RAPPORT

Projet 4A

Intelligence Artificielle

Auteurs :
Coutarel Allan, <allan_coutarel@etu.u-bourgogne.fr >
Devoucoux Maxime, <maxime_devoucoux@etu.u-bourgogne.fr >

ESIREM 4A Infotronique - ILC groupe TP2

2022 - 2023

SOMMAIRE

Introduction

Dans ce rapport de projet, nous allons explorer différentes notions et différents modèles d'intelligence artificielle afin de mieux comprendre leurs fonctionnements. Nous commencerons par une partie théorique sur le machine learning et ses différentes applications, puis nous passerons à l'analyse pratique de différents modèles en examinant leurs performances et leurs limites. L'objectif de ce projet est d'atteindre une meilleure compréhension du machine learning.

Dans la partie pratique, voici les 2 problèmes que nous traiterons afin de réaliser notre analyse :

- prédire si une personne va se faire vacciner contre la grippe H1N1
- prédire l'état de fonctionnement d'une pompe à eau.

Théorie

Les problèmes de régression

Un problème de régression en intelligence artificielle consiste à prédire une valeur numérique à partir de données d'entrée. Les modèles de régression sont souvent utilisés pour prévoir des tendances à long terme et pour faire des prévisions quantitatives. Pour résoudre un problème de régression, on peut utiliser différents types de modèles d'apprentissage automatique, tels que les réseaux de neurones, les arbres de décision, etc. Le but est de construire un modèle qui soit capable de prédire la sortie souhaitée avec une précision élevée en utilisant des données d'entrée. Nous pouvons donner un exemple de problème de régression :

Supposons que nous travaillons dans le contexte d'un restaurant et que nous voulons prédire le montant d'un pourboire laissé par le client en fonction de différents facteurs tels que le montant total de sa note, son sexe, si cette personne est fumeuse ou non, le jour de la semaine, le moment de la journée où le client consomme et enfin le nombre de personnes à table. Nous disposons de données de nombreux clients, comprenant ces différents facteurs et le montant du pourboire laissé par chacun d'entre eux. Nous pouvons utiliser ces données pour entraîner un modèle de régression qui sera capable de prédire le montant du pourboire laissé par le client en fonction de ces facteurs. Dans ce cas, le montant du pourboire est la variable à prédire et les différents facteurs sont les variables d'entrée. Les variables d'entrée font partie d'un "dataset".

Les problèmes de classification

Un problème de classification consiste à attribuer une étiquette ou une catégorie à un objet en fonction de ses caractéristiques. Pour résoudre un problème de classification, on utilise un algorithme de machine learning qui prend en entrée des données étiquetées, représentant des objets étiquetés, et apprend à prédire l'étiquette de ces objets en se basant sur les différentes caractéristiques qui les définissent. Ces caractéristiques correspondent à des colonnes de données dans le dataset. Une fois entraîné, l'algorithme peut être utilisé pour prédire l'étiquette de nouveaux objets pour lesquels l'étiquette n'est pas connue. De nombreux algorithmes de machine learning peuvent être utilisés pour résoudre des problèmes de classification, tels que l'algorithme de k plus proches voisins (KNN), la régression logistique, l'arbre de décision, etc. Nous pouvons donner un exemple de problème de classification :

Imaginons que nous voulions différencier des animaux sur une photographie afin de les classer dans différentes catégories. Nous disposons d'un échantillon d'images sur lesquelles apparaissent plusieurs animaux différents. En utilisant un algorithme de machine learning sur ces photos, nous pourrions entraîner notre modèle pour que celui-ci apprenne les différentes caractéristiques qui permettent de différencier les animaux (apprentissage non supervisé) ou nous pourrions entraîner notre modèle en lui indiquant quel animal se trouve sur la photo (apprentissage supervisé). Une fois le modèle suffisamment entraîné et validé, il sera capable de classer les animaux dans différentes catégories d'espèces, et cela même à partir d'une image qui ne lui a jamais été présentée.

À partir de cette définition, il convient naturellement que les problèmes que nous souhaitons traiter dans la partie pratique sont des problèmes de classification. En effet, prédire si une personne va se faire vacciner contre la grippe H1N1 revient à classer cette personne dans 2 catégories : oui, elle va se faire vacciner, ou non, elle ne se fera pas vacciner. Le problème ayant pour objectif de prédire l'état de fonctionnement d'une pompe à eau est lui aussi un problème de classification, cette fois-ci multiclasses : la pompe à eau est soit fonctionnelle, soit fonctionnelle mais nécessite des réparations ou soit non fonctionnelle.

L'apprentissage supervisé et non supervisé

L'apprentissage supervisé est une forme d'apprentissage dans laquelle un modèle d'IA est entraîné sur un ensemble de données d'entraînement qui comprend des données d'entrées (features) associées à des sorties connues (labels). L'objectif de l'apprentissage supervisé est d'entraîner un modèle afin qu'il soit en mesure de prédire des sorties à partir de données d'entrées inconnues au préalable. L'algorithme va utiliser les données d'entraînement pour apprendre à associer les entrées aux sorties et ainsi généraliser cette association à de nouvelles entrées.

Par opposition, l'apprentissage non supervisé est une forme d'apprentissage dans laquelle un modèle d'IA est entraîné sur des données sans labels (les sorties ne sont pas connues). Le modèle doit donc découvrir par lui-même les structures et les patterns présents dans les données, uniquement à partir de données d'entrées. L'apprentissage non supervisé est principalement utilisé dans "*Le classement de données, le calcul approximatif de la densité de distribution et la réduction des dimensions.*" (DataScientest.com).

Dans la partie pratique, nous nous intéresserons uniquement à l'apprentissage supervisé. La suite de ce rapport ne traite pas de l'apprentissage non supervisé.

L’entraînement d’un modèle

L'entraînement d’un modèle peut être vu comme de “l’apprentissage”. L’objectif de cette étape est de maximiser l’efficacité du modèle, c'est-à-dire de minimiser ses erreurs.

Pour cela, le modèle d’IA produit des prédictions à partir des sous-ensembles de données destinés à l'entraînement. Ces données constituent alors les entrées, appelées features. Le modèle d’IA transforme ensuite ces entrées en sorties, avec une méthode qui lui est propre. Cette méthode est caractéristique du modèle. Il est donc important de comprendre que les résultats obtenus seront différents en fonction du modèle sélectionné. La sortie prédite est ensuite comparée au label correspondant, c’est-à-dire à la sortie attendue qui était connue à l’avance et qui fait partie des sous-ensembles de données destinés à l'entraînement. Le label correspond donc à la sortie attendue pour un exemple d’objet des features. À partir de cette comparaison, les modèles d’IA, comme les réseaux de neurones notamment, peuvent ajuster leurs paramètres afin de s’améliorer dans leurs prédictions, c’est-à-dire d’obtenir des sorties prédites les plus proches possibles des sorties attendues.

Néanmoins, ce n’est pas le cas de tous les modèles. En effet, certains ne procèdent ni à des prédictions, ni à des ajustements de paramètres durant cette étape d’apprentissage. L'entraînement permet alors à ces modèles de constituer une structure interne de données, à partir des features et des labels d'entraînement, sur laquelle ils se basent pour généraliser des nouvelles entrées, c’est-à-dire pour effectuer des prédictions sur des nouvelles données.

L'évaluation d’un modèle

Après l'entraînement vient l’évaluation du modèle d’IA.

L’évaluation d’un modèle (ou validation, ou encore test) consiste à mesurer les performances de celui-ci. L’objectif de cette étape est d’évaluer le modèle sur des données qui n’ont pas servi à l’apprentissage afin de mesurer le taux d’erreur et la capacité du modèle à généraliser, c'est-à-dire la capacité à s’adapter à de nouvelles données, à prédire avec précision une sortie à partir de données que le modèle n’a pas apprises durant l'entraînement.

Pour cela, le modèle d’IA va réaliser des prédictions à partir des sous-ensembles de données destinés à l’évaluation. Nous retrouvons à nouveaux des features et des labels, cette fois-ci destinés à l'évaluation. Le modèle effectue des prédictions à

partir des features de test. Les sorties prédites vont être comparées aux labels de test. Cette “comparaison” se fait à travers l’utilisation de métriques. Les métriques mettent en jeu différents procédés mathématiques pour déterminer des mesures permettant de quantifier l’efficacité et les performances du modèle. Nous pouvons par exemple citer l’accuracy qui mesure le nombre de prédictions correctes parmi toutes les prédictions effectuées avec les features de test, ou encore l'erreur absolue moyenne qui *“est calculée en additionnant toutes les erreurs absolues et en les divisant par le nombre d'erreurs : ”* *(IBM)*

$$\text{Mean absolute errors} = \frac{\text{SUM} \ | \text{Yi} - \text{Xi} |}{\text{number of errors}}$$

Formule mathématique pour le calcul de la MAE

Il est important de noter que les métriques utilisées pour mesurer les performances d’un modèle sur un problème de régression ne sont généralement pas les mêmes pour un problème de classification. Le choix des métriques est à réfléchir en fonction du problème à traiter et des résultats recherchés.

Choix des métriques adaptées aux problèmes de classification

Avant de mettre en place nos modèles d’IA dans la partie pratique, il convient d’abord de réfléchir aux métriques adaptées à la classification, étant donné que nous traiterons 2 problèmes de classification comme énoncé précédemment. Nous évaluerons nos modèles à l’aide des métriques suivantes :

- La métrique *accuracy_score* calcule la précision du modèle, c’est-à-dire la proportion de prédictions correctes.

  Si $\hat{y}_i$ est la valeur prédite du i-ème échantillon et y_i est la valeur attendue, alors la précision du modèle sera calculée selon la formule :

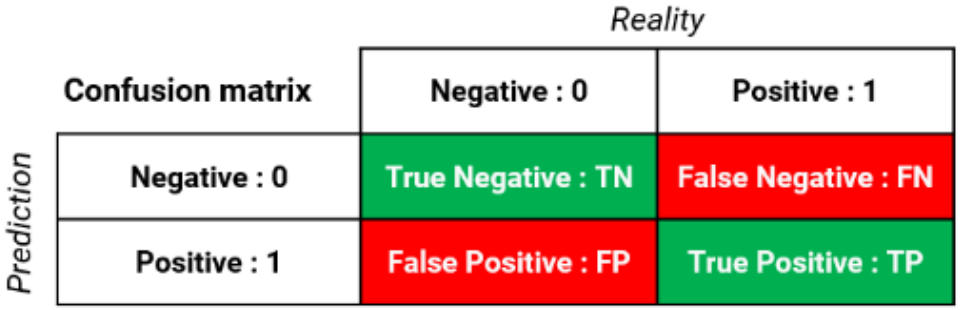
  $$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

  Cette métrique est très simple à interpréter, plus le résultat est proche de 1 et plus le modèle est performant. À l’inverse, plus le résultat est proche de 0 et plus le modèle produit des erreurs, des prédictions fausses.

Il est intéressant de noter que nous afficherons, dans le cas du train/test split seulement, l'*accuracy* calculée sur les données utilisées à l'entraînement et l'*accuracy* calculée sur les données de validation, ce qui peut donner une indication sur la capacité du modèle à généraliser, et donc cela permet de détecter un potentiel overfitting (sur-apprentissage).

- La métrique *balanced_accuracy_score* calcule la précision du modèle en prenant en considération les déséquilibres existants entre les classes de données du dataset, c'est-à-dire en tenant compte de la présence de classes sous-représentées ou sur-représentées dans les données. Nous pouvons retrouver l'explication détaillée de la méthode de calcul de la métrique dans la documentation de scikit-learn.
  Cette métrique est donc essentielle dans le cas d'un dataset déséquilibré puisque la métrique *accuracy_score* serait quant à elle peu représentative des performances réelles du modèle. Si le dataset est équilibré, les résultats de *accuracy_score* et *balanced_accuracy_score* devraient être proches.

- La métrique *confusion_matrix* calcule une matrice, qui *"dans le cas binaire (i.e. à deux classes, le cas le plus simple), est un tableau à 4 valeurs représentant les différentes combinaisons de valeurs réelles et valeurs prédites comme dans la figure ci-dessous : "*



Matrice de confusion (Source)

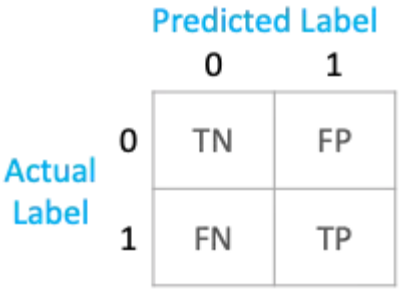
La matrice de confusion peut être affichée sous différentes formes avec sklearn comme des graphes de points, des courbes… Évidemment, la matrice de confusion est également fonctionnelle pour un problème de classification multi-classes.

Cette métrique nous permet donc de visualiser le nombre de prédictions correctes et fausses par classe. Cela est extrêmement utile pour de nombreux problèmes. En effet, il est préférable pour certains problèmes d'avoir un nombre de faux négatifs le plus faible possible (par exemple pour des problèmes médicaux, on cherche généralement à avoir le moins de faux négatifs possible). Pour d'autres problèmes, un équilibre entre faux positifs et

faux négatifs peut être recherché. La matrice de confusion va donc nous permettre de savoir si notre modèle a une tendance à prédire plus de cas étant faux positifs quand la prédiction est incorrecte, ou alors une tendance à prédire plus de cas faux négatifs.

En fonction des performances recherchées pour le modèle et après analyse de la matrice de confusion, il sera alors possible d'ajuster le seuil de classification afin d'obtenir un modèle qui prédira plus ou moins de cas faux positifs ou faux négatifs (pour les modèles qui utilisent un seuil de classification bien sûr). À noter que le seuil de classification intervient lors de la détermination d'une prédiction et permet de convertir le score de confiance, pouvant être vu comme la probabilité d'un objet d'appartenir à une classe, en véritable prédiction. Par exemple dans le cas binaire, le seuil de classification est par défaut défini à 0.5 avec scikit-learn. Cela signifie que si le score de confiance obtenue lors d'une prédiction pour une entrée donnée est 0.7, la prédiction sera 1. Si le score de confiance obtenue lors d'une prédiction pour une entrée donnée est 0.3, la prédiction sera 0.

Enfin il est important de noter que par défaut, la matrice de confusion obtenue en utilisant la métrique de scikit-learn suit la représentation suivante :



Matrice de confusion par défaut de scikit-learn

- Enfin, nous utiliserons les 2 métriques suivantes : *precision_score* et *recall_score*. *"Intuitivement, la precision_score est la capacité du classifieur à ne pas étiqueter comme positif un échantillon négatif, et le rappel est la capacité du classifieur à trouver tous les échantillons positifs"* (documentation scikit-learn). Ces métriques nous donnent donc une interprétation supplémentaire de la matrice de confusion. Si l'on considère d'ailleurs la matrice de confusion présentée précédemment, la précision peut être définie comme étant TP/(TP+FP) et le rappel comme étant TP/(TP+FN).

  La precision_score mesure donc la capacité du modèle à ne pas classer un cas négatif en cas positif (faux positif). Plus la precision_score est grande et plus cette capacité du modèle est élevée. Cela est donc un bon indicateur pour les problèmes nécessitant un faible taux de cas faux positifs.

Le rappel mesure quant à lui la capacité du modèle à ne pas classer un cas positif en cas négatif (faux négatif). Plus le rappel est grand et plus cette capacité du modèle est élevée. Cela est donc un bon indicateur pour les problèmes nécessitant un faible taux de cas faux négatifs, comme pour les problèmes médicaux énoncés en exemple dans le paragraphe sur la matrice de confusion.

Le découpage des données

Le découpage des données consiste à séparer les données utilisées, le dataset, afin de réaliser l'entraînement et l'évaluation du modèle d'IA à partir de différents sous-ensembles, certains dédiés à l'entraînement et d'autres à l'évaluation. L'objectif est de pouvoir mesurer la performance d'un modèle sur des données indépendantes pour évaluer sa capacité à généraliser ses prédictions à de nouvelles données. Cela permet de s'assurer que le modèle n'a pas simplement mémorisé les données d'entraînement (sur-apprentissage), mais qu'il a appris à généraliser ses prédictions à de nouvelles données. Il faudra également veiller à garder un nombre suffisant de données dédiées à l'entraînement afin que le modèle d'IA ne soit pas sous-entraîné, surtout si les données sont complexes. Le découpage des données va aussi permettre une meilleure répartition des données dans le dataset et peut également réduire l'impact des déséquilibres des classes sur les performances du modèle, nous en parlerons plus en détail par la suite.

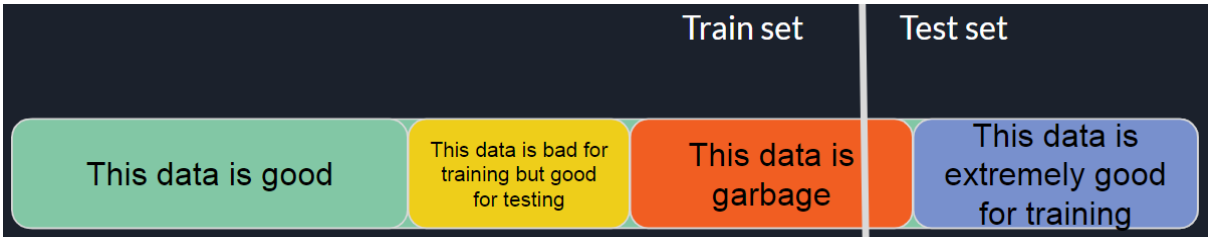
Nous allons détailler les deux méthodes de découpage de données les plus populaires : le train/test split et le k-fold.

La méthode train/test split a un fonctionnement simple. Nous commençons par séparer les données en deux sous-ensembles, un d'entraînement et un de test. Le modèle est ensuite logiquement entraîné à partir de l'ensemble d'entraînement et évalué avec l'ensemble de test. Ce découpage de données est utile quand le volume de données du dataset est plutôt faible, mais il n'est pas vraiment optimisé pour un gros volume de données. Les exemples suivants sont réalisés à partir du cours du module de Systèmes intelligents réalisés à l'Esirem par Kévin Naudin.

Si l'on considère un dataset avec la répartition suivante :

Exemple de répartition d'un dataset

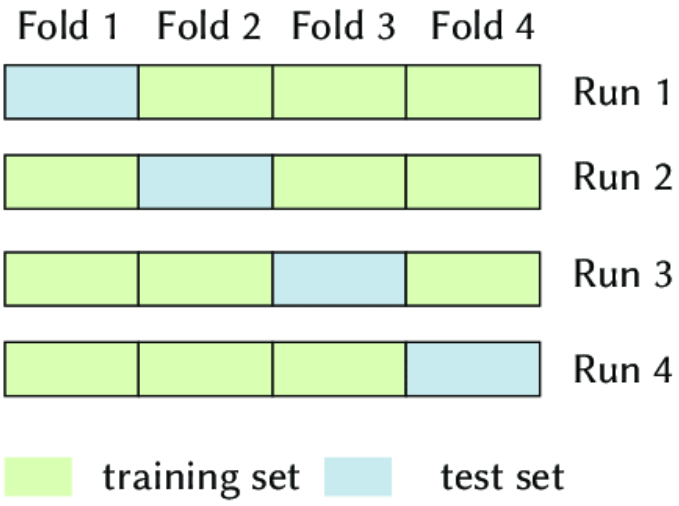
Voici un exemple de découpage que le train/test split pourrait donner :



Exemple de train/test split sur le dataset

Nous nous apercevons alors que la répartition des données n'est pas optimale. Pour un train/test split plus efficace, nous pouvons mélanger les données avant de les découper, même si le risque d'utiliser de "mauvaises" données aussi bien pour l'entraînement que la validation reste présent. La taille des sous-ensembles doit aussi être ajustée afin d'éviter le sous-apprentissage, le sur-apprentissage et les validations peu représentatives des performances du modèle. Cette méthode ne permet donc pas de tirer parti de l'ensemble du dataset pour l'entraînement et l'évaluation.

Dans la méthode k-fold, on découpe l'ensemble des données du dataset en plusieurs sous-ensembles, appelés folds, puis on entraîne et évalue un modèle sur chaque fold de manière itérative. On utilise à chaque itération un fold différent pour les données de test et les autres folds pour l'apprentissage :



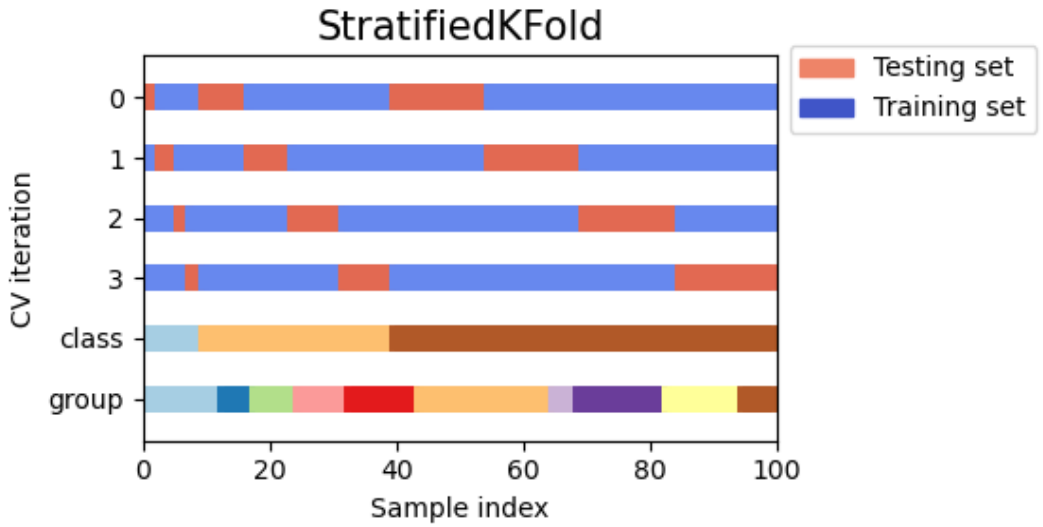
Explication schématique d'un k-fold à 4 folds (Source)

Nous pouvons, après avoir entraîné et évaluer le modèle sur tous les folds (le modèle repart de zéro à chaque itération), calculer la moyenne des scores de toutes les itérations. Le résultat nous donne alors la performance globale du modèle. Un k-fold permet donc d'utiliser toutes les données du dataset au moins une fois pour l'évaluation et pour l'apprentissage, ce qui n'est pas le cas avec le train/test split.

Nous pouvons également récupérer le modèle qui aura donné les meilleurs scores parmi tous les modèles de toutes les itérations afin de l'utiliser pour réaliser des prédictions sur des données de production par exemple.

Nous avons aussi découvert d'autres méthodes de découpage de données.

Le k-fold stratifié est une variante du découpage k-fold, qui permet de diviser le dataset en k sous-ensembles, tout comme le k-fold. La différence réside dans le fait que chaque fold d'un k-fold stratifié contient la même proportion des classes de données du dataset, aussi bien pour les features que pour les labels. Les principes d'entraînement et d'évaluation restent identique au k-fold "classique" :



Explication schématique d'un k-fold stratifié à 4 folds (Source)

L'utilisation d'un k-fold stratifié semble donc plus adapté pour des données déséquilibrées, c'est-à-dire quand une ou plusieurs classes du dataset sont sur-représentées ou sous-représentées par rapport aux autres.

Nous avons également découvert le découpage leave-p-out, consistant à découper le dataset en différents sous-ensembles afin de sélectionner p sous-ensembles destinés à l'évaluation. Les autres sous-ensembles servent à l'apprentissage. À chaque itération, les p sous-ensembles de validation sont différents. Il est important de noter que la documentation sklearn nous indique que cette méthode de découpage de données peut conduire à un coût élevé en temps de traitement et ressources : *"En raison du nombre élevé d'itérations qui croît de manière combinatoire avec le nombre d'échantillons, cette méthode de validation croisée peut être très coûteuse. Pour les grands ensembles de données, il faut privilégier KFold, StratifiedKFold ou ShuffleSplit."*

Nous avons alors découvert le découpage ShuflleSplit dans lequel le dataset est mélangé aléatoirement avant d'être divisé en plusieurs sous-ensembles pour la

validation croisée (plusieurs sous-ensembles destinés à la validation et les autres à l'entraînement). On effectue ensuite l'entraînement et l'évaluation du modèle d'IA. Ce processus est répété un certain nombre de fois. Cela permet de mélanger les classes de données et de toutes les parcourir durant l'apprentissage. C'est à l'utilisateur de spécifier le nombre de sous-ensembles, la proportion de données de test ainsi que le nombre d'itérations.

Dans la partie pratique, nous nous intéresserons au train/test split, au k-fold et au k-fold stratifié. Nous comparerons les résultats obtenus pour nos 2 datasets, et cela pour 4 modèles de classification différents.

Le choix d'un estimateur

L'estimateur est une fonction qui prend en entrée un ensemble de données et renvoie une valeur qui doit être la plus proche possible de la valeur réelle.

Un classifieur est un type d'estimateur dans lequel l'estimateur est "*utilisé pour associer des données en entrée à des catégories ou classes en sortie*", selon la définition de DataFranca.org.

La documentation scikit-learn nous propose une carte afin de choisir le bon estimateur en fonction du problème à traiter et des données utilisées.

Nous allons donc utiliser des classifieurs issus de scikit-learn pour construire nos modèles d'IA dans la partie pratique. Le premier que nous allons étudier est le classifieur K Nearest Neighbors (KNN) qui repose sur la méthode des k plus proches voisins.

La documentation de scikit-learn nous permet d'obtenir plus d'informations sur ce classifieur. Nous pouvons apprendre que "*la classification basée sur les voisins est un type d'apprentissage basé sur les instances ou apprentissage non généralisant, c'est-à-dire qu'il ne tente pas de créer un modèle interne général. La classification est calculée à partir d'un vote à la majorité simple des proches voisins de chaque point. Un point par requête est attribué à la classe de données qui a le plus de membres parmi les plus proches voisins du point.*"

Les schémas suivants résument parfaitement le fonctionnement du classifieur KNN :

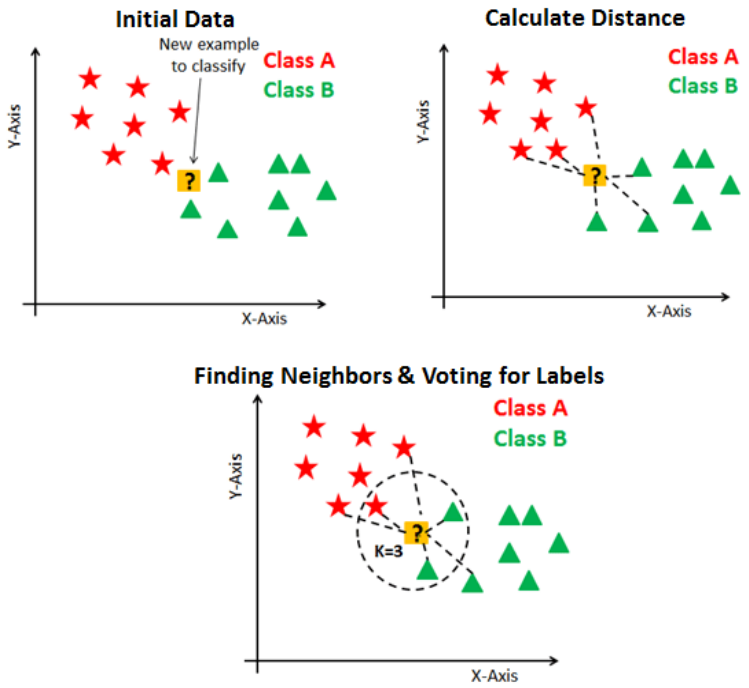


Schéma résumant le fonctionnement du classifieur *KNN (Source)*

Le classifieur KNN permet un apprentissage basé sur k (donné par l'utilisateur) voisins les plus proches de chaque point de requête. Si la valeur de k est trop petite, l'algorithme est sensible au bruit des données et une grande valeur de k enlève le bruit mais rend la classification moins précise. Le bruit de données se caractérise par des données manquantes, aberrantes ou incorrectes. Il faut de plus déterminer une mesure de distance qui correspond à la proximité des observations. Cette distance peut se calculer à l'aide de plusieurs fonctions comme nous l'explique l'article Introduction à l'algorithme K Nearst Neighbors (K-NN) : "*la distance euclidienne, la distance de Manhattan, la distance de Minkowski, celle de Jaccard, la distance de Hamming…etc. On choisit la fonction de distance en fonction des types de données qu'on manipule".*

Nous allons également utiliser la classification SVC (Support Vector Classification). Cette méthode de classification repose sur des machines à vecteur de support (SVM). L'article Machine à vecteurs de support (SVM) définition et cas d'usage du site Journaldunet.fr nous explique leur fonctionnement : *"Le principe des SVM consiste à ramener un problème de classification ou de discrimination à un hyperplan (feature space) dans lequel les données sont séparées en plusieurs classes dont la frontière est la plus éloignée possible des points de données (ou "marge maximale"). D'où l'autre nom attribué aux SVM : les séparateurs à vaste marge. Le concept de frontière implique que les données soient linéairement séparables. Pour y parvenir, les support vector machines font appel à des noyaux,*

c'est-à-dire des fonctions mathématiques permettant de projeter et séparer les données dans l'espace vectoriel, les "vecteurs de support" étant les données les plus proches de la frontière. C'est la frontière la plus éloignée de tous les points d'entraînement qui est optimale, et qui présente donc la meilleure capacité de généralisation."

Les SVM peuvent être utilisés dans des problèmes de régression et de classification. Nous pouvons noter qu'il existe deux types de SVM, les linéaires et les non linéaires. Les linéaires cherchent à créer une séparation des données en deux classes en partant de l'hypothèse que les données soient séparables par une ligne (ou hyperplan) comme l'explique le cours sur les SVM linéaires de Karl Pearson. Ce schéma issu de la page Wikipédia des SVM permet d'illustrer la séparation des données :

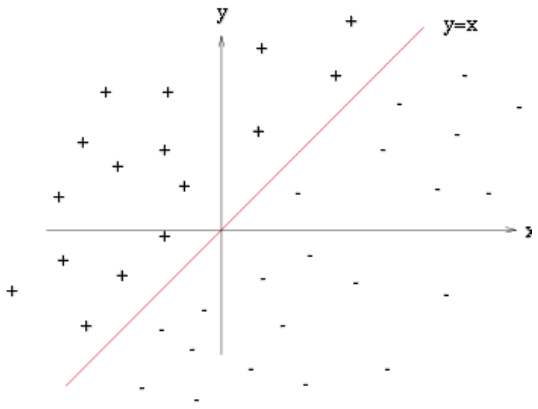


Illustration de la séparation linéaire des données.

Les deux classes de données différentes sont symbolisées par des formes (+ et -) et la droite tente de les séparer au mieux.

Les non linéaires utilisent des techniques de transformation pour mettre les données dans un espace de dimension plus grande à l'aide d'une fonction noyau pour effectuer une séparation linéaire. Cette figure issue de la page Wikipédia des SVM nous permet de mieux comprendre le fonctionnement de la fonction noyau :

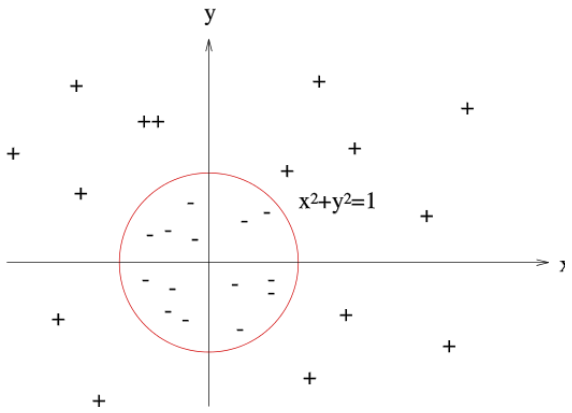


Illustration de données non séparables linéairement.

Exemple simple de transformation : le problème n'est pas linéairement séparable en coordonnées cartésiennes, par contre en coordonnées polaires, le problème devient linéaire

La documentation scikit-learn de SVC nous permet de connaître la manière d'utiliser ces différents classifieurs, à savoir SVC, LinearSVC ou NuSVC (classifieur similaire à SVC comportant un paramètre permettant de choisir le nombres de vecteurs de support).

La prochaine classification que nous allons utiliser est la classification naïve bayésienne. Cet algorithme de classification est basé sur le théorème de Bayes. Il est dit naïf car les hypothèses ont une forte indépendance. Cet algorithme repose en effet sur l'hypothèse que les caractéristiques d'une classe sont indépendantes entre elles. L'exemple de la page Wikipédia des classifications naïve bayésienne permet de comprendre cette notion d'indépendance : *Un fruit peut être considéré comme une pomme s'il est rouge, arrondi, et fait une dizaine de centimètres. Même si ces caractéristiques sont liées dans la réalité, un classifieur bayésien naïf déterminera que le fruit est une pomme en considérant indépendamment ces caractéristiques de couleur, de forme et de taille.* Le classifieur GaussianNB utilise également des probabilités pour prédire quelle est la classe des données analysées. Ces probabilités sont obtenues à partir d'une distribution gaussienne, ou loi normale, que nous avons pu étudier en cours. La loi normale est caractérisée par une courbe en forme de cloche centrée autour de la moyenne et dont la largeur est définie par la variance. La densité de probabilité de la loi normale est définie par :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \, \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Densité de probabilité de la loi normale

Où μ est l'espérance et σ l'écart type.

Enfin nous allons étudier le classifieur de régression logistique. Ce classifieur permet de prédire une variable binaire en prenant en paramètre des variables indépendantes. Cette variable binaire permet de définir la probabilité de classification dans une catégorie. Les données indépendantes et la probabilité d'observation est définie par une fonction de régression logistique. La documentation de scikit-learn nous donne l'expression de la fonction de régression logistique :

$$\hat{p}(X_i) = \operatorname{expit}(X_i w + w_0) = \frac{1}{1 + \exp(-X_i w - w_0)}.$$

Avec Xi un vecteur des variables indépendantes. Il reste ensuite à définir les conditions qui déterminent l'attribution à une catégorie en fonction de la probabilité calculée.

Mise en pratique

Le code de ce projet est accessible via ce lien.

Pour rappel, nous allons dans cette partie pratique traiter 2 problèmes de classification, un visant à prédire si une personne va se vacciner ou non contre la grippe H1N1, l'autre visant à prédire l'état de fonctionnement d'une pompe à eau (fonctionnelle, fonctionnelle mais nécessite des réparations ou non fonctionnelle). Comme évoqué précédemment, nous allons nous intéresser à 4 modèles de classification différents (KNN, SVC, Naïve Bayes et Régression logistique).

À noter que le dataset des pompes à eau comporte 59 400 lignes et 40 colonnes pour 4317 exemples de pompes à eau fonctionnelles mais nécessitant des réparations, 22 824 pompes à eau non fonctionnelles et 32 259 pompes à eau fonctionnelles. Le dataset des vaccins compte quant à lui 26 707 lignes et 38 colonnes pour 21 033 exemples de personnes non favorables à se faire vacciner et 5674 favorables. Nous remarquons que les datasets sont déséquilibrés, nous verrons en pratique comment cela impacte les performances des modèles.

Le traitement des données

Avant de mettre en place les modèles, il est nécessaire de s'intéresser à nos datasets. En effet, un dataset est rarement directement prêt à être utilisé dans un modèle de machine learning pour l'entraînement ou l'évaluation.

La première étape consiste à remplacer toutes les variables catégoriques par des nombres. En effet, l'apprentissage, l'évaluation, les prédictions, etc, reposent essentiellement sur des calculs mathématiques, il faut donc des nombres pour chaque valeur d'entrée.

Deuxièmement, il est très commun de retrouver des datasets ayant des données manquantes (NaN) dans certaines catégories. Si tel est le cas, il est alors nécessaire de supprimer les lignes du dataset contenant un manque d'informations puisque le modèle ne pourra pas traiter des entrées vides.

Ensuite, il convient de déterminer si le dataset est équilibré ou non afin de choisir un découpage de données approprié, entre autres. Cette étape peut être réalisée en comptant le nombre d'occurence de chaque valeur unique pour toutes les catégories du dataset (features et labels), en utilisant des graphiques comme des histogrammes ou des métriques spécifiques. Comme énoncé précédemment, nos datasets sont déséquilibrés (comptage des classes réalisé avec la méthode

value_counts de pandas). Il serait préférable d'équilibrer les datasets en réalisant des opérations de sous-échantillonnage ou sur-échantillonnage des classes pour maximiser les performances des modèles, mais nous ne réaliserons pas cette étape. En effet, nous testerons directement plusieurs méthodes de découpage de données pour comprendre leurs fonctionnements en pratique et leurs différences de performances sur des données déséquilibrées, puis nous sélectionnerons celle qui aura offert les meilleurs résultats pour réaliser l'étude de nos modèles.

Enfin, il est généralement préférable de supprimer les lignes des données de test contenant des valeurs aberrantes dans une ou plusieurs catégories. Pour les données d'entraînement, ces valeurs représentent des biais, qui peuvent tout de même être utiles pour l'entraînement. Néanmoins pour l'ensemble de test, supprimer les valeurs aberrantes permet de s'assurer que l'évaluation est réalisée sur des données propres et non biaisées, plus représentatives de la réalité.

Il peut être aussi nécessaire de normaliser les features, c'est-à-dire *" redimensionner les variables numériques pour qu'elles soient comparables sur une échelle commune."* (DataScientest). Certains modèles comme le SVC ont besoin de données normalisées afin de réaliser leurs procédés mathématiques.

Explications de code

Le code se découpe en plusieurs fonctions. Il est commenté afin de faciliter sa compréhension.

Nous avons implémenté des fonctions qui permettent de déclarer nos modèles et d'effectuer leur entraînement (*getKNeighborsClassifier, getSVClassifier, getGaussianNBClassifier, getLogisticRegressionClassifier*) ainsi qu'une fonction permettant de réaliser l'évaluation d'un modèle, afficher les résultats des métriques que nous avons sélectionnées et retourner un tableau contenant les valeurs prédites pour l'évaluation et les valeurs attendues (*evaluateModel*).

Nous retrouvons ensuite 2 fonctions, *get_Results_Dataframe* qui permet de concaténer les valeurs prédites et les valeurs attendues dans un dataset (méthode utilisée pour simplifier l'affichage du résultat retourné par l'évaluation) et *printResults* qui permet d'afficher un dataset en fonction du nombre de lignes que nous souhaitons afficher.

Nous créons ensuite, après la définition de ces méthodes, des blocs de codes afin de diviser différentes étapes :

- Importation du dataset

- Remplacement des variables catégoriques par des nombres (une simple analyse visuelle du dataset a été réalisée au préalable pour identifier les catégories contenant des chaînes de caractères)
- Suppression des lignes du dataset ayant des catégories manquantes
- Séparation des features et des labels du dataset
- Découpage des données
- Suppression des valeurs aberrantes dans les données de test
- Normalisation des features
- Manipulation des modèles : entraînement, évaluation, affichage des résultats

Dans le cas d'un découpage de données k-fold ou k-fold stratifié, nous utilisons *cross_validate* de scikit-learn qui permet de réaliser la validation croisée très simplement et permet d'obtenir les résultats des métriques que nous avons choisies. À noter que *cross_validate* ne permet pas de supprimer les valeurs aberrantes dans l'ensemble des données de test.

Le choix du découpage des données

L'objectif de cette partie est de comparer les résultats obtenus pour différentes méthodes de découpage de données : train/test split, k-fold et k-fold stratifié. Pour ce faire, nous allons comparer les résultats des métriques obtenus pour chaque classifieur et pour chaque méthode de découpage.

Il est nécessaire de réaliser l'entraînement et la validation des modèles dans des conditions les plus proches possibles d'une méthode découpage de données à l'autre. Cela permet de s'assurer que les performances des modèles seront uniquement influencées par les méthodes de découpage de données. Les modèles seront tous définis par défaut, les hyper-paramètres resteront donc inchangés pour chaque méthode de découpage. Enfin, pour pouvoir comparer de manière équitable les métriques, il est important d'avoir la même proportion de données dédiées à l'apprentissage pour chaque méthode de découpage de données, idem pour les données dédiées à la validation. Pour le train/test split, cette proportion est définie par le paramètre *test_size,* correspondant au pourcentage de données du dataset à utiliser pour la validation. Pour les k-folds, la proportion de données utilisées pour la validation sera égale à 1/k (cf. partie Théorie).

<u>Pour le dataset des vaccins :</u>

Nous décidons d'utiliser 80% du dataset pour l'entraînement et 20% pour l'évaluation. Pour le train/test split, nous définirons donc *test_size* égale à 0.2. Pour respecter cette proportion avec le k-fold et le k-fold stratifié, nous définirons le nombre de folds à 5.

Pour le dataset des pompes à eau :

Le dataset étant volumineux, le temps d'exécution des méthodes de cross validation est relativement important. Nous décidons donc d'utiliser seulement 3 folds pour le k-fold et le k-fold stratifié. Cela signifie que ⅔ du dataset seront dédiés à l'entraînement et ⅓ à l'évaluation. Pour respecter cette proportion avec le train/test split, nous définirons *test_size* égale à 0.3 pour le dataset des pompes à eau.

Lors de nos essais, nous nous sommes aperçu que le modèle SVC est très long à entraîner. Nous supprimerons donc volontairement, et de manière aléatoire, 50% des données du dataset des pompes à eau (uniquement pour le modèle SVC).

Train/test split :

Avant de réaliser le split des données, nous choisissons de les mélanger aléatoirement afin d'améliorer leur répartition dans le dataset. Le *random_state* correspond à la graine du générateur pseudo-aléatoire servant à mélanger aléatoirement les données avant de les découper. Nous le fixons à 42, en référence à la réponse à la grande question sur la vie, l'univers et le reste (c'est une valeur généralement utilisée, même dans la documentation scikit-learn).

Dataset vaccins

```
KNN CLASSIFIER :

Accuracy of the model (with training data) : 0.8556090336912254
Accuracy : 0.8133828996282528
Balanced accuracy : 0.6974994383830289
Confusion matrix : [[941  82]
 [169 153]]
Precision : 0.6510638297872341
Recall : 0.4751552795031056

SVC CLASSIFIER :

Accuracy of the model (with training data) : 0.8416327286190299
Accuracy : 0.8475836431226765
Balanced accuracy : 0.7529674626448819
Confusion matrix : [[956  67]
 [138 184]]
Precision : 0.7330677290836654
Recall : 0.5714285714285714

GNB CLASSIFIER :

Accuracy of the model (with training data) : 0.7742502776749353
Accuracy : 0.779182156133829
Balanced accuracy : 0.7271543323436731
Confusion matrix : [[846 177]
 [120 202]]
Precision : 0.5329815303430079
Recall : 0.6273291925465838

LR CLASSIFIER :

Accuracy of the model (with training data) : 0.8414476119955572
Accuracy : 0.8408921933085501
Balanced accuracy : 0.7432484532765038
Confusion matrix : [[952  71]
 [143 179]]
Precision : 0.716
Recall : 0.5559006211180124
```

Dataset pompes à eau

```
KNN CLASSIFIER :

Accuracy of the model (with training data) : 0.7609187109187109
Accuracy : 0.7426775253607658
Balanced accuracy : 0.5703772521417362
Confusion matrix : [[7199  111  684]
 [ 688  225  187]
 [1876   56 2972]]
Precision : 0.6949031829522947
Recall : 0.5703772521417362

SVC CLASSIFIER :

Accuracy of the model (with training data) : 0.6375180375180375
Accuracy : 0.634804972138877
Balanced accuracy : 0.41521960189554896
Confusion matrix : [[7185    0  809]
 [ 982    0  118]
 [3203    0 1701]]
Precision : 0.4263954651148782
Recall : 0.41521960189554896

GNB CLASSIFIER :

Accuracy of the model (with training data) : 0.5441077441077441
Accuracy : 0.5292184597799686
Balanced accuracy : 0.50839902406908
Confusion matrix : [[4306 1947 1741]
 [ 406  502  192]
 [1544  760 2600]]
Precision : 0.47276861512353857
Recall : 0.50839902406908

LR CLASSIFIER :

Accuracy of the model (with training data) : 0.6404521404521405
Accuracy : 0.6362337476782397
Balanced accuracy : 0.43035731257388116
Confusion matrix : [[6759    4 1231]
 [ 941   11  148]
 [2765    3 2136]]
Precision : 0.6215532176630575
Recall : 0.43035731257388116
```

K-fold :

Nous mélangeons à nouveau les données avant leur découpage afin de nous assurer que les sous-ensembles résultants contiennent des éléments de toutes les classes. Evidemment, cela reste aléatoire et ne garantit rien. De plus, cela ne prend pas en compte le déséquilibre des classes. C'est donc pour ces diverses raisons que nous aurons recours au k-fold stratifié par la suite.

Dataset vaccins

Classifieur KNN :

```
Accuracy : [0.80643967 0.80451685
0.79340985 0.80007405 0.79526101]
Mean Accuracy : 0.7999402858441825
Balanced accuracy : [0.73953533
0.73412831 0.73479683 0.73763398
0.7308517 ]
Mean Balanced accuracy :
0.7353892316771897
Precision : [0.69692308 0.70253165
0.7012987  0.6958457  0.70757576]
Mean Precision : 0.7008349757393065
Recall : [0.58151476 0.56632653
0.58064516 0.58333333 0.56469166]
Mean Recall : 0.5753022888684064
```

Classifieur SVC :

```
Accuracy : [0.80643967 0.80451685
0.79340985 0.80007405 0.79526101]
Mean Accuracy : 0.7999402858441825
Balanced accuracy : [0.73953533
0.73412831 0.73479683 0.73763398
0.7308517 ]
Mean Balanced accuracy :
0.7353892316771897
Precision : [0.69692308 0.70253165
0.7012987  0.6958457  0.70757576]
Mean Precision : 0.7008349757393065
Recall : [0.58151476 0.56632653
0.58064516 0.58333333 0.56469166]
Mean Recall : 0.5753022888684064
```

Classifieur GNB :

```
Accuracy : [0.77831236 0.77823029
0.77304702 0.77378749 0.77082562]
Mean Accuracy : 0.7748405544345639
Balanced accuracy : [0.76368593
0.75782606 0.75888 0.75761841
0.75681669]
Mean Balanced accuracy :
0.758965635550126
Precision : [0.59414226 0.59978425
0.61382114 0.60041623 0.60569106]
```

Dataset pompes à eau

Classifieur KNN :

```
Accuracy : [0.75045455 0.74787879
0.7470202 ]
Mean Accuracy : 0.7484511784511785
Balanced accuracy : [0.60005258
0.60452316 0.60178515]
Mean Balanced accuracy :
0.6021202964523646
Precision : [0.75045455 0.74787879
0.7470202 ]
Mean Precision : 0.7484511784511785
Recall : [0.75045455 0.74787879
0.7470202 ]
Mean Recall : 0.7484511784511785
```

Classifieur SVC :

```
Accuracy : [0.63949495 0.64515152
0.63818182]
Mean Accuracy : 0.6409427609427609
Balanced accuracy : [0.4316604
0.43721806 0.43332878]
Mean Balanced accuracy :
0.43406907826509467
Precision : [0.63949495 0.64515152
0.63818182]
Mean Precision : 0.6409427609427609
Recall : [0.63949495 0.64515152
0.63818182]
Mean Recall : 0.6409427609427609
```

Classifieur GNB :

```
Accuracy : [0.5410101  0.55737374
0.42641414]
Mean Accuracy : 0.5082659932659933
Balanced accuracy : [0.51972741
0.5166385  0.50244302]
Mean Balanced accuracy :
0.5129363107406023
Precision : [0.5410101  0.55737374
0.42641414]
Mean Precision : 0.5082659932659933
```

```
Mean Precision : 0.6027709875792789
Recall : [0.72913992 0.70918367
0.72162485 0.71766169 0.72067715]
Mean Recall : 0.7196574569917866
```

Classifieur LR :

```
Accuracy : [0.85418209 0.84228064
0.83561644 0.84116994 0.83672714]
Mean Accuracy : 0.8419952475314816
Balanced accuracy : [0.80209637
0.78824834 0.78940281 0.79412276
0.79113504]
Mean Balanced accuracy :
0.7930010628233763
Precision : [0.78603269 0.7647929
0.77103448 0.76223776 0.76510989]
Mean Precision : 0.769841544792956
Recall : [0.67907574 0.65943878
0.66786141 0.6778607  0.67351874]
Mean Recall : 0.6715510724785754
```

```
Recall : [0.5410101  0.55737374
0.42641414]
Mean Recall : 0.5082659932659933
```

Classifieur LR :

```
Accuracy : [0.64494949 0.64035354
0.63893939]
Mean Accuracy : 0.6414141414141413
Balanced accuracy : [0.44710777
0.44570636 0.44404247]
Mean Balanced accuracy :
0.4456188681784871
Precision : [0.64494949 0.64035354
0.63893939]
Mean Precision : 0.6414141414141413
Recall : [0.64494949 0.64035354
0.63893939]
Mean Recall : 0.6414141414141413
```

Stratified K-fold :

Pour le découpage en k-folds stratifiés, notre approche est identique à celle du k-fold. Néanmoins, le découpage des données tient désormais compte du déséquilibre des classes. Voyons si cela se reflète sur les performances des modèles.

Dataset vaccins

Classifieur KNN :

```
Accuracy : [0.79348631 0.80673825
0.78711588 0.79711218 0.80932988]
Mean Accuracy : 0.7987564986074462
Balanced accuracy : [0.7243532
0.73785232 0.72351166 0.73170515
0.74718569]
Mean Balanced accuracy :
0.7329216038918581
Precision : [0.69362364 0.72539683
0.66960352 0.69545455 0.71879699]
Mean Precision : 0.7005751053505879
Recall : [0.55266419 0.56699752
0.56575682 0.56947891 0.59305211]
Mean Recall : 0.5695899096306819
```

Classifieur SVC :

```
Accuracy : [0.83604737 0.85042577
0.83783784 0.84450204 0.84116994]
Mean Accuracy : 0.8419965903463668
Balanced accuracy : [0.78422151
0.80071495 0.78604006 0.79506734
```

Dataset pompes à eau

Classifieur KNN :

```
Accuracy : [0.75191919 0.74267677
0.74676768]
Mean Accuracy : 0.7471212121212121
Balanced accuracy : [0.59967273
0.60327531 0.60239759]
Mean Balanced accuracy :
0.6017818731613791
Precision : [0.75191919 0.74267677
0.74676768]
Mean Precision : 0.7471212121212121
Recall : [0.75191919 0.74267677
0.74676768]
Mean Recall : 0.7471212121212121
```

Classifieur SVC :

```
Accuracy : [0.64545455 0.63656566
0.63919192]
Mean Accuracy : 0.6404040404040404
Balanced accuracy : [0.44025904
0.43347171 0.43516542]
```

```
0.79411865]
Mean Balanced accuracy :
0.7920325003434442
Precision : [0.76224784 0.79130435
0.76589595 0.7765043  0.76363636]
Mean Precision : 0.771917760366132
Recall : [0.65551425 0.67741935
0.65756824 0.67245658 0.67741935]
Mean Recall : 0.6680755547765981
```

Classifieur GNB :

```
Accuracy : [0.77720207 0.78119215
0.76897445 0.77378749 0.77156609]
Mean Accuracy : 0.7745444500501638
Balanced accuracy : [0.76147823
0.76420841 0.75264933 0.75465342
0.76055671]
Mean Balanced accuracy :
0.7587092189529507
Precision : [0.60665973 0.61327713
0.5942029  0.6031746  0.59516616]
Mean Precision : 0.6024961056281783
Recall : [0.72242875 0.72208437
0.71215881 0.70719603 0.73325062]
Mean Recall : 0.7194237149507566
```

Classifieur LR :

```
Accuracy : [0.83197631 0.85042577
0.83820807 0.8430211  0.8430211 ]
Mean Accuracy : 0.8413304719501042
Balanced accuracy : [0.77989622
0.80000196 0.78559092 0.79365543
0.79722039]
Mean Balanced accuracy :
0.7912729851515055
Precision : [0.75322812 0.79300292
0.76855895 0.77285714 0.76527778]
Mean Precision : 0.7705849817136761
Recall : [0.65055762 0.67493797
0.65508685 0.67121588 0.68362283]
Mean Recall : 0.6670842288782091
```

```
Mean Balanced accuracy :
0.4362987232345117
Precision : [0.64545455 0.63656566
0.63919192]
Mean Precision : 0.6404040404040404
Recall : [0.64545455 0.63656566
0.63919192]
Mean Recall : 0.6404040404040404
```

Classifieur GNB :

```
Accuracy : [0.535      0.45964646
0.53888889]
Mean Accuracy : 0.5111784511784512
Balanced accuracy : [0.52057768
0.50503581 0.51724262]
Mean Balanced accuracy :
0.51428536847884
Precision : [0.535      0.45964646
0.53888889]
Mean Precision : 0.5111784511784512
Recall : [0.535      0.45964646
0.53888889]
Mean Recall : 0.5111784511784512
```

Classifieur LR :

```
Accuracy : [0.64530303 0.64141414
0.6379798 ]
Mean Accuracy : 0.6415656565656566
Balanced accuracy : [0.44779315
0.44715198 0.44238003]
Mean Balanced accuracy :
0.445775055474539
Precision : [0.64530303 0.64141414
0.6379798 ]
Mean Precision : 0.6415656565656566
Recall : [0.64530303 0.64141414
0.6379798 ]
Mean Recall : 0.6415656565656566
```

<u>Analyse des résultats :</u>

Toutes les métriques ne nous intéressent pas pour cette analyse. Nous allons nous concentrer principalement sur l'*accuracy* et la *balanced_accuracy*. Si nous considérons ces métriques sur les données des vaccins, nous nous apercevons qu'elles restent relativement proches sur les 3 types de découpages de données. Un bon indicateur concernant le déséquilibre des données est de comparer l'écart entre l'*accuracy* et la *balanced_accuracy*, et de regarder l'évolution de cette écart entre les méthodes de découpage de données. Nous pouvons alors supposer que le dataset des vaccins est légèrement déséquilibré.

Concernant les données des pompes à eau, nous constatons le même effet avec une *accuracy* et *balanced_accuracy* légèrement meilleure pour le découpage *Stratified KFold*. Néanmoins, l'écart entre l'*accuracy* et la *balanced_accuracy* est important (pratiquement 0.2 pour chaque modèle). Nous pouvons donc en déduire que le dataset des pompes à eau est déséquilibré. Nous remarquons néanmoins que les modèles généralisent pratiquement de la même manière, peu importe le découpage de données utilisé, la *balanced_accuracy* n'est pas vraiment meilleure avec un k-fold stratifié. Nous supposons qu'il faudrait ajuster les paramètres des modèles et le seuil de classification (si possible) par exemple pour de meilleurs résultats, ou utiliser d'autres méthodes pour lutter contre les déséquilibres de classes.

Nous avons remarqué néanmoins que les ressources et la durée nécessaires pour mettre en place des méthodes de validation croisée sont très importantes. N'étant pas dans une démarche de recherche de la performance optimale d'un modèle mais plutôt dans une démarche d'analyse et de compréhension, nous décidons d'utiliser un train/test split avec une *test_size* à 0.3 pour la suite de notre projet afin d'obtenir un sous-ensemble d'entrainement et un sous-ensemble de test pour chacun des datasets. Nous allons désormais, à partir de ces sous-ensembles, entraîner et évaluer tous les modèles afin de comparer leurs résultats.

Nearest Neighbors Classifier

Avec les hyper-paramètres configurés par défaut, voici les résultats que nous obtenons :

```
KNN CLASSIFIER :

Accuracy of the model (with training data) : 0.8537127141950497
Accuracy : 0.808532249873032
Balanced accuracy : 0.7019666720770438
Confusion matrix : [[1348  116]
 [ 261  244]]
Precision : 0.6777777777777778
Recall : 0.48316831683168315
```

Scores du modèle KNN sur le dataset des vaccins, n_neighbors = 5

```
KNN CLASSIFIER :

Accuracy of the model (with training data) : 0.8201539201539202
Accuracy : 0.7477496785255037
Balanced accuracy : 0.5977570793028777
Confusion matrix : [[6940  231  823]
 [ 620  292  188]
 [1539  130 3235]]
Precision : 0.6572605489726712
Recall : 0.5977570793028777
```

Scores du modèle KNN sur le dataset des pompes à eau, n_neighbors = 5

Nous remarquons dans les 2 cas un probable léger overfitting (*accuracy* supérieure sur les données d'entraînement que sur les données de validation). Nous remarquons également le déséquilibre des données par l'écart entre l'*accuracy* et la *balanced_accuracy.* Ce qui est intéressant à noter est la capacité du modèle à avoir un score de précision assez élevé (*precision et accuracy*), c'est-à-dire que le modèle montre une bonne capacité à ne pas classer un cas négatif en cas positif.

Voyons désormais l'influence de *n_neighbors,* à savoir le nombre de voisins les plus proches à considérer dans la recherche des plus proches voisins. Pour commencer, nous allons passer cette valeur à 2.

```
KNN CLASSIFIER :

Accuracy of the model (with training data) : 0.8726464988364714
Accuracy : 0.7846622651091925
Balanced accuracy : 0.6268949845804253
Confusion matrix : [[1392   72]
 [ 352  153]]
Precision : 0.68
Recall : 0.30297029702970296
```

Scores du modèle KNN sur le dataset des vaccins, n_neighbors = 2

```
KNN CLASSIFIER :

Accuracy of the model (with training data) : 0.8590909090909091
Accuracy : 0.7301043006143735
Balanced accuracy : 0.5699748083963264
Confusion matrix : [[7260  300  434]
 [ 712  281  107]
 [1979  246 2679]]
Precision : 0.6337816135206019
Recall : 0.5699748083963264
```

Scores du modèle KNN sur le dataset des pompes à eau, n_neighbors = 2

Le changement le plus remarquable est la dégradation du rappel du modèle lorsque le nombre de voisins diminue. Le modèle va donc produire un ratio de faux négatifs plus importants. Nous pouvons également constater une légère diminution de l'*accuracy* et de la *balanced accuracy* du modèle. Nous pouvons donc en conclure qu'un nombre de voisins trop faible fait diminuer la performance globale du modèle.

Il est néanmoins intéressant de remarquer que l'*accuracy* calculée à partir des données d'entraînement a quant à elle augmenté. Cela provient du fait que les données d'entraînement constituent la structure interne sur laquelle se base le modèle KNN pour réaliser des prédictions. Lors de l'évaluation des données d'entraînement, le modèle va donc chercher les plus proches voisins de points qu'il connaît déjà. Ainsi, plus le nombre de voisins les plus proches tend vers 1 et plus la

prédiction se rapproche du point lui-même, ce qui explique que l'*accuracy* augmente avec les données d'entraînement quand *n_neighbors* diminue.

Maintenant, avec un nombre de voisins les plus proches plus important, *n_neighbors* = 20 :

```
KNN CLASSIFIER :

Accuracy of the model (with training data) : 0.8314998942246669
Accuracy : 0.8237684103605891
Balanced accuracy : 0.6959983498349835
Confusion matrix : [[1403   61]
 [ 286  219]]
Precision : 0.7821428571428571
Recall : 0.43366336633663366
```

Scores du modèle KNN sur le dataset des vaccins, n_neighbors = 20

```
KNN CLASSIFIER :

Accuracy of the model (with training data) : 0.7609187109187109
Accuracy : 0.7426775253607658
Balanced accuracy : 0.5703772521417362
Confusion matrix : [[7199  111  684]
 [ 688  225  187]
 [1876   56 2972]]
Precision : 0.6949031829522947
Recall : 0.5703772521417362
```

Scores du modèle KNN sur le dataset des pompes à eau, n_neighbors = 20

Avec un nombre de voisins les plus proches plus important, nous observons une nette augmentation de la précision, tout en gardant une bonne *accuracy,* relativement similaire à *n_neighbors = 5*. Le classifieur KNN semble donc intéressant pour les problèmes nécessitant une bonne précision (un faible nombre de faux positifs). Nous remarquons également que plus le nombre de voisins est faible et plus le modèle semble sensible aux bruits des données.

Support Vector Classifier

Avec les hyper-paramètres configurés par défaut, voici les résultats que nous obtenons :

```
SVC CLASSIFIER :

Accuracy of the model (with training data) : 0.8422889782102814
Accuracy : 0.8481462671406805
Balanced accuracy : 0.7655744467889412
Confusion matrix : [[1369   95]
 [ 204  301]]
Precision : 0.76010101010101
Recall : 0.596039603960396
```

Scores du modèle SVC kernel Linear sur le dataset des vaccins

```
SVC CLASSIFIER :

Accuracy of the model (with training data) : 0.6375180375180375
Accuracy : 0.634804972138877
Balanced accuracy : 0.41521960189554896
Confusion matrix : [[7185    0  809]
 [ 982    0  118]
 [3203    0 1701]]
Precision : 0.4263954651148782
Recall : 0.41521960189554896
```

Scores du modèle SVC kernel Linear sur le dataset des pompes à eau

Nous constatons qu'il n'y a presque aucun overfitting avec le classifieur SVC. Nous pouvons confirmer que les données sont déséquilibrées car nous observons encore un décalage entre *accuracy* et *balanced accuracy*. Nous pouvons noter que dans le cas d'un dataset assez déséquilibré comme le dataset des pompes à eau, le modèle a tendance à générer des faux positifs et à générer des faux négatifs.

Ce modèle semble assez sensible aux données très déséquilibrées. En effet, si nous comparons les résultats du modèle SVC et du modèle KNN, le modèle SVC est nettement moins performant avec le dataset des pompes à eau (très déséquilibré) alors que les résultats restent relativement proches pour le dataset des vaccins. De plus, nous remarquons une colonne de 0 dans la matrice de confusion du dataset des pompes à eau, qui signifie que la classe minoritaire (pompes à eau fonctionnelles mais nécessitant des réparations) n'a pas suffisamment d'exemples pour que le modèle puisse la séparer efficacement des autres classes.

Gaussian Naive Bayes Classifier

Avec les hyper-paramètres configurés par défaut, voici les résultats que nous obtenons :

```
GNB CLASSIFIER :

Accuracy of the model (with training data) : 0.7743812143008251
Accuracy : 0.7836465210766886
Balanced accuracy : 0.7442514743277606
Confusion matrix : [[1208  256]
 [ 170  335]]
Precision : 0.5668358714043993
Recall : 0.6633663366336634
```

Scores du modèle GNB sur le dataset des vaccins

```
GNB CLASSIFIER :

Accuracy of the model (with training data) : 0.5441077441077441
Accuracy : 0.5292184597799686
Balanced accuracy : 0.50839902406908
Confusion matrix : [[4306 1947 1741]
 [ 406  502  192]
 [1544  760 2600]]
Precision : 0.47276861512353857
Recall : 0.50839902406908
```

Scores du modèle GNB sur le dataset des pompes à eau

Nous pouvons remarquer une différence significative de performances entre le dataset des vaccins et le dataset des pompes à eau. Nous pouvons supposer que ce classifieur est sensible aux données déséquilibrées dans notre cas. De plus, nous remarquons un overfitting moins présent. Pour le dataset des vaccins, ce modèle offre de bon résultat avec un rappel plus élevé que la précision, et plus élevé que les modèles précédents, ce qui peut en faire un bon choix si nous souhaitons un nombre de faux négatifs faible.

Pour le dataset des pompes à eau, nous observons par la matrice de confusion que le modèle GNB est capable de généraliser des données de la classe minoritaire, contrairement au modèle SVC.

Logistic Regression Classifier

Avec les hyper-paramètres configurés par défaut, voici les résultats que nous obtenons :

```
LR CLASSIFIER :

Accuracy of the model (with training data) : 0.8416543262111276
Accuracy : 0.845606907059421
Balanced accuracy : 0.7619210896499486
Confusion matrix : [[1367   97]
 [ 207  298]]
Precision : 0.7544303797468355
Recall : 0.5900990099009901
```

Scores du modèle Logistic Regression sur le dataset des vaccins

```
LR CLASSIFIER :

Accuracy of the model (with training data) : 0.6404521404521405
Accuracy : 0.6362337476782397
Balanced accuracy : 0.43035731257388116
Confusion matrix : [[6759    4 1231]
 [ 941   11  148]
 [2765    3 2136]]
Precision : 0.6215532176630575
Recall : 0.43035731257388116
```

Scores du modèle Logistic Regression sur le dataset des pompes à eau

Comme le classifieur précédent, nous observons une bonne capacité du modèle à généraliser les données. Il semble éprouver des difficultés avec les forts déséquilibres de classes, tout comme le modèle SVC, puisque nous remarquons dans la matrice de confusion du dataset des pompes à eau que la classe minoritaire est très mal représentée.

Malgré des datasets déséquilibrés et des hyper-paramètres gérés par défaut, les modèles offrent des performances correctes dans l'ensemble. Il serait tout à fait envisageable de sélectionner un modèle en particulier pour chaque dataset afin de l'utiliser en production. Evidemment, il faudrait alors corriger le déséquilibre des datasets et adapter les hyper-paramètres afin d'obtenir des performances maximales. Le dataset des vaccins pourrait être utilisé avec les 4 modèles de classifications étudiés, le choix dépendrait entre autres de l'importance du nombre de faux négatifs ou faux positifs dans le problème. Il faudrait donc étudier précisément les métriques *precision* et *recall*. Pour le dataset des pompes à eau, le modèle le plus adapté semble être le modèle KNN, lequel a donné les meilleurs résultats parmi tous les modèles que nous avons comparés. Il serait néanmoins intéressant de tester tous les modèles après avoir traiter le déséquilibre du dataset afin d'observer les performances qu'ils peuvent offrir sur des données plus "propres".

Conclusion

En conclusion, ce projet nous a permis d'explorer différents modèles de classification et de les mettre en pratique pour résoudre des problèmes concrets. Nous avons pu constater que chacun de ces modèles possède ses propres avantages et limites, et qu'il est important de bien comprendre les données sur lesquelles nous travaillons pour choisir le modèle le plus approprié. Ce projet nous a également permis de mieux comprendre les concepts fondamentaux du machine learning tels que l'apprentissage supervisé, la validation croisée, les métriques, etc. L'ajustement des paramètres d'un modèle est un exercice complexe mais fondamental pour maximiser ses performances.

Pour améliorer ce projet, une piste intéressante serait d'explorer l'influence des hyper-paramètres de tous les modèles que nous avons utilisés afin d'améliorer notre compréhension des résultats. Nous pourrions également aborder les problèmes dans une démarche de recherche de performance optimale suivant une problématique donnée, ce qui est indispensable dans une optique d'utilisation du modèle avec des données de production. Il serait alors intéressant de découvrir les techniques de validation automatique des hyper-paramètres.

Bibliographie

- *sklearn.svm.SVC*, scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- *sklearn.neighbors.KNeighborsClassifier*, scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.htm
- *Choosing the right estimator*, scikit-learn. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
- *Support Vector Machines with Scikit-learn Tutorial*, DataCamp. https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
- Machine Learnia. (2019, 15 novembre). *PYTHON SKLEARN - MODEL SELECTION*. YouTube. https://www.youtube.com/watch?v=w_bLGK4Pteo
- Machine Learnia. (2019, 02 novembre). *PYTHON SKLEARN : KNN, LinearRegression et SUPERVISED LEARNING*. YouTube. https://www.youtube.com/watch?v=P6kSc3qVph0
- Apprentissage supervisé VS apprentissage non supervisé, Mobiskill. https://mobiskill.fr/blog/conseils-emploi-tech/apprentissage-supervise-vs-apprentissage-non-supervise/
- Apprentissage non supervisé : Principe et utilisation, DataScientest. https://datascientest.com/apprentissage-non-supervise
- Udacity. (2015, 23 février). *K-Fold Cross Validation - Intro to Machine Learning*. YouTube. https://www.youtube.com/watch?v=TIgfjmp-4BA
- Définition du classificateur, DataFranca.org https://datafranca.org/wiki/Classificateur
- Documentation Sickit Learn de KNeighborsClassifier. https://scikit-learn.org/stable/modules/neighbors.html#classification
- Article Machine à vecteurs de support (SVM) définition et cas d'usage, Journal Du Net. https://www.journaldunet.fr/web-tech/guide-de-l-intelligence-artificielle/1501879-machine-a-vecteurs-de-support-svm-definition-et-cas-d-usage/
- Documentation Scikit learn de SVC. https://scikit-learn.org/stable/modules/svm.html#svm-classification
- Page Wikipédia des classifications naïve bayésiennes. https://fr.wikipedia.org/wiki/Classification_na%C3%AFve_bay%C3%A9sienne
- Documentation Scikit learn de la régression logistique. https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
- *KNN Classification Tutorial using Scikit-learn, DataCamp.* https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn
- *3.3. Metrics and scoring : quantifying the quality of predictions*. (s. d.). scikit-learn. https://scikit-learn.org/stable/modules/model_evaluation.html
- Gallic, E. (s. d.). *Python pour les économistes*. https://egallic.fr/Enseignement/Python/pandas.html

- Mr.Mint, Introduction à l'algorithme K Nearst Neighbors (K-NN). https://mrmint.fr/introduction-k-nearest-neighbors
- *How to Implement K fold Cross-Validation in Scikit-Learn*. (s. d.). Engineering Education (EngEd) Program | Section. https://www.section.io/engineering-education/how-to-implement-k-fold-cross-validation/
- Cours sur les SVM Linéaires, Karl Pearson, CNAM. https://cedric.cnam.fr/vertigo/cours/ml2/coursSVMLineaires.html
- Page Wikipédia des SVM. https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support
- *sklearn.naive_bayes.GaussianNB*. (s. d.). scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
- *sklearn.linear_model.LogisticRegression*. (s. d.). scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression
- R, A. (2022, 24 mai). *Hello Daniel, qu'est-ce que la normalisation des données ?* Formation Data Science | DataScientest.com. https://datascientest.com/normalisation-des-donnees
- *Visualizing cross-validation behavior in*. (s. d.). scikit-learn. https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html