Université Sorbonne

Master DAC

Département informatique

BDLE

Devoir Maison n°2

18 novembre 2022

par

Oussama Sahli

Encadrant universitaire : Hubert Naacke/ Camelia Constantin/ Mohamed-amine Baazizi/ Bernd Amann

Table des matières

Table des figures

Résumé

Les entreprises et les institutions modernes s'appuient sur les données pour guider chaque processus commercial et chaque décision. Des informations manquantes ou incorrectes compromettent sérieusement tout processus de décision en aval. Par conséquent, une tâche cruciale, mais fastidieuse, pour toute personne impliquée dans le traitement des données, est de vérifier la qualité de leurs données. Nous présentons un système pour automatiser la vérification de la qualité des données à l'échelle, qui répond aux exigences des cas d'utilisation en production. Notre système fournit une API déclarative, qui combine des contraintes de qualité communes avec du code de validation défini par l'utilisateur, et permet ainsi des "tests unitaires" pour les données. Nous exécutons efficacement la charge de travail de validation des contraintes qui en résulte en la traduisant en requêtes d'agrégation sur Apache Spark. Notre plateforme prend en charge la validation incrémentale de la qualité des données sur des ensembles de données de plus en plus nombreux, et exploite l'apprentissage automatique, par exemple, pour améliorer les suggestions de contraintes, pour estimer la "prévisibilité" d'une colonne, et pour détecter les anomalies dans les séries chronologiques historiques de qualité des données. Nous discutons de nos décisions de conception, décrivons l'architecture du système résultant et présentons une évaluation expérimentale sur divers ensembles de données.

Format de référence PVLDB : Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann et Andreas Grafberger. Automatisation de la vérification de la qualité des données à grande échelle. PVLDB, 11 (12) : 1781-1794, 2018. DOI : https ://doi.org/10.14778/3229863.3229867

Chapitre 1

Introduction

Les données sont au centre des entreprises et des institutions modernes. Les détaillants en ligne, par exemple, s'appuient sur les données pour aider les décisions d'achat des clients, pour prévoir la demande [7], pour programmer les livraisons et, plus généralement, pour guider chaque processus et décision de l'entreprise. Des informations manquantes ou incorrectes compromettent gravement tout processus décisionnel en aval, ce qui nuit finalement à l'efficacité et à l'efficience globales de l'organisation. La qualité des données a des effets [1] à travers les équipes et les frontières organisationnelles, en particulier dans grandes organisations dotées de systèmes complexes qui entraînent des dépendances de données complexes. En outre, on observe une tendance dans différentes industries vers une plus grande automatisation des processus métier à l'aide de techniques d'apprentissage machine (ML). Ces techniques sont souvent très sensibles aux données d'entrée, puisque les modèles déployés reposent sur des hypothèses fortes concernant la forme de leurs données d'entrée [42], et les erreurs subtiles introduites par des changements des données peuvent être très difficiles à détecter [34]. En même temps, il est largement prouvé que le volume de données disponibles pour l'entraînement est souvent un facteur décisif pour les performances d'un modèle [17, 44]. Dans les infrastructures d'information modernes, les données se trouvent à différents (par exemple, dans des bases de données relationnelles, dans des "lacs de données" sur des systèmes de fichiers distribués, derrière des systèmes d'archivage, etc. de données " sur des systèmes de fichiers distribués, derrière des API REST ou encore ou sont constamment extraites de ressources web) et se présentent dans de nombreux formats différents. Nombre de ces sources de données ne prennent pas en charge les contraintes d'intégrité et les contrôles de qualité des données, et souvent il n'y a même pas

1. travail effectué chez Amazon Research

de schéma d'accompagnement disponible, car les données sont consommées à la manière d'un "schéma à la lecture", où une une application particulière se charge de l'interprétation. En outre, il existe une demande croissante d'applications consommant des données semi-structurées telles que du texte, des vidéos et des images. Dans ces conditions, toutes les équipes et tous les systèmes impliqués dans le traitement des données doivent, d'une manière ou d'une autre, prendre en charge la validation des données, ce qui entraîne souvent un travail fastidieux et répétitif. À titre d'exemple concret, imaginez une plateforme de vidéo à la demande où les machines qui diffusent des vidéos aux utilisateurs écrivent des fichiers journaux sur l'utilisation de la plateforme. Ces fichiers journaux doivent régulièrement être ingérés dans un magasin de données central pour rendre les données pour une analyse plus approfondie, par exemple, comme données d'entraînement pour les systèmes de recommandation. Comme toutes les données produites dans le monde réel, ces fichiers journaux peuvent présenter des problèmes de qualité des données, par exemple en raison de bogues dans le code du programme ou de changements dans la sémantique des attributs. Ces problèmes peuvent entraîner l'échec du processus d'ingestion. Même si le processus d'ingestion fonctionne toujours, les erreurs dans les données peuvent provoquer des erreurs inattendues dans les systèmes en aval qui consomment les données. Par conséquent, l'équipe qui gère le processus d'ingestion quotidien doit valider la qualité des données des fichiers journaux, par exemple en vérifiant l'absence de valeurs manquantes, d'enregistrements en double ou d'anomalies de taille.

Nous postulons donc qu'il existe un besoin urgent d'une automatisation accrue de la validation des données. Nous présentons un système que nous avons construit pour cette tâche et qui répond aux exigences des cas d'utilisation en production. Le système est construit sur les principes suivants : au cœur de notre système se trouve la déclarativité. Nous voulons que les utilisateurs passent du temps à réfléchir à l'aspect de leurs données et qu'ils n'aient pas à se préoccuper de la manière de mettre en œuvre les contrôles de qualité réels. Par conséquent, notre système offre une API déclarative qui permet aux utilisateurs de définir des contrôles sur leurs données en composant une grande variété de contraintes disponibles. De plus, les outils de validation des données doivent offrir une grande flexibilité à leurs utilisateurs. Les utilisateurs doivent être en mesure d'exploiter des données externes et du code personnalisé pour la validation (par ex, appeler un service REST pour certaines données et écrire une fonction complexe qui compare le résultat à une statistique calculée sur les données). Notre vision est que les utilisateurs devraient être

en mesure d'écrire des "tests unitaires" pour les données (section 3.1), par analogie avec les pratiques de test établies dans le génie logiciel. En outre, les systèmes de validation des données doivent tenir compte du fait que les données sont produites en permanence. Ils doivent donc permettre l'intégration d'ensembles de données croissants. Le système que nous proposons prend explicitement en charge le calcul incrémental des mesures de qualité sur des ensembles de données croissants (section 3.2), et permet à ses utilisateurs d'exécuter des algorithmes de détection d'anomalies sur les séries temporelles historiques de mesures de qualité (section 3.4). Le dernier principe à aborder est l'évolutivité : les systèmes de validation des données doivent s'adapter de manière transparente aux grands ensembles de données. Pour répondre à cette exigence, nous avons conçu notre système pour traduire les calculs de métriques de données nécessaires en requêtes d'agrégation, qui peuvent être exécutées efficacement à l'échelle moteur de flux de données distribué tel que Apache Spark [50].

Les contributions de cet article sont les suivantes ing :

- Nous présentons une API déclarative combinant des contraintes de qualité communes avec un code de validation défini par l'utilisateur, ce qui permet de réaliser des "tests unitaires" pour les données (section 3.1).
- Nous discutons de la manière d'exécuter efficacement la validation des contraintes en traduisant les vérifications en requêtes d'agrégation avec un support dédié pour le calcul incrémental sur des ensembles de données croissants (sections 3.2 et 4).
- Nous donnons des exemples de la façon dont l'apprentissage automatique peut être exploité dans la vérification de la qualité des données, par exemple pour améliorer les suggestions de contraintes, pour estimer la prédictibilité d'une colonne, et pour détecter les anomalies dans les séries chronologiques historiques de la qualité des données (sections 3.2 et 4). section 3.3 et 3.4
- Nous présentons une évaluation expérimentale de nos approches proposées (Section 5).

Chapitre 2

DIMENSIONS DE LA QUALITÉ DES DONNÉES

Nous commençons par examiner la littérature sur la qualité des données pour comprendre les dimensions communes de la qualité. La qualité des données peut se référer à l'extension des données (c.-à-d. les valeurs des données) ou à l'intention des données (c.-à-d. le schéma) [4]. Nous nous concentrons sur la qualité des données extensionnelles dans ce qui suit. Nous traitons les problèmes de schéma avec le concept de complétude. Par exemple, si aucune valeur d'attribut n'est spécifiée dans le schéma ouvert d'une entité, nous la considérons comme manquante. Il existe de nombreuses études sur la mesure de la qualité des données extensionnelles [4, 30, 39]. Dans ce qui suit, nous décrivons brièvement les dimensions les plus courantes. La complétude fait référence au degré auquel une entité comprend les données nécessaires pour décrire un objet du monde réel. Dans les tables des systèmes de bases de données relationnelles, l'exhaustivité peut être mesurée par la présence de valeurs nulles, ce qui est généralement interprété comme une valeur manquante. Dans d'autres contextes, par exemple dans un catalogue de produits, il est important de calculer la complétude en tenant compte du contexte correct, c'est-à-dire du schéma de la catégorie de produits. Par exemple, l'absence d'une valeur pour l'attribut taille de chaussure n'est pas pertinente pour les produits de la catégorie ordinateurs portables. Dans ce cas, la valeur de l'attribut est manquante car l'attribut n'est pas applicable [37]. Nous sommes seulement intéressés à mesurer la complétude lorsqu'un attribut est applicable. La cohérence est définie comme le degré de violation d'un ensemble de règles sémantiques. Les contraintes intra-relationnelles définissent une plage de valeurs admissibles, comme un type de données spécifique, un intervalle pour une colonne numérique ou

un ensemble de valeurs pour une colonne catégorielle. Elles peuvent également impliquer des règles sur plusieurs colonnes, par exemple, ”si la catégorie est t-shirts, alors l'intervalle de la taille est S, M, L”. Les contraintes d'interrelation peuvent impliquer des colonnes de plusieurs tables. Par exemple, une colonne customerId ne peut inclure que les valeurs d'une table de référence de tous les clients. La précision est l'exactitude des données et peut être mesurée selon deux dimensions : syntaxique et sémantique. syntaxique compare la représentation d'une valeur avec un domaine de définition correspondant, tandis que la précision sémantique compare une valeur avec sa représentation dans le monde réel. Par exemple, pour l'attribut couleur d'un produit, une valeur bleue peut être être considérée comme syntaxiquement exacte dans le domaine donné même si la valeur correcte serait rouge, alors qu'une valeur XL serait considérée comme n'étant ni sémantiquement ni syntaxiquement exacte.

Chapitre 3

APPROCHE

Nous présentons notre mécanisme général de vérification automatisée de la qualité des données à grande échelle. Nous présentons d'abord notre API déclarative, qui permet aux utilisateurs de spécifier des contraintes sur leurs ensembles de données, et nous détaillons comment nous traduisons ces contraintes en calculs de métriques sur les données, ce qui nous permet d'évaluer ultérieurement les contraintes (Section 3.1). Ensuite, nous discutons de la manière d'étendre cette approche à des scénarios dans lesquels nous devons évaluer de telles contraintes pour des données à croissance incrémentale (par exemple, dans le cas de pipelines d'ingestion dans un entrepôt de données) ; dans la section 3.2. Enfin, nous décrivons les extensions de notre approche telles que la suggestion de contraintes (Section 3.3) et la détection d'anomalies sur les séries temporelles d'un ensemble de données (Section 3.4).

3.1 Des "tests unitaires" pour les données

L'idée générale derrière notre système est de permettre aux utilisateurs de définir facilement des "tests unitaires" pour leurs ensembles de données de manière déclarative [10]. Ces "tests unitaires" consistent en des contraintes sur les données qui peuvent être combinées avec des fonctions définies par l'utilisateur, Par exemple, du code personnalisé. Le tableau 1 3.1 montre les contraintes disponibles pour nos utilisateurs. Nous voulons qu'ils se concentrent sur la définition de leurs contrôles et de leur code de validation, mais pas sur le calcul des métriques requises pour les contraintes. Par conséquent, nous concevons notre système pour traduire les contraintes définies par l'utilisateur en un calcul de métrique efficacement exécutable. Définition déclarative des contraintes de qualité des

données. Dans notre système, les utilisateurs définissent des contrôles pour leurs ensembles de données, ce qui entraîne des erreurs ou des avertissements pendant l'exécution, si la validation échoue. Cette approche offre une grande flexibilité aux utilisateurs : ils peuvent écrire des fonctions complexes et utiliser des bibliothèques existantes pour leur code de validation ; ils peuvent utiliser des données externes ou même appeler des services externes. Afin de présenter notre système, nous introduisons un cas d'utilisation exemplaire sur une plateforme de vidéo à la demande. Supposons que les machines qui diffusent des vidéos aux utilisateurs écrivent des fichiers journaux sur l'utilisation de la plateforme, avec des détails tels que le type d'appareil utilisé, la durée de la session, la durée de la session, l'identifiant du client ou le lieu. Ces fichiers journaux doivent être régulièrement ingérés

Table 1: Constraints available for composing user-defined data quality checks.

constraint	arguments	semantic
dimension *completeness*		
`isComplete`	column	check that there are no missing values in a column
`hasCompleteness`	column, udf	custom validation of the fraction of missing values in a column
dimension *consistency*		
`isUnique`	column	check that there are no duplicates in a column
`hasUniqueness`	column, udf	custom validation of the unique value ratio in a column
`hasDistinctness`	column, udf	custom validation of the unique row ratio in a column
`isInRange`	column, value range	validation of the fraction of values that are in a valid range
`hasConsistentType`	column	validation of the largest fraction of values that have the same type
`isNonNegative`	column	validation whether all values in a numeric column are non-negative
`isLessThan`	column pair	validation whether values in the 1s column are always less than in the 2nd column
`satisfies`	predicate	validation whether all rows match predicate
`satisfiesIf`	predicate pair	validation whether all rows matching 1st predicate also match 2nd predicate
`hasPredictability`	column, column(s), udf	user-defined validation of the predictability of a column
statistics (can be used to verify dimension *consistency*)		
`hasSize`	udf	custom validation of the number of records
`hasTypeConsistency`	column, udf	custom validation of the maximum fraction of values of the same data type
`hasCountDistinct`	column	custom validation of the number of distinct non-null values in a column
`hasApproxCountDistinct`	column, udf	custom validation of the approx. number of distinct non-null values
`hasMin`	column, udf	custom validation of a column's minimum value
`hasMax`	column, udf	custom validation of a column's maximum value
`hasMean`	column, udf	custom validation of a column's mean value
`hasStandardDeviation`	column, udf	custom validation of a column's standard deviation
`hasApproxQuantile`	column, quantile, udf	custom validation of a particular quantile of a column (approx.)
`hasEntropy`	column, udf	custom validation of a column's entropy
`hasMutualInformation`	column pair, udf	custom validation of a column pair's mutual information
`hasHistogramValues`	column, udf	custom validation of column histogram
`hasCorrelation`	column pair, udf	custom validation of a column pair's correlation
time		
`hasNoAnomalies`	metric, detector	validation of anomalies in time series of metric values

FIGURE 3.1 – Tableau 1 : Contraintes disponibles pour composer des contrôles de qualité des données définis par l'utilisateur.

dans un magasin de données central afin de rendre les données disponibles pour une analyse plus approfondie, Par exemple, comme données d'entraînement pour les systèmes de recommandation. Analogues à toutes les données produites dans des scénarios réels, ces fichiers journaux peuvent présenter des problèmes de qualité des données, par exemple en raison de bogues dans le code du programme, de pertes de données, de redéploiements de

```
val numTitles = callRestService(...)
val maxExpectedPhoneRatio = computeRatio(...)

var checks = Array()

checks += Check(Level.Error)
 .isComplete("customerId", "title",
  "impressionStart", "impressionEnd",
  "deviceType", "priority")
 .isUnique("customerId", "countryResidence",
  "deviceType", "title")
 .hasCountDistinct("title", _ <= numTitles)
 .hasHistogramValues("deviceType",
   _.ratio("phone") <= maxExpectedPhoneRatio)

checks += Check(Level.Error)
 .isNonNegative("count")
 .isLessThan("impressionStart", "impressionEnd")
 .isInRange("priority", ("hi", "lo"))

checks += Check(Level.Warning, on="delta")
 .hasNoAnomalies(Size, OnlineNormal(stdDevs=3))
checks += Check(Level.Error, on="delta")
 .hasNoAnomalies(Size, OnlineNormal(stdDevs=4))

checks += Check(Level.Warning)
 .hasPredictability("countryResidence",
   ("zipCode", "cityResidence"), precision=0.99)

Verification.run(data, checks)
```

FIGURE 3.2 – Listing 1 : Exemple de définition de contraintes déclaratives de qualité des données à l'aide de notre API.

services ou de changements dans la sémantique des colonnes de données. Ces problèmes peuvent potentiellement avoir des conséquences négatives, par exemple l'échec du processus d'ingestion et la nécessité de devoir le redémarré manuellement après communication avec le fournisseur de données. Même si le processus d'ingestion fonctionne toujours, les erreurs dans les données pourraient provoquer des erreurs inattendues dans les systèmes en aval qui consomment les données. Dans de nombreux cas, ces erreurs peuvent être difficiles à détecter, par exemple, elles peuvent entraîner des régressions dans la qualité de prédiction d'un modèle d'apprentissage automatique, qui fait des hypothèses sur la forme de caractéristiques particulières calculées à partir des données d'entrée [34]. Par conséquent, le service de streaming vidéo pourrait utiliser notre système pour valider la qualité des données avant de commencer le processus d'ingestion des données, en déclarant un ensemble personnalisé de vérifications à effectuer sur les données. La liste 1 présente un

exemple de ce que pourrait être un tel contrôle de qualité déclaratif pour les journaux de flux vidéo et met en évidence la combinaison de définitions de contraintes déclaratives et de code personnalisé. Des données externes sont récupérées au début et utilisées tout au long du contrôle de qualité. REST est appelé pour déterminer le nombre total de films dans le système et le ratio attendu de spectateurs de smartphones est calculé (voir lignes 1 et 2). Ensuite, un ensemble de contrôles d'exhaustivité et de cohérence est défini, par exemple, les colonnes customerId, title, impressionStart, impressionEnd, deviceType et priority doivent être complètes (lignes 7 à 9), et nous imposons que la combinaison de colonnes clientId, countryResidence, deviceType et title soit unique dans les données à disposition (lignes 10 et 11). Nous nous assurons que le nombre de valeurs distinctes dans la colonne titre est inférieur ou égal au nombre total de films dans le système (ligne 12) et nous vérifions que la proportion d'appareils " téléphone " répond à nos attentes en étudiant un histogramme de la colonne deviceType aux lignes 13 et 14. Par la suite, nous émettons un autre ensemble de contrôles de cohérence qui définissent la forme attendue des données (par exemple, aucune valeur négative dans la colonne count, une relation happens-before entre les horodateurs de visualisation et un ensemble de valeurs valides pour la colonne lignes 16 à 19).

Ensuite, nous avons deux vérifications qui reposent sur des comparaisons avec des métriques calculées précédemment sur des versions antérieures de l'ensemble de données (disponibles dans une "base de données métriques" centrale) : nous conseillons au système de détecter les anomalies dans la série chronologique des tailles des enregistrements qui ont été ajoutés à l'ensemble de données au fil du temps et d'émettre un avertissement si la taille s'éloigne de plus de trois écarts types de la moyenne précédente et d'émettre une erreur si elle s'éloigne de plus de quatre écarts types (cf. lignes 21 à 24). Enfin, nous définissons un contrôle de prévisibilité pour la colonne countryResidence, qui stipule que notre système doit être capable de prédire les valeurs de cette colonne avec une précision de 99 % en inspectant les valeurs correspondantes dans les colonnes zipCode et cityResidence.

Traduire les contraintes en calculs métriques. Dans ce qui suit, nous détaillons comment notre système exécute la vérification de la qualité des données. La définition déclarative des contraintes (qui sont évaluées par le code utilisateur) repose sur un ensemble particulier de métriques de qualité des données que notre système calcule à partir des données

Table 2: Computable metrics to base constraints on.

metric	semantic
dimension *completeness*	
`Completeness`	fraction of non-missing values in a column
dimension *consistency*	
`Size`	number of records
`Compliance`	ratio of columns matching predicate
`Uniqueness`	unique value ratio in a column
`Distinctness`	unique row ratio in a column
`ValueRange`	value range verification for a column
`DataType`	data type inference for a column
`Predictability`	predictability of values in a column
statistics (can be used to verify dimension *consistency*)	
`Minimum`	minimal value in a column
`Maximum`	maximal value in a column
`Mean`	mean value in a column
`StandardDeviation`	standard deviation of the value distribution in a column
`CountDistinct`	number of distinct values in a column
`ApproxCountDistinct`	number of distinct values in a column estimated by a hyperloglog sketch [21]
`ApproxQuantile`	approximate quantile of the value in a column [15]
`Correlation`	correlation between two columns
`Entropy`	entropy of the value distribution in a column
`Histogram`	histogram of an optionally binned column
`MutualInformation`	mutual information between two columns

FIGURE 3.3 – Tableau 2 : Métriques calculables sur lesquelles baser les contraintes

disponibles. Le système inspecte les contrôles et leurs contraintes, et collecte les métriques nécessaires à l'évaluation des contrôles. Le tableau 2 énumère toutes les métriques de qualité des données prises en charge par notre système. Nous traitons directement les dimensions de qualité des données que sont la complétude et la cohérence, énumérées dans la section 2. Soit D l'ensemble de données avec N enregistrements, sur lequel nous opérons, et que cv dénote la cardinalité de la valeur v dans une colonne particulière de l'ensemble de données D. En outre, laissons V désigner l'ensemble de valeurs uniques dans une colonne particulière de l'ensemble de données D. Nous calculons la complétude comme la fraction de valeurs non manquantes dans une colonne : $|\{d \in D | d(col) \neq null\}|/N$. Pour la mesure de la cohérence, nous fournissons des métriques sur le nombre de valeurs uniques,

les types de données, la taille de l'ensemble de données, les plages de valeurs, et une métrique générale de correspondance des prédicats. La métrique de la taille par exemple, se réfère au nombre d'enregistrements N, tandis que la métrique de conformité dénote le ratio d'enregistrements qui satisfont un prédicat particulier : $|\{d \in D|p(d)\}|/N$. La métrique Unicité désigne le rapport des valeurs uniques [19] dans une colonne particulière : $|\{v \in V|c_v = 1\}|/|V|$, tandis que la Distinction correspond au rapport unique de rangées |V | / N dans la colonne. De plus, nous implémentons des statistiques sommaires standard pour colonnes numériques, qui peuvent être utilisées pour définir des règles sémantiques supplémentaires sur les ensembles de données, telles que Minimum, Maximum, Moyenne, écart-type, histogramme et entropie, que nous calculons par exemple comme $-\sum_v \frac{C_v}{N} log(\frac{C_v}{N})$. Nous incluons également des statistiques standard telles que la Corrélation et l'information mutuelle pour mesurer le degré d'association entre deux colonnes, où cette dernière est calculée comme suit : $\sum_{v1} \sum_{v2} \frac{C_{v1v2}}{N} log(\frac{C_{v1v2}}{C_{v1}C_{v2}})$. Comme certaines métriques sont assez coûteuses à calculer et impliquer le re-partitionnement ou le tri des données, notre système fournit des approximations de mesures telles que les quantiles sous la forme ApproxQuantile (calculé via un algorithme en ligne efficace [15]) ou ApproxCountDistinct pour estimer le nombre de valeurs distinctes avec une esquisse hyperlogarithmique [21].

Enfin, nous proposons une implémentation de la Prédictibilité. Dans une tentative d'automatiser la vérification de l'exactitude des valeurs, nous formons un modèle d'apprentissage automatique qui prédit une valeur pour une colonne cible t d'un enregistrement particulier à partir de toutes les k valeurs observées $l1, ..., lk \in Vt$ dans la colonne cible, étant donné les valeurs correspondantes li1 , . . . , lin des colonnes d'entrée i1, . . . , in pour l'enregistrement particulier, par exemple, en utilisant la règle de décision a posteriori : argmaxk p(lk|li1 , . . . , lin). Un exemple serait de prédire la valeur d'une colonne "couleur" dans une table de produits à partir du texte des colonnes "description" et "nom". Nous entraînons ce modèle sur un échantillon de valeurs observées dans la colonne cible, et nous mesurons sa qualité de prédiction sur le reste des données retenues. Nous retournons le score de qualité, calculé en utilisant des mesures standard telles que la précision, le rappel ou le score F1 des prédictions, comme valeur de la métrique. Après avoir inspecté les vérifications et collecté les métriques, le système déclenche le calcul efficace des métriques (voir la section 4 pour plus de détails sur la façon dont nous exécutons physiquement ces calculs), invoque le code de validation défini par l'utilisateur à partir des contraintes, et évalue les résultats.

Résultat. Après l'exécution de la vérification de la qualité des données, notre système indique les contraintes qui ont réussi et celles qui ont ont échoué, y compris des informations sur le prédicat appliqué à la métrique dans laquelle la contrainte a été traduite, ainsi que la valeur qui a fait échouer une contrainte.

Le Listing 2 3.4 montre un extrait d'une sortie potentielle pour notre exemple. Nous

```
...
Success("isComplete(title)",
 Completeness("title") == 1.0)),
Success("isNonNegative(count)",
 Compliance("count >= 0") == 1.0)),
Failure("isUnique(customerId, countryResidence,
  deviceType, title)",
 Uniqueness("customerId", "countryResidence",
 "deviceType", "title") == 1.0, 0.9967),
Failure("isInRange(priority, ('hi', 'lo'))",
 Compliance("priority IN ('hi', 'lo')") == 1.0,
  0.833),
...
```

FIGURE 3.4 – Listing 2 : Exemple de résultat de la vérification de la qualité des données montrant les métriques, les prédicats appliqués et les résultats.

voyons que notre contrainte isComplete(title) a été traduite en un prédicat Completeness(title) == 1.0 qui a tenu sur les données. De manière analogue, notre contrainte isNonNegative(count) se traduit par le prédicat Conformité("count >= 0") == 1.0 et correspondait également à tous les enregistrements. Au contraire, notre contrainte unique a échoué, puisque seulement 99,67% des enregistrements ont été identifiés comme uniques, et le prédicat que le système a généré à partir de la contrainte isInRange ne correspondait qu'à 83,3 % des enregistrements.

3.2 Calcul incrémental de métriques pour des ensembles de données croissants

Dans les déploiements du monde réel, les données sont rarement statiques. Nous rencontrons généralement des systèmes qui produisent continuellement des données (par exemple, en interagissant avec les utilisateurs). Par conséquent, il est donc de la plus

haute importance que les systèmes de validation des données comme le nôtre supportent des scénarios où nous ingérons continuellement de nouveaux lots d'enregistrements pour un ensemble de données particulier. Dans de tels cas, nous avons besoin d'avoir accès à des métriques mises à jour pour l'ensemble des données ainsi que pour les nouveaux enregistrements et nous devons pouvoir mettre à jour ces métriques de manière incrémentielle sans avoir à accéder aux données précédentes (voir la figure 1 pour plus de détails). Dans ce qui suit, nous présentons notre machine de calcul de métriques incrémentielles construite pour cette tâche.

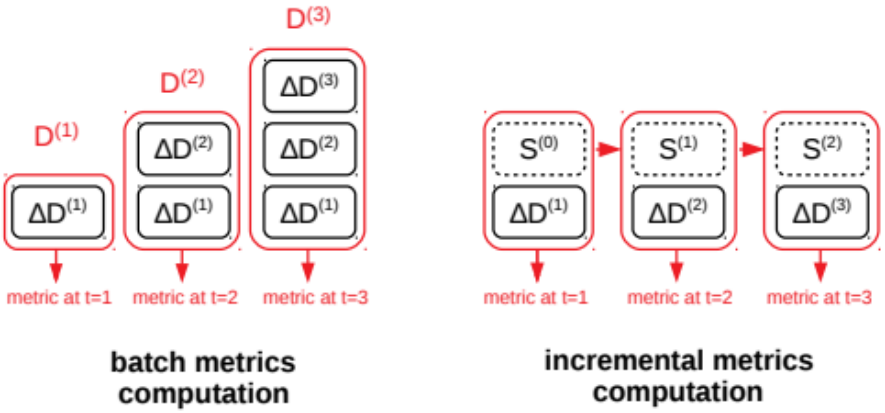

FIGURE 3.5 – Figure 1 : Au lieu d'exécuter de manière répétée le calcul par lot sur des données d'entrée D de plus en plus volumineuses, nous proposons l'exécution d'un calcul incrémental qui n'a besoin de consommer le dernier ensemble de données delta $\triangle D^{(t)}$ et un état S du calcul.

Modèle de calcul. Soit $D^{(t)}$ désigne l'instantané de l'ensemble de données au temps t et que $\triangle D^{(t)}$ désigne les données delta au temps t (les enregistrements supplémentaires) nécessaires pour former $D^{(t+1)}$.

Notez que nous nous limitons aux cas d'appendice seulement où un ensemble de données au moment t est simplement l'union de tous les deltas précédents : $D^{(t)} = \cup_{k=1}^{t} \triangle D^{(k)}$. Au lieu de calculer des métriques pour l'ensemble de données croissant à partir de tous les instantanés, le calcul incrémental introduit un état S, une fonction f pour mettre à jour l'état à partir d'un delta et de l'état précédent, et une fonction g pour calculer la métrique

réelle à partir de l'état S, de sorte que $m^{(t)} = g(S^{(t)})$. En outre, nous avons besoin d'un état initial "vide" $S^{(0)}$. l'avantage du calcul incrémental est qu'il nous permet de calculer la série de métriques pour l'ensemble de données de données via un calcul récursif qui ne consomme que les deltas : $S^{(t)} = f(\triangle D^{(t)}, S^{(t-1)})$.

Reformulation des mesures de qualité. Dans ce qui suit, nous présentons un ensemble de reformulations des métriques existantes afin de permettre un calcul incrémental de celles-ci. Pour chaque métrique, nous montrons comment " diviser " le calcul des métriques pour le nouveau jeu de données $D^{(t+1)}$ en calculant des statistiques suffisantes sur l'ensemble de données précédent D et le delta de l'ensemble de données $\triangle D$ (nous supprimons les indices ici pour améliorer la lisibilité). Une fois qu'une telle reformulation est donnée, nous pouvons effectuer le calcul pour $D^{(t+1)}$ en chargeant les statistiques suffisantes persistantes pour D et en les mettant à jour à partir de valeurs calculées uniquement sur les enregistrements $\triangle D$ nouvellement ajoutés.

Notation : Soit N et $\triangle N$ le nombre d'enregistrements dans les ensembles de données D et $\triangle D$. Soit V et $\triangle V$ toutes les valeurs uniques dans une colonne particulière de l'ensemble de données D ou $\triangle D$. Le site ensemble de valeurs uniques dans le nouvel ensemble de données $V^{(t+1)}$ est simplement l'union $V \cup \triangle V$ des ensembles de valeurs uniques de l'ensemble de données et de l'ensemble de données delta. De plus, laissons c_v et $\triangle c_v$ désignent la cardinalité de la valeur v dans une colonne particulière de l'ensemble de données D ou $\triangle D$.

Le nombre d'enregistrements Taille est la mesure la plus simple à réécrire, car la taille du nouvel ensemble de données $D^{(t+1)}$ est simplement la somme $N + \triangle N$ de la taille N de l'ensemble de données précédent D plus la taille $\triangle N$ de l'ensemble de données delta $\triangle D$. Pour une version incrémentale de Compliance, nous devons conserver deux résultats intermédiaires, à savoir le nombre absolu d'enregistrements $|\{d \in D | p(d)\}|$ qui correspondaient précédemment au prédicat ainsi que la taille N de l'ensemble de données précédent D. Ensuite, nous pouvons calculer la conformité pour le nouvel ensemble de données $D^{(t+1)}$ à partir de ces valeurs retenues et du nombre d'enregistrements $|\{d \in \triangle D | p(d)\}|$ qui correspondaient au prédicat dans le delta ainsi que de la taille $\triangle N$ du delta : $\frac{|\{d \in D | p(d)\}| + |\{d \in \triangle D | p(d)\}|}{N + \triangle N}$

Nous pouvons reformuler la complétude comme une conformité avec un prédicat "n'est

pas nul". Le calcul incrémental de l'unicité exige que nous connaissions les cardinalités c_v de la valeur v dans l'ensemble de données précédent D ainsi que l'ensemble de valeurs distinctes distinctes V . Nous devons inspecter la somme des cardinalités $c_v + \triangle c_v$ pour chaque valeur v dans l'ensemble de données précédent et le delta : $\frac{|\{v \in V \cup \triangle V | c_v + \triangle c_v = 1\}|}{|V \cup \triangle V|}$

Nous calculons également la Distinction incrémentale de la même manière en comparant le nombre de valeurs distinctes dans les données $|V \cup \triangle V|$ à la taille des données $N + \triangle N$: $\frac{|V \cup \triangle V|}{N + \triangle N}$

Le calcul incrémentiel de l'entropie nous oblige à estimer la probabilité $p(v)$ qu'une valeur particulière v se produise dans la colonne, à partir de la cardinalité c_v de la valeur dans les données précédentes, de sa cardinalité $\triangle c_v$ dans le delta et des tailles N et $\triangle N$ de l'ensemble de données précédent et du delta : $-\sum_v \frac{c_v + \triangle c_v}{N + \triangle N} log \frac{c_v + \triangle c_v}{N + \triangle N}$

le calcul incrémental de l'Information Mutuelle nous oblige à maintenir des histogrammes sur les cardinalités c_{v1} de la première colonne, c_{v2} de la deuxième colonne, ainsi que les comptes de cooccurrence c_{v1v2} pour toutes les occurrences par paire, et es fusionner avec les comptes correspondants $\triangle c_{v1v2}$ pour l'ensemble de données delta :

$$\sum_{v1} \sum_{v2} \frac{c_{v1v2} + \triangle c_{v1v2}}{N + \triangle N} log \frac{c_{v1v2} + \triangle c_{v1v2}}{(c_{v1} + \triangle c_{v1})(c_{v2} + \triangle c_{v2})}$$

Pour calculer notre métrique de prédictibilité, nous évaluons la qualité de prédiction d'un modèle naïf bayes multinomial (formé sur les caractéristiques extraites des colonnes d'entrée spécifiées par l'utilisateur) pour la colonne cible. Les paramètres sont généralement estimés en utilisant une version lissée de l'estimation du maximum de vraisemblance :

$$argmax_k \sum_i fi \; log \; \frac{N_{ki} + \triangle N_{ki} + ai}{N_k + \triangle N_k + a}$$

Les structures de données que nous utilisons pour les mesures ApproxQuantile et ApproxCountDistinct supportent naturellement les calculs incrémentaux et ne requièrent donc aucun soin particulier de notre part.

3.3 Contrainte Suggestion

Les avantages de notre système pour les utilisateurs dépendent fortement de la richesse et de la spécificité des contrôles et des contraintes que les utilisateurs définissent et pour lesquelles notre système calculera régulièrement des mesures de qualité des données. Par conséquent, il est très important pour un système comme le nôtre de rendre le processus d'adoption aussi simple que possible. C'est pourquoi nous fournissons des mécanismes permettant de suggérer automatiquement des contraintes et d'identifier des types de données pour les ensembles de données (même si aucun schéma n'est disponible). Cette fonctionnalité de suggestion peut ensuite être intégrée dans les pipelines d'ingestion et peut également être utilisée pendant l'analyse exploratoire des données. Le point de départ de notre mécanisme de suggestion de contraintes est un ensemble de données dont les colonnes individuelles sont connues et ont des noms, mais aucune autre information de schéma telle que les types de données ou les contraintes n'est disponible. Un exemple classique pour un tel ensemble de données serait un fichier CSV vivant dans un système distribué. Notre système aide l'utilisateur à identifier les types de données des colonnes et suggère des contraintes potentielles aux utilisateurs, qu'ils peuvent utiliser comme base pour concevoir des contrôles déclaratifs pour l'ensemble de données à disposition.

Heuristique sur les statistiques sommaires. Notre fonctionnalité de suggestion de contraintes est construite sur une approche basée sur une heuristique utilisant le profilage à une seule colonne [1]. Bien qu'un profilage plus complexe des données plus complexe serait certainement utile, nous devons être en mesure de consommer des tables de la taille d'un téraoctet avec plusieurs milliards de lignes, et nous devons donc nous limiter à de simples statistiques. Comme nous l'avons déjà expliqué, l'utilisateur fournit un jeu de données seule sans information sur le type et le schéma, à l'exception du nom des colonnes. En outre, l'utilisateur peut éventuellement spécifier un ensemble de colonnes à inspecter (et une taille d'échantillon à utiliser pendant la phase de suggestion) pour accélérer le processus. Notre système exécute ensuite le profilage de colonne unique en trois passes sur les données. Dans la première passe, nous calculons la taille des données, exécutons la détection du type de données sur chaque colonne, et nous calculons la complétude ainsi que le nombre approximatif de valeurs distinctes via des croquis hyperlogarithmiques [21, 18] pour chaque colonne d'intérêt. Les tâches de profilage de la deuxième passe opèrent sur les colonnes que nous avons identifiées comme ayant des types numériques. Pour chacune de ces colonnes, nous calculons des statistiques sommaires telles que le minimum, le maxi-

mum, la moyenne, l'écart standard et les quartiles approximatifs [15]. Dans un troisième temps, nous calculons la distribution de fréquence des valeurs pour les colonnes dont la cardinalité est inférieure à un seuil spécifié par l'utilisateur (afin de limiter la mémoire requise). Ensuite, notre système recommande des contraintes pour le jeu de données en question, sur la base d'heuristique qui exploite les résultats du profilage. Dans ce qui suit, nous énumérons une sélection des règles heuristiques que nous appliquons :

- Si une colonne est complète dans l'échantillon en question, nous proposons une contrainte isComplete (non nulle).
- Si une colonne est incomplète dans l'échantillon en question, nous suggérons une contrainte hasCompleteness. Nous modélisons le fait qu'une valeur soit présente ou non comme une variable aléatoire distribuée par Bernoullid, estimons un intervalle de confiance pour la probabilité correspondante, et renvoyons la valeur de départ de l'intervalle comme limite inférieure de la complétude des données.
- Si le type détecté de la colonne est différent de 'string', nous suggérons une contrainte hasConsistentType pour le type détecté.
- Pour la découverte de clés, nous étudions une approximation de "ratio de rangs uniques" [12] : si le ratio de la taille de l'ensemble de données par rapport au nombre approximatif de valeurs distinctes dans cette colonne se situe dans la limite d'erreur d'approximation de l'esquisse hyperlogue utilisé, nous suggérons une contrainte isUnique.
- Si une colonne est numérique et que ses valeurs observées se situent dans un certain intervalle, nous suggérons une contrainte de conformité. Ccertaine plage, nous suggérons une contrainte Compliance avec un prédicat qui ne correspond qu'aux valeurs comprises dans cette plage (par exemple, une plage de valeurs positives uniquement si le minimum observé est 0).
- Si le nombre de valeurs distinctes dans une colonne est inférieur à un seuil particulier, nous interprétons la colonne comme étant catégorique et suggérons une contrainte isInRange qui vérifie si les valeurs futures sont contenues dans l'ensemble des valeurs déjà valeurs déjà observées.

Noter que nous considérons la suggestion de contraintes comme un processus "humain dans la boucle" avec un faible budget de calcul et donc l'utilisateur final pour sélection-

ner et valider nos suggestions qui ne seront pas nécessairement valables pour les données futures (ou même pour l'échantillon en question dans le cas de contraintes uniques).

Apprendre la sémantique des noms de colonnes et de tables. Nous remarquons que les noms réels des colonnes contiennent souvent une sémantique inhérente qui permet aux humains de déduire intuitivement les types de colonnes et les contraintes. Des exemples de tels noms sont 'id', qui est couramment utilisé pour une colonne clé primaire artificielle de type string ou int, 'is deleted', qui se réfère probablement à une colonne booléen, ou "prix par unité", qui indique une colonne numérique. Nous formons donc un modèle d'apprentissage automatique pour prédire les contraintes solennelles en se basant sur le nom de la table et de la colonne, ainsi que de son type. Les données d'apprentissage pour ce modèle sont extraites des schémas de tables de projets open source. Notre système intègre ce modèle en tirant parti de ses prédictions pour améliorer (et potentiellement corriger) les suggestions faites par notre heuristique. Dans le cas où nos règles heuristiques suggèrent une contrainte isUnique, nous consultons la prédiction probabiliste du classificateur pour décider s'il faut suivre la suggestion ou non.

3.4 Détection d'anomalies

La détection d'anomalie dans notre système fonctionne sur des séries temporelles historiques de mesures de qualité des données (par exemple, le ratio des valeurs manquantes pour différentes versions d'un ensemble de données). Nous ne posons aucune restriction sur l'algorithme de détection d'anomalie à appliquer. Le système est livré avec une poignée d'algorithmes standard. Voici quelques exemples, un algorithme qui vérifie simplement les seuils définis par l'utilisateur. L'algorithme de notre exemple appelé OnlineNormal calcule une moyenne courante et une estimation de la variance et compare les valeurs de la série à une limite définie par l'utilisateur sur le nombre d'écarts types qu'elles peuvent différer de la moyenne. Une méthode supplémentaire permet aux utilisateurs de spécifier le degré de différenciation appliqué avant d'exécuter la détection d'anomalie. Cela donne aux utilisateurs la possibilité d'appliquer une technique simple pour stationnariser la série temporelle à analyser [22]. Les mesures de qualité des données, telles que le nombre de valeurs manquantes dans un ensemble de données produites en continu peuvent être à la saisonnalité ou à des tendances (par exemple, la perte ne se produit qu'à certains moments lorsque le système est soumis à une forte charge). Dans ces cas, il n'est pas toujours pos-

sible d'affirmer un comportement correct avec des seuils fournis par l'utilisateur. À cette fin, nous permettons aux utilisateurs d'intégrer leurs propres algorithmes de détection d'anomalies et de prédiction de séries temporelles.

Chapitre 4

MISE EN ŒUVRE

Nous implémentons notre bibliothèque de validation de données au-dessus du moteur de flux de données distribué Apache Spark [50], en utilisant l'infrastructure AWS pour le stockage. Il est à noter que notre bibliothèque ne dépend pas d'une fonctionnalité exclusive à Spark, et pourrait être facilement extensible pour exploiter différents moteurs d'exécution, à condition qu'ils prennent en charge les requêtes SQL, les fonctions d'agrégation définies par l'utilisateur et les modèles d'apprentissage automatique simples [2]. Nous avons opté pour Spark parce qu'un environnement Scala/JVM permet aux utilisateurs d'écrire très facilement du code de vérification personnalisé et d'interagir avec des bibliothèques et des systèmes externes. La figure 2 4.1 donne un aperçu de l'architecture appliquée. Notre système fonctionne sur des DataFrames, une abstraction relationnelle pour une table partitionnée (et souvent dénormalisée). L'API orientée vers l'utilisateur consiste en des contrôles et des contraintes, qui permettent aux utilisateurs de définir de manière déclarative sur quelles statistiques des données, leur code de vérification doit être exécuté. Lors de l'exécution des contrôles, notre bibliothèque inspecte les contraintes contenues dans les données, et identifie les métriques nécessaires qui doivent être calculées sur les données afin d'exécuter le code de vérification des contraintes défini par l'utilisateur.

2. Un système avec le support des vues matérialisées nous permettrait même de simplifier notre machinerie de calcul incrémental

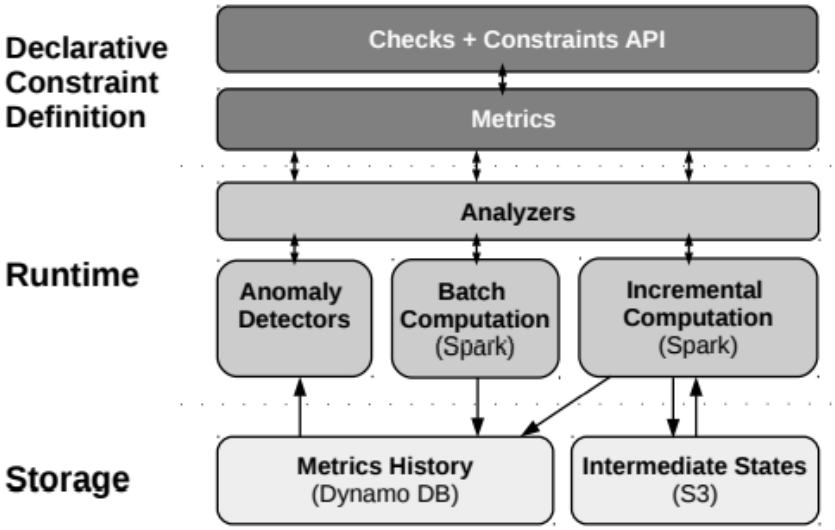

Figure 2: System architecture: users declaratively define checks and constraints to combine with their verification code. The system identifies the required metrics and efficiently computes them in the runtime layer. The history of metrics and intermediate states of incremental computations are maintained in AWS storage services.

FIGURE 4.1 – Figure 2 : Architecture du système : les utilisateurs définissent de manière déclarative déclarative des vérifications et des contraintes à combiner avec leur code de vérification. Le système identifie les métriques nécessaires et les calcule efficacement dans la couche d'exécution. L'historique des métriques et les états intermédiaires des calculs incrémentiels sont conservés dans les services de stockage AWS.

Pour chaque métrique, notre bibliothèque choisit un Analyseur (qui peut être vu comme un opérateur physique) capable de calculer la métrique particulière sur les données. Les Analyseurs sélectionnés sont donnés à un AnalysisRunner dans notre couche d'exécution qui programme l'exécution du calcul de la métrique. Ce runner applique un ensemble de simples d'optimisations simples pour le calcul de plusieurs métriques. Pour toutes les métriques qui ne nécessitent pas de repartitionner les données, le programme d'exécution collecte les fonctions d'agrégation requises et les exécute dans une seule requête SparkSQL générée sur les données afin de bénéficier du partage de ~~balayage~~ document. Dans l'exemple de la section 3.1, ces mesures seraient la taille de l'ensemble de données, l'~~exhaustivité de six colonnes, comme le montre le tableau ci-dessous~~. Complétude de six colonnes, ainsi que la Conformité pour les trois contraintes de satisfaction. Toutes ces métriques seront cal-

culées simultanément en un seul passage sur les données. Les métriques résultantes sont finalement stockées dans une base de données de documents (DynamoDB) pour une récupération ultérieure (et une utilisation par des algorithmes de détection d'anomalies). Le runtime pour les calculs incrémentaux stocke les états des analyseurs incrémentaux dans un système de fichiers distribué (S3).

Pour l'estimation de la prévisibilité, nous devons former un modèle d'apprentissage automatique sur les colonnes d'entrée spécifiées par l'utilisateur et évaluer la capacité du modèle à prédire les valeurs de la colonne cible. Nous avons développé une architecture pluggable, dans laquelle nous featuralisons les colonnes d'entrée en concaténant leurs représentations de chaîne, et les tokeniser et les hacher via les transformateurs Tokenizer et HashingTF de Spark. Ensuite, n'importe quel algorithme de classification de SparkML [27] peut être utilisé pour apprendre un modèle de prédiction. Bayes [36] de Sparks, car elle offre une limite inférieure évolutive sur la précision de prédiction, ne nécessite pas d'optimisation des hyperparamètres et est simple à former de manière incrémentielle. Nous appliquons le modèle de classification formé pour prédire des valeurs sur une fraction retenue des données et nous rendons compte de la qualité de la prédiction (par exemple, mesurée en utilisant la précision) en tant que valeur de prédictibilité.

4.1 Calcul incrémental

Dans ce qui suit, nous détaillons comment rendre les analyseurs de notre système " conscients de l'état " pour leur permettre d'effectuer des calculs incrémentaux. Une classe de base correspondante en Scala est présentée dans le Listing 3, où M désigne le type de métrique à calculer et S désigne le type d'état requis. Persistance et la récupération de l'état sont gérés en dehors de l'implémentation par un StateProvider. La méthode initialState produit un état initial vide, apply produit un état et la métrique correspondante pour un ensemble de données initial, et update consomme l'état actuel et un ensemble de données delta et produit l'état mis à jour, ainsi que les métriques correspondantes, à la fois pour le jeu de données dans son ensemble et pour le delta, sous la forme d'un tuple (S, M, M). En outre, la méthode applyOrUpdateFromPersistedState exécute le calcul incrémental et se charge de gérer les états concernés à l'aide de StateProviders.

```scala
trait IncrementalAnalyzer[M, S]
  extends Analyzer[M] {

 def initialState(initialData: DataFrame): S

 def update(
   state: S,
   delta: DataFrame): (S, M, M)

 def updateFromPersistedState(
   stateProvider: Option[StateProvider],
   nextStateProvider: StateProvider,
   delta: DataFrame): (M, M)
}

trait StateProvider {

 def persistState[S](
   state: S,
   analyzer: IncrementalAnalyzer[M, S])

 def loadState[S](
   analyzer: IncrementalAnalyzer[M, S]): S
}
```

FIGURE 4.2 – Listing 3 : Interface pour les analyseurs incrémentaux.

Compte tenu de ces éléments, nous appelons applyOrUpdateFromPersistedState qui calculera la métrique et fera persister l'état.

Gestion de l'état. Afin d'exécuter le calcul incrémental, un utilisateur doit configurer des fournisseurs d'état pour permettre la gestion de l'état. En général, l'état d'un instantané particulier de l'ensemble de données réside dans un répertoire sur S3. Nous fournissons la mise en œuvre du fournisseur correspondant. ~~calculera la métrique et fera persister l'état~~. Pour calculer la métrique actualisée pour le prochain instantané de l'ensemble de données, nous avons besoin de deux StateProviders, l'un qui fournit l'état de l'ancien instantané, et un autre qui recevra l'état mis à jour calculé à partir de l'ancien état et du delta. Notez que cette API interne est généralement cachée aux utilisateurs, à qui il est conseillé de programmer notre système en utilisant l'API de vérification déclarative de la section 3.1. Dans ce qui suit, nous discutons des détails d'implémentation supplémentaires. Lorsque l'on calcule de manière incrémentielle des mesures qui nécessitent un re-partitionnement des données (par exemple, l'entropie et l'unicité qui exigent que ~~nous l'entropie et l'unicité qui~~ nous obligent à regrouper les données par colonne respective), nous mettons en œuvre le schéma incrémentiel comme suit. L'état S est composé d'un histogramme sur

les données (le résultat de delta.select(columns).groupBy(columns).count()). La fonction de mise à jour fusionne l'histogramme précédent pour les données courantes avec l'histogramme du delta via une jointure externe sur les colonnes de regroupement et calcule les comptes correspondants pour le delta et l'ensemble des données. L'analyseur calcule ensuite la métrique de l'état par une agrégation sur l'histogramme. Nous présentons un exemple de mise à jour incrémentielle de l'entropie de la colonne dans la figure 3.

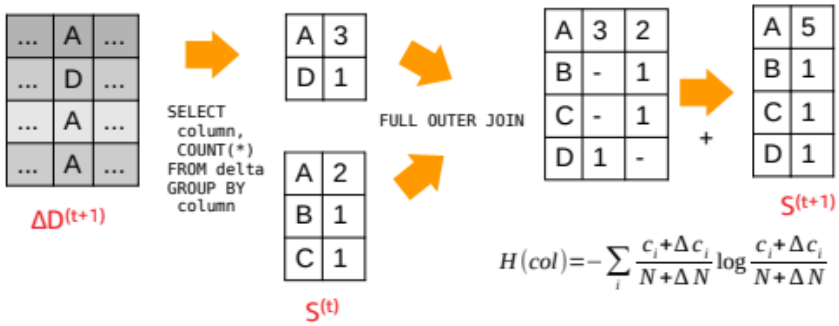

FIGURE 4.3 – figure 3 : Exemple de mise à jour incrémentale de l'entropie d'une colonne : les fréquences des valeurs dans les enregistrements delta $\triangle D^{(t+1)}$ sont calculées via une requête de regroupement et fusionnées avec l'état précédent $S^{(t)}$ via une jointure externe complète. Après avoir additionné les comptes, on a l'état mis à jour $S^{(t+1)}$, à partir duquel l'entropie de la colonne dans l'ensemble de données actualisé $D^{(t+1)}$ peut être calculée.

Optimisations. Pendant le calcul de plusieurs métriques, nous appliquons un ensemble d'optimisations de requêtes appliquées manuellement : (a) nous mettons en cache le résultat de l'opération de comptage sur des dataframes, car de nombreuses métriques requièrent la taille du delta, par exemple ; (b) nous appliquons le partage du scan pour les agrégations : nous exécutons toutes les agrégations qui reposent sur le même regroupement (ou aucun regroupement) des données dans le même passage sur les données.

4.2 Suggérer efficacement des contraintes

Le principal objectif de conception de notre composant de suggestion de contraintes est de maintenir le calcul de statistiques afin qu'il puisse être exécuté au cours d'un pi-

peline d'ingestion pour de grands ensembles de données. Il est donc crucial de garder le nombre de passages sur les données est indépendant du nombre de colonnes dans le cadre de données en question. Nous supposons que notre entrée est un cadre de données avec des colonnes nommées de types inconnus (initialement de type 'string' pendant l'ingestion, par exemple lors de la lecture de fichiers CSV). Pour le calcul des statistiques récapitulatives requises par notre heuristique de la section 3.3, nous n'utilisons que des agrégations qui ne nécessitent pas de repartitionnement de la table, et nous n'effectuons que deux passages sur les données, où les agrégations partagent les balayages. De plus, nous avons une estimation du nombre de valeurs distinctes par colonnes intéressantes après le premier passage, ce qui nous permet de contrôler la mémoire pour les croquis et les histogrammes (par exemple, en calculant seulement la distribution complète des valeurs pour les colonnes avec faible cardinalité) utilisée dans la deuxième passe. Comme indiqué dans la section 3.3, nous utilisons un modèle d'apprentissage automatique pour décider de la suggestion de contrainte unique. Les entrées de ce modèle sont le nom de la table, ainsi que le nom et le type de colonne. Comme données d'entraînement pour ce modèle, nous extrayons un ensemble de données de 2 453 tuples (nom de la table, nom de la colonne, type, est unique) de tuples provenant des schémas de base de données de plusieurs projets open source tels que mediawiki, wordpress et oscommerce. Sur ces données de schéma, nous entraînons un modèle de régression logistique en utilisant des n-grammes de caractères hachés des noms et un codage à un coup du type comme caractéristiques.

Nous utilisons le SGDClassifier combiné avec le HashingVectorizer de scikit-learn [32], et ajustons les hyperparamètres du modèle (dimensionnalité du vecteur de caractéristiques, régularisation, taille des n-grams) en utilisant une validation croisée à cinq reprises. Nous obtenons un score AUC de 0,859 pour la courbe ROC, en utilisant une fonction de perte logistique avec une régularisation L1 et un facteur de régularisation de 0,001 sur des ngrammes d'une taille maximale de 5 à partir des données d'entrée hachées en vecteurs de caractéristiques à 108 dimensions. Nous tirons parti de la prédiction probabiliste de ce modèle (qui nous donne un indice indiquant si le nom de la colonne indique une contrainte unique) comme un score pour notre suggestion de contrainte unique basée sur des règles et suggestion à l'utilisateur que si le modèle lui attribue une probabilité supérieure à 50 %.

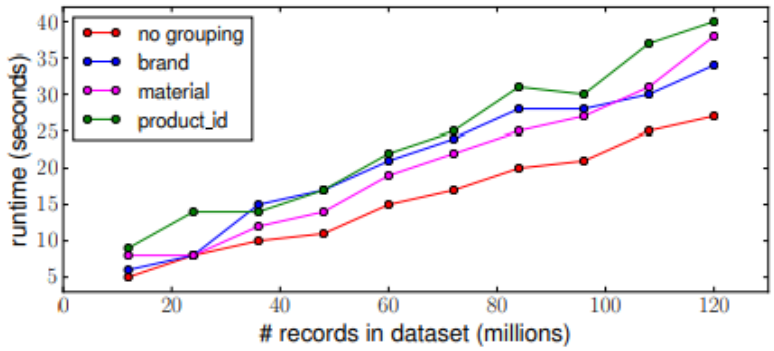

FIGURE 4.4 – Figure 4 : Augmentation linéaire du temps d'exécution pour différents calculs de métriques par lot sur un ensemble de données croissant avec jusqu'à 120 millions d'enregistrements.

Chapitre 5

ÉVALUATION EXPÉRIMENTALE

Dans ce qui suit, nous réalisons un ensemble d'expériences de scalabilité pour notre calcul de métriques par lots, nous appliquons notre estimation de la prévisibilité à un ensemble de données de produits (Section 5.1) et nous étudions les avantages de l'application du calcul incrémentiel à des ensembles de données croissants (Section 5.2). Enfin, nous évaluons notre fonctionnalité de suggestion de contraintes sur deux ensembles de données externes (Section 5.3) et présentons un cas d'utilisation simple de détection d'anomalies dans la Section 5.4.

Pour nos expériences basées sur Spark, nous exploitons un jeu de données de données représentant un échantillon d'un catalogue de produits interne, qui d'environ 120 millions d'enregistrements, où chaque chaque enregistrement décrit un produit avec plusieurs centaines d'attributs ; La taille des données est d'environ 50 Go en format parquet. Nous imitons un cas d'utilisation avec un ensemble de données en annexe seulement, et nous partitionnons aléatoirement nos données de produits en 10 "deltas" d'environ 12 millions d'enregistrements pour cela. De plus, nous utilisons deux jeux de données externes pour l'évaluation. Le premier ensemble de données [3] est composé de commentaires de mai 2015 dans plusieurs forums de discussion sur le site web d'agrégation de nouvelles sociales reddit.com. Le second jeu de données contient des informations sur 5180 utilisateurs de twitter, que nous avons extraites du flux d'échantillons Twitter disponible publiquement.

Les expériences basées sur Spark exploitent un cluster sur Elastic MapReduce avec 5 travailleurs (c3.4xlarge instances) exécutant Apache Spark 2.0.2 et HDFS 2.7.3.

3. https ://www.kaggle.com/reddit/reddit-comments-may-2015

5.1 Calculs par lots

Dans cette première série d'expériences, nous évaluons l'efficacité du calcul de notre métrique s'adapte à de grands ensembles de données et nous montrons l'efficacité de notre estimation de la prévisibilité basée sur l'apprentissage automatique.

Mise à l'échelle des calculs de métriques. Afin d'évaluer l'évolutivité du calcul des métriques par lots de notre système, nous calculons un ensemble de métriques de qualité sur l'ensemble de données de produits en croissance, que nous lisons à partir de S3. La figure 4 4.4 montre les résultats, où chaque point représente le temps d'exécution sur une version particulière de l'ensemble de données en croissance. Le graphique intitulé " pas de regroupement " fait référence aux résultats du calcul d'un ensemble de six métriques (taille des données et exhaustivité de cinq colonnes) qui ne nécessitent pas de repartitionner les données. Par conséquent, ces mesures peuvent être calculées par des agrégations en un seul passage sur les données. Les lignes restantes font référence au calcul de métriques telles que l'entropie et l'unicité sur les colonnes marque, matériau et identifiant de produit, qui nécessitent de répartir les données (par exemple, en les groupant par la colonne pour laquelle nous pour laquelle nous voulons calculer ces métriques). En raison du re-partitionnement inhérent, ces métriques sont généralement plus coûteuses à calculer et leur coût est lié à la cardinalité de la colonne respective. Néanmoins, les quatre charges de travail évaluées présentent un temps d'exécution qui croît de façon linéaire avec la taille de l'ensemble de données, ce qui est attendu puisque notre système génère en interne des requêtes d'agrégation simples avec des fonctions d'agrégation personnalisées à exécuter par SparkSQL [3].

Estimation de la prévisibilité avec des bayes naïves. Nous démontrons notre fonctionnalité d'estimation de la prédictibilité sur un ensemble de 845 000 articles de mode de 10 marques populaires que nous avons extraites d'un ensemble de données de produits plus large. Nous avons fixé comme tâche de prédire la valeur de la colonne de la marque à partir d'autres d'autres colonnes, telles que le nom, la description, la taille, le fabricant et leurs combinaisons. Nous exécutons les expériences correspondantes avec Spark sur une seule instance c4.8xlarge. Nous prenons différents échantillons de l'ensemble de données (100k enregistrements, 200k enregistrements, 400k enregistrements, 600k enregistrements, ensemble de données complet). Sur chacun d'entre eux, nous entraînons un modèle naïf de Bayes avec des colonnes d'entrée hachées comme caractéristiques pour prédire la colonne de marque sur 80 % de l'échantillon. Enfin, nous calculons le score pondéré F1 pondéré

pour les prédictions sur les 20% restants. Nous répétons ceci pour différentes colonnes d'entrée et des échantillons des données. Nous constatons que la colonne nom seule est déjà un très bon prédicteur de la marque, puisqu'il donne un score F1 pondéré pondéré de plus de 97% sur tous les échantillons de l'ensemble de données. Les meilleurs résultats sont obtenus en combinant la colonne nom avec les colonnes bulletpoints, description et fabricant, où nous atteignons des scores F1 de 97,3%, 98.8%, 99.4%, 99.5% pour les échantillons de 100k, 200k, 400k, 600k, et de 99.5% pour les échantillons de 99.5%. des données, et de 99,5% pour l'ensemble complet de données. Sur la base de ces résultats (qui peuvent être calculés à l'aide d'un AnalysisRunner de la section 4), un utilisateur peut configurer une contrainte pour les données futures afin d'être notifié lorsque la prévisibilité tombe en dessous des valeurs observées, par exemple :

```
Check(Level.Warning)
 .hasPredictability("brand", ("name",
   "bulletpoints", "description",
   "manufacturer"), f1=0.97)
```

Nous enregistrons en outre le temps d'exécution de l'apprentissage du modèle pour différentes featurisations. Nous constatons que le temps d'exécution linéairement pour des données croissantes et dépend principalement de la longueur des chaînes de caractères dans les colonnes d'entrée (par exemple, l'entraînement d'un modèle sur la colonne description seule avec beaucoup de texte prend plus de temps que l'entraînement sur la colonne des noms combinée à la colonne des points de suspension). Ceci est attendu car les bayes naïfs effectue un seul passage à travers les données et additionne les vecteurs de caractéristiques par classe, qui résultent de la tokenisation et du hachage des colonnes d'entrée. Notez que l'entraînement est très efficace ; elle prend moins de dix secondes dans tous les cas, même sur l'ensemble complet de données.

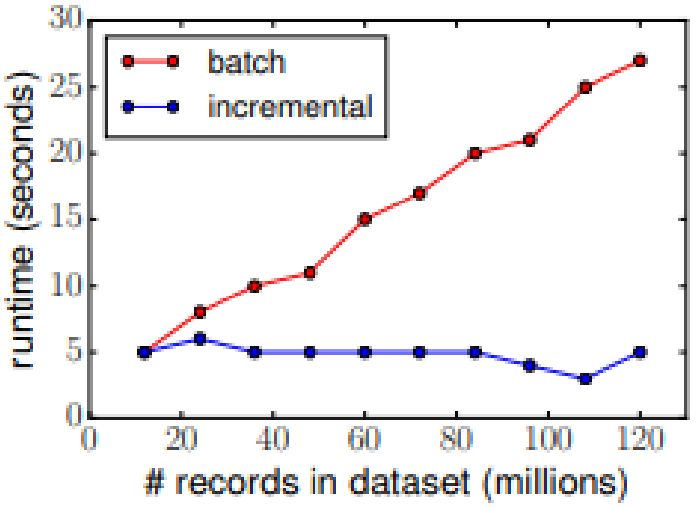

FIGURE 5.1 – Figure 5 : Temps d'exécution pour la taille et l'exhaustivité sur l'id du produit, le matériau, couleur, marque.

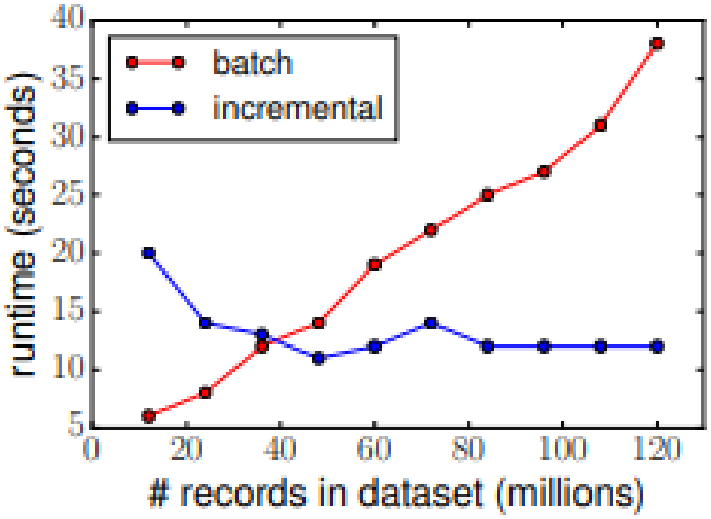

FIGURE 5.2 – Figure 6 : Temps d'exécution pour l'unicité et l'entropie sur le matériau.

5.2 Avantages du calcul incrémental

Nous revisitons notre expérience de scalabilité sur des données croissantes pour valider nos hypothèses sur les avantages du calcul incrémental Nous comparons l'analyse par lot,

qui doit toujours consommer l'ensemble de données comme un tout (l'union de tous les deltas observés jusqu'à présent) contre notre approche incrémentale qui maintient un état et opère toujours sur cet état et sur le delta delta actuel uniquement.

La figure 5 5.1 montre les résultats pour calculer à nouveau les métriques qui ne nécessitent pas de repartitionner les données (nous avons qualifié cette expérience de " non-groupement "). Ces métriques peuvent être calculées en un seul passage sur les données, et l'état réel pour le calcul incrémental est minuscule ici, puisque un ou deux nombres par métrique doivent être maintenus. Alors que le temps d'exécution de l'analyse par lot croît de façon linéaire avec la taille de l'ensemble de données, le temps d'exécution reste constant dans le cas incrémentiel, car il ne dépend que de la taille du delta (qui est constant dans notre configuration). Ensuite, nous revisitons le calcul de métriques telles que l'entropie et l'unicité qui nécessitent de repartir les données. Ces métriques sont généralement plus coûteuses à calculer et le calcul incrémental est également plus difficile, car nous devons maintenir en interne d'un histogramme des fréquences par valeur dans la colonne (le résultat de l'opération de regroupement). nous calculons d'abord ces métriques sur les colonnes matériau et marque qui ont une cardinalité assez faible. Les résultats sont présentés dans la Figure 6 **??**et la figure 7 5.3. Nous voyons que l'approche incrémentale a une surcharge substantielle dans ce cas (persistance et jointure de l'histogramme l'histogramme maintenu), cependant son temps d'exécution reste à peu près constant et il surpasse l'analyse par lot après trois ou quatre deltas. La figure 8 **??**montre les temps d'exécution résultants pour le calcul de l'entropie et de l'unicité pour la colonne product id. Cette colonne est particulière dans cet ensemble de données car elle est composée uniquement de valeurs uniques. En raison de cette caractéristique, le temps d'exécution de l'approche incrémentale montre le même comportement de croissance que celui de l'analyse par lots (croissance linéaire avec la taille des taille des données), car chaque delta introduit un ensemble de nouvelles valeurs, et l'histogramme que le calcul incrémental maintient n'est fondamentalement qu'une copie de la colonne originale. La surcharge de maintenir cet histogramme est aussi ce qui rend le calcul incrémental toujours moins performant. Bien que ce soit un inconvénient, l'approche incrémentale a toujours l'avantage de ne pas nécessiter l'accès à l'ensemble des données pendant le calcul, ce qui simplifie grandement les pipelines d'ingestion.

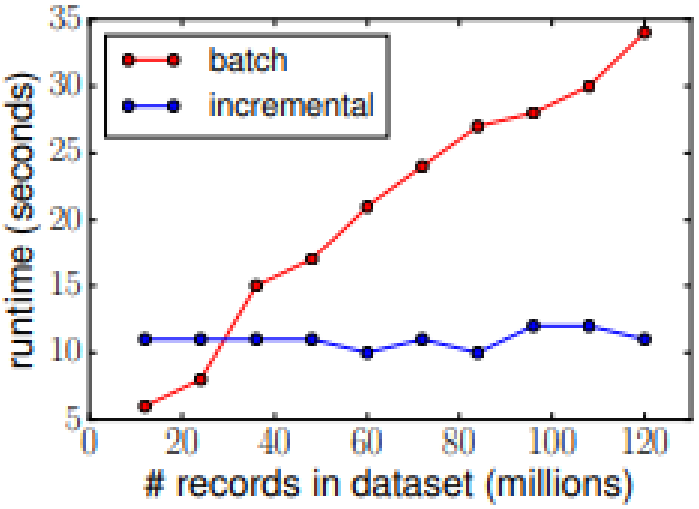

FIGURE 5.3 – Figure 7 : Runtimes pour l'unicité et l'entropie sur la marque.

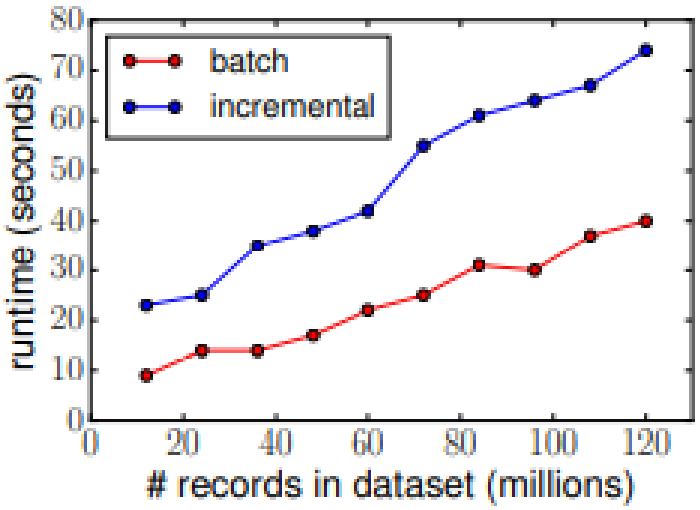

FIGURE 5.4 – Figure 8 : Temps d'exécution pour l'unicité et l'entropie sur l'identifiant du produit.

5.3 Contrainte Suggestion

Nous évaluons notre composant de suggestion de contraintes sur un échantillon de 50 000 enregistrements provenant de l'ensemble de données reddit ainsi que sur l'ensemble de données des utilisateurs de Twitter. Dans chaque cas, nous prenons un échantillon aléatoire de 10% des enregistrements, nous demandons à notre système de suggérer des

contraintes basées sur les données échantillonnées et nous et calculons la couverture de ces contraintes sur les 90% restants des ensembles de données. Les contraintes suggérées ainsi que leur couverture sont présentées dans le tableau 3 5.5. L'ensemble de données reddit est un cas très facile, car toutes les colonnes sont complètes et n'ont que des types string et integer. Cette structure simple est reflétée par le fait que toutes les contraintes suggérées tiennent sur l'ensemble de test. L'expérience sur le jeu de données des utilisateurs de Twitter est plus intéressante, car nous avons des colonnes avec des valeurs manquantes, comme le lieu, et des colonnes avec un petit ensemble de valeurs discrètes, comme la langue. L'exhaustivité de la colonne emplacement dans l'échantillon est de 0,28076 et la contrainte suggérée hasCompleteness >= 0.28 tient sur les données de test, c'est-à-dire que le ratio de valeurs manquantes n'augmente pas. Le système suggère correctement une contrainte isUnique pour les colonnes id et screen name qui sont en fait des clés primaires pour les données. Cependant, le système suggère également deux contraintes qui ne sont pas valables pour les données. La première est la plage de valeurs pour la colonne lang. Ici, nous avons identifié dix valeurs différentes qui représentent seulement plus de 99% des enregistrements dans les données de test, mais qui manquent des langues rares comme le turc ou le hongrois. Une contrainte d'échec sur cette colonne peut néanmoins être utile ; nous avons découvert par une investigation manuelle que les données de cette colonne ne sont pas correctement normalisées, par exemple, il y a différentes capitalisations pour une même langue ; comme 'en-gb' et 'en-GB'. Dans le second cas, le système suggère par erreur une contrainte isUnique pour la colonne de comptage des statuts, en raison du fait qu'il existe de nombreuses valeurs différentes pour cette colonne dans l'échantillon à disposition et nous ne connaissons qu'une approximation du nombre de valeurs La valeur d'unicité de cette colonne n'est que de 64% dans les données de test. La deuxième erreur est cependant corrigée lorsque nous exploitons les prédictions de notre classificateur pour les contraintes uniques : alors que le classificateur attribue une probabilité élevée de 81% que notre contrainte unique suggérée sur la colonne id est valide, il n'attribue qu'une probabilité de 2 % à la colonne de comptage des statuts pour qu'elle soit unique. Malheureusement, le classificateur produit un faux négatif pour la colonne screen name, qui est en effet unique pour notre échantillon à portée de main. Cependant, cela ne serait pas non plus immédiatement évident pour les humains (car différents utilisateurs peuvent avoir le même nom d'écran sur de nombreuses plateformes de réseaux sociaux), et nous préférons les suggestions conservatrices et robustes (par exemple, plutôt des faux négatifs que des faux positifs), qui renforcent la confiance de nos utilisateurs.

dataset	column	suggested constraints	coverage	classifier score
reddit-comments	id	`isComplete`,	1.0	-
		`isUnique`	1.0	0.83
	created_utc	`isComplete`, `hasConsistentType(integral)`, `isNonNegative`	1.0	-
	subreddit	`isComplete`	1.0	-
	author	`isComplete`	1.0	-
	ups	`isComplete`, `hasConsistentType(integral)`	1.0	-
	downs	`isComplete`, `hasConsistentType(integral)`, `isNonNegative`	1.0	-
	score	`isComplete`, `hasConsistentType(integral)`	1.0	-
	edited	`isComplete`, `isNonNegative`	1.0	-
	controversiality	`isComplete`, `hasConsistentType(integral)`, `isNonNegative`,	1.0	-
		`isInRange(0, 1)`	1.0	-
	text	`isComplete`	1.0	-
twitter-users	id	`isComplete`, `hasConsistentType(integral)`, `isNonNegative`,	1.0	-
		`isUnique`	1.0	0.81
	screen_name	`isComplete`,	1.0	-
		`isUnique`	1.0	0.01
	lang	`isComplete`,	1.0	-
		`isInRange('en', 'pt', ...)`	0.991	-
	location	`hasCompleteness >= 0.28`	1.0	-
	followers_count	`isComplete`, `hasConsistentType(integral)`, `isNonNegative`	1.0	-
	statuses_count	`isComplete`, `hasConsistentType(integral)`, `isNonNegative`,	1.0	-
		`isUnique`	0.636	0.02
	verified	`isComplete`, `hasConsistentType(boolean)`	1.0	-
	geo_enabled	`isComplete`, `hasConsistentType(boolean)`	1.0	-

FIGURE 5.5 – Tableau 3 : Suggestion de contraintes et prédiction du type pour le jeu de données des commentaires de reddit et des utilisateurs de twitter. Les contraintes sont suggérées sur la base d'un échantillon de 10% des données, et leur couverture est calculée sur les 90% restants. Nous utilisons un modèle d'apprentissage automatique formé sur les noms de colonnes pour décider des contraintes uniques potentielles.

5.4 Détection d'anomalies

Afin de présenter notre fonctionnalité de détection des anomalies, nous l'appliquons à un cas d'utilisation fictif sur l'ensemble de données Reddit. Supposons que nous voulons exploiter les données de discussion pour une tâche d'extraction d'information telle que la réponse à des questions. Un problème potentiel de qualité des données serait alors que les données pourraient contenir de grandes quantités de spam et de trolling, ce qui aurait une influence négative sur le modèle d'apprentissage automatique que nous souhaitons former sur ces données. Si nous ingérons régulièrement des données de reddit, nous aimerions être alarmés s'il y a des signes d'augmentation de l'activité de spamming ou de trolling. Les données de reddit contiennent un champ de controverse pour chaque message, et la série du ratio de messages de ce type dans une par jour pourrait être un bon signal pour détecter un trolling potentiel. Afin d'exploiter notre fonctionnalité de détection d'anomalies pour cette tâche, nous inspectons les séries temporelles historiques de la métrique Mean(controversiality) par forum de discussion (subreddit) que nous aurions besoin de calculer pendant les ingestions. Dans notre API déclarative, le contrôle correspondant se présente comme suit :

```
Check(Level.Warning, groupBy="subreddit")
 .hasNoAnomalies("controversiality", Mean,
   OnlineNormal(upperDeviationFactor=3))
```

Cela indique que nous voulons être avertis si la controverse moyenne d'un jour donné dans un forum de discussion est supérieure de plus de trois écarts-types par rapport à la moyenne précédente. La figure 9 illustre le résultat de cette analyse pour une sélection de forums de discussion de l'ensemble de données Reddit. Nous constatons que certains forums de discussion présentent une variance relativement faible de la controverse au fil du temps, comme les anime et askreddit. Cependant, il y a aussi des forums avec des pointes de la controverse, comme cringepics et chicagobulls, qui sont marqués par notre approche de détection des anomalies. L'examen manuel de ces pics a révélé qu'ils sont fortement corrélés au nombre d'utilisateurs supprimés le jour donné, ce qui indique qu'ils résultent effectivement d'un comportement de trollage.

Chapitre 6

APPRENTISSAGES

Nous rendons compte des apprentissages à différents niveaux que nous avons obtenus des utilisateurs de notre système de validation des données Au niveau organisationnel, l'utilisation d'une bibliothèque commune de qualité des données. Une telle bibliothèque commune permet d'établir un vocabulaire partagé entre les équipes pour discuter de la qualité des données et à établir les meilleures pratiques sur la façon de mesurer la qualité des données, ce qui conduit à une méthode commune de suivi des métriques des ensembles de données. Il est également très avantageux que les producteurs et es consommateurs d'ensembles de données utilisent le même système pour vérifier la qualité des données, car ils peuvent réutiliser les vérifications et les contraintes de l'un à l'autre, par ex. réutiliser les contrôles et les contraintes de l'un et de l'autre. des consommateurs en aval plus tôt dans le pipeline de traitement des données. Sur le plan technique, les utilisateurs ont souligné le fait que notre bibliothèque de qualité des données s'exécute sur Spark, un logiciel de gestion de la qualité des données ce qui, selon eux, constitue un moyen rapide et évolutif de traiter les données, en partie grâce aux optimisations appliquées par notre plateforme. Notre système a permis de réduire les analyses manuelles et ad-hoc sur leurs données, par exemple, l'échantillonnage et l'observation des résultats pour identifier d'éventuels problèmes tels que les champs incomplets, les valeurs aberrantes et les dérivations par rapport au nombre de lignes prévu. Au lieu de cela, de tels contrôles peuvent désormais être exécutés de manière automatisée dans le cadre de pipelines d'ingestion. En outre, les producteurs de données peuvent exploiter notre système pour interrompre leurs pipelines de publication de données lorsqu'ils rencontrent des anomalies dans les données. Ainsi, ils peuvent s'assurer que le traitement des données en aval, qui inclut souvent l'apprentissage de modèles ML, ne fonctionne qu'avec des données approuvées.

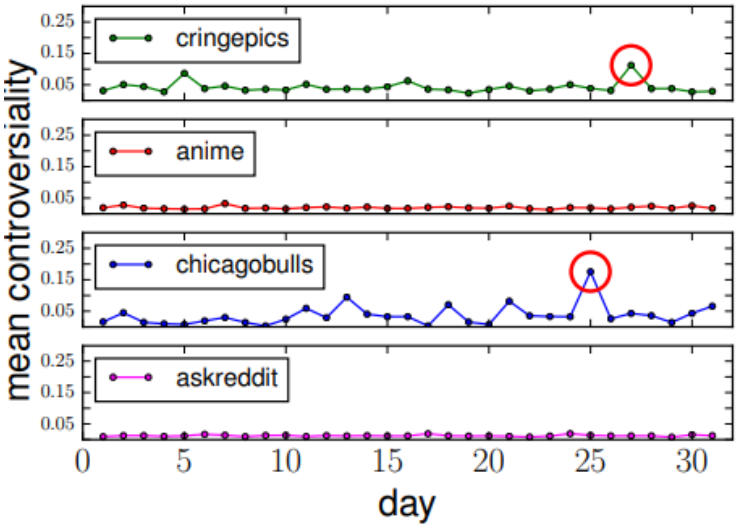

FIGURE 6.1 – Figure 9 : Anomalies détectées dans la série temporelle de la métrique Mean(controversiality) pour différents conseils de l'ensemble de données reddit, qui indiquent un comportement de trollage susceptible de diminuer la qualité des données.

Chapitre 7

TRAVAIL RELATIF

Le nettoyage des données est un domaine de recherche actif depuis des décennies,Voir les études récentes [1, 10, 38] pour une vue d'ensemble. Vérification déclarative de la qualité des données L'idée de permettre des définitions déclaratives des normes de qualité des données est bien établie. Ilyas et al. fournissent un aperçu des définitions de cohérence standard [23]. En fait, tous les SGBD supportent des contraintes d'intégrité standard telles que les contraintes de clé ou les champs annulables. Les contraintes de déni constituent une extension pertinente d'ordre qui permet la couverture d'un plus grand nombre de règles commerciales sur deux tuples [11]. Un paradigme populaire pour définir les dépendances entre les colonnes sont les dépendances fonctionnelles et les dépendances fonctionnelles conditionnelles [6]. Il existe des algorithmes rapides et approximatifs pour les découvrir [31]. Contrairement à cette ligne de travail, notre métrique de prédictibilité s'appuie sur le ML pour apprendre les relations entre les colonnes et utilise des tests empiriques sur des ensembles de données en attente pour la validation. Nous pensons qu'en raison de leur robustesse inhérente aux valeurs aberrantes, les méthodes ML sont plus adaptées à notre cas d'utilisation pour détecter automatiquement les changements de qualité des données sur de nombreux grands ensembles de données. Galhardas et al. proposent un langage déclaratif AJAX pour le nettoyage de données comme une extension de SQL ainsi qu'un modèle pour l'exécution de programmes de nettoyage de données [14]. De même, nous optimisons la validation de nos contraintes déclaratives de qualité des données afin de minimiser l'effort de calcul. Nous combinons un ensemble plus large de contraintes dans un cadre unifié, mais nous ne supportons pas l'exécution automatique des méthodes de réparation des données.

ML pour le nettoyage des données De nombreux chercheurs ont suggéré d'utiliser le ML pour le nettoyage des données. Alors que les méthodes traditionnelles peuvent être utilisées pour générer des candidats pour corriger les données incorrectes (ex, violations des dépendances fonctionnelles), les méthodes d'apprentissage actif peuvent être utilisées pour sélectionner et prioriser l'effort humain [49]. ActiveClean utilise de manière similaire l'apprentissage actif pour la hiérarchisation,mais en même temps, il apprend et met à jour un modèle de perte convexe [24]. convexe [24]. HoloClean génère un modèle probabiliste sur un ensemble de données qui combine des contraintes d'intégrité et des sources de données externes pour générer des suggestions de réparation de données [35]. BoostClean sélectionne automatiquement un ensemble de combinaisons de détection et de réparation d'erreurs en utilisant le boosting statistique [25].

Validation des données dans les applications d'apprentissage automatique. Les défis liés à la construction d'applications complexes d'apprentissage automatique dans le monde réel ont récemment été mis en évidence par de nombreux chercheurs, par exemple, en ce qui concerne la gestion des modèles résultants [26], l'ingénierie logicielle [42, 8], les abstractions de pipeline [2,43, 46], et les enseignements tirés des systèmes du monde réel [34, 7]. A l'apprentissage automatique à grande échelle. Les exemples incluent la gestion des métadonnées des artefacts produits pendant les charges de travail d'apprentissage automatique, y compris les schémas et les statistiques sommaires des ensembles de données utilisés pour la formation et les tests [47, 48, 40, 29, 28, 20, 33, 41], la découverte et l'organisation d' ensembles de données d'entreprise [16], et les contrôles d'intégrité spécifiques à l'apprentissage automatique [45]. Par conséquent, les plateformes modernes d'apprentissage automatique commencent à avoir des composants explicites de validation des données [7, 5, 9].

Chapitre 8

Conclusion

Nous avons présenté un système d'automatisation des tâches de vérification de la qualité des données, qui s'adapte aux grands ensembles de données et répond aux exigences des cas d'utilisation en production. Le système fournit une API déclarative à ses utilisateurs, qui combine des contraintes de qualité communes avec du code de validation personnalisé, et permet des "tests unitaires" pour les données. Nous avons discuté de la manière d'exécuter efficacement la validation des contraintes en traduisant les vérifications en calculs de métriques évolutifs, et nous avons élaboré des reformulations des métriques pour permettre des calculs incrémentiels sur des ensembles de données croissants. En outre, nous avons fourni des exemples sur l'utilisation de techniques d'apprentissage automatique dans la vérification de la qualité des données, par exemple pour améliorer les suggestions de contraintes, pour l'estimation de la prédictibilité d'une colonne et la détection des anomalies dans les séries chronologiques historiques de qualité des données.

À l'avenir, nous souhaitons étendre notre suggestion de contraintes basée sur l'apprentissage automatique en exploitant davantage de métadonnées ainsi que des données historiques sur les contraintes définies avec notre API. De plus, nous étudierons les avantages de l'adaptation des distributions bien connues aux colonnes numériques afin de pouvoir comprendre ces données de manière plus détaillée et de suggérer des contraintes plus fines. Une autre direction consiste à fournir aux utilisateurs des messages d'erreur plus complets en cas d'échec des vérifications et leur permettre d'accéder facilement aux enregistrements qui ont fait échouer une contrainte particulière. En outre, nous appliquerons les méthodes saisonnières ARIMA [22] et des prévisions de séries temporelles basées sur des réseaux neuronaux [13] afin d'améliorer notre fonctionnalité de détection des anomalies et de pou-

voir également traiter des séries temporelles saisonnières et intermittentes.

Enfin, nous explorerons en continu la validation de la qualité des données dans des scénarios de streaming, ce qui devrait être une extension naturelle de notre cas d'utilisation incrémental discuté.