Chapitre 5

Evaluation du modèle

Plan du chapitre

1. Introduction
2. Métriques pour la classification
3. Métriques pour la régression
4. Conseils pour un bon modèle

01
Introduction

Introduction

- Une fois que les modèles de données ont subi une formation et que les prédictions ont été générées, l'étape suivante du cycle de vie de la science des données consiste à évaluer les résultats

- Les Data Scientists évaluent les performances de leurs modèles et valident l'exactitude des prédictions par rapport à la vérité terrain ou aux résultats connus

- Ce processus d'évaluation joue un rôle crucial pour déterminer l'efficacité des modèles et obtenir des informations précieuses sur les données analysées

- Au cours de la phase d'évaluation, les Data Scientists utilisent diverses techniques pour analyser et interpréter les résultats

Introduction

- Les résultats de la prédiction peuvent être évalués à l'aide de diverses métriques pour mesurer la performance du modèle et identifier d'éventuelles opportunités d'amélioration

- Ces métriques peuvent inclure l'exactitude, la précision, le rappel, le score F1 ou d'autres mesures spécifiques au domaine en fonction de la nature du problème

- Si les mesures de performances ne sont pas satisfaisantes, les Data Scientists revisitent les étapes antérieures du cycle de vie de la science des données pour affiner la préparation des données, la sélection des fonctionnalités ou les techniques de modélisation

Introduction

- Les performances prédictives du modèle sont mises à l'épreuve avec l'échantillon test

- Le modèle prédit les résultats y sur la base des données X de l'échantillon test

- Si les prédictions et les données y réelles (= observées) sont proches, alors l'algorithme a une bonne performance

- Il existe des indices qui mesurent cette performance:
  - Indices de performance pour les problématiques de classification
  - Indices de performance pour les problématiques de régression

02

Métriques pour la classification

Classification

- Dans un problème de classification, on utilise des données étiquetées pour prédire à quelle classe un objet appartient
- On s'intéresse surtout à la classification binaire, où il s'agit de distinguer si un objet appartient ou non à une classe
- Dans les problématiques de classification, la plupart des indices de performance sont calculés à partir d'une matrice de confusion
- Cette matrice affiche le nombre de succès et d'échecs de prédiction pour chaque catégorie de la variable y
- Pour une variable y binaire (survie/mort, fraude oui/non, …), la matrice de confusion comporte 4 cellules.

Classification

Matrice de confusion

		Classe prédite	
		+	-
Classe réelle	+	TP : True Positives (Vrai Positifs)	FN : False Negatives (Faux Négatifs)
	-	FP : False Positives (Faux Positifs)	TN : True Negatives (Vrai Négatifs)

- **TP :** le nombre d'instances correctement prédites de classe positive
- **FP : le** nombre d'instances de la classe négative qui n'ont pas été prédites correctement
- **TN :** le nombre d'instances correctement prédites de classe négative
- **FN :** le nombre d'instances de la classe positive qui n'ont pas été prédites correctement

➔ À partir de la matrice de confusion on peut dériver plusieurs critères de performance

Classification

Exactitude (Accuracy)

L'exactitude mesure la proportion totale d'observations correctement classées par le modèle par rapport au nombre total d'observations

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

C'est une métrique simple mais peut être trompeuse dans des situations où les classes sont déséquilibrées

➔ Par exemple, dans un cas où 95 % des échantillons appartiennent à la classe A et seulement 5 % à la classe B, un modèle qui prédit constamment la classe A pourrait encore atteindre une exactitude de 95 %, mais ne serait pas utile pour détecter la classe B

Classification

Exactitude (Accuracy)

```
# Import des bibliothèques
from sklearn.metrics import accuracy_score

# Calcul de l'exactitude
accuracy = accuracy_score(y_test, y_pred)
print(f'Exactitude : {accuracy:.2f}')
```


Classification

Rappel ou Sensibilité (Recall ou Sensitivity)

Donne le **taux d'instances positives correctement prédites** parmi toutes les instances réellement positives.
Mesure la capacité du modèle à détecter l'ensemble des individus positifs.

$$rappel = \frac{TP}{TP + FN}$$

On peut facilement avoir un très bon rappel en prédisant systématiquement "positif". Néanmoins, le modèle ne sert pas à grand chose

Classification

Rappel ou Sensibilité (Recall ou Sensitivity)

```
# Import des bibliothèques
from sklearn.metrics import recall_score

# Calcul du rappel
recall = recall_score(y_test, y_pred)
print(f'Rappel (Sensibilité) : {recall:.2f}')
```


Classification

Précision (Precision)

La proportion de prédictions correctes parmi les points que l'on a prédits positifs

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

Cette métrique est inappropriée pour des cas où les faux positifs ont des conséquences graves

Classification

Précision (Precision)

```
# Import des bibliothèques
from sklearn.metrics import precision_score

# Calcul de la précision
precision = precision_score(y_test, y_pred)
print(f'Précision : {precision:.2f}')
```


Classification

Spécificité (Specificity/selectivity)

Donne le **taux d'instances négatives correctement prédites.**
Mesure la capacité d'un modèle à identifier correctement les exemples négatifs et à éviter les faux positifs.

$$spécificité = \frac{TN}{TN + FP}$$

Métrique importante dans des situations où les conséquences des faux positifs sont graves

Classification

Spécificité (Specificity)

```
# Import des bibliothèques
from sklearn.metrics import accuracy_score, recall_score,
precision_score, confusion_matrix

# Calcul de la spécificité
conf_matrix = confusion_matrix(y_test, y_pred)
specificity = conf_matrix[0, 0] / (conf_matrix[0, 0] +
conf_matrix[0, 1])
print(f'Spécificité : {specificity:.2f}')
```


Classification

Mesure F1 (F1-Score)

Il s'agit d'un compromis entre rappel et précision (leur moyenne harmonique)

$$\boldsymbol{Mesure\ F1 = 2\frac{Précision \times Rappel}{Précision + Rappel} = \frac{2TP}{2TP + FP + FN}}$$

Utile lorsque l'équilibre entre la précision et le rappel est important

Classification

Mesure F1 (F1–Score)

```
# Import des bibliothèques
from sklearn.metrics import f1_score

# Calcul de la mesure F1
f1 = f1_score(y_test, y_pred)
print(f'F1 Score: {f1:.2f}')
```


Classification

- La bibliothèque scikit-learn possède une fonction très utile `classification_report`
- Elle sert à évaluer les performances d'un modèle de classification
- Cette fonction génère un rapport détaillé qui comprend plusieurs métriques pour évaluer la qualité de la classification , comme l'exactitude , la précision , le rappel , le F1-Score, …

```
# Import des bibliothèques
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

03

Métriques pour la régression

Régression

- En régression, les valeurs prédites sont des valeurs numériques plutôt que des classes
- Plusieurs métriques d'évaluation sont couramment utilisées pour mesurer la performance du modèle pour les problèmes de régression
- L'Erreur Quadratique Moyenne est la métrique la plus utilisée parmi toutes les autres métriques

Régression

L'Erreur Quadratique Moyenne (MSE), appelée aussi Perte Quadratique, mesure la racine carrée des différences quadratiques moyennes entre les Y observés et prédits. Cet indice est à minimiser.

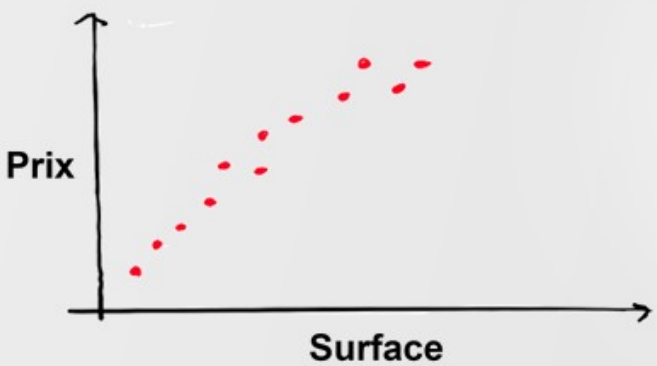
MSE = $1/N \sum_{(x,y)\in D}(y - prediction(x))2$ avec

- (x, y) un exemple (caractéristiques, étiquette)
- prediction(x) est le résultat du modèle pour l'entrée x
- D la base contenant l'ensemble des exemples
- N le nombre d'exemples.

Régression

L'Erreur Quadratique Moyenne

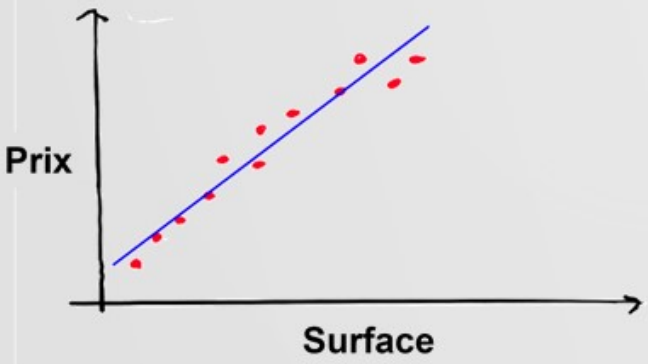
Exemple : Le prix d'une maison en fonction de la surface



- Les données : on a quelques exemples de prix de vente de maisons en fonction de la surface
- Le but : trouver une règle (un modèle) qui permet de lier le prix de vente et la surface d'une maison

Régression

L'Erreur Quadratique Moyenne

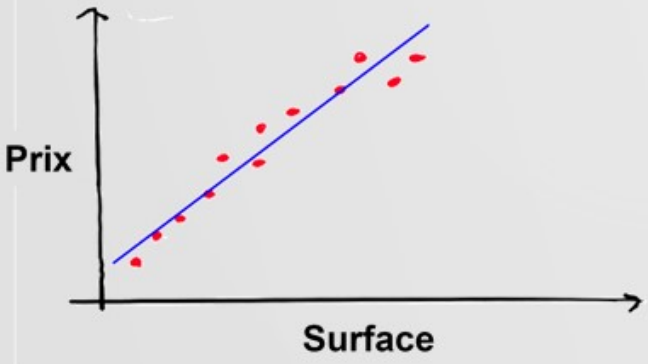


$y = mx + b$ avec

- y le prix de la maison, ce qu'on cherche à prédire
- x sa surface, la valeur de la caractéristique d'entrée
- m la pente de la droite
- b l'ordonnée à l'origine

Régression

L'Erreur Quadratique Moyenne



Par convention, on note

$y = w_1x_1 + b$ avec

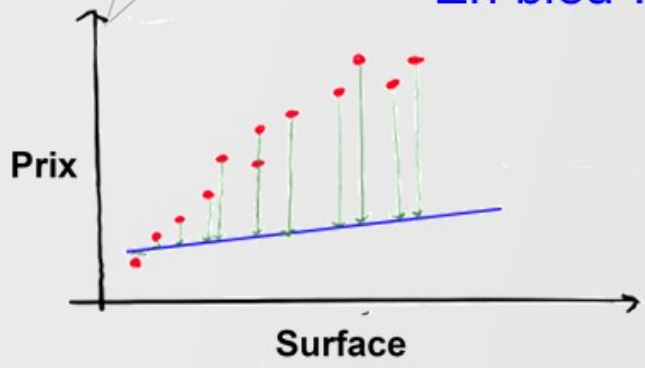
- y le prix de la maison, ce qu'on cherche à prédire
- x un exemple
- w_1 le poids de la première caractéristique
- b le biais, aussi nommé w_0

Régression

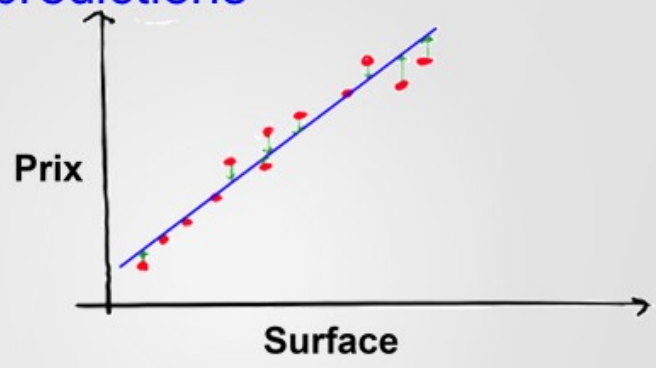
L'Erreur Quadratique Moyenne

En rouge : la perte

En bleu : les prédictions



⇒ Grande perte, mauvais modèle.



⇒ Faible perte, bon modèle.

Régression

L'Erreur Quadratique Moyenne

```
# Import des bibliothèques
from sklearn.metrics import mean_squared_error

# Calcul de L'Erreur Quadratique Moyenne
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error (MSE): {mse:.2f}')
```

04

Conseils pour un bon modèle

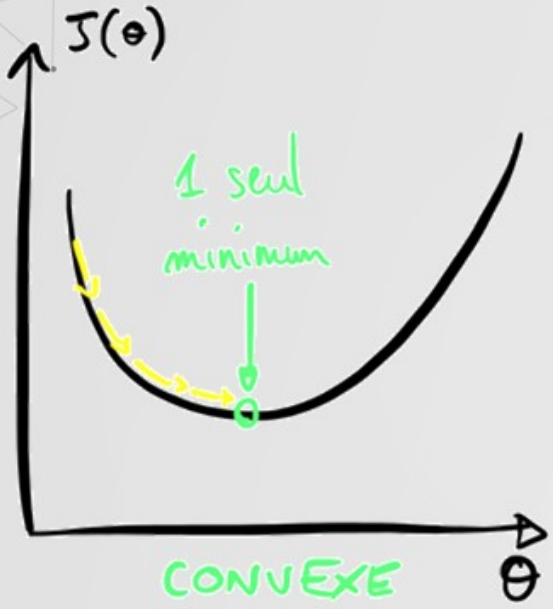
Minimiser la perte

- Dans l'apprentissage supervisé, la machine cherche les paramètres de modèle qui minimisent la fonction coût. Pour trouver les paramètres qui minimisent la fonction coût, il existe un paquet de stratégies.

- On pourrait par exemple développer un algorithme qui tente au hasard plusieurs combinaisons de paramètres, et qui retient la combinaison avec la fonction coût la plus faible.

- Une autre stratégie est de considérer la fonction coût comme une fonction convexe, c'est-à-dire une fonction qui n'a qu'un seul minimum, et de chercher ce minimum avec un algorithme de minimisation appelé **Descente de Gradient**. Cette stratégie apprend de façon graduelle, et assure de converger vers le minimum de la fonction coût .

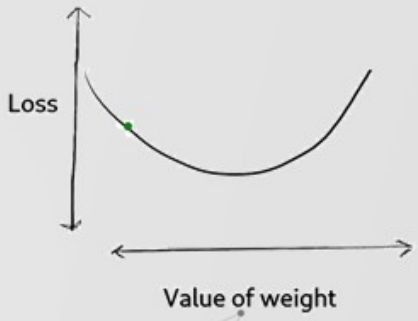
Minimiser la perte



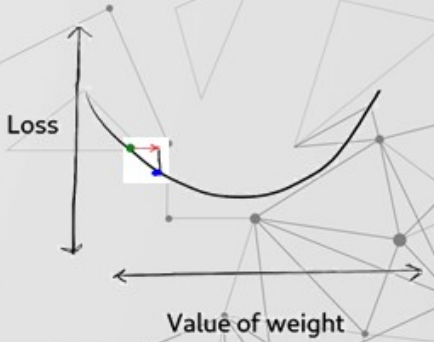
Minimiser la perte

La descente de gradient

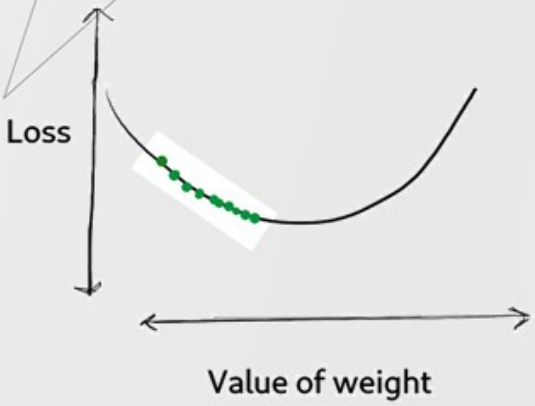
- Calcul du gradient à partir du point initial, on a :
- une direction. une magnitude.



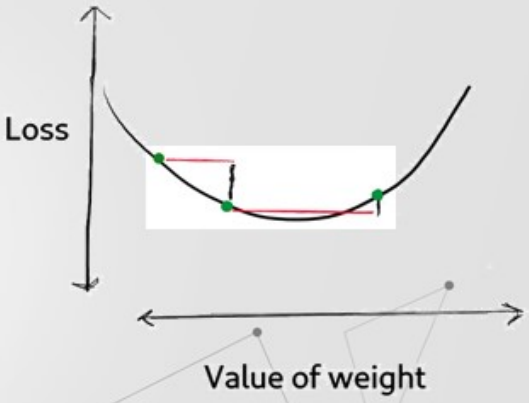
- En vert : point initial,
- En rouge : gradient négatif,
- En bleu : nouveau point,



Minimiser la perte



Le pas est trop petit



Le pas est trop grand

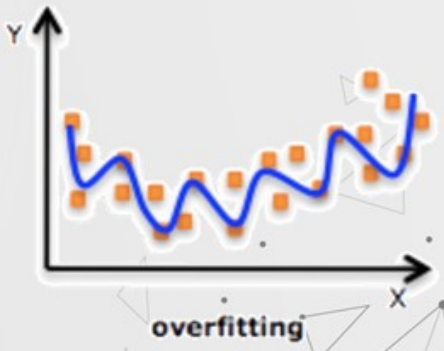
Sur-apprentissage (Overfitting)

L'Overfitting désigne le fait que le modèle prédictif produit par l'algorithme de Machine Learning s'adapte **trop bien** au Training Set.

- Faible erreur sur les données d'entrainement
- Mauvais en prédiction (nouvelles données)
- Le modèle est trop complexe

⇒ C'est un modèle trop spécialisé sur les données du Training Set (**grande variance**) et qui se généralisera mal.

variance = bruit dans les prédictions



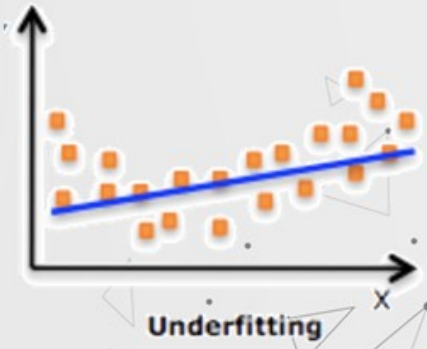
Sous-apprentissage (Underfitting)

L'Underfitting désigne le fait que le modèle prédictif généré lors de la phase d'apprentissage, s'adapte **mal** au Training Set.

- Ne capturer pas les corrélations du Training Set
- Grand erreur sur les données d'entrainement
- Mauvais en prédiction (nouvelles données)

⇒ C'est un modèle généraliste incapable de fournir des prédictions précises. On dit qu'il souffre d'un grand **biais**.

biais = manque de capacité à prédire correctement



Le meilleur modèle

On cherche un modèle qui est au « juste milieu ». Il ne doit souffrir ni d'Underfitting ni d'Overfitting.

Le meilleur modèle

On cherche un modèle qui est au « juste milieu ». Il ne doit souffrir ni d'Underfitting ni d'Overfitting.

En d'autres termes, il ne souffre ni d'un grand **bias** ni d'une grande **variance**.

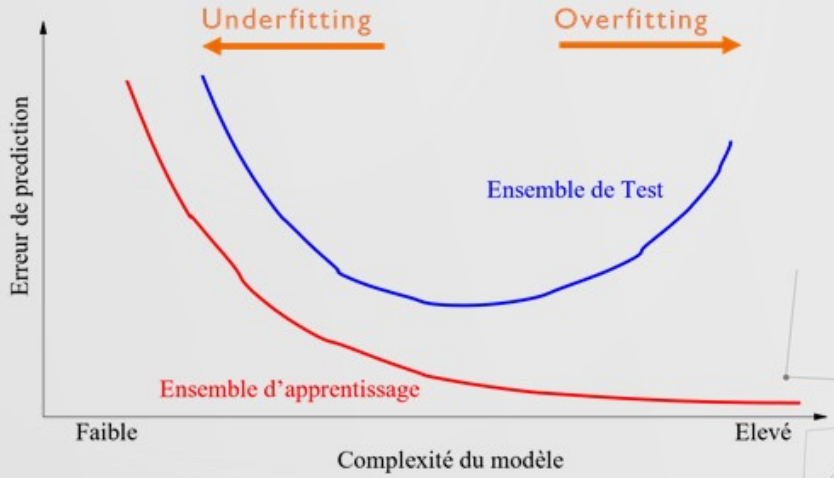
Le meilleur modèle

- Quand l'algorithme de Machine Learning apprend depuis le Training Set, les paramètres du modèle prédictif vont s'affiner au fil du temps. En conséquence, le coût d'erreur sur le Training Set continuera à se réduire. Parallèlement, le coût d'erreur des prédictions sur les données de test (non vu par l'algorithme) se réduira également.

- Mais si on continue à entrainer longuement le modèle, le coût d'erreur sur le Training Set continuera à se réduire. Mais celui du Test Set pourra commencer à augmenter. Cette augmentation signifie que le modèle prédictif s'est trop spécialisé sur les données d'apprentissage. Le point où commence l'augmentation du coût d'erreur dans le Test Set est le commencement du Overfitting.

Le meilleur modèle

Le juste milieu est le point juste avant le commencement de l'augmentation du coût d'erreur dans le Test Set. C'est à ce stade que le modèle sera abouti et bien généralisable.



Le meilleur modèle

Comment trouver le bon compromis ?

1. Réduire les dimensions

→ Quand on se trouve dans le cas d'une variance excessive (trop forte dépendance au data set d'entraînement), réduire le nombre de dimensions du modèle (c.-à-d. le nombre de variables en entrée) permet de diminuer la variance en simplifiant la complexité du modèle

Le meilleur modèle

Comment trouver le bon compromis ?

2. Sélectionner et entraîner le bon modèle

Il existe plusieurs méthodes de sélection de modèle qui permettent de trouver la complexité optimale pour faire la balance entre le biais et la variance. La manière d'entraîner le modèle est primordiale aussi et il existe des méthodes pour minimiser la prise en compte de variances non-représentatives du modèle présentes dans le dataset.

Le meilleur modèle

Comment trouver le bon compromis ?

3. Utiliser des méthodes ensemblistes

Il faut aussi savoir qu'il existe toute une famille d'algorithmes appelés les méthodes d'ensembles qui se basent sur la combinaison de plusieurs modèles à haute variance et les agrègent (par exemple en les moyennant) pour réduire la variance finale.

Ajouter Matrice ROC