Listes Algorithmes Importants en Deep Learning

Réseaux de neurones artificiels (ANN)

- **Utilisation : Classification, régression, traitement du langage naturel.**

- **Fonctionnement : Modèle de calcul inspiré du cerveau humain avec des couches de neurones interconnectés.**

Réseaux de neurones convolutifs (CNN)

- **Utilisation : Vision par ordinateur, reconnaissance d'images.**

- **Fonctionnement : Utilisation de couches de convolution pour extraire des caractéristiques d'images.**

Réseaux de neurones récurrents (RNN)

- **Utilisation : Traitement séquentiel, traitement du langage naturel.**

- **Fonctionnement : Modèles qui prennent en compte la séquence temporelle.**

Réseaux de neurones récurrents à mémoire court et long terme (LSTM)

- **Utilisation : Traitement du langage naturel, séquences temporelles.**

- **Fonctionnement : Modèles RNN améliorés avec une meilleure gestion de la mémoire à long terme.**

Réseaux de neurones récurrents à portes (GRU)

- **Utilisation : Traitement du langage naturel, séquences temporelles.**

- **Fonctionnement : Variante des RNN avec des portes pour contrôler le flux d'informations.**

Réseaux de neurones auto-encodeurs

- **Utilisation : Réduction de dimension, débruitage, génération d'images.**

- **Fonctionnement : Réseau qui tente de reproduire l'entrée en passant par une couche cachée de dimension réduite.**

Réseaux de neurones génératifs adversaires (GAN)

- **Utilisation : Génération d'images, de texte, de sons.**

- **Fonctionnement : Deux réseaux (générateur et discriminateur) s'entraînent en opposition pour générer des données réalistes.**

Auto-encodeurs variationnels (VAE)

- **Utilisation : Génération d'images, réduction de dimension.**

- **Fonctionnement : Variante des auto-encodeurs qui modélise la distribution probabiliste des données latentes.**

Réseaux de neurones récurrents bidirectionnels (Bi-RNN)

- **Utilisation : Traitement du langage naturel, reconnaissance de séquences.**

- **Fonctionnement : Les informations sont propagées dans les deux sens, du passé au futur et du futur au passé.**

Réseaux de neurones récurrents à mémoire adaptative (AM-Net)

- **Utilisation : Traitement du langage naturel, mémoire améliorée.**

- **Fonctionnement : Introduit une mémoire adaptative pour gérer des informations plus complexes.**

Réseaux de neurones convolutifs 1D (Conv1D)

- **Utilisation : Traitement de séquences temporelles unidimensionnelles.**

- **Fonctionnement : Applique des opérations de convolution sur des données 1D.**

Réseaux de neurones résiduels (ResNets)

- **Utilisation : Réseaux très profonds, vision par ordinateur.**

- **Fonctionnement : Introduit des connexions résiduelles pour faciliter l'apprentissage de réseaux profonds.**

Réseaux de neurones pré-entraînés (example Word2Vec, GloVe)

- **Utilisation : Traitement du langage naturel.**

- **Fonctionnement : Apprend des représentations de mots à partir de grands corpus de texte.**

Réseaux de neurones profonds récurrents (Deep RNN)

- **Utilisation : Modèles profonds pour le traitement séquentiel.**

- **Fonctionnement : Empile plusieurs couches RNN pour des modèles plus complexes.**

Réseaux de neurones mémorisants (Memory Networks)

- **Utilisation : Traitement du langage naturel, mémorisation de séquences.**

- **Fonctionnement : Introduit des mécanismes de mémoire pour stocker et récupérer des informations importantes.**

Réseaux de neurones de convolution 3D (3D CNN)

- **Utilisation : Traitement de données volumétriques ou séquences 3D.**

- **Fonctionnement : Extension des CNN pour des données 3D.**

Réseaux de neurones entièrement convolutifs (FCN)

- **Utilisation : Segmentation d'images, traitement sémantique.**

- **Fonctionnement : Utilise uniquement des couches de convolution, pas de couches entièrement connectées.**

Réseaux neuronaux adversaires conditionnels (cGAN)

- **Utilisation : Génération d'images conditionnelles.**

- **Fonctionnement : Les GANs conditionnels génèrent des données en fonction d'une condition donnée.**

Réseaux neuronaux récurrents à mémoire à long terme avec oubli de porte (LSTM-FO)

- **Utilisation : Traitement du langage naturel.**

- **Fonctionnement : Une variante de LSTM qui gère la mémoire de manière plus fine.**

Réseaux de neurones profonds convolutionnels stratifiés (ResNeXt)

- **Utilisation : Vision par ordinateur.**

- **Fonctionnement : Une variante des ResNets qui utilise des stratifications parallèles pour augmenter la performance.**

Transformers

- **Utilisation : Traitement du langage naturel, traduction automatique.**

- **Fonctionnement : Modèles basés sur des mécanismes d'attention pour capturer les relations entre les éléments de séquences.**

Flow neuronaux

- **<u>Utilisation</u> : Génération d’images, densité de probabilité.**

- **<u>Fonctionnement</u> : Modèles qui modélisent la distribution de probabilité des données et permettent la génération d'échantillons.**

Conseil pour le choix d'un modèle :

1-La Quantité de données que vous avez.

Par exemple, si vous avez de gros Dataset, les modèles adaptés seront plus la Régression Logistique / les Réseaux de Neurones.

A l'inverse si vous avez peu de données, prenez plutôt des algo comme le Support Vector Machines / K-Nearest Neighbours.

=> Si vous avez moins de -100 000 points -> n'importe quels modèles de ML fera l'affaire.

=> Si vous avez plus de +100 000 points -> modèles qui fonctionnent avec la Descente de Gradient.

2-Travailler uniquement avec les algo que vous comprenez.

Sinon, vous risquez de mal déployer l'algo et ou de mal optimisé ses hyper-paramètres.

3-La Structure de vos données (Structurées / Non-Structurées).

On considère en ML que les données qui sont des images, du texte, du son etc

=> ce sont des données non structurées.

A l'inverse, données tabulaires (Excel, Csv..)
=> ce sont des données structurées.

Quand on est en présence de données non-structurés :
Utilisation de réseaux de neurones (Deep-Learning).

Quand on est en présence de données structurés :
Utilisation de modèles (machine-Learning).

4-La Normalité des données.
Il existe 2 types de modèles, Paramétriques / Non-Paramétriques.

=> modèle paramétrique = c'est un modèle qui correspond à une fonction avec un nombre de paramètre limité et défini à l'avance.
Exemples :

- **modèles de régression linéaire**
- **modèles de régression logistique**
- **modèles de Naives Bayes**
- **modèles de petit réseau de neurones**

Explication rapide :

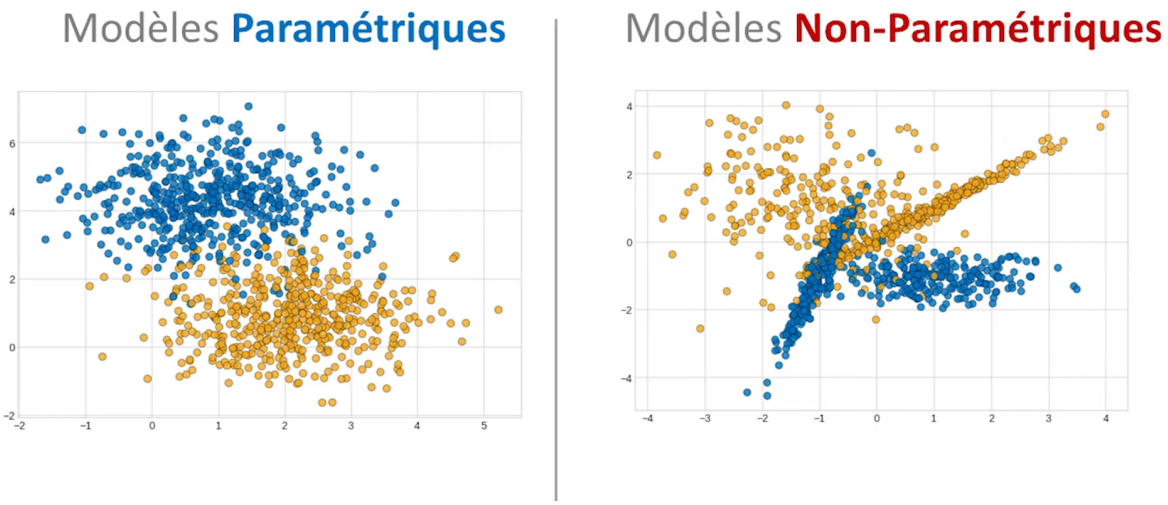
Si on prend par exemple la régression linéaire, alors f(x)=ax+b, on a 2 paramètres ici a et b, ni plus ni moins.

=> modèles non-paramétriques = c'est un modèle qui correspond à une fonction avec un nombre de paramètres infini et inconnu à l'avance.

Exemples :

- **Support Vector Machines**
- **Arbres de décision**
- **Random Forest**
- **K-Nearest Neighbors**

Les modèles paramétriques fonctionnent très bien sur les données qui suivent des lois de probabilités normales.

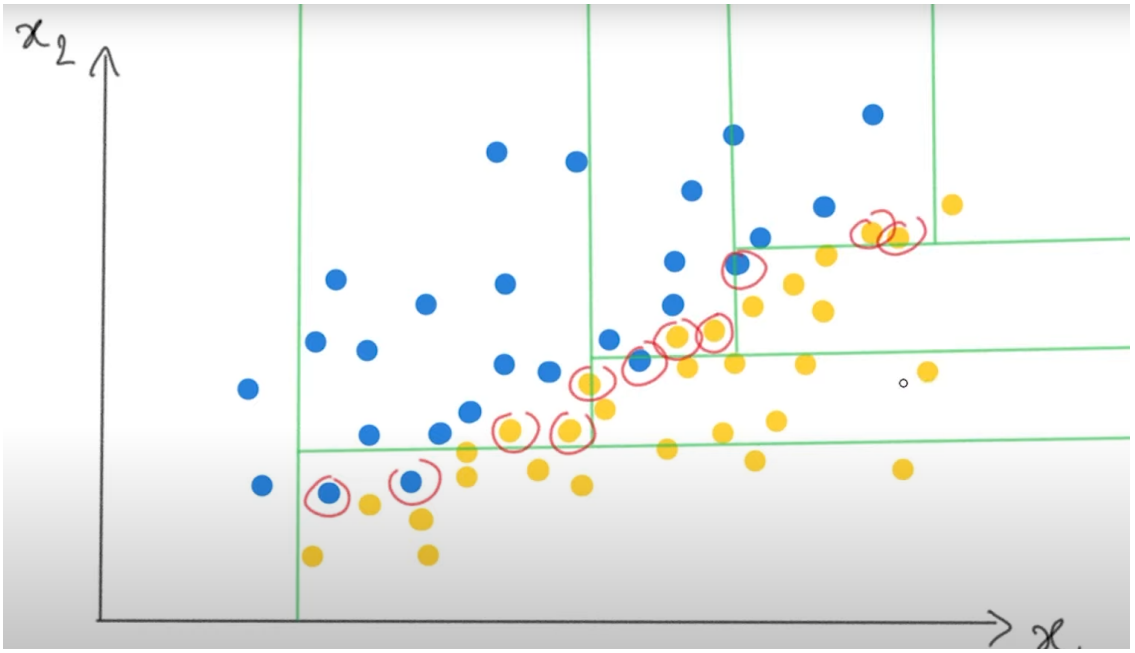


On peut parfois (ça dépend des données) en faisant du pré-processing et ou du features engineering

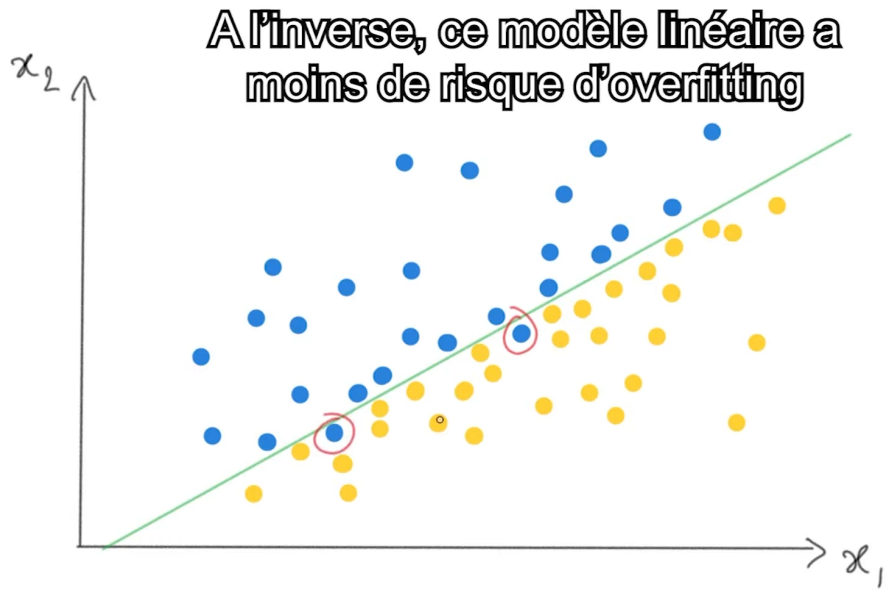
transformer nos données de sortes à ce qu'elles suivent des lois de probabilités normales.

5-Quantitées de Variables Quantitatives / Qualitatives dans notre Dataset.

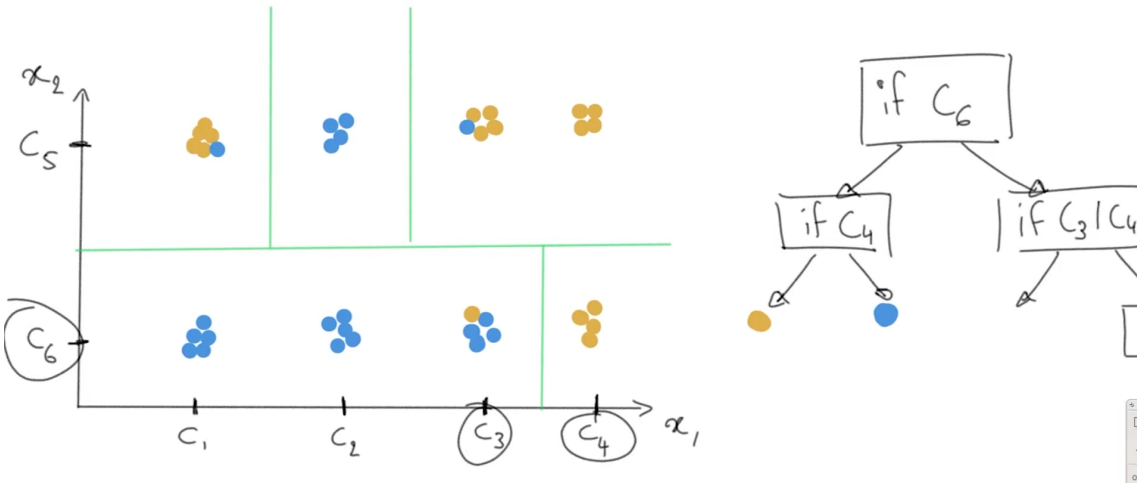
=>Il existe certains types de modèles comme les arbres de décisions qui ne sont pas efficaces lorsqu'on a beaucoup de variables quantitatives et surtout quand on observe des relations linéaires dans ces variables quantitatives.



On voit ici qu'il y a une classification mal faite à cause du système des modèles comme les arbres de décisions qui vont classer en traçant des droites orthogonales (forme d'escalier).



On voit à l'inverse qu'ici un modèle linéaire fait moins d'erreurs.



Mais si on a des données avec beaucoup de catégories, il ne faut pas hésiter à utiliser des arbres de décisions.

Conseil Bonus : Commencez TOUJOURS par utiliser le modèle le plus simple et après si besoin allez vers du plus complexe.

Dans la Réalité, le mieux étant de tester tous les modèles (faire une Pipeline par exemple), et regarder celui qui va fournir la meilleure performance.

```
preprocessor = make_pipeline(PolynomialFeatures(2, include_bias=False), SelectKBest(f_classif, k=10))
```

```
RandomForest = make_pipeline(preprocessor, RandomForestClassifier(random_state=0))
AdaBoost = make_pipeline(preprocessor, AdaBoostClassifier(random_state=0))
SVM = make_pipeline(preprocessor, StandardScaler(), SVC(random_state=0))
KNN = make_pipeline(preprocessor, StandardScaler(), KNeighborsClassifier())
```

Les données sont toujours différentes, on ne peut pas faire de règle bien précise pour la distribution de ces données réelles, donc la meilleure méthode et de tester les algorithmes et dans les faits voir celui qui a la meilleure performance.

Le plus important restera la manière dont sont traitées nos données (pre-processing) avant même le choix du modèle (qui jouera une influence moindre).