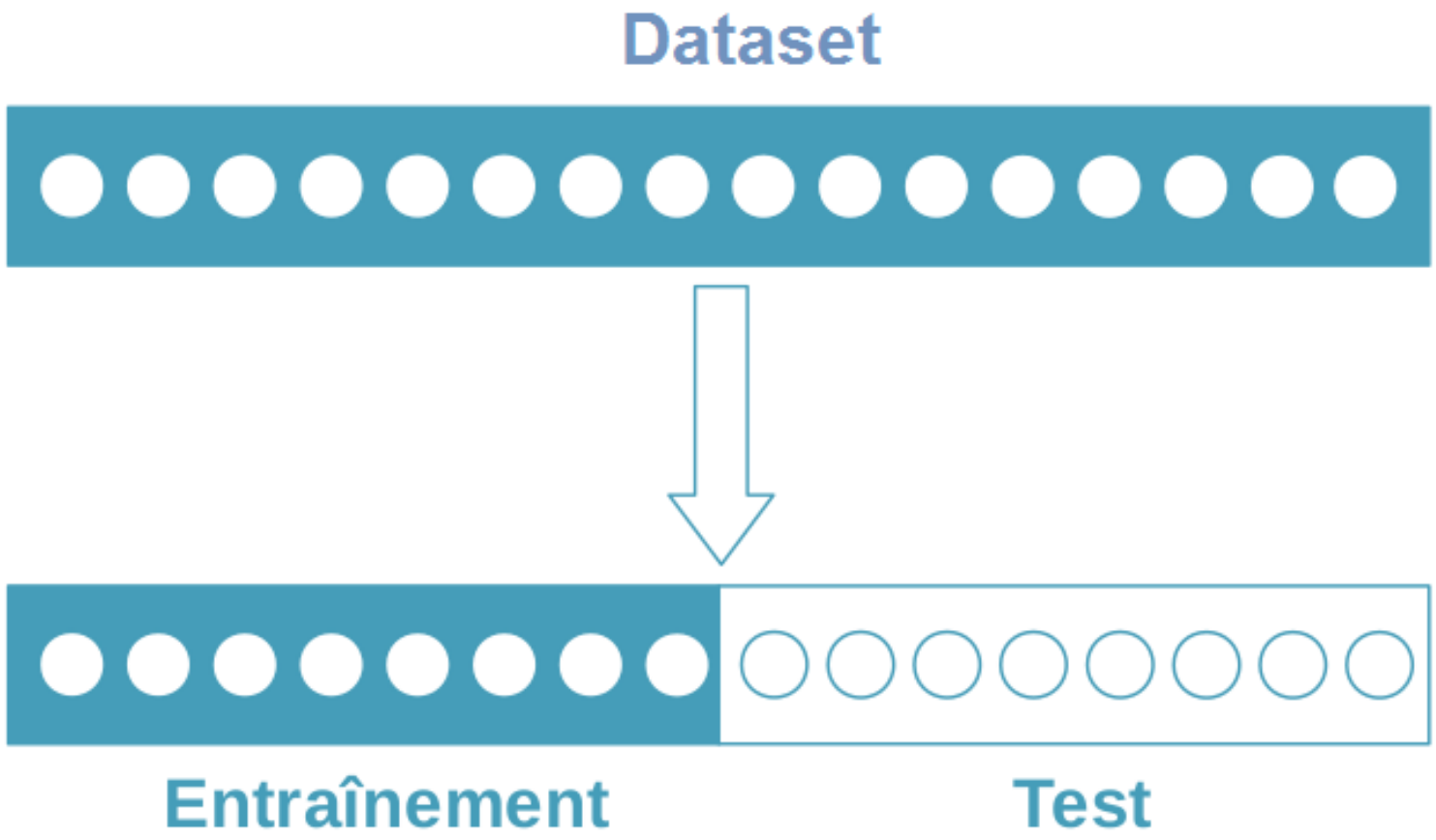
- L'entraînement d'un modèle revient à mesurer l'erreur de la sortie de l'algorithme avec les données d'exemples, et chercher à la minimiser.

- Un premier piège à éviter est donc d'évaluer la qualité du modèle final à l'aide des mêmes données qui ont servies pour l'entraînement. En effet, le modèle est complètement optimisé pour les données à l'aide desquelles il a été créé. L'erreur sera précisément minimum sur ces données. Alors que l'erreur sera toujours plus élevée sur des données que le modèle n'aura jamais vues !

Exploitation des données

- La meilleure approche est de séparer la ***dataset*** dès le départ:
  - **Training set**: qui va nous permettre d'entraîner notre modèle, et sera utilisé par l'algorithme d'apprentissage. C'est celui dont on a parlé depuis le début.
  - **Testing set**: qui permet de mesurer l'erreur du modèle final sur des données qu'il n'a jamais vues. On va simplement passer ces données comme s'il s'agissait de données que l'on n'a encore jamais rencontrées et mesurer la performance de notre modèle sur ces données. On appelle aussi cela ***held-out data***, vraiment pour souligner que ce sont des données auxquelles on ne va pas toucher avant la toute fin pour pouvoir être bien sûr que le modèle fonctionne.
- C'est à vous de définir la proportion de la dataset que vous souhaitez allouer à chaque partie.

Exploitation des données



Exploitation des données

- Si on entraîne le modèle avec des données, il va naturellement être plus performant sur ces données-là. Ce qui nous intéresse c'est de mesurer sa performance sur des données qu'il n'a jamais vues puisque c'est ce qui va se passer en pratique.

- ***Qu'est ce qu'on veut alors?***

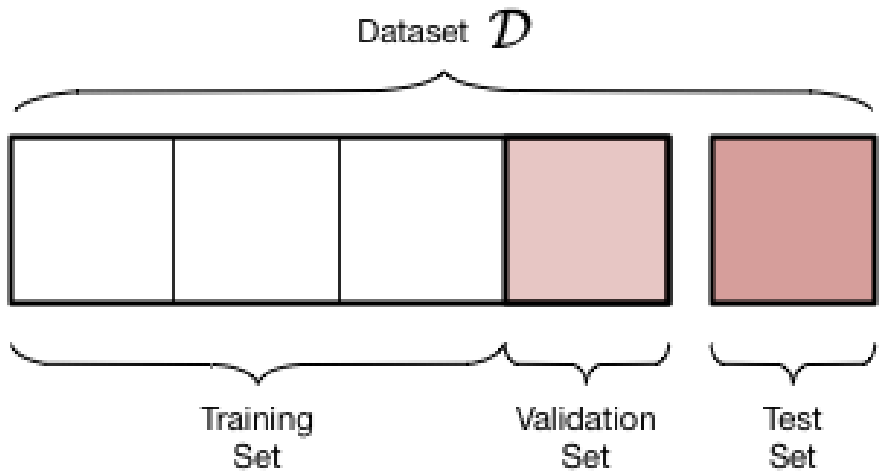
Exploitation des données

- Si on entraîne le modèle avec des données, il va naturellement être plus performant sur ces données-là. Ce qui nous intéresse c'est de mesurer sa performance sur des données qu'il n'a jamais vues puisque c'est ce qui va se passer en pratique.

- La **généralisation** du modèle : sa capacité à effectuer des prédictions de qualité sur des situations jamais rencontrées.

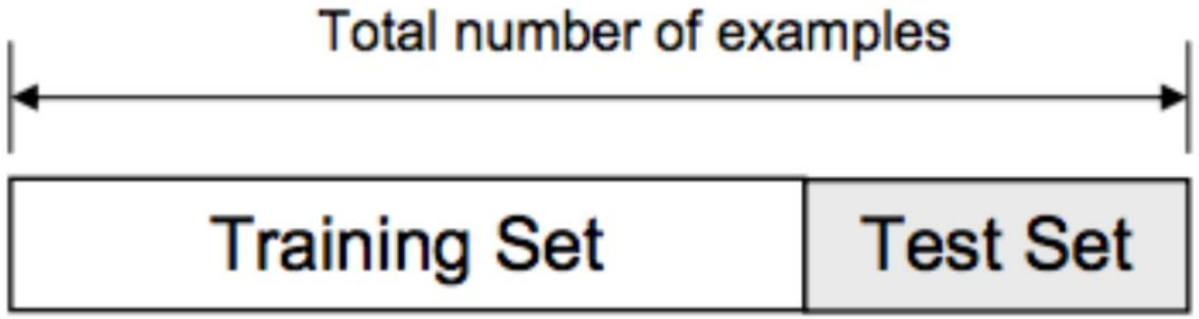
Exploitation des données

- Des fois on voudra tester la performance de notre modèle pour changer ses paramètres et le rendre plus performant. Dans ce cas on ne peut pas utiliser le test set. Parce que si on adapte notre modèle, en changeant ses paramètres, en utilisant le test set. Ces données test ne vont plus être jamais-vues. Et donc le modèle ne se sera pas généralisable. Alors dans ce cas on a besoin de :
  - **Validation set: un autre ensemble de données qui permet de mesurer l'erreur de prédiction pour choisir entre plusieurs modèles, il est aussi souvent utilisé.**



La validation: Percentage Split

- Dans le cas où on a une seule dataset, on va devoir le diviser en training set et test set. On pourrait faire cela de façon aléatoire en utilisant « Percentage split ».
- Par exemple, On pourrait diviser le dataset en **2/3** $\approx 66\%$ pour l'entrainement et **1/3** $\approx 34\%$ pour tester.
- L'erreur est estimée en calculant la performance du modèle sur le test set , par exemple l'erreur quadratique moyenne (RMSE).



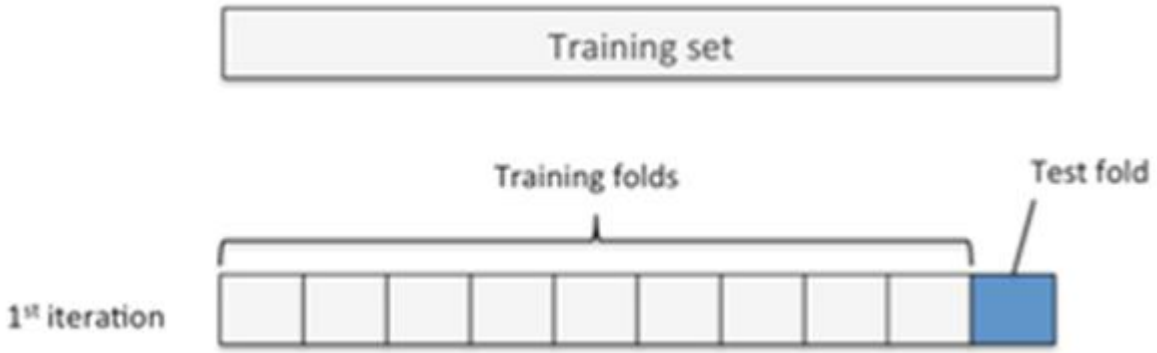
Problème?!

Nous n’avons utilisé qu'une partie du dataset 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** (erronée) !

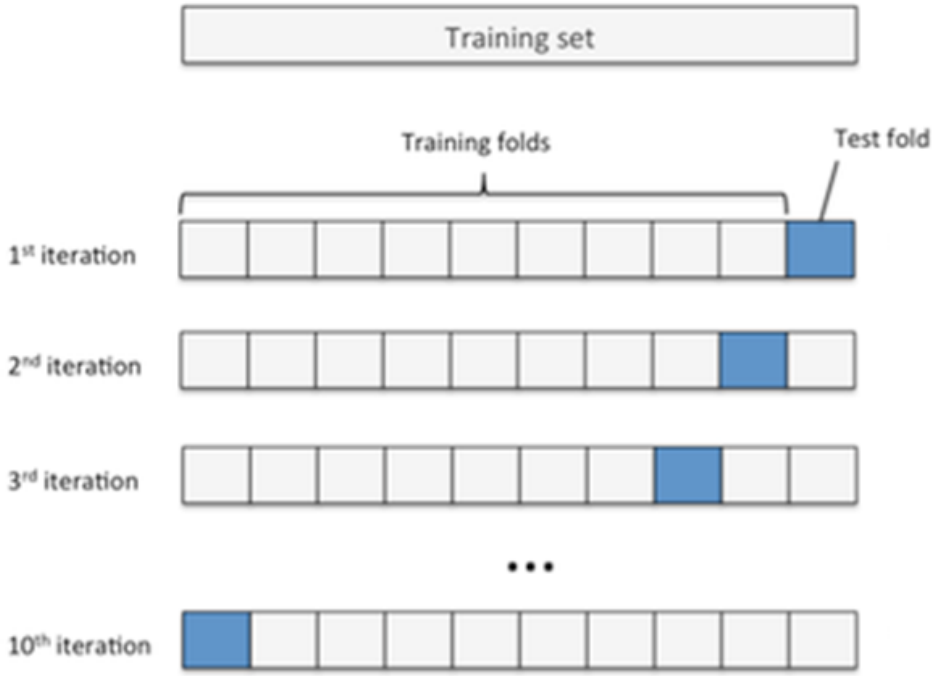
La validation: K-fold Cross-validation

- On divise le dataset original en ***k sets***, puis on sélectionne un des ***k sets*** comme **test set** et les (k-1) autres sets constitueront **le training set**.



La validation: K-fold Cross-validation

- On divise le dataset original en ***k sets***, puis on sélectionne un des ***k sets*** comme **test set** et les (k-1) autres sets constitueront **le training set**. On calcule alors la performance.
- Puis on répète l'opération en sélectionnant un autre **test set** parmi les (k-1) sets qui n'ont pas encore été utilisés comme test set.



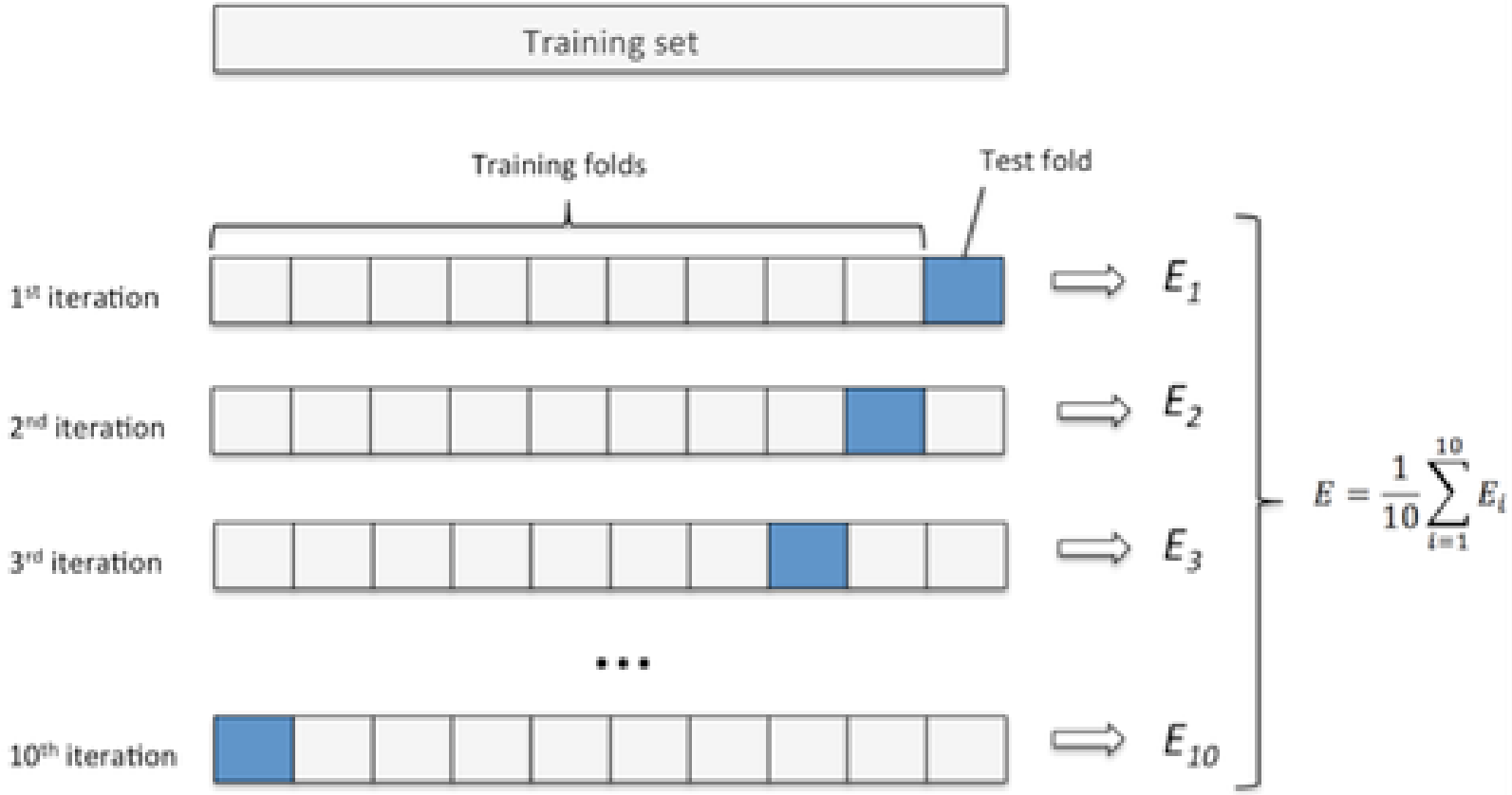
La validation: K-fold Cross-validation

- On divise le dataset original en ***k sets***, puis on sélectionne un des ***k sets*** comme **test set** et les (k-1) autres sets constitueront **le training set**. On calcule alors la performance. Puis on répète l'opération en sélectionnant un autre **test set** parmi les (k-1) sets qui n'ont pas encore été utilisés comme test set.

- L'opération se répète ainsi k fois (dans ce cas 10 fois) pour qu'en fin de compte **chaque set** ait été utilisé exactement **une fois comme test set**. Et on calcule par exemple, la moyenne des k erreurs quadratiques moyennes pour estimer l'erreur de prédiction.

Validation: K-fold Cross-validation

- Exemple de 10-fold cross-validation

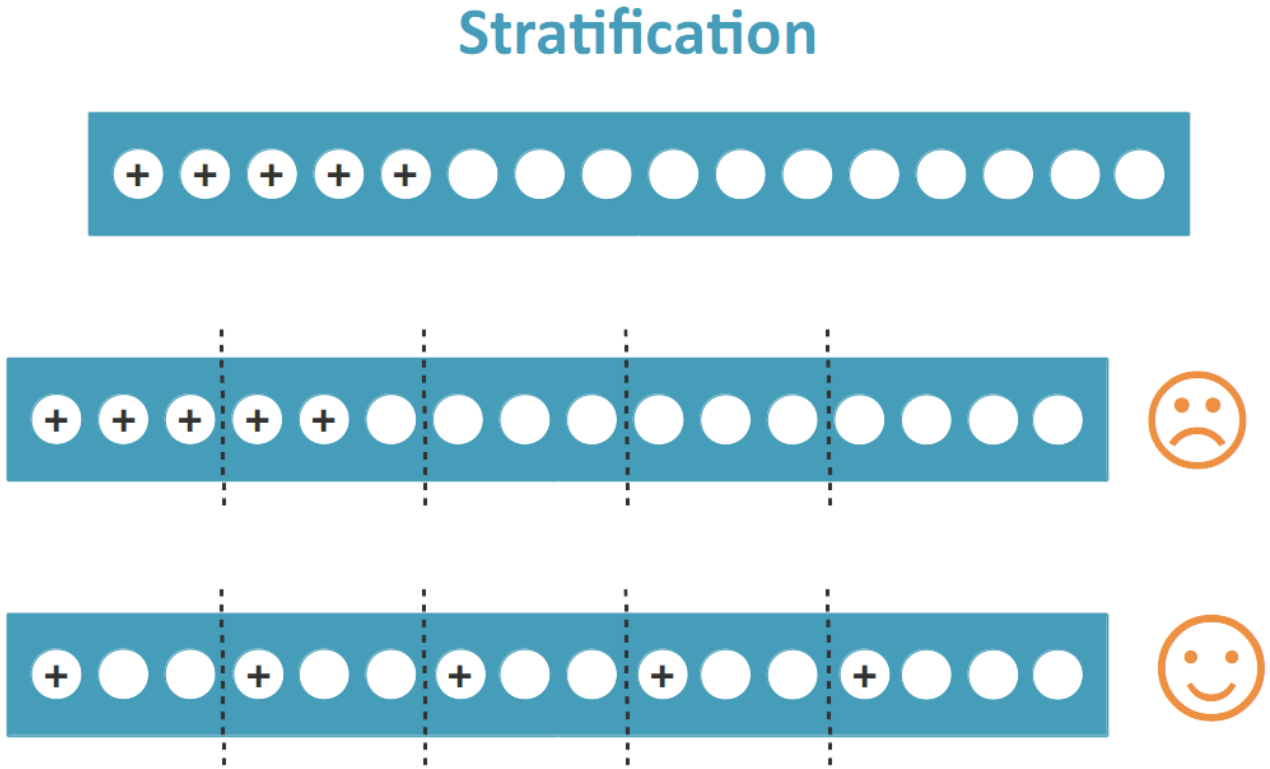


La validation: Stratified Cross-Validation

- La **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 dataset complet.

- On cherche à éviter de créer un ***Training set*** ne contienne que des exemples positifs et que ***Test set*** ne contienne que des exemples négatifs, ce qui va affecter négativement la performance du modèle !

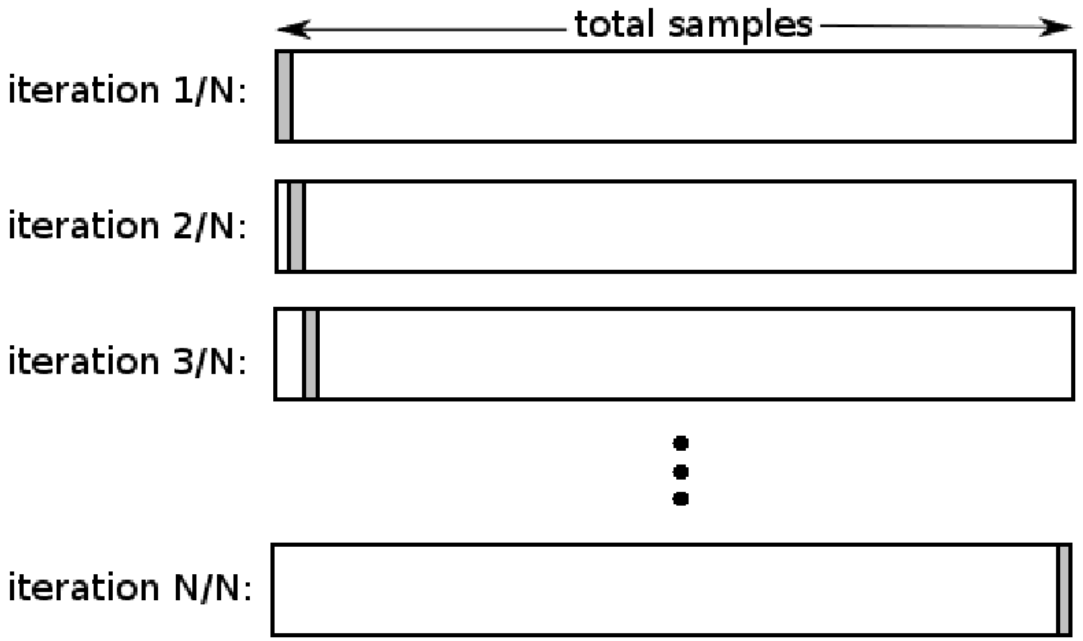
La validation: Stratified Cross-Validation



Si l'on ne répartit pas les points positifs de manière équilibrée entre les différents folds, le training set et de test set auront des proportions différentes de positifs et négatifs, ce qui peut **biaiser** les résultats.

Validation: Leave-one-out cross validation

- un cas particulier de « K-fold cross-validation » où K=N, c'est-à-dire que l'on apprend sur (N-1) observations puis on valide le modèle sur la énième observation et l'on répète cette opération N fois.



Résumé

- Il ne faut **jamais** évaluer un modèle sur des Data Points qui ont été utilisés pour l'entraîner.

- On sépare donc les données entre **Training set**, sur lequel on apprend le modèle, et **Test set**, 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**.

Les mesures de performance

Rappel

- Le **Machine Learning** est l'apprentissage d'un modèle statistique par la machine, grâce à des données d'entraînement.

- Un problème de Machine Learning comporte **4** éléments spécifiques :
  - **Des données**
  - **Une tâche à accomplir**
  - **Un algorithme d'apprentissage**
  - **Une mesure des performances**

un ordinateur apprend à partir de données pour résoudre une tâche en faisant attention à mesurer les performances. S'il améliore les performances sur cette tâche, lorsqu'on lui fourni les données d'entraînement, on dit alors qu'il apprend.

La classification

- On utilise des données étiquetées pour prédire à quelle ***classe*** un objet appartient.
- On parle de **classification binaire**, quand il s'agit de distinguer si un objet appartient ou non à une classe.

Exemple:

- 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.

Matrice de Confusion

- *Prédire un incendie quand il y en a un* **(TP)**
- *Prédire un incendie quand il n'y en a pas* **(FP)**
- *Ne pas prédire un incendie quand il y en a un* **(FN)**
- *Ne pas en prédire quand il n'y en a pas…* **(TN)**

- On peut s'y perdre vite ! Pour y voir plus clair, on utilise une **matrice de confusion** (le terme fait allusion à la confusion du modèle).

Matrice de Confusion

Appelons "**positive"** la classe correspondant à un incendie et "**négative**" l'autre.

		Classe réelle	
		-	+
Classe prédite	-	True Negatives *(vrais négatifs)*	False Negatives *(faux négatifs)*
	+	False Positives *(faux positifs)*	True Positives *(vrais positifs)*

Matrice de Confusion

- 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.** (TP)
- Si par contre cette prédiction est incorrecte, il s'agit d'un **faux positif.** (FP)
- La même chose pour le **vrai négatif** et le **faux négatif**. (TN et FN)

		Classe réelle	
		-	+
Classe prédite	-	True Negatives *(vrais négatifs)*	False Negatives *(faux négatifs)*
	+	False Positives *(faux positifs)*	True Positives *(vrais positifs)*

Confusion matrix

- La **matrice de confusion**, utilisée dans l'apprentissage supervisé, est un outil servant à mesurer la qualité d'un système de classification.

- Chaque ligne de la matrice représente le nombre d'occurrences d'une classe estimée, tandis que chaque colonne représente le nombre d'occurrences d'une classe réelle (on peut les inverser).

- La matrice de confusion montre rapidement si le système parvient à classifier correctement.

Confusion matrix (Exemple)

- On considère un système de **classification** dont le but est de classer du courrier électronique en deux classes : **courriels normaux ou pourriels** (Ham vs Spam).
- On veut savoir combien de courriels normaux seront faussement estimés comme des pourriels (fausses alarmes) et combien de pourriels ne seront pas estimés comme tels (non détections).
- On va supposer qu'on a testé notre classificateur avec 100 courriels normaux et 100 pourriels. Ainsi, la matrice suivante se lit comme suit :
  - **Sur les 100 courriels normaux, 95 seront estimés comme tels et 5 seront estimés comme pourriel ;**
  - **sur les 100 pourriels, 3 seront estimés comme courriels normaux, et 97 seront estimés comme pourriels ;**
  - **sur les 98 courriels que le système a estimé comme normaux, 3 sont en fait des pourriels ;**
  - **sur les 102 courriels que le système a estimé comme des pourriels, 5 sont en fait des courriels normaux.**

		Classe estimée	
		normal	**pourriel**
Classe réelle	**normal**	95	5
	pourriel	3	97

		Classe réelle	
		normal	**pourriel**
Classe prédite	**normal**	95	3
	pourriel	5	97

Les mesures par classe

- Pour une classe donnée, un classificateur, et un exemple, quatre cas peuvent se présenter :
  1. L'exemple est de cette classe, et le classificateur ne se trompe pas : c'est un vrai positif (TP).
  2. L'exemple est de cette classe, mais le classificateur se trompe : c'est un faux négatif (FN).
  3. L'exemple n'est pas de cette classe, mais le classificateur la lui attribue : c'est faux positif (FP).
  4. L'exemple n'est pas de cette classe, et le classificateur ne le range pas non plus dans cette classe : c'est un vrai négatif (TN).

	Classe prédite	
	prédite	non prédite
Exemples de cette classe	Vrais positifs	Faux négatifs
Exemple n'appartenant pas à cette classe	Faux positifs	Vrais négatifs

TP Rate

- Rapport des vrais positifs. Il correspond a :

$$\boldsymbol{TP\ Rate} = \frac{\mathbf{TP}}{\mathbf{TP+\ FN}}$$

- C'est donc le rapport entre le nombre d’éléments bien classés et le nombre total d‘éléments qui devraient être bien classés.

FP Rate

- Rapport des faux positifs. Il correspond, symétriquement à la définition précédente:

$$\boldsymbol{FP\ Rate} = \frac{\mathbf{FP}}{\mathbf{FP+\ TN}}$$

Exemple

- Les taux TP Rate et FP Rate permettent de reconstruire la matrice de confusion pour une classe donnée.
- Symétriquement, la matrice de confusion permet de calculer TP Rate et FP Rate.
- Calculez le TP Rate pour chaque classe?

	a	b	c	total
a	50	0	0	50
b	0	49	1	50
c	0	1	49	50
total	50	50	50	

Exemple

- Les cinquante exemples de classe a sont bien classés (pas de faux négatifs) : donc **TP Rate=1.**

- 49 exemples de la classe b sont bien classes, mais le dernier est mal classe : **TP Rate=49/50 = 0.98**

- 49 exemples de la classe c sont bien classés, sur un total de 50: **TP Rate= 49/50 =0.98**

	a	b	c	total
a	50	0	0	50
b	0	49	1	50
c	0	1	49	50
total	50	50	50	

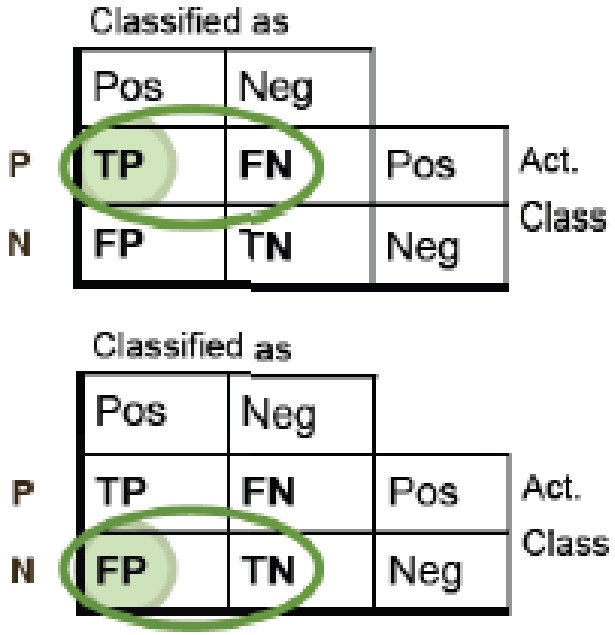
Accuracy

- Pour mesurer la qualité d'un modèle on se base sur la valeur « **Accuracy** »:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Tel que :

- TP: Les vrais positifs
- TN: les vrais négatifs
- FP: les faux positifs
- FN: les faux négatifs



Recall

- Le **Recall** (**Rappel** en français) ou **sensibilité**, est le **taux de vrais positifs**, c'est à dire la proportion de positifs que l'on a correctement identifiés.

$$\boldsymbol{Recall = \frac{TP}{TP + FN}}$$

- *Exemple:* C'est la capacité de notre modèle à détecter tous les incendies.
- Un Recall de 1 signifie que tous les exemples positifs ont été trouvés.

Recall (Problème?!)

- On peut facilement avoir un très bon Recall…
- 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.

Precision

- La **Precision** mesure la proportion de prédictions correctes parmi les points que l'on a prédits positifs.
- C'est le rapport entre le nombre de vrais positifs et la somme des vrais positifs et des faux positifs. Une valeur de 1 exprime le fait que tous les exemples classés positifs l'étaient vraiment. Elle est définit par:

$$\boldsymbol{Precision = \frac{TP}{TP + FP}}$$

Precision (Problème?!)

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

- Pour trouver **un compromis entre recall et precision**, on peut calculer la "**F-measure**".

F-Measure

- Moyenne harmonique de la « precision » et du « recall ». Elle Mesure la capacité du modèle à donner toutes les solutions pertinentes et à refuser les autres.
- La **F-mesure** correspond à un compromis de la précision et du rappel donnant la **performance du modèle**. Ce compromis est donné de manière simple par la moyenne harmonique de la précision et du rappel.

$$\mathbf{F}-\boldsymbol{mesure} = \frac{\mathbf{2} \times \boldsymbol{Recall} \times \boldsymbol{Precision}}{\boldsymbol{Recall}+\boldsymbol{Precision}}$$

$$\boldsymbol{F-mesure} = \frac{\mathbf{2}\,\boldsymbol{TP}}{\mathbf{2}\,\boldsymbol{TP}+\boldsymbol{FP}+\boldsymbol{FN}}$$

Specificity

- La mesure « **Specifcity** » ou spécificité donne 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 (Recall).

$$\boldsymbol{Specificity = \frac{TN}{TN + FP}}$$

Exercice

- 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.

Exercice

- Voici les résultats

de l'expérience:

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

1. Calculez toutes les mesures qu'on vient de voir (Accuracy, Recall, Precision, F-mesure, Specificity) ?
2. Quelle est la probabilité de ne pas avoir de cancer quand le frottis est négatif ? Alors qu'est ce que vous pouvez dire sur l'outil?
3. Quelle est la probabilité d'avoir un cancer quand le frottis est positif? Alors qu'est ce que vous pouvez dire sur l'outil?

Formules

- $\boldsymbol{Accuracy = \frac{TP+TN}{TP+TN+FP+FN}}$
- $\boldsymbol{Recall = \frac{TP}{TP+FN}}$
- $\boldsymbol{Precision = \frac{TP}{TP+FP}}$
- $\boldsymbol{\mathrm{F}-mesure = \frac{2 \times Recall \times Precision}{Recall+Precision}}$ ou $\boldsymbol{F - mesure = \frac{2\,TP}{2\,TP+FP+FN}}$
- $\boldsymbol{Specificity = \frac{TN}{TN+FP}}$

Exercice

- 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**.
- On a les valeurs suivantes :
  - accuracy ≈ 95%
  - recall = 95%
  - specificity= 94.5 %
  - precision = 47.5%
  - F-measure ≈ 64 %

La classification avec un Score

- Bien que le but d'un algorithme de classification soit de faire des prédictions binaires, un grand nombre d'algorithmes retournent **un nombre réel**.

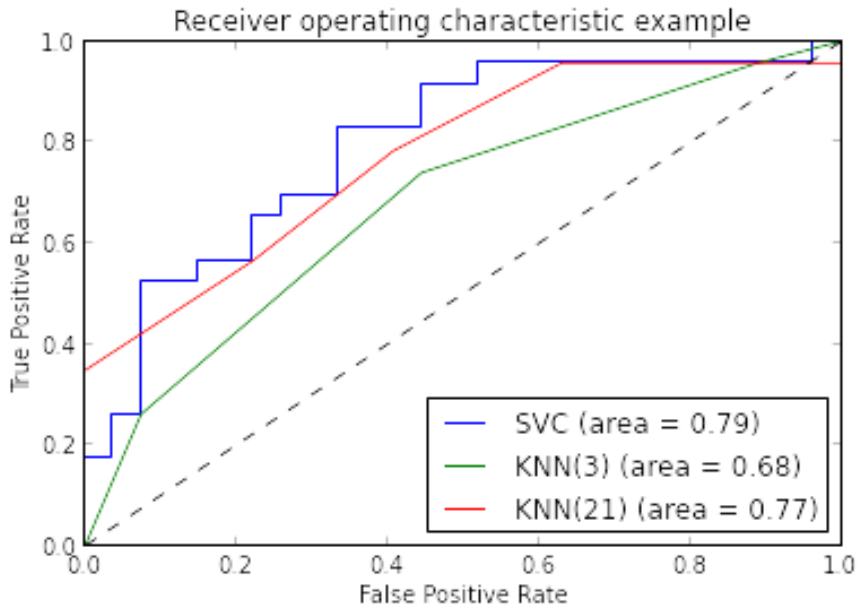
- ***Plus cette valeur est élevée, plus le point est susceptible d'être positif***.

- 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.

- 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***.

La courbe ROC

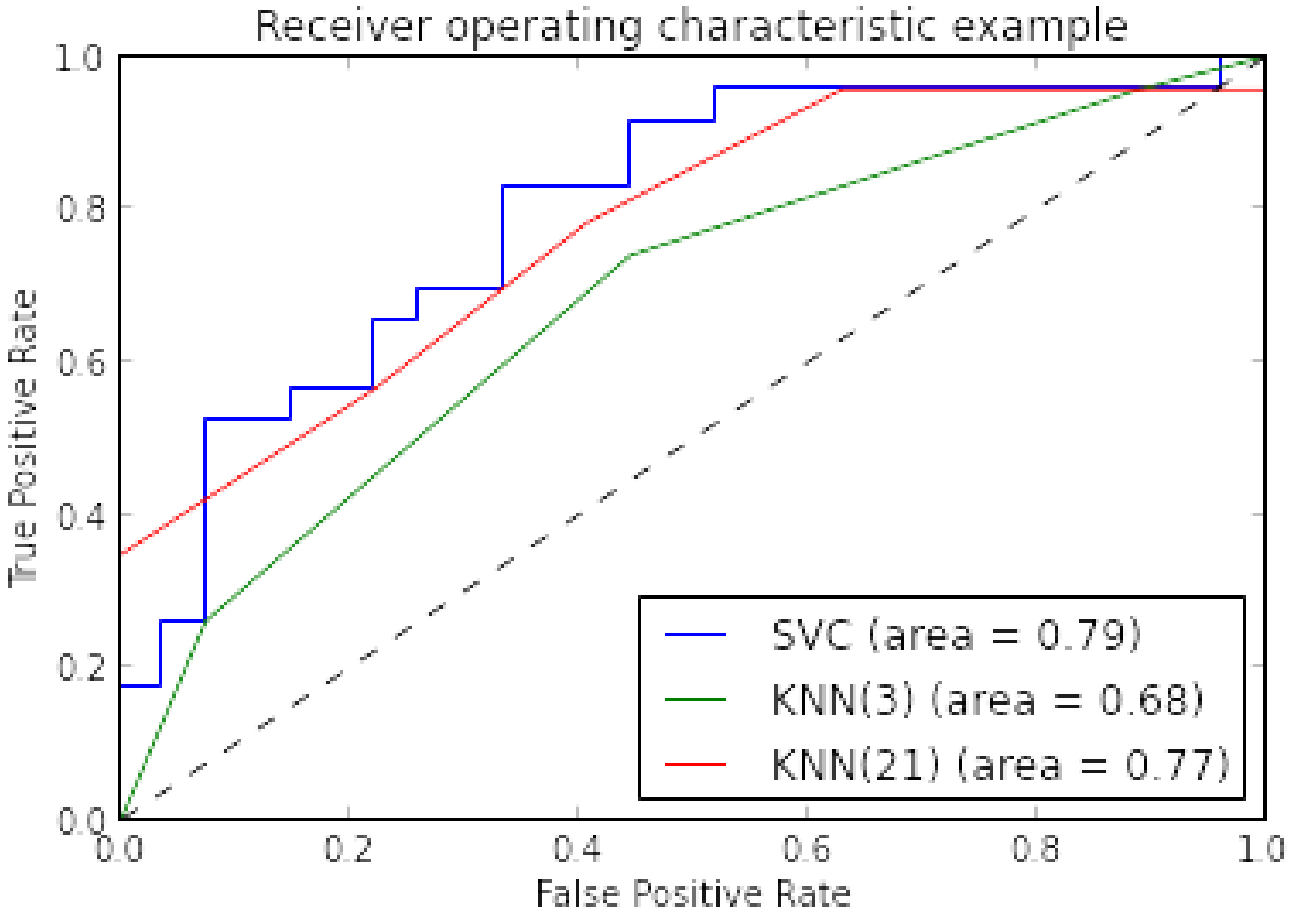
- On va faire l'évaluation du TP rate par rapport au FP rate d'un modèle de prédiction binaire, ***pour chaque seuil de décision possible***.
- On construit une courbe pour montrer comment le TP Rate évolue en fonction du FP Rate (ou bien aussi le Recall en fonction de (1 moins la spécificity))
- Cette courbe s'appelle la **courbe ROC**, pour « *Receiver-Operator Characteristic* ».



Area Under the Curve (AUC)

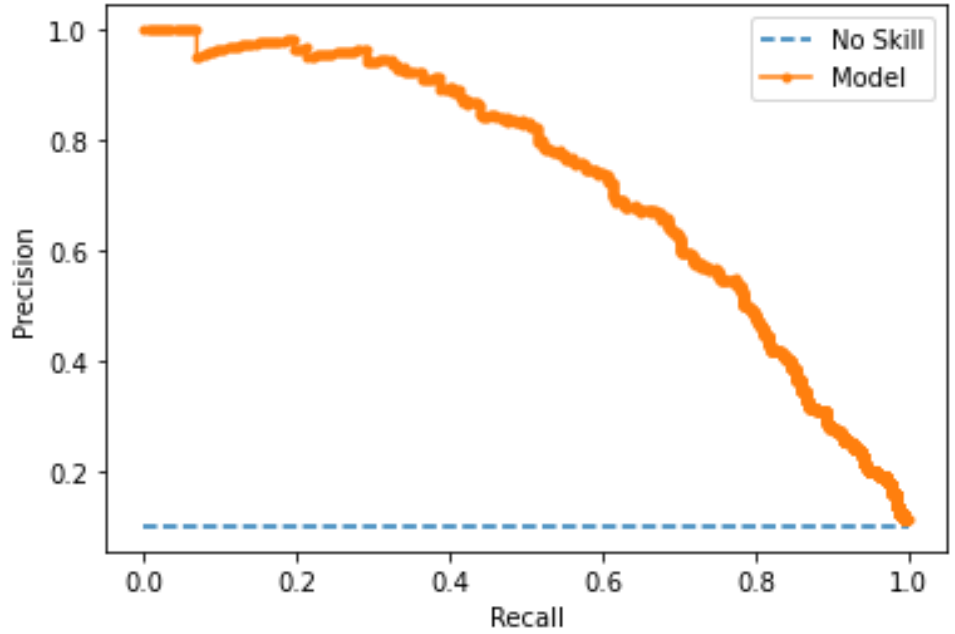
Area Under the Curve (AUC)

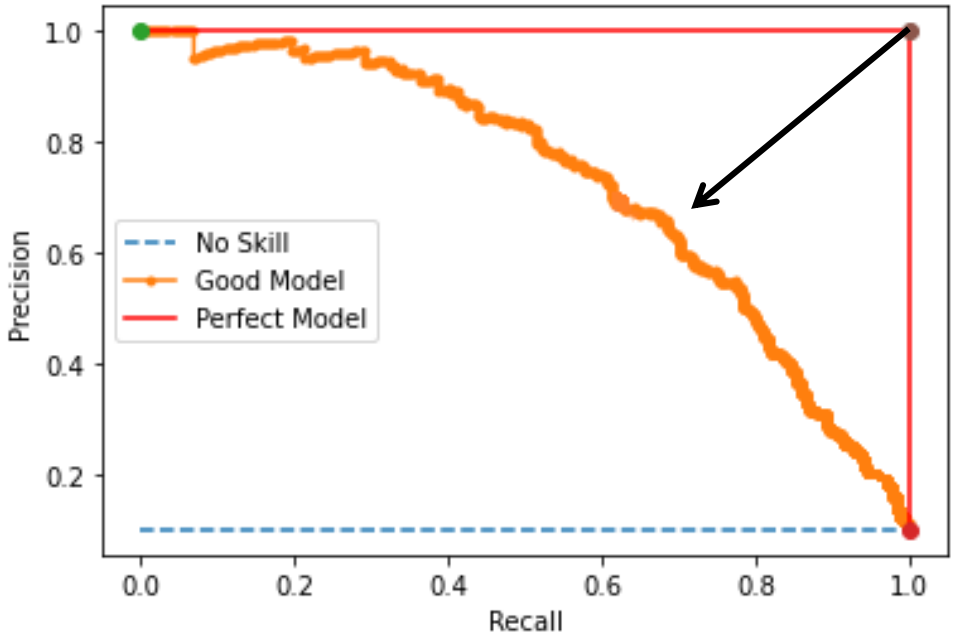
- ROC AUC est égal à la probabilité qu'un classificateur classe un cas positif choisi au hasard plus haut qu'un cas négatif choisi au hasard.
- ROC AUC de 0,5 est un classificateur inutile, **1 est parfait**



La courbe Recall-Precision

- Le diagramme Recall-Precision est un graphe d'évaluation du modèle qui repose sur deux mesures d'évaluation de base : recall et precision.
  - Recall est une mesure de performance de toutes les valeurs positives d'un ensemble de données
  - La mesure precision est une mesure de performance des prédictions positives seulement.
- C'est une mesure de la confiance qu'a le classifieur dans ses propres prédictions par rapport à la réalité.





La régression

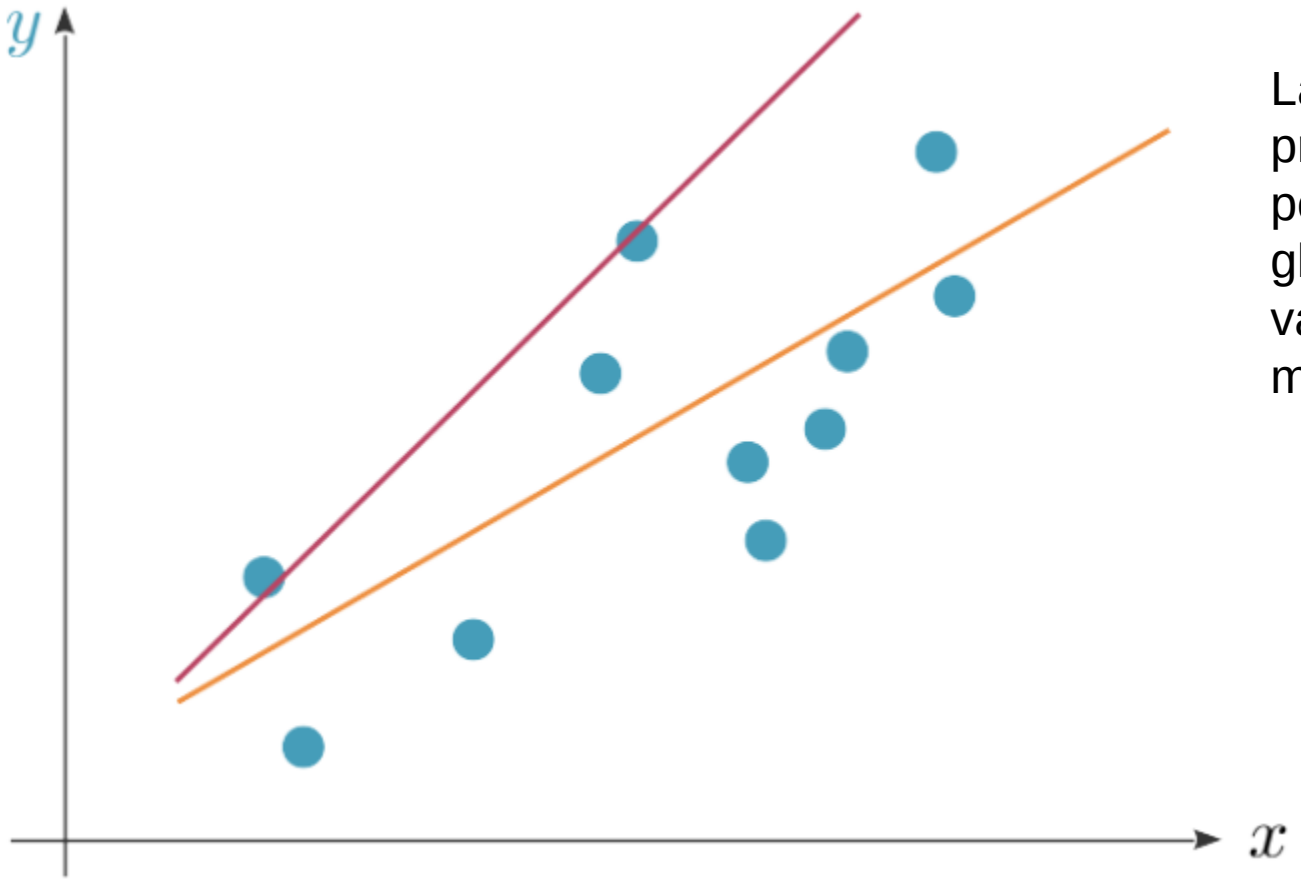
- Quand nous avons parlé d'évaluer des modèles de classification, nous avons commencé par compter le nombre d'erreurs de prédiction que fait le modèle.

- **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 régression



La ligne rouge 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.

Residual Sum of Squares

- Quantifions cette notion de « **plus près des vraies valeurs** ».
- Nous allons calculer pour chaque point x_i 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*.

$$\boldsymbol{RSS = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2}$$

Mean Squared Error

- 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 Test set. On obtient ainsi l'**erreur quadratique moyenne**, ou **MSE**, pour *Mean Squared Error*.

$$\boldsymbol{MSE = \frac{1}{n}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

Root 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.*

$$\boldsymbol{RMSE} = \sqrt{\frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}(\boldsymbol{\hat{y}_i} - \boldsymbol{y_i})^{\mathbf{2}}}$$