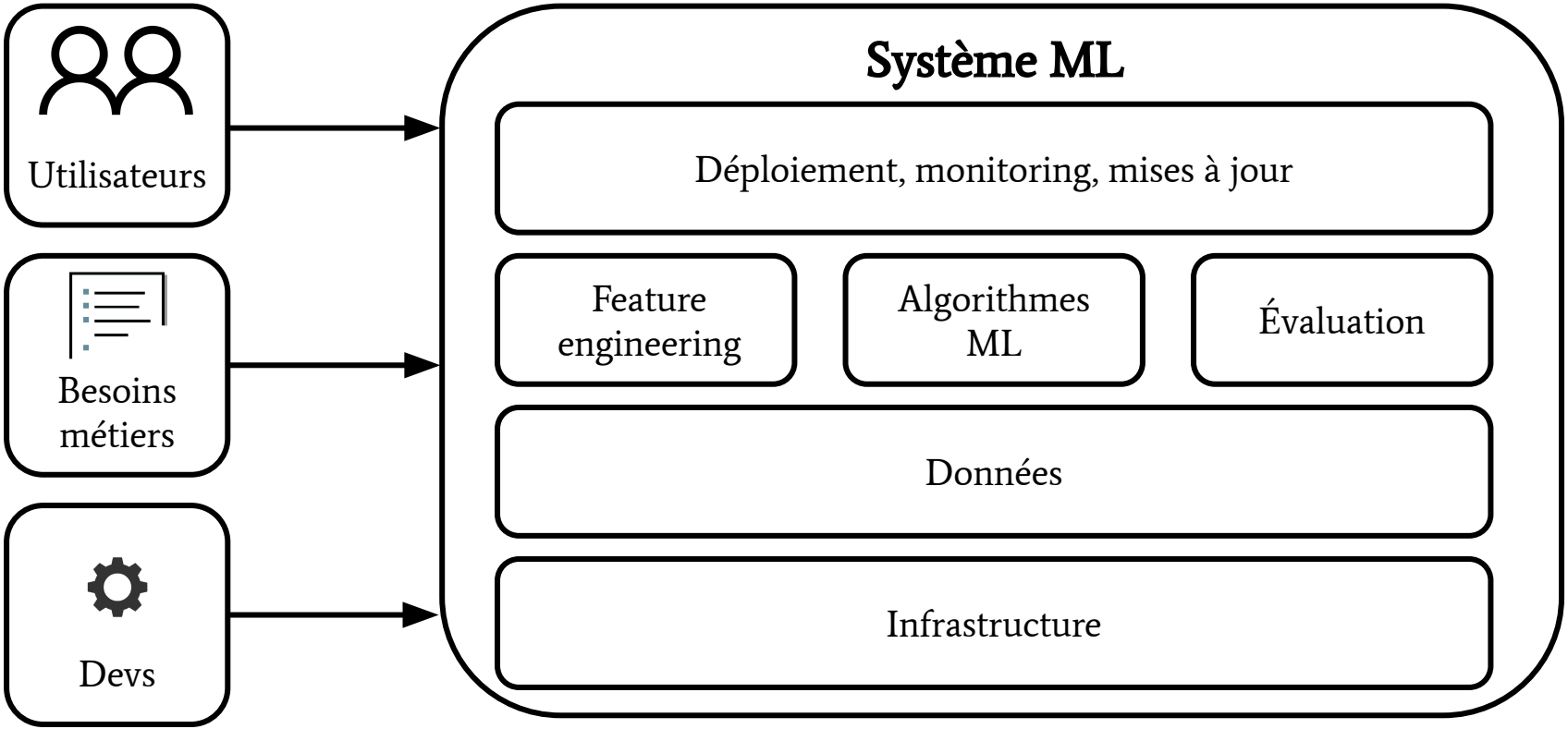
Le design des systèmes de machine learning

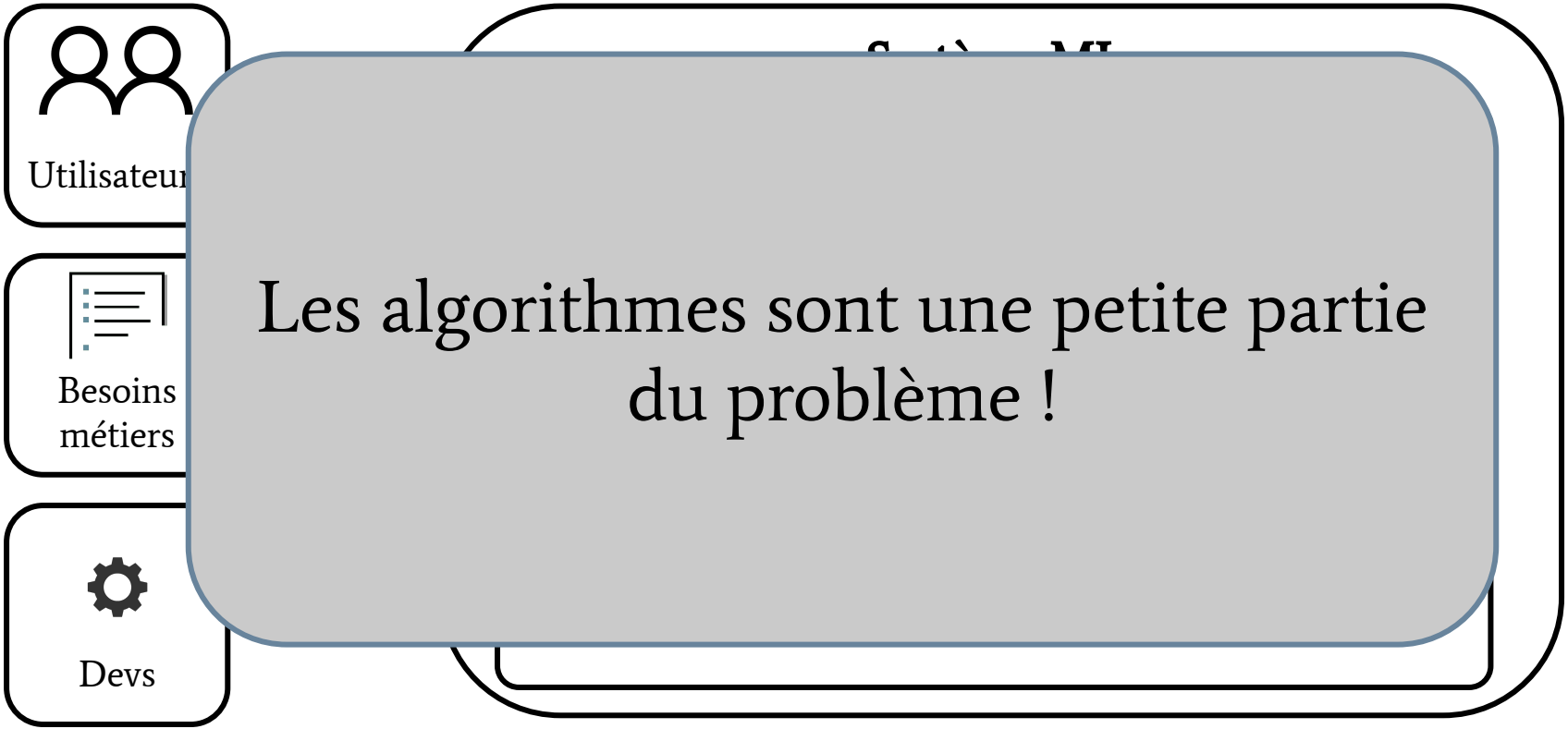
Julien Romero

Aperçu des systèmes de machine learning

Machine learning en production



Machine learning en production



Il faut bien définir le problème avant de s'engager dans l'implémentation

Quand utiliser du machine learning ?

- Il doit y avoir quelque chose à **apprendre**
  - Pas une base de données
  - Il faut des données pour apprendre, des entrées et sorties claires (ex: prédiction prix maison)
- Les motifs à apprendre sont **complexes**
  - Simple = algorithmique classique (tri, if/else, accès base de données)
- Il faut des **données**
  - On peut les obtenir ou les collecter
  - On n'entraîne pas de modèle sans données
- Il faut faire des **prédictions...**
- ... sur des données **nouvelles**
  - Il doit y avoir une relation entre les données d'entraînement et les données en pratique (distributions similaires)

Tâches parfaites pour le machine learning

- Répétitives
  - Faciles à apprendre
- Le coût d'une erreur est faible
  - Prédictions jamais parfaites
- Passage à l'échelle
  - Faire pleins de prédictions
- Les motifs changent au fil du temps
  - Pas possible de hardcoder la solution

Quand ne pas utiliser du machine learning ?

- Pas éthique
- Des solutions plus simples font l'affaire
- Pas rentable

Machine learning dans la recherche vs en production

	Recherche	**Production**
Objectif	Modèles au niveau de l'état de l'art sur des benchmarks	Dépend à qui on demande (ingénieur, vendeur, équipe produit, DevOps)

Des objectifs différents

- Une application qui recommande des restaurants et prend 10% de la vente
  - Ingénieur ML : Meilleure recommandation
  - Ventes : Restaurants les plus chers rapportent plus
  - Équipe produit : Rétention client avec une faible latence
  - DevOps : Doit pouvoir passer à l'échelle
  - Manager : Maximiser le profit (en virant des ingénieurs ?)
- On ne peut pas satisfaire tous les besoins

Machine learning dans la recherche vs en production

	Recherche	**Production**
Objectif	Modèles au niveau de l'état de l'art sur des benchmarks	Dépend à qui on demande (ingénieur, vendeur, équipe produit, DevOps)
Puissance de calcul	Entraînement rapide, large bande passante (batch)	Inférence rapide, latence faible
Données	Statiques, propres	En mouvement perpetuel, bruitées
Équité	Rarement considérée	Doit être prise en compte
Interprétabilité	Rarement considérée	Doit être prise en compte

Systèmes ML vs génie logiciel

- Séparation code/données en génie logiciel
  - Code, données, et artéfacts sont très liés en ML
  - Souvent, on améliore les données avant le code
- Test et versionnage du code en génie logiciel
  - + test et versionnage des données en ML
- Modèles ML lourds
  - Déploiement rapide ?
  - Monitoring ? Debugging ?

Introduction au design des systèmes pour le machine learning

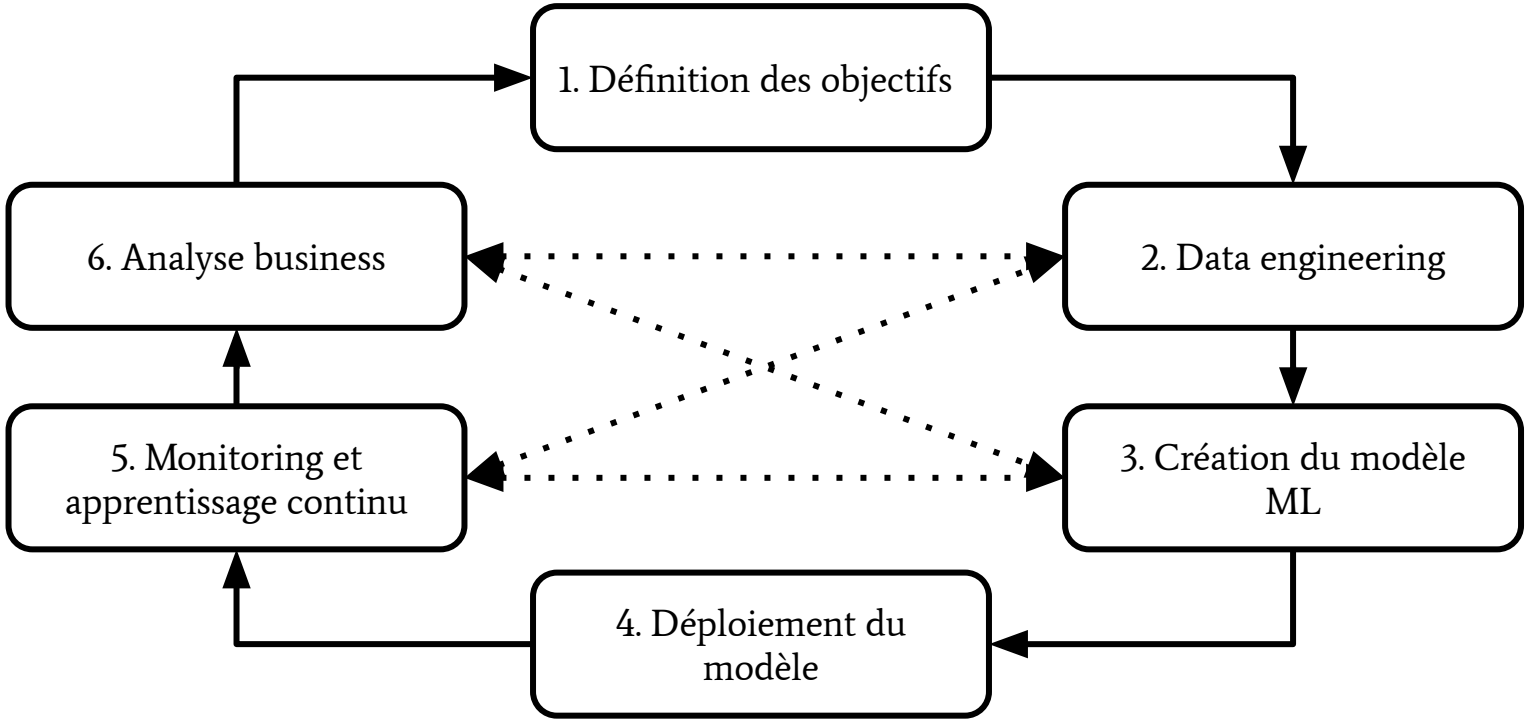
Divergences des métriques

- En ML, on cherche à augmenter des métriques bien définies
  - Précision, Recall, F1, RMSE, AUC, ...
- En pratique on veut augmenter le profit
  - Relation peu claire avec les métriques de ML
  - ML n'est qu'une part d'un grand système
  - Influence directe : augmentation des ventes, réduction des coûts
  - Influence indirecte : satisfaction client, temps passé sur le site

Exigences pour un système ML

- Fiabilité
  - Continue de fonctionner même en cas de problème hardware, software ou humain
  - Qu'est-ce qu'une erreur dans un modèle de ML ? Mauvaise prédiction ?
- Scalabilité
  - Taille du modèle, traffic, nombre de modèles
  - Automatisation du monitoring et de l'entraînement
- Maintenabilité
  - Versionage, documentation, reproducibilité
- Adaptabilité
  - Continue de fonctionner même quand on change les données ou les besoins
  - Possible d'améliorer le système

Le cycle de développement d'un système ML



Définir le problème ML

- On veut une définition en terme d'entrées et sorties claires
  - Différent besoin business
  - Ex : On veut augmenter l'efficacité d'un centre d'appel. Après inspection, problème dans la répartition des demandes aux différents services. Problème ML = entrée : texte demande, sortie : service concerné
- Il faut définir le type de problème et la fonction de coût optimisée
  - Régression, classification (binaire, multi-classes, multi-labels)
  - Attention à bien choisir ! Ex : prédiction de la prochaine application utilisée, multi-classes (1 par application) vs binaire (features utilisateur + application -> score)
  - Transformer les besoins business en fonction de coût

Les fonctions de coûts

- Les modèles de ML tentent d'optimiser une fonction de coût
  - Dépend du type de problème et de la sortie attendue
- En pratique, on veut optimiser plusieurs objectifs à la fois
  - Ex: Je veux recommander des contenus de qualités et sur lequel un utilisateur a une forte probabilité de cliquer
  - Pas forcement corrélés
  - Le loss devient α*loss_quality + β*loss_click, où α et β sont des hyperparamètres à régler
- Comment optimiser les composantes du coût ?
  - Ensemble, avec un seul modèle
    - Rapide mais demande un réentraînement à chaque changement de α et β
  - Séparément
    - Plus flexible, mais demande plus de ressources en général

Data Engineering

Sources de données

- Entrées des utilisateurs (explicites)
  - Texte, audio, vidéos, fichiers, clicks, pages visitées, ...
  - Souvent très bruitées, avec des erreurs.
  - À traiter rapidement
- Données générées par le système
  - Logs, prédiction des modèles
  - Peuvent être traitées plus tard, de façon périodique
- Bases de données internes
  - Stocks, liste des clients, liste des produits, ...
- Données externes (third-party)
  - First-party : Données de notre entreprise sur nos clients
  - Second-party : Données d'une autre entreprise sur ses clients
  - Third-party : Données d'une autre entreprise sur des utilisateurs qui ne sont pas ses clients
  - Souvent, données de tracking pour la pub

Formats de données

Format	Binaire/Texte	Lisible humain ?	Exemple utilisation
JSON	Text	Oui	Partout
CSV	Text	Oui	Partout
Parquet	Binaire	Non	Hadoop, Amazon Redshift
Avro	Binaire	En partie	Hadoop
Protobuf	Binaire	En partie	Google, TensorFlow
Pickle	Binaire	Non	Python, Pytorch

Modèles de données

- Description de comment les données sont représentées
  - Dépend de l'usage
- Modèle relationnel
  - Organisation en tables, lignes et colonnes
- NoSQL
  - Stockage clef/valeur, base de données graphs, base de données de documents, base de données tabulaires
- Données structurées vs non-structurées
  - Structurées = schéma prédéfini, difficile à changer mais facilitant l'utilisation
  - Non-Structurées = text, image, vidéo, audio, liste de nombres, ...

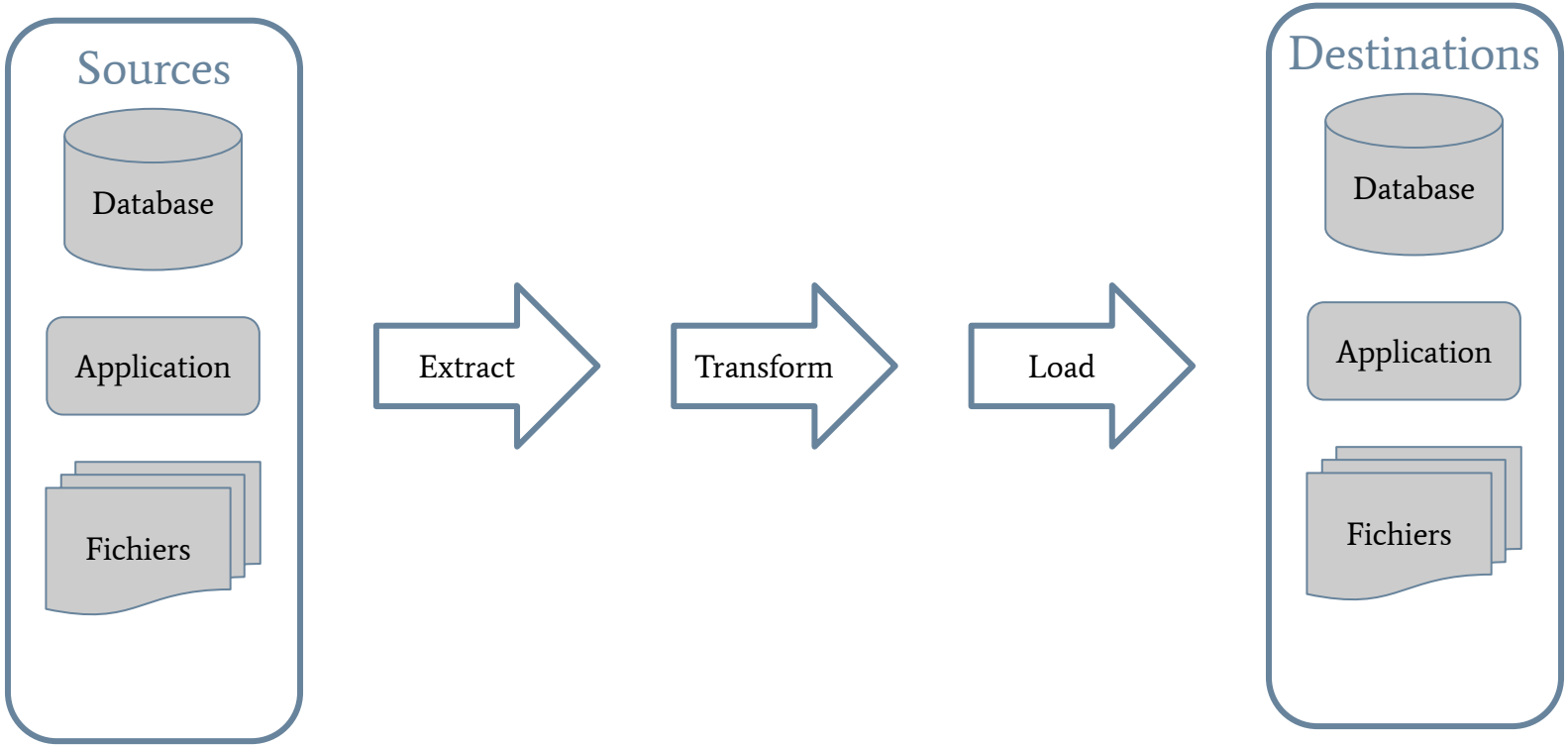
Données structurées vs non-structurées

Structurées	**Non-structurées**
Schéma prédéfini	Pas de schéma obligatoire
Recherche et analyse facile	Rapide à générer et à ingérer
Seulement pour les données avec le bon schéma	Pour les données venant de n'importe où
Changer le schéma est difficile et coûteux	Quand le contenu change, le stockage ne change pas, mais les applications doivent s'adapter
Entrepôts de données	Lac de données

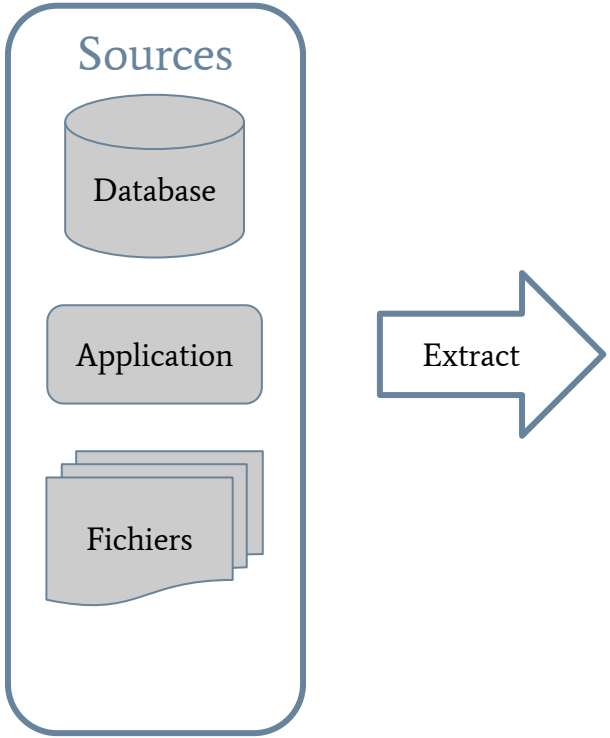
Traitement transactionnel et analytique

- Transaction = échange d'informations cohérentes entre l'utilisateur et le service
  - Ex: écrire un post Linkedin, regarder une vidéo Youtube, soumettre son DM sur Moodle
  - Online Transaction Processing (OLTP)
  - Besoin d'un traitement rapide
  - Souvent associé avec ACID (Atomicité, Cohérence, Isolation, et Durabilité)
- Analytique = traitements lents, demandant d'agréger des données
  - Ex: Calculer la moyenne des ventes sur le dernier semestre
  - Online Analytical Processing (OLAP)
- De moins en moins de séparation entre les deux types de fonctionnements
  - Certains bases de données sont meilleures sur une catégorie que sur l'autre

ETL : Extract, Transform, Load

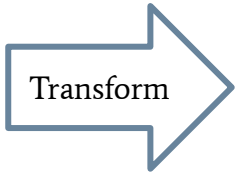


ETL : Extract, Transform, Load



Extraction des données depuis des sources quelconques
On sélectionne et valide les données importantes.

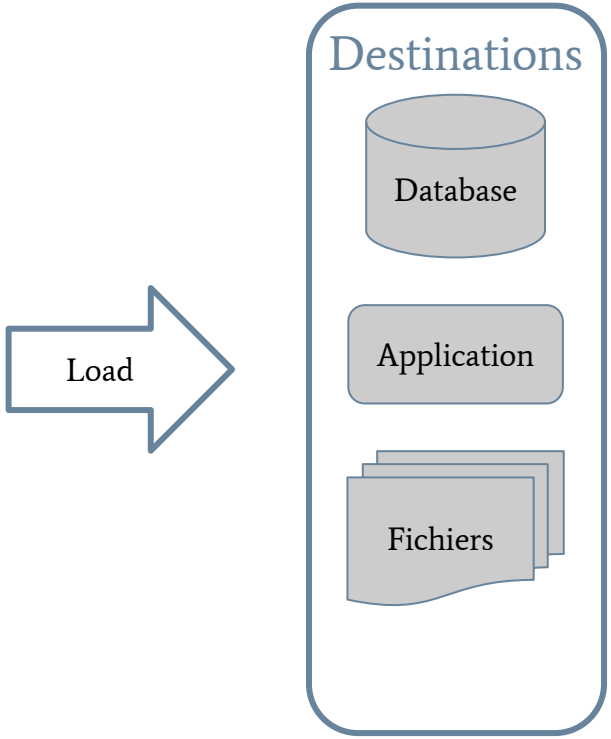
ETL : Extract, Transform, Load



Transformation = Nettoyage et pré-processing des données
Normalisation, déduplication, création de features, ...

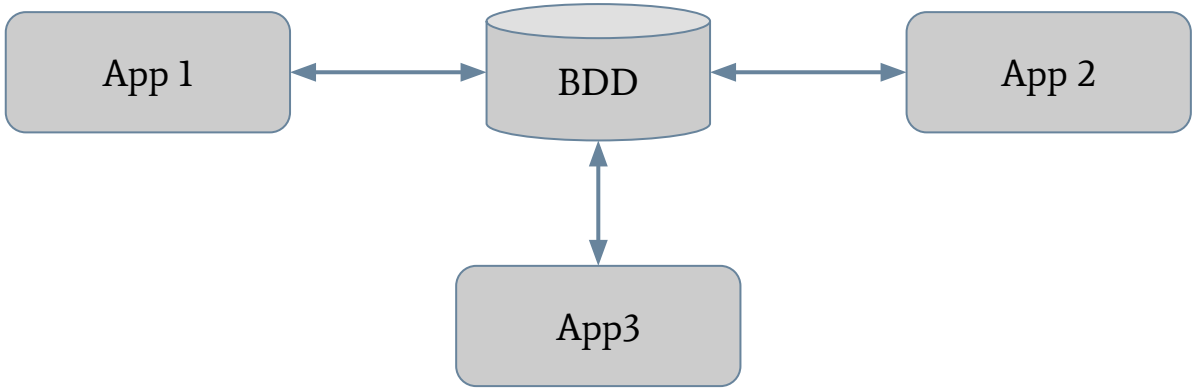
ETL : Extract, Transform, Load

Load = Stratégie de stockage des données



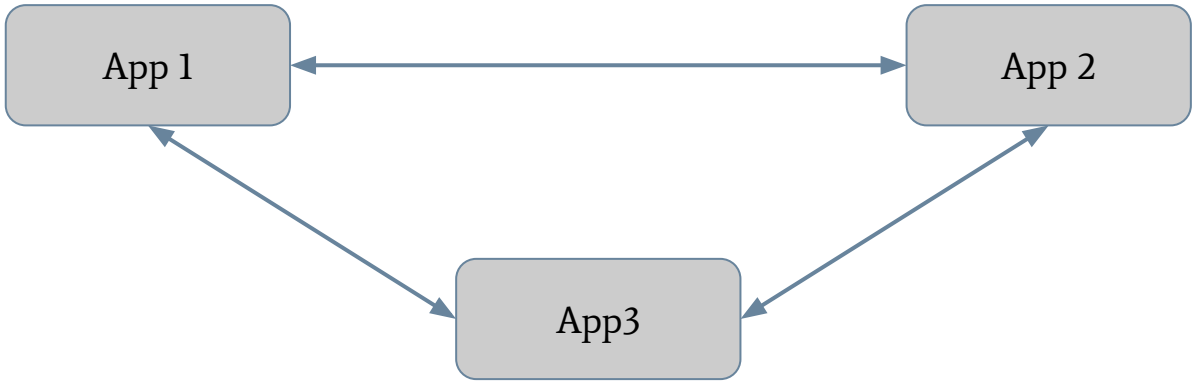
Transferts de données

- À travers une base de données
  - Lectures et écritures lentes
  - Pas de communications hors de l'entreprise



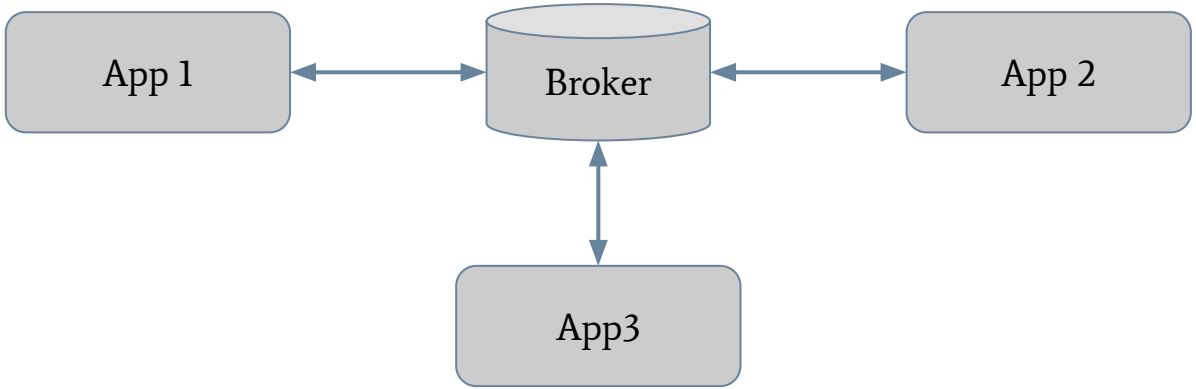
Transferts de données

- Communication directe entre les services
  - Plus rapide
  - Inter-entreprise
  - REST et RPC
  - Difficile à gérer avec beaucoup de services, gestion des erreurs à la main, inter-blocages



Transferts de données

- Système de messages temps réel
  - Publish-subscribe : Kafka, RabbitMQ
  - Un broker central en charge de passer les messages
  - Broker = base de données spécialisée dans le temps réel



Traitement en batch ou en ligne (streaming)

- Batch = Calcul longs, sur beaucoup de données
  - Hadoop, Spark
- Streaming = Données traitées en temps réels (ou sur des fenêtres courtes)
  - Temps de latence faibles
  - Spark Streaming, Apache Flink, Kafka
- Certaines features demandent soit un type de traitement, soit un autre.

Données d'entraînement

Échantillonnage

- On ne peut pas utiliser toutes les données, il faut choisir ce qui est important
  - Quelles données je veux obtenir ? Facile d'accès ?
  - Est-ce que je peux utiliser toutes les données accessibles ou serait-ce trop long ?
  - Comment choisir les données à utiliser ?
- Échantillonnage = prendre un sous-ensemble des données

Échantillonnage non probabiliste

- On suit une procédure non-aléatoire pour sélectionner des données
- **Échantillonnage par commodité** : on prend ce que l'on peut
- **Échantillonnage boule de neige** : on utilise les échantillons actuels pour trouver les suivants. Ex: le web crawling
- **Échantillonnage par des experts** : des experts choisissent les données à utiliser
- **Échantillonnage par quota** : on collecte un nombre de données fixe dans plusieurs catégories. Ex: sondage par groupe d'âge

En suivant ces méthodes, on obtient souvent des biais dans les données, mais parfois nous n'avons pas le choix. Ex: LLM entraîné sur le web qui est très bruité, labellisation automatique de sentiments à partir des notes (uniquement informations sur les gens qui donnent une note)

Échantillonage probabiliste

- **Échantillonnage aléatoire** : Toutes les données possibles ont la même probabilité d'être utilisé. Classes rares n'apparaissent pas.
- **Échantillonnage stratifié** : On sépare les données en groupe, et dans chaque groupe on fait un échantillonnage automatique. Pas toujours possible ou groupes pas clairs
- **Échantillonnage pondéré** : On donne des poids à chaque échantillon. Plus le poids est élevé, plus on a de chance de sélectionner le poids. Ex: privilégie les données récentes, utilisation d'experts
- **Échantillonnage par réservoir** : Algorithme pour faire de l'échantillonnage sur un flux de données
- **Échantillonnage par importance** : Algorithme pour échantillonner à partir d'une distribution trop difficile à calculer à partir d'une distribution plus simple. Utile pour l'apprentissage par renforcement

Étiquetage des données

Comment obtenir des étiquettes pour entraîner des modèles supervisés ?

- **Étiquetage à la main**
  - Bonne qualité
  - Cher, il faut parfois des experts (Ex: identification cancer)
  - Problèmes de vie privées : on ne peut pas partager ses données avec une personne quelconque
  - Lent
  - Changement de la tâche = on recommence
  - Conflits entre plusieurs annotateurs (réponse pas claire ou tâche mal définie), voir le Kappa de Cohen
  - Il faut se rappeler d'où viennent les données (data lineage) pour éviter de mélanger des données mal annotées avec des données bien annotées

Étiquetage des données

Comment obtenir des étiquettes pour entraîner des modèles supervisés ?

- **Étiquetage naturel**
  - On peut obtenir un label à partir des données elles-mêmes
  - Ex: Prédiction du cours de la bourse, systèmes de recommandation, retours utilisateurs (like/dislike)
  - Toutes les étiquettes n'ont pas la même importance : clic, mise au panier, achat

Comment palier au manque d'étiquettes ?

- **Weak supervision** : On utilise des règles et algorithmes classiques pour annoter les données, puis on entraîne un modèle
  - On espère que le modèle se généralise mieux que les règles
  - Rapide, peu cher, pas de problème de vie privée, réutilisable
- **Semi-supervision** : Utiliser des données annotées existantes pour annoter d'autres données
  - Self-training : On entraine un modèle, qu'on utilise pour labelliser des données, et on recommence
  - Propagation des étiquettes aux données les plus proches
  - Perturbation (petite) des données pour créer de nouvelles données avec le même étiquette
- **Transfer learning**
- **Active learning :** Méthodes pour choisir quels sont les points les plus importants à étiqueter

Déséquilibre entre les classes

- Quand les proportions de chaque classes ne sont pas identiques
  - Ex: Identification de cancers, estimation de maisons de luxe (régression)
- Difficulté d'apprentissage pour les modèles de machine learning
  - Pas assez d'informations sur la classes sous-représentées
  - Local optimum si on prédit la classe majoritaire (si 99% de non-cancer, 99% d'accuracy)
  - Le coût d'une erreur n'est pas le même entre toutes les classes (cancer plus grave)
- Sources : naturelles, artificielles (labellisation de spams, mais les spams ont déjà été filtrés), ou humaines (mauvaise annotation avec l'oubli d'une classe)

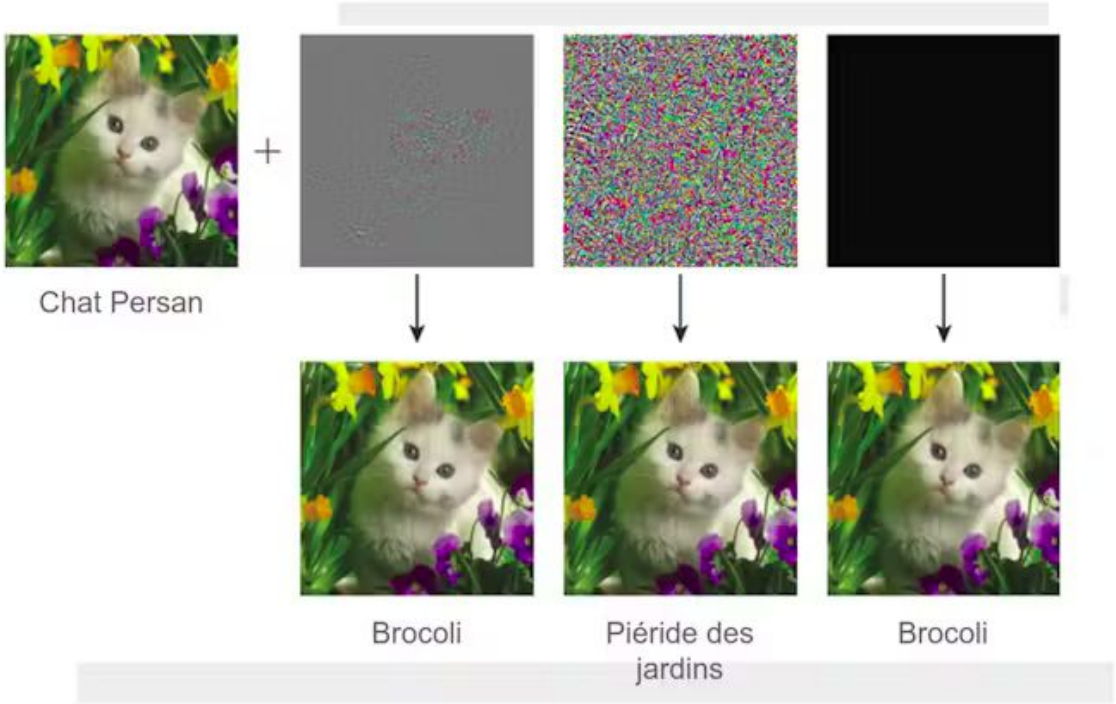
Comment résoudre les déséquilibres ?

- Adaptation des métriques
  - Se concentrer sur les classes positives : Recall, F1, AUC
- Rééchantillonnage
  - Sous-échantillonnage : On réduit la classe majoritaire (aléatoire, Tomek links)
  - Sur-échantillonnage : On augmente la classe minoritaire en créant artificiellement de nouveaux points (voir SMOTE)
  - Attention ! On n'évalue pas sur des données rééchantillonnées !
- Adaptation des algorithmes d'apprentissage
  - Modification de la fonction de coût, avec un poids différent suivant l'erreur

Augmentation de données

- **Création de nouveaux points de données à partir d'anciens**
  - But: Augmenter performances et résistance au bruit
- **Transformations préservant les étiquettes**
  - Image : recadrage, redimension, rotation, changement couleurs, ...
  - Texte : remplacement de mots par des synonymes
- **Perturbations** : Modifier les entrées sans changer les étiquettes
  - Voir apprentissage antagoniste

Augmentation de données



Augmentation de données

- **Création de nouveaux points de données à partir d'anciens**
  - But: Augmenter performances et résistance au bruit
- **Transformations préservant les étiquettes**
  - Image : recadrage, redimension, rotation, changement couleurs, ...
  - Texte : remplacement de mots par des synonymes
- **Perturbations** : Modifier les entrées sans changer les étiquettes
  - Voir apprentissage antagoniste
- **Synthèse de données**
  - Texte : utilisation de templates, demander à un LLM
  - Image : Mix d'images, génération avec diffusion ou GAN

Feature Engineering

Qu'est-ce que le feature engineering ?

Feature engineering = sélection, transformation et adaptation des entrées d'un modèle de machine learning

- Amélioration des performances
  - Souvent plus que jouer sur des hyperparamètres
- Le deep learning n'est-il pas suffisant ?
  - Pour du texte brut et des images brutes, souvent
  - Pour le reste, on doit parfois sélectionner des features parmi de millions de possibilités, et les adapter (données tabulaires)
  - Ex: Classificateur de spam sur un forum prenant en entrée des données supplémentaires (nombre de like, dislike, informations sur l'utilisateur, données sur le nombre de vues)

Opérations courrentes

- Données manquantes
  - Raisons : Non aléatoire (utilisateur ne veut pas renseigner son salaire car trop élevé), partiellement aléatoire (une catégorie d'utilisateur a une probabilité plus forte de ne pas renseigner un champs), totalement aléatoire (pas de motif)
  - Solutions : retraits des colonnes ou des lignes, remplacement (valeur par défaut, moyenne, médiane)
- Mise à l'échelle : toutes les features doivent avoir la même échelle
  - Convergence plus rapide, meilleures performances
  - Attention au data leakage et changements de distributions entre train et test
- Discretisation : Création d'une variable catégorielle à partir d'une variable continue
- Encodage des features catégorielles
  - One-hot encoding, ordinal encoding
  - Quand beaucoup de catégories, utiliser le hashing trick = l'index d'une catégorie est son hash modulo la taille de l'encodage (mais il y a des collisions)

Opérations courrentes

- Feature crossing : combiner plusieurs features en une seule
  - Encode des relations non linéaires
- Techniques avancées comme les Fourier features qui permettent de créer des embeddings pour n'importe quel vecteur.

Qu’est-ce que le data leakage

Quand une information sur l’étiquette non disponible à l’inférence se retrouve dans le dataset d’entraînement et de test.

Ex :

- Les patients atteints d’un cancer utilisent une autre machine
- Apprendre à reconnaître les bateaux de pêche plutôt que les poissons

Les causes d'un data leakage

- Données de séries temporelles coupée aléatoirement au lieu de temporellement
- Mise à l'échelle avant le découpage train/val/test
- Valeurs manquantes calculée en utilisant aussi le test set
- Duplication de données entre le train et le test
- Group leakage : données fortement corrélés séparées dans le train et test
  - Ex: Deux radios d'un même patient, photos prises l'une après l'autre
- Fuite à cause du procédé de génération
  - Utilisation de machine différentes pour les scanners

Très important de comprendre d'où viennent les données et comment elles sont générées

Détecter les data leakage

- Est-ce qu'une ou plusieurs features semblent permettre de donner des résultats très bons ?
  - Vérifier les corrélations trop élevées
- Faire une étude par ablation : réentraîner le modèle en enlever une ou plusieurs features
  - Si enlever une feature dégrade trop les performances, étudier pourquoi
- Toujours rester suspicieux quand ajouter une feature améliorer beaucoup les performances
- Faire très attention quand on utilise le test set

Pourquoi limiter le nombre de features ?

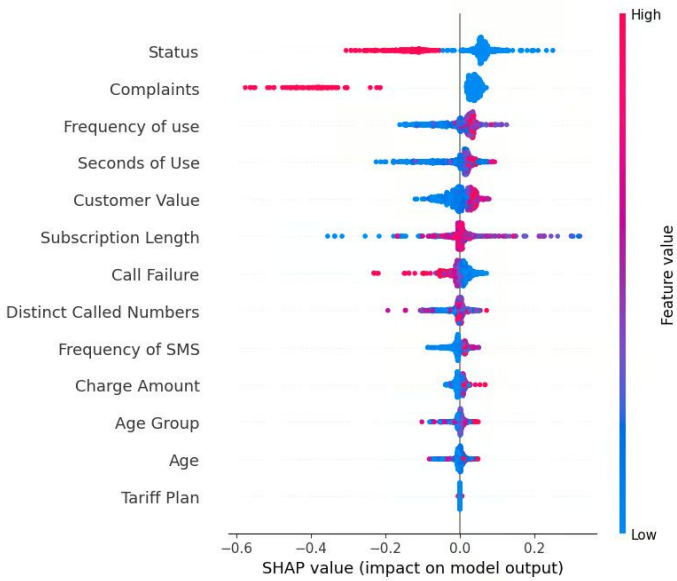
- Plus de risques de data leakage
- Plus de risques d'overfit
- Demande plus importante de ressources
- Latence pour le calcul des features
- Il faut maintenir les features à chaque changement de l'application
  - Ex: que se passe-t-il si on retire l'âge des informations à renseigner, mais des features en ont besoin ?

Importance des features

Comment évaluer l'importance d'une feature ?

- Utiliser une fonction dédiée pour un modèle (par exemple, dans XGBoost)
- Afficher les valeurs SHAP
  - Permet de mesurer l'impact positif ou négatif d'une feature sur le modèle

Souvent peu de features permettent d'obtenir de bons résultats, le reste donne de faibles améliorations



Évaluer la généralisabilité d'une feature

Généralisabilité = la feature sera utile à l'inférence

Ex: pour la détection de spam, ID du post non généralisable, ID de l'utilisateur oui

Heuristiques pour évaluer la généralisabilité :

- Pas trop de valeurs manquantes
- Les valeurs sur le train ont l'air similaires aux valeurs du test
    - Ex: Apparition d'une nouvelle catégorie dans le test

En conclusion

- Avant d'intégrer une solution de machine learning, il faut bien définir le problème et les attentes
- La gestion des données est un aspect crucial des systèmes de machine learning
- On doit penser :
  - Au format des données
  - La source des données
  - Le stockage des données
- Il faut être très prudent en créant les données d'entraînement et les features