N°5 | STEPWISE

RÉMI ADON | LUIS BLANCHE | JONATHAN CASSAIGNE | FLORIAN GARDIN | ALBERTO GUGGIOLA
GUILLAUME HOCHARD | GRÉGOIRE MARTINON | GUILLAUME MOCQUET | AURÉLIA NÈGRE

IA EN PRODUCTION

CYCLE DE VIE ET DÉRIVE DES MODÈLES

Quantmetry
Building AI with pioneers

3. EVOLUTIONS PROGRAMMÉES ET GESTION DU RÉ-ENTRAINEMENT

***Contributeurs** : Guillaume Mocquet*

Les diagnostics issus du *monitoring* de modèles, évoqués dans le chapitre précédent, ne peuvent rester sans action. En pratique, il doit naturellement découler de la recherche des causalités d'un problème sur les prédictions du modèle une ou plusieurs modifications portant sur le modèle lui-même, ainsi que ses données d'entrée.

Ce chapitre vise donc à recenser les méthodes et les bonnes pratiques permettant d'accélérer, et pourquoi pas d'actionner le ré-entraînement des modèles dans un projet, et ce à plusieurs niveaux :

- côté code d'abord, en se munissant des outils adaptés, dès le début du projet ;
- côté architecture ensuite, via la mise en place de composants et de processus dédiés ;
- côté gestion de projet enfin, par l'identification et la prise en charge des parties non automatisables.

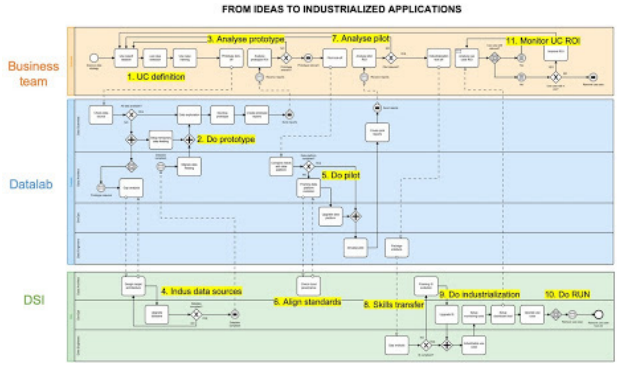
A. ORGANISATION

Avant d'aborder les enjeux techniques, il est important de sensibiliser l'ensemble des membres qui interagissent de près ou de loin avec le(s) modèle(s). L'objectif premier est de **désiloter l'organisation** pour faciliter les tests de nouveaux modèles, augmenter la fréquence des mises en production, réduire les risques lors du déploiement d'une nouvelle version...

Les projets de *Machine Learning* (ML) / *Data Science* (DS) requièrent une multitude de corps de métier qui n'ont pas les mêmes objectifs et responsabilités : Data Architects, Solutions Architects, Data Engineers, Data Scientists, Administrateurs Systèmes & Réseaux, Développeurs Back-end / Front-end / API, équipes métiers, experts DataViz, etc. Dans les grandes organisations, il est fréquent que l'ensemble des parties prenantes au projet de ML/DS ne se parlent pas. De plus, ces dernières sont généralement rattachées à différentes directions, voire réparties sur différents sites géographiques.

La clef de la réussite d'un projet de ML/DS passe par la mise en place d'un **SI agile transverse à l'organisation**. Il est important que l'ensemble des acteurs puissent interagir de façon (relativement) autonome sur l'ensemble des phases de développement, recette, mise en production, etc. La mise en production d'un nouveau modèle ne doit pas dépendre de la disponibilité d'une personne/équipe. Le cas échéant, cela créer un goulot d'étranglement qui ralentit l'actualisation des modèles. Or augmenter la fréquence des mises en production contribue à réduire le niveau de stress / risque des mises à jour.

Pour se faire, il est important de connaître précisément les interactions entre les différentes *business units* avec un focus sur les zones "d'échanges d'actions". La réalisation du **diagramme de collaboration** suivant la norme **BPMN** 2.0 est un excellent point de départ. Ci-dessous, un exemple de modélisation du processus d'industrialisation des cas d'usages de ML/DS au sein d'un grand groupe coté en bourse :



À partir du diagramme de collaboration détaillé, il est facile d'identifier les principales zones fonctionnelles et les interactions entre les différentes *Business Units*. Dans l'exemple ci-dessus, il s'agit des 11 points annotés en noir sur fond jaune. Les interactions entre les business units sont représentées par les flèches en pointillé. À noter la présence d'échanges de messages entre les différentes business units pour se synchroniser aux étapes clés lors de l'avancement des tâches dans le workflow.

Les "piscines" — grandes bandes de couleurs verticales — représentent chaque *Business Unit*. À l'intérieur, les tâches sont décomposées suivant les différents corps de métier : Data Architect, Data Scientist, etc.

Ce diagramme fait ressortir une vérité bien souvent oubliée : le design et le développement des modèles sont réalisés par l'équipe du datalab, par contre le **RUN est opéré par l'équipe de la DSI**. Dans bien des cas, ce sont deux mondes totalement distincts !

Le travail réalisé ci-dessus est un bon point de départ pour définir le workflow de mise en production transverse à l'organisation des cas d'usages à base de ML/DS. Pour aller plus loin dans la gouvernance du cycle de vie des modèles et les actions à mettre en place, consulter le chapitre 6.

B. AUTOMATISATION

La réussite du déploiement des cas d'usages ML/DS réside dans la capacité de l'organisation à **industrialiser et automatiser** ses workflows de mise en production sans créer de goulot d'étranglement. Dans cette optique, il convient d'utiliser les outils utilisés par les principes **DevOps**. Attention toutefois à ne pas tomber dans le dogme ! En effet, contrairement à un projet informatique "standard", les projets de ML/DS requièrent une plus forte implication des utilisateurs métiers. Ainsi, il convient d'être extrêmement **agile** et d'accompagner ces derniers dans la prise de responsabilité du workflow de mise en production. C'est un fort challenge car ces profils sont des experts sur le plan métier et business mais généralement néophytes sur le plan technique. Il est important de décomplexer et automatiser les tâches techniques du workflow. Ce point est paradoxal, car les projets ML/DS requièrent une expertise de plus en plus pointue sur le plan technique.

L'automatisation du workflow est une tâche complexe, qui nécessite de prendre en considération pléthore de sujets :

- Workflow / gestion de projet ;
- L'ossature de l'architecture ;
- Stack CI/CD ;
- Gestion des versions des applications ;
- Gestion des versions des modèles.

WORKFLOW / GESTION DE PROJET

Il est important que l'ensemble des parties prenantes collaborent ensemble via le même outil et le même workflow. Cela passe par l'utilisation d'un outil de ticketing/ gestion de projet sur mesure. Dans les grandes organisations, on retrouve bien souvent le logiciel **JIRA** édité par la société Atlassian. Il s'agit de la référence incontestable en la matière. À l'instar de SAP, c'est un progiciel complexe à appréhender mais nécessaire pour modéliser un workflow digital qui "colle" à la philosophie, aux usages et aux particularités de l'organisation. C'est l'outil qui doit s'adapter à la façon de travailler des collaborateurs et non l'inverse. Utiliser un produit (trop) simple s'avère rapidement contre productif car trop restrictif !

L'automatisation du workflow nécessite d'utiliser une solution de *Continuous Integration* (**CI**) / *Continuous Delivery* et *Continuous Deployment* (**CD**) couplé à l'exploitation des **APIs** JIRA.

L'OSSATURE DE L'ARCHITECTURE

Mettre au point un projet de ML/DS ne doit pas faire perdre de vue les standards mis au point depuis des décennies par l'architecture logicielle "standard". Il n'est pas rare, en début de projet ML/DS, de voir les *Data Scientists* et *Data Engineers* foncer sur le développement du code en omettant les bonnes pratiques du développement logiciel :

- Absence d'**outil de gestion de version** telle la référence **GIT.**
- Non implémentation des **design patterns** favorisant la réutilisation du code sans copier/coller, tels que la **modélisation objet.**
- Application monolithique faiblement **modulaire** : il est difficile de **réutiliser un composant développé dans un autre contexte.** Par exemple, utiliser du code issu d'une application de production en mode exploratoire via un notebook.
- Traitements **automatisés** quasiment inexistants. Dans bien des cas, il s'agit de divers scripts bash et/ou python "bricolés" par divers développeurs / *Data Scientists* sans réelle cohérence ni vision d'industrialisation sur le long terme.

Pour que les applications avec des cas d'usages ML/DS soient en mesure de scaler correctement une fois en production, il est important d'investir dès le début du projet sur les problématiques d'architecture et mettre en place le plus tôt possible les bonnes pratiques évoquées dans le paragraphe précédent. Plus les notions d'architecture sont abordées tardivement, plus la dette technique est importante et dure à supprimer !

Sur les projets informatiques qui manipulent beaucoup de données, il est important de centraliser les *features* / KPIs générés de façon à maximiser la réutilisation du travail réalisé par une équipe. En 2017, Uber théorise le concept de ***Feature Store*** qui permet aux équipes de partager, de découvrir et d'utiliser un ensemble de *features* / KPIs conservés comme point d'entrée d'un nouveau

cas d'usage ML/DS. Ce concept est né du constat que de nombreux problèmes de modélisation chez Uber utilisent des fonctionnalités identiques ou similaires, et qu'il est très utile de permettre aux équipes de partager des fonctionnalités entre leurs propres projets et aux équipes de différentes organisations de partager des fonctionnalités entre elles. Les bénéfices sont doubles :

- Raccourcir les délais de développement des projets ML/DS : capitalisation des traitements.
- Augmenter la qualité et la disponibilité des applications à base de ML/DS : suppression de l'effet copier/coller..

STACK CI/CD

Mettre en place une stack de CI/CD est un point essentiel qui est souvent omis lors de la construction de la roadmap des projets ML/DS car jugé (à tort) "*overkill*". Posséder une telle stack constitue pourtant un **pilier de la philosophie DevOps.** L'outil de prédilection est **Jenkins**. Véritable pionnier, il s'agit d'un produit Open Source fort d'une dizaine d'années d'existence. Cette ancienneté lui confère une communauté riche et active. De plus, ce produit dispose de plus de 1 000 plugins pour s'interfacer facilement avec les principaux outils / technologies du marché : GIT, Docker, unit tests, JIRA, etc.

Suivant le niveau de maturité, le paramétrage de la chaîne de CI/CD peut être réalisée en mode clic bouton ou **intégralement en code** via le langage **Groovy.** Ce dernier point est particulièrement intéressant pour historiser les changements de votre chaîne de CI/CD simplement dans un *repository GIT*, au même titre que l'ensemble des sources du projet.

La bonne pratique repose sur l'utilisation des ***pipelines*** Jenkins. Les grandes étapes du *workflow* sont les suivantes :

1. Checkout de la dernière version de la branche develop
2. Génération du nouveau numéro de version
3. Création d'un nouveau Tag GIT (sans publier sur le remote)
4. Exécution des tests unitaires
5. Exécution des tests d'intégrations

6. Merge la branch develop sur la branche master
7. Commit et publication de la branche master (inclus le Tag GIT précédemment créé) sur le remote
8. Checkout du Tag GIT depuis la branche master
9. Création des artefacts (i.e. build)
10. Publication des artefacts sur le service de repository binaire

À noter : Si les tests unitaires et intégrations ne passent pas, les étapes suivantes ne sont pas exécutées.

Contrairement à l'utilisation des produits proposés par les hébergeurs de solutions cloud tel AWS ou GCP, l'utilisation de **Jenkins évite le *vendor lock-in.*** En effet, les scripts développés sont portables sur n'importe quelle autre stack : aussi bien on-premise que sur un autre hébergeur.

GESTION DES VERSIONS DES APPLICATIONS

Au bout de la chaîne de CI/CD (cf. section précédente), il est important de conserver l'historique des versions générées, dépendances incluses, en vue de **pouvoir rétablir rapidement une ancienne version de l'application**. A première vue, agir de la sorte peut sous-entendre une duplication de l'historique des versions dont le code source est déjà stocké au sein d'un ou plusieurs dépôts GIT. Cependant, tracer les évolutions des versions avec les bonnes versions des dépendances peut vite s'avérer être un enfer !

De plus, un système de gestion des versions binaires prend en charge l'hébergement local des dépendances tierces. Ceci permet de prémunir le SI de la disparition et/ou de l'indisponibilité d'une bibliothèque publiée sur Internet / GitHub.

Pour finir, ce type d'outil permet une gestion accrue des droits. Il est possible de différencier les utilisateurs autorisés à "tirer" les artefacts, des utilisateurs qui publient de nouvelles versions (généralement la machine en charge de la CI/CD). Dans le cas d'un SI composé de plusieurs écosystèmes, il est important de sélectionner un outil qui prenne en charge l'ensemble des technologies utilisées (Python, Docker, Java, ...).

Nexus Repository Manager est une solution de premier choix. Celle-ci est disponible en deux éditions :

- OSS : l'offre de base gratuite (largement suffisant pour démarrer dans de bonnes conditions).
- PRO : un produit payant pour les entreprises en quête de besoin de haut niveau (haute disponibilité, support technique premium, etc.).

Ce produit à l'avantage de gérer la majorité des formats manipulés dans l'écosystème ML/DS tel **Python, Docker, APT**/YUM, Maven/Java, npm, P2 et bien d'autres.
L'administration des utilisateurs est réalisée via une interface graphique simple d'utilisation. Il est possible de configurer un connecteur **LDAP** pour bénéficier du dictionnaire des utilisateurs et groupes déjà en place au sein de l'organisation.
La mise à disposition des divers artefacts applicatifs facilite l'urbanisation et la sécurité du SI :

- Gestion centralisée des accès ;
- Gestion unifiée des différents modules, bibliothèques, et dépendances.

GESTION DES VERSIONS DES MODÈLES

A l'instar de la gestion des versions des sources & artefacts, il est important de procéder de même pour les modèles. L'intérêt principal est identique, à savoir pouvoir (ré-)exécuter un ancien modèle **facilement**. Outre cet aspect, il est intéressant de pouvoir se servir de différentes versions d'un même modèle en **simultané**. Charge à l'application de sélectionner la version du modèle qu'elle souhaite solliciter. C'est un élément essentiel pour automatiser la mise en production des nouveaux modèles publiés au sein du gestionnaire de code source (i.e. **trigger sur un push GIT**) . La **chaîne de CI/CD met automatiquement** à disposition chaque nouvelle version publiée sur la branche **develop** GIT. Chaque modèle est "monitoré", l'application compare les résultats des différents modèles. Dès lors qu'un nouveau modèle possède un meilleur score que le modèle actuellement en production, celui-ci devient alors le nouveau modèle en production. Ce principe est inspiré du mode de déploiement **Canary Release** que l'on rencontre dans l'univers des services web à très fort trafic. Avec l'avènement des containers, ce mode de déploiement est plébiscité par l'écosystème Kubernetes.

Dans l'écosystème ML/DS, ce concept n'est pas récent mais manque de maturité. Il y a encore peu de temps, il n'existait aucun framework Open Source en charge de cette problématique. Ce vide est sur le point d'être comblé par la percée fulgurante de **MLflow**. Il s'agit d'une solution Open Source éditée par **Databricks,** acteur de référence dans l'écosystème Big Data. Ce dernier est principalement connu pour éditer des solutions autour de la base de données **Cassandra**. En avril 2019, Microsoft intègre le projet MLflow et ajoute un support natif à ce produit au sein d'Azure ML. La version 1.0.0 est toute récente, en date du 13 juin 2019 !

Avec MLflow, les *Data Scientists* peuvent suivre et partager des expériences localement (sur un ordinateur portable) ou à distance (dans le *cloud*), regrouper et partager des modèles à travers des frameworks et déployer des modèles pratiquement partout.

MLflow est organisé en trois composantes : **Tracking, Projects et Models.** Chaque composant est utilisable séparément - par exemple, l'export des modèles au format de modèle de MLflow ne nécessite pas Tracking ou Projets - mais ils sont également conçus pour bien fonctionner ensemble.

La philosophie de base de MLflow est d'imposer le moins de contraintes possibles au *workflow* : il est conçu pour fonctionner avec n'importe quelle bibliothèque d'apprentissage machine, déterminer la plupart des choses sur le code par convention, et nécessiter un minimum de modifications pour s'intégrer dans une base de code existante. En même temps, MLflow vise à rendre reproductible et **réutilisable** par de multiples *Data Scientists* n'importe quelle base de code écrite dans son format.

Après de nombreuses réflexions concernant l'industrialisation et l'intelligibilité des modèles de Machine et Deep Learning, les projets data font aujourd'hui face à la problématique du maintien en conditions opérationnelles. C'est un sujet d'autant plus complexe qu'il implique la coordination entre plusieurs métiers (data scientists, data engineers, DSI, utilisateurs finaux) afin d'assurer des modèles robustes, un ROI effectif et une réactivité suffisante en cas de problème.

La recherche autour des problématiques de sensibilité et de robustesse des modèles est en pleine expansion, de même que les outils de versionning et monitoring. Chez Quantmetry, nous sommes convaincus qu'il est crucial pour les entreprises de mettre en place des standards à la fois techniques, méthodologiques et organisationnels pour faire face à cet enjeu. Ainsi, nous avons créé un pôle d'expertise R&D afin d'approfondir ce sujet et vous proposer des éléments de réponse.

Ce livre blanc aborde ce sujet sous ces trois axes, proposant des éléments de réponse théoriques et concrets, complétés de retours d'expérience variés d'entreprises et de chercheurs.

Bonne lecture !