Il est recommandé de mélanger puis de séparer les données en trois ensembles : *train*, *test* et *validation*. Les pourcentages conseillés sont de 60 %, 20 % et 20 % respectivement. Ensuite, le modèle est entrainé sur l'ensemble *train*, il est perfectionné sur l'ensemble de validation, et testé sur l'ensemble de *test*. Enfin, une fois que le modèle est optimisé, sa performance finale est évaluée sur l'ensemble de *test* pour estimer sa capacité à généraliser à de nouvelles données non vues. Ce processus permet d'obtenir une évaluation fiable de la performance du modèle et d'éviter le sur-ajustement aux données d'entraînement.

État de l'art des modèles de régression

Modèles simples:

Régression linéaire

Principe :
À partir d'un ensemble de variables indépendantes [X1,X2…Xn] appelées *features*, on cherche à déterminer une relation linéaire avec une variable d'une dimension en sortie appelée variable cible. On suppose la relation linéaire.
Il faut tenter de prédire les coefficients bêta et epsilon dans la relation :
$Y=\beta 0 +\beta 1\ X1 +\beta 2\ X2 +...+\beta n\ Xn +\epsilon$
Ces coefficients représentent les effets de chaque variable sur la variable cible.
En entrée du modèle : deux sets de variables, une variable qu'on cherche à prédire (dimension 1) et un tableau de features, qui serviront à déterminer la variable de sortie en fonction d'elles. Il est idéal, pour les features, de sélectionner les colonnes voulues. Il est parfois conseillé de normaliser les données, pour aider à la convergence de l'algorithme.
Évaluation du modèle : De nombreuses métriques peuvent être utilisées, telle que R^2, RMSE, etc. La validation croisée est également souvent utile.
Conseils : La cross-validation est souvent utile. Pour éviter le sur-ajustement, il est utile d'utiliser d'autres types de régression (lasso, ridge etc).
Plus d'info sur : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Lasso:

Cette régression est similaire à la régression linéaire, mais elle ajoute une pénalité L1 à la fonction de perte, ce qui favorise la sélection de variables en forçant certains coefficients de régression à zéro. Cela permet une sélection automatique de variables et peut aider à réduire la dimensionnalité et à prévenir le surajustement (overfitting). La pénalité L1 est :

$$\text{Pénalité L1} = \alpha \sum_{i=1}^{n} |\beta_i|$$

α est le paramètre de pénalité, qui contrôle l'intensité de la pénalité. Plus α est élevé, plus la pénalité est forte.
β sont les coefficients du modèle.
La pénalité L1 encourage les coefficients du modèle à être aussi petits que possible et favorise la parcimonie en poussant certains coefficients vers zéro.
En entrée du modèle: On a la même entrée que pour la régression linéaire.

Conseils: Faire attention au paramètre alpha.
Pour plus d'infos:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

Ridge:

Principe : Similaire à la régression linéaire, mais avec une pénalité L2 ajoutée à la fonction de perte.
Utilisation : Modèle utilisé pour prédire une variable cible en fonction des features, tout en réduisant la variance du modèle et en évitant le surajustement.
Pénalité L2 : La pénalité L2 est définie comme la somme des carrés des coefficients multipliée par un paramètre de régularisation alpha.
Avantages : Réduit la variance du modèle en imposant des contraintes sur les coefficients, ce qui peut améliorer la généralisation et réduire le surajustement.
Paramètre clé : Alpha (α) contrôle l'intensité de la régularisation. Une valeur plus élevée de α correspond à une régularisation plus forte.
Évaluation : Les mêmes métriques que pour la régression linéaire peuvent être utilisées, telles que R^2, RMSE, etc. La validation croisée est également recommandée.
Conseils : Tout comme avec la régression Lasso, il est important de sélectionner judicieusement le paramètre alpha pour obtenir de bons résultats. La régression Ridge est particulièrement utile lorsque les données présentent une multicollinéarité élevée entre les features.

Modèles à dataset très large

- SGDRegressor (pas assez de données ?)

Principe : SGDRegressor est un modèle de régression linéaire qui utilise la descente de gradient stochastique pour minimiser la fonction de perte et estimer les coefficients du modèle.
Utilisation : Il est adapté aux ensembles de données volumineux en raison de sa capacité à effectuer des mises à jour de modèle incrémentielles sur de petits lots de données.
Entraînement : Le modèle est entraîné par itérations successives sur des mini-batchs de données, où les coefficients du modèle sont mis à jour à chaque itération pour minimiser la fonction de perte.
Avantages : Il peut converger plus rapidement que les méthodes traditionnelles de régression linéaire sur de grands ensembles de données, car il ne nécessite pas de charger tous les données en mémoire à la fois.
Inconvénients : La convergence dépend des hyperparamètres tels que le taux d'apprentissage et le nombre d'itérations, et il peut être plus sensible au bruit que les méthodes de descente de gradient batch.
Hyperparamètres : Parmi les hyperparamètres importants, on trouve le taux d'apprentissage (learning rate) et le nombre d'itérations (epochs), qui doivent être ajustés pour obtenir de bonnes performances.

Régularisation : La régularisation peut être appliquée à SGD Regressor pour contrôler le surajustement, notamment à l'aide de pénalités L1 ou L2.
Évaluation : Les mêmes métriques que pour la régression linéaire peuvent être utilisées pour évaluer les performances du modèle.
Conseils : Il est recommandé de normaliser les données pour faciliter la convergence de l'algorithme et de faire une sélection judicieuse des hyperparamètres pour obtenir de bons résultats.

- SVR

Principe : La SVR est une technique d'apprentissage supervisé utilisée pour la prédiction de variables continues. Elle est basée sur le même principe que les SVM pour la classification, mais adaptée à la régression.
Objectif : L'objectif de la SVR est de trouver une fonction (ou un hyperplan) qui s'ajuste au mieux aux données tout en limitant la marge d'erreur par rapport à la fonction de prédiction.
Fonctionnement : Contrairement à la régression linéaire traditionnelle qui cherche à minimiser les erreurs résiduelles, la SVR cherche à minimiser la violation de la marge (c'est-à-dire les erreurs de prédiction qui dépassent une certaine tolérance définie par un paramètre ε).
Entraînement : L'entraînement de la SVR consiste à régler les paramètres du modèle (comme les poids des caractéristiques et le biais) en minimisant une fonction de coût qui comprend à la fois la perte de régularisation et les erreurs de prédiction.
Utilisation : Une fois que le modèle SVR est entraîné, il peut être utilisé pour prédire les valeurs cibles pour de nouveaux points de données en appliquant simplement la fonction de prédiction apprise.
Avantages : La SVR est efficace pour la modélisation de relations non linéaires, et elle peut être robuste face au bruit et aux données aberrantes.
Inconvénients : Elle peut être sensible à la sélection des paramètres, comme le noyau utilisé, et peut nécessiter un réglage minutieux pour obtenir de bonnes performances.
Hyperparamètres : Des hyperparamètres tels que le paramètre de régularisation C et le noyau (linéaire, polynomial, gaussien, etc.) doivent être ajustés pour optimiser les performances du modèle.
Évaluation : Les performances de la SVR peuvent être évaluées à l'aide de métriques appropriées telles que l'erreur quadratique moyenne (RMSE) ou le coefficient de détermination (R^2).
Conseils : Il est souvent utile d'utiliser des techniques comme la validation croisée pour sélectionner les meilleurs hyperparamètres et éviter le surajustement.

Modèles non paramétriques

Arbres de décision

Si on a beaucoup de variables quantitatives, surtout quand on observe des relations linéaires entre les variables il faut éviter le modèles basées sur des arbres de décision car ils augmentent le risque d’over-fitting.

Principe : Les arbres de décision sont des modèles d'apprentissage supervisé utilisés pour la classification et la régression. Ils fonctionnent en divisant récursivement l'ensemble de données en sous-ensembles homogènes en fonction des caractéristiques.
Structure : Ils se présentent sous la forme d'une structure arborescente où chaque nœud interne représente une caractéristique, chaque branche représente une décision basée sur cette caractéristique, et chaque feuille représente une prédiction.
Fonctionnement : Les données sont divisées de manière itérative en fonction des caractéristiques qui maximisent la pureté (ou minimisent l'impureté) des sous-groupes résultants, généralement mesurée par des critères tels que l'indice de Gini ou l'entropie.
Apprentissage : Les arbres de décision sont construits en parcourant l'ensemble de données et en choisissant la caractéristique qui offre la meilleure séparation entre les classes ou les valeurs de sortie.
Utilisation : Ils sont largement utilisés dans de nombreux domaines pour la classification et la prédiction, en raison de leur facilité d'interprétation, de leur capacité à gérer à la fois des données numériques et catégoriques, et de leur robustesse face à des données bruitées ou manquantes.
Avantages : Ils sont simples à comprendre et à interpréter, peuvent gérer des données non linéaires, et ne nécessitent pas de prétraitement intensif des données.
Inconvénients : Ils ont tendance à être sensibles au surajustement, en particulier avec des arbres profonds et des ensembles de données de petite taille, et peuvent ne pas être aussi performants que d'autres modèles sur des ensembles de données complexes.
Hyperparamètres : Des hyperparamètres tels que la profondeur maximale de l'arbre, le nombre minimum d'échantillons par feuille, etc., doivent être ajustés pour optimiser les performances et éviter le surajustement.
Évaluation : Les performances des arbres de décision peuvent être évaluées à l'aide de métriques appropriées pour la tâche, telles que l'accuracy, la précision, le rappel, etc.
Conseils : Il est souvent utile d'utiliser des techniques telles que la validation croisée ou l'utilisation d'ensembles d'arbres (comme les forêts aléatoires) pour améliorer la généralisation et réduire le surajustement.

Random Forest Regression

Spécialisation des arbres de décision.

Une construction à partir de multiples arbres . Chaque arbre est construit en sélectionnant aléatoirement un sous-ensemble d'observations (bootstrap) ainsi qu'un sous-ensemble aléatoire de caractéristiques à considérer pour chaque division de nœud.
Une fois que les arbres individuels sont construits, la prédiction finale est obtenue en agrégeant les prédictions de tous les arbres. En régression, cela peut être réalisé en prenant la moyenne des prédictions des différents arbres.

Réduction du surajustement : En utilisant une technique d'agrégation, le RFR réduit le risque de surajustement par rapport à un seul arbre de décision. Cela est dû au fait que les erreurs individuelles des arbres sont compensées par d'autres arbres dans l'ensemble.

Le RFR est connu pour sa stabilité et sa performance dans un large éventail de problèmes de régression. Il peut gérer efficacement les données bruitées ou manquantes et s'adapte bien à des ensembles de données complexes avec de nombreuses variables.

Tuning des Hyperparamètres : Tout comme pour les arbres de décision, le RFR comporte également des hyperparamètres tels que le nombre d'arbres dans la forêt, la profondeur maximale des arbres individuels, le nombre minimum d'échantillons par feuille, etc. L'optimisation de ces hyperparamètres peut être réalisée à l'aide de techniques telles que la validation croisée.

Évaluation de la Performance : La performance du RFR peut être évaluée à l'aide de diverses mesures telles que l'erreur quadratique moyenne (RMSE), l'erreur absolue moyenne (MAE), le coefficient de détermination (R^2), etc.

Modèle “à neurone”:

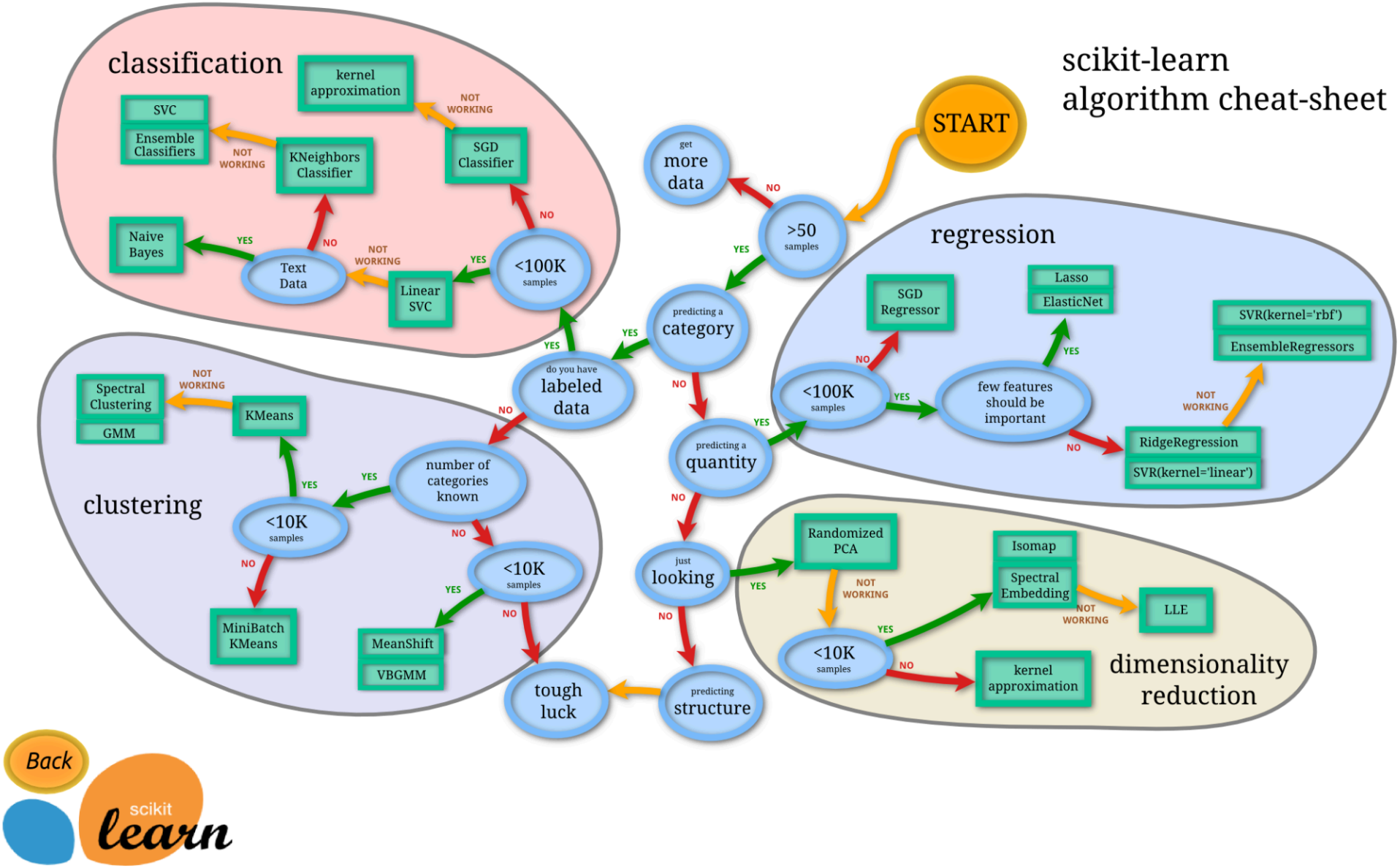
Réseaux de neurones (de préférence petits réseaux):

(plus adapté aux données non structurées : texte, image, son, etc) . (A voir si utile car les données seront mélangées…)
Principe : Les petits réseaux de neurones sont des réseaux de neurones artificiels (ANN) composés d'un petit nombre de couches et de neurones.
Structure : Ils peuvent avoir une architecture simple, souvent avec une ou deux couches cachées, et un nombre limité de neurones dans chaque couche.
Fonctionnement : Les données sont propagées à travers le réseau de neurones, où chaque neurone calcule une sortie en fonction des poids et des fonctions d'activation associés.
Apprentissage : Les petits réseaux de neurones peuvent être entraînés à l'aide d'algorithmes d'optimisation tels que la descente de gradient stochastique (SGD) ou des variantes comme Adam.
Utilisation : Ils sont souvent utilisés pour des tâches simples ou des ensembles de données de petite à moyenne taille, où des architectures plus complexes comme les réseaux de neurones profonds peuvent être excessifs ou entraîner un surajustement.
Avantages : Ils sont plus simples à entraîner et à comprendre, nécessitent moins de ressources computationnelles et sont moins susceptibles de souffrir de surajustement sur des ensembles de données de petite taille.
Inconvénients : Ils peuvent manquer de capacité à modéliser des relations complexes dans les données et peuvent ne pas être suffisamment performants pour des tâches plus complexes.
Hyperparamètres : Les hyperparamètres tels que le nombre de couches, le nombre de neurones par couche, les fonctions d'activation et le taux d'apprentissage doivent être ajustés pour obtenir de bonnes performances.
Évaluation : Les performances des petits réseaux de neurones peuvent être évaluées à l'aide de métriques de performance appropriées pour la tâche, telles que l'accuracy, la précision, le rappel, etc.

Conseils : Il est important de commencer avec une architecture simple et d'ajuster progressivement la complexité du modèle en fonction des performances sur les données de validation.

Suggestions de Thomas:
Gradient boosting, XG-Boost,Light GBM (on prend un modèle de base, et on l'optimise.

Modèle	**Commentaires**	**Attribution**
Régression Linéaire	Commencer par des modèles simples	Fahd (Personne A)
Lasso		Fahd(Personne A)
Ridge	Risque d'être moins pertinent dans notre cas parce qu'on a un nombre de features en entrée très limité	Fahd (Personne A)
SVR		Nouh(Personne B)
SGDRegressor		Adil (Personne C)
Random Forest		Hajar + Fahd (Personne D)
Gradient boosting, XG-Boost,Light GBM (on prend un modèle de base, et on l'optimise.		Hajar (Personne D)
Petits réseaux de neurones	Marche mieux avec beaucoup de données ou avec des données non structurées telles que l'image, le son et le texte, ce qui n'est pas notre cas.	Personne E (Surement Noura et on l'aide si besoin)


scikit-learn
algorithm cheat-sheet
START
>50 samples
NO
get more data
YES
predicting a category
NO
predicting a quantity
NO
just looking
NO
predicting structure
tough luck
do you have labeled data
YES
<100K samples
NO
Naive Bayes
YES
Text Data
NO
NOT WORKING
Linear SVC
YES
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
NOT WORKING
kernel approximation
NOT WORKING
SGD Classifier
NO
classification
YES
<100K samples
NO
SGD Regressor
Lasso
ElasticNet
YES
few features should be important
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
regression
YES
Randomized PCA
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
YES
<10K samples
NO
kernel approximation
dimensionality reduction
NO
number of categories known
YES
<10K samples
NO
MiniBatch KMeans
YES
<10K samples
NO
MeanShift
VBGMM
NOT WORKING
KMeans
Spectral Clustering
GMM
clustering
Back
scikit learn