C1: Fouille de données: étude de la collecte, nettoyage, traitement, analyse et connaissances issus des données. Systèmes automatisés génèrent données à des fins de diagnostic ou d'analyse. Données brutes sont souvent non structurées. Pipeline de la fouille de données :Datacollection→DataPreprocessing(Feature extraction|Cleaning and integration)→ Analyctical Processing(Building block1|Building block2)→ Output for analyst. Nettoyage : logs contiennent bcp d'informations supplémentaires qui ne sont pas utiles pour le détaillant. Extraction d'attributs : le détaillant décide de créer un enregistrement pour chaque client, avec un choix spécifique d'attributs. Intégration des données : ces enregistrements sont combinés aux données démographiques des clients stockées chez le détaillant. Analyse des données l'analyste doit décider cmt utiliser cet ensemble de données nettoyé pour faire des recommandations. Data science vs. Computer science : informaticiens n'apprécient pas les données → ce sont juste des choses à exécuter dans un programme. Tester performances d'un algo consiste à exécuter l'implémentation sur des données aléatoires. Ensembles de données intéressantes sont rares, demande du travail pour les obtenir. ***Science traditionnelle***:Hypothesis-driven :fait des questions et ensuite génère données spécifiques qui sont nécessaires pour confirmer ou réfuter l'hypothèse. ***Nouvelle science*** : Data-driven :axée génération de données, en croyant que de nouvelles découvertes seront faites aussitôt que l'on sera capable de les explorer. C2 : Data munging : acquérir données et les préparer pour analyse. Notebooks : permettent combiner code, données, résultats, texte et rendent projets reproductibles, modifiables, documentés. Maintiennent séquence des procédures utilisées sur les données du début à la fin du projet. Où sont les données? 1ère étape d'un projet de fouille de données est de **trouver où sont les données** dont on aura besoin. Métadonnées : les **grandes BD contiennent souvent de métadonnées**(informations),grandes sources de données ex : Wikipedia. Sources de données : 1)**Données privées**(Google,Fb ex :nom, date naissance)→ organisations ont bases de données internes d'intérêt pour leur business. Obtenir accès extérieur à ces données est généralement impossible (Desjardins est une exception car leur data a leak). Entreprises publient API qui permettent de récupérer "quelques"données.**2)Données gouvernemaentales**→Villes et pays s'engagent de + en + à ouvrir leurs données. Question clé quand base de données publique est la préservation de l'anonymat.**3) Données académiques**→revues, recherches scientifiques par (ex :Rendre données disponibles est maintenant une exigence pour les publications). Permettent trouver données économiques, médicales, démographiques et météorologiques. Repérez les documents pertinents et demandez aux propriétaires. Cependant, ces données ont été déjà très explorées. **4)Web search et web scrapping**(technique d'extraction du contenu de sites Web)→ Scraping : l'art de récupérer des données d'une page Web. Faut d'abord vérifier s'il y a des APIs rendues disponibles par le site et s'il y a qq1 qui a déjà codé un scraper. Conditions de service limitent ce qu'on peut légalement faire. **5)Internet of Things** (IoT)→ Sources de données sont très variées. Le stockage n'est pas cher! Type de données : **1)Données indépendantes** : soit au niveau des individus, soit au niveau des caractéristiques. ***Données multidimensionnelles*** : Un jeu de données *D* est un ensemble de *n* enregistrements X1, … Xn où chaque enregistrement Xi contient un ensemble *d* d'attributs nommé $(X_i^1, \dots X_i^d)$. Ex : chien contient (race, couleur), personne(grandeur, poids), plante(nom latin, médicaments utilisés). Nature des attributs classifie données comme numériques, catégoriques, binaires, textuelles. **2)Données dépendantes** : présentent des relations (ex. réseau sociaux, séries temporelles) Ex: chaque élève on enregistre son poids --> caractéristiques (ex: taille et poids) dépendantes et (ex: taille et lunettes) indépendantes. Connaissance des dépendances préexistantes modifie la fouille de données. **Dépendantes vs. Indépendantes** :Données dépendantes sont + complexes en raison des relations préexistantes entre les éléments. Séries temporelles : contiennent des valeurs générées par mesures continues dans le temps. Ces données ont des dépendances implicites intégrées aux valeurs observées au fil du temps. Certaines séries peuvent montrer des patrons périodiques d'attribut mesuré au fil du temps. Séquences discrètes sont la version catégorique des séries temporelles. Données spatiales : Attributs non spatiaux (ex. température, pression, intensité de couleur de pixel d'image) peuvent être mesurés dans l'espace. Les ***données spatio-temporelles*** contiennent à la fois des attributs spatiaux et temporels. **Portabilité** : capacité à s'adapter facilement en vue de fonctionner dans différents environnements d'exécution. La portabilité du type de données est un élément crucial en fouille de données, car données sont hétérogènes et peuvent contenir plusieurs types d'attributs. C'est toujours + compliqué et long de développer des algos qui puissent marcher pour des jeux de données hétérogènes. **Les outils *off the shelf*** *(license publique)* sont souvent pour un type spécifique de données. Numérique → Catégorique 1)**Discrétisation**→divise l'intervalle de valeurs d'un attribut numérique en ϕ intervalles. Alors, l'attribut est supposé d'avoir 1, . . , ϕ valeur catégorique. Ex. attribut numérique âge : Création de 4 intervalles : [0, 20]; [20, 40]; [40, 60]; [60, 100]. La valeur symbolique d'un enregistrement dans l'intervalle [20, 40] est 2, tandis qu'un autre dans l'intervalle [60, 100] est 4. Variations dans intervalle sont perdues après la discrétisation.Données peuvent être non uniformément distribuées à travers les intervalles. Il faut faire attention aux tailles des intervalles en fonction de la distribution des données par rapport aux attributs discrétisés.**2) Binarisation**→prends les ϕ valeurs possibles d'un attribut catégorique et crés ϕ nouveaux attributs binaires (numériques). L'un des ϕ attributs prend la valeur de 1, et le reste prend la valeur 0. Texte → numérique : 1) **Bag-of-words** : * attributs correspondent aux mots, et valeurs correspondent aux fréquences de ces attributs (ex : des 1 et 0 quand blond, brun roux). *Représentation très creuse (≈ 105 mots différents dans langue anglaise)→ par contre si on a 10 000 mots, pas possible de tous les représenter avec des 1 et des 0! Perte des relations entre mots. **2)Word embeddings** : transformation de n'importe quel type de données en numérique avec de l'apprentissage profond. ex : 10 valeurs aléatoires pour chaque mot, on donne en entrée les 10 valeurs d'un mot, on veut en sortie les mots à proximité du milieu.Série temporelle → numérique : 1)Transformée en ondelettes discrètes ***wavelets*** : convertit données de séries temporelles en données multidimensionnelles, en un ensemble de coefficients représentant les différences moyennes entre différentes parties de la série. Un sous-ensemble de plus grands coefficients peut être utilisé pour réduire la taille des données numériques. Les données résultantes sont moins dépendantes que la série temporelle originale. N'importe quel type → Graphe : Il est possible de convertir n'importe quel type de données en graphe. 1) Graphe de voisinage : Chaque enregistrement du jeu de données est un noeud du graphe. Une arête existe entre deux éléments *i* et *j* si leur distance *d(i, j)* est inférieure à un seuil. Le poids *w(i, j)* de l'arête *(i, j)* est égal à une fonction kernel appliquée sur *d(i, j)*. Ex. *heat kernel* : $w(i,j) = e^{-d(i,j)^2/t^2}$ où t est un hyperparamètre. Nettoyage de données : !**Erreurs** : informations perdues lors de leurs acquisitions (ex: on capte températures et capteur ne marche pas bien, donc données fausses,données perdues). !**Artefacts** : problèmes découlant du traitement de données (peuvent être traités et viennent du coût de données réversibles, donc sont réversibles). Compatibilité de données : données ont besoin d'être compatibles pour faire des comparaisons (ex: Sur Mars, les unités pas les mêmes pour le robot, dans l'espace en Newton et sur le terrain en Lb, donc robot a craché). Conversion d'unités : Même la décision pour le système métrique a des incohérences potentielles : cm, m, km ? Attention au moment d'intégrer des données de bases différentes (ex : pieds vs. Mètres). Si distribution comporte deux valeurs, alors → problème de compatibilité. Représentation numérique : Attributs discrets doivent toujours être des entiers. Attributs mesurés physiquement ne sont jamais quantifiés avec précision, alors ils doivent être représentés par des chiffres réels(ex :22,3 cm). Les quantités fractionnaires doivent être décimales, éviter de les représenter par *(q,r)* ex :(livres, oz) ou (pieds, pouces). Représentation de caractères : Un problème de nettoyage désagréable dans les données textuelles est l'unification des représentations de code de caractères : ISO 8859-1 est un code à one byte pour l'ASCII (originalement à 7 bits) qui inclut quelques caractères et ponctuations pour les langues latines. UTF-8 est un codage multi octets pour tous les caractères Unicode, compatible avec ASCII.Alors,il faut toujours!! générer de texte en UTF-8!! **Unification de noms** : **1) Utiliser transformations simples** pour unifier les noms(ex. minuscules). **2) Considérer les méthodes phonétiques**(essaie de suppléer aux défauts de l'alphabet en le remplaçant, pour l'enseignement de la lecture, par un alphabet spécial, où chaque signe a une valeur fixe) de hashing comme Soundex et Metaphone(ex : élimination des lettres doublées). **3) Compromis entre faux positifs et négatifs**(ex : deux personnes avec même nom→ faux positifs: 2 personnes pour la même personne comme Daniel, Daniel Aloise peut être compris pour Daniel Aloase).

Unification de dates dans le temps : Aligner événements temporels de différents ensembles de données/systèmes peut être problématique. Alors, utiliser le temps universel (UTC) ou le temps UNIX. Unification financière : Nécessaire de **tenir compte de la valeur de l'argent** au cours du temps (inflation) pour faire des comparaisons justes à long terme. Utiliser rendements/changements de % au lieu des changements de prix absolus pour comparer pls variables dans temps(ex : prix de l'action de McDonald's et celui du pétrole sont corrélés sur 30 ans). Normalisation : les différents attributs ont différentes échelles de référence et ne peuvent pas être comparés entre eux(ex : âge et salaire). Toute fonction agrégée calculée sur différents enregistrements (ex : les distances euclidiennes) sera dominée par l'attribut de + grande magnitude. Considérons μ_j la moyenne de l'attribut *j* et σ_j son écart type : **standardisation ou z-scores** : $z_i^j = \frac{x_i^j - u_j}{\sigma_j}$ →**Standardisation réduit et centre les données**. Prendre les valeurs, soustraire la moyenne et diviser l'écart type suppose une distribution normale de moyenne 0 et d'écart type 1. Le biais optimal est 0 lorsque données centrées (moyenne = 0). Il est possible interpréter coefficients de la régression si attributs ont la même échelle. Pour éviter les instabilités numériques, il faut que échelle des attributs soit petite. Donc réduire les données (écart type = 1). **Min-max scaling** : ∈ [0, 1], sensible aux données aberrantes : $y_i^j = \frac{x_i^j - min^j}{max^j - min^j}$ → les valeurs entre 0 et 1, mais maximum n'est pas garanti(ex: age, c'est pas garanti jusqu'à quand on va vivre). Min-max scaling implique écarts + petits. Clustering, PCA → z-scores. Traitement d'images, données ∈ [0, 1] → min-max scaling. Valeurs manquantes : aspect important du nettoyage des données est de représenter correctement les données manquantes. Mettre ces valeurs à 0 est une mauvaise idée ! ex: quand un individu ne mentionne s'il a des lunettes ou pas[Personne 1 -> lunettes Oui, Personne 2 -> lunettes Non, Personne 3 -> rien, donc mettre 0 pas bonne idée, on va essayer d'estimer valeur (imputation de valeurs manquantes). **Imputation(affectation) de valeurs manquantes** : souvent préférable d'estimer ou d'affecter des valeurs manquantes au lieu de les laisser vides(ex. une bonne estimé pour votre année de décès est l'année de naissance + 80). **valeur moyenne** : conserve la moyenne. **valeur aléatoire** : sélection de valeurs aléatoires permet évaluation quantitative de impact de l'imputation et de la qualité du modèle. **Régression** : l'utilisation régression linéaire pour prévoir valeurs manquantes fonctionne bien si peu de champs sont manquants par enregistrement(ex : l'utilisation de la régression linéaire pour prévoir les valeurs manquantes fonctionne bien pas bcp de champs manquants par enregistrement. Si valeur a impact sur notre analyse, on doit faire une regression). **Coefficients régression linéaire** : Ex :lorsque des attributs se multiplient→ Si données ont été normalisées,coefficients reflètent importance des attributs qu'ils multiplient par rapport à la sortie. Ils peuvent être comparés entre eux. Si données ne sont pas standardisées, alors impossible de comparer l'importance des attributs avec valeur des coefficients (ex. l'âge avec un coefficient 1 peut avoir la même importance que le salaire avec un coefficient 0.0001 si les données pas standardisées). Détection de données aberrantes : Regarder de manière critique les valeurs max et min pour toutes les variables. Les données distribuées ne devraient pas avoir de grandes valeurs aberrantes. Éviter solution facile de juste la supprimer. Visuellement, il est facile de détecter les valeurs aberrantes, mais seulement dans les espaces de faible dimension. Considéré comme un problème d'apprentissage non supervisé. C3 : Réduction de données : Lorsque la taille des données est + petite, il est + facile et performant d'appliquer des algos sophistiqués et complexes. + facile d'interpréter les résultats et de les visualiser. Réduction des données peut se faire en termes du **nb de lignes (enregistrements)** et/ou en termes du **nb de colonnes (dimensions)**. Réduction de données entraîne trjs une perte d'information. **1) Échantillonnage de données** : les enregistrements(nb lignes) sont échantillonnés pour créer BD bcp + petite. Une fraction de données est sélectionnée et retenue pour l'analyse. **A)** Dans ***l'échantillonnage biaisé,*** certains enregistrements sont + probablement retenus en raison de leur grande importance pour l'analyse(ex : certains types de données temporelles).**B)**Dans ***l'échantillonnage stratifié***, les données sont divisées en ensemble de strates souhaitées, puis échantillonnage est fait indépendamment à partir de chacune des strates en fonction de proportions prédéfinies(ex : sondage sur style de vie des individus d'une population ; un échantillon aléatoire d'1 million de participants peut ne pas capturer un milliardaire) **2) Sélection d'attributs**: seul un sous-ensemble des attributs est utilisé dans le processus analytique(sélectionner les colonnes). **Moyenne** : $\bar{X}^s = \frac{1}{n}\sum_{i=1}^n X_i^s$ **Écart-type(variance^2)** : $\sigma(X^s) = \sqrt{\frac{1}{n}\sum_{i=1}^n (X_i^s - \bar{X}^s)^2}$ **Covariance** : $Cov(X^s, X^t) = \frac{1}{n}\sum_{i=1}^n (X_i^s - \bar{X}^s) \times (X_i^t - \bar{X}^t)$ **3)Réduction de dimension** : corrélations entre lignes ou colonnes sont exploitées pour représenter données dans une dimension + petite. Matrice de covariance : En calculant la covariance entre chaque paire d'enregistrements (ou chaque pair d'attributs), on obtient matrice de covariance contenant chaque paire d'attributs. Centralisation par la moyenne : $X_i^s \leftarrow X_i^s - \bar{X}^s \ \forall i = 1, \dots, n; \forall s = 1, \dots, d$. Une matrice *d* × *d* de produits scalaires, mesurant "le synchronisme" entre les attributs → $\sum = \frac{X^T \times X}{n}$. Une matrice *n* × *n* de produits scalaires, mesurant "le synchronisme" entre les enregistrements (individus ou les lignes) → $\sum = \frac{X \times X^T}{d}$ Sélection des attributs : Coefficient de corrélation(tenir compte de l'écart type de nos attributs) → $Corr(X^s, X^t) = \frac{Cov(X^s, X^t)}{\sigma(X^s) \times \sigma(X^t)}$ Matrice corrélation permet évaluer slm des pairs d'attributs. On peut évaluer des triples ou des corrélations pour des sous-ensembles d'attributs + grands. Cependant, le nb de sous-ensembles est exponentiel. La plupart des méthodes de sélection évaluent les attributs indépendamment les uns des autres. Les méthodes sont différentes en fonction des données disponibles (avec ou sans étiquettes). Filtrage : Classe chaque attribut en fonction d'une mesure univariée. Sélectionne les attributs avec les mesures les plus élevées. La mesure doit refléter le pouvoir discriminant de chaque attribut. Ex : Score de Fisher → $F(s) = \frac{\sum_{j=1}^k p_j (\bar{X}_j^s - \bar{X}^s)^2}{\sum_{j=1}^k p_j (\sigma(X_j^s))^2}$ où *pj* est la fraction de données appartenant à la classe *j*. Filtrage corrélé : Algorithme itératif → **1)** sélectionner un attribut ***s*** (selon une métrique quelconque). **2)** vérifier corrélation de ***s*** avec attributs déjà sélectionnés (par pair d'attributs). **3)** Si somme de ces corrélations dépasse un seuil, enlève ***s*** de la sélection et STOP ! Filtrage→ **Avantages** : Rapide à calculer. **Désavantages** : Un attribut qui n'est pas "utile" tout seul peut être très utile lorsqu'il est combiné avec d'autres. **Discriminant linéaire de Fisher** : filtrage des **combinaisons linéaires** des attributs. Emballage : Influencé par le modèle choisi (biaisé). Coûteux en temps de calcul. Si pas d'étiquettes,exécuter une méthode de *clustering* pour en avoir. Réduction de dimension : Décomposition par valeur propres → Toute matrice *A* ***carrée symétrique*** de taille *n* × *n* peut être décomposée de la façon suivante : $A = \sum_{i=1}^n \lambda_i U_i U_i^T$ où *Ui* est le vecteur propre associé à la valeur propre *λi*. Remarque :les produits les + importants sont ceux associés aux valeurs propres les plus grandes. Analyse de composantes principales : **La matrice de covariance** $\sum = \frac{X^T \times X}{n}$ est symétrique *(d × d)* et semidéfinie positive. Elle se décompose par : $\sum = \sum_{i=1}^d \lambda_i U_i U_i^T$ où tous les *λi ≥ 0, i = 1, . . , d*. Les vecteurs *Ui , i = 1, . . . , d* sont nommés composantes principales pour la PCA et sont triés en ordre croissant par rapport à *λi*. On dénote *X'* la matrice *n* × *d* obtenue par : $X' = X \times U$ Pour la **PCA**, slm *k << d* colonnes de *X'* présentent des valeurs qui varient de façon significative. On peut prouver que vecteurs propres *Ui* représentent de solutions orthogonales successives au problème de la maximisation de la variance $v^T \Sigma v$ le long d'une direction unitaire *v*. Le problème réduction de dimension d'une matrice X de données peut aussi être vu comme celui de factoriser une matrice : X = B * C où B et C sont de taille *n* × *k* et *k* × *d*. Si k < min(n, d), B et C réduisent X. Décomposition par valeurs singulières : La **SVD** d'une matrice X de taille *n* × *d* la factorise comme : $X = WDV^T$ où D est une matrice diagonale rectangulaire *n* × *d*, W est de taille *n* × *n* et V est de taille *d* × *d*. Les matrices *W* et *V* sont orthogonales. Donc, *WD* pondère chaque colonne de W par D, de même pour DV^T . Le fait de ne conserver que les lignes/colonnes avec des poids d_i importants (**singular values**) nous permet de compresser X avec relativement peu de perte. Soit A et B,2 vecteurs de taille *n* × *1* et *1* × *m*, le **produit externe** des vecteurs donne une matrice est : P = A ⊗ B avec P[i, j] = A[i]B[j]. La matrice X peut être exprimée par la somme des produits externes de la SVD avec les termes : $(WD)_k \ et \ (V^T)_k$ → $X = WDV^T = \sum_k WD_k \otimes V_k^T$ En additionnant slm les + grands produits matriciels, on obtient une approximation de X. Multidimensional scaling (MDS) : projette un espace de grande dimension d'origine sur un espace de petite dimension en préservant les distances entre les paires. **input** : matrice de distances D de taille *n* × *n* obtenue dans l'espace original, et la dimension *k* < *d* de la projection. **output** : une configuration *Y1, . . . , Yn* de *n* points dans R^k. **metric** : **MDS** minimise : $\min_{Y \in R^k} \sum_{i=1}^{n-1} \sum_{j=i+1}^n (D_{ij} - ||Y_i - Y_i||)^2$ Un algorithme de descente du gradient peut être utilisé pour optimiser MDS. ISOMAP : Les données font partie d'un *manifold* (tubulure) non linéaire de dimension plus faible dedans un espace de plus grande dimension (on calcule la distance bleue alors qu'on veut la rouge). **ÉTAPE 1)** Pour chaque *Xi , i = 1, . . . , n*, nous trouvons ses voisins situés à l'intérieur d'une petite distance euclidienne de *Xi* . **ÉTAPE 2)** Construire un graphe avec une arête entre tous les points voisins. **ÉTAPE 3)** La distance géodésique entre deux points quelconques est ensuite approchée par le chemin le plus court entre les points dans le graphe. **ÉTAPE 4)** MDS est appliqué aux distances obtenues pour produire une représentation en dimension faible. t-distributed stochastic neighbor embedding (t-SNE) : Chaque donnée multidimensionnelle est modélisée par une autre à plus petite dimension. t-SNE préserve la structure locale en maintenant autant que possible la structure globale intacte. **Modèle probabiliste** : Enregistrements similaires sont modélisés par pts voisins et des données dissemblables (divergents) sont modélisées par des pts éloignés avec une **probaba élevée** dans espace original et dans espace réduit. **ÉTAPE 1)** Distribution de probaba sur paires de données est construite de manière que données similaires ont forte proba d'être sélectionnées ensemble. **ÉTAPE 2)** Chercher distribution semblable pour données mappées en dimension réduite. Enfin, la divergence de Kullback-Leibler (une mesure de divergence) est minimisée entre les 2 distributions (problème d'optimisation non convexe). Transformation spectrale : Graphe de similarité ⇒ données multidimensionnelles. Conserve structure de similarité d'un point de vue **local**. Très utile étant donné l'importance des graphes comme structure de représentation. Soit G(N, A) un graphe non dirigé → On assume que |N| = n. Une matrice symétrique W de taille *n* × *n* contient similarités entre noeuds liés par une arête. Cette matrice est **creuse**. On veut incorporer noeuds de ce graphe dans espace de dimension *d* afin que structure de similarité des données soit préservée. Transformation spectrale : cas simple → *n* noeuds sur un ensemble de valeurs *y1, . . . , yn* à 1D. Il n'est pas souhaitable que des noeuds connectés avec des arêtes de poids élevé (très similaires) soient mappés sur des points éloignés de cette ligne. On veut minimiser alors : $O = \min \sum_{i=1}^n \sum_{j=1}^n w_{ij}(y_i - y_j)^2$. *O* peut être réécrit en termes de la **matrice Laplacienne L** de W → $O = 2y^T L y^T$ où *D* est une matrice diagonale telle que $D_{ii} = \sum_{j=1}^n w_{ij}$ et L = D − W . O est minimisé avec $y = (y_1 \dots y_n)^T$ égal au + petit vecteur propre de la relation $D^{-1}Ly = \lambda y$ Le + petit vecteur propre associé n'est pas informatif *(λ = 0)*. On prend le **vecteur propre associé à la deuxième + petite valeur propre**. Les autres vecteurs propres U2,U3, . . . ,Uk+1 associés aux valeurs propres *λ2 ≤ λ3 ≤ . . . ≤ λk+1* forment coordonnées de la matrice multidimensionnelle de taille *n* × *k*. C4 : Big Data : analyse de données massives. Les données massives se distinguent des données qui rentrent en mémoire ou qui peuvent être analysées par un seul ordinateur. Cela requiert des infrastructures à grande échelle et robustes(l'impact que toutes les séquences de défaillance peuvent avoir sur le système). Les 3 « V » - **Volume** : Infrastructures de stockage sophistiquées. (ex : ensemble des produits et des avis Amazon) Algos linéaires en temps. Les 3 « V » - **Variété** : sources de données sont très hétérogènes.(ex : posts Fb peuvent contenir texte, émojis, images, vidéo, fichier audio).Technique d'intégration *ad hoc*(tous ensemble et connectés). Les algorithmes d'analyse sont très différents selon type des données (textuelle, vidéo, audio). Les 3 « V » **Vitesse** : Systèmes **temps réel** (live) et collection de données en **continu** (always on). (ex :flots de données, requêtes Google) Infrastructures sophistiquées pour collecter, indexer, récupérer, et visualiser les données. Utilisateurs souhaitent accéder aux dernières données en temps réel. Ingénierie et technologie. **Big Data limitations** Données massives limitations : **Participation non-représentative** →Données de n'importe quel réseau social ne reflètent pas idées des personnes qui ne l'utilisent pas(ex : IG les jeunes, New York Times les riches). **Données générées par des machines** → Une armée de critiques (reviewers) écrit chaque jour de faux commentaires. De nombreux agents numériques (bots) écrivent et consomment massivement des tweets et autres textes ! 90% des mails envoyés sont des pourriels (spams). Les données nous mentent peut-être! Twitter estime que 23 M de ces utilisateurs actifs sont des agents numériques. **Trop de redondance** → Majorité des données correspondent à objets déjà connus.Plupart des données que nous voyons sont des choses que nous avons déjà vues. La déduplication,c-à-d la suppression des doublons est une étape essentielle de beaucoup d'analyses. (ex : utiliser des photos prises sur internet pour identifier des bâtiments). Exploiter stockage hiérarchique : Les algos d'analyse des données massives sont souvent limités par **stockage** ou le **débit** plutôt que par puissance de calcul(Il faut 30 min pour lire 1 To depuis un HDD, et 5 minutes depuis un SSD). Infrastructure est importante ! car performance dépend + de gestion des données que de qualité des algos. Gestion des données massives : La **latence** suit une remise de volume(le 1er accès aux

données est + coûteux que les accès suivants. **Organiser les calculs** pour tenir compte de cette remise en analysant les données sous la forme d'un flot (stream), en pensant « gros fichier » plutôt que « dossier », en compressant les données. Paradigmes modernes de calculs : *Ordinateurs individuels*→Téléphone ≈ 0.005 TFLOPS, Ordinateur central (mainframe) 143 000 TFLOPS. *Matériel spécialisé*→Concentre sur sous-ensemble d'opérations, les GPUs ex : NVIDIA A100 : 19 TFLOPS (FP32), 156 TFLOPS (TF32 Tensor Core), 312 TFLOPS (TF32 Tensor Core + Sparsity). *Système distribué*→de nombreux ordinateurs « peu puissants » qui travaillent ensemble. Peut atteindre 100 000 TFLOPS. Calcul distribué : UTILES pour données massives et pour accélérer calculs. Calculs + rapides en décomposant problèmes en pls sous-problèmes identiques.Diviser problème en sous-problème + simple à résoudre : **1)** Ordinateurs (peu puissant) résolvent simultanément 1 sous-problème chacun. **2)** Combiner les sln des sous-problèmes pour résoudre le prob initial. Ex :compter le nb de parcmètres à Montréal → **A) Approche centralisée (1 ordinateur)** : 1 marathonien parcourt toute la ville et compte parcmètres. 1 système de comptage automatique à partir d'images satellites. **B)Approche distribuée (beaucoup d'ordinateurs)** :1,000 personnes parcourent une petite zone géographique et comptent les parcmètres. Une fois terminé, chacun envoie son rapport au QG. Parallélisme de donnée : Meilleure façon d'exploiter le parallélisme pour traiter données massives. **Problème** :données trop massives. Envoyer données sur réseau prend du temps. **Solution** : rapprocher calculs des données. Traiter données séquentiellement car recherches sont coûteuses. Stocker données plusieurs fois pour augmenter fiabilité. Algo Big Data classique :**1)** Itérer sur grand nombre d'enregistrements. (Work1, Work2, Work3) **2)** Extraire de chaque enregistrement quelque chose d'intérêt(worker).**3)** Mélanger et trier résultats intermédiaires(r1, r2, r3).**4)** Agréger les résultats intermédiaires.**5)** Générer sortie finale(result). Difficultés de la parallélisation : Cmt assigner les sous-problèmes aux workers. Que faire si + de sous-problèmes que workers. Cmt savoir si tous workers ont fini. SLN → **MapReduce** stocker + traiter big data) : framework permettant distribuer un problème en divisant les calculs en sous-problèmes. Modèle de programmation parallèle simple, mise à l'échelle à des milliers d'ordinateurs simples, tolérance aux panes grâce à de la redondance. Composants de Hadoop : **1)MapReduce**→traitement données massives de manière parallèle/distribuée (fault-tolerant, scheduler, execution). **2) HDFS**→système de fichiers distribué (fault-tolerant, high-bandwidth, high-availability). Map et Reduce : **Map** : fonction exécutée par chaque ordi pour résoudre sous-problème. **Reduce** : fonction combinant solutions de chaque sous-problème en la solution finale. Il y a + de sous-problèmes que de machines dispo. Temps linéairement proportionnel au nb de machines. Si une machine crash, faut recalculer le sous-problème associé. Données sont lues depuis disque au début et écrites à la fin. Services de cloud computing : Plateformes comme Amazon AWS, Google Cloud, Microsoft Azure rendent facile location d'un grand nb de machines pendant une période courte. Coût difficile à évaluer, car pls facteurs: processeurs, cartes graphiques, mémoire vive, stockage à long terme et bande passante. Possible de réduire les coûts pr certains usages : *spot instances* permettent payer uniquement le temps où instances sont utilisées. ***Reserved instances*** permettent réserver des instances pour longue période. **Éthique et société Big Data** : **Transparence et propriété des données**(si organisation suit bonnes pratiques de stockage et d'utilisation des données, si les erreurs peuvent se propager et mécanismes de correction). **Biais des modèles**(algos de machine learning héritent des biais des données d'apprentissage, ex : fil actual qui renforce la polarisation politique, search engine montre meilleures opportunités de job aux hommes qu'aux femmes). **Préserver sécurité données massives**(encrypter et supprimer données pour éviter failles de sécurité→adresses, id, et mdp divulgués persistent des années). Préserver anonymat dans données agrégées(utilisateurs ne doivent pas être identifiables mm si les noms, adresses, et id sont supprimés).C5 : Big Data – SQL : Un **attribut** est un id (un nom) décrivant une info stockée dans une base(ex :âge d'une personne). Le **domaine** d'un attribut est l'ensemble, fini ou infini, de ses valeurs possibles(ex : l'attribut numéro de sécurité sociale a pour domaine l'ensemble des combinaisons de 15 chiffres). Une **relation** est un sous-ensemble du produit cartésien de *n* domaines d'attributs et est représentée sous la forme d'un tableau à 2D dans lequel les *n* attributs correspondent aux titres des *n* colonnes. Une **occurrence**, ou n-uplets (**tuples**), est un élément de l'ensemble figuré par une relation. Une occurrence est une ligne du tableau qui représente la relation. **Schéma de relation** précise nom de relation ainsi que liste des attributs avec leurs domaines. Relation stockée avec service de fichiers répartis : Une relation de schéma Lien(url1 : Chaîne, url2 : Chaîne) décrivant la structure du web contient ≈ 109 occurrences. Cette relation peut être stockée comme un fichier dans un service de fichiers répartis. La relation est partitionnée en plein de petits fichiers qui sont stockés sur plusieurs nœuds. **Algèbre relationnelle-Sélection** ex→ selection (age > 20). Génère relation regroupant exclusivement toutes les occurrences de la relation qui satisfont la condition. **Map** : pour chaque ligne *r* dans la relation, *retourne (r,r)* si et seulement si la condition est satisfaite. Autrement, la fonction ne retourne rien. **Reduce** : retourne la clé reçue en entrée. De manière informelle, la fonction « ne fait rien ». **Algèbre relationnelle – Projection** ex→ projection(Name, isActive). Supprime attributs d'une relation qui ne sont pas sélectionnés et élimine occurrences en double apparaissant dans nouvelle relation. **Map** : pour chaque ligne *r* dans la relation, *retourne (r ′ ,r ′)* tel que *r ′* ne contienne que les attributs sélectionnés. Après suppression des attributs, il est possible que sortie contienne des occurrences en double. **Reduce** : reçoit en entrée *(r ′ , [r ′ ,r ′ ,r ′ , . . .])* et retourne une seule paire *(r ′ ,r ′)*, ce qui supprime les doublons. **Algèbre relationnelle – Union** : opération portant sur 2 relations ayant le même schéma et construisant une 3e relation constituée des occurrences appartenant à chacune des deux relations sans doublon. **Map** : pour chaque ligne *r*, retourne *(r,r)*. **Reduce** : reçoit en entrée *(r ′ , [r ′ ,r ′ ,r ′ , . . .])* et retourne une seule paire *(r ′ ,r ′)*, ce qui supprime les doublons. **Algèbre relationnelle – Intersection** : opération portant sur 2 relations ayant le même schéma et construisant une 3ième relation dont les occurrences sont constituées de ceux appartenant aux deux relations. **Map** : pour chaque ligne *r*, *retourne (r,r)*. **Reduce** : chaque clé peut être associée à 1 ou 2 valeurs (car nous supposons qu'il n'y a pas de doublons dans relations). Dans le cas où il y a 2 valeurs, la fonction *retourne (r,r)*, autrement rien n'est retourné. **Algèbre relationnelle - Jointure naturelle** : EXEMPLE : **P = M * N** $p_{i,k} = \sum_j m_{i,j} \times n_{j,k}$ La jointure retourne (i, j, k, $m_{i,j}$, $n_{j,k}$). **Grouper** selon les attributs *i* et *k*, puis **écraser** en faisant somme des produits $m_{i,j} \times n_{j,k}$. **Map** retourne l'ensemble des données pour calculer chaque élément de *P* . Chaque élément de M et N contribue à plusieurs éléments de *P* et doit être retourné plusieurs fois. Chaque clé *(i, k)* est associée à une liste contenant les tuples *("M", j, mij)* et *("N", j, njk)* pour toutes les valeurs de *j*. **Reduce** doit connecter les 2 valeurs $m_{i,j}$ et $n_{j,k}$ qui ont le même j, pour chaque j. Ces valeurs sont multipliées, et l'ensemble des produits est sommé pour obtenir $p_{i,k}$. **Jointure naturelle** : opération portant sur 2 relations contruisant une 3e relation **regroupant tt les possibilités de combinaison** des occurrences des relations qui satisfont un test d'égalité entre les attributs qui portent le même nom dans les relations. Ces attributs ne sont pas dupliqués, mais fusionnés en une seule colonne par couple d'attributs. **MapReduce** : Si 2 valeurs avec la même clé proviennent de relations différentes, alors faut créer une occurrence contenant ces 2 valeurs. La jointure peut faire exploser le nb d'occurrences puisqu'il est possible de créer chaque combinaison possible entre les occurrences des deux relations. **Map** : pour les 2 relations *Table1(A, B)* et *Table2(B, C)*, la fonction Map retourne *(b,[T1, a])* ou *(b,[T2, c])* selon la relation d'origine. **Reduce** : Pour chaque clé *b*, la fonction construit toutes les paires possibles contenant une valeur de chaque relation. La fonction retourne toutes les combinaisons sous la forme *(b, [a, c])* représentant une occurrence *(a, b, c)* dans la nouvelle relation. Si une clé ne contient que des valeurs de T1 ou T2, alors la fonction Reduce ne retourne rien. **Algèbre relationnelle - Regrouper et agréger** : regroupe occurrences selon un ensemble d'attributs et effectue une opération d'agrégation (**sum, count, max, min**) pour chaque groupe sur un autre attribut. *Supposons que nous regroupons les occurrences selon l'attribut A et que nous agrégeons selon l'attribut B. **Map** : Pour chaque ligne (a,b,c) dans la relation, retourne (a,b). **Reduce** : Chaque clé ***a*** représente un groupe. La fonction applique l'opération d'agrégation (**sum, count, max, min**) sur les valeurs et retourne le résultat. C5-suite : **MapReduce - Produit Matriciel** : **Cas simple** : Le vecteur *v* est stocké entièrement dans la mémoire de chaque worker. Supposons que les données de M et v soient stockées sous la forme *(i, j, mij)* et *(j, vj)*. **Map** retourne (i, mij × vj). **Reduce** retourne la somme de toutes les valeurs associées à la clé i. **Map(Tuple (i, j, mij))** // Suppose que v soit stocké en mémoire emit(i, mij × vj). **Reduce(Int i, List products) -- xi ← sum(products) -- emit(i, xi)**. Cas réaliste : Le vecteur v ne peut pas être stocké entièrement dans mémoire de chaque worker. Il suffit lire le vecteur v depuis disque chaque fois que nécessaire → cela entraîne très grand nb de communications et une **baisse des performances**! SLN : découper M en stripes verticales de taille égale, et v en un nombre identique de stripes horizontales. La i-ème stripe de M ne multiplie que la i-ème stripe v. Chaque Map reçoit un stripe de M et de v. **Map et Reduce** fonctionnent de la même manière que dans cas simple.**1) Jointure naturelle** : Faire la jointure naturelle de M et N, c-à-d selon l'attribut j. La jointure retourne (i, j, k, $m_{i,j}$, $n_{j,k}$). Grouper selon les attributs i et k, puis agréger en faisant la somme des produits mij × njk. **Pour la matrice M** : **Map(Tuple (i, j, mij)) --emit(j, ("M", j, mij))**. **Pour matrice N** : **Map(Tuple (j, k, njk)) -- emit(j, ("N", k, njk))**. **Reduce(Int j, List values) -- for each ("M", i, mij) and ("N", k, njk) in values do -- emit((i, k), mij × njk) -- end. // Pour chaque combinaison de (i, k)**.**2) Groupe et agrège** : **Map(Tuple (i, k), Int pik) -- //** Fonction identité -- emit((i, k), pik). **Reduce(Tuple (i, k), List produits) -- pik ← sum(produits) -- emit((i, k), pik)**. **Solution en une étape** : **Map** retourne l'ensemble des données nécessaires pour calculer chaque élément de **P**. Chaque élément de **M** et **N** contribue à plusieurs éléments de **P** et doit donc être retourné plusieurs fois. Chaque clé *(i, k)* est associée à une liste contenant les tuples *("M", j, mij)* et *("N", j, njk)* pour toutes les valeurs de *j* possible. **Reduce** doit connecter les 2 valeurs *mij* et *njk* qui ont le même *j*, pour chaque *j*. Ces valeurs sont ensuite multipliées, et l'ensemble des produits est sommé pour obtenir *pik* . **matrice M** : **Map(Tuple (i, j, mij)) -- // Pour chaque colonne de N -- for k = 0, 1, ...,r do -- emit((i, k), ("M", j, mij))-- end**. **matrice N** : **Map(Tuple (j, k, njk)) -- // Pour chaque ligne de M -- for i = 0, 1, ..., p do -- emit((i, k), ("N", j, njk)) -- end**. **Reduce(Tuple (i, k), List values) -- m ← dict() -- n ← dict() -- for matrix, j, value do -- if matrix is equal to "M" then -- m[j] ← value -- end -- if matrix is equal to "N" then -- n[j] ← value -- end -- end -- pik ← 0 -- for all j do -- pik ← pik + m[j] × n[j] -- end -- emit((i, k), pik)**.C6 : Régression linéaire(mesurée en 2 pts slm) : Soient X un ensemble de données et y les **valeurs numériques à prédire**. La régression linéaire permet trouver l'hyperplan qui prédit au mieux *yi* en fonction de *Xi*(Ex : Les prix des logements augmentent linéairement avec la superficie, poids augmente linéairement avec glaces). Mesurer l'erreur : L'**erreur résiduelle** est la différence entre valeurs prédites et valeurs réelles : $r_i = f(X_i) - y_i$. Régression linéaire, ici nous voulons bien minimiser : $O = \sum_{i=1}^{n} (f(X_i) - y_i)^2$, où $f(X_i) = w_0 + \sum_{j=1}^{d} w^j X_i^j$ pk les erreurs au carré ? Pénaliser les grandes erreurs, éviter d'avoir de grandes erreurs. Le **coefficient *w0* correspond à l'ordonnée à l'origine *(y-intercept)*** de la droite de la régression. On n'a pas besoin de *w0* si les données sont centrées. Régression linéaire – améliorations : suppression des valeurs aberrantes (outliers), ajustement par des fonctions non linéaires, mise à l'échelle variables dépendantes et indépendantes, nettoyage des attributs fortement corrélés. Suppression de valeurs aberrantes : En raison du poids quadratique des résidus, points périphériques peuvent affecter bcp la régression. Identifier points périphériques et supprimer de manière raisonnée peut rendre l'ajustement plus robuste. Régression linéaire est sensible aux données aberrantes. On peut repérer ces valeurs erronnées et les supprimer. Ajustement des fonctions non linéaires : régression linéaire ajuste données par des droites uniquement! Les ajuster aussi par des fonctions quadratiques en créant une autre variable avec la valeur x^2 dans notre matrice de données. Le modèle 1-D $f(x) = w_1 x + w_2 x^2$ est quadratique, mais linéaire en *w*. Nous pouvons faire de même avec des fonctions arbitraires en ajoutant les colonnes correspondantes dans X :ex $(\sqrt{x}, \log x, x^3, \frac{1}{x})$. **L'inclusion explicite de tous les termes non linéaires possibles n'est pas pratique**. Mise à l'échelle : Des attributs sur de larges intervalles numériques (ex : population nationale par opposition aux fractions des gens) requièrent des coefficients à différentes échelles pour être rassemblés (ex : f (x) = 0.03 × x1 + 300000000 × x2). Difficile à savoir quel attribut est le + important pour la régression(ex: x1: 100 000 et x2: 0.1) on sait pas lequel est + important). Apporte des problèmes de précision. **SLN→** normaliser attributs de la matrice de données ! Mise à l'échelle sous-linéaire Ex : modèle linéaire qui prédit années d'éducation d'un enfant basé sur revenu familial.Enfants Bill Gates en cmb d'années? Normalisation d'un attribut distribué selon loi de puissance est inutile car elle ne fait qu'une transformation linéaire des valeurs de l'attribut. Un écart énorme entre les valeurs les + grandes/les + petites et les médianes signifie qu'aucun coefficient ne peut utiliser la fonction de régression sans exploser sur les grandes valeurs.**Sln→** remplacer ces attributs x par des attributs sous-linéaires comme **log x et $\sqrt{x}, \sqrt[2]{y}$.** Essayer prédire grands revenus avec attributs normalisés est problématique(ex : obtenir 100K à partir d'attributs de petite échelle). Considérer le **logarithme de grande valeur cible** donne de meilleurs modèles. On ne peut pas ajuster f (x) = 20000x1x2 cette fonction par régression linéaire sur x1 et x2. On sait que log f (x) = log(20000x1x2) = log(20000) + log(x1) + log(x2). On peut ajuster cette fonction avec log x1, log x2 comme colonnes de la matrice X. Nettoyage des attributs fortement corrélés Ex : 2 attributs parfaitement corrélés(hauteur en pieds/mètres). Attributs parfaitement corrélés ne fournissent aucune information supplémentaire pour modélisation (**sinon, on les ajouterait successivement pour obtenir un modèle parfait !**). la matrice $X^T X$ n'est pas inversible ($X^T X$ a des lignes dépendantes).**SLN→**identifier attributs hautement corrélés. N'importe quel attribut peut être retiré sans grandes conséquences. On peut aussi réduire la dimension des données automatiquement par SVD ou PCA. Régression comme problème d'optimisation : Calculer coefficients par l'équation : $w = (X^T X)^{-1} X^T y$ pose problèmes : Calculer l'inverse d'une matrice est coûteux et mène à l'instabilité numérique. La matrice $X^T X$ peut être même singulière. Nous cherchons des **coefficients** qui minimisent l'erreur quadratique moyenne des points : $J(w) = \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - y_i)^2$ où la droite de la régression est donnée par $f(X_i) = \sum_{j=1}^{d} w^j X_i^j$ La fonction d'erreur *J* est **convexe** : Le seul optimum local est un optimum global. Quand un espace de recherche est convexe, facile de trouver minimum : continuer simplement à descendre. La direction la + rapide vers le minimum est définie par le **gradient**. Cela motive l'approche de la **descente du gradient** pour résoudre la régression. Descente du gradient : gradient obtenu par calcul de la dérivée partielle de *J(w)* pour chaque coefficient w^j. $\frac{\partial J}{\partial w^j} = 2 * \sum_{i=1}^{n} X_i^j (f(X_i) - y_i)$ L'évaluation de chaque dérivée partielle prend un temps linéaire en termes du nb *n* d'enregistrements ! Descente du gradient stochastique : n'utiliser que quelques enregistrements n ′ < n pour estimer la dérivée. L'ajuste du taux d'apprentissage et de taille du batch (n ′) (hyperparamètres) conduit à une optimisation très rapide des fonctions convexes. Régularisation : Ajouter pénalités à la fonction objectif cherchant à garder les coefficients petits → $J(w) = \sum_{i=1}^{n} (f(X_i) - y_i)^2 + \lambda \sum_{i=1}^{d} (w^j)^2$. Récompense mise à zéro des coefficients. Interprétation/Pénalisation : **Ridge régression** : pénalisations des carrées(mise à zéro des coefficients).**LASSO régression** : pénalisations valeurs absolues → $J(w, t) = \sum_{i=1}^{n} (f(X_i) - y_i)^2$

5 à 7 étapes prétraitement des données : 1) **uniformiser dates** afin qu'elle soit au même format que autres valeurs. 2) **Convertir date (chaîne de caractères) en temps UTC**/UNIX interprétable par la régression linéaire. 3) Ajuster montant prêt et salaires en fonction de la date du prêt afin **tenir compte de l'inflation**. 4) **Affecter valeur manquante**, ex :valeur « vide » par la valeur majoritaire « BAC », car régression linéaire ne supporte pas valeurs manquantes. 5) **Normaliser attributs numériques** (âge, salaire, date du prêt, montant du prêt) afin d'éviter instabilités numériques et de pouvoir comparer les poids de la régression. 6) **Mettre en majuscule toutes les valeurs** de diplômes afin d'éviter que « PhD » et « PHD » soient considérés comme valeurs différentes. 7) **Binariser attribut diplôme, car régression linéaire n'accepte pas chaînes caractères en entrée, mais slm les valeurs numériques.** 8)**Identifier attributs hautement liés et les éliminer**, car il ne contribuent pas à la prédiction.