# Machine Learning
# & Intelligence Artificielle




Manuel Simoes
manuel.simoes@cpc-analytics.fr

# Apprentissage artificiel

Précédemment nous avons vu :

- Dans la méthode de régression linéaire
  - Le modèle mathématique
  - Le processus itératif
  - La fonction coût
  - Les hypothèse et faiblesse de l'algorithme

On va maintenant intégrer cet algorithme dans une démarque Data Science

# Learning algorithm

**PHASE D'ANALYSE DU PROBLÈME**

**Choix des variables**

**X** -> Surface
**Y** -> Prix

**Choix de l'algorithme**

***La Régression***
**+** Hypothèse d'un modèle linéaire (avec taux d'apprentissage).

**Données d'apprentissage**
Nous avons un jeu de données dédié : Training Set (environ de 60 % à 80 % du total de vos données).
$\mathbf{X}_{\text{Train}}$ Superficie donnée (training)

**PHASE D'APPRENTISSAGE**

Régression
$h_\theta(x) = \theta_0 + \theta_1 x$

$\theta_0$ $\theta_1$
Minimisation de la Fonction coût

$\mathbf{Y}_{\text{Train}}$ Prix réel (training)

**Données de test**
Nous avons un AUTRE jeu de données dédié : Testing Set (environ de 20 % à 40 % du total de vos données).
$\mathbf{X}_{\text{Test}}$ Superficie données (testing)

$\mathbf{Y}_{\text{Test}}$ Prix réel (testing)

**PHASE D'ANALYSE de la qualité du modèle**

Calculer / analyser la qualité du modèle (la différence entre les valeurs prédites et les valeurs réelles) et retour aux données ou au modèle/algo si nécessaire.

**PHASE DE TEST**

$\mathbf{X}_{\text{Test}}$ Superficie données (testing)

$h_{\Theta_1,\Theta_2}(x)$

Prix estimé : $\mathbf{Y}_{\text{Prédit}}$

$\mathbf{Y}_{\text{Test}}$ Prix réel

$\mathbf{Y}_{\text{Prédit}}$ Prix estimé

# Méthode Générale

1. Compréhension de la problématique et du but à atteindre
2. Collecte et acquisition de données
3. Exploration numérique/mathématique et visualisation des données pour mieux les comprendre
4. Découper le jeu de données en jeu d'entraînement et jeu test (voir de validation)
5. Préparation des données
6. Sélection d'une fonction coût et entraînement de l'algorithme.
7. Sélection d'un modèle prédictif
8. Évaluation du modèle : ajustement et affinement du modèle
9. Vérification du modèle sur le jeu de test [retour possible vers 2, 3, 4 ou 6]
10. Présentation de la solution
11. Déploiement du modèle

# Jeux de Validation croisée

**Le jeu de données de validation** prend son sens quand on souhaite entraîner un modèle et/ou supprimer les hyperparamètres d'un algorithme d'apprentissage automatique. D'abord, on applique des algorithmes sur le jeu de données d'entraînement et on obtient des modèles. Puis, on utilise le jeu de validation pour calculer la performance de chacun d'eux et ne retenir que celui qui obtient le meilleur score. Une fois ce modèle retenu, on calcul sa performance sur le jeu de données test qui n'a jamais été utilisé jusqu'alors.

Cette approche offre la possibilité d'optimiser les paramètres des modèles tout en gardant un jeu pour calculer leurs capacités de généralisation.

**Proportion possible des données dans chaque ensemble.**

60-50 % -> Pour l'apprentissage
20 % -> Cross validation
20-30 % -> Pour le test

80-60 % -> Pour l'apprentissage
20-40 % -> Pour le test

# Validation croisée en K passes

**La méthode de validation croisée en *K* passes** (*K-fold cross validation*) est une technique statistique visant à calculer la performance des modèles. Il ne s'agit pas d'une métrique de performance, mais d'un procédé pour constituer les jeux test.

Le jeu de données est divisé en *K* sous-ensemble. On en sélectionne un comme jeu test et les autres représentent l'ensemble d'apprentissage. La performance du modèle est calculée avec cette configuration.

- Ensuite, on répète l'opération en sélectionnant un autre échantillon comme jeu de test et le reste comme jeu d'entraînement.
- L'opération se répète ainsi *K* fois de telle façon que, à la fin, chaque sous-ensemble a été utilisé une fois comme ensemble test.
- La mesure de performance finale est la moyenne des mesure de performance des *K* passes.

Il est à noter qu'avec cette méthode, on entraîne *K* fois l'algorithme d'apprentissage. Cela risque d'être coûteux si le jeu de données est grand. Toutefois, la mesure obtenue sur **la capacité de généralisation du modèle** est généralement satisfaisante.

# Apprentissage artificiel

Une définition de l'apprentissage artificielle donné par *Tom Mitchell* en *1997*

Étant donné :

de l'expérience E, → Les données d'apprentissage.

une classe de tâches T → Modéliser des données.

et une mesure de performance P → La fonction coût.

On dit d'un ordinateur qu'il apprend si :

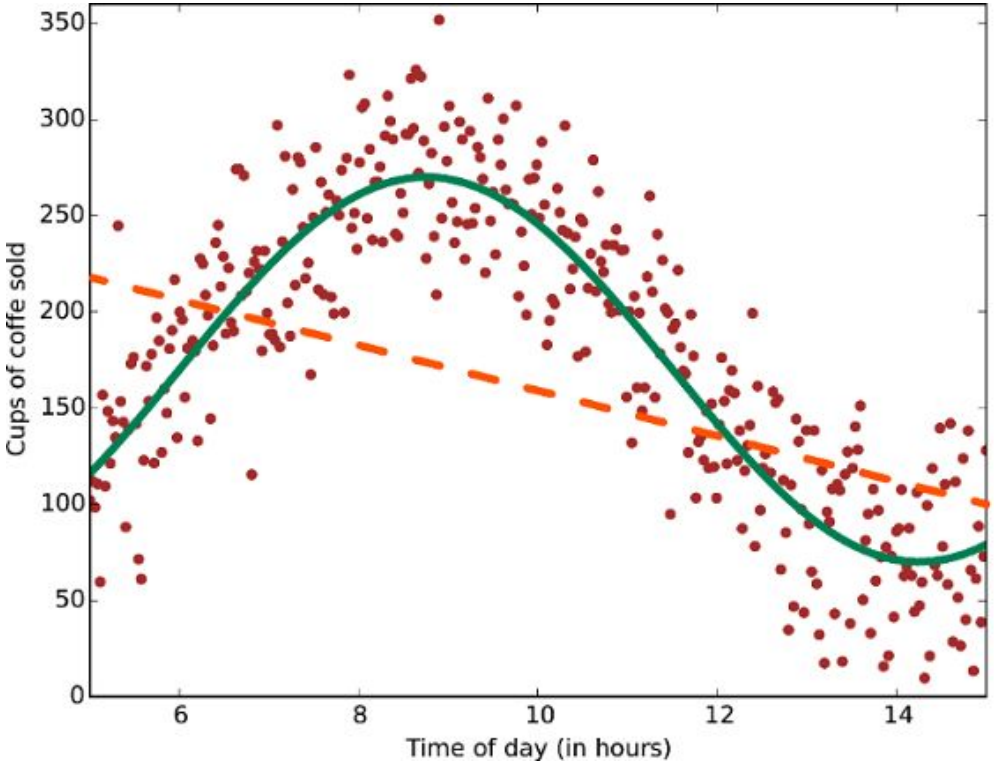
sa performance sur une tâche de T

mesurée par P

augmente avec l'expérience E

# - Qualité d'un modèle -
# &
# Sous-ajustement et Sur-ajustement
# -
# Biais et Variance

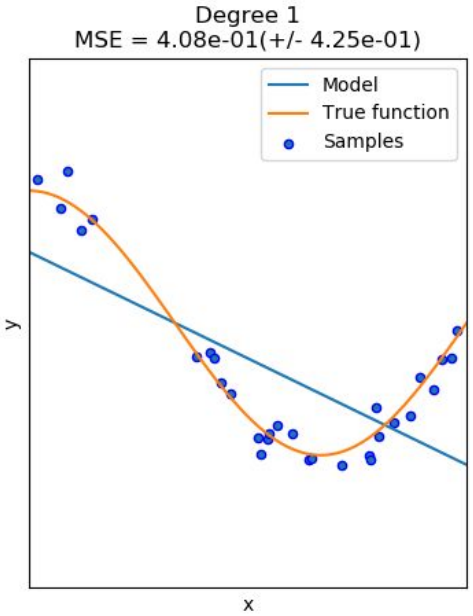
# Fonction de base non linéaire

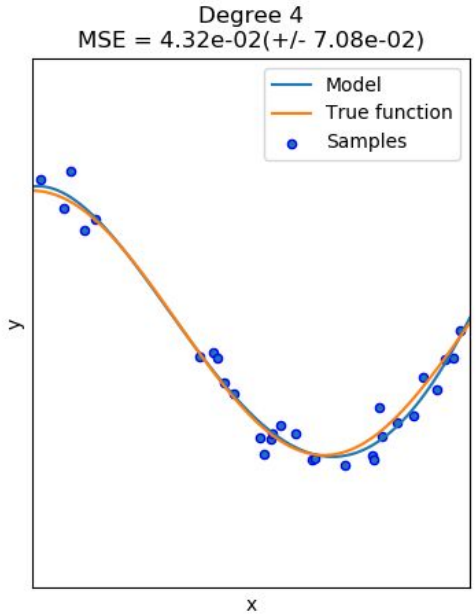


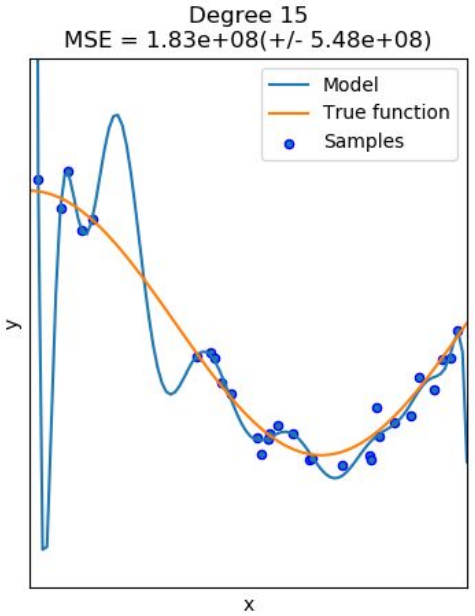

Ici on doit utiliser un polynôme.

Jusqu'au faut-il "modéliser" nos données ?

# Biais *v.s.* Variance




Biais / sous-ajustement

OK

Variance / sur-ajustement

https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

# Biais *v.s.* Variance

**Définition du Bias**

Le biais est le fait de formuler des suppositions et les appliquer sur un modèle prédictif. Par exemple, on peut supposer une relation linéaire entre deux variables ***X*** et ***Y*** ; cette hypothèse est un biais. Pour un pouvoir prédictif optimal, il ne faut pas avoir un fort biais, au risque de tombé dans le sous-ajustement.

**Définition de la Variance**

La variance reflète la sensibilité du modèle face à un changement dans les données d'entraînement. Un modèle avec une forte variance est bien adapté aux données d'entraînement mais se généralise difficilement (cela varie beaucoup) à des données nouvelles.

# Biais *v.s.* Variance

## Définition la régulation

La régulation est une technique qui vise à réduire le surajustement d'un modèle prédictif en ajoutant des contraintes lors de l'apprentissage. Le but est de réduire la variance du modèle pour jouir d'une meilleure capacité de généralisation.

La contrainte introduite par la régulation est une fonction de pénalité qui va réduire la magnitude des paramètres appris par le modèle. La fonction de pénalité a pour but de rétrécir l'espace des valeurs que peuvent prendre les paramètres du modèle et de garder leurs petites valeurs.

En pratique, de petites valeurs de paramètres rendent plus généralisable avec une fonction plus lissée (faible variance).

# Fonctions non linéaires possibles

**Régression de Ridge**

Ridge est une régression linéaire multivariée régularisée. La fonction de coût comporte un terme de pénalité. La fonction de coût pour une régression Ridge utilise une fonction de pénalité basée sur la norme L2 (distance euclidienne) :

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} h_\theta (x_i - y_i)^2 + \lambda \sum_{j=1} \theta_j^2$$

Le paramètre Lambda sert comme levier pour augmenter ou réduire l'effet de pénalité et plus petites seront les valeurs des paramètres du modèle. Plus forte est la valeur de la pénalité et plus petites seront les valeurs des paramètres du modèle.

Par conséquent, en augmentant la valeur de lambda, on réduit la variance et on augmente le biais.

## Gradient descent avec la régulation : Lambda

Précédemment $(n = 1)$ :

repeat {

$$\theta_0 := \theta_0 - \alpha \overbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}^{\frac{\partial}{\partial\theta_0}J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)} \quad + \lambda\,\theta_1^2$$

simultaneously update : $\theta_0, \theta_1$

}

$x_1^{(i)}$

Nouvel algorithme $(n \geq 1)$ :

repeat {

$$\theta_j := \theta_j - \alpha \overbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}}^{\frac{\partial}{\partial\theta_j}J(\theta)}$$

simultaneously update $\theta_j$ for $j = 0, \ldots, n)$

}

$x_0^{(i)} = 1$

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)} \quad + \lambda\,\theta_1^2$$

$$\theta_2 := \theta_2 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)} \quad + \lambda\,\theta_2^2$$

# Fonctions non linéaires possibles

## Régression LASSO

LASSO (*Least Absolute Shrinkage and Selection Operator*) est une régression linéaire multivariée régularisée. La fonction de coût comporte un terme de pénalité basé sur la norme l1 (distance de Manhattan) :

Le paramètre Lambda sert comme levier pour augmenter ou réduire l'effet de pénalité et plus petites seront les valeurs des paramètres du modèle. Plus forte est la valeur d modèle.

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} h_\theta (x_i - y_i)^2 + \lambda \sum_{j=1} |\theta_j|$$

**λ** est le facteur de régulation (l'intensité de régulation).

*Particularité :*

Il est à notre que la pénalité LASSO a la particularité de pouvoir sélectionner un sous-ensemble de variables, en fixant certains coefficients à zéro. Cela a pour conséquence de simplifier davantage le modèle. À titre de comparaison, la méthode Ridge ne fait que réduire la magnitude des paramètres, mais sans les annuler.

# Fonctions non linéaires possibles

## Régression ElasticNet

ElasticNet est une régression linéaire multivariée régularisé. La fonction de coût, qu'on cherche à minimiser, comporte un terme de pénalité. Elle tire avantage de $J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} h_\theta (x_i - y_i)^2 + \lambda \sum_{j=1}^{n} \left[ (1-\alpha)\theta_j^2 + \alpha|\theta_j| \right]$

En pratique Ridge donne de meilleurs résultats que LASSO lorsque les variables sont corrélées (ce qui est souvent le cas). Cependant, LASSO a la capacité d'annuler les coefficients de certaines variables et, par conséquent, de simplifier le modèles.

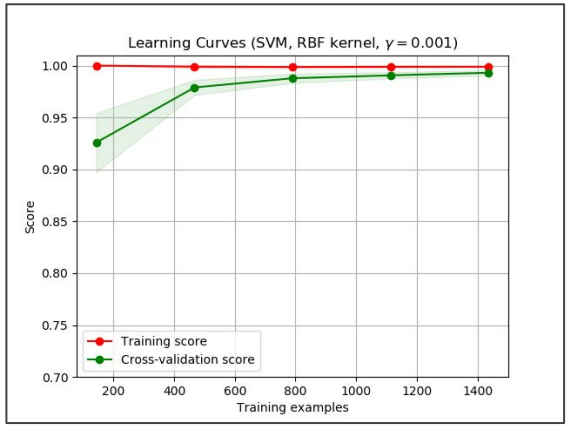
**λ** est le facteur de régulation (l'intensité de régulation).
**α** est un paramètre compris entre 0 et 1, qui sert à définir un équilibre entre les méthodes LASSO (**α**=1) et ridge (**α**=0).

*Particularité:*

En jouant sur la valeur du paramètre, on peut opter pour un comportement proche de l'une ou de l'autre régularisation.

# Learning Curve

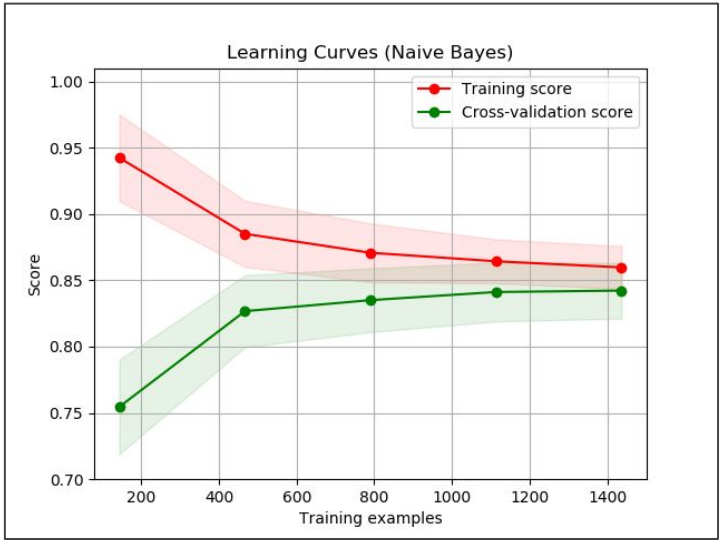
On analyse la qualité de l'apprentissage
sur les données d'apprentissage *versus* les données test / validation.




Pour obtenir cette courbe nous créons des ensembles de taille croissantes du nombre de données dans l'ensemble "Training". Pour chacun de ces ensembles, nous entraînons l'algorithme, et on calcul le score des prédictions (en générale $R^2$) sur ensemble Training ET sur l'ensemble Testing/Cross-Validation correspondant.

**Objectif : Évaluer si l'augmentation du nombre de donnée améliore votre modèle ?**

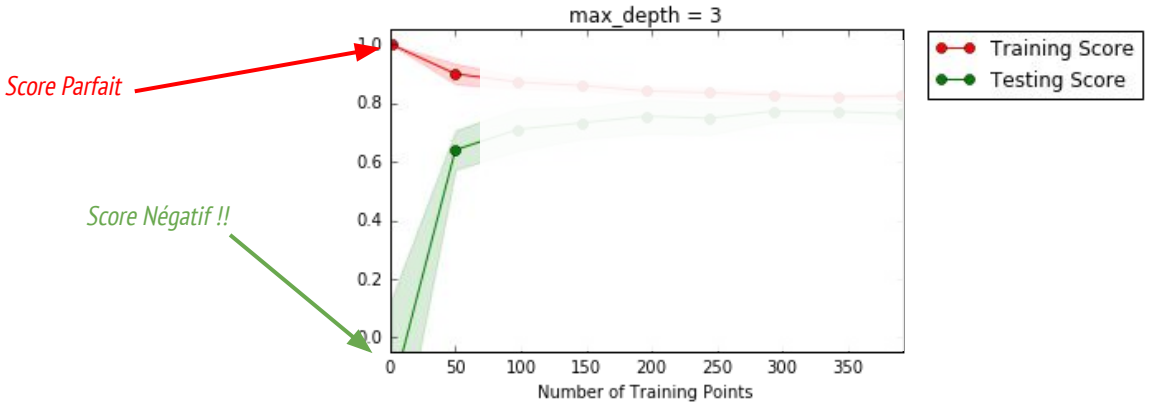
Cette courbe permet d'évaluer la qualité du modèle durant la phase de l'apprentissage (puisqu'on augmente progressivement l'ensemble Training) depuis "le point de vue" des données de Training (courbe rouge) et depuis les données de Cross-Validation. Un bon modèle est un modèle qui continue de s'améliorer (même faiblement) avec la taille des données. Mais ces courbes permettent d'évaluer sur quels types de problèmes nous faisons face (biais, over-fitting, variance, manque de données).



# Learning Curve

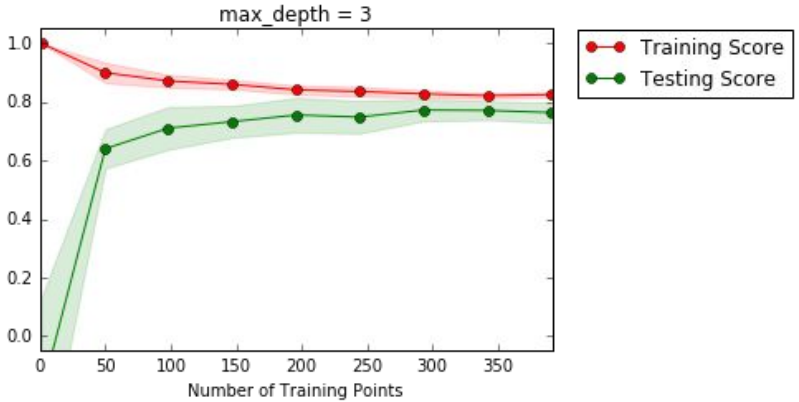
Les premiers points de la Learning Curve donnent souvent de très bon résultats pour le Training et de très mauvais scores pour le Cross-Validation.

- Imaginez que l'ensemble Training ne contient que 2 points. Ces 2 points pourront parfaitement être modélisés par une droite d'où le score parfait. Par contre, il y a peu de chance que ces 2 points décrivent au mieux vos données. Vous êtes dans le sur-apprentissage.
- Il y a peu de chance que les points qui sont contenues dans le Cross-Validation soit le long de la droite que vous avez défini avec 2 points (dans le cas de l'exemple précédent) d'où le score très médiocre.



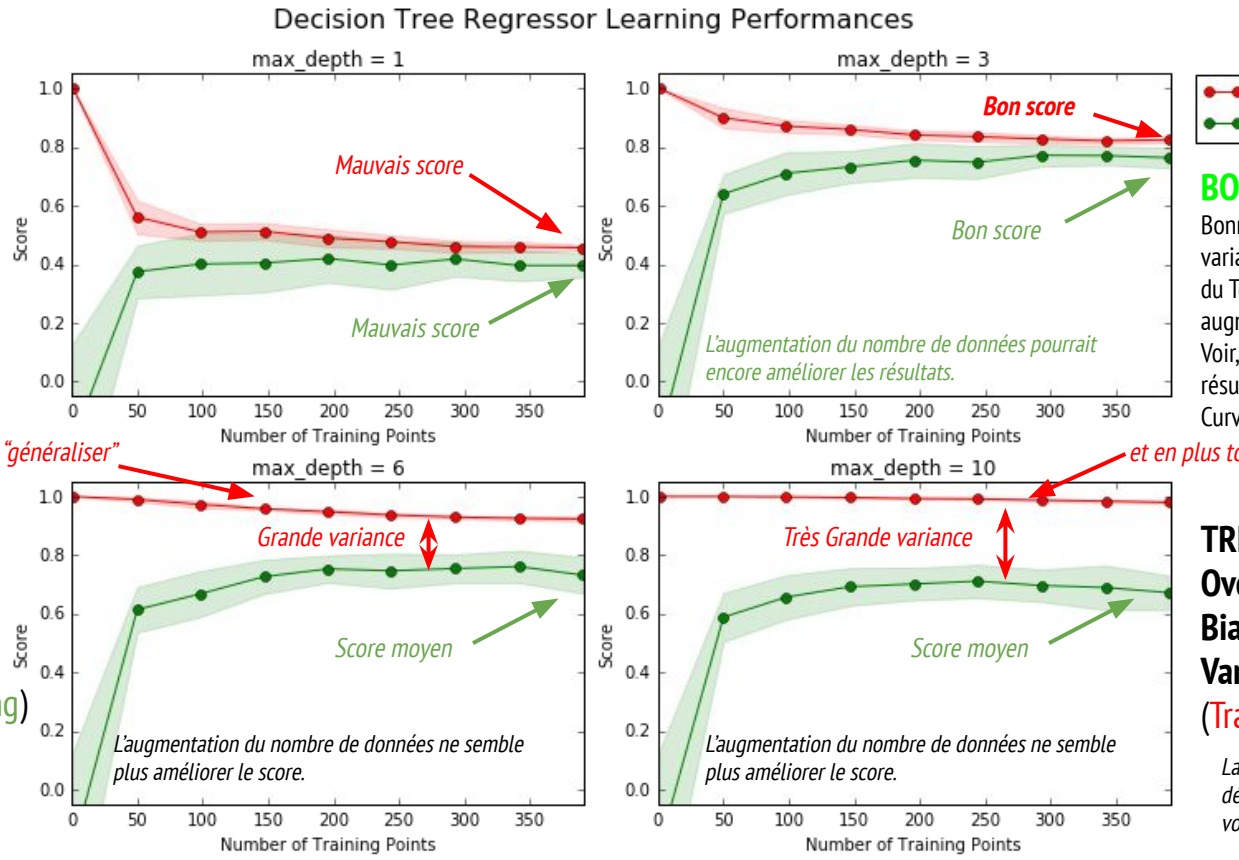
# Learning Curve

En augmentant le nombre de données dans l'ensemble Training, nous réduisons son score. Dans le même temps, l'amélioration du score sur l'ensemble Testing démontre que votre modèle commence à généraliser ses prédictions. Ici c'est les courbes sont autour d'un bon score ($R^2$ ~0.78) : c'est un bon apprentissage que l'on peut maintenant essayer d'améliorer par d'autres méthodes.



# Learning Curve

Contrôler la qualité de l'apprentissage en évaluant le type de problème auquelle vous faites face : **Biais *versus* Overfitting**




**MAUVAIS**
**Biais** sur le Training
et sur le Testing

**BON**
Bonne généralisation avec une faible variance et un bon score. Mais le score du Testing "plafonne" même en augmentant le nombre de données. À Voir, si vous pouvez améliorer vos résultats en passant par la Validation Curve.

**MOYEN**
**Overfitting** (Training) +
**Biais** (Testing) +
**Variance** (Training/Testing)

**TRÈS MOYEN**
**Overfitting** (Training) +
**Biais** (Testing) +
**Variance**
(Training/Testing)

*La notion de bon ou mauvais score dépend de l'objectif de qualité que vous fixez.*

# Validation Curve

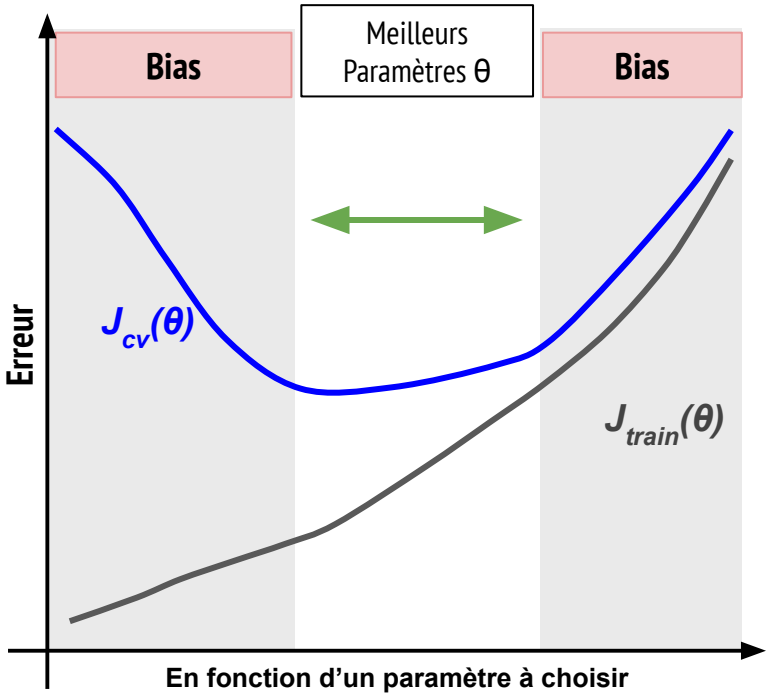
On étudie la qualité de l'apprentissage en fonction d'un paramètre.

C'est le même principe que la Learning Curve mais dans ce n'est plus la taille de l'ensemble de Testing que nous varions mais n'importe quel autre paramètre qui pourrait faire varier la qualité des prédictions.

Cela vous permet d'identifier les paramètres qui influencent le plus votre modèle.
NE SURTOUT PAS calculer le score sur l'ensemble TESTING au risque d'ajuster votre modèle aux données tests comme vous le faite quand votre algorithme ajuste le modèle sur les données du Training durant l'apprentissage.

Les données de l'ensemble Testing doivent, au mieux, représenter les données qui proviennent "du terrain" et vous comporter comme tel. Dans la vraie vie, vous ne pourrez pas ajuster votre modèle avec les données de terrain en temps réel, donc comportez vous de la même manière avec vos données tests.
Mode Maniac : Si vous avez un grand nombre de donnée et de nombre

# Biais *v.s.* Variance : Paramétrisation




$J_{train}(\theta)$ *Erreur sur l'ensemble des données d'apprentissage*

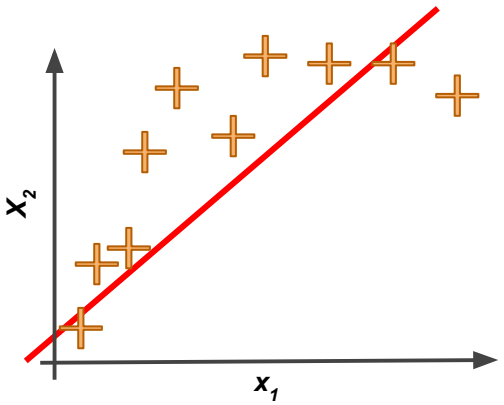
$J_{cv}(\theta)$ *Erreur sur l'ensemble des données de Cross Validation.*

Si vous devez ajuster le modèle (pas l'algorithme), comme le nombre degrée d'une régression polynomial ou tout autre paramètre définie dans votre modèle (mathématique). Dans ce cas, vous ne pouvez pas utiliser le jeu de données Test au risque d'introduire un Biais.

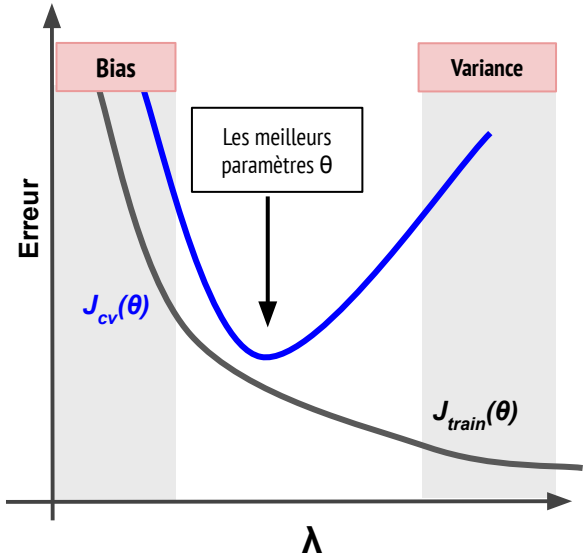
Pour éviter ce biais, vous devez définir un nouvel ensemble de données qui ne sont ni présente dans l'ensemble d'apprentissage, ni dans l'ensemble de test. : l'espace de validation (cross-validation).

Vous pouvez également utiliser la "*Learning Curve*" pour définir les paramètres de votre modèle.

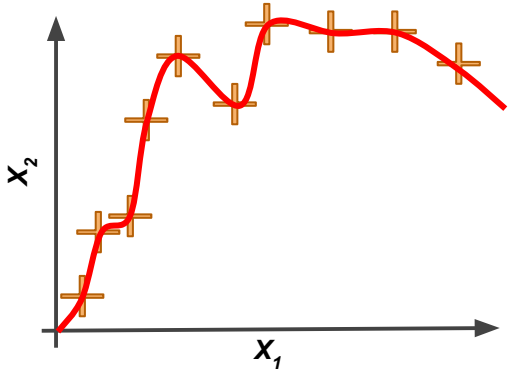
# Biais *v.s.* Variance




***Sous-apprentissage*** ou
***underfitting*** ou
***Sous-ajustement*** ou encore
***Biais***

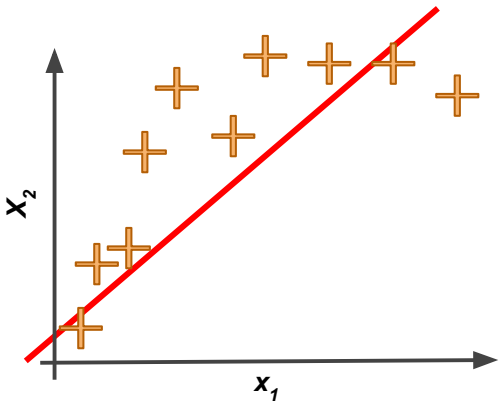



On calcul de l'erreur ***J*** sur
- le groupe de donnée Training
- le groupe « Cross Validation »

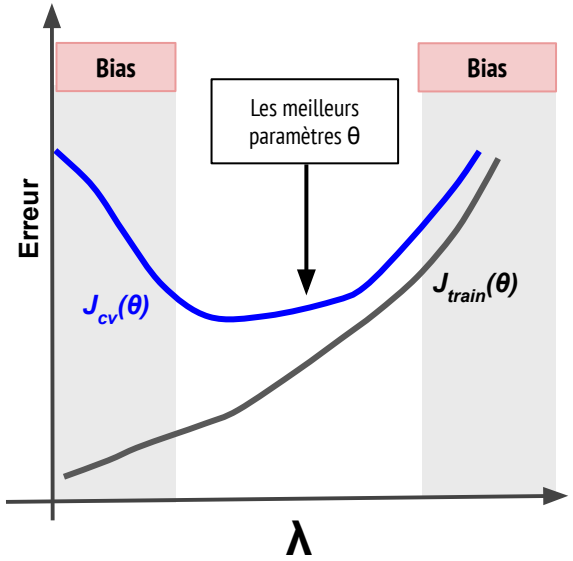


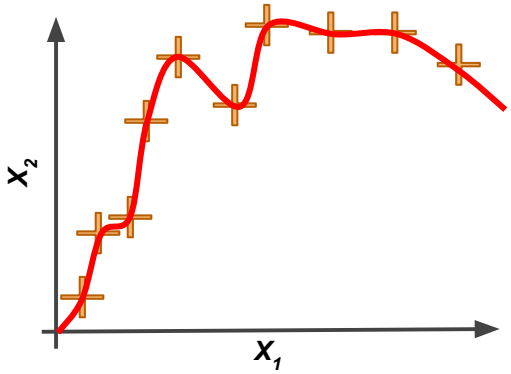

***Sur-Apprentissage*** ou
***Overfitting*** avec un forte
Variance sur les données tests

# Biais *v.s.* Variance




Sous-Apprentissage
Biais (underfitting)




Sur-Apprentissage
(overfitting)

# Quoi faire ?

## Pour éviter l'Over-fitting / Sur-ajustement

- Utiliser les techniques d'échantillonnage pour mieux prédire les performance du modèle (*K-Fold Cross Validation*)
- Réduire le nombre de nombre de dimension/features (colonnes) du jeu de données et ne garder que celles affectant fortement la prédiction
- Augmenter le nombre de données
- Utiliser les techniques de régulation en fonction de l'algorithme
- Changer de cost-function ou de scoring prenant en compte les classes qui sont sous représentées.

## Pour éviter le Biais / Sous-ajustement

- Choisir un modèle prédictif plus performant, notamment un algorithme non paramétrique (en effet, il est possible que l'algorithme utilisé souffre d'un fort biais)
- Augmenter la taille du jeu de données d'apprentissage
- Augmenter le nombre de dimension/feature (colonne) du jeu de données
- Si on a appliqué une régulation au modèle, réduire la pénalité de cette dernière pour réduire le biais.

# Quoi faire ?

- Changer d'algo ou changer ses paramètres (degrée des polynômes par exemple)
  → résout prb de grand biais / variance

- Développer son algo (ou modifier un algo existant)
  → résout prb de grande variance / biais

- Varier Lambda [régularisation]
  → résout prb de grande variance

- Réduire Lambda [régularisation]
  → résout prb de grand biais

- Obtenir plus de données
  → peut résoudre prb de grande variance

- Développer de nouvelles features / + grand réseau de neurone (+ de couche + unités)
  → résout prb de grand biais

- Supprimer des features/colonnes ou changer le modèle du réseau de neurone
  → résout prb de grande variance

- Revoir le nettoyage des données
  → résout prb de grand biais / variance

# Avez-vous tout tenté ?

Votre learning Curve est bonne mais vous voulez améliorer vos résultats.

**- Étudier les données qui donnent de mauvais résultats**

→ Qu'est-ce qui caractérise les données (X) qui produisent de mauvaises prédictions (Y) par rapport aux données qui fournissent de bonnes prédictions ?

→ En fonction des ensembles de données qui fournissent les meilleures prédictions, réduisez votre domaines d'études. Il faut que cela soit toujours une prédiction "utile" (*exemple: vous réduisez l'intervalle de l'âge de la population sur lequel vous vouliez travailler,...*)

**- La qualité de vos données est-elle égale ?**

→ Pouvez vous évaluer la qualité de vos données les unes par rapport aux autres et introduire un scoring (une nouvelle feature) qui mesure cette qualité ? Cette features donnera un score de "confiance" sur chaques données qui pourraient être prises en compte dans un algorithme (comme dans le cas des données non balancées).

**- Au cas par cas**

→ Ajouter une feature qui correspond à un calcul fait sur d'autres features existantes (quand cela a du sens, *exemple calculer la valeur de l'accélération si vous avez la vitesse et le temps*).

→ Ajouter des features provenant d'apprentissage non supervisé appliqué dans un autre ensemble Training. Avec le KNN vous pouvez classifiez les données en plusieurs ensembles (définissez plusieurs nombre de clusters et/ou définissez des métriques). Une fois que vous avez un modèle/algorithme qui classifie vos données, vous utilisez cette "nouvelle features" à vos données. Attention, dans la vraie vie, vous augmentez le pré-traitement des données avant la prédiction.

→ Créer des données synthétiques et augmenter la taille du nombre de données d'entraînement (et uniquement dans la phase d'entraînement).

- Régression linéaire -

&

- Normalisation -

# Feature Scaling : Normalisation

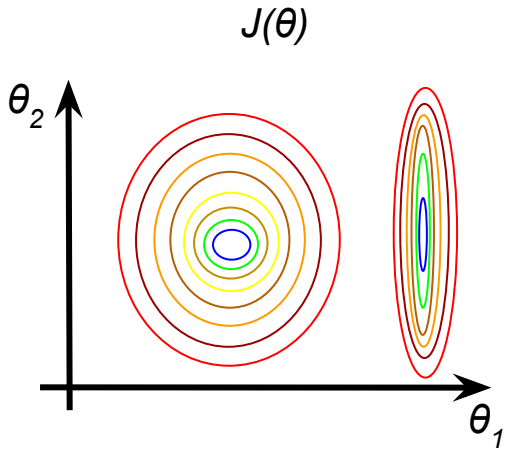
Cette fois nous avons plusieurs colonnes

$x_1$ = surface (0-2000 feet$^2$)
$x_2$ = nombre de chambre (1-5)

$$x_1 = \frac{\text{surface (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{Nombre de chambre}}{5}$$

Lorsque les caractéristiques/colonnes/features étudiées ont des ordres de grandeur différents, on déforme la forme de la fonction coût (gradient).




**Objectif :** Obtenir des « features » ayant le même ordre de grandeur préférentiellement entre

$$-1 \leq x_i \leq 1$$

# Feature Scaling : Min-Max Scaler

sklearn.preprocessing.scale

## *Min-Max Scaler*

La normalisation (min-max scaling) est utile lorsque les caractéristiques étudiées ont des ordres de grandeur différents. À l'issue de la transformation, elles se retrouvent bornées dans l'intervalle [0, 1]. La technique consiste, pour chaque caractéristique, à soustraire la valeur minimale et à diviser le tout par l'étendue (max-min) :

$$X_{Normalisé} = \frac{X - min(X)}{max(X) - min(X)}$$

Sensible aux données aberrantes (outlier).

# Feature Scaling : Min-Max Scaler

sklearn.preprocessing.scale

**Objectif :** Obtenir des features ayant le même ordre de grandeur

$x_1$ = surface (0-2000 feet$^2$)

= nombre de chambre (1-5)

## Mean normalization

$$x_1 = \frac{\text{Surface} - 1000}{2000}$$

$$x_2 = \frac{\text{Nbr de chambres} - 2}{5}$$

Remplacer $x_i$ avec pour que les features aient une moyenne autour de 0 (Ne pas appliquer à ).

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

# Feature Scaling : Robust Scaler

sklearn.preprocessing.RobustScaler

## *Robust Scaler*

La technique  Robust Scaler utilise le même principe de mise à l'échelle que Min-Max scaling. Néanmoins, elle utilise l'intervalle interquartile au lieu du min-max, ce qui la rend plus fiable vis à vis des outliers.

$$X_{Robust\ Scaler} = \frac{x - Q(x)}{Q_3(x) - Q_1(x)}$$

1er quartile correspond au 25e centile d'une distribution d'une distribution:

il est noté $Q_1$

3ème quartile correspond au 75e centile d'une distribution d'une distribution:

il est noté $Q_3$

En statistique, un centile (appelé aussi percentile) est l'observation pour laquelle X % des valeurs du jeu de données sont plus petites que la valeur du centile.

index=(Percentile/100) * Nbr d'observation

# Feature Scaling : Standardisation

sklearn.preprocessing.standardisation

## *Standardisation*

La standardisation (*Z-score normalisation*) est utile lorsque les caractéristiques étudiées ont des ordres de grandeur/magnitudes différents. À l'issue de cette manipulation, chaque caractéristique répond à une loi normale (loi gaussienne) ayant une moyenne nulle et un écart-type de 1. Pour chaque caractéristique/feature/colonne, la technique consiste à soustraire la moyenne (pour avoir une moyenne nulle) et à diviser par son écart-type. Les valeurs obtenues sont en majorité entre dans l'intervalle [-1, 1] .

$$x' = \frac{x - \bar{x}}{\sigma}$$

La standardisation est moins affectée que la normalisation par les données aberrantes (outlier).

# Feature Scaling : Normalisation

Si vous utilisez un algorithme qui utilise la notion de distance ou la normalisation, il faut **“scaler” les features**.

- **k-nearest neighbors** avec des distances Euclidienne est sensible aux magnitudes différentes entre les features. Les features doivent donc être normalisées.
- Le Scaling est critique lorsque l’on utilise le **Principal Component Analysis(PCA)**. Le PCA analyse la variance maximum des features.
- La normalisation permet d'accélérer l’algorithme le “**Gradient Descent**”. En effet, Alpha descendra rapidement sur de petites étendues et lentement sur de grandes étendues, et ira donc de manière inefficace vers l'optimum lorsque les variables sont très inégales.
- Les modèles basés sur **des arbres** ne sont pas des modèles basés sur la distance et peuvent gérer diverses gammes de fonctionnalités. Par conséquent, la mise à l'échelle n'est pas requise lors de la modélisation des arbres.
- Des algorithmes tels que l'**Analyse Discriminante Linéaire** (LDA), **Naive Bayes**, sont par nature conçus pour gérer la différence de magnitude et permettent de pondérer les caractéristiques en conséquence. Effectuer une mise à l'échelle des fonctionnalités dans ces algorithmes peut ne pas avoir beaucoup d'effet.

# - Qualité d’un modèle -
# &
# Métrique

# Erreur quadratique moyenne (RMSE)

L'erreur quadratique moyenne (Root-Mean-Square Error - RMSE) est une métrique de performance d'un modèle de régression qui mesure l'écart entre les valeurs prédites et les valeurs réelles. Elle calcule la racine carrée de la moyenne de la somme des carrés des écarts entre une observation et la valeur prédite par un estimateur.

$$\text{RMSE} = \sqrt{\frac{1}{M} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Comme elle a la même unité que la variable à expliquer, cela lui donne une signification fonctionnelle. Plus sa valeur est basse, plus l'écart est petit et meilleur est le modèle.

Cette mesure pénalise plus fortement les grands écarts (les grandes erreurs). Par conséquent, elle est sensible à la présence de nombreuses valeurs aberrantes.

$\hat{y}_i$

*M* est le nombre d'observations (lignes) du jeu de données.
est la valeur prédite par un estimateur pour la *i*-ème observation.
$y_i$ est la valeur observée dans le jeu de données.

# Erreur absolue moyenne (MAE)

L'erreur absolue moyenne (*Mean Absolute Error - MAE*) mesure la performance des modèles de régression.

C'est la moyenne de la somme des valeurs absolue des erreurs de prédiction :

$$\text{MAE} = \frac{1}{M} * \sum_{i=1}^{n} \left| y_i - \hat{y}_i \right|$$

$\hat{y}_i$

$M$ est le nombre d'observations (lignes) du jeu de données.
est la valeur prédite par un estimateur pour la $i$-ème observation.
$y_i$ est la valeur observée dans le jeu de données.