Le design des systèmes de machine learning 2

Julien Romero

Développement de modèles et évaluation offline

Évaluer et choisir un modèle

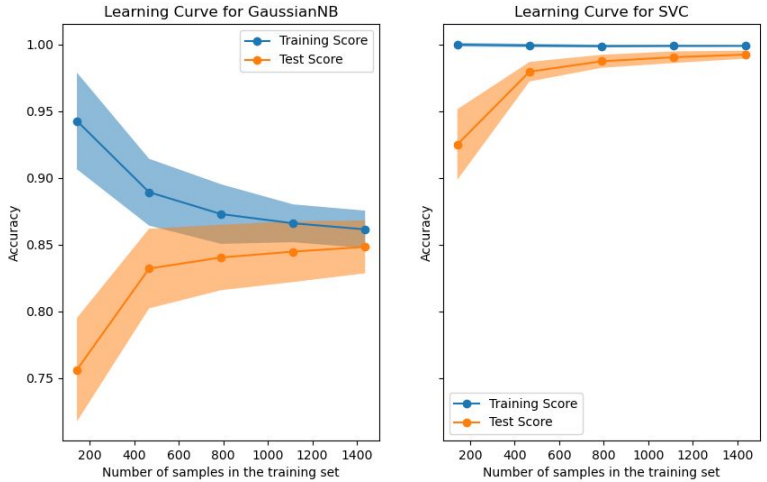
- Avec des ressources illimitées, on pourrait tout tester
  - Impossible en pratique
- Bien comprendre son problème et connaître les tâches classiques en ML est crucial
  - Détection de tweets toxiques -> classification de texte -> transformers, RNN, naive Bayes, régression logistique
  - Détection de transactions frauduleuses -> détection d'anormalités -> k-NN, isolation forest, clustering
- Les métriques de performances ne sont pas les seules à prendre en compte :
  - Quantité de données, ressources, et temps nécessaires pour l'entraînement
  - Latence à l'inférence
  - Interprétabilité

Comment sélectionner des modèles ?

- Faire attention aux derniers modèles à la mode
  - Souvent meilleurs sur des datasets très précis
  - Plus chers et complexes à mettre en place
- Commencer par des modèles simples
  - Rasoir d'Ockham
  - Facile à déployer, valider, comprendre, débugger
  - On ajoute des éléments petit à petit
  - Servent de baselines pour savoir si on progresse
- Éviter les biais humains
  - Excitation par un modèle => plus de tests => de meilleures performances
  - Il faut être équitable entre chaque configuration

Comment sélectionner des modèles ?

- Prévoir qu'un modèle peut s'améliorer
  - Au début, peu de données
  - Learning curves : performance du modèle en fonction du nombre de points d'entraînement



Comment sélectionner des modèles ?

- Évaluer les compromis
  - On ne peut pas être parfait partout
  - Faux positifs vs faux négatifs (biométrie vs détection COVID)
  - Ressources de calcul vs performance
  - Interprétabilité vs performance
- Quelles sont les hypothèses des modèles ?
  - Prédiction possible, variables indépendantes et identiquement distribuées, continuité (des entrées proches ont des sorties proches), tractabilité, existences de frontières (classification linéaire), indépendance conditionnelle (naive Bayes), loi normale, ...

Méthodes ensemblistes

- Idée : Combiner plusieurs modèles
- Permettent de gagner un peu de performance
- Mais plus difficile à déployer et maintenir
- Types de méthodes ensemblistes :
  - Bagging : Création de plusieurs datasets différents par échantillonnage et entraînement d'un modèle pour chaque dataset
  - Boosting : Entraînement itératif où les nouveaux modèles se concentrent sur les erreurs précédentes
  - Stacking : Entraînement de plusieurs modèles différents que l'on combine

Suivi et versionnage des expériences

- Réplicabilité cruciale : on veut pouvoir recréer un artéfact de manière sûre
  - Artéfact = fichier généré pendant l'entraînement (loss curve, évalutations, logs, paramètres du modèle, ...)
- Suivi d'expérience = Au niveau d'une seule expérience, comprendre ce qui se passe
- Versionnage d'expérience = logger les détails des expériences pour les rendre reproductibles et comparer les résultats de plusieurs expériences

Suivi d'expérience

- Nous voulons suivre :
  - La courbe de loss sur le train et le val
  - Les métriques de performance (F1, RMSE, accuracy, ...) sur le train et le val
  - Les entrées, les sorties du modèle, et la vraie prédiction : utile pour analyser plus tard
  - Vitesse du modèle : itérations par secondes, tokens par secondes, ...
  - Métriques du système (utilisation CPU, GPU, mémoire)
  - Paramètres et hyperparamètres qui changent : learning rate, la norme des gradients, la normes des poids, ...

Versionnage d'expérience

- Versionnage du code assez commun (Git)
- Quid des données ?
  - Plus larges, modifications difficiles à calculer et peu efficaces
  - Impossible de multiplier un dataset, ne tient pas forcément sur une seule machine
  - Lois (RGPD) demandent de supprimer des données
- Pourtant crucial pour la reproductibilité
  - Mais souvent ignoré

Comment débugger un modèle de ML ?

- Difficile :
  - Pas d'erreur : le modèle continue de faire des prédictions, mais fausses
  - Long de vérifier si le bug est corrigé (il faut ré-entraîner, re-déployer, ...)
  - De nombreuses composantes (et équipes) : données, étiquettes, features, algorithme, code, infrastructure, ...
- Sources :
  - Contraintes théoriques (voir plus haut)
  - Erreur d'implémentation
  - Mauvais choix des hyperparamètres
  - Problème avec les données (collecte, préprocessing, mauvaise normalisation, ...)
  - Mauvais choix de features

Comment éviter les bugs ?

(Similaire à la procédure du cours de DL)

1. Commencer simple et ajouter des composantes petit-à-petit
2. Overfit sur peu de données du train
3. Fixer la seed : la seed contrôle l'aléatoire, la fixer permet de reproduire exactement les mêmes résultats, ou presque

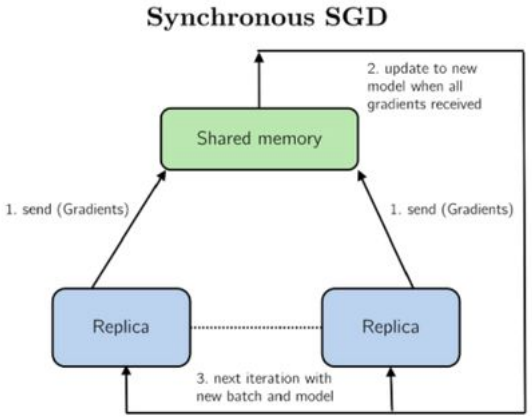
Entraînement distribué

- Nécessaire pour passer à l'échelle
  - Beaucoup de calculs
  - Données ne rentrent pas sur une machine
  - Batch size trop petite

Parallélisation des données

Idée : distribuer les données sur plusieurs machines, entraîner le modèle et accumuler les gradients

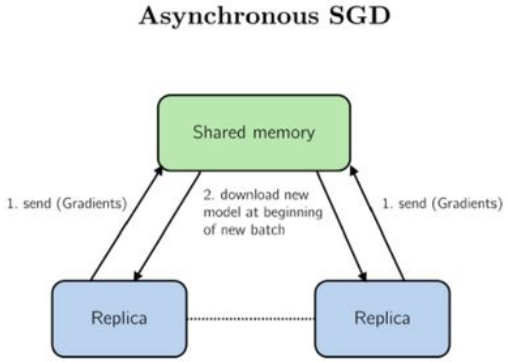
- **Synchronous stochastic gradient descent** : chaque machine calcule les gradients, on attend que tout le monde ait fini, on accumule les gradients et on fait la mise à jour
  - Lent
  - Ressources non utilisées pendant l'attente
  - Problème si une machine ne répond pas



Parallélisation des données

Idée : distribuer les données sur plusieurs machines, entraîner le modèle et accumuler les gradients

- **Asynchronous stochastic gradient descent** : chaque machine calcule les gradients, on accumule les gradients et on fait les mises à jour au fur et à mesure
  - Convergence demande plus d'itérations (ok en pratique)

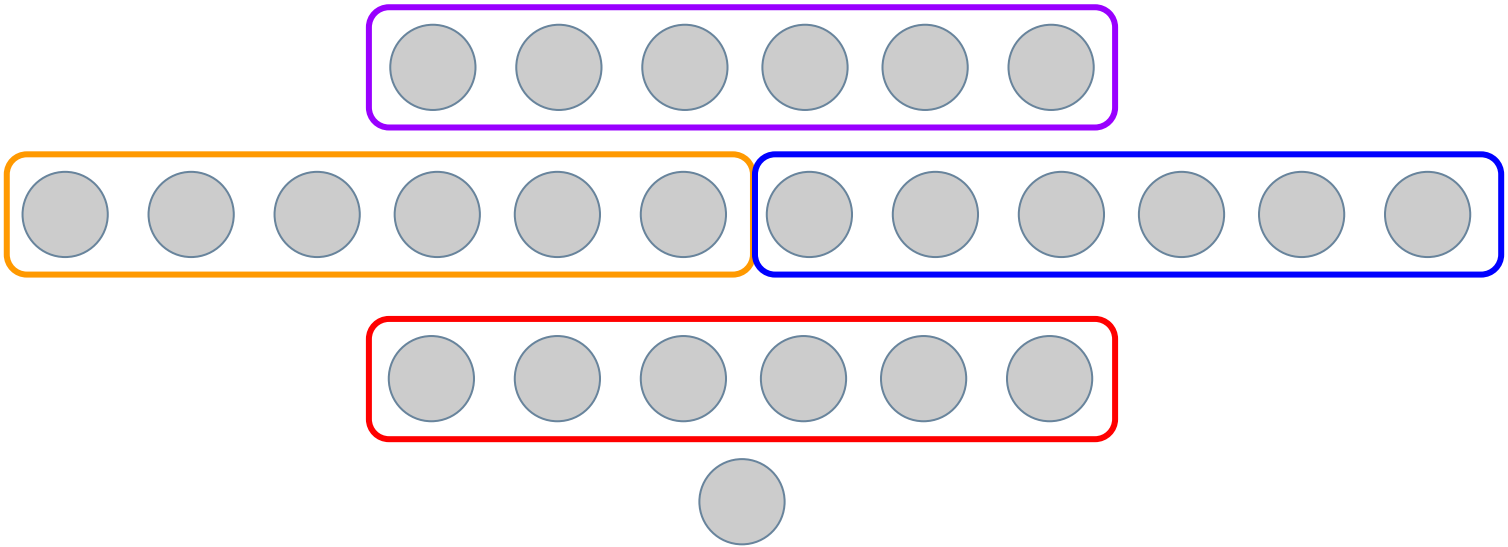


Problèmes de la parallélisation :

- Grandes batch size (=> moins d'itération dans une époque)
- Il faut du workload balancing

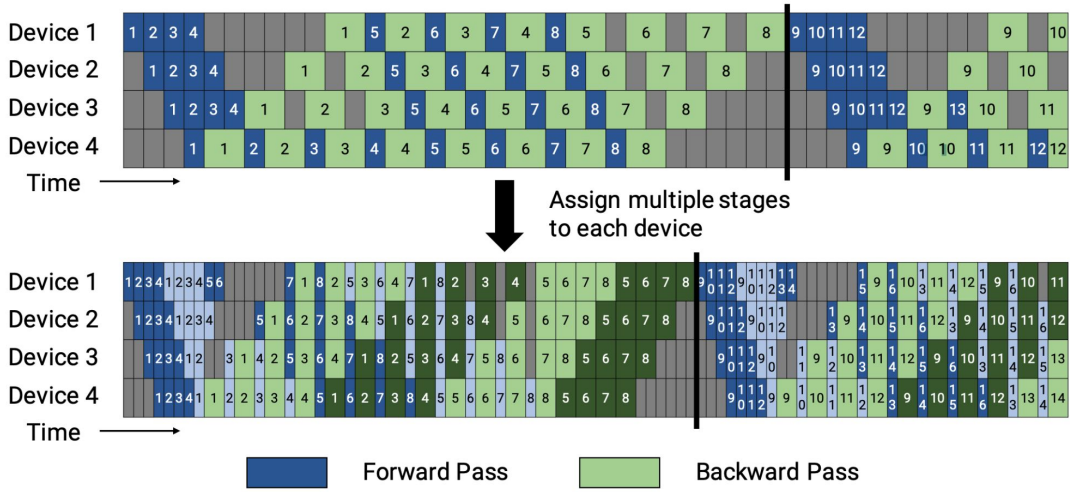
Parallélisation du modèle

- Idée : on entraîne les différentes parties du modèles sur les machines différentes



Parallélisation du modèle

- Pipeline de parallélisation : pour ne pas perdre de ressource de calcul, il faut suivre une pipeline permettant de paralléliser les calculs sur plusieurs machines, sans tout le temps attendre



AutoML

- Processus automatique pour trouver un algorithme de ML pour résoudre un problème
  - Trouver les hyperparamètres (random search, grid search, optimisation bayesienne)
    - Rappel : On n'utilise jamais le test set pour trouver les hyperparamètres
  - Trouver l'architecture/le modèle : Neural Architecture Search
    - Il faut un ensemble de modèles possibles, une métrique d'évaluation des métriques, et une stratégie d'exploration (aléatoire, apprentissage par renforcement, algorithme génétique, ...)

Les quatre phases du développement d'un modèle de ML

1. Avant le ML, implémenter une solution sans ML
   a. Algorithme simple, heuristique
   b. Ex: Système de recommandation de news montrant les plus récentes
2. Algorithme simple de ML
3. Optimisation de l'algorithme simple
4. Exploration des algorithmes plus complexes

Évaluation des modèles offline

- Dans l'idéal, même évaluation online et offline
  - Ok avec des labels naturels (mais avec des biais)
  - Pas d'étiquette en online
- Baseline : Il faut toujours comparer son approche à des baselines pour montrer son utilité
  - Baseline aléatoire : on retourne une classe de manière aléatoire
  - Heuristique simple : algorithme simple
  - Classe la plus utilisée : globalement ou au niveau d'un individu/groupe
  - Baseline humaine : nous donne un objectif de ce qui est possible de faire
  - Solution existante

Méthodes d'évaluation avancées

À ajouter au test set en général

- Tests de perturbation : Ajout d'un bruit pour tester si le modèle est robuste
  - Ex : classification d'appel, le dataset est propre mais en pratique bruit de fond d'un call center
- Tests d'invariance : Modification de paramètres qui ne sont pas censés changer la sortie
  - Ex : changer le sexe ou l'ethnicité

Calibration de modèle

On veut vérifier que les sortie du modèle correspondent bien à la réalité, en particulier quand on prédit une probabilité.

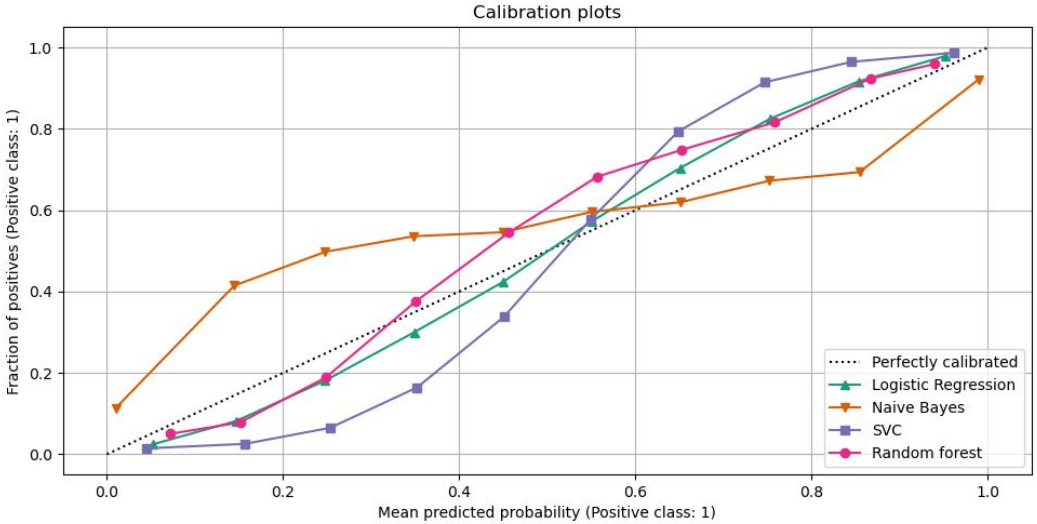
- Vérifier que quand un modèle prédit qu'un événement à X% de chance de se passer, il se passe vraiment X% du temps
  - Ex : Si on prédit qu'une équipe A bat une équipe B 80% du temps, après 100 matchs, on devrait avoir 80 victoires pour A
- Vérifier que la distribution de la sortie correspond à la réalité quand on considère plusieurs prédictions
  - Ex : Dans un système de recommandation, si un utilisateur regarde 80% de comédies romantiques et 20% de thrillers, le système de recommandation devrait suivre ces valeurs

Calibration de modèle

- On peut mesurer la calibration du modèle en comptant
- Pour calibrer un modèle, on peu utiliser le Platt scaling

  (A et B sont appris, f est le classifier)

$$P(y=1|x) = \frac{1}{1+exp(Af(x)+B)}$$



Évaluation sur des sous-ensembles

- Pour mieux comprendre les résultats, il est souvent nécessaires d'analyser les performances sur différentes sous-catégories (slice-based)
  - Utile pour détecter de la discrimination et des erreurs sur une catégorie en particulier
  - Certains groupes sont plus importants que d'autres (membres qui paient vs qui ne paient pas)
- Comment découper ?
  - Heuristique : connaissances du domaines
  - Analyse des erreurs : regarder les erreurs et trouver des motifs
  - Slice finder : Outils de développement pour automatiser le processus

Déploiement de modèles et services de prédiction

Du développement à la production

- La production est un large spectre qui va de la présentation d'un notebook à répondre à des milliers de requêtes par secondes
- Déployer le modèle peut être facile
  - On met la méthode *predict* dans un appel POST d'un serveur web avec Flask ou FastAPI, on encapsule le tout dans un container Docker et voilà !
- Ce qui est compliqué, c'est passé à l'échelle
  - Millions d'utilisateurs
  - Faible latence
  - 99% de uptime
  - Notification en cas de problème
  - ...

Quelques mythes sur le déploiement de modèles

Mythe 1 : On n'a besoin de déployer seulement un ou deux modèles à la fois

- Vrai dans l'académie
- En pratique, il y a beaucoup de modèles
  - Chez Uber : Prédiction de la demande, de la disponibilité des conducteurs, du temps d'arrivée, des prix, des transactions frauduleuses, ... Souvent, des modèles différents pour chaque pays et langue.

Quelques mythes sur le déploiement de modèles

Mythe 2 : Si on ne fait rien, les performances restent constantes dans le temps.

- Besoin de mettre à jour les algorithmes et le code
- Les données peuvent changer (data distribution shift)

Quelques mythes sur le déploiement de modèles

Mythe 3 : Il n'est pas nécessaire de mettre à jour les modèles souvent

- En pratique, on veut aussi souvent que possible
  - Pour certaines entreprise, toutes les 10 minutes environ.

Quelques mythes sur le déploiement de modèles

Mythe 4 : Les ingénieurs ML ne doivent pas se soucier du passage à l'échelle

- Ce n'est pas un problème que pour les grosses entreprises
- Même un entreprise avec une centaines d'employés à ce genre de problèmes

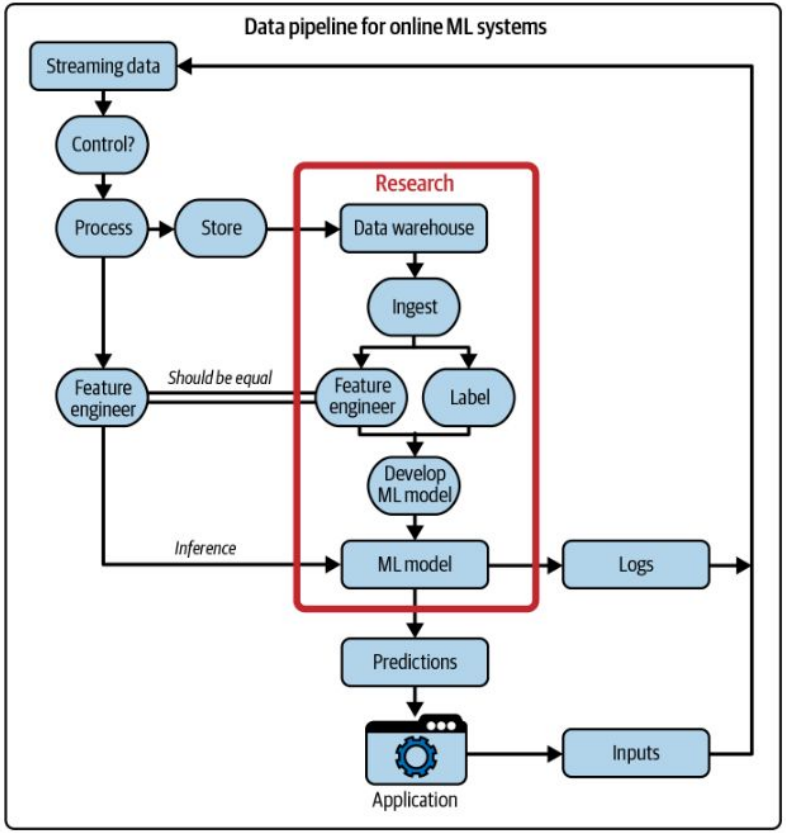
Prédiction en batch et en ligne

- Prédiction en ligne (online) = on doit répondre à la requête aussi vite que possible
  - Ex : Traduction sur Google Translate
  - Faible latence (= le modèle doit être rapide)
  - Obligatoire dans de nombreuses applications : high frequency trading, véhicules autonomes, assistants vocaux, reconnaissance d'empreintes digitales, détection de chutes, ...
- Prédiction en batch = les prédictions sont générées périodiquement et stockées
  - Ex : Les recommandations Netflix sont générées une fois toutes les quelques heures
  - Grande bande passante
  - Mais on risque de faire des calculs qui ne servent à rien
  - On ne s'adapte pas rapidement à un changement de préférence (ex, commence une nouvelle série)
- Attention, ici batch != batch en deep learning
- Avec les améliorations hardware et software, la prédiction en ligne devient de plus en plus présente.

Batch features et streaming features

- Batch features = features calculées en avance sur l'historique des données
  - Ex : Le temps moyen de préparation d'un repas pour un restaurant à une heure donnée
  - Mises à jour de temps en temps avec des frameworks comme Hadoop ou Spark (voir cours de Big Data)
- Streaming features = features calculées à la volée sur les données entrantes
  - Ex : Le nombre de commandes d'un restaurant sur les dix dernières minutes
  - Outils comme Kafka ou Apache Flink
- Il faut faire très attention d'avoir les mêmes features à l'entraînement et à l'inférence

Batch features et streaming features

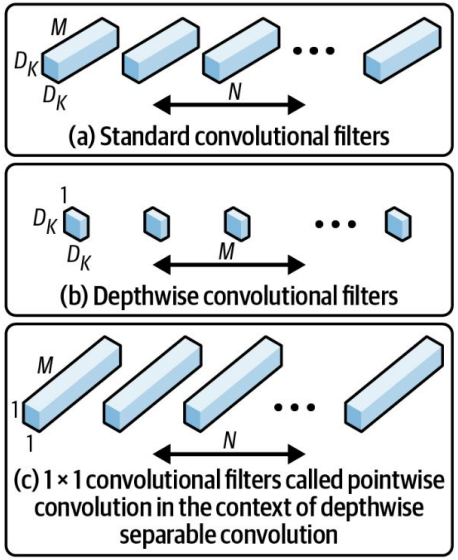


Compression de modèle

- Compresser un modèle consiste à en réduire la taille
- Avantages :
  - Inférence plus rapide
  - Modèle plus petit
  - Calculs hardware plus rapide
- Inconvénient : on perd en qualité

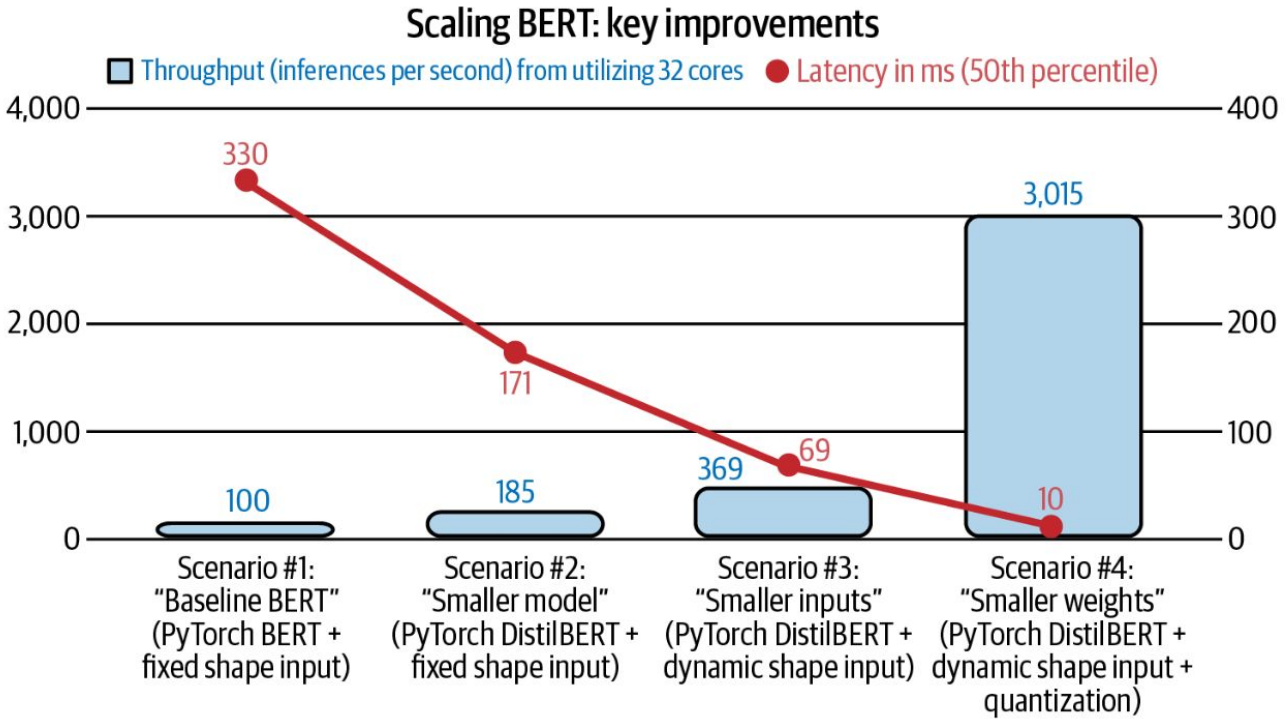
Techniques de compression

- Low-Rank Factorisation : Remplacer des tenseurs de haute dimension par des tenseurs plus petits
- Knowledge distillation : Entraînement d'un modèle plus petit (student) avec un modèle plus grand (teacher)
- Pruning : Retirer des nœuds d'un réseau, ou mettre des poids à 0 (sparsité)
- Quantization : Utiliser une précision plus faible pour stocker les flottants (32 bits vers 8 bits en général)
  - Plus grand batch size, calculs plus rapides
  - Mais moins de valeurs possible, perte de performance



MobileNet

Exemple pratique : Roblox et BERT



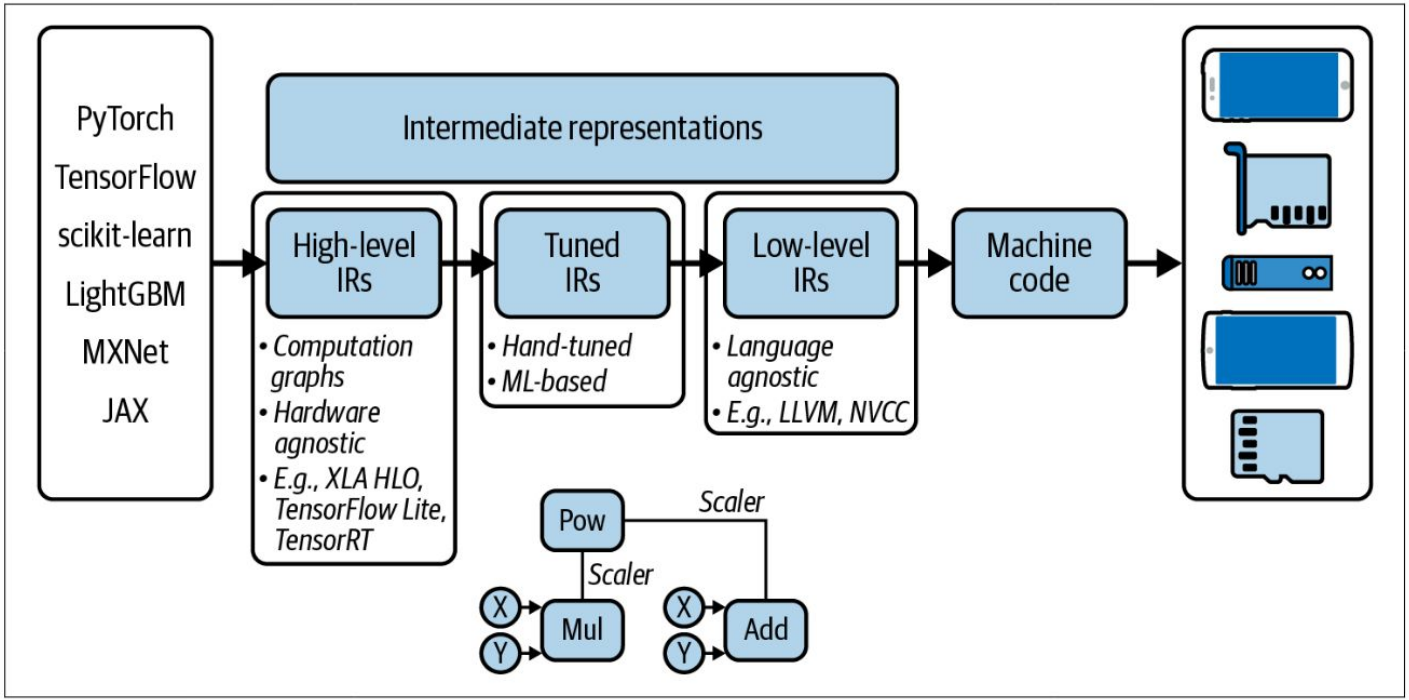
Déploiement sur le cloud ou sur l'edge

- Cloud = sur des machines connectées à internet qui n'appartiennent pas aux clients
  - Facile à utiliser, de nombreux outils
  - Cher
  - Nécessite une connexion à internet
- Edge = sur la machine des clients (navigateur, téléphone, ordinateurs, voitures, robots, ...)
  - Difficile à mettre en place à cause de la variété
  - Souvent machines peu performants (calculs, mémoire, batterie)
  - Gratuit pour l'entreprise
  - Pas besoin d'internet, pas de latence réseau
  - Peut traiter des données sensibles et privées sans les partager (RGPD)

Compiler et optimiser des models pour le edge

- Problème : le framework (ex : Torch) doit être compatible avec le hardware
  - Beaucoup d'optimisations sont spécifiques à un hardware (exploitation des opérations, de la mémoire, des caches, des registres, d'accélérations hardware, ...)
  - Pour les hardwares très utilisés (GPUs et CPUs classiques), déjà beaucoup d'investissements
  - Pour les hardwares peu utilisés, très compliqué de faire le portage
- Idée pour généraliser : utiliser des représentations intermédiaires qui servent à implémenter le code pour un hardware.

Représentations intermédiaires

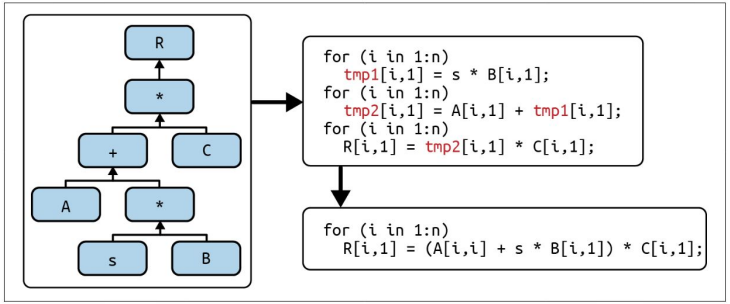


Optimisation de modèle

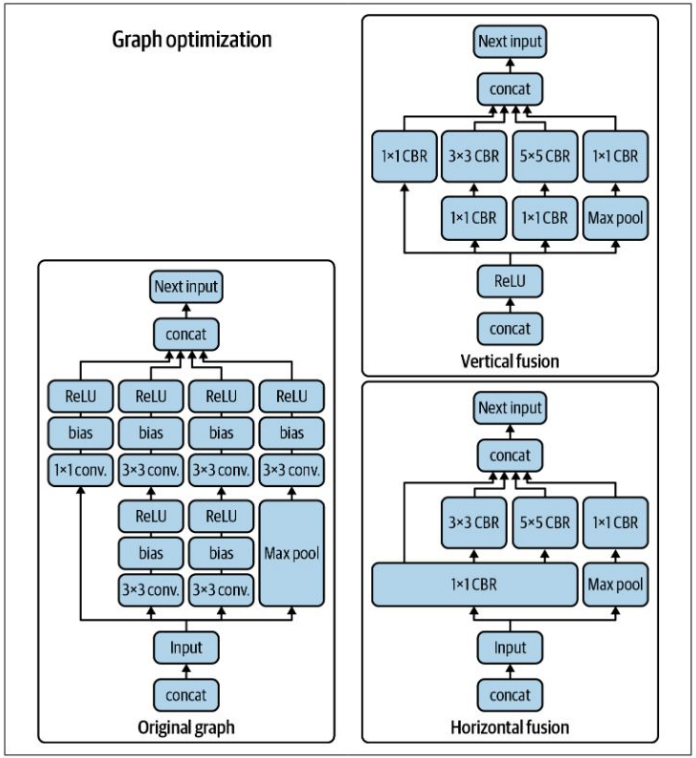
- Après avoir transformé notre modèle en langage machine, on a souvent des problèmes de performance
  - Difficile d'optimiser pour toutes les bibliothèques (scikit-learn, tensorflow, torch, pandas, numpy, transformers, ...)
  - Ingénieur Optimisation ML : connaissance à la fois ML et hardware
- Deux types de optimisations:
  - Locales = optimisation d'une opération ou d'un ensemble d'opérations
  - Globales = optimisation au niveau du graphe de calcul

Techniques d'optimisation

- Vectorization : traiter plusieurs itérations d'une boucle en parallèle
- Parallélisation : Diviser le travail en tâches indépendantes pouvant s'exécuter en parallèle
- Loop tiling : Changer l'ordre d'accès aux données pour mieux utiliser la mémoire et les caches
- Fusion d'opérations : fusionner des opérations pour éviter les accès mémoires redondants



Exemple de fusions avec des convolutions



Techniques d'optimisation

- Vectorization : traiter plusieurs itérations d'une boucle en parallèle
- Parallélisation : Diviser le travail en tâches indépendantes pouvant s'exécuter en parallèle
- Loop tiling : Changer l'ordre d'accès aux données pour mieux utiliser la mémoire et les caches
- Fusion d'opérations : fusionner des opérations pour éviter les accès mémoires redondants
- Algorithme ML : apprendre à prédire les temps de calculs pour trouver plus facilement les optimisations utiles (cuDNN autotune, autoTVM)
    - Très lent, doit être entraîné à chaque fois

ML dans un navigateur

- Utilisation de Javascript ou de WebAssembly
  - TensorFlow.js, Synaptic, brain.js
- Marche sur toutes les machines avec un navigateur
- Mais extrêment lent

Changement de distribution de données et monitoring

Après le déploiement du modèle

- Déployer le modèle n'est pas la fin du développement
- On observe souvent des performances moindre en production, qui diminue au fur et à mesure du temps
- Il va être important de contrôler nos modèles en production

Causes des pannes des systèmes de ML

- Un système ML peut avoir un problème avec ses métriques opérationnelles (latence trop élevée) ou sur les métriques de performance (accuracy)
  - Erreur opérationnelle plutôt facile à trouver
  - Erreur sur les performances difficiles à trouver, souvent silencieuses
- Deux types de pannes :
  - Logicielles (dépendances, déploiement, hardware, crash) : Génie logiciel très important pour l'ingénieur ML, à l'origine de la plupart des bugs
  - Spécifiques au ML : déjà vu quelques unes (collecte de données, hyperparamètres, différences entre entraînement et inférence, ...)

Données en production différentes des données d'entraînement

- Supposition des cours de ML/DL : les données d'entraînements et de tests viennent de la même distribution, qui est stationnaire
  - Données similaires et ne changeant pas, même comportements
- Mais supposition souvent incorrecte :
  - De nombreux biais dans la création du dataset (sélection, échantillonage)
  - Données non-stationnaires (data shift) : changement soudains (annonces de Trump sur l'Ukraine), graduel (apparition d'un nouveau produit), ou saisonniers (vélos en été)
  - Des erreurs humains (pipeline de données, valeurs manquantes, mauvaises entrées, incohérence entre features d'entraînement et d'inférence, ...) peuvent impacter les performances, souvent difficile à différencier des data shifts

Cas limites

- Les cas limites sont des données extrêmes qui entraînent des erreurs catastrophiques
  - Ex : véhicules autonomes, diagnostic médical
  - Même si 99.99% d'accuracy, le 0.01% peuvent être très coûteux

Boucles de rétroactions dégénérées

- Pour la création d'étiquettes naturelles, on enregistre ce que fait l'utilisateur pour mieux le prédire
  - Quand l'algorithme de ML est déployé, il impacte ce que fait l'utilisateur, ce qui impacte les données d'entraînement, ce qui impact ce qui est montré, etc.
  - Les modèles deviennent trop confiants dans leurs prédictions : on a une boucle de rétroaction dégénérée
  - Particulièrement présente dans les systèmes de recommandation et les prédictions de clics
  - Crée des biais et des bulles : bias d'exposition, biais de popularité, bulles de filtres, echo chambers

Boucles de rétroactions dégénérées - Détection

- Difficile à détecter offline, on doit mettre le système en production
- Il existe des métriques :
  - Basées sur la popularité des items dans les systèmes de recommandation : aggregate diversity, average coverage of long-tail items
  - Elles empirent avec le temps

Boucles de rétroactions dégénérées - Correction

- Introduire de l'aléatoire dans les prédictions
  - Mettre des recommandations aléatoires
  - Chez TikTok, pour chaque nouvelle vidéos, apparait 100 fois de manière aléatoire, puis apprentissage
  - Détériore l'expérience utilisateur
- Encoder la position dans une feature et la cacher à l'inférence
  - Les premiers objets recommandés ont tendance à être plus populaires
  - On peut rajouter la position, ou simplement si c'est le premier objet de la liste

Data Distribution Shift

Phénomène qui survient en apprentissage supervisé quand les données avec lesquelles un modèle travaille changent avec le temps

Ex : Changement de président, apparition d'un nouveau produit, phénomènes cycliques comme les saisons, ...

Types de data distribution shift

- On représente les données avec une distribution P(X, Y) qui nous donne la probabilité d'une entrée et d'une sortie
  - On peut écrire P(X, Y) = P(Y|X)P(X) = P(X|Y)P(Y)
- Covariate shift : Quand P(X) change mais pas P(Y|X)
  - La probabilité d'une entrée change mais si on savait déjà prédire la sortie pour une entrée, pas de changement
  - Ex : Vous voulez détecter un cancer du sein. Pendant l'entrainement, beaucoup d'exemple de femmes ayant plus de 40 ans, mais à l'inférence, population plus répartie
    - Cela n'impacte pas les prédictions
  - Causes : biais pendant la sélection des données d'entraînement, altération artificielle des données (pour compenser une classe sous représentée), changement dans l'application ou son usage (vise une clientèle plus fortunée)
  - Problématique si shift vers des données trop différentes ou plus complexes

Types de data distribution shift

- On représente les données avec une distribution P(X, Y) qui nous donne la probabilité d'une entrée et d'une sortie
  - On peut écrire P(X, Y) = P(Y|X)P(X) = P(X|Y)P(Y)
- Label shift : Quand P(Y) change mais pas P(X|Y)
  - Par exemple, pour détecter le cancer, à l'entrainement, vous avez autant de labels cancer que non-cancer. Par contre, à l'inférence, beaucoup plus de non-cancer.
    - Par contre, si vous prenez deux patients ayant le cancer au hasard dans l'entraînement et à l'inférence, ils auront tous les deux la même probabilité d'avoir plus de 40 ans.
  - Assez similaire au covariate shift

Types de data distribution shift

- On représente les données avec une distribution P(X, Y) qui nous donne la probabilité d'une entrée et d'une sortie
  - On peut écrire P(X, Y) = P(Y|X)P(X) = P(X|Y)P(Y)
- Concept drift : Quand P(Y|X) change mais pas P(X)
  - Les entrées sont similaires, mais la prédiction associée à une entrée change
  - Ex : Vous avez un site de vente de biens immobilier. Le COVID arrive et les gens quittent les villes. Résultat : les prix chutent pour des biens similaires
  - Souvent cycliques ou saisonnier
    - Prix des billets de train, des courses de taxis, de l'affluence en gare, ...

Data shifts dus au système

- Changement dans les features (ajout, retrait, modification)
- Changement des labels possibles (ajout, division ou retrait de labels)
    - Par exemple, modèle qui peut diagnostiquer une nouvelle maladie

Détecter les data shifts

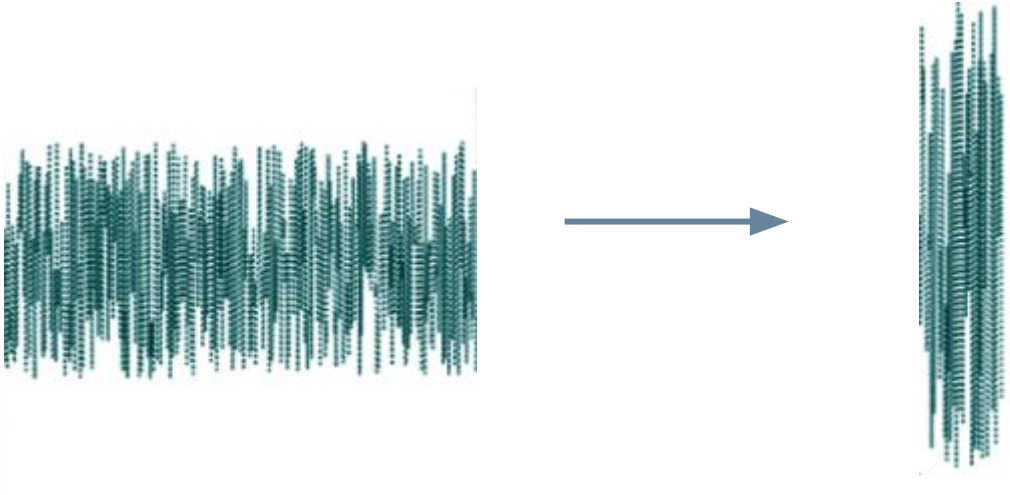
- Les data shifts ne sont pas toujours problématiques
  - Il faut contrôler les métriques d'accuracy
  - Problème : ok si labels naturels, sinon difficile
- La plupart des méthodes font du monitoring sur les données en entrées P(X)
  - Indépendant des labels

Détection statistiques

- On va logger différentes statistiques et les comparer au cours du temps
  - Min, max, mean, median, variance, quantiles, ...
  - Même si métriques similaires, aucune garantie qu'il n'y ait pas eu de shift
- Utilisation de tests statistiques
  - Two-sample hypothesis test
  - Vous disent si deux distributions sont statistiquement différentes ou pas
  - Attention aux conditions et limites d'utilisation
  - Test de Kolmogorov-Smirnov : pas de paramètre ni de supposition, mais que vous des données unidimensionnelles. Test cher et avec beaucoup d'alertes faux positifs.
  - Least-squares density difference, Maximum Mean Discrepancy, ...
  - Voir la bibliothèque **Alibi Detect**

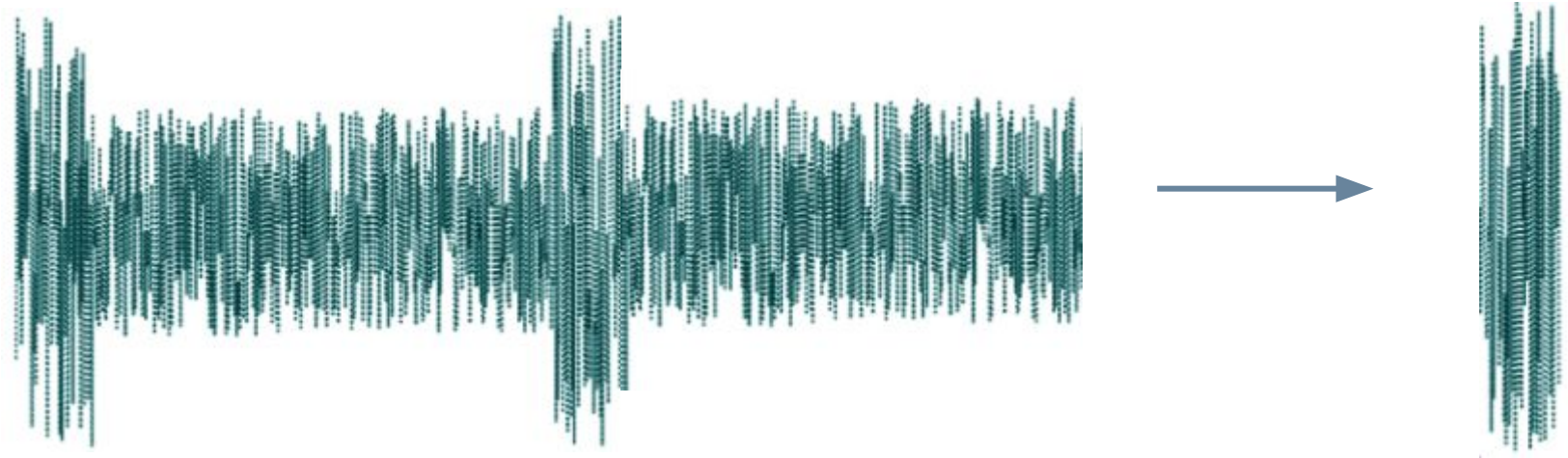
Quelle fenêtre de temps pour détecter un data shift ?

- Certain changements sont brusques, d'autres très lents et graduels
- Pour les shifts dans le temps difficile de savoir ce qui est cyclique ou non



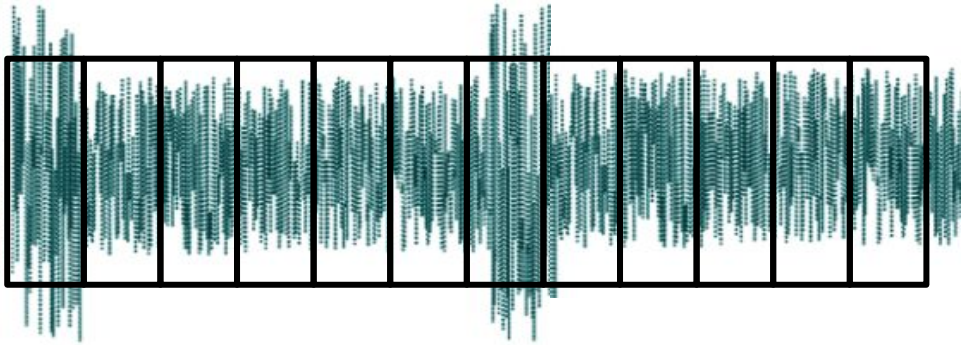
Quelle fenêtre de temps pour détecter un data shift ?

- Certain changements sont brusques, d'autres très lents et graduels
- Pour les shifts dans le temps difficile de savoir ce qui est cyclique ou non



Quelle fenêtre de temps pour détecter un data shift ?

- Certain changements sont brusques, d'autres très lents et graduels
- Pour les shifts dans le temps difficile de savoir ce qui est cyclique ou non
- Deux types de métriques
  - Fenêtre glissante : on déplace un fenêtre avec un certain pas et on calcule des statistiques dans cette fenêtre



Quelle fenêtre de temps pour détecter un data shift ?

- Certain changements sont brusques, d'autres très lents et graduels
- Pour les shifts dans le temps difficile de savoir ce qui est cyclique ou non
- Deux types de métriques
  - Fenêtre glissante : on déplace un fenêtre avec un certain pas et on calcule des statistiques dans cette fenêtre
  - Statistiques cumulatives : on accumule une statistique au fur et à mesure du temps
    - Ex : Moyenne glissante
- Pour une analyse poussée, il faut étudier les techniques d'analyse de séries temporelles
- En pratique, on choisit une échelle de temps pour le monitoring comme l'heure ou la journée

Solutions au data shift

1. Entraîner sur beaucoup de données
    a. On espère qu'il va contenir tous les cas possibles
2. Essayer d'adapter un modèle existant
    a. Toujours du domaine de la recherche
3. Réentraîner le modèle avec les nouvelles données annotées
    a. Soit de zéro, soit faire du finetuning
    b. Bien choisir les données : dernière 24 heures, semaine, six mois ?
4. Rendre les features moins changeantes
    a. Discrétisation : 1-10 = 1, 10-100 = 2, 100-1000 = 3, ...
    b. Compromis entre stabilité de la feature et performance
5. Adapter le système pour traiter les shifts plus facilement
    a. Ex: pour prédire le prix des maisons, un modèle pour Paris et les grandes villes, un modèle pour les campagne. Les prix changent plus vite à Paris.

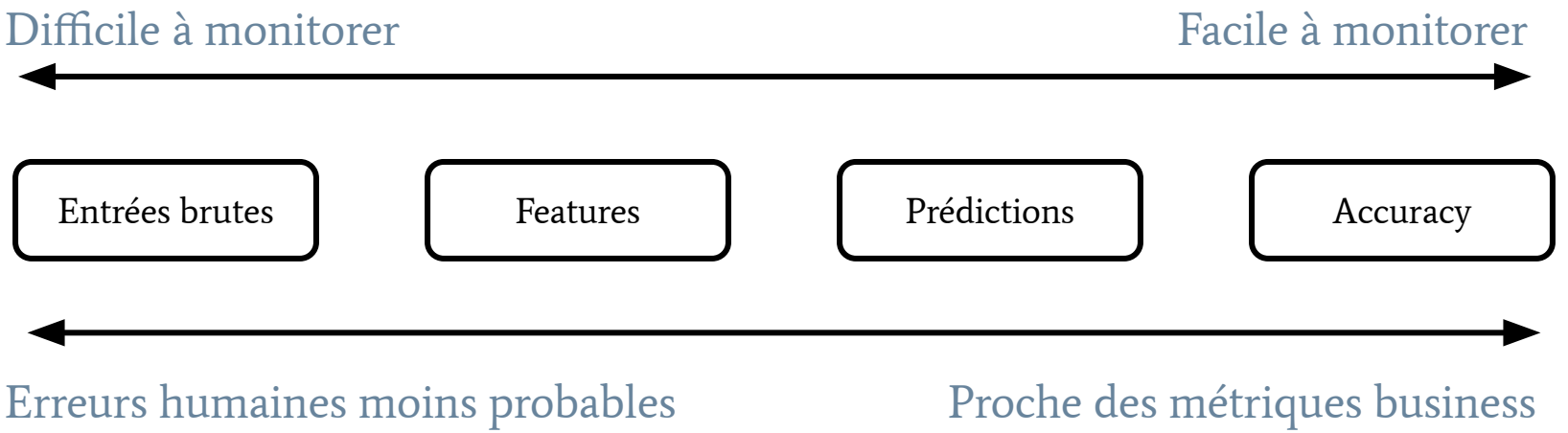
Monitoring vs Observabilité

- Monitoring = suivi de différentes métriques pour savoir **quand** il y a un problème
- Observabilité = suivi de différentes métriques pour savoir **pourquoi** il y a un problème
- L'observabilité fait partie du monitoring

Les métriques opérationnelles

- On veut contrôler les systèmes, indépendamment du ML
  - Latence, bande passante, nombre de prédictions demandées par seconde, heure ou jour, pourcentage de requête 200, utilisation du CPU/GPU, utilisation mémoire

Métriques spécifiques au ML

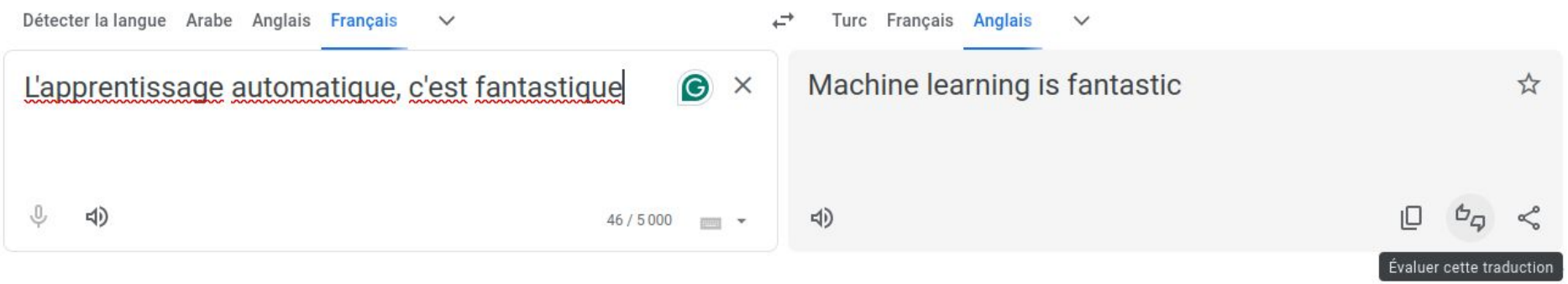


Métriques spécifiques au ML - Accuracy

- Il faut des étiquettes naturelles pour calculer des métriques sur les sorties de notre modèle
- On peut aussi suivre n'importe quel feedback utilisateur : click, achat, upvote, downvote, favoris, partage, ...
  - Important à suivre car souvent corrélée à la sortie de notre modèle
  - Dans les systèmes de recommandation : click-through-rate = click sur un recommandation, completion-rate = à regarder la vidéo recommandée en entier
- Ne pas hésitez à ajouter dans votre système la possibilité à votre utilisateur de faire un retour

Métriques spécifiques au ML - Accuracy

- Il faut des étiquettes naturelles pour calculer des métriques sur les sorties de notre



un retour

Métriques spécifiques au ML - Prédictions

- Facile à visualiser et interpréter
- On peut utiliser des tests statistiques
- On peut essayer de logger des événements douteux
  - Le modèle prédit la même sortie pendant beaucoup de fois consécutives
  - Les valeurs sont constamment au dessus d'un seuil

Métriques spécifiques au ML - Features

- Validation de features : on veut vérifier que les features respectent certaines caractéristiques
  - Min, max, mediane dans un intervalle satisfaisant
  - Feature suit une regex (date par example)
  - Feature dans un set prédéfit (catégories)
  - Règles entre plusieurs features : F1 est toujours supérieur à F2
- Problèmes majeurs
  - Si beaucoup de features, peut ralentir le système et augmenter la latence
  - Bien pour débugger, moins utile pour détecter une dégradation des performances du modèle
  - Difficile de trouver la cause exacte : l'extraction de feature utilise de nombreuses étapes et bibliothèques
  - Le schéma des features change avec le temps => fausses alertes. Besoin de versionner.

Métriques spécifiques au ML - Entrée brutes

- Souvent, pas d'accès aux entrées brutes
- C'est aux ingénieurs data de gérer ça

Comment faire du monitoring : les logs

- Classique en génie logiciel, on enregistre tout ce qui nous intéresse : initialisation d'un conteneur, mémoire utilisée, appel de fonction, crash, stack trace, code d'erreur, ...
- Beaucoup de composantes dans les systèmes modernes
  - On veut savoir quand il y a un problème et où !
  - Donner des identifiants uniques aux processus et logger des metadata (heure, service concerné, fonction, utilisateur, ...)
- Peut devenir très dur à traiter beaucoup de logs
  - Badoo, une application de rencontre, avait 20 milliards de requêtes par jours
  - Il faut des algorithmes de détection d'anomalie automatique utilisant du ML
  - On utilise des outils comme Spark, Hadoop, Hive, Kafka, Flink, ...

Comment faire du monitoring : les dashboard

- On représente des métriques avec des courbes
  - Facile à visualiser, même pour des gens non techniques (même si un expérience en statistique est utile)
  - Par contre, on doit limiter l'information à visualiser

Comment faire du monitoring : les alertes

- On peut déclencher une alerte (par mail ou Slack par exemple) à chaque fois qu'un événement suspect se produit
- Il faut :
  - Une politique d'alerte = condition
  - Canal de notification = qui prévenir et comment
  - Description de l'alerte = ce qui ne va pas
- Attention, trop d'alertes tue l'alerte

L'observabilité

- On veut comprendre d'où vient le problème
- Problématique quand beaucoup de services dont certains ne sont pas gérés par nous
  - Il faut, uniquement à partir des logs, trouver d'où vient le problème
- Nous voulons être sur d'avoir assez d'information pour retrouver l'erreur et ce qui l'a causé à posteriori
  - Métriques très précises
  - Moyen de faire des requêtes sur les logs ("Montre moi tous les outliers dans les dix dernières minutes")
- L'observabilité est souvent lié à la notion d'interprétabilité

En résumé

- Développement d'un modèle et entraînement
- Évaluation Offline
- Prédiction en ligne vs en batch
- Compression de modèle
- Le cloud et l'edge
- Les erreurs dans les systèmes ML
- Data shift
- Monitoring et observabilité