nexialog CONSULTING

Comment rendre opérationnel un modèle de machine learning ?

Marvin SUZANNE

THINK SMART ACT DIFFERENT

TABLE DES MATIÈRES

INTRODUCTION

Le MLOps peut se définir comme un cadre de travail spécifique aux projets de Machine Learning.

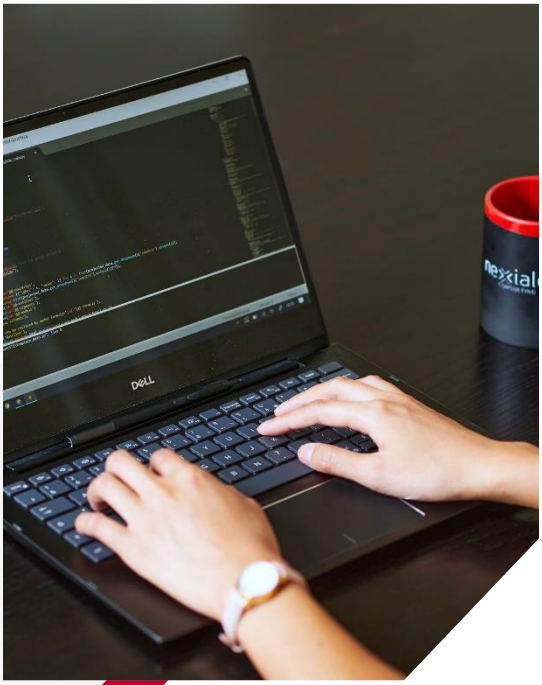
Ce framework donne les principes permettant de cadrer, développer et mettre en production un modèle. Une attention particulière est accordée au monitoring, qui vise à garantir des performances similaires en phase de développement et en phase de production.

Cette note aborde en Partie I les motivations propres au MLOps, en mettant en avant les différences entre développement et production, puis les différences entre logiciel classique et logiciel basé sur des techniques de machine learning. Bien que récent, le MLOps s'inspire très largement des principes bien connus du DevOps.

En Partie II, les différentes étapes constituants la vie d'un projet ML seront présentées, indépendamment du cadre MLOps.

Trois grandes phases seront exposées : Data Engineering, Model Engineering, Déploiement.

En Partie III, l'apport du MLOps à chacun de ces blocs fonctionnels sera souligné et illustré à partir d'outils open source.



I. Quelles sont les motivations du MLOps ?

1. L'étape de déploiement

Développer et déployer : des enjeux différents

Même si un POC (Proof Of Concept) peut être très utile pour illustrer une méthodologie ou un concept, un modèle de machine learning générera davantage de valeur s'il est mis en production.

Tout d'abord que signifie mettre un modèle en production ?
Cela consiste à rendre un modèle accessible à un tiers, dans un environnement dit de production. On parle d'environnement de production par opposition à l'environnement de développement, dans lequel des bugs peuvent survenir et des correctifs peuvent être apportés. En production, le système doit être stable afin de créer de la valeur.

Cette entité tiers ou utilisateur final, peut être un client qui va directement utiliser les prédictions, ou une autre brique du système d'information, comme un logiciel. Beaucoup d'entreprises sous-estiment la complexité et les challenges induits par le déploiement de systèmes de machine learning (ML), et concentrent leurs efforts sur la phase de développement.

Les applications à base de ML sont différentes des logiciels standards dans la mesure où leur performance repose essentiellement sur de la donnée. Comme les données évoluent en permanence, les modèles en production doivent être monitorés, re-entraînés et redéployés, afin de garantir un niveau de performance similaire en production et en phase de développement.

Les données peuvent également revêtir un caractère confidentiel, selon leur nature, et doivent donc être utilisées avec précaution. Il est par exemple possible d'avoir recours à des techniques d'anonymisation, comme générer des données synthétiques, ayant les mêmes propriétés statistiques que les données initiales.

Déployer un modèle ML est chronophage

Le data processing a longtemps été la partie la plus chronophage du métier de data scientist, mais avec le développement de solutions innovantes pour adresser ce problème, le goulot d'étranglement se situe désormais au niveau du déploiement.

La **Figure 1** ci-contre est tirée de l'étude "2021 Entreprise Trends in Machine Learning", partagée par la start-up américaine Algorithmia.

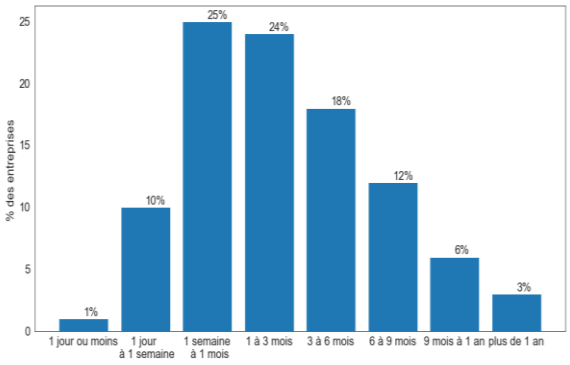


Figure 1 – Vitesse de déploiement des entreprises

Elle montre que pour 64% des entreprises, il faut plus d'un mois pour déployer un modèle une fois développé. Toujours d'après ce rapport, la plupart des data scientists passent au moins 25 % de leur temps à déployer des modèles.

Il apparaît donc crucial de mettre en place des procédures d'industrialisation des déploiements, afin de gagner en productivité.

Cette vitesse de déploiement peut s'expliquer par plusieurs facteurs.

Les challenges rencontrés par les entreprises sont de natures variées et peuvent représenter un frein à la mise en production des modèles.

La **Figure 2,** tirée d'une étude d'Algorithmia (2021), permet de remarquer que 56 % du panel interrogé remontent des difficultés au niveau IT, et près de 50 % ont des problèmes liés à l'intégration des modèles et à leur compatibilité avec des entités préexistantes.

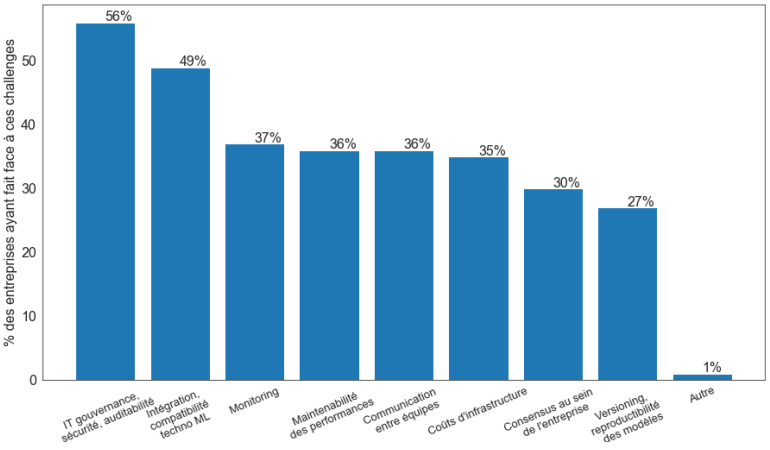
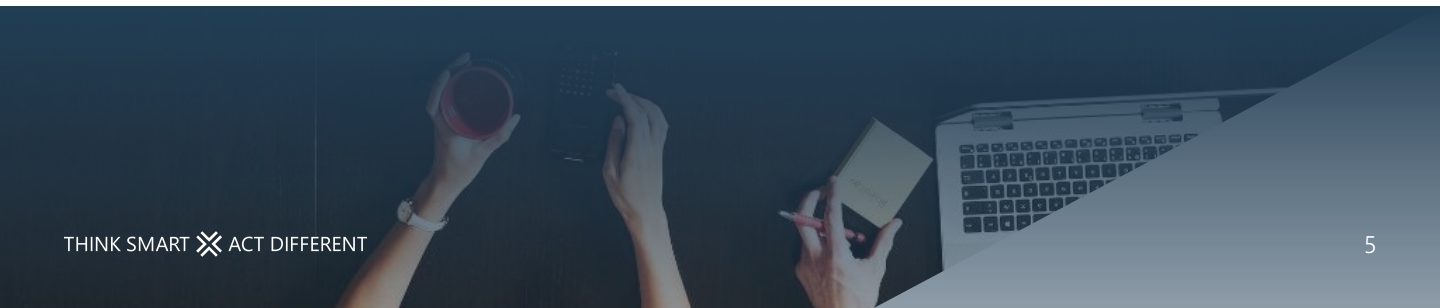


Figure 2 – Challenges rencontrés par les entreprises



2. Un environnement dynamique

Comme évoqué précédemment, une différence notable entre les logiciels traditionnels et ceux basés sur le machine learning est la dépendance envers la donnée. Cela signifie que de manière intrinsèque tout changement dans la distribution des données utilisées pour générer des prédictions peut nécessiter de réentraîner les modèles.

Afin de réagir à temps et d'assurer un bon niveau de performance en production, il est important de comprendre comment ces changements peuvent impacter le modèle.

Il est par exemple possible d'étudier des cas d'usages similaires, afin d'anticiper de possibles changements dans les données.

Une méthode complémentaire consiste à monitorer le modèle en production et de réentraîner le modèle en cas de dégradation de performance.

Prenons l'exemple d'un modèle de prédiction des consommations électriques, qui aurait été entraîné avant 2020. Ce type de système est notamment utilisé par EDF, afin d'adapter l'offre d'énergie à la consommation prévue.

Ce modèle aurait nécessité un réentraînement, dû aux changements de comportement induits durant la pandémie Covid-19, comme la baisse de l'activité économique.

En **Figure 3**, un décalage significatif vers la gauche de l'histogramme des consommations électriques de 2020 par rapport à celui de 2019 peut être observé, traduisant une nette baisse (-19.3 % en moyenne). Cette chute de consommation peut s'interpréter comme résultant d'une diminution de la production dans diverses secteurs économiques. La Figure 3 montre également une légère hausse de la consommation électrique entre avril 2018 et avril 2019 (+4.6 % en moyenne), qui peut être reliée à la hausse tendancielle du PIB (+1.5 %). Dans les deux cas, une analyse visuelle rapide montre un changement de distribution, ce changement étant bien sûr plus conséquent lorsqu'on compare 2019 et 2020. Ces histogrammes représentent une mesure qualitative d'une rupture de tendance ou drift. Mais existe-t-il des indicateurs plus quantitatifs ?

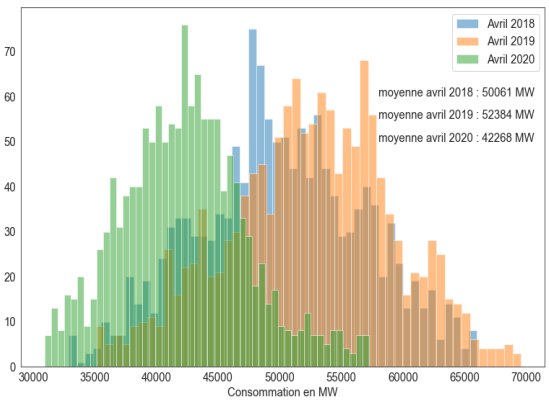


Figure 3 – Histogrammes des consommations électriques (source RTE)

Il existe plusieurs techniques pour quantifier le drift sur un ensemble de données. Une manière de procéder est d'entraîner un modèle de classification pour discriminer les données provenant de l'ensemble A et B.

Ici, deux modèles sont entraînés. Dans le premier les données provenant de 2019 sont labellisées à 0 et celle de 2020 à 1. Pour le second, les données 2019 sont à nouveau labellisées 0 et celles de 2018 sont à 1.

Si le modèle parvient à bien séparer les données, les deux ensembles peuvent être considérés comme fortement différents. Il est à noter que pour fonctionner, cette méthode utilise des données récoltées *a posteriori*. Certaines méthodes permettent de détecter le drift online, comme expliqué dans l'article Bifet *et al.* (2016) .

Il faut pouvoir quantifier le degré de séparabilité du modèle entraîné. Une métrique adaptée est l'AUC (Area Under the Curve), qui peut s'interpréter comme la probabilité qu'un individu de la classe positive ait un score supérieur à un de la classe négative. Plus l'AUC est proche de 1 et meilleur est le pouvoir de séparabilité du modèle, et donc plus les deux ensembles de données peuvent être considérés comme différents.

Un avantage supplémentaire d'utiliser un modèle ML pour quantifier le drift est que l'on dispose pour de nombreuses technologies de modèles d'un feature importance, donnant l'importance de chaque variable prédictive.

Une démarche similaire peut être envisagée en considérant les coefficients d'une régression logistique.

En **Figure 4**, sont données les courbes ROC et l'AUC pour les deux modèles entraînés. Ces métriques ont été calculées sur un échantillon de test correspondant à 20% des données avril 2018 & 2019, puis 20% des données avril 2019 & 2020. Pour plus de simplicité, les variables retenues pour la prédiction sont la consommation électrique, les jours, heures et minutes d'échantillonnage.

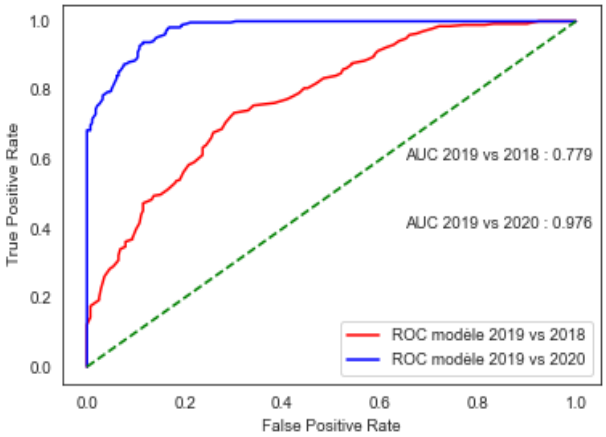


Figure 4 – Courbes ROC et AUC

La ligne en vert correspond à un modèle de type pile ou face, qui déterminerait au hasard à quelle classe appartient un individu. Plus la courbe suit le coin supérieur gauche, meilleur est le modèle. L'AUC est l'aire au-dessous de cette courbe. Il apparaît donc que le modèle 2019 vs 2020 sépare quasi parfaitement les données, confirmant l'étude qualitative par histogramme. Les données ont également driftés entre 2018 et 2019, mais dans une moindre mesure.

La Figure 5 nous permet de mesurer l'importance relative des variables dans la prédiction du drift. Sans surprise, la consommation électrique arrive loin devant.

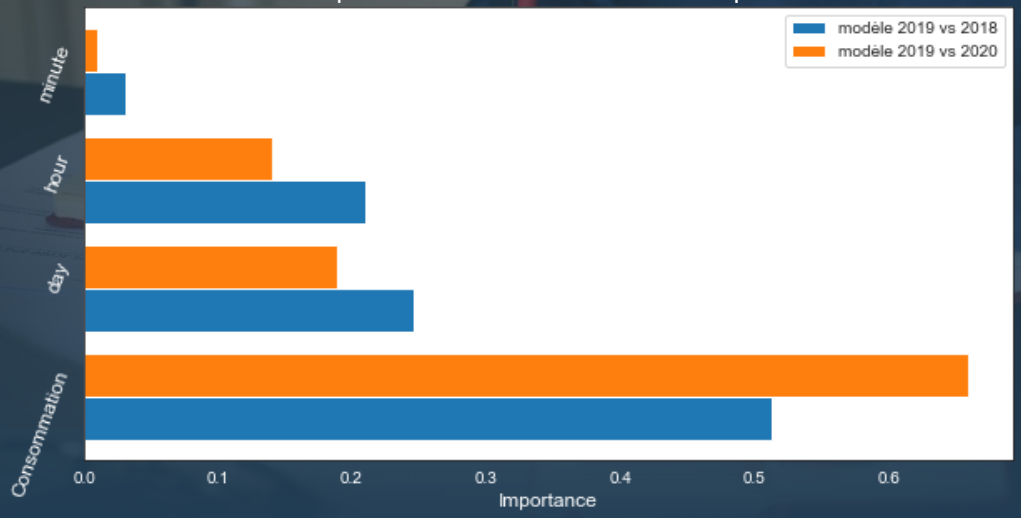


Figure 5 – Feature Importance

3. MLOps : les principes DevOps appliqués au ML

Le MLOps tient son origine dans les principes du DevOps, qui est un cadre méthodologique visant à uniformiser les pratiques de développement et d'opérationnalisation des systèmes informatiques. Avant cela, des équipes de développement s'occupaient de construire le logiciel et les équipes opérationnelles avaient pour rôle de le mettre en production et de l'y maintenir. Ce mode de fonctionnement engendre d'importantes disparités entre les deux phases de vie d'un logiciel, ainsi qu'un cloisonnement des équipes. La célèbre phrase de Werner Volgels en 2006, CTO d'Amazon "You build it, you run it", met en avant la nécessité d'avoir une procédure unifiée et donc des ingénieurs ayant la double compétence. Aujourd'hui, le mode de pensée DevOps est très bien implanté au sein des équipes IT, et le monde du machine learning a naturellement voulu s'en inspirer. Il repose notamment sur le développement agile, c'est-à-dire la division d'un projet en sous-projets, appelés sprints, permettant une meilleure organisation des tâches.

Les objectifs de chaque sprint sont prédéfinis, mesurables. Cela permet de livrer au plus tôt et de manière constante.

Cette méthode va de paire avec le principe CI/CD, pour Continuous Integration / Continuous Delivery, visant à automatiser les étapes de création, test et déploiement d'applications. Pour plus d'informations, il est possible de consulter l'article de Atlassian sur les principes DevOps.

Le MLOps peut donc être considéré comme l'application du DevOps au Machine Learning : il s'agit de combler le fossé entre le développement de solution ML et leur mise en production. Ce cadre de travail nécessite donc des compétences à la fois de statisticien et d'ingénieur logiciel. La **Figure 6** résume les différentes étapes qui ont mené au MLOps

Dans la suite de cette note, les étapes nécessaires à la construction d'un modèle de ML seront exposées.

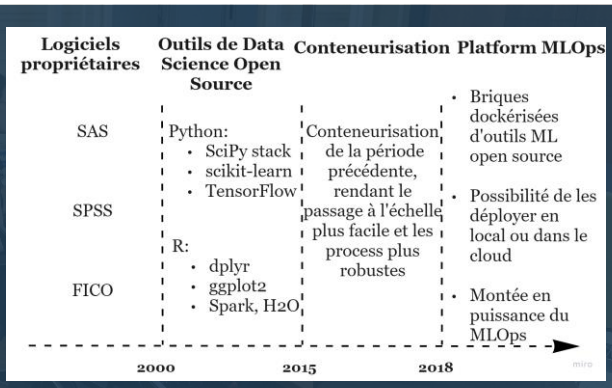


Figure 6 – Evolution du MLOps

II. Etapes de construction d'un modèle de ML

Cette partie donne une vision fonctionnelle des différentes tâches constituants un projet ML.

2.1 Data Engineering

Data ingestion

Dans la majorité des cas d'usage, les données proviennent de sources variées. Pour construire un modèle qui prédit si un client va résilier son contrat, il est possible d'utiliser des données statiques comme sa date de naissance, son adresse, nom, prénom provenant d'un CRM (Customer Relationship Management), des données dynamiques comme les achats réalisés, les moyens de paiement, également des données propres aux produits.

Toutes ces données sont généralement stockées dans des bases de données différentes, parfois même au sein de services différents.

Une première étape est donc de réunir en un même lieu, sous leur forme brute, c'est-à-dire sans avoir opéré aucune transformation. Cette forme de stockage porte habituellement le nom de Data Lake.

Une fois ces différentes données réunies en un même lieu, il est possible de constituer le "cube de données", en joignant selon des clés qui identifient les différentes entités (identifiant client, identifiant produit, date …). Ces clés dépendent bien sûr du cas d'usage et de la nature des données. Comme illustré en **Figure 7**, le dataset qui sera utilisé pour résoudre le cas d'usage est obtenu à l'issue de cette phase.

L'étape suivante consiste à réaliser l'analyse exploratoire des données ou EDA en anglais.

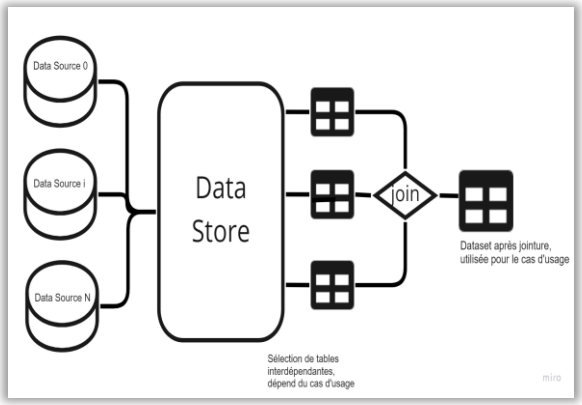


Figure 7 – Data Ingestion

Validation, Séparation, Exploration

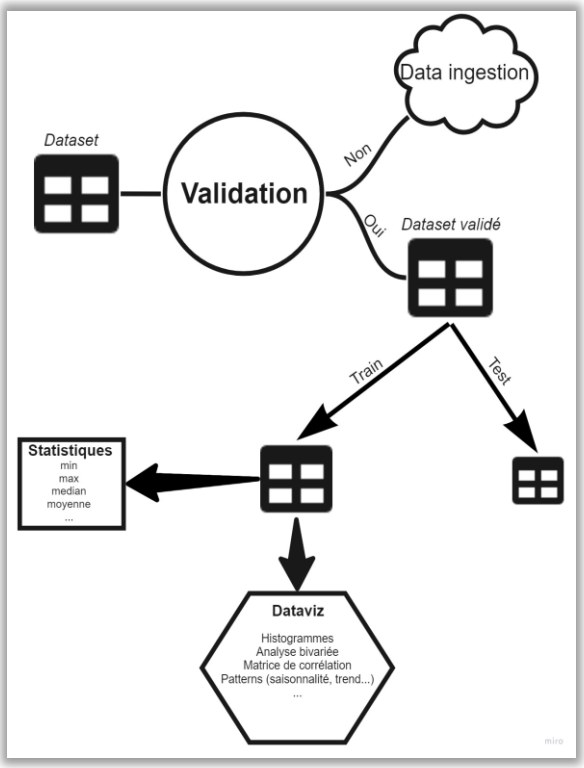


Figure 8 – Analyse Exploratoire des Données

Cette phase est essentielle à la bonne réalisation d'un projet d'intelligence artificielle, car elle va permettre d'avoir une première idée du phénomène que l'on cherche à modéliser. La variable cible est-elle saisonnière, a-t-on une tendance, a-t-on un échantillon assez volumineux pour représenter cette saisonnalité ?

Avant de se lancer dans les premières analyses, il est primordial de procéder à la validation des données. Les données fournies sont-elles fiables ?

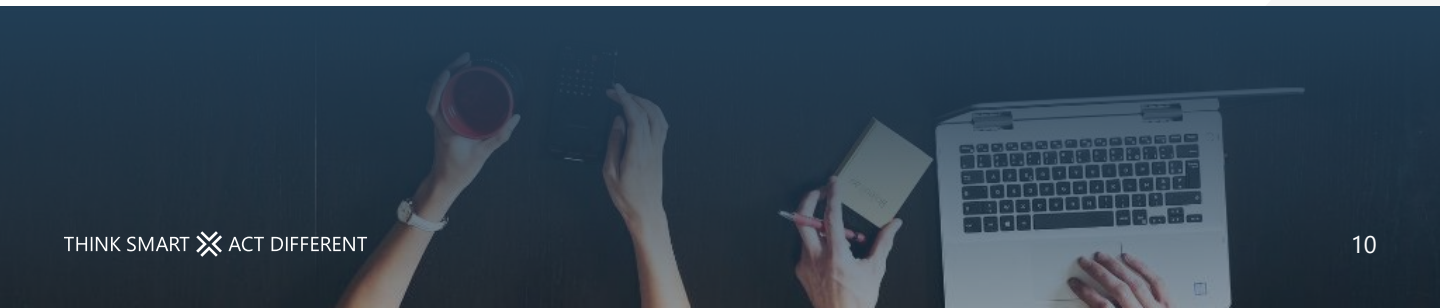
Les outliers rencontrés sont-ils des erreurs de saisie ou des représentants d'événements rares ? Qu'en est-il des valeurs manquantes ? Ce travail doit être réalisé en étroite collaboration avec les métiers. Tout d'abord pour bénéficier de leur expertise, mais aussi pour les inclure dans le processus de transformation digitale et en faire des parties prenantes du projet.

Si les données reçues ne sont pas valides, il faut repasser par la phase d'ingestion de données.

L'étape suivante est la séparation des données entre un jeu d'entraînement et un jeu de test. C'est une des phases critiques car elle permettra par la suite de vérifier la capacité des modèles à généraliser et d'éviter l'overfitting.

Le jeu de test ne doit être utilisé qu'à la fin du développement et n'intervenir pour le calibrage d'aucun paramètre du modèle, comme le choix de la stratégie de remplacement des valeurs manquantes, les hyper paramètres des modèles, les seuils de décision pour la classification, etc...

Il est également nécessaire de réaliser cette séparation avant les premières analyses exploratoires, afin de ne pas récolter d'informations sur le jeu de test, ce qui pourrait biaiser les résultats lors de l'évaluation.



Généralement, 80% des données disponibles sont gardées pour l'entraînement et 20% pour le test. Cela dépend bien sûr du cas d'usage et du volume de données disponibles.

La phase suivante est définie par l'analyse exploratoire des données. Lors de cette exploration, il est possible de confronter les premières hypothèses tirées des données avec les connaissances des collaborateurs "métier". Il n'y a pas de questions bêtes, car toute information permettra de mieux comprendre les interactions entre les différentes variables.

Il est d'usage de commencer par calculer le maximum, le minimum et la moyenne, afin de mieux comprendre les phénomènes observés. Ou encore étudier la médiane et autres quantiles. Cela permet de découvrir comment se répartissent les variables et de détecter d'éventuelles erreurs de saisie. Par exemple, une variable température d'air présentant un maximum de 100°C doit éveiller quelques soupçons.

Des analyses plus avancées et portant davantage d'information peuvent être effectuées, comme des histogrammes, de l'analyse bivariée, ainsi que la décomposition en tendance, saisonnalité et résidus dans le cas de séries temporelles.

L'étude des interactions entre variables grâce à la matrice de corrélation est également pertinente. De nombreux exemples d'analyses exploratoires sont disponibles sur le site des concours Kaggle.

Nettoyage

Une fois que les données nécessaires à l'étude ont été convenablement explorées et validées, il convient de les nettoyer.

Il est utile d'harmoniser les noms des colonnes, en supprimant les espaces, les majuscules, les accents, etc... Une des raisons premières est de ne pas générer de bugs lors des phases d'utilisation du modèle.

Par exemple, les variables “age”, “Age” ou “âge” seront reconnus comme des variables différentes par le modèle. S’il a été entraîné avec “age” et que la variable “Age” est fournie en production, la prédiction n’aura pas lieu. Les données envoyées au modèle par un utilisateur tiers peuvent varier dans leur format, il faut donc les normaliser.

Il est également nécessaire de filtrer les données hors périmètre. Par exemple, si le cas d’usage porte sur les clients particuliers d’une banque et que le dataset contient également les professionnels, ils doivent être écartés.

Il faut aussi imaginer des procédures pour traiter les valeurs manquantes et les outliers. Faut-il remplacer les valeurs manquantes par une chaîne de caractères comme “-” ou bien les remplacer par la valeur médiane ou la moyenne ? Peuvent-elles être naturellement déduites à partir d’autres variables ou encore développer un modèle pour prédire ces valeurs ? Concernant les outliers, la ligne en contenant doit-elle être supprimée ou remplacée ?

Les réponses à ces questions ne peuvent être déterminées *a priori*. Différents plans peuvent être envisagés et comparés durant la phase d’entraînement des modèles. Il y a un compromis à trouver entre performance et complexité.

Les réponses à ces questions ne peuvent être déterminées *a priori*. Différents plans peuvent être envisagés et comparés durant la phase d’entraînement des modèles. Il y a un compromis à trouver entre performance et complexité.

Le pipeline de nettoyage devra être appliqué au dataset de test en fin de projet, lorsque tous les paramètres des modèles auront été fixés lors de l’entraînement

Pour davantage d’informations, il est possible de consulter l’article de Liu *et al.* sur la détection d’anomalie, ainsi que l’article Nexialog d’Ahmad Charaf sur le traitement des valeurs manquantes.

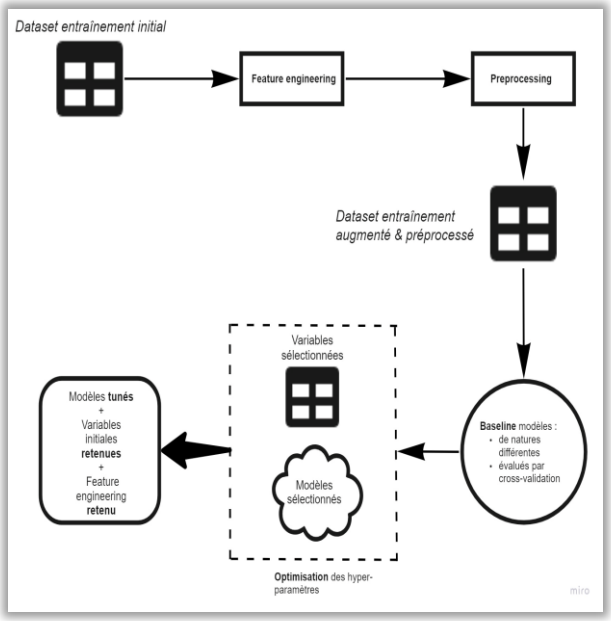
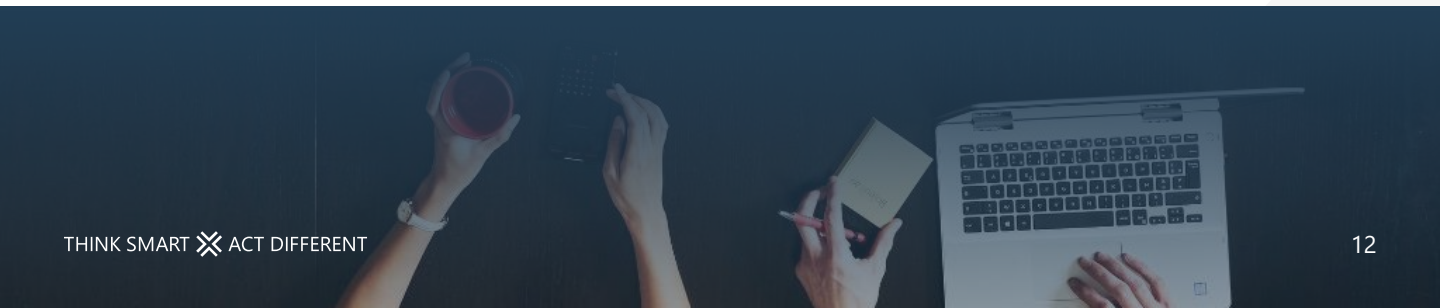


Figure 9– Pipeline d’entraînement



2.2 Model Engineering

Entraînement

Ce module peut être divisé en deux étapes distinctes : l'entraînement et l'évaluation du modèle entraîné.

Comme dit précédemment, l'entraînement est réalisé sur le jeu d'entraînement. Le jeu de test, aussi appelé hold-out, doit être laissé de côté afin de ne pas influencer le choix des paramètres. Le jeu de test sert à mimer les données de production, sur lesquelles nous n'avons pas d'information *a priori*. Il sera utilisé lors de l'évaluation du modèle.

Avant de se lancer dans l'entraînement des modèles, il est important de rappeler qu'un projet ML de qualité repose avant tout sur des données de qualité. Les données initiales ne contiennent pas toujours une quantité de signal suffisante pour prédire la variable cible avec précision.

C'est ici qu'intervient l'étape de feature engineering, qui consiste à transformer, agréger les données initiales pour obtenir de nouvelles variables, parfois plus informatives, qui vont s'ajouter aux premières. Des transformations simples peuvent être envisagées, comme le passage au logarithme, à la racine, au produit, au quotient.

Dans le cas de séries temporelles, il est parfois utile d'utiliser des lags ou bien des moyennes mobiles. Une bonne pratique est de ne pas se restreindre en début de projet et d'étudier une large gamme de feature engineering.

La sélection aura lieu lors de l'entraînement des modèles à proprement parler.

Vient ensuite une étape de préprocessing, qui va mettre la donnée dans une forme acceptable par les modèles de ML. Ces derniers ne comprennent que les chiffres, il faut donc encoder les variables catégorielles. Une autre étape essentielle du preprocessing est la standardisation, où l'on va retirer à chaque variable sa moyenne et la diviser par son écart-type.

L'objectif est double : lisser les différences d'échelles entre les variables, et permettre aux algorithmes de converger plus vite lors des descentes de gradient. La plupart des algorithmes ne supportent pas les valeurs manquantes, il faut donc les remplacer, en utilisant une des méthodes étudiées durant la phase de nettoyage. De nombreux exemples de préprocessing sont disponibles sur la page scikit-learn.

La donnée étant maintenant dans une forme digeste pour un modèle de ML, l'entraînement peut avoir lieu. Une bonne pratique est de commencer par des modèles simples ou baselines. Dans un premier temps, les hyperparamètres sont fixés à leur valeur par défaut et différentes familles de modèles sont testées : modèles linéaires, modèles à base d'arbre (Decision Tree, Random Forest, XgBoost, LightGBM etc..), SVM, k-nearest neighbors, et bien d'autres.

Ces modèles peuvent aussi être utilisées pour opérer la sélection des variables pertinentes : certains algorithmes linéaires, tels que le Lasso vont avoir tendance à donner des poids proches de 0 aux variables peu importantes pour la prédiction. Pour plus d'informations, il est possible de consulter l'article de R. Muthukrishnan (2016). Le feature importance des algorithmes à base d'arbres, tels que Random Forest, est également utilisable.

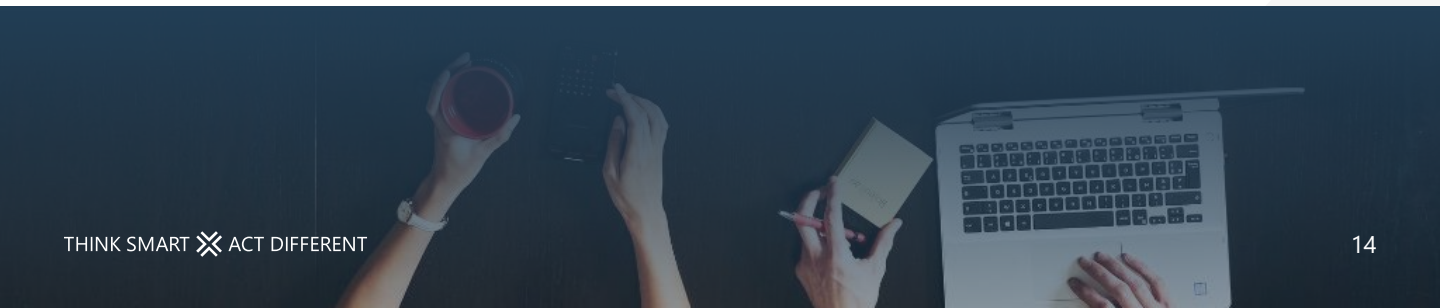
Ces modèles sont évalués par validation croisée, qui consiste à séparer le jeu d'entrainement en sous-blocs d'entraînement et de test. Une explication détaillée est disponible sur le site de scikit-learn.

L'objectif est maintenant de sélectionner quelques modèles prometteurs et de passer à la phase de recherche des meilleurs hyperparamètres.

C'est un problème d'optimisation, puisqu'on va chercher le plus souvent à minimiser une fonction de coût ayant de bonnes propriétés. Dans le cas où la fonction de coût n'est pas convexe, c'est aussi un problème gourmand en ressources, car l'espace dans lequel vivent ces hyper-paramètres est souvent grand. Il convient donc d'échantillonner l'espace avec diverses techniques. Il est possible de choisir les paramètres à tester (Grid Search), les tirer dans un certain espace (Random Grid Search), ou adopter des techniques plus élaborées, qui vont utiliser des a priori statistiques (Bayesian Search). C'est cette dernière technique qui donne généralement les meilleurs résultats.

Pour en savoir davantage sur l'optimisation des hyperparamètres, le site de scikit-learn recense les techniques classiques et l'article de Will Koehrsen (2018) développe l'approche bayésienne.

Une fois les modèles entraînés, il est envisageable de créer un méta-modèle, qui en moyenne les prédictions. Différentes pondérations peuvent être testées, il est d'usage de donner davantage de poids aux meilleurs modèles. Cela donne généralement de très bons résultats, mais ajoute une complexité qui peut être difficile à maintenir en production. Encore une fois, il y a un compromis entre complexité et performance.



Validation & test

Il est maintenant possible d'utiliser le jeu de test, mis de côté au début.

Il faut au préalable appliquer toutes les étapes de nettoyage et preprocessing retenues durant l'entraînement à ce jeu de test, comme illustré en **Figure 10**.

Les performances doivent être validées sur les métriques business, celles qui intéressent le plus l'utilisateur final. Il faut également s'assurer que les niveaux de performance observés lors de l'entraînement se vérifient sur le jeu de test. L'overfitting a ainsi été évité, et il est possible d'aborder sereinement la mise en production.

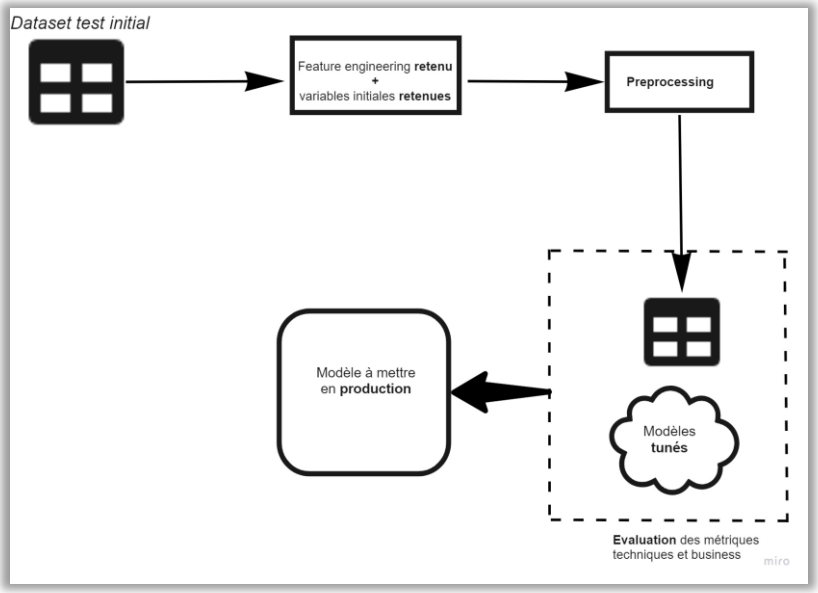
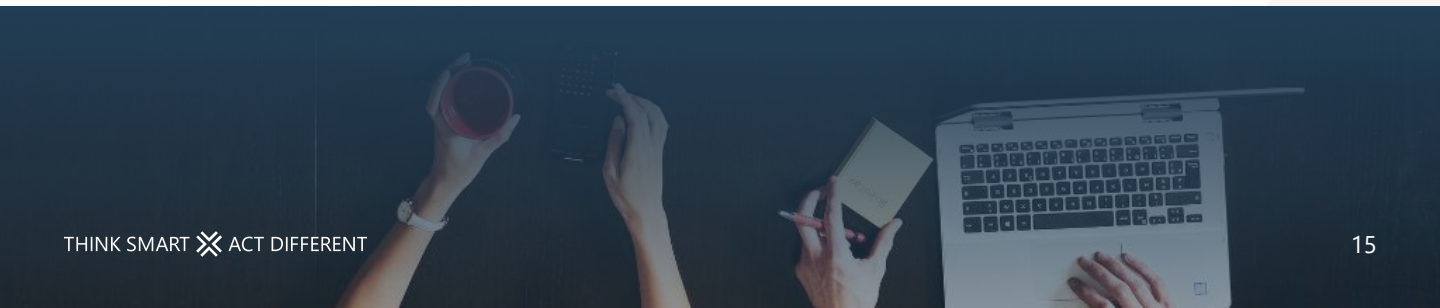


Figure 10– Pipeline de test

2.3 Déploiement

Lorsque l'entraînement est terminé et qu'un modèle valide tous les critères de mise en production, il est temps de le déployer.

Le déploiement d'un modèle peut être fait de multiples manières, qui dépendent notamment de la façon dont les prédictions vont être consommées en production. Les étapes clefs du déploiement sont présentées : le packaging du modèle et de ses dépendances, la transformation en API (Application Programming Interface) et la conteneurisation.



Packaging

Comme toute ressource informatique, un modèle doit être sauvegardé pour être réutilisé. Un modèle est un objet plus complexe qu'un simple fichier, il doit être converti dans un format qui permet son stockage et qui lors de son utilisation permet sa restauration, avec ses paramètres et hyperparamètres. Ce procédé se nomme sérialisation. Les formats les plus courants sont joblib, pickle, onnx, hdf5. Il est possible de consulter le blog de Christopher Flynn pour plus de détails.

Un modèle est en interaction avec d'autres éléments : les packages et librairies dont il dépend, des variables d'environnement, un système d'exploitation. Les packages utilisés lors du développement ne seront peut-être pas les mêmes que ceux installés sur l'environnement de production. Il faut donc figer toutes ces dépendances, afin qu'elles soient transmises à l'environnement de production. Il est souhaitable que le modèle s'intègre au système tiers sans perturbé son fonctionnement. Comme il sera discuté dans les pages suivantes, c'est là que la conteneurisation intervient.

Transformation en API

Pour permettre à un tiers de consommer les prédictions générées par un modèle, il ne suffit pas de lui fournir un script Python avec une méthode *predict*.

Le modèle doit s'intégrer et s'adapter à l'environnement cible, pas l'inverse. L'API va permettre de mettre en place une architecture client-serveur, faisant du modèle une web-app.

Le système qui consomme les prédictions enverra une requête contenant les données, et le modèle apifié renverra la prédiction.

Toutes les transformations nécessaires à l'ingestion des données par le modèle doivent être faites côté serveur. La requête et la réponse devront respecter certaines contraintes d'homogénéité., comme présenté sur le site redhat.com.

De nombreux frameworks permettent de transformer un modèle en web-app, notamment Flask et FastAPI.

Conteneurisation

Commençons par montrer l'utilité de la conteneurisation par un exemple. Imaginons qu'un modèle est développé sur un MacBook, avec Python 2.7. Ce modèle utilise entre autres librairies pandas et scikit-learn, avec des versions bien définies. Plusieurs questions se posent alors :

• Comment savoir si la configuration et les variables d'environnement en production seront compatibles avec l'environnement de développement (MacOS vs Windows ?)

•Python est-il installé ? avec la bonne version ?

• Les librairies de machine learning disponibles en production seront-elles compatibles avec le modèle développé ?

• ...

Pour le cas d'un système complexe, c'est-à-dire composé de nombreuses entités, les questions sont sans fin.

La conteneurisation, dont Docker est la technologie phare, résout ces problèmes.

Voici la définition d'un conteneur, provenant du site docker.com "A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another" .

La **Figure 11** récapitule les étapes nécessaires au déploiement

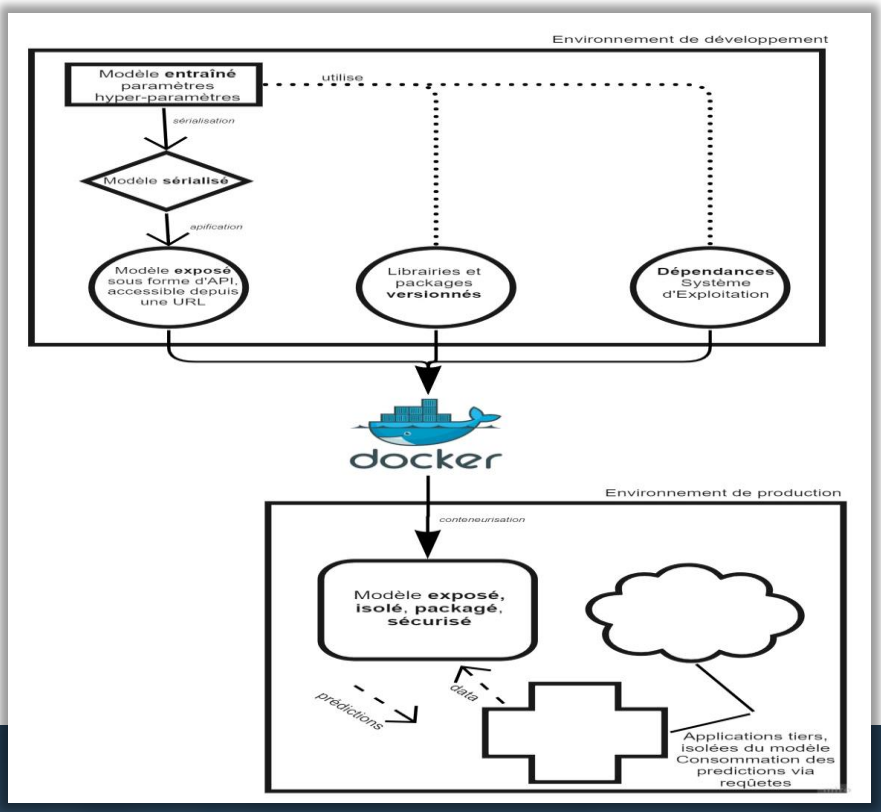


Figure 11– Déploiement d'un modèle

Dans la partie suivante, le cadre MLOps sera appliqué aux différentes étapes du cycle de vie d'un modèle.

III. Comment mettre en place le cadre MLOps

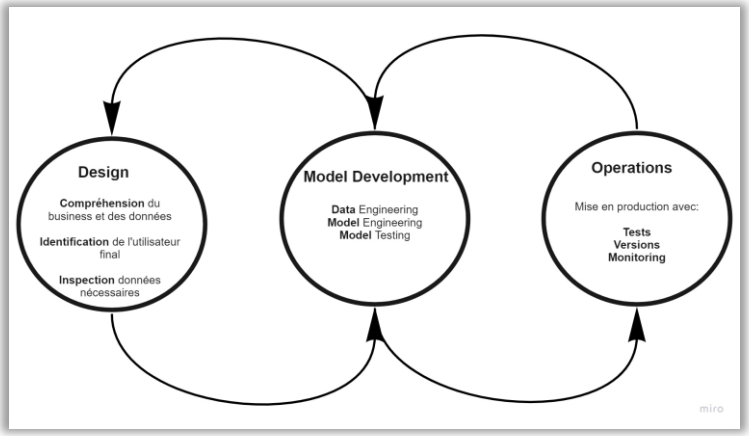


Figure 12– MLOps : un processus itératif et incrémental

Dans la partie précédente, les différentes étapes d'un projet de ML ont été exposées, allant de l'ingestion des données jusqu'à la mise en production. Dans cette partie, les pratiques MLOps seront appliquées aux trois étapes vues précédemment : Data Engineering, Model Engineering, Déploiement.

La **Figure 12** permet de remarquer que les différentes concepts MLOps sont interdépendants et connectés. La phase de design correspond au cadrage du projet. Les questions suivantes seront notamment posées :

• A quoi servira le modèle développé, quelle est la proposition de valeur ?

• A-t-on réellement besoin de l'intelligence artificielle pour répondre à la problématique ?

• Qui est l'utilisateur final ? A quelle fréquence les prédictions doivent-elles être générées ?

• De quelles données a-t-on *a priori* besoin ?

Cette première phase est l'occasion de discuter avec les métiers pour mieux comprendre leur problématique et leur business.

Elle ne doit pas être négligée car elle aura une influence considérable sur la suite du projet. Elle devra également être documentée le plus possible.

Comme vu précédemment, la phase de développement a pour objectif la création d'un modèle, qui sera ensuite mis en production. Cette mise en production se fait généralement en deux temps dans le cadre MLOps.

Le modèle est d'abord placé dans un environnement de pré-production, ou d'intégration, qui aura les mêmes caractéristiques que l'environnement de production mais ne sera pas accessible à l'utilisateur final.

Des tests en condition réelle peuvent ainsi être exécutés, sans risque de perte de valeur pour le client. Une fois ces tests validés, le modèle peut être mis en production.

Les déploiements doivent être versionnés, ce qui permet un passage rapide d'une version à une autre. Il est question de "roll-back" lorsqu'une version précédente est redéployée. Le monitoring des modèles est également essentiel, car il permet de vérifier si les performances se dégradent et donc si un ré-entraînement est nécessaire.

Le MLOps repose notamment sur différents principes : le versioning, les tests, l'automatisation, la reproductibilité et le monitoring.

Ces principes vont être appliqués aux étapes de construction d'un modèle de ML, vues en Partie II. De plus en plus, se pose le problème de l'interprétabilité et l'explicabilité des modèles. Pour plus d'information, consulter l'article Nexialog de Victor Nguyen et Nazaire Kadio à ce sujet.

Les solutions proposées dans cette note d'introduction au MLOps ne sont bien évidemment pas les seules possibles. Le MLOps étant un domaine en expansion, de nombreuses technologies et principes émergent.

3.1 MLOps Data Engineering

Comme vu précédemment, cette partie va de l'ingestion des données à leur nettoyage. Une brique essentielle au MLOps et qui sera utile tout au long du cycle de vie d'un projet ML va être ajoutée : un data registry, qui va permettre de stocker différents types de données et les versionner.

A tout moment, il sera possible de basculer vers les données d'une version précédente, soit pour tester de nouvelles idées, soit pour reproduire des comportements passés.

Imaginons que trois mois après la mise en production, un problème apparaisse.

Si un registre de données n'est pas accessible et que les données initiales ne sont plus disponibles chez le client, il sera difficile de trouver la cause du problème.

Le data registry va également permettre de stocker certaines statistiques qui pourront être utilisées durant l'entraînement et la production, comme par exemple les moyennes et écarts-type, nécessaires à la standardisation des données

Il est recommandé d'utiliser un outil gérant lui-même ce versionnage des données, comme par exemple DVC (Data Version Control), qui est open-source est construit sur Git.

La **Figure 13** ci-dessous illustre le caractère central du data registry dans la phase de Data Engineering :

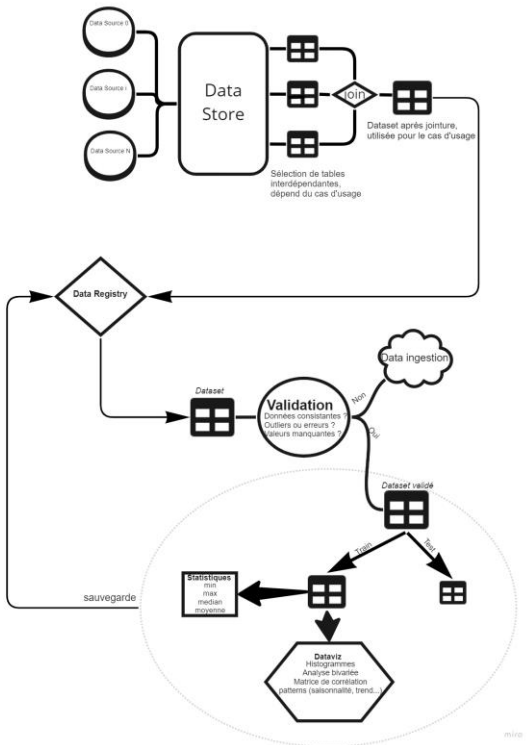


Figure 13– MLOps Data Engineering

3.2 MLOps Model Engineering

Un nouvel artefact est introduit dans cette phase de Model Engineering : le model registry. De la même manière qu'un data registry permet de stocker et versionner des données, un model registry sert à stocker et versionner des modèles, permettant leur réutilisation.

Il est également possible de conserver les hyper-paramètres et métriques des modèles entraînés. C'est l'experiment tracking. Ceci est particulièrement utile dans le processus de décision lié à la mise en production du modèle.

Comme le montre la **Figure 14** ci-dessous, tous les datasets créés à différentes étapes du pipelines sont sauvegardés dans le data registry. Lorsqu'un dataset déjà sauvegardé doit être utilisé, il suffit de le charger en prenant la bonne version depuis le registre. Le même principe s'applique aux modèles.

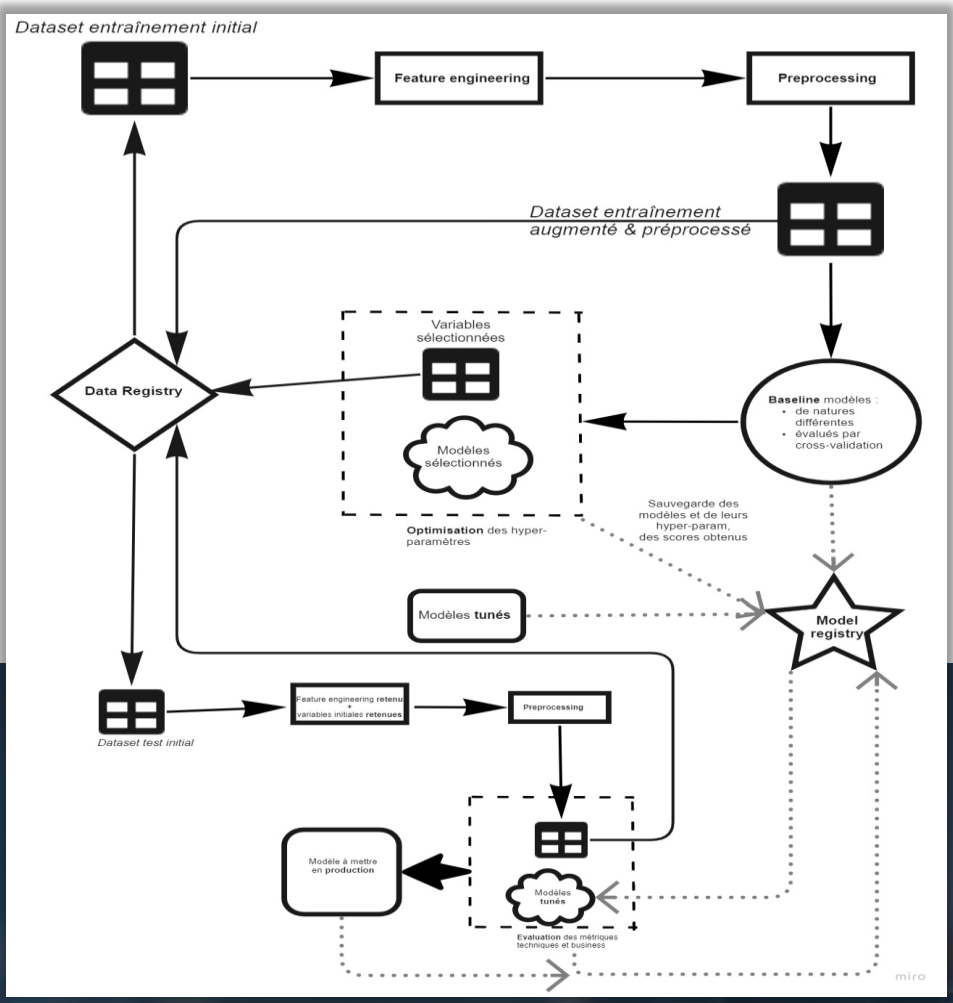


Figure 14 – MLOps Model Engineering

Model registry & Experiment tracking

Pour la partie model registry et experiment tracking, une solution consiste à utiliser MLFlow, un projet open source qui offre ces deux fonctionnalités.

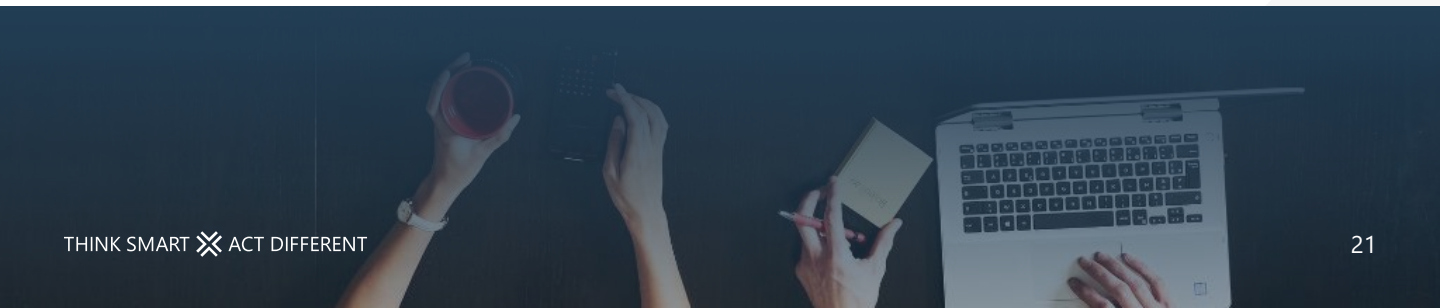
L'interface utilisateur présentée en **Figure 15**, bien que sommaire, permet de visualiser les résultats d'un entraînement.

Les hyper-paramètres des modèles (colonne Parameters ci-dessus) sont disponibles, ainsi que les métriques d'évaluation, choisies par l'utilisateur et sauvegardées durant l'entraînement. Tout cela est par ailleurs versionné.

Dans l'hypothèse où des visuels plus étoffés seraient nécessaires, il est possible d'utiliser un outil de dashbording tel que Grafana, les données MLFlow étant exposées via une API.

					Parameters		Metrics		
	Date	User	Source	Version	alpha	l1_ratio	mae	r2	rmse
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	1	0.649	0.04	0.862
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.5	0.648	0.046	0.859
☐	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.2	0.628	0.125	0.823
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	1	0	0.619	0.176	0.799
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	1	0.648	0.046	0.859
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.5	0.628	0.127	0.822
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.2	0.621	0.171	0.801
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0	0.615	0.199	0.787
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	1	0.578	0.288	0.742
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.5	0.578	0.288	0.742
☐	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.2	0.578	0.288	0.742
☐	2018-06-04 23:00:08	mlflow	train.py	05e956	0	0	0.578	0.288	0.742

Figure 15– MLOps Data Engineering



3.3 MLOps Déploiement

Il a été dit qu'une fois le meilleur modèle sélectionné, il est possible de le déployer en production. Cependant, il peut sembler dangereux d'intégrer une nouvelle entité à des systèmes déjà en production. Il serait tentant de penser que si tout s'est bien déroulé lors du développement, il devrait en être de même pour la prod. De même qu' un dataset de test est utilisé pour valider les résultats de cross-validation, un environnement de pré-production, aussi appelé environnement d'intégration, va mimiquer l'environnement final.

Il s'agit alors de tester le modèle et l'API dans un environnement proche des conditions réelles. Il est possible à cette fin d'utiliser le framework GitOps. Divers tests peuvent être imaginés, selon la nature du cas d'usage. Si les prédictions sont réalisées en temps réel, autrement dit à la demande, il peut être bon d'envoyer de nombreuses requêtes, simultanément ou pas, et analyser comment le système se comporte. Il faut également vérifier qu'il n'y a pas de biais au niveau du pipeline de production. Les modèles dans l'environnement de développement et de production doivent générer les mêmes prédictions à partir des mêmes données.

Lorsque cette batterie de tests est validée, il est possible de sereinement mettre le modèle en production. Commence alors une étape primordiale du cadre MLOps : le monitoring. Durant la phase de développement, certaines métriques, qu'elles soient business ou techniques, ont été validées par le client comme conformes à ses attentes.

Il faut donc apporter la preuve que ce niveau de performance est atteint en production et s'il ne l'est pas, proposer des solutions. Il faut donc mesurer la qualité de la prédiction réalisée, en comparant les métriques de développement aux métriques de production. De manière générale, il faut monitorer le plus de données possible. Voici une liste non exhaustive des quantités à mesurer :

- Nombre de requêtes faites à l'api à différentes mailles temporelles
- Temps de réponse de l'API
- Monitorer des changement de distribution dans les variables, pouvant induire un drift
- Monitorer les KPIs (business ou techniques) observés sur les données de production

Ce monitoring doit être couplé à un alerting, permettant d'informer les décisionnaires et d'enclencher un ré-entraînement si besoin. Le seuils d'alertes dépendent du cas d'usage et des attentes de performance du client.

Ces étapes sont illustrées en **Figure 16**.

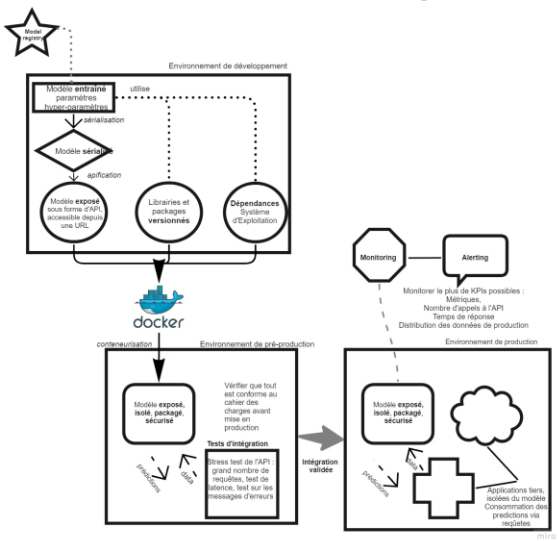


Figure 16– MLOps Déploiement

CONCLUSION

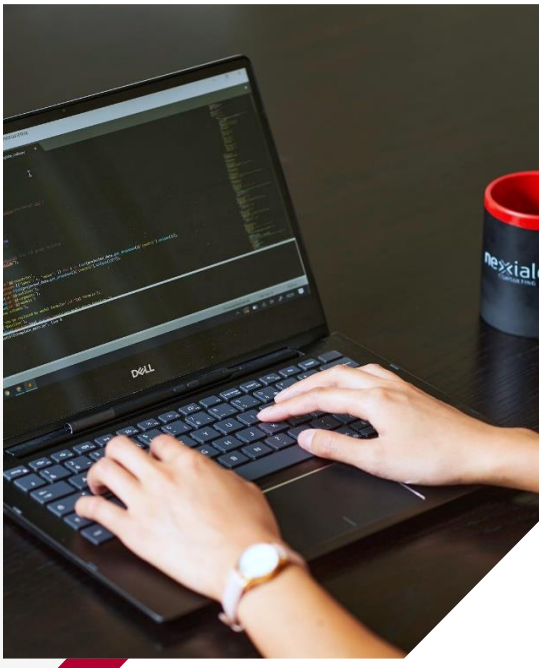
Le MLOps est l'application des principes DevOps au machine learning. Compte tenu de la nature dynamique des données, ce cadre doit être adapté pour tenir compte des différences entre développement logiciel traditionnel et basé sur des modèles entraînés.

Certaines bonnes pratiques ont été abordées dans cette note, qui n'a pas vocation à être exhaustive, mais plutôt à familiariser le lecteur aux enjeux et difficultés soulevés par la mise en production de produits ML.

Les processus à mettre en place sont bien sûr dépendants du secteur d'activité, du cas d'usage et de la fréquence de prédictions.

Pour un système recevant des centaines de requêtes à la minute, la structure sera différente de celui qui envoie cent prédictions par semaine.

L'essentiel est de se poser les bonnes questions et de tester le plus de cas possibles. Pour aller plus loin, on pourra consulter la page GitHub sig-mlops qui donne les directives de ce que devrait accomplir le MLOps.



Références

[1] Algoritmia (2021), « 2021 Entreprise Trends in Machine Learning »

[2] Albert Bifet et al., (2006) « Early Drift Detection Mehtod », *academia.edu*

[3] Atlassian, « Principes DevOps », www.atlassian.com/fr/devops/what-is-devops

[4] Tuatini Godard, « Detailed exploratory data analysis with python » in Kaggle, www.kaggle.com/code/ekami66/detailed-exploratory-data-analysis-with-python/notebook

[5] F. T. Liu et al., (2008), « Isolation Forest », *2008 Eighth IEEE International Conference on Data Mining*

[6] Ahmad Charaf, « Traitement des données manquantes dans le milieu bancaire », Nexialog Consulting

[7] Scikit-learn, « Preprocessing data », https://scikit-learn.org/stable/modules/preprocessing.html

[8] R. Muthukrishnan, (2016), « LASSO : A feature selection technique in predictive modeling for machine learning », *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*

[9] Scikit-learn, « Cross-validation : evaluationg estimator performance », https://scikit-learn.org/stable/modules/cross_validation.html

[10] Scikit-learn, « Tuning the hyper-parameters of an estimator », https://scikit-learn.org/stable/modules/grid_search.html

[11] Will Koehrsen, (2018), « An Introductory Example of Bayesian Optimization in Python with Hyperopt », in Towards Data Science

[12] Christopher Flynn, (2020), « Machine Learning Model Serialization »

[13] Red Hat, (2020), « Une API REST, qu'est-ce que c'est ? »

Nexialog Consulting

STRATÉGIE | ACTUARIAT | GESTION DES RISQUES

Nexialog Consulting est un cabinet de conseil spécialisé en Stratégie, Actuariat et Gestion des risques qui dessert aujourd'hui les plus grands acteurs de la banque et de l'assurance. Nous aidons nos clients à améliorer de manière significative et durable leurs performances et à atteindre leurs objectifs les plus importants.

Les besoins de nos clients et les réglementations européennes et mondiales étant en perpétuelle évolution, nous recherchons continuellement de nouvelles et meilleures façons de les servir. Pour ce faire, nous recrutons nos consultants dans les meilleures écoles d'ingénieur et de commerce et nous investissons des ressources de notre entreprise chaque année dans la recherche, l'apprentissage et le renforcement des compétences.

Quel que soit le défi à relever, nous nous attachons à fournir des résultats pratiques et durables et à donner à nos clients les moyens de se développer.

CONTACTS

Ali Behbahani
Associé, Fondateur
+ 33 (0) 1 44 73 86 78
abehbahani@nexialog.com

Christelle BONDOUX
Associée, Directrice commerciale
+ 33 (0) 1 44 73 75 67
cbondoux@nexialog.com

Adrien Misko
Responsable R&D
+ 33 (0) 6 69 27 62 26
amisko@nexialog.com

Paul-Antoine DELETOILLE
Responsable de Compte Banque et AM
+33 (0)1 44 73 75 70
+33 (0)7 64 57 86 69
padeletoille@nexialog.com

Eric BAESEN
Senior Manager Global Markets
ebaesen@nexialog.com

www.nexialog.com

Retrouvez toutes nos publications sur Nexialog R&D